VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

# DIPLOMOVÁ PRÁCE

Brno, 2021

Bc. Jan Benda


# 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


# KLASIFIKACE VAD
DEFECTS CLASSIFICATION

**DIPLOMOVÁ PRÁCE**
MASTER'S THESIS


**AUTOR PRÁCE** — **Bc. Jan Benda**
AUTHOR

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


**BRNO 2021**

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. Jan Benda ***ID:*** 195270
***Ročník:*** 2 ***Akademický rok:*** 2020/21

**NÁZEV TÉMATU:**

## Klasifikace vad

**POKYNY PRO VYPRACOVÁNÍ:**

Cílem práce je navrhnout a vytvořit klasifikátor defektů vad nalezených na pásech kontinuální výroby.

1. Zpracujte rešerši používaných metod pro klasifikaci obrazu.
2. Analyzujte vlastnosti dodaných vzorků vad na textilii.
3. Navrhněte vhodné rozhraní pro použití klasifikátoru(DLL) a základní GUI pro práci s klasifikátorem.
4. Navrhněte vhodný klasifikátor a event. rozšiřte příznakový vektor o parametry zajišťující spolehlivou klasifikaci do tříd.
5. Implementujte klasifikátor a natrénujte pro dodaný dataset.
6. Zhodnoťte - určete spolehlivost.

**DOPORUČENÁ LITERATURA:**

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

***Termín zadání:*** 8.2.2021 ***Termín odevzdání:*** 17.5.2021

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

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

## ABSTRAKT

Práce se zabývá návrhem a vytvořením klasifikátorů defektů vad nalezených na pásech kontinuální výroby. Nejprve se práce věnuje rešerši používaných metod pro klasifikaci obrazu a analýze vlastností dodaných vzorů vad na textilii. Poté je popsáno vytvořené rozhraní klasifikátoru a vytvořené grafické rozhraní pro práci s klasifikátorem. Závěrečná část práce je věnována implementaci klasifikátorů a zhodnocení jejich spolehlivosti na dodaných vzorcích vad.



## KLÍČOVÁ SLOVA

Klasifikace vad, klasifikátor, grafické uživatelské rozhraní, OpenCV, segmentace



## ABSTRACT

The thesis deals with a concept and creation of classifiers of defects found on continuous production lines. The first part presents an overview of methods used for image classification and a analysis of defects. The main part of the thesis consist of a description of created classifier interface and graphical user interface for classifier. The last part sums up reliability of each implemented classifer.



## KEYWORDS

Defects classification, classifier, graphical user interface, OpenCV, segmentation

BENDA, Jan. *Klasifikace vad*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2021, 104 s. Diplomová práce. Vedoucí práce: Ing. Peter Honec, Ph.D.

# Prohlášení autora o původnosti díla

| | |
|---|---|
| **Jméno a příjmení autora:** | Bc. Jan Benda |
| **VUT ID autora:** | 195270 |
| **Typ práce:** | Diplomová práce |
| **Akademický rok:** | 2020/2021 |
| **Téma závěrečné práce:** | Klasifikace vad |

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/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ..... .....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

# PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Peterovi Honecovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

# Obsah

# Seznam obrázků

# Seznam tabulek

# Seznam výpisů

# Úvod

Cílem diplomové práce je navrhnout a vytvořit klasifikátor defektů vad nalezených na pásech kontinuální výroby vyrábějících netkanou textilii. Výroba netkané textilie spočívá ve spojení vláken v jednom směru nebo nahodile ve více směrech. Spojení vláken probíhá třením, kohezí nebo adhezí. V dnešní době se netkaná textilie využívá např. na hygienické a čistící prostředky nebo při výrobě ochranných oděvů.

Při výrobě netkané textilie vznikají vady, k jejichž detekci je na lince kontinuální výroby nainstalován kamerový systém. Tento systém dokáže vady v reálném čase spolehlivě detekovat, ale nedokáže je však rozpoznat do tříd. Cílem práce je tedy navrhnout metodu, která by současný proces klasifikace vylepšila, což povede k ulehčení práce obsluhy a k získání kvalitativních statistik, které mohou přispět k nastavení vhodných výrobních parametrů a eliminovat tak počet vad v materiálu. Klasifikační nástroj umožní vady prvotně roztřídit.

Nejprve bude provedena literární rešerše používaných metod pro klasifikaci obrazu. Při práci s obrazem bude využívána segmentace a popis objektu. Součástí literární rešerše budou typy segmentačních metod a možné typy příznaků popisujících objekty v obrazu.

K práci jsou dodány vzorky vad různých tříd. Další úkol bude spočívat v analýze vlastností dodaných vzorků. Dodané vzorky vad lze dělit do čtyř skupin. Uvnitř každé skupiny se nachází několik tříd. Cílem tohoto úkolu bude pro každou třídu vybrat jeden vhodný vzorek, na kterém budou popsány vlastnosti dané třídy.

Hlavním úkolem je vytvoření funkčního rozhraní pro použití klasifikátoru a grafického rozhraní pro práci s klasifikátorem. Bude vytvořena knihovna umožňující přidání a odebrání jednotlivých trénovacích vzorů vad, přidání a odebrání celé třídy vzorů. Knihovna bude dále umožňovat natrénování klasifikátoru a následné třídění vzorků vad s neznámou příslušností k třídě. Grafické rozhraní bude umožňovat použití funkcí knihovny, navíc bude také nabízet možnost uložení natrénovaného klasifikátoru a vytvořených množin do souboru a následné načtení ze souboru zpět do programu.

Závěrem bude na dodaných vzorcích natrénován klasifikátor, jehož spolehlivost bude následně zhodnocena v závislosti na posunu od stávajícího klasifikátoru, který je na lince v současnosti využíván.

# 1 Teoretická část práce

Obor počítačového vidění se zabývá zpracováním obrazu. Při zpracování obrazu dochází v řetězci k několika postupným krokům, viz Obr. 1.1. Pořízený obraz je na začátku řetězce nejprve předzpracován. Při předzpracování dochází například k jasové transformaci, geometrické transformaci nebo k odstranění šumu. Po předzpracování obrazu dochází k segmentaci. Předzpracování a segmentace se řadí mezi méně časově náročné operace. Mezi více časově náročné operace se řadí popis. Poté, pokud to aplikace vyžaduje, dojde ke klasifikaci obrazu.[1]


Obr. 1.1: Schéma zpracování obrazu

## 1.1 Segmentace

Účelem segmentace je vyčlenit z celého obrazu pouze určitou část, jejíž pixely reprezentují snímaný objekt ve scéně. Existuje několik typů segmentací, mezi které patří: [2]

- segmentace na základě prahování,
- segmentace z obrazu hran,
- segmentace založená na regionech,
- srovnávání se vzorem.

### 1.1.1 Segmentace prahováním

Obraz je obvykle reprezentován maticemi jasových hodnot v rozsahu 0-255. Pokud se jedná o obraz šedotónový, je obraz složen pouze z jedné této matice. Naopak obraz barevný je složen z více matic, kde jednotlivé matice reprezentují jednotlivé barevné složky. Nejjednodušším způsobem segmentace prahováním je využití šedotónového obrazu.

Ve snímané scéně se hledaný objekt často odlišuje od svého pozadí jinou jasovou úrovní. Hlavní úkol pak spočívá v určení hraniční jasové hodnoty, která je typická pro daný objekt. Pokud je určena pouze jedna jasová hraniční hodnota, výsledkem segmentace je binární obraz. Jasové hodnoty pixelů, které jsou vyšší než hraniční jasová hodnota, nabývají maximální hodnoty. Naopak hodnoty menší než

[1] V rámci literární rešerše byla využita literatura [1].

hraniční hodnota nabývají hodnoty nulové. Další možností segmentace prahováním je víceprahová segmentace. Ze vstupního obrazu je zvoleno více jasových úrovní. Výsledkem pak není binární obraz, ale obraz šedotónový. Na rozdíl od vstupního obrazu se ve výstupním obrazu nacházejí pouze určité hodnoty jasových úrovní.

Problém volby prahové jasové hodnoty se dá řešit dvěma způsoby. Prvním způsobem je volba hraniční jasové hodnoty pro celý obraz jako celek. Druhým způsobem je rozdělení obrazu do menších částí a následná volba prahové jasové hodnoty pouze pro určitou část obrazu. Po získání prahové jasové hodnoty $T$ je úkolem segmentačního algoritmu projít všechny pixely $f(x, y)$ vstupního obrazu $f$. Pokud daný pixel $f(x, y)$ splňuje podmínku $f(x, y) \geq T$, nabývá pixel ve výstupním obrazu $g$ na souřadnicích $(x, y)$ maximální jasové hodnoty. Pokud podmínka splněná není, nabývá daný pixel ve výstupním obrazu $g$ minimální jasové hodnoty. Výsledkem segmentace je poté binární obraz. Zjištění hodnoty jako jedné konstanty pro celý obraz nemá ve většině případu využití. Jasová hodnota pixelů objektu by se musela lišit od pixelů pozadí, což je však především kvůli šumu ve většině případu nemožné splnit. Při druhém způsobu je na začátku zvoleno více prahů $T_1, T_2, ..T_{\mathrm{N}}$, výstupní obraz $g$ bude šedotónový.

Ke zjištění prahu lze využít histogramu. Histogram obsahuje informaci o kvantitativním zastoupení jednotlivých jasových úrovní v obrazu. Pokud se například jedná o bimodální histogram, tedy histogram, ve kterém jsou patrná dvě navzájem odlišitelná maxima, hledaná prahová hodnota se nachází v minimální hodnotě mezi maximy. Avšak toto řešení by mohlo přinést i nepřesné výsledky, kdyby se ve vstupním obrazu nacházel šum typu sůl a pepř. Histogram by tak mohl být sice bimodální, ale jednotlivá maxima by neodpovídala pouze objektu a pozadí, ale hodnoty by se nacházely v obou prvcích obrazu. Další metodou zisku prahové hodnoty je například algoritmus Otsu.

### 1.1.2 Segmentace z obrazu hran

Pro segmentaci z obrazu hran je třeba nejprve hrany získat. Pouhá aplikace např. Sobelova filtru není dostatečná, jelikož ne každá takto získaná hrana patří objektu, který má být segmentován. Je nutno zjistit, které hrany objektu přísluší. K tomuto účelu slouží několik postupů, mezi které se řadí například prahování obrazu hran. Touto metodou se z obrazu hran odfiltrují hrany, které ve vstupním obrazu signalizují malé změny jasové hodnoty. Mezi malé změny jasové hodnoty se řadí například šum.

Dalším problémem je tloušťka hran. Velké změny jasové hodnoty v obrazu jsou značené hranou, jejíž šířka může být několik pixelů. Ke zúžení hrany na šířku jednoho pixelu se využívá nemaximální potlačení vstupních hran. K dalšímu odfiltrování ne-

potřebných hran se využívá hystereze. Jako hrany jsou uznány pouze ty pixely, které se nacházejí v rozmezí dvou hodnot, značených $[t_0, t_1]$. Používá se dvouprůchodový algoritmus. Bod je automaticky uznán jako součást hrany, pokud jeho jasová hodnota překročí hodnotu $t_1$. V druhém průchodu se pixely porovnávají s hodnotou $t_0$. Má-li bod hodnotu větší než $t_0$ a současně se v jeho osmi-okolí nachází pixel, který byl uznán jako součást hrany při prvním průchodu, je tento pixel také uznán jako součást hrany.

Sledování hran může být použito jako další způsob segmentace. Existuje několik typů sledování hran, mezi které se řadí sledování vnitřní, vnější nebo prodloužené hrany. Trasování vnitřní hranice spočívá v nalezení prvního bodu, který náleží regionu. Dále se prohlíží osmi nebo čtyř okolí nalezeného pixelu. Pokud je v některém z okolních bodů zjištěna přítomnost hrany, daný pixel se vloží do zásobníku. Zároveň je uložen směr, kterým se z původního pixelu získal nový pixel. Trasování končí v případě, že se algoritmus dostane do stavu, kdy znovu porovnává první dva pixely. Trasování vnější hranice se zaměřuje na čtyř-okolí bodů na hranici. Bod je uznán jako součást venkovní hrany, pokud byl v předešlém kroku při zjišťování vnitřní hranice testován. Posledním typem je trasování prodloužené hranice. Při této metodě se okolí testovaného bodu porovnává s vyhledávací tabulkou. Vyhledávání probíhá do té doby, než vznikne uzavřená hranice.

### 1.1.3 Segmentace založená na regionech

Při segmentaci založené na regionech se využívá homogenit uvnitř obrazu. Cílem segmentace je rozdělení obrazu do částí, které jsou uvnitř homogenní. Existuje více možných typů kritérií homogenit, která lze uplatnit. Patří mezi ně například šedotónová nebo barevná jasová hodnota, textura a tvar. Nejjednodušším způsobem využití regionů je nejprve rozdělení obrazu na regiony, které odpovídají jednotlivým pixelům a tyto regiony následně spojovat. Regiony se spojují do té doby, dokud platí podmínka homogenity uvnitř regionu a dokud není splněna podmínka maximálního regionu. Podmínka maximálního regionu znamená, že kdyby se region spojil s některým z okolních regionů, nebyla by nadále splněna podmínka homogenity v rámci nově vzniklého regionu.

Dalším způsobem je dělení regionů. Začíná se s jedním regionem, který reprezentuje celý obraz. Tento region je postupně dělen na menší regiony, dokud nepřestanou platit podmínky homogenity. Výsledky rozdělující a spojující metody se mohou ve většině případů lišit. Výhody obou metod využívá metoda dělení a následného spojování. Pokud je některý z regionů nehomogenní, je rozdělen do čtyř menších regionů, dokud není uvnitř rozdělené části splněna podmínka homogenity. Pokud by poté u některých ze sousedních regionů došlo ke splnění podmínek homogenity i po

spojení těchto regionů, jsou tyto regiony spojeny.

Další metodou je záplavová metoda. V obrazu se nacházejí místa s vyšší jasovou úrovní a místa s nižší jasovou úrovní. Místa s nižší jasovou úrovní se přirovnají k údolí, místa s vyšší jasovou úrovní jsou přirovnána k vyvýšeninám. Jak z názvu metody vyplývá, začne se vyplňovat oblast od nejnižších jasových úrovní. Pokud by došlo k protnutí dvou různých „zaplavených“ míst, vytvoří se na pomezí dvou regionů hranice. Tímto způsobem dochází k segmentaci obrazu. Ke zpřesnění metody je vhodné před první iterací označit místa, ve kterých by měl algoritmus s výplní oblasti začít.

### 1.1.4 Srovnání se vzorem

Další technikou, jejímž účelem je segmentování obrazu, je srovnání se vzorem. Metoda v obrazu hledá předem známý objekt. Úkolem je najít pozici objektu v obrazu. Tím, že je hledaný objekt předem znám, se tato metoda liší od metod předchozích. V předchozích metodách nebyl hledaný objekt dopředu definován. Jednou z možností je hledání přesné kopie. Druhou možností je hledání poněkud deformovaného objektu - natočení nebo odlišná velikost objektu.

## 1.2 Popis

Vstupem popisu obrazu je již segmentovaný obraz. Výstupem popisu je příznakový vektor, který popisuje segmentovaný objekt. Příznakový vektor se skládá z jednotlivých příznaků, které mohou být dvou základních typů - radiometrické nebo geometrické deskriptory. Radiometrické deskriptory využívají jasových hodnot jednotlivých pixelů objektu, geometrické deskriptory využívají tvaru segmentovaného objektu.

#### Radiometrické deskriptory

Mezi radiometrické deskriptory patří:

- průměrná jasová hodnota,
- maximální jasová úroveň,
- minimální jasové úroveň,
- jednotlivé kvantily jasových úrovní.

#### Geometrické deskriptory

Mezi geometrické deskriptory patří:

- obvod,
- obsah,

- natočení,
- konvexnost (poměr obsahu objektu ku obsahu konvexního obalu objektu),
- kompaktnost (kvadrát hodnoty obvodu ku obsahu objektu),
- poměr stran (šířka opsaného obdélníku ku výšce opsaného obdélníku),
- rozsah (obsah objektu ku obsahu opsaného obdélníku),
- homogenita objektu vady.

## 1.3 Klasifikace

Klasifikace slouží k roztřídění dat do obvykle předem definovaného počtu ($R$) tříd. Hlavním prvkem klasifikace je tzv. klasifikátor. Na vstup klasifikátoru přichází $n$–rozměrný vektor příznaků: $x = (x_1, x_2, \ldots x_\mathrm{n})$, který je získán z popisu segmentovaného objektu. Výstupem klasifikátoru je jeden ze symbolů $\omega_1, \omega_2, \ldots \omega_\mathrm{R}$, který je následně interpretován jako jedna z $R$ výstupních tříd klasifikátoru. Čím větší příznakový vektor by byl poslán na vstup klasifikátoru, tím více by rostla jeho spolehlivost. Tímto by však zároveň rostla i výpočetní náročnost. Aby byl rozměr příznakového vektoru co nejmenší, při volbě příznaků je nutné klást důraz na následující vlastnosti:

- spolehlivost – příznak má u objektů ze stejné třídy srovnatelné hodnoty,
- rozlišitelnost – příznak má u objektů z různých tříd rozdílné hodnoty.

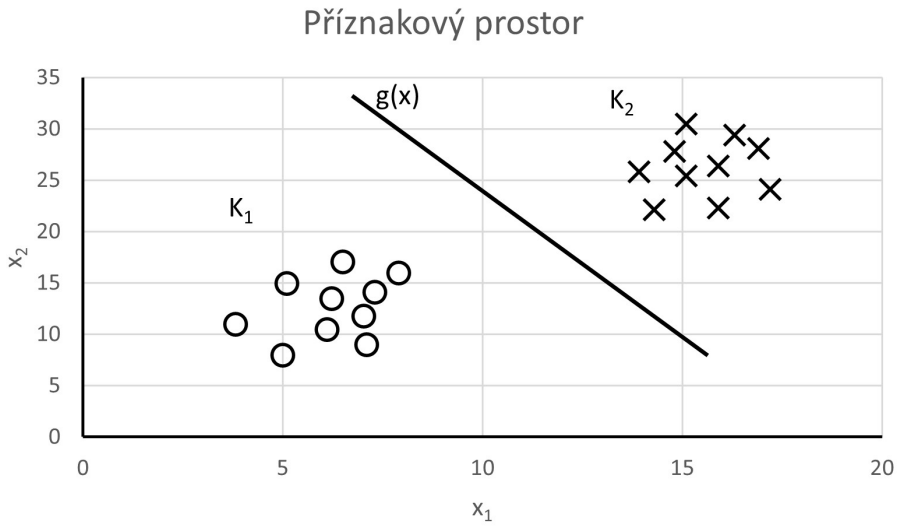
Proces učení klasifikátoru může být dvou typů. Metodu, při které je klasifikátoru předána vstupní množina dat a každý vzor této množiny má známou třídu, se nazývá učení s učitelem. Oproti tomu při učení bez učitele je klasifikátoru předána trénovací množina s vzory, u nichž není dopředu známa příslušnost ke třídě.

### 1.3.1 Klasifikátor s diskriminační funkcí

Příznaky z jednotlivých příznakových vektorů se dají zobrazit v prostoru do tzv. příznakového prostoru. Vztah mezi vstupem a výstupem klasifikátoru je definován jako rozhodovací pravidlo $d(x) = \omega_\mathrm{r}$. Rozhodovací pravidlo tedy dělí příznakový prostor na $R$ podmnožin. Hranice mezi jednotlivými podmnožinami jsou definovány diskriminační funkcí pro dvourozměrný prostor a diskriminační hyper-plochou pro více rozměrný prostor.

V závislosti na rozlišitelnosti tříd se třídy dělí na oddělitelné a neoddělitelné. V ideálním případě by byly třídy oddělitelné a při klasifikaci by nedocházelo k chybným klasifikacím. V reálných případech se však často jedná o neoddělitelné třídy a dochází k chybným klasifikacím, kdy je objekt z určité třídy chybně zařazen do nesprávné třídy. Pokud se jedná pouze o dvourozměrný příznakový vektor, lze jej zobrazit do dvourozměrného příznakového prostoru. Na Obr 1.2 je znázorněn dvourozměrný příznakový prostor. Na obrázku lze vidět dvě ideálně oddělitelné třídy $K_1$

a $K_2$. Jelikož se jedná o dvourozměrný prostor, jsou třídy oddělené diskriminační funkcí. Pokud je možné třídy v příznakovém prostoru rozdělit lineární diskriminační funkcí $g_{\mathrm{r}}(x)$, jedná se o lineární klasifikátor. Pokud nelze v příznakovém prostoru aplikovat lineární diskriminační funkci, řeší se tento problém pomocí transformace příznakového prostoru. Příznakový prostor je transformován transformační funkcí $\Phi$. Na nově vzniklý příznakový prostor je poté aplikována lineární diskriminační funkce: $g_{\mathrm{r}}(x) = q_{\mathrm{r}}(x) * \Phi$. Při vícerozměrném příznakovém vektoru vzniká více dimenzionální prostor, tím pádem se třídy nedají rozdělit pouze jednou diskriminační funkcí. K rozdělení se využívá hyper-ploch. Hyper-plochu lze popsat $R$ diskriminačními funkcemi $(g_1, g_2, \ldots g_{\mathrm{R}})$. Pokud existují dvě třídy $K_{\mathrm{r}}$ a $K_{\mathrm{s}}$, tak diskriminační funkce $g_{\mathrm{r}}(x)$ nabývá pro $x \in K_{\mathrm{r}}$ maximální hodnoty. A tudíž pro $x \in K_{\mathrm{r}}$ platí následující rovnice $g_{\mathrm{r}}(x) \geq g_{\mathrm{s}}(x)$, kde $s \neq r$.




Obr. 1.2: Dvourozměrný příznakový prostor

### 1.3.2 Nejbližší soused

K rozhodování o příslušnosti k vhodným třídám lze využít metodu hledání minimálních vzdáleností. Jedná se o případ využití diskriminační funkce. Jak bylo zmíněno výše, klasifikátor zařazuje prvky do jednotlivých tříd. K tomuto účelu slouží rozhodovací pravidlo. Rozhodovací pravidlo udává vztah mezi vstupem klasifikátoru a výstupem klasifikátoru: $d(x) = \omega_{\mathrm{r}}$. Klasifikátor využívající metodu nejbližšího souseda, zařadí vzor do té třídy, jejíž prvky jsou tomuto vzoru nejblíže v rámci příznakového prostoru. Jelikož je výpočet vzdálenosti pro větší množství dat s vektorovým

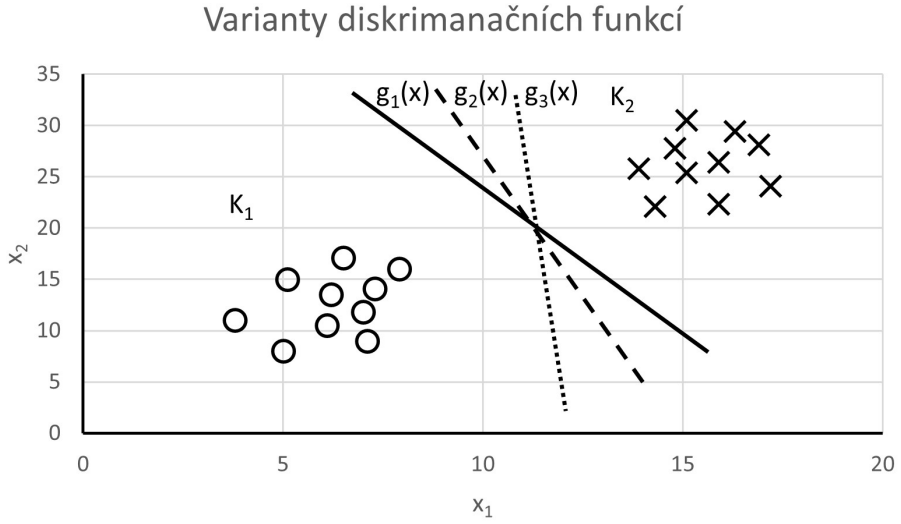
příznakem velkého rozměru časově náročný, používá se několik optimalizací, které zrychlují klasifikaci.

Mezi takové optimalizace patří rozdělení dat. K tomuto účelu se používají K-D stromy [3]. Strom K-D je struktura, která dělí větší skupinu vstupních dat do více menších skupin. Vstupními daty jsou body v prostoru. Výstupní menší množiny jsou složeny z bodů, které mají mezi sebou malou vzdálenost. Díky tomu dojde ke zrychlení algoritmu. Testovány jsou pak pouze tyto menší množiny.

### SVM

Ke zlepšení výsledků při klasifikaci dvou oddělitelných tříd pomáhá rozšíření mrtvé zóny příznakového prostoru mezi těmito dvěma třídami. K tomuto účelu slouží pomocné vektory, které definují diskriminační funkci.

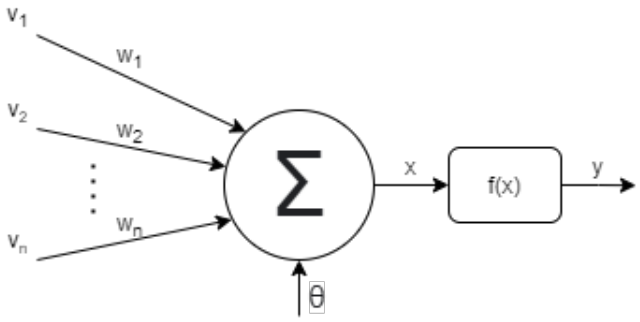
Na Obr. 1.2 byla znázorněna pouze jedna diskriminační funkce. Avšak takovýchto funkcí, které by dané dvě třídy dokázaly oddělit, lze, jak je znázorněno na Obr. 1.3, najít celou řadu. K volbě nejvhodnější diskriminační funkce lze využít pomocné vektory. Lineární diskriminační funkce je v obecném tvaru: $g_{\mathrm{r}}(x) = q_{\mathrm{r}_0} + q_{\mathrm{r}_1} x_1 + \cdots + q_{\mathrm{r}_n} x_{\mathrm{n}}$. Pro $x$ nad $g_{\mathrm{r}}(x)$ platí $g_{\mathrm{r}}(x) > 0$, pro $x$ pod $g_{\mathrm{r}}(x)$ platí $g_{\mathrm{r}}(x) < 0$. Snahou SVM je tedy dosáhnout takového $M$, které bude maximální, ale zároveň pro něj bude platit $\omega_{\mathrm{i}} g_{\mathrm{r}}(x) \geq M$. Dvě třídy, mezi kterými se rozhoduje o diskriminační funkci mají hodnoty $\omega_{\mathrm{i}} \in \{-1, 1\}$. Tedy aby výraz $\omega_{\mathrm{i}} g_{\mathrm{r}}(x)$ nabýval kladných hodnot i pro $x$ pod $g_{\mathrm{r}}(x)$.




Obr. 1.3: Varianty diskriminačních funkcí

### 1.3.3 Neuronové sítě

Základem neuronové sítě je jedna procesní jednotka nazývaná neuron. Většina neuronových sítí pak vzniká kombinací těchto základních prvků. Každá z procesních buněk má větší počet vstupů $v_1, v_2, \ldots v_\mathrm{n}$ a jediný výstup $y$. Funkcí procesní buňky bývá váhování vstupů váhami $w_1, w_2, \ldots w_\mathrm{n}$ a následné provedení sumy. Základní procesní buňka je zobrazena na Obr. 1.4.




Obr. 1.4: Základní procesní buňka neuronových sítí [1]

Celkový vstup neuronu lze vyjádřit jako $x = \sum_{i=1}^{n} v_\mathrm{i} w_\mathrm{i} + \theta$, kde $\theta$ značí práh. Výstup zajišťuje přenosová funkce $f(x)$. Možné tvary přenosové funkce: [4]

Identická:

$$f(x) = x \tag{1.1}$$

Symetrický sigmoid:

$$f(x) = \beta * (1 - e^{-\alpha x})/(1 + e^{-\alpha x}) \tag{1.2}$$

Gaussova funkce:

$$f(x) = \beta(1 - e^{-\alpha x * x}) \tag{1.3}$$

ReLU funkce:

$$f(x) = max(0, x) \tag{1.4}$$

Leaky ReLU funkce:

$$f(x) = \begin{cases} x & \text{pro } x > 0. \\ \alpha x & \text{pro jinak.} \end{cases} \tag{1.5}$$

Pojem neuronové sítě tedy popisuje objekt, ve kterém jsou jednotlivé buňky propojeny. Výstup jedné buňky je vstupem další buňky. Takovýto systém buněk má $n$ vstupů a $m$ výstupů a používá se například při klasifikaci. Vstupními $n$ prvky mohou být například vypočtené příznaky, výstupními $m$ prvky pak mohou být jednotlivé třídy, do kterých je vstupní objekt klasifikován.

#### Dopředné sítě

Jedním z používaných algoritmů dopředných sítí je back–propagation algoritmus. Ten slouží k natrénování neuronové sítě, která má předem definovaný počet vrstev a předpokládá se, že mezi vstupní a výstupní vrstvou existuje minimálně jedna další vnitřní vrstva. Data ze vstupu putují jedním směrem k výstupu. Dopředné neuronové sítě jsou učené z předem roztříděných dat takovým způsobem, že se na výstupu počítá chyba mezi předpokládaným výstupem a výstupem, který síť vyprodukovala. Na základě této chyby algoritmus upravuje váhy jednotlivých buněk za účelem minimalizování chyby.

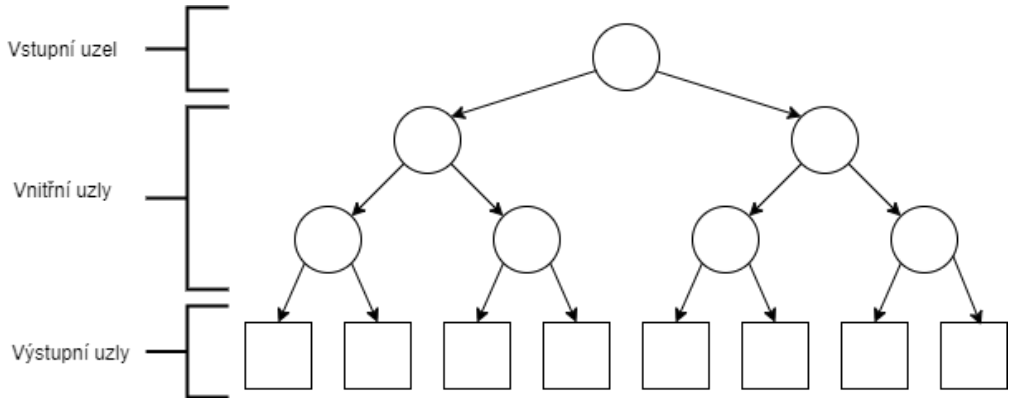
Učící algoritmus tedy nejprve v první iteraci nastaví váhy na náhodné malé hodnoty. V druhé iteraci je na vstup sítě předložen první vzor z trénovací množiny a je vypočítán výstup. Na základě chyby jsou upraveny váhy. Tento proces trvá tak dlouho, dokud pro všechny vstupní vzory nedosahuje hodnota chyby definované hodnoty. Tento typ sítí se řadí mezi klasifikátory s učitelem.

Jednou z modifikací dopředné sítě může být „pružná“ dopředná síť, v anglické literatuře označována jako RPROP (resilent backpropagation). Jedná se o typ učení, který na rozdíl od obyčejného algoritmu dopředné sítě aktualizuje váhu každé neuronové buňky nezávisle na ostatních.

### 1.3.4 Rozhodovací stromy

Tento typ klasifikátoru je vhodný pro klasifikaci do většího množství tříd. Je také vhodné mít k dispozici velkou trénovací množinu. Algoritmus slouží pro klasifikaci a pro regresi dat. Rozhodovací strom se skládá z několika vnitřních uzlů. Uzel má jeden vstup a dva výstupy. Každý uzel tak vstupní hodnotu pošle do dvou různých pod-uzlů. Celý strom rozhodovacích uzlů má tedy jeden vstup a $n$ výstupů. Počet výstupů $n$ odpovídá počtu výsledných tříd klasifikace. Na Obr. 1.5 lze vidět příklad stromové struktury.

Obraz je předán vstupnímu uzlu. V závislosti na vlastnostech vstupního vzoru postupuje obrazový vzor strukturou do pravého nebo do levého podřadného uzlu. Pokud je rozhodovací strom použit pro klasifikaci, ukládá se statistická informace o pravděpodobnosti každé z výstupních tříd. Nejlepších výsledků se dosáhne v tom případě, kdy je vytvořen soubor lehce odlišných rozhodovacích stromů. Každý ze stromů je trénován na náhodně zvolené podmnožině trénovací množiny, čímž vznikne více stromů s lehce odlišnými výsledky. Takováto konstrukce je dobře odolná vůči šumu a vykazuje vyšší přesnost, v anglické literatuře se označuje jako *Random forests*.




Obr. 1.5: Stromová struktura [1]

### 1.3.5 Rozpoznávání vzorů

Při metodě rozpoznávání vzorů se z obrazu místo vypočtení příznaků získávají primitiva, která popisují objekt ve vstupním obrazu. Vztahy mezi jednotlivými primitivy popisují daný objekt. Vhodné je využít co nejmenší počet typů primitiv. Zároveň však taková, aby bylo možné z nich vytvořit dostatečně silný popis objektu. Primitiva by měla být z obrazu snadno získatelná a rozpoznatelná. Primitiva by měla zachytit a popsat významný prvek obrazu. Při fázi učení tak dochází k definovaní primitiv a hledání možných spojení. Tím je sestrojen slovník, dle kterého dochází k reprezentaci objektu primitivy. Pro každou třídu by se měly definovat vztahy mezi primitivy. V rámci rozpoznání pak dochází k hledání spojení nalezených primitiv v obrazu a určení výsledné třídy. Metoda se používá v případech, kdy není možné pomocí příznaků popsat komplexní objekt nebo pokud lze objekt vhodně popsat hierarchickou strukturou sestávající z jednodušších částí.

# 2 Analýza vlastností dodaných vzorů vad

Vzniku vad v netkané textilii nelze nikdy zabránit. Pouze vhodným nastavením výrobní linky, technologických parametrů a čistotou vstupních materiálů lze omezit jejich výskyt. Některé vady jsou „tolerovatelné“. Jejich přítomnost v materiálu nebrání jeho následnému použití. Jde zejména o drobné vady v položení vláken (pokud je taková vada jen v jedné vrstvě), drobné vady v nehomogenitě materiálu nebo i výraznější vady, které jsou však měkké (zvlákněný materiál). Některé vady jsou však kritické a není možné produkt expedovat. Jde zejména o díry, tvrdé úkapy a kontaminaci (olej, hmyz), která je v hygienickém průmyslu naprosto nepřípustná.

Některé vady lze do kategorií vzhledem k jejich charakteru roztřídit jednoduše, některé třídy jsou však vzhledově podobné a klasifikace takových vad je poté obtížná. Dodané vzory vad lze rozdělit do čtyř skupin. Obrazy vad jsou pořizovány dvěma kamerami. Jedna kamera snímá při podsvícení, druhá kamera při nadsvícení. Každá z kamer generuje dva typy snímků: světlé vady a tmavé vady. Tímto způsobem vznikají čtyři typy snímků, které mají následné označení.

- ID 9 - kamera 1, podsvit, tmavé
- ID 10 - kamera 1, podsvit, světlé
- ID 11 - kamera 2, nadsvit, tmavé
- ID 12 - kamera 2, nadsvit, světlé

Jednotlivé skupiny obsahují třídy, které přímo odpovídají typům vad. V následující sekci jsem pro každou skupinu nejprve vyjmenoval jednotlivé typy vad (třídy). Poté jsem se snažil v každé skupině pro každý typ vady vybrat jeden snímek, na kterém je vzor s typickými parametry dané třídy. Avšak u některých tříd platí, že se v rámci jedné třídy vyskytuje více vzorů, které mají odlišné vlastnosti. Tento problém se vyskytuje například u vady typu: díra - podsvit, viz Obr. 2.4.

## 2.1 Tmavé vady - podsvit

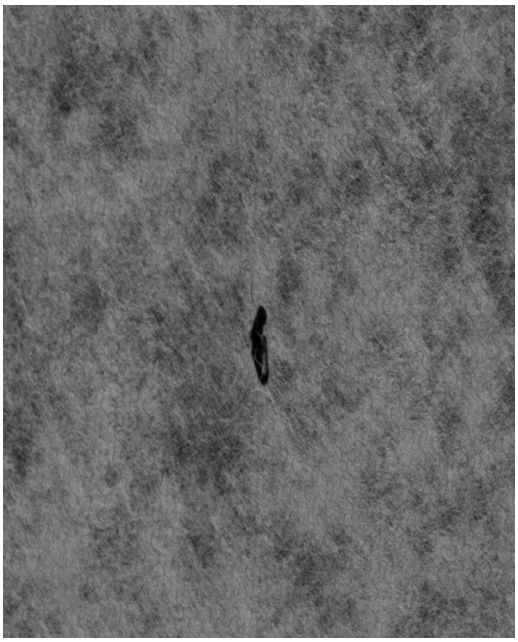
Prvním typem vad jsou tmavé vady vzniklé při podsvícení. V této skupině se nachází celkem sedm tříd, mezi které patří:

- nečistota,
- průlet,
- úkap,
- úkap s dírou,
- úlet,
- zeslabené místo,
- díra.

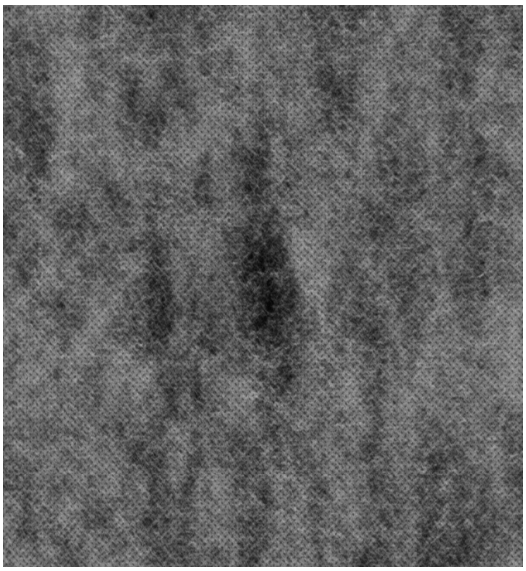
V této skupině se nacházejí dvě třídy - průlet a úlet, které se od ostatních vad liší svým protáhlým tvarem ve vertikálním směru. Dalšími odlišujícími parametry jsou rozsah, konvexnost a radiometrické parametry: minimální jasová úroveň uvnitř vady, maximální jasová úroveň uvnitř vady a kvantily jasových úrovní (25., 50. a 75.) uvnitř objektu vady. Dále také 50. kvantil jasových úrovní v okolí objektu vady.

### Nečistota

Na Obr. 2.1a je znázorněna vada typu nečistota - podsvit. Tento typ vady dosahuje vysokých hodnot v poměru stran. Objekt vady dosahuje vyšších hodnot také v parametrech konvexnosti a rozsahu. Na Obr. 2.1a lze vidět, že se vada bude řadit mezi tmavší vady v rámci skupiny. Radiometrické příznaky by tak měly nabývat nižších hodnot.



(a) ID 9 - nečistota



(b) ID 9 - zeslabené místo

Obr. 2.1: ID 9 - nečistota, zeslabené místo

### Zeslabené místo

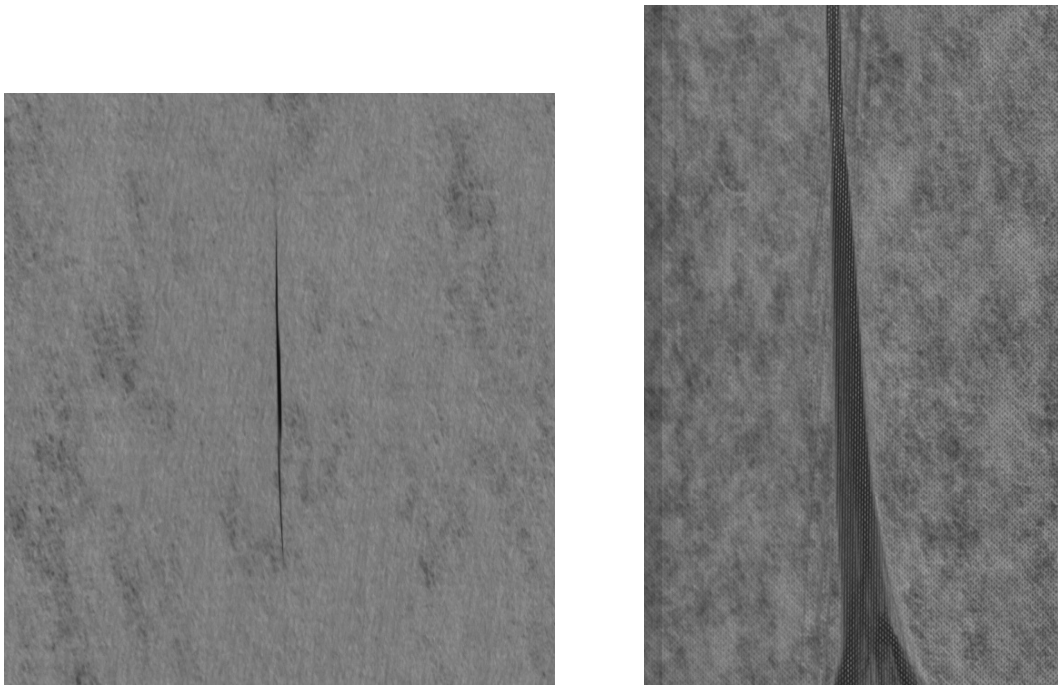
Na Obr. 2.1b je znázorněna vada typu zeslabené místo - podsvit. Jak lze vidět na obrázku, jedná se o jednu ze světlejších vad v této skupině. Dále bude mít tato vada malou hodnotu rozsahu a konvexnosti, neboť lze vidět, že obal objektu vady je ve srovnání s vadou typu nečistota více členitý. Hodnota poměru stran bude ve srovnání například s vadou typu průlet nebo úlet nabývat vyšších hodnot.

### Průlet

Na Obr. 2.2a je znázorněna vada typu průlet - podsvit. U vady typu průlet dosahuje hodnota poměru šířky ku výšce objektu velmi malých hodnot. V tomto parametru má vada srovnatelné hodnoty s vadou typu úlet a některých vad typu úkap.

Objekt vady dosahuje malých hodnot šířky objektu. Vada typu průlet se nenachází na povrchu materiálu, jedná se pouze o průlet pod objektivem kamery. Z tohoto důvodu by se uvnitř této vady neměla vyskytovat žádná textura, jako tomu je na Obr. 2.2b. Tím pádem by se vady měly lišit v parametrech: rozsah a konvexnost.

Rozdílem mezi vadou typu průlet a vadou typu úkap je pravděpodobně průměrná jasová hodnota, kde vada typu průlet nabývá nižších hodnot.



(a) ID 9 - průlet

(b) ID 9 - úlet

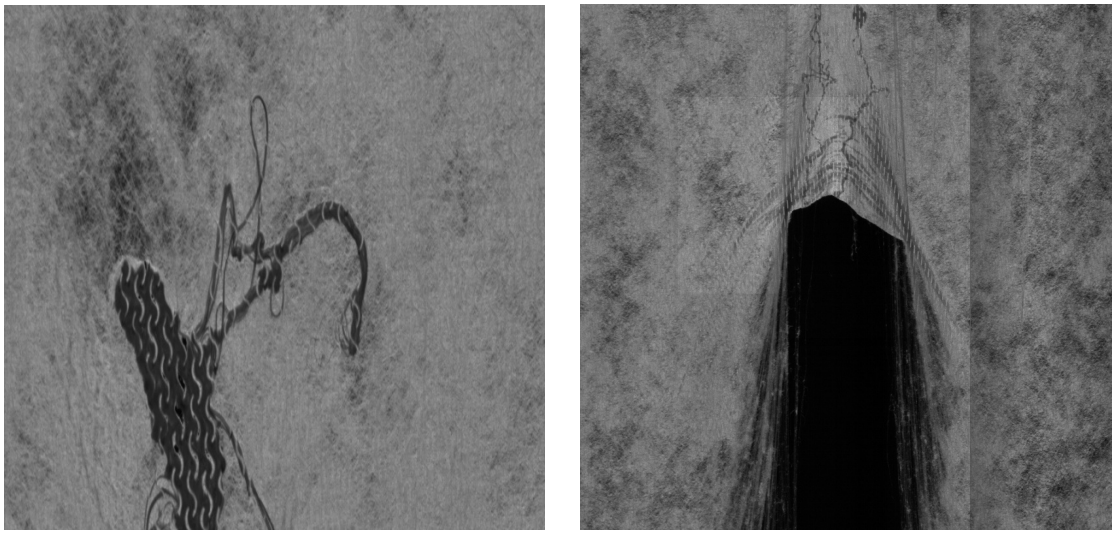
Obr. 2.2: ID 9 - průlet, úlet

### Úlet

Na Obr. 2.2b je zobrazena vada typu úlet - podsvit. Tato vada dosahuje podobných hodnot v parametru poměru stran jako vada průlet. Již bylo zmíněno, jakým způsobem se dané vady dají odlišit. Úlet dosahuje menší hodnoty v konvexnosti a rozsahu než průlet, neboť se uvnitř vady typu úlet často vyskytuje textura látky. Vady úlet dosahují také větších maximálních šířek objektu.

### Úkap

Na Obr. 2.3a je znázorněna vada typu úkap - podsvit. Hlavním parametrem, který odlišuje vadu od ostatních, je rozsah. Pokud bude této vadě opsán obdélník, bude poměr obsahu vady ku obsahu obdélníku malé číslo. Dalším parametrem, který bude nabývat menších hodnot než u ostatních vad, by měla být konvexnost. Jelikož se většinou jedná o členitý objekt, bude hodnota konvexnosti dosahovat menších hodnot než u ostatních vad.



(a) ID 9 - úkap

(b) ID 9 - úkap s dírou

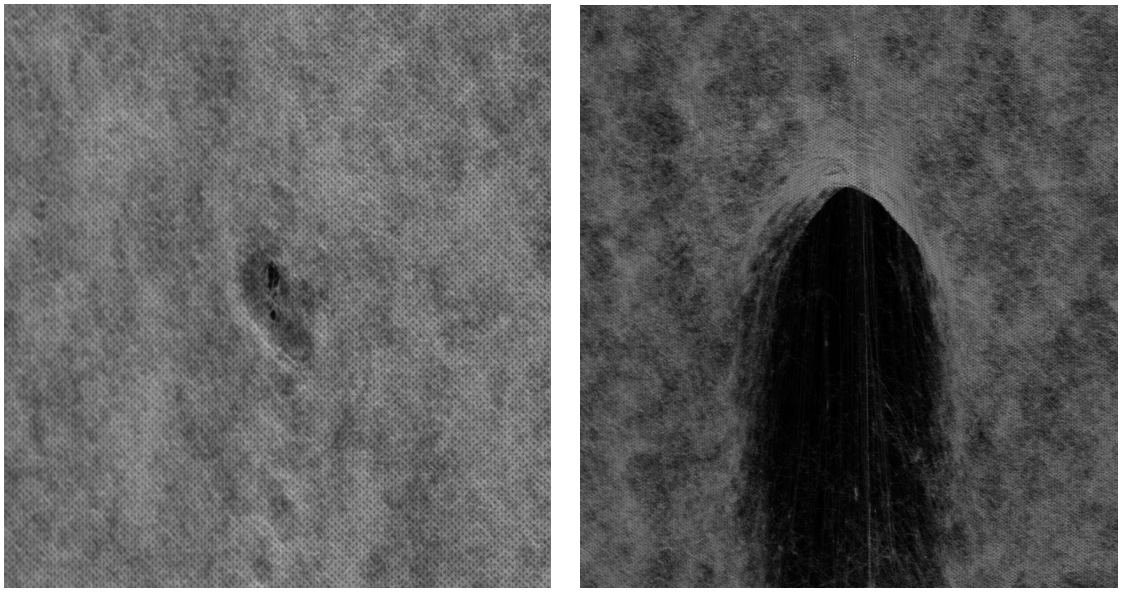
Obr. 2.3: ID 9 - úkap, úkap s dírou

### Úkap s dírou

Na Obr. 2.3b je znázorněna vada typu úkap s dírou - podsvit. Objekt vady nabývá zpravidla velkých hodnot obsahu. V tomto parametru se však shoduje s některými vadami typu díra. Jak je vidět z rozdílu mezi Obr. 2.3b a Obr. 2.4, úkap s dírou obsahuje ukápnutý materiál, tím pádem bude opsaný obdélník větší než u vady díra. Měla by tedy u vady typu úkap s dírou být menší hodnota rozsahu a konvexnosti, než tomu bude u vady typu díra.

### Díra

Na Obr. 2.4 jsou zobrazeny dvě vady typu díra - podsvit. Hned při srovnání dvou vad ze stejné třídy lze vidět, že definice parametrů není jednotná. Mezi oběma vadami je vidět rozdíl v obsahu, šířce a výšce. Vada na Obr. 2.4a dosahuje malých hodnot obsahu. S vadami typu zeslabené místo je téměř totožná. Není vhodné tyto vady sloučit, neboť vada na Obr. 2.4b se od vad typu zeslabené místo liší. Naopak větší vada typu díra se shoduje s vadou typu úkap s dírou, avšak z podobného důvodu, jako v předešlém případě, nelze tyto vady sloučit.



(a) ID 9 - díra (malá)

(b) ID 9 - díra (velká)

Obr. 2.4: ID 9 - díra, dva typy defektu

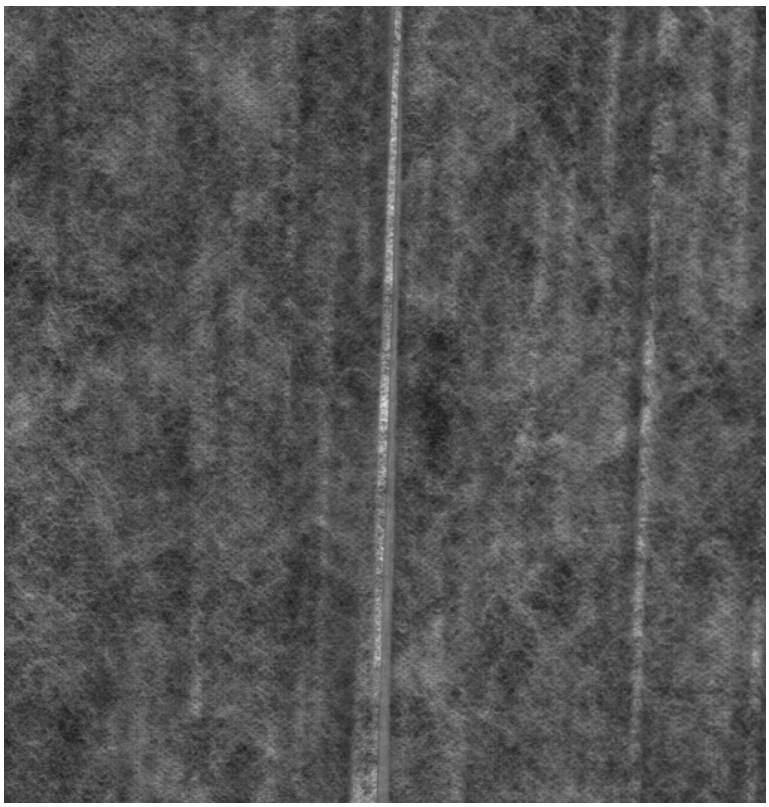
## 2.2 Světlé vady - podsvit

Druhou skupinou vad jsou světlé vady vzniklé při podsvícení. V této skupině se nachází celkem šest typů vad. U této skupiny bude již na první pohled obtížné některé vady rozlišit. Například při pohledu na Obr. 2.6a a Obr. 2.6b lze vidět, že tyto vady vypadají podobně. Mezi parametry, které budu používat pro rozlišení vad patří rozsah, maximální šířka a maximální výška objektu vady. Z radiometrických příznaků využiji maximální jasovou hodnotu a průměrnou hodnotu pixelů v okolí objektu vady. Při pohledu na obrázky vad této třídy se nabízí možnost využití některé z morfologických operací. Buď pouhá dilatace nebo metoda uzavření. Mezi vady této skupiny patří:

- úkap,
- utržené vlákno,
- vlaštovka,
- zesílené místo,
- sklad,
- aviváž.

### Sklad

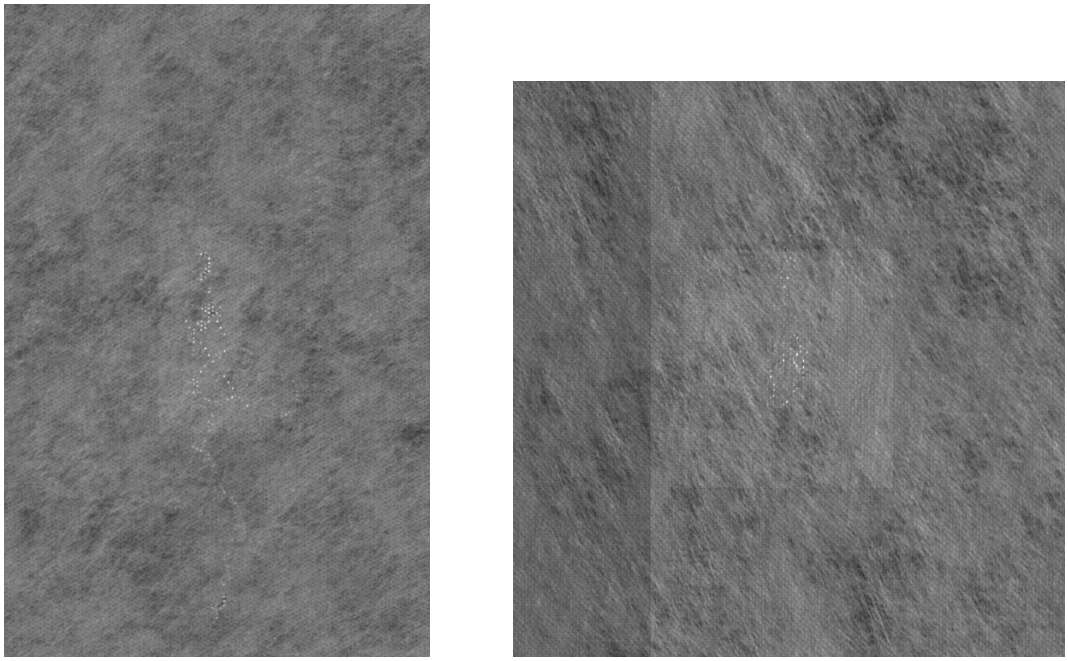
Na Obr. 2.5 je zobrazena vada typu sklad - podsvit. Jelikož se vada vyskytuje téměř na celém snímku, pro analýzu vlastností bude záležet na tom, jakým způsobem bude vada segmentována. Pokud budou segmentovány všechny sklady v obrazu, opsaný obdélník bude pokrývat velkou část obrazu a parametr šířky bude dosahovat nejvyšších hodnot v rámci skupiny. Pokud dojde k segmentaci pouze jednoho z více skladů, bude mít vada parametry srovnatelné s vadami typu úlet a utržené vlákno. Objekt vady bude mít malé hodnoty rozsahu a na tuto skupinu nízkou maximální jasovou úroveň. Spolehlivost klasifikace bude pravděpodobně ovlivněna nízkým zastoupením vzorů této vady v dodaném datasetu.



Obr. 2.5: ID 10 - sklad

### Úkap

Na Obr. 2.6a je znázorněna vada typu úkap - podsvit. U vad tohoto typu dosahuje hodnota maximální šířky nižších hodnot. Naopak hodnota maximální výšky dosahuje větších hodnot. Při srovnání s vadou typu utržené vlákno, která je zobrazena na Obr. 2.6b, nejsou vidět žádné rozdíly. Proto by mohlo dojít ke sloučení těchto dvou tříd do jedné. Po použití morfologických operací bude tento typ vad nabývat malých hodnot rozsahu. Ve srovnání s vadou typu sklad mají pixely v okolí objektu vady typu úkap vyšší jasové hodnoty.



(a) ID 10 - úkap (b) ID 10 - utržené vlákno

Obr. 2.6: ID 10 - úkap, utržené vlákno

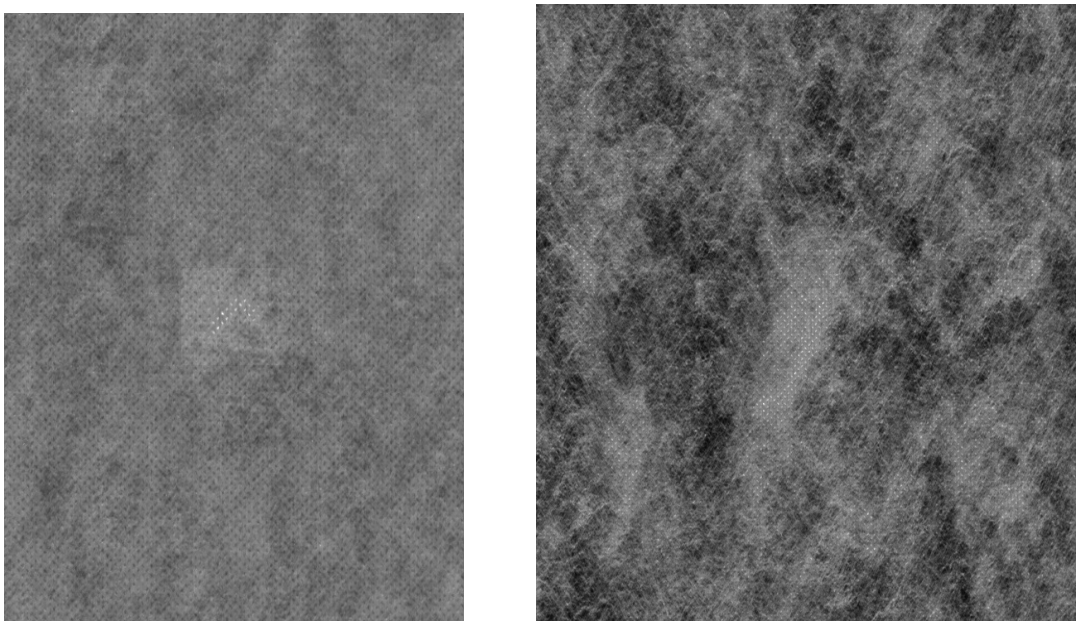
### Utržené vlákno

Na Obr. 2.6b je zobrazena vada typu utržené vlákno - podsvit. Jak bylo zmíněno v předchozí podkapitole, vady typu úkap a utržené vlákno jsou záměnné. Při klasifikaci by se mohlo stávat, že tyto vady budou navzájem vadně klasifikovány, neboť i při použití morfologických operaci bude obtížné tyto vady rozlišit. Vlastnosti této vady se s shodují s vlastnostmi popsanými v předešlé podkapitole.

### Vlaštovka

Na Obr. 2.7a je zobrazena vada typu vlaštovka - podsvit. Ve srovnání s předchozími dvěma vadami je zřejmé, že maximální výška této vady bude mít nejnižší hodnotu, což by tuto vadu mělo odlišit od ostatních vad této skupiny s výjimkou vady typu zesílené místo a aviváž. Hodnota rozsahu by u této vady měla dosahovat menších hodnot. Okolní pixely mají srovnatelnou jasovou úroveň jako u vady typu úkap.

Hodnoty šířky by u této vady měly dosahovat velkých hodnot při srovnání s vadami typu úlet, úkap a utržené vlákno.



(a) ID 10 - vlaštovka

(b) ID 10 - zesílené místo

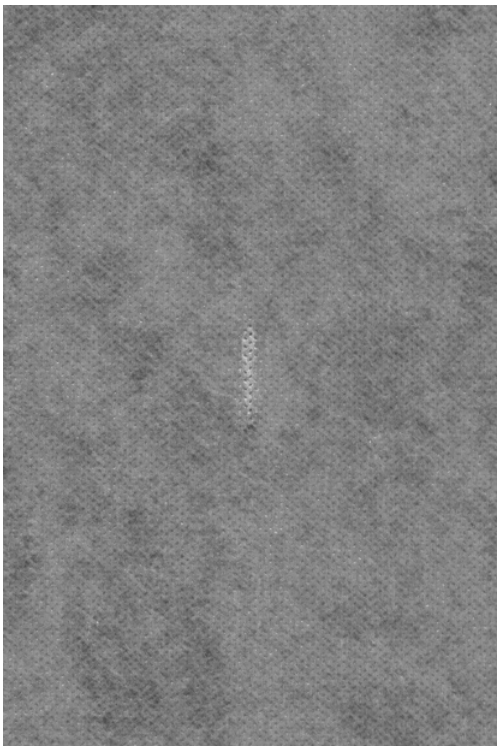
Obr. 2.7: ID 10 - vlaštovka, zesílené místo

### Zesílené místo

Na Obr. 2.7b je zobrazena vada typu zesílené místo - podsvit. U této vady by měla mít hodnota parametru rozsahu na tuto skupinu vad vysokou hodnotu. Při provedení jedné z výše zmíněných morfologických operací by měla hodnota rozsahu dosahovat vyšších hodnot než u ostatních typů vad. Jelikož byly v dodaném datasetu pouze 3 vzorky s vadou typu zesílené místo - podsvit, klasifikace této vady by nemusela dosahovat vysokých hodnot spolehlivosti.

#### Aviváž

Na Obr. 2.8 je zobrazena vada typu aviváž - podsvit. Tyto vady budou mít v rámci skupiny vysoké hodnoty rozsahu. Pixely objektu vady dosahují nízkých jasových úrovních. V porovnání s ostatními vadami však dosahují vyšších hodnot jasové hodnoty pixelů mimo objekt vady.



Obr. 2.8: ID 10 - aviváž

## 2.3 Tmavé vady - nadsvit

Třetím typem vad jsou tmavé vady vzniklé při nadsvícení. V této třídě se nacházejí celkem čtyři typy vad. Mezi typy vad z této skupiny patří:

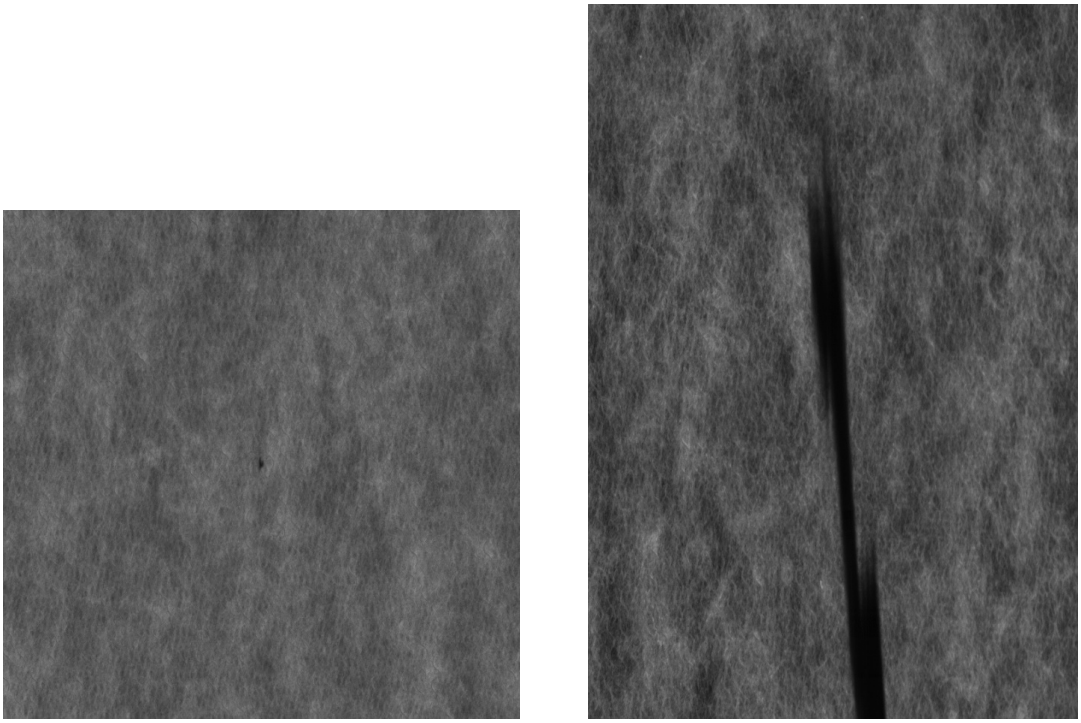
- nečistota,
- průlet,
- zeslabené místo,
- muška.

Vada typu průlet se od ostatních odlišuje poměrem stran. Zbylé tři třídy by se měly lišit v parametrech konvexnosti, rozsahu a obsahu. Zároveň využiji k rozlišení vad i radiometrické příznaky.

### Muška

Na Obr. 2.9a je zobrazena vada typu muška - nadsvit. Objekt této vady má malý obsah a velký poměr stran, který by se měl blížit 1. V porovnání s vadami typu zeslabené místo a nečistota má velké hodnoty v parametrech konvexnosti a rozsahu.

V porovnání s vadou typu zeslabené místo by se měl tento typ vady lišit v jasových hodnotách pixelů okolí, kde by měl dosahovat vyšších hodnot.



(a) ID 11 - muška (b) ID 11 - průlet

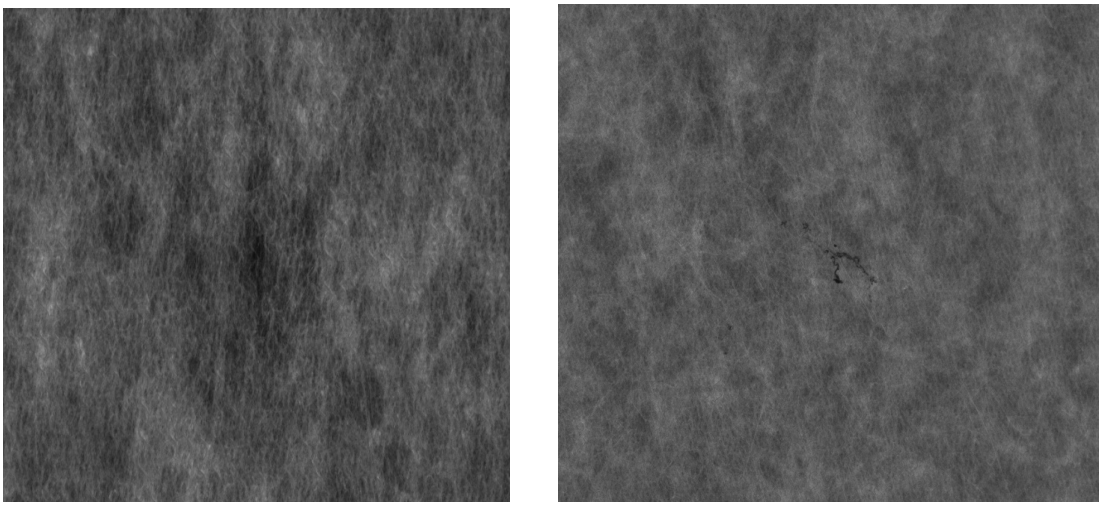
Obr. 2.9: ID 11 - muška, průlet

### Průlet

Na Obr. 2.9b je zobrazena vada typu průlet - nadsvit. Jedná se vždy o podlouhlý objekt ve vertikálním směru. Vada se vyznačuje malou hodnotou poměru stran. Parametry konvexnosti i rozsahu budou dosahovat menších hodnot než tomu bylo u vady typu muška. Hodnota průměrného jasu se u vzorků liší. Hodnota konvexnosti bude podobně jako u nečistoty dosahovat malých hodnot, avšak tyto dvě vady se dají rozlišit pomocí parametru poměru stran. Typickou vlastností vzorků této vady je také velikost, především výška opsaného obdélníku.

### Zeslabené místo

Na Obr. 2.10a je zobrazena vada typu zeslabené místo - nadsvit. Vada bude mít vyšší hodnoty v parametru poměru stran. Pokud proběhne segmentace tím způsobem, že celková vada bude složena z několika malých objektů, budou hodnoty konvexnosti a rozsahu dosahovat menších hodnot. Dalším společným parametrem je na vady z této třídy poměrně vysoká hodnota průměrné jasové úrovně pixelů uvnitř vady a naopak nízká jasová úroveň pixelů vně objekt vady.



(a) ID 11 - zeslabené místo

(b) ID 11 - nečistota

Obr. 2.10: ID 11 - zeslabené místo, nečistota

### Nečistota

Na Obr. 2.10b je zobrazena vada typu nečistota - nadsvit. U většiny vzorků této vady dosahuje hodnota poměru stran podobných hodnot jako u vady typu zeslabené místo. Dalšími parametry, které jsou u vzorků této vady společné, jsou konvexnost a rozsah. U některých vzorků vad bude také vyšší průměrná jasová hodnota než u vady typu muška, se kterou se v některých parametrech podobají.

## 2.4 Světlé vady - nadsvit

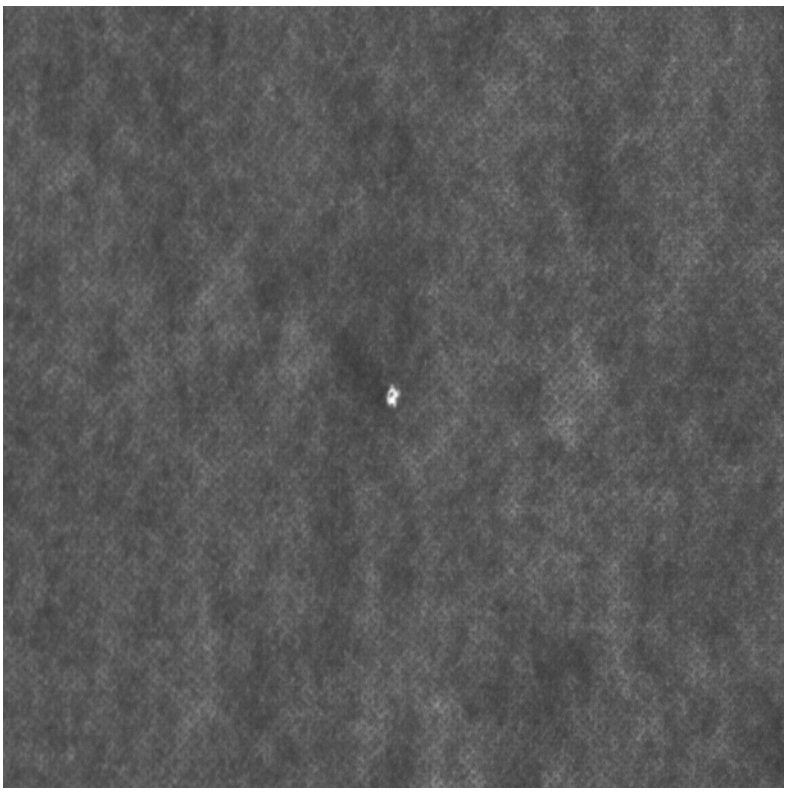
Poslední skupinou vad jsou světlé vady vzniklé při nadsvícení. V této třídě se nachází celkem devět typů vad. Vady této třídy se dají rozdělit podle maximální výšky a maximální šířky. Dalšími geometrickými parametry, které jsem pro rozlišení použil, jsou konvexnost a úhel natočení objektu vady. Z geometrických parametrů jsem

využil maximální jasovou úroveň uvnitř vady, průměrnou jasovou úroveň v okolí vady a homogenitu objektu vady. Mezi vady této skupiny patří:

- úkap,
- úkap s dírou,
- úlet,
- utržené vlákno,
- vlaštovka,
- zesílené místo,
- sklad,
- prášení,
- dlouhý tenký úlet.

### Prášení

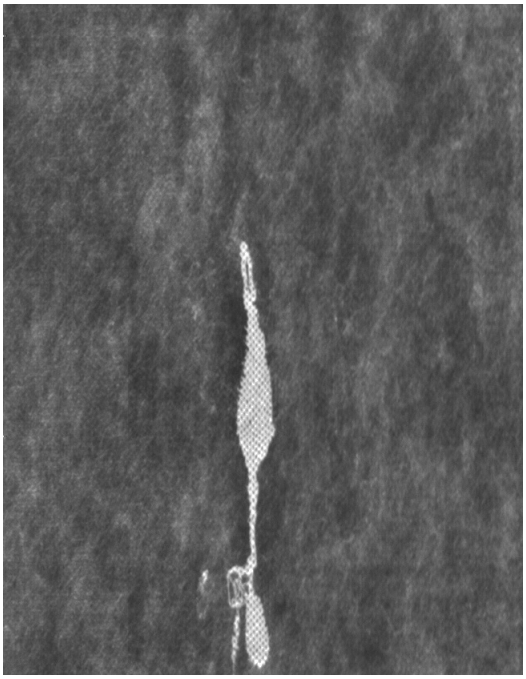
Na Obr. 2.11 je zobrazena vada typu prášení - nadsvit. Vada se vyznačuje malými rozměry, vysokou jasovou hodnotou uvnitř objektu vady a nízkou jasovou hodnotou v okolí vady. Objekt vady dosahuje také vysokých hodnot konvexnosti. Objekt vady je také velmi homogenní ve srovnání s ostatními vadami v této skupině.



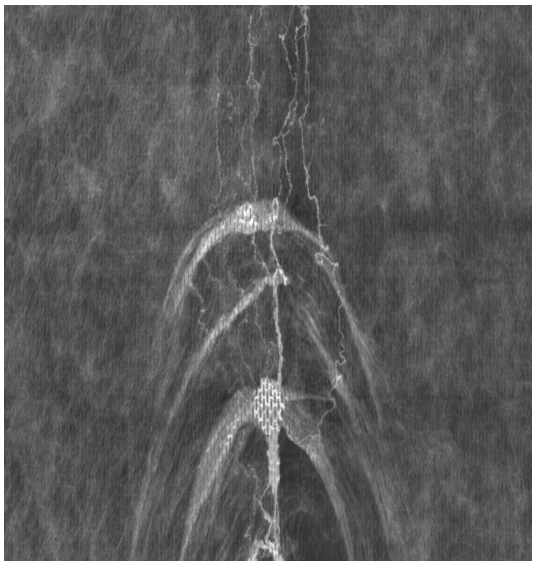
Obr. 2.11: ID 12 - prášení

### Úkap

Na Obr. 2.12a je zobrazena vada typu úkap - nadsvit. Vada se vyznačuje malou hodnotou poměru stran, protáhlým tvarem a nízkou hodnotou konvexnosti, díky které ji lze odlišit od některých ostatních vad. Vada dosahuje vysokých hodnot průměrné jasové úrovně. Vada má nízkou hodnotu šířky a velkou hodnotu výšky, z čehož vyplývá malá hodnota poměru stran.



(a) ID 12 - úkap



(b) ID 12 - úkap s dírou

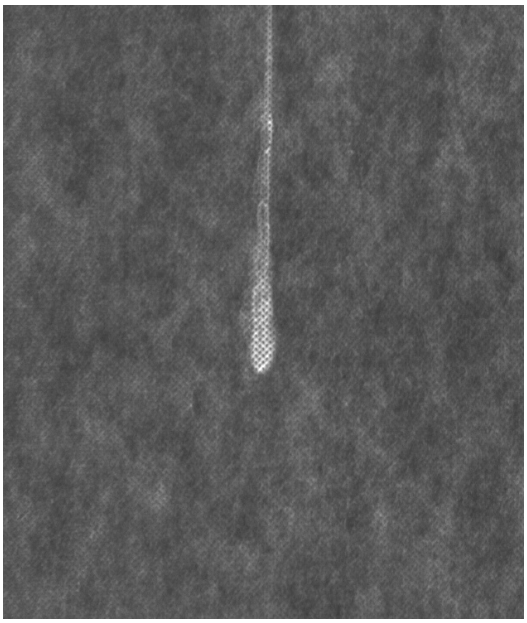
Obr. 2.12: ID 12 - úkap, úkap s dírou

### Úkap s dírou

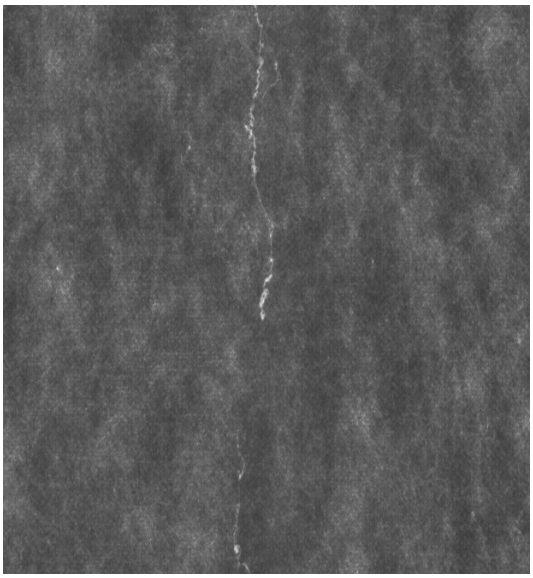
Na Obr. 2.12b je zobrazena vada typu úkap s dírou - nadsvit. Vada se vyznačuje velikou šířkou a výškou a velmi nízkou hodnotou konvexnosti. Vada bude mít v porovnání s ostatními vadami nízkou průměrnou jasovou hodnotu.

### Úlet

Na Obr. 2.13a je zobrazena vada typu úlet - nadsvit. Vada typu úlet se řadí do skupiny vad s nízkou hodnotou poměru stran, má tedy malou šířku a větší hodnotu výšky. Ve srovnání s vadou typu dlouhý tenký úlet se však ve zmíněných dvou parametrech liší. Vada dosahuje nižších hodnot průměrné jasové úrovně než úkap.



(a) ID 12 - úlet



(b) ID 12 - utržené vlákno

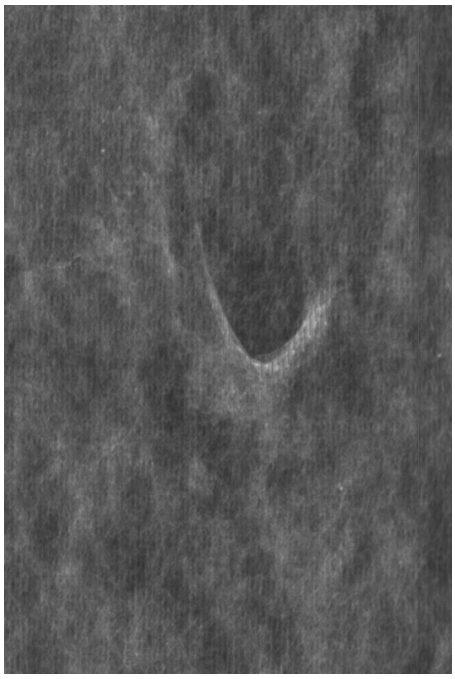
Obr. 2.13: ID 12 - úlet, utržené vlákno

### Utržené vlákno

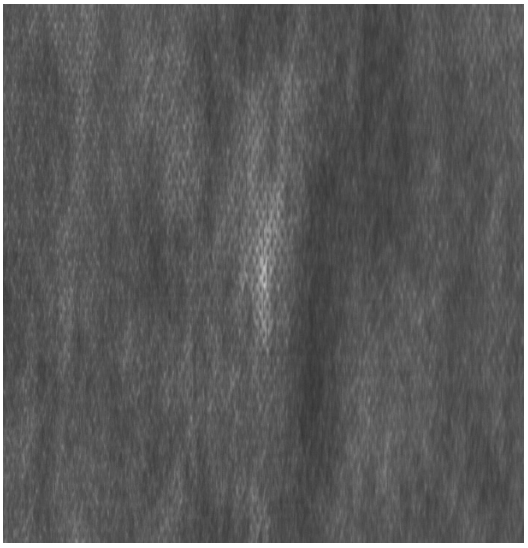
Na Obr. 2.13b je zobrazena vada typu utržené vlákno - nadsvit. Vada by měla mít současně malý poměr stran, malý obsah a malou hodnotu konvexnosti. Tím se liší od ostatních vad s malým poměrem stran. Tento typ vady dosahuje na tuto skupinu vad nízké průměrné hodnoty pixelů uvnitř objektu vady.

### Vlaštovka

Na Obr. 2.14a je zobrazena vada typu vlaštovka - nadsvit. Tato vada se odlišuje od ostatních tím, že dosahuje zároveň nízkých hodnot výšky a vysokých hodnot šířky. Vada typu vlaštovka bude mít nízké hodnoty konvexnosti a nízkou hodnotu průměrné jasové úrovně uvnitř i mimo objekt vady.



(a) ID 12 - vlaštovka



(b) ID 12 - zesílené místo

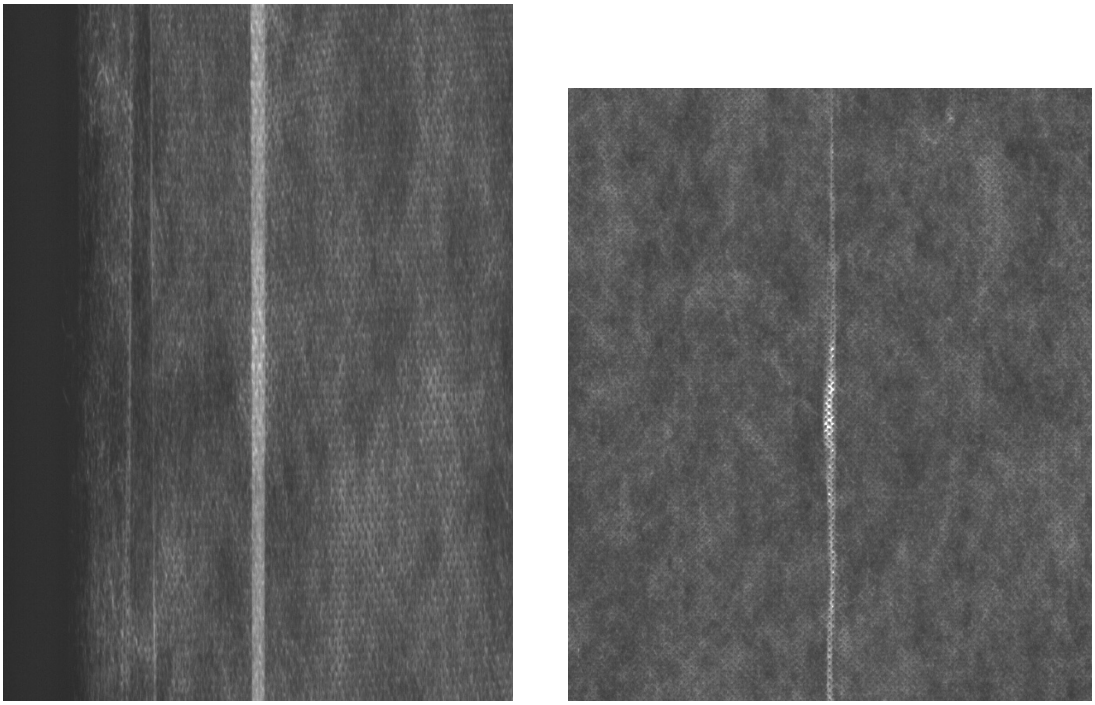
Obr. 2.14: ID 12 - vlaštovka, zesílené místo

### Zesílené místo

Na Obr. 2.14b je zobrazena vada typu zesílení místo - nadsvit. Tato vada se řadí do skupiny s vysokou hodnotou poměru stran. Do stejné skupiny se řadí vady prášení, vlaštovka a úkap s dírou. Od vady typu prášení a úkap s dírou se liší svojí velikostí. Od vady typu vlaštovka se liší v parametru konvexnosti, neboť tento parametr bude u vady zesíleného místa nabývat vyšších hodnot než u vady typu vlaštovka. Průměrná hodnota jasových úrovní mimo objekt vady bude obdobně jako u vady typu vlaštovka dosahovat poměrně nízkých hodnot.

### Sklad

Na Obr. 2.15a je zobrazena vada typu sklad - nadsvit. Tato vada se řadí do skupiny, u které dosahuje hodnota poměru stran nízkých hodnot. V rámci této skupiny se u vady typu sklad objevuje nejmenší členitost objektu vady. Hodnota konvexnosti tak bude dosahovat velkých hodnot ve srovnání s ostatními vadami. Vada má nízkou průměrnou jasovou úroveň.



(a) ID 12 - sklad

(b) ID 12 - dlouhý tenký úlet

Obr. 2.15: ID 12 - sklad, dlouhý tenký úlet

### Dlouhý tenký úlet

Na Obr. 2.15b je zobrazena vada typu dlouhý tenký úlet - nadsvit. Jak bylo zmíněno výše, vada se řadí do skupiny vad s malým poměrem stran. Od ostatních vad této třídy se liší výškou a šířkou. Vada dosahuje v rámci třídy nižších průměrných jasových úrovní.

# 3 Grafické rozhraní a rozhraní klasifikátoru

V této kapitole budu popisovat vytvoření programového vybavení skládajícího se z knihovny tvořící rozhraní klasifikátoru a grafického rozhraní využívajícího tuto knihovnu. Cílem je vytvoření grafického rozhraní, které uživateli umožní nahrání vzorů s již definovanou třídou do trénovací množiny, na základě dat vytvořené trénovací množiny umožní natrénování sady klasifikátorů a pomocí natrénovaných klasifikátorů zprostředkuje klasifikaci nahrávaných testovacích vzorů s předem neznámou příslušností k třídě. Trénovací a testovací vzory jsou v rámci aplikace ukládány do seznamů. Výběrem vzoru ze seznamu testovacích vzorů dochází k zobrazení predikovaných tříd jednotlivých klasifikátorů pro vybraný vzor. Tímto vzniká několik požadavků jak na grafické rozhraní, tak na rozhraní klasifikátoru.

Základním požadavkem na knihovnu je definice proměnných sloužících pro uložení dat. Do paměti programu se budou ukládat data trénovací množiny, data testovací množiny a parametry klasifikátorů. Dále je potřeba vytvořit funkce, které umožní vzory do proměnných množin nahrát a následně z množin odstranit. Při nahrávání je zároveň třeba spočítat každému vzoru hodnoty příznakového vektoru, který bude použit pro následné trénování/testování klasifikátorů. Aby bylo možné z obrazu vzoru hodnoty příznaků získat, je třeba z obrazu segmentovat objekt vady. Poté je třeba vytvořit funkce, které umožní natrénování klasifikátorů a klasifikaci neroztříděných vzorů.

Úkolem grafického rozhraní je umožnit uživateli využít možnosti knihovny, tedy vkládání dat do množin a používání klasifikátorů. Grafické rozhraní také umožňuje uložení dat (trénovací množiny a natrénovaných klasifikátorů) do souboru. Při zavření a otevření aplikace nebude nutné opět vytvářet trénovací množinu a trénovat klasifikátory. Současně s vkládáním vzorů vytvářím seznamy prvků v trénovací a testovací množině.

Grafické rozhraní jsem vyřešil vytvořením desktopové aplikace. Knihovnu i desktopovou aplikaci jsem vytvářel pomocí programu Microsoft Visual Studio. Nejprve jsem však zvažoval možnost vypracování zadání pomocí programu Matlab. Důvodem využití programu Matlab by byla předešlá zkušenost použití tohoto programu při zpracování obrazu a nulová zkušenost zpracování obrazu s využitím programovacího jazyka C++ (Microsoft Visual Studio). Naopak jsem měl předešlou zkušenost s vytvářením grafického rozhraní v programu Microsoft Visual Studio pomocí jazyka C++, tím pádem jsem nakonec zvolil použití tohoto programu i pro vytváření své diplomové práce. Pro práci s obrazem jsem využil volně dostupnou knihovnu OpenCV. Návod na přidání knihovny OpenCV se nachází na [5].

## 3.1 Knihovna DLL

Pro lepší přehlednost kódu jsem funkce a definice proměnných rozložil do několika jmenných prostorů. Funkce jsou děleny v závislosti na použití. Jmenný prostor `Dataset` obsahuje funkce a proměnné pro práci s množinami. Konkrétně tedy obsahuje definice proměnných pro uložení testovacích a trénovacích vzorů do příslušných množin a funkce pro přidání a odebrání vzorů z těchto množin.

Uvnitř jmenného prostoru `Description` se vyskytují funkce sloužící pro práci s obrazem. Jedná se o funkci sloužící pro segmentaci objektu vady a funkci pro následné vypočítání hodnot do příznakového vektoru.

Dále bylo potřeba definovat funkce pro práci s klasifikátory. Jedná se o funkci pro natrénování klasifikátorů a funkci pro klasifikaci neroztříděných vzorů. Obě funkce jsou součástí jmenného prostoru `Classification`. Jelikož lze dodaný dataset vad rozdělit do čtyř skupin, rozhodl jsem se pro vytvoření čtyř nezávislých sad klasifikátorů. Tím se vyhnu chybné klasifikaci mezi skupinami.

Vytvořené množiny a natrénované klasifikátory je třeba uložit do souboru. K tomuto účelu jsem vytvořil jmenný prostor `FileManager`, do kterého jsem umístil funkce sloužící pro uložení natrénovaných klasifikátorů a vytvořených trénovacích množin do souboru. Dále se zde nachází funkce, která umožní uložená data opět načíst do proměnných programu.

Jelikož jsem při vytváření rozhraní implementoval některé funkce, které využívám ve více jmenných prostorech, a nejedná se o funkce, které by svým účelem spadaly do některého z dříve zmíněných prostorů, vytvořil jsem další jmenný prostor, do kterého jsem tyto funkce umístil. Tento jmenný prostor jsem pojmenoval `General`.

### 3.1.1 Jmenný prostor Dataset

Uvnitř jmenného prostoru definuji dvě základní struktury pro uložení dat vzorů jednotlivých vad. Jelikož o každém typu vzoru (testovací/trénovací) ukládám data v jiné formě, musel jsem vytvořit dvě odlišné struktury. Pro data vzorů patřících do trénovací množiny jsem definoval strukturu `TrainingSample`, pro testovací vzory strukturu `TestingSample`. Výsledné množiny jsou tvořeny složením struktur do vektorů. Definici struktury pro uložení trénovacího vzoru lze vidět na Výpis 3.1

Hodnoty příznaků využívám při trénování klasifikátorů. Názvy příznaků využiji při sestavení příznakového vektoru, neboť příznakové vektory různých skupin jsou složeny z různých příznaků. Názvy příznaků dále slouží pro lepší orientaci v souboru s uloženými daty. Přijde mi vhodné ukládat názvy příznaků, aby bylo zřejmé, který příznak v příznakovém vektoru se pod danou hodnotou skrývá. Název třídy je zobrazen v okně aplikace, zároveň také slouží k vytvoření matice, která slouží

jako vstup při trénování klasifikátoru. Název souboru ukládám pro jednoznačnou identifikaci vzoru v rámci jedné množiny. Při vkládání nového vzoru je tak možné ověřit, zda se daný vzor v trénovací množině již nenachází. Název souboru zobrazuji také v seznamu vzorů v rámci grafického rozhraní.

Výpis 3.1: Definice struktury pro uložení trénovacího vzoru

```
struct TrainingSample
    {
    std::vector<float> iFeatureValues;
    std::vector<std::string> iFeatureNames;
    std::string iLabel;
    std::string iSampleName;
    };
```

Jmenný prostor `Dataset` dále obsahuje definici struktury pro ukládání dat testovacích vzorů. Testovací vzory se ukládají do vektoru s prvky typu `TestingSample`. Definici této struktury lze vidět na Výpis 3.2.

Výpis 3.2: Definice struktury pro uložení testovacího vzoru

```
struct TestingSample
    {
    std::vector<float> iFeatureValues;
    std::vector<std::string> iFeatureNames;
    Predictions iPredictions;
    std::string iSamplePath;
    std::string iSegmentedPath;
    };
```

Opět ukládám názvy a hodnoty příznaků v příznakovém vektoru do proměnných `iFeatureValues` a `iFeatureNames`. Příznakový vektor je využit při klasifikaci vzoru. Rozdíl mezi testovacím a trénovacím vzorem je v počtu přiřazených tříd k jednomu konkrétnímu vzoru. Trénovacímu vzoru je přiřazena pouze jedna třída při vkládání do množiny. Testovacímu vzoru je však po klasifikaci přiřazeno hned několik tříd. Důvodem přiřazení většího množství tříd je použití většího množství klasifikátorů pro zajištění větší spolehlivosti klasifikace. Výsledné predikce klasifikátorů ukládám do proměnné `iPredictions`, jejíž definici lze vidět na Výpis 3.3.

Predikce jednotlivých klasifikátorů jsou ukládány do proměnných s odpovídajícím názvem. Od každého klasifikátoru je až na jednu výjimku uložen vždy název jedné predikované třídy. Výjimku tvoří pouze neuronová sít, jejímž výstupem je trojice predikovaných tříd. Multivrstvá neuronová síť přiřadí každé třídě (obsažené v trénovacích datech) váhu, jež značí příslušnost testovaného prvku k této třídě.

V grafickém rozhraní zobrazuji tři nejvíce pravděpodobné třídy. Je tedy nutné predikci ukládat jako vektor jednotlivých tříd. Spolu s názvy tříd ukládám také odpovídající váhy příslušností k daným třídám.

Výpis 3.3: Definice struktury pro uložení predikovaných tříd testovacímu vzoru

```
struct Predictions
    {
    std::string bayesLabel;
    std::string knearestLabel;
    std::string svmLabel;
    std::string rtreesLabel;
    std::vector<std::string> annLabels;
    std::vector<float> annWeights;
    };
```

Další rozdíl mezi daty testovacího a trénovacího vzoru spočívá v ukládání názvu souboru s obrazem vady. U testovacího vzoru se ukládá celá cesta k souboru (nikoliv pouze název souboru). Důvodem je možnost zobrazení obrazu testovací vady v grafickém rozhraní, což je umožněno načtením obrazu pomocí cesty k souboru. Cestu k obrazu vady ukládám do proměnné `iSamplePath`. Je uložena cesta, která je vybrána v grafickém rozhraní při nahrávání vzoru. V grafickém rozhraní dále umožňuji zobrazení obrazu se segmentovaným objektem vady, čímž vzniká požadavek na uložení tohoto obrazu a na přiřazení cesty k tomuto souboru pro každý testovací vzor. Při segmentaci testovacího vzoru ukládám obraz s vyznačeným objektem vady do složky `segmentation`. Cestu k segmentovanému obrazu ukládám do proměnné `iSegmentedPath`.

Výše zmíněné proměnné zapouzdřuje struktura `Dataset`, v níž se nacházejí čtyři trénovací množiny a jedna množina testovacích vzorů. Definici této struktury lze vidět na Výpis 3.4

Výpis 3.4: Definice proměnné Dataset

```
struct Dataset
    {
    std::vector<TrainingSample> iTraining9, iTraining10,
                                iTraining11, iTraining12;
    std::vector<TestingSample> iTesting;
    };
```

### Přidání vzoru

Přidání vzoru do trénovací a testovací množiny funguje na obdobném principu. Obě množiny jsou proměnné typu `std::vector`. Přidání do množiny tedy probíhá funkcí `push_back`. Rozdíl nastává při vytváření prvku, který je do množiny vkládán. Jelikož je každá ze struktur `TrainingSample` a `TestingSample` tvořena odlišnými vnitřními proměnnými, rozhodl jsem se pro vytvoření dvou samostatných funkcí. K přidání prvku do trénovací množiny slouží funkce `addTrainingSample`, k přidání prvku do testovací množiny funkce `addTestingSample`. První rozdíl nastává při přiřazení třídy vkládanému vzoru. U trénovacího vzoru probíhá přiřazení třídy vkládanému vzoru současně při vkládání vzoru do množiny. Lze tedy název třídy do struktury uložit uvnitř funkce `addTrainingSample`. U testovacího vzoru však prvek nejprve vložím do množiny pomocí funkce `addTestingSample`, poté zavolám klasifikační funkci `testSample` ze jmenného prostoru `Classification` a třídy do struktury zapisuji až v klasifikační funkci. Jiným řešením by bylo volání klasifikační funkce uvnitř funkce `addTestingSample`, kdy by se v grafickém rozhraní nemusely volat obě funkce. Avšak zvolil jsem první možnost, neboť mi přijde přehlednější.

Při vkládání nového vzoru do množiny se ověřuje, zda se právě vkládaný vzor v množině již nenachází. Kontrola probíhá při vkládání vzoru do trénovací i testovací množiny. Možný duplikát je zjištěn přes název souboru. Funkcím pro přidání vzoru je přes vstupní parametry funkce předána celá cesta k souboru. Celá cesta k souboru je předávána z toho důvodu, aby bylo možné v dalších funkcí volaných z `addTestingSample` a `addTrainingSample` načíst obraz do proměnných programu. Avšak není vhodné kontrolovat duplikát přes celou cestu k souboru. Mohlo by se stát, že dojde k pokusu o vložení souboru stejného názvu ze dvou různých úložišť. Druhý vkládaný soubor by byl vložen a v množině by se nacházely dva stejné vzory. Kontrolu je nutné provádět pouze přes název souboru. K zisku názvu souboru z celé cesty slouží funkce `getFilename` ze jmenného prostoru `General`. Název nově vkládaného vzoru je v případě trénovacího vzoru porovnáván s vnitřní proměnnou `iSampleName` všech stávajících vzorů dané množiny. Pro trénovací množinu je kontrola duplicity zobrazena na Výpis 3.5.

Výpis 3.5: Kontrola duplicity

```
using namespace General;
for(size_t i= 0; i< iTrainingSamples.size(); i++)
    if(iTraining.at(i).iSampleName==getFilename(iFilepath))
        return false;
```

Pokud dojde ke shodě, je funkce `addTrainingSample` ukončena a vrací hodnotu *false*. Obdobně probíhá kontrola při vkládání testovacího vzoru uvnitř funkce `addTestingSample`. Zde probíhá kontrola přes vnitřní proměnnou `iSamplePath`,

v níž je však uložena celá cesta k souboru. Je tedy navíc nutné z cesty získat funkcí `getFilename` pouze název souboru.

Když vkládaný vzor projde přes kontrolu duplicity, je cesta k souboru s obrazem vady předána funkci `getFeatures` ze jmenného prostoru `Description`. Funkce vrací hodnoty a názvy příznaků z příznakového vektoru, které jsou uloženy do dočasných proměnných. Hodnoty příznaků jsou uloženy do proměnné `temp_feature_values`, názvy příznaků jsou uloženy do proměnné `temp_feature_names`. Je vytvořen dočasný prvek `temp_sample` typu `TrainingSample/TestingSample`. Do tohoto prvku je uložen název souboru/cesta k souboru, hodnoty příznaků, názvy příznaků a v případě trénovacího vzoru i název třídy. Poté je prvek vložen na poslední místo vektoru (trénovací/testovací množiny). Na Výpis 3.6 lze vidět část kódu sloužící pro přidání trénovacího vzoru do trénovací množiny.

Výpis 3.6: Přidání trénovacího vzoru

```
TrainingSample temp_sample;

temp_sample.iSampleName = General::getFilename(iFilepath);
temp_sample.iFeatureValues = temp_feature_values;
temp_sample.iFeatureNames = temp_feature_names;
temp_sample.iLabel = iLabel;

iTraining.push_back(temp_sample);
```

### Odebrání vzoru

Pro odebrání vzoru z množiny jsem opět vytvořit dvojici funkcí. Avšak rozdíl mezi funkcemi je pouze v typech proměnných předávaných jako parametry. Tudíž bych mohl vytvořit pouze dvě přetížené funkce s názvem `deleteSample`. Avšak abych zachoval rozdělení funkcí pro testovací a trénovací vzory, pojmenoval jsem funkci pro trénovací data `addTrainingSample` a funkci pro testovací vzory `addTestingSample`. Ukázka kódu, který slouží pro vymazání vzoru z trénovací množiny je na Výpis 3.7.

Výpis 3.7: Odebrání trénovacího vzoru

```
for(size_t i= 0; i< iTrainingSamples.size(); i++)
    if(iTrainingSamples.at(i).iSampleName == iFilename)
        {
        iTrainingSamples.erase(iTrainingSamples.begin() + i);
        iTrainingSamples.shrink_to_fit();
        }
if(iTrainingSamples.size() == 0)
    iTrainingSamples.shrink_to_fit();
```

Nejprve je uvnitř funkce nalezen index prvku v rámci vektoru tvořícího danou množinu. Prvek je hledán na základě hodnoty v `iSampleName` u trénovacího vzoru a na základě hodnoty získané z proměnné `iSamplePath` u testovací množiny. U testovacího vzoru je název z cesty k souboru získáván opět funkcí `getFilename`. Pokud se prvek v dané množině nachází, je prvek z dané množiny odstraněn a vektoru je odpovídajícím způsobem upravena rezervovaná kapacita paměti.

### 3.1.2 Jmenný prostor Description

Při přidávání vzoru do množiny je potřeba z obrazu segmentovat vadu a následně ze segmentovaného objektu vady vypočítat hodnoty do příznakového vektoru. K tomuto účelu slouží funkce umístěné ve jmenném prostoru `Description`. K segmentaci objektu vady jsem implementoval funkci `imageSegmentation` a k výpočtu hodnot příznaků do příznakového vektoru funkci `getFeatures`. Vypočtené příznaky slouží k natrénování klasifikátoru a ke klasifikaci neroztříděných testovacích vzorů do tříd.

#### Segmentace objektu vady

Nejprve je nutné data obrazu načíst do proměnných programu. K tomuto účelu využívám funkci `imread` z knihovny OpenCV, která umožňuje načtení obrazu ve více možných barevných složeních. Pro účely segmentace jsem zvolil načtení v šedotónovém módu. Pro účel následného uložení obrazu se segmentovanou vadou jsem zvolil barevný mód, neboť výsledný segmentovaný objekt zvýrazňuji modrou barvou. Použití této funkce pro dva odlišné barevné módy lze vidět na Výpis 3.8. Cesta k souboru se nachází v proměnné `iFilename`.

Výpis 3.8: Načtení obrazu do proměnných programu

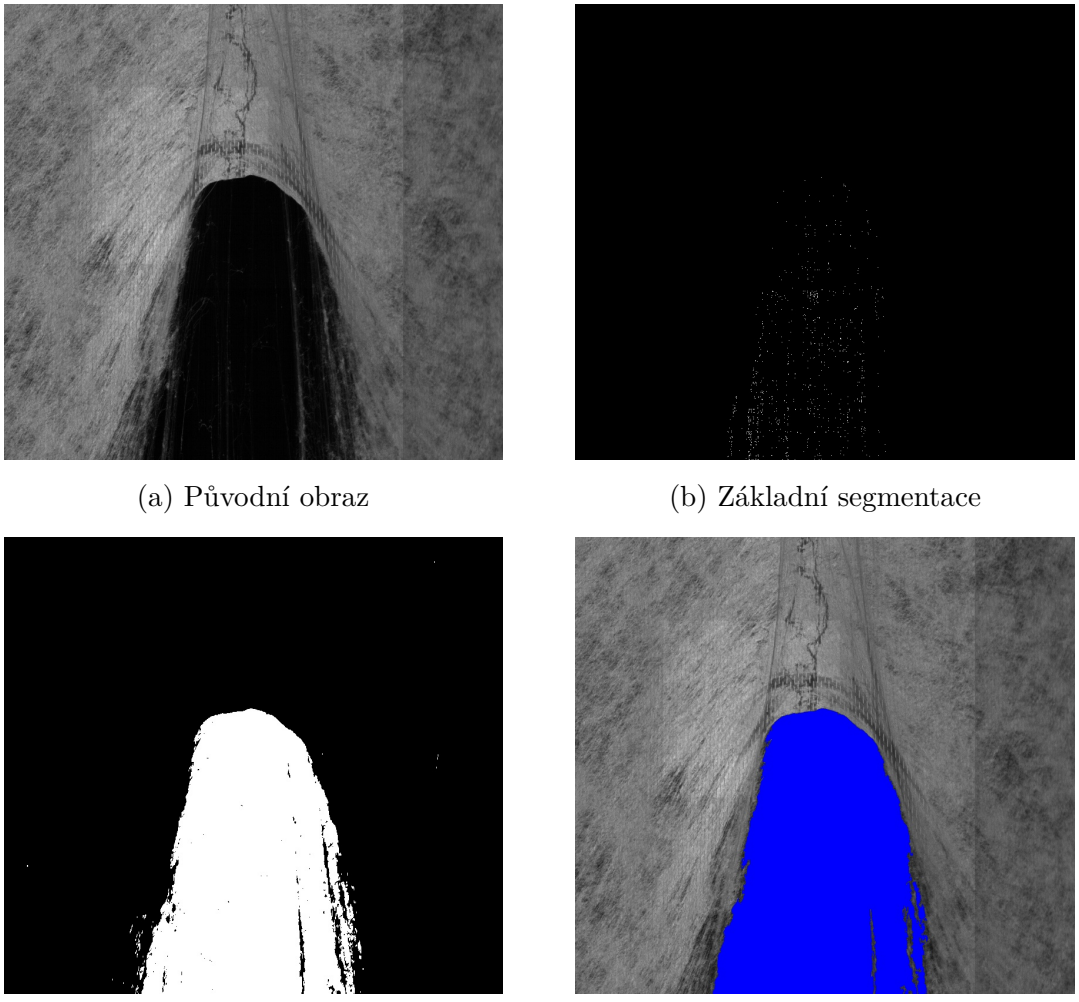
```
cv::Mat input_grayscale, input_color;
input_grayscale = cv::imread(iFilename,cv::IMREAD_GRAYSCALE);
input_color = cv::imread(iFilename,cv::IMREAD_COLOR);
```

Spolu s obrazem se v souboru *jpg* nacházejí také informace o obrazu. Mezi takové informace patří například: datum pořízení, čas pořízení, ID třídy (informace o kameře - podsvitu/nadsvitu), vypočtené příznaky nebo poloha vady v obrazu. Parametry, které z obrazu získávám, jsou:

- resultID - o jakou skupinu vad se jedná,
- QMIN - minimální jasová úroveň objektu vady,
- QMAX - maximální jasová úroveň objektu vady,
- limit1 - jasová hodnota pozadí světlých vad.

Proměnnou *resultID* používám u trénovacích dat pro ověření, zda uživatel v grafickém rozhraní přiřadil ke zvolenému vzoru třídu ze správné skupiny. U testovacích

dat slouží tento parametr k volbě vhodné skupiny klasifikátorů. Parametry *QMIN*, *QMAX* a *limit1* využívám při samotné segmentaci. Konkrétně *QMAX* a *limit1* využívám při segmentování světlých vad a parametr *QMIN* využívám při segmentaci tmavých vad. Chybějící parametr *limit* pro tmavé vady dopočítávám jako 95. kvantil jasových úrovní pixelů pozadí. K zisku zmíněných parametrů z obrazu jsem implementoval funkci `getImageInfo`. Parametry jsou v souboru s obrazem uloženy v pevně daném formátu. Funkce tedy postupně prochází obraz v binární formě a hledá definovaný formát. Pokud hledaný parametr nebyl nalezen, funkce vrací zápornou hodnotu. Hledanými parametry jsou jasové hodnoty, proto nepředpokládám, že by byly záporné. Pokud funkce vrací zápornou hodnotu, je vyhozena výjimka, která informuje o neúspěšném nalezení daného parametru. Výsledná segmentace je složena z výsledků dvou dílčích segmentací. Postup segmentace, vstupní obraz a výsledný obraz lze vidět na Obr. 3.1.



(a) Původní obraz

(b) Základní segmentace

(c) Limitní segmentace

(d) Výsledek segmentace

Obr. 3.1: Proces segmentace obrazu

V první části hledám ty pixely vady, o nichž mohu jednoznačně říci, že jsou součástí vady. Pro tmavé vady jsou to pixely s hodnotou nižší než *QMIN*, pro světlé vady jsou to pixely s jasovou hodnotou vyšší než *QMAX*. V druhé části pak pro tmavé vady hledám pixely, jež mají jasovou úroveň nižší než je jasová hodnota pozadí, pro světlé vady naopak hledám ty pixely, jež mají jasovou úroveň vyšší než pozadí. Pro každou skupinu vad je tak třeba nastavit dvojici parametrů: `threshold` (první část segmentace) a `limit` (druhá část segmentace). Nastavení jednotlivých parametrů pro všechny čtyři skupiny vad lze vidět na Výpis 3.9. Jelikož jsem po vyzkoušení prvotních předpokladů zjistil, že je třeba jednotlivé hodnoty upravit, ve výsledném nastavení se objevují empiricky získané hodnoty.

Výpis 3.9: Nastavení parametrů segmentace pro jednotlivé skupiny

```
// tmavé vady
if(iID == 9)
    {
    threshold = QMIN + 2;
    limit = percentile95(input_grayscale, QMAX + 10);
    }
if(iID == 11)
    {
    threshold = QAVG;
    limit = percentile95(input_grayscale, QAVG + 5);
    }
// světlé vady
if(iID == 10)
    threshold = QMIN;
if(iID == 12)
    {
    threshold = QMAX - 20;
    limit -= 55;
    }
```

Po nastavení parametrů následuje již samotná segmentace. Na Obr. 3.1b je zobrazena první část segmentace. Z obrázku je patrné, že touto částí segmentace je segmentováno menší množství pixelů. U některých dodaných vzorů jsem se setkal s tím, že výsledkem této segmentace nebyl žádný pixel ve výstupním obrazu. Proto jsem se rozhodl, že funkci pro segmentaci zapouzdřím do smyčky `while`, která skončí až v případě, že ve výsledném obrazu dojde k segmentaci nenulového počtu pixelů. V každé iteraci je upravována proměnná `threshold`. U tmavých vad dochází ke zvyšování hodnoty, u světlých vad dochází ke snižování této hodnoty. Část kódu zajišťující první část segmentace lze vidět na Výpis 3.10.

Výpis 3.10: První část segmentace

```
cv::Mat object_points;
while(object_points.rows <= 0)
    {
    // tmavé vady
    if(iID == 9 || iID == 11)
        {
        cv::threshold(input_grayscale, first_segmentation,
        threshold, 255, cv::THRESH_BINARY_INV);
        threshold += 1;
        }
    // světlé vady
    if(iID == 10 || iID == 12)
        {
        cv::threshold(input_grayscale, first_segmentation,
        threshold, 255, cv::THRESH_BINARY);
        threshold -= 1;
        }
    cv::findNonZero(input_binary, object_points);
    }
```

K segmentaci obrazu využívám funkci `threshold` z knihovny OpenCV. Funkce umožňuje několik typů segmentací. Pro světlé vady volím `THRESH_BINARY`. Tento typ segmentace nastaví ve výstupním obrazu do jasové hodnoty 255 (tuto hodnotu také volím) ty pixely, jež měly ve vstupním obrazu jasovou úroveň vyšší, než je jasová úroveň prahu. Vstupním obrazem je `input_grayscale`, výstupní obraz je uložen do `first_segmentation`, prahová úroveň je nastavována proměnnou `threshold`. Naopak u tmavých vad používám typ `THRESH_BINARY_INV`, jež ve výstupním obrazu nastaví do hodnoty 255 ty pixely, jež měly ve vstupním obrazu jasovou hodnotu nižší, než je prahová jasová úroveň. K segmentaci druhé části opět využívám funkci `threshold`, avšak prahová úroveň segmentace je nastavována proměnnou `limit` a výsledek je ukládán do obrazu `second_segmentation`. Výsledek této části lze vidět na Obr. 3.1c.

Po provedení segmentací je potřeba obě části spojit do jedné. Tedy z obrazu `second_segmentation` vzít pouze ty oblasti, v nichž se nachází alespoň jeden výsledný pixel z obrazu `first_segmentation`. K tomuto účelu využívám kontury, které mi definují hranice oblastí v jednotlivých obrazech a následně hledám pouze ty, které se překrývají. K zisku kontur objektů z obrazu využívám funkci `findContours` z knihovny OpenCV. Část kódu, která z obrazu získá pixely kontur lze vidět na Výpis 3.11. Zisk kontur z druhé segmentace probíhá obdobným způsobem.

Výpis 3.11: Zisk kontur objektů

```
std::vector<std::vector<cv::Point>> first_contours;
cv::findContours(first_segmentation, first_contours,
cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);
```

Poté procházím kontury z první segmentace a zjišťuji, zdali se nachází v některé z kontur druhé segmentace. Pokud je kontura z první segmentace obsažena v některé kontuře z druhé segmentace, je daný objekt z druhé segmentace uznán jako část vady. Tímto způsobem vzniká výsledný obraz segmentace, který je možné vidět na Obr. 3.1d. Výstupem funkce `imageSegmentation` je dvojice obrazů. Jedním je oříznutý obraz Obr. 3.1d (bez modrého vyznačení segmentace), druhým je obraz se segmentovaným objektem v binární podobě. Tyto obrazy slouží k výpočtu geometrických a radiometrických příznaků pomocí funkce `getFeatures`.

### Výpočet příznaků

Jakmile je ze vstupního obrazu získán binární a šedotónový obraz vady, lze přistoupit k výpočtu hodnot příznakového vektoru. K tomuto účelu jsem implementoval funkci `getFeatures`. Výše zmíněná funkce `imageSegmentation` je v rámci programu volána z funkce `getFeatures`. Segmentací obrazu vady vznikají dva obrazy: `croopedBinary` (binární obraz) a `croopedGrayscale` (šedotónový obraz). Každá skupina vad používá pro klasifikaci jinou sestavu příznaků, avšak pro každý vzor vady vypočítávám všechny příznaky. Z těchto příznaku se pak volí vhodná kombinace při trénování a testování klasifikátoru. V rámci funkce `getFeatures` vypočítávám ze segmentovaného objektu vady následující sadu příznaků:

- Area - obsah,
- Solidity - konvexnost,
- Aspect_ratio - poměr stran,
- Extent - rozsah,
- Max_width - maximální šířka,
- Max_ height - maximální výška,
- Angle - úhel natočení,
- QMAX - maximální jasová úroveň,
- QMIN - minimální jasová úroveň,
- Q25 - 25. kvantil jasové úrovně,
- Q50 - 50. kvantil jasové úrovně,
- Q75 - 75. kvantil jasové úrovně,
- Q50_out - 50. kvantil jasové úrovně mimo objekt vady,
- Homogenity - homogenita objektu vady.

Obsah objektu vady získávám pomocí funkce `findNonZero` z knihovny OpenCV. Funkci je předán obraz v binární podobě (získaný funkcí `imageSegmentation`) a matice pro uložení nenulových prvků. Počet řádku matice odpovídá počtu nenulových pixelů vstupního obrazu. Část kódu zajišťující výpočet obsahu a jeho následné vložení do výstupního vektoru funkce lze vidět na Výpis 3.12.

Výpis 3.12: Výpočet obsahu objektu

```
cv::Mat object_points;
cv::findNonZero(croopedBinary, object_points);
int area = object_points.rows;
iFeatureNames.push_back("Area");
iFeatureValues.push_back((float)area);
```

Obsah je využit i při výpočtu některých dalších parametrů. Konvexnost se vypočítá jako plocha objektu vady ku ploše jejího konvexního obalu. Ke zjištění konvexního obalu objektu vady využívám funkci `convexHull` z knihovny OpenCV. Nejprve je nalezen konvexní obal objektu. Konvexní obal je tvořen pixely hraničních hodnot (podobně jako kontura). Pomocí funkce `drawContours` z knihovny OpenCV vykreslím tuto konturu do obrazu. Vykreslená kontura je zároveň vyplněna. Následně spočítám obdobně jako u obsahu nenulové prvky, jejichž počet odpovídá ploše konvexního obalu. Poté již pouze vydělím obsah plochou konvexního obalu a hodnotu s názvem vložím do výstupních vektorů. Stejným způsobem počítám i hodnotu rozsahu. Jedná se o hodnotu poměru obsahu opsaného obdélníku ku obsahu objektu vady. Následující část kódu na Výpis 3.13 ukazuje postup výpočtu. Nejprve je získán nejmenší možný opsaný obdélník, poté je vypočítána a uložena hodnota rozsahu.

Výpis 3.13: Výpočet hodnoty rozsahu

```
cv::RotatedRect rotated_rect=cv::minAreaRect(object_points);

float extent = float(area) / float(rotated_rect.size.width *
rotated_rect.size.height);

iFeatureNames.push_back("Extent");
iFeatureValues.push_back(extent);
```

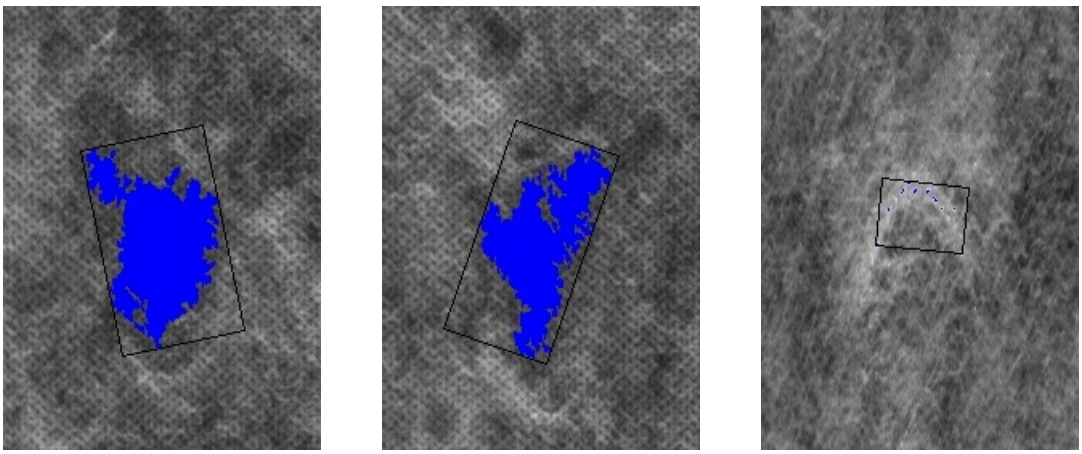
Z geometrických příznaků dále počítám poměr stran, maximální šířku, maximální výšku objektu vady a úhel natočení. K výpočtu všech čtyř příznaků využívám proměnnou `rotated_rect` získanou v části kódu na Výpis 3.13. Textilie, ze které se snímají vzorky vad, se pohybuje pod kamerou pouze v horizontálním směru. Proto je potřeba některé získané hodnoty z proměnné `rotated_rect` upravit. Ze struktury s názvem `RotatedRect` používám vnitřní proměnné `width` (šířka), `height` (výška)

a `angle` (úhel natočení). Hodnoty zmíněných proměnných jsou pro vady z Obr. 3.2 zobrazeny v tabulce 3.1.

Tab. 3.1: Neupravené hodnoty parametrů výšky, šířky a úhlu pro tři varianty vad

| | Šířka [px] | Výška [px] | Úhel [°] |
|---|---|---|---|
| Varianta 1 | 77.2 | 130.9 | -11.5 |
| Varianta 2 | 137.9 | 68.1 | -70.9 |
| Varianta 3 | 41.7 | 54.3 | -84.6 |

Ze tří variant jsou správně určeny pouze hodnoty parametrů u varianty 1, neboť jsou vypočtené parametry orientovány ve vertikálním směru. Hodnota výšky je vyšší než hodnota šířky a úhel odpovídá natočení. U zbylých dvou variant parametry neodpovídají posunu textilie ve vertikálním směru. Je tedy třeba tuto hodnotu upravit.



(a) Varianta 1 (b) Varianta 2 (c) Varianta 3

Obr. 3.2: Různé verze zisku opsaného obdélníku

Korekce takovýchto chyb probíhá na základě hodnoty poměru šířky/výšky a úhlu natočení. Pokud je hodnota parametru výšky nižší než hodnota parametru šířky a úhel natočení je mimo rozsah (-25°, +25°) (empiricky zjištěná hodnota), je třeba korigovat parametr úhlu natočení (±90°) a zaměnit parametry výšky a šířky. Tuto variantu lze vidět na Obr. 3.2b. Pokud je hodnota parametru šířky menší než hodnota parametru výšky, ale úhel natočení je opět mimo rozsah (-25°, +25°), je třeba opět korigovat úhel natočení a parametry výšky a šířky. Korigované parametry jsou zobrazeny v tabulce 3.2. Hodnota šířky je zapsána do příznaku `max_width`, hodnota výšky do příznaku s názvem `max_height`.

Tab. 3.2: Upravené hodnoty parametrů výšky, šířky a úhlu pro tři varianty vad

| | Šířka [px] | Výška [px] | Úhel [°] |
|---|---|---|---|
| Varianta 1 | 77.2 | 130.9 | -11.5 |
| Varianta 2 | 68.1 | 137.9 | 19.1 |
| Varianta 3 | 54.3 | 41.7 | 5.4 |

Dále používám příznak poměru stran. Do příznaku s názvem `Aspect_ratio` se ukládá podíl šířky ku výšce opsaného obdélníku, jejichž hodnoty jsou použity z již zmíněných proměnných `max_width` a `max_height`.

Do příznakového vektoru vkládám také radiometrické příznaky. K výpočtu radiometrických příznaků využívám histogram. Jelikož pracuji i s body v blízkém okolí vady, je třeba vypočítat histogram pro pixely uvnitř vady a histogram pro pixely v blízkém okolí vady. Před výpočtem histogramu v blízkém okolí vady je potřeba získat pixely v tomto okolí. Podobně jako je pro objekt vady určen binární obraz `croopedBinary`, zanesu pixely okolí vady do binárního obrazu `surroundings`. Zisk tohoto obrazu spočívá nejprve v zisku kontur objektu vady a následném zanesení těchto kontur do obrazu `surroundings` s šířkou požadovaného okolí (volím 20).

Výpis 3.14: Výpočet histogramů

```
for(size_t c = 0; c < contours.size(); c++)
    cv::drawContours(surroundings, contours, c, 255, 20);

surroundings = surroundings & (255~- croopedBinary);

for(int y = 0; y < croopedGrayscale.rows; y++)
  for(int x = 0; x < croopedGrayscale.cols; x++)
    if(croopedBinary.at<uchar>(y, x) > 0)
      histogram[croopedGrayscale.at<uchar>(y, x)]++;
    else if(surroundings.at<uchar>(y, x) > 0)
      {
      histogram_out[croopedGrayscale.at<uchar>(y, x)]++;
      area_out++;
      }
```

Poslední částí je zajištění, že se v obrazu `surroundings` nebudou nacházet pixely objektu vady, což se provede logickým součinem obrazu okolních pixelů s negovaným obrazem vady. Používám zde pojem binární, ale v programu využívám hodnot 0 a 255. Zisk obrazu okolí a výpočet histogramů lze vidět na Výpis 3.14. Získané kontury se nacházejí v proměnné `contours`.

Jako radiometrické příznaky vypočítávám určité hodnoty kvantilů zmíněné na začátku této podkapitoly. Ukázka výpočtu `Q25` je na Výpis 3.15.

Výpis 3.15: Výpočet 25. kvantilu

```
int temp_q25 = 0;
iFeatureNames.push_back("Q25");
for(int~i~= 0;~i~< 256; i++)
    {
    temp_q25 = temp_q25 + histogram[i];
    if(temp_q25 >= (int) (area * 0.25))
        {
        iFeatureValues.push_back((float) i);
        break;
        }
    }
```

Posledním příznakem, jehož hodnotu počítám a ukládám do příznakového vektoru, je homogenita objektu vady. Homogenitu zjišťuji pomocí matice společného výskytu. Matice vyjadřuje prostorový vztah mezi pixely. Z takto získané matice lze získat některé statistické parametry, například kontrast, energii nebo homogenitu, jejíž hodnotu uložím do příznakového vektoru.

### 3.1.3 Jmenný prostor Classification

Funkce umístěné uvnitř tohoto jmenného prostoru zajišťují natrénovaní klasifikátorů na základě trénovací množiny a přiřazení tříd k neroztříděným vzorům testovací množiny. Funkce zajišťující natrénovaní klasifikátorů jsou volány uvnitř funkce `trainClassifiers`. Pro přiřazení tříd k neroztříděným vzorům jsem implementoval funkci `testSample`. Nejprve jsem zkoušel použití pouze jednoho klasifikátoru, a to neuronové sítě. Zjistil jsem však, že jiné typy klasifikátorů dosahují v některých případech vyšších úspěšností. Jelikož nelze zvolit jeden typ klasifikátoru, který by pro všechny zvolené třídy dosahoval vyšších úspěšností než zbylé typy, rozhodl jsem se pro použití sady více klasifikátorů. Pro svoji práci jsem volil následující typy klasifikátorů: [10]

- Normal Bayes Classifier - Bayesův klasifikátor,
- K-Nearest - nejbližší soused,
- Support Vector Machine - metoda podpůrných vektorů,
- Random Trees - rozhodovací stromy,
- ANN-MLP - umělá vícevrstvá neuronová síť.

Pro lepší čitelnost kódu a snazší vývoj rozhraní klasifikátorů jsem všechny typy zabalil do jedné struktury s názvem `Classifiers`. Definici této struktury lze vidět

na Výpis 3.16. Uvnitř struktury se také nachází proměnná `iLabels`. Jedná se o vektor, ve kterém se nacházejí názvy všech tříd, které byly použity pro natrénování daných klasifikátorů. Tento vektor slouží pro vytvoření trénovacích dat a klasifikaci neroztříděných vzorů, neboť výstupem některých typů klasifikátorů je pouze index třídy.

Výpis 3.16: Struktura obsahující různé typy klasifikátorů

```
struct Classifiers
    {
    cv::Ptr<cv::ml::NormalBayesClassifier> bayesClassifier;
    cv::Ptr<cv::ml::KNearest> knearestClassifier;
    cv::Ptr<cv::ml::SVM> svmClassifier;
    cv::Ptr<cv::ml::RTrees> rtreesClassifier;
    cv::Ptr<cv::ml::ANN_MLP> annClassifier;
    std::vector<std::string> iLabels;
    };
```

V grafickém rozhraní je umožněno vymazání vnitřních proměnných aplikace (trénovací množina, testovací množina, klasifikátory). Pro jednodušší mazání struktury `Classifiers` jsem implementoval funkci `clear`, jejíž část kódu pro vymazání Bayesova klasifikátoru lze vidět na Výpis 3.17.

Výpis 3.17: Vymazání dat struktury Classifiers

```
if(!iClassifier.bayesClassifier.empty())
    {
    iClassifier.bayesClassifier->clear();
    iClassifier.bayesClassifier = nullptr;
    }
```

Dále je v aplikaci potřeba funkce, jejíž návratová hodnota bude záviset na tom, zda jsou všechny klasifikátory uvnitř struktury `Classifiers` natrénovány. K tomuto účelu jsem vytvořil funkci `trained`. Návratovou hodnota funkce je *true* v případě, že je každý z klasifikátorů natrénován. V opačném případě (alespoň jeden natrénován není) je vrácena logická hodnota *false*. Funkce je v grafickém rozhraní využita pro zakázání/povolení přidávání vzorů do testovací množiny.

### Natrénování klasifikátorů

Před natrénováním klasifikátorů je potřeba vytvořit data ve správném formátu. Data jsou klasifikátorům předávána formou matic a vektorů. Matice mohou být orientovány v řádkovém nebo sloupcovém směru. Zvolil jsem sloupcovou orientaci matic. Každý trénovací vzor je v matici reprezentován jedním sloupcem. Klasifikátoru je

třeba předat příznakové vektory trénovacích vzorů a třídy přiřazené k jednotlivým vzorům (odezvy).

Pro všechny typy klasifikátorů se shoduje formát matice obsahující příznakové vektory. Matice příznaků je tvořena funkcí `createFeatureMatrix`, uvnitř které dochází ke skládání příznakových vektorů v horizontálním směru (sloupcový formát matice). Pro každou skupinu vad je vhodná jiná kombinace příznaků. Zároveň je třeba vybrat vhodnou kombinaci příznaků pro každý typ klasifikátoru. Hodnoty příznaků jsou vyčítány z vnitřní proměnné `iFeatureValues` struktury `TrainingSample`. V tomto vektoru se však nacházejí všechny vypočtené příznaky. Před spojením příznakových vektorů do matice je třeba vybrat z `iFeatureValues` pouze některé hodnoty. K výběru hodnot slouží funkce `createFeatureVector`, jejímiž vstupními parametry jsou: číslo verze příznakového vektoru, názvy a hodnoty příznaků v příznakovém vektoru daného vzoru. Výstupem funkce je vektor hodnot sloužící pro tvorbu příznakové matice. S verzí příznakového vektoru pracuji v bitovém formátu. Každý bit tohoto čísla odpovídá jednomu příznaku, což je ukázáno v tabulce 3.3. V této tabulce je zároveň uveden příklad pro kombinaci příznakového vektoru s číslem 1974. Je-li bit na daném indexu roven jedné, je hodnota příznaku s tímto názvem vložena do výstupního vektoru funkce.

Tab. 3.3: Výběr kombinace příznaků

| Bitová hodnota | Název příznaku | Verze - 1974 |
|---|---|---|
| $2^0$ | Solidity | 0 |
| $2^1$ | Angle | 1 |
| $2^2$ | Extent | 1 |
| $2^3$ | Max_width | 0 |
| $2^4$ | Max_height | 1 |
| $2^5$ | Aspect_ratio | 1 |
| $2^6$ | Area | 0 |
| $2^7$ | QMIN | 1 |
| $2^8$ | QMAX | 1 |
| $2^9$ | Q25 | 1 |
| $2^{10}$ | Q50 | 1 |
| $2^{11}$ | Q50_out | 0 |
| $2^{12}$ | Q75 | 0 |
| $2^{13}$ | Homogenity | 0 |

Formát matice odezev (příslušnosti trénovacích vzorů ke třídám) již není stejný pro všechny typy klasifikátorů. Vstupním formátem dat se neuronová síť liší od

ostatní typů klasifikátorů, neboť neuronová síť požaduje odezvy v maticovém formátu a ostatní klasifikátory ve vektorovém formátu. Pro oba typy jsem vytvořil odlišné funkce. K vytvoření vektorů odezev slouží funkce `createLabelVector`. Funkce vezme vnitřní proměnnou `iLabel` struktury `TrainingSample`, převede ji do proměnné typu `int` a vloží ji do výsledného vektoru. Tento vektor je poté předán klasifikátoru spolu s maticí příznaků. Převod názvu třídy na číselnou hodnotu probíhá pomocí zjištění indexu pozice názvu této třídy ve vnitřní proměnné `iLabels` struktury `Classifiers`. Tato vnitřní proměnná je uvnitř funkce `trainClassifiers` vytvořena před zavoláním funkce `createLabelVector`. K vytvoření matice tříd pro neuronovou síť jsem implementoval funkci `createLabelMatrix`. Třída daného vzoru je reprezentována vektorem s hodnotami 0 a 1. Matice tříd je opět sloupcového formátu. Počet sloupců tak odpovídá počtu prvků trénovací množiny a počet řádků odpovídá celkovému počtu tříd uvnitř dané trénovací množiny. Každý řádek zde představuje jednu třídu. V každém sloupci se nachází pouze jedna hodnota 1 a to vždy na takovém řádku, který odpovídá příslušnosti daného prvku k dané třídě. Ostatní prvky v daném sloupci mají nulovou hodnotu.

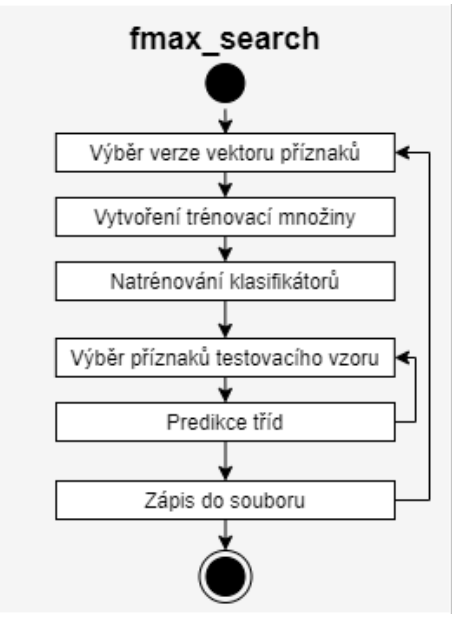
Po převedení příznakových vektorů a názvů tříd do příslušného formátu jsou tato data předána funkci `train` z knihovny OpenCV, která již slouží pro natrénování klasifikátoru. Návratovou hodnotou funkce `trainClassifiers` je proměnná typu `Classifiers`, jež má natrénovány všechny klasifikátory a naplněný vektor `iLabels`.

### Přiřazení tříd testovacím vzorům

Klasifikace probíhá uvnitř funkce `testSample` na základě hodnot příznaků uložených v proměnné `iFeaturesValues` struktury `TestingSample`. Výběr příznaků probíhá pomocí stejné funkce jako při trénování klasifikátoru (`createFeatureVector`). Příznaky jsou předány funkci `predict` z knihovny OpenCV. Funkce vrací pro neuronovou síť vektor hodnot. Velikost vektoru odpovídá počtu tříd v trénovací množině, na které byl daný klasifikátor natrénován. Jednotlivé prvky vektoru odpovídají váze příslušnosti testovaného vzoru k třídě na odpovídajícím indexu. Do vnitřní proměnné `iPredicitons` struktury `TestingSample` ukládám tři třídy s nejvyšší váhou příslušnosti. Pro ostatní typy klasifikátorů vrací funkce `predict` index třídy. Do proměnné `iPredctions` tedy ukládám název třídy z vnitřní proměnné `iLabels` struktury `Classifiers`, který se nachází na daném indexu. K zobrazení výsledků v rámci aplikace implementuji funkci `LUTabel`, uvnitř které dojde k převodu názvu třídy z číselného do slovního formátu. Například pro skupinu 9 dojde při zařazení vzoru do třídy *90901* k zobrazení výsledku *„Necistota-podsvit“*.

### Funkce pro výběr verze příznakového vektoru

Pro účel tvorby a ladění rozhraní klasifikátoru jsem nejprve používal pouze vícevrstvou umělou neuronovou síť. Pro klasifikaci jsem používal všechny vypočtené příznaky a prozatím jsem neřešil spolehlivost zvoleného klasifikátoru. Jakmile jsem se však snažil zvýšit úspěšnost klasifikace, zjistil jsem, že i jiné typy klasifikátorů dosahují pro některé třídy vyšších úspěšností než neuronová síť. Rozhodl jsem se tedy rozhraní rozšířit o další typy klasifikátorů. Po dokončení rozhraní bude potřeba pro každý typ klasifikátoru v každé ze čtyř skupin vad zvolit vhodnou kombinaci příznaků v příznakovém vektoru. Je třeba spočítat spolehlivost pro různé kombinace příznaků pro různé typu klasifikátorů u každé ze čtyř skupin vad. Vytvořil jsem funkci, která postupně prochází všechny kombinace příznaků a u každého z klasifikátorů výsledné úspěšnosti ukládá do souboru. Funkci jsem pojmenoval `fMaxSearch`. Princip činnosti funkce je znázorně na Obr. 3.3.




Obr. 3.3: Princip funkce fMaxSearch

Základním úkolem funkce je pro každou možnou kombinaci příznaků z příznakového vektoru vypočítat úspěšnost klasifikace všech typů klasifikátorů. K výběru hodnot do příznakového vektoru dochází ve funkci `createFeatureVector`. Funkci je předána spolu s daty vzoru také verze příznakového vektoru jako šestnáctibitové číslo. Na základě verze jsou funkcí vráceny hodnoty příznaků daného vzoru. Výběr kombinace příznaků probíhá stejným způsobem při vytváření trénovací množiny a při klasifikaci testovacího vzoru.

Funkce hledající nejvyšší spolehlivost tak nejprve vybere kombinaci příznaků pro

trénovací množinu. Trénovací množinu vytvoří. Poté vytvoří a natrénuje klasifikátory na datech právě vytvořené trénovací množiny. Následně projde testovací množinu, klasifikuje každý vzor různými klasifikátory (při klasifikaci je použita kombinace příznaků shodná s kombinací příznaků použité u trénovací množiny). Nakonec do *csv* souboru zapíše výslednou spolehlivost dané kombinace pro každý klasifikátor. Spolehlivost je vypočítávána na základě umístění souboru s obrazem. Z cesty k souboru se vyjme pouze název složky, ve které je soubor uložen (je ignorována složka *test*). Tento název je poté porovnáván s názvy tříd uložených ve vnitřní proměnné `iPredictions` struktury `TestingSample`. Tímto způsobem projde funkce všechny možné kombinace příznaků. Pro každou kombinaci je vždy nově vytvořena trénovací množina a nově natrénován každý z klasifikátorů. Šipkami zpětných vazeb je demonstrována smyčka `for`. Funkce není použitelná pro klasické používání aplikace, neboť předpokládám, že soubory s neznámou příslušností k třídě nebudou roztříděny v příslušných složkách. Funkci jsem však použil při nastavování a ladění klasifikátorů, z tohoto důvodu se o ní zde zmiňuji.

### 3.1.4 Jmenný prostor FileManager

Před uložením dat je třeba definovat, do kterého souboru tato data uložit. K výběru cesty bude sloužit funkce `OpenDialog`, která je však součástí grafického rozhraní. Proto zde budu uvažovat stav, kdy je cesta k souboru již známá. Zvolená cesta je poté předána funkci `writeXML` jako vstupní parametr. Funkce data do tohoto souboru uloží v předem definovaném formátu, aby bylo možné data do programu opětovně načíst funkcí `readXML`. Pro výběr souboru, ze kterého má dojít k načtení dat, bude opět sloužit funkce `OpenDialog`.

#### Funkce writeXML

V rámci aplikace jsou data uložena ve dvou hlavních strukturách. Trénovací množiny jsou uloženy ve strukturách `Dataset` a jednotlivé klasifikátory zapouzdřené uvnitř struktur `Classifiers`. K zapsání vnitřních proměnných obou struktur jsem vytvořil dvě oddělené funkce, které jsou poté volány ve funkci `writeXML`.

Funkcí `writeClassifiers` dojde k uložení všech typů klasifikátorů jedné skupiny a vektoru s názvy tříd této skupiny. Do souboru se automaticky zapisují data všech typů klasifikátorů. Pro ukládání dat klasifikátorů používám funkci `write` z knihovny OpenCV. Při ukládání může dojít k situaci, kdy klasifikátory z některé skupiny nejsou natrénovány. Při situaci, kdy je uložen funkcí `write` prázdný klasifikátor, dochází při následném načtení funkcí `read` (slouží k načtení klasifikátorů ve funkci `readXML`) k vyhození výjimky. Tento stav je potřeba ošetřit. Před data jednotlivých klasifikátorů ukládám proměnnou s názvem `State`. Je-li proměnná v hodnotě 1,

byl uložený klasifikátor natrénovaný, hodnota 0 značí prázdný klasifikátor. Tento mechanismus je využit při načítání dat funkcí `readXML`. Proměnná `State` a data příslušného klasifikátoru jsou zabalena v jednom uzlu.

Dále jsou do souboru zapsána data jednotlivých trénovacích množin. K uložení jedné trénovací množiny slouží funkce `writeDatasetXML`. Formát ukládání dat trénovací množiny je ukázán pro trénovací množinu 9 na Výpis 3.18.

Výpis 3.18: Uložení trénovací množiny do souboru

```
<Data9>
  <nSamples>1</nSamples>
  <nFeatures>4</nFeatures>
  <FeatureNames>
    Area Solidity Aspect_ratio Extent</FeatureNames>
  <SampleData0>
    <SampleName>"name.jpg"</SampleName>
    <Features>
      <Area>1228.</Area>
      <Solidity>8.7277895212173462e-01</Solidity>
      <Aspect_ratio>5.2631580829620361e-01</Aspect_ratio>
      <Extent>7.1812868118286133e-01</Extent></Features>
    <Label>"90901"</Label>
  </SampleData0>
</Data9>
```

Data jsou zapouzdřena v uzlu *„Data9“*. Ukládá se informace o počtu vzorů, která se využívá při následovném načítání zpět do aplikace. Ze stejného důvodu ukládám i počet příznaků. Dále ukládám názvy příznaků, aby došlo ke korektnímu načítání hodnot příznaků. Dále pomocí funkce `for` postupně uložím data všech vzorů. Každý vzor je v samostatném uzlu, v rámci kterého jsou příznaky uloženy opět v dalším samostatném uzlu.

### Funkce readXML

Funkce slouží k opětovnému načtení dat ze souboru *xml* do proměnných aplikace. Cesta k souboru je opět získána přes dialogové okno funkcí `OpenDialog`. Při načtení dat je postupováno obdobně jako při jejich ukládání. Co se týče klasifikátorů, je pro každý typ klasifikátoru vytvořen samostatný uzel, ve kterém je kontrolován stav uložení klasifikátoru. Pro načtení dat klasifikátoru využívám funkci `read` z knihovny OpenCV. Načtení všech typů klasifikátorů a vektorů s názvy tříd je zapouzdřeno ve funkci `readClassifiersXML`, která je ve funkci `readXML` volána pro klasifikátor z každé skupiny vad zvlášť.

Trénovací množiny se v *xml* souboru nacházejí v uzlu s názvem příslušné množiny. Trénovací množina každé skupiny vad je uvnitř funkce `readXML` opět načítána pomocí funkce `readDatasetXML` zvlášť. Nejprve je načten počet vzorů a velikost příznakového vektoru množiny. Načítání jednotlivých vzorů je podmíněno existencí alespoň jednoho a více vzorů v uložené množině. Dále jsou načítána data jednotlivých vzorů, která se nacházejí v uzlech s názvy `SampleDataX`, kde $X$ indikuje pozici daného vzoru. Pro každý vzor je v příslušném uzlu uložen název vzoru, hodnoty a názvy příznaků a název třídy, do které daný vzor patří. Při načítání vzorů ze souboru do programu je třeba kontrolovat, zda se vzor s daným názvem v množině již nenachází. Tato kontrola probíhá prostřednictvím vnitřní proměnné `iSampleName` struktury `TrainingSample`.

### 3.1.5 Jmenný prostor General

První funkcí umístěnou ve jmenném prostoru `General` je funkce `getIndex`. Funkcí je předán vektor s hodnotami a hodnota, jež je v daném vektoru hledána. Návratovou hodnotou je pozice tohoto prvku v daném vektoru. Funkci využívám např. ve funkci `createFeatureVector` pro nalezení indexu názvu příznaku, abych mohl následně hodnotu na tomto indexu z vektoru s hodnotami příznaků vložit do výsledného vektoru. V tomto případě jsou prvky vektoru typu `string`.

Další funkcí je `getID`. Vstupním parametrem funkce je název třídy typu `string`. Návratovou hodnotou funkce je ID skupiny, do níž daná třída patří. Funkce je využívána při vkládání vzoru do množiny. Např. v grafickém rozhraní se zadává název třídy, ze kterého se touto funkcí zjistí ID skupiny, a díky tomu lze funkci `addTrainingSample` pomocí vstupních parametrů předat vhodnou trénovací množinu.

Poslední funkcí je `getFilename`, která slouží k získání názvu souboru z cesty k souboru. Ze vstupní proměnné, kterou je celá cesta k souboru, je odstraněna cesta a funkcí je vrácen pouze název souboru. Funkce se využívá například ve funkci `addTraninigSample` k přidání názvu souboru do vnitřní proměnné `iSampleName` struktury `TraninigSample`.

## 3.2 Grafické rozhraní

Grafické rozhraní jsem vyřešil vytvořením desktopové aplikace, která využívá Windows API. Princip fungování aplikace je odchytávání událostí a generování vhodné odezvy. Událost je vyvolána použitím ovládacích prvků, například stiskem tlačítka. Uvnitř programu běží smyčka `while`, která události odchytává a přeposílá je funkci, která na základě `switch-case` struktury generuje odezvu na dané události. Každý

ovládací prvek aplikace má specifický identifikátor, díky kterému je vyvolána vhodná odezva.

### 3.2.1 Princip činnosti aplikace

Před tím, než je aplikaci možné ovládat, je třeba vykreslit okno aplikace a ovládací prvky. K vykreslení okna aplikace a vytvoření instance aplikace slouží funkce `InitInstance`. Před vykreslením okna aplikace je potřeba zaregistrovat třídu aplikace, což probíhá ve funkci `MyRegisterClass`. Uvnitř této funkce dochází k nastavení stylu okna. Je zde definováno například překreslení celého okna, když je aplikace posunuta v rámci plochy displeje. Dále se zde nastavuje obrázek ikony aplikace nebo vzhled kurzoru myši.

#### Tvorba okna aplikace

Po registraci třídy je již uvnitř funkce `InitInstance` vytvořeno okno aplikace pomocí funkce `CreateWindowW` [6], viz Výpis 3.19.

Výpis 3.19: Tvorba hlavního okna aplikace

```
HWND hWnd = CreateWindowW(L"CLASSIFIERGUI", L"ClassifierGUI",
    WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
    1150, 600, nullptr, nullptr, hInstance, nullptr);
```

Vstupními parametry, které se zadávají při tvorbě hlavního okna, jsou: název třídy, název okna, styl okna, pozice, velikost a instance. Konkrétní parametry, které byly zadány pro vytvoření hlavního okna aplikace, jsou na Výpis 3.19. Hlavní okno bude mít název *ClassifierGUI*. Styl okna je definován pomocí jediného parametru (`WS_OVERLAPPEDWINDOW`), který obsahuje více stylů: definování překrývajícího se okna s ohraničením, viditelnými prvky pro minimalizaci, maximalizaci, zavření okna a viditelný název okna. Okno je umístěno do základní pozice. Šířka okna má velikost 1150 pixelů, výška okna má velikost 600 pixelů. Velikost okna jsem volil v návaznosti na umístění a velikostech ovládacích prvků.

#### Tvorba ovládacích prvků

Po vytvoření hlavního okna aplikace dochází k vytvoření ovládacích prvků stejnou funkcí, jakou bylo vytvořeno hlavní okno. Mezi třídy prvků, které jsem při vytváření aplikace použil, patří:

- `WC_BUTTON` - tlačítka,
- `WC_LISTBOX` - seznamy přidaných vzorů,
- `WC_EDIT` - textové pole pro zadání třídy,

- `WC_STATIC` - textová pole pro zobrazení klasifikace a počítadel.

Na Výpis 3.20 je příklad použití funkce pro vytvoření tlačítka s názvem TRAIN. Opět jako v případě vytvoření okna jsou zadány parametry: třída, název, styl, pozice, velikost a instance. Na rozdíl od hlavního okna jsou zde parametry navíc. Těmito parametry jsou: hlavní okno aplikace a jedinečný identifikátor prvku, který je posílán ve zprávě v parametru `WPARAM`. Identifikátor daného prvku, který ve funkci `WinProc` (funkce se `switch-case` strukturou) slouží ke zvolení správného `case` a vyvolání správné odezvy, má například pro tlačítko TRAIN konkrétní hodnotu `IDC_BUTTON_TRAIN`. Pokud skončí vytváření ovládacího prvku chybou, je vyhozena výjimka a aplikace je ukončena.

Výpis 3.20: Tvorba ovládacího prvku - tlačítko TRAIN

```
if(CreateWindowW(WC_BUTTON, L"TRAIN", WS_TABSTOP |
   WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON,
   860, 30, 125, 50, hWnd, (HMENU) IDC_BUTTON_TRAIN,
   hInstance, NULL) == NULL)
throw std::exception();
```

V okně aplikace se vyskytují dva seznamy. Jedním je seznam vzorů v množině trénovacích dat, druhým je seznam vzorů v množině testovacích dat. Při práci se seznamem jsem potřeboval implementovat funkci pro vložení prvku, získání prvku a odstranění prvku.

K vložení prvku do seznamu jsem vytvořil funkci `insertItem`. Uvnitř funkce dochází k zavolání funkce `SendMessageW`. Funkci je předán identifikátor seznamu a operátor `LB_ADDSTRING`, který vloží proměnnou typu `string` na konec seznamu. Část kódu funkce sloužící pro vkládání prvku do množiny se nachází na Výpis 3.21.

Výpis 3.21: Vkládání prvku do seznamu

```
void insertItem(HWND iHWND, int iIDC, std::string iItem)
    {
    HWND editHWND = GetDlgItem(iHWND, iIDC);
    SendMessageW(editHWND, LB_ADDSTRING, 0,
                 (LPARAM)s2ws(iItem).c_str());
    }
```

Pro získání textové hodnoty položky seznamu jsem vytvořil funkci `getItem`. Funkce nejprve zjistí, který prvek je vybrán. Výběr probíhá pomocí kurzoru myši. Index prvku je vracen funkcí `SendMessageW` s parametrem `LB_GETCURSEL`. Poté je třeba získat délku textu položky a text položky přečíst. Jsou využity dva operátory ve funkci `SendMessageW`. Prvním je `LB_GETTEXTLEN`, který získá délku textu. Druhým je `LB_GETTEXT`, který již získá text položky. Prvky seznamu trénovacích

dat ukládám ve formátu „třída - název souboru“. Při práci s jednotlivým prvky potřebuji tyto dvě informace separovat. Rozhodl jsem se pro vytvoření dalších dvou funkcí: `itemFilename` vracející název souboru a `itemLabel` vracející název třídy.

K odstranění prvku ze seznamu slouží funkce `deleteItem`. Opět je využita funkce `SendMessageW`, nyní s operátorem `LB_DELETESTRING`.

### Řízení programu

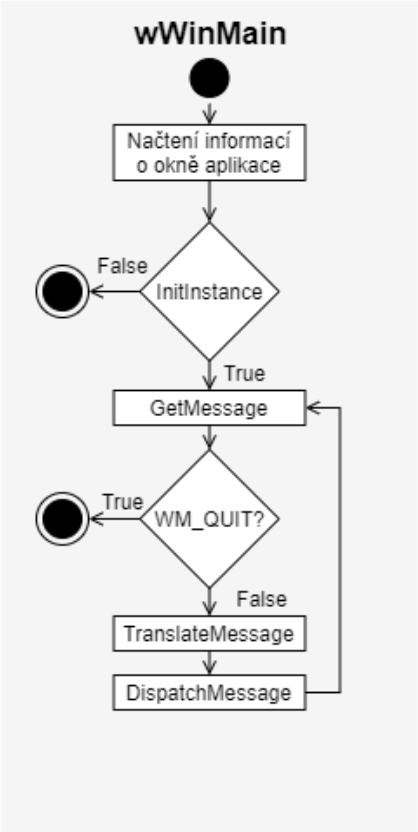
Činnost programu je řízena pomocí nekonečné smyčky `while`. Uvnitř smyčky se nacházejí funkce, které odchytávají a přeposílají události do další funkce, která v závislosti na události generuje odezvu. Část kódu obsahující smyčku `while` se nachází na Výpis 3.22.

Výpis 3.22: Smyčka while uvnitř funkce wWinMain

```
while(GetMessage(&msg, nullptr, 0, 0))
    {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    }
```

Uvnitř smyčky se nacházejí tři funkce, které slouží ke zpracování událostí/zpráv. První funkcí je `GetMessage`, která odchytává jednotlivé události [7]. Návratovou hodnotou funkce je záporné číslo při chybě, nula při obdržení `WM_QUIT` a kladná hodnota při obdržení jiné události. Po obdržení `WM_QUIT` není splněna podmínka ve funkci `while`, dojde k ukončení vykonávání smyčky a dochází tak k zavření celé aplikace. Pokud není obdržen požadavek na uzavření aplikace a funkce nehlásí žádnou chybu, návratová hodnota funkce `GetMessage` je kladná. Je tedy splněna podmínka pro vykonání smyčky `while` a zpráva je předána funkci `TranslateMessage` [8]. Jelikož aplikace využívá vstup z klávesnice, je třeba tento vstup korektně přeložit. Funkce zpracovává pouze zprávy týkající se klávesnicového vstupu a na ostatní události nemá vliv. Po překladu pomocí funkce `TranslateMessage` je zpráva předána funkci `DispatchMessage` [9], která zprávu odesílá funkci pro zpracování událostí. Touto funkcí je `WndProc`, jejímž hlavním účelem je pomocí `switch-case` struktury zprostředkovávat odezvu na odchycené události.

Na Obr. 3.4 je zobrazen vývojový diagram funkce `wWinMain` (hlavní funkce aplikace, v níž je umístěna smyčka `while`). Nejprve dochází k načtení informací o okně, mezi které patří název okna, velikost kurzoru, vzhled kurzoru, styl okna, ikona aplikace a další. Poté je proveden pokus o inicializaci okna. Při selhání dochází k zavření aplikace. Po úspěšné inicializaci se program dostává do výše zmíněné smyčky `while`.




Obr. 3.4: Vývojový diagram funkce wWinMain

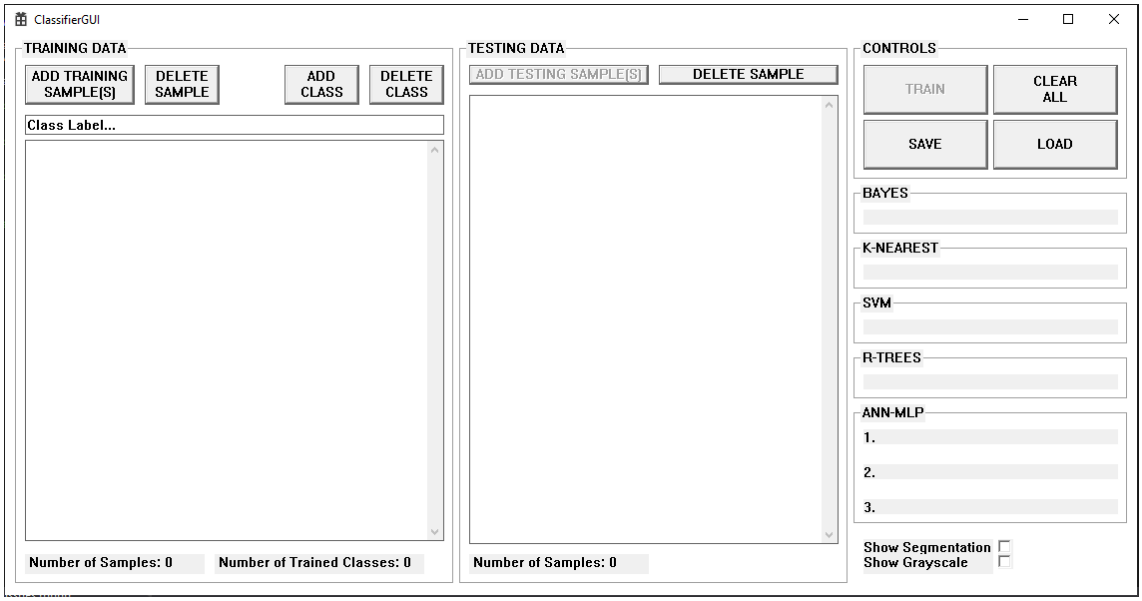
#### Funkce generující odezvu aplikace

Funkce `WndProc` v závislosti na přijaté zprávě vyvolává odezvu aplikace. Vstupními parametry funkce jsou: okno, ve kterém došlo k vyvolání události, událost typu `UINT`, parametry `WPARAM` a `LPARAM`. Ve zmíněných dvou parametrech se nachází například identifikace ovládacího prvku, který událost vyvolal. V aplikaci se odchytávají celkem tři typy zpráv. Prvním typem jsou zprávy s identifikátorem `WM_COMMAND`, který je odeslán při stisku tlačítka, při zapsání hodnoty do textového pole nebo při výběru položky ze seznamu. Druhým identifikátorem je `WM_DESTROY`, který je odchycen při kliknutí na vypínací tlačítko aplikace. Posledním identifikátorem je `WM_LBUTTONDOWN`, který je odeslán při stisku levého tlačítka myši. Tento typ zprávy využívám k zavření náhledu obrazu testovacího vzoru.

### 3.2.2 Ovládání aplikace

Vzhled aplikace je zobrazen na Obr. 3.5. Okno aplikace je rozděleno do menších bloků. Nejvíce vlevo se nachází blok TRAINING DATA, který slouží pro práci s trénovací množinou. Uprostřed okna aplikace se nachází blok TESTING DATA sloužící

pro práci s testovací množinou. V pravé části okna se nachází CONTROLS, ve kterém se nacházejí tlačítka pro načítání a ukládání souborů, trénování klasifikátorů a vymazání paměti aplikace. V poslední části okna dochází k zobrazení výsledků predikce tříd u jednotlivých klasifikátorů pro právě zvolený testovací vzor. Pod zobrazením výsledků se nacházejí dva zaškrtávací prvky, kterými je umožněno zobrazit obraz vady, popř. obraz segmentovaného objektu vady v rámci aplikace. Lze zobrazit pouze vady ze seznamu testovacích vad.




Obr. 3.5: Vzhled grafického rozhraní

### Blok TRAINING DATA

Uvnitř tohoto bloku se nacházejí ovládací prvky sloužící pro práci s trénovací množinou. Blok obsahuje čtyři tlačítka. Tlačítkem ADD TRAINING SAMPLE(S) lze přidat jeden či více trénovacích vzorů. Před přidáním vzoru je však nutné do kolonky s textem *„Class Label...“* zadat název třídy, která bude vkládaným vzorům přiřazena. Tlačítkem DELETE SAMPLE lze ze seznamu a množiny odstranit zvolený vzor. Dalším tlačítkem je ADD CLASS, které slouží k přidání celé třídy vzorů. Posledním tlačítkem v tomto bloku je DELETE CLASS, které vymaže z trénovací množiny třídu, jejíž název je nutné opět zadat do pole pod tlačítky. Součástí bloku je seznam přidaných vzorů, které jsou vypisovány ve formátu *„třída - název souboru“*. V seznamu se tak může objevit například položka *„90901 - 20171229-024920-983a-aX-11685-3526-0004-BP.jpg“*. Pod tímto seznamem se nacházejí dvě počítadla. Počítadlo vzorů `Number of Samples` a počítadlo již natrénovaných tříd `Number of Trained Classes`.

Pro přidání vzorů do trénovací množiny tedy slouží dvojice tlačítek. Jejich funkce je obdobná. Stiskem každého z tlačítek dochází k vyvolání dialogového okna, ve kterém uživatel volí vzory, jež mají být načteny do množiny. Před stiskem tlačítka je třeba do kolonky pod tlačítky napsat název třídy. Název třídy je nutné vepsat ve formátu *„xYYxx“*, lze tedy vepsat například *„90901“*. Především jde o dodržení druhého a třetího znaku, neboť na základě těchto znaků je kontrolováno, zda uživatel vkládá vzor do správné množiny. Je kontrolována tato hodnota s hodnotou uloženou v souboru (hodnota *resultID*). Při stisku tlačítka ADD TRAINING SAMPLE(S) je vyvoláno dialogové okno, v němž je možné volit soubory. Oproti tomu při stisku tlačítka ADD CLASS se vyvolá dialogové okno, ve kterém je možné vybírat pouze složky. Potvrzením výběru v dialogovém okně je získána cesta k vybraným souborům, která je předána funkci `addTrainingSample`, která segmentuje, popisuje obraz a vypočtené příznaky spolu s ostatními informacemi vkládá do struktury `TrainingSample`. Pokud dojde k úspěšnému vložení vzoru do množiny, je název třídy a název vzoru vložen jako další prvek do seznamu trénovacích vzorů a je adekvátně inkrementováno počítadlo indikující počet vzorů uvnitř množiny. Počítadlo indikující počet natrénovaných tříd je upravováno až po stisku tlačítka TRAIN v bloku CONTROLS. Je-li vkládán soubor nevhodného formátu nebo například nedošlo ke korektnímu načtení obrazu, nedochází k přidání vzoru do množiny, tudíž není ani přidán prvek do seznamu. Pokud uživatel zadal v názvu třídy špatný formát, vyskočí okénko informující o této situaci.

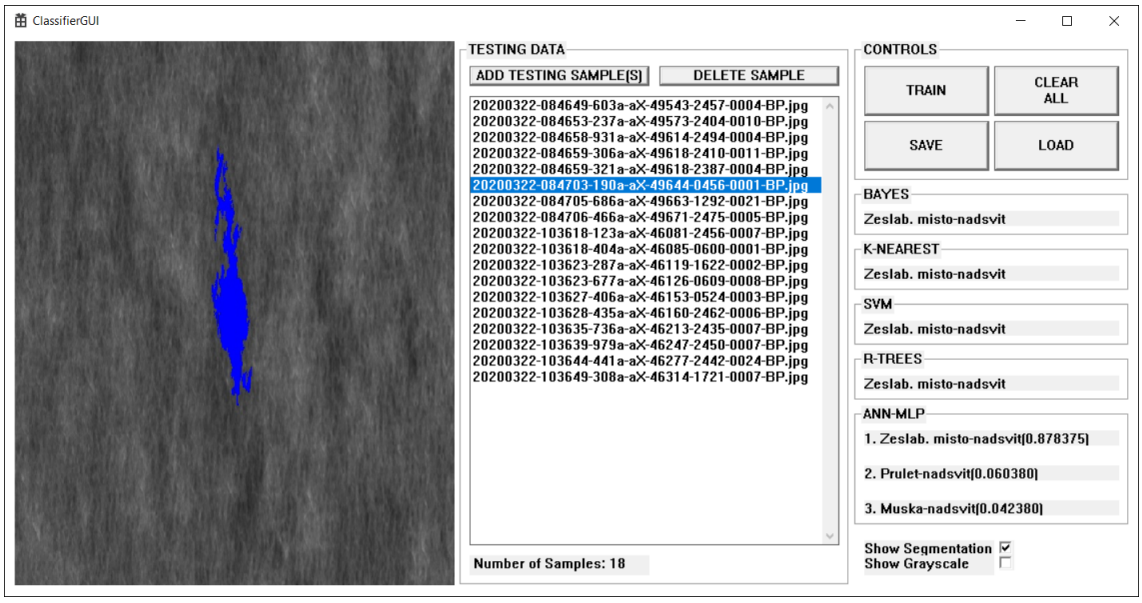
Stisku tlačítka DELETE SAMPLE by mělo předcházet zvolení prvku ze seznamu trénovacích vzorů pomocí kurzoru myši. Po stisku tlačítka dojde k vymazání prvku s tímto názvem z trénovací množiny a k vymazání prvku ze seznamu prvků. Zde využívám funkci `itemFilename` pro získání názvu souboru z prvku seznamu. Tento název je předán funkci `deleteSample`, která již vymaže vzor z trénovací množiny. K vymazání celé třídy slouží tlačítko DELETE CLASS. Před stiskem tohoto tlačítka je potřeba vepsat název třídy do textového pole. Po stisku tlačítka dojde k vymazání všech prvků s danou třídou z trénovací množiny. Dochází zároveň k odstranění všech prvků ze seznamu vzorů. Poté je opět adekvátně upraveno počítadlo vzorů trénovací množiny nacházející se pod seznamem.

### Blok TESTING DATA

Blok TESTING DATA je vzhledově podobný předešlému bloku. V tomto bloku se nacházejí dvě tlačítka. Tlačítko ADD TESTING SAMPLE(S) pro přidání jednoho či více testovacích vzorů a tlačítko DELETE SAMPLE sloužící pro odstranění zvoleného vzoru ze seznamu. Tlačítko pro přidání vzoru nelze ovládat do té doby, než je natrénována nejméně jedna ze čtyř sad klasifikátorů. Po stisku tlačítka dochází

k otevření dialogového okna, ve kterém lze zvolit jeden či více testovacích vzorů formátu *jpg*. Opět dojde k předání cesty funkci `addTestingSample`. Funkce uloží do proměnné typu `TestingSample` název souboru a vypočtené příznaky, avšak neuloží přiřazenou třídu. K přiřazení třídy dochází až ve funkci `testSample`, která je zavolána po funkci `addTestingSample`. Tlačítko DELETE SAMPLE slouží obdobně jako stejné tlačítko v okně TRAINING DATA k odstranění zvoleného vzoru ze seznamu a z testovací množiny. Opět se zde nachází seznam vzorů uvnitř testovací množiny. Položky v seznamu testovacích vzorů obsahují pouze název souboru, nezobrazuje se přiřazená třída ke vzoru, neboť je k jednomu souboru přiřazeno několik tříd. Pod seznamem se nachází počítadlo `Number of Samples`, které slouží k počítání vzorů uvnitř testovací množiny.

Pro testovací vzory jsem přidal možnost zobrazit obraz vady v rámci okna aplikace. Umožňuji zobrazit její šedotónovou variantu, ale také variantu se zvýrazněným segmentovaným objektem vady. Segmentovaný objekt vady je v obrazu zvýrazněn modrou barvou. Volba mezi oběma variantami probíhá v pravém dolním rohu aplikace. Pokud uživatel zvolí možnost zobrazení některé varianty, ukáže se po výběru vzoru z testovací množiny v levé části okna aplikace (místo bloku TRAINING DATA) obraz se zvolenou variantou. Tato situace je znázorněna na Obr. 3.6. Pro zavření náhledu obrazu vady je třeba kliknout do oblasti mimo seznam testovacích vzorů, čímž se opět zobrazí blok TRAINING DATA.




Obr. 3.6: Vzhled grafického rozhraní při zobrazení segmentace testovacího vzoru

Aby bylo možné zobrazit obraz, je třeba aplikaci předat cestu k tomu souboru. Toto je důvod vnitřních proměnných `iSamplePath` a `iSegmentedPath` uvnitř

struktury `TestingSample`. Obraz vzoru je brán z cesty, která byla zadána při vkládání vzoru. Obraz s vyznačeným objektem segmentované vady je ukládán do složky `segmentation` do adresáře, ze kterého byla spuštěna aplikace. Po zavření aplikace a po stisku tlačítka CLEAR ALL z bloku CONTROLS dochází k vymazání obsahu uvnitř této složky. K odstranění jednotlivých obrazů z této složky dochází také v případě, kdy je testovací vzor odstraněn z množiny tlačítkem DELETE SAMPLE.

### Blok CONTROLS

Prvním ovládacím prvkem uvnitř tohoto bloku je tlačítko TRAIN, po jehož stisknutí dojde k natrénování klasifikátorů. Tlačítko pro natrénování klasifikátoru však nelze ovládat do té doby, dokud se v nejméně jedné ze čtyř trénovacích množin nenachází alespoň dvě různé třídy trénovacích vzorů. Dochází k natrénování pouze té sady klasifikátorů, k níž byla do trénovací množiny vložena data. Po úspěšném natrénování vyskočí signalizační okénko indikující úspěšné natrénování klasifikátoru s nápisem *„Trained“*. Tlačítko LOAD slouží k načtení trénovací množiny a dat klasifikátoru ze souboru formátu *xml* do programu. Ke zvolení souboru dochází stejným způsobem jako v případě volby testovacích a trénovacích vzorů (pomocí dialogového okna). Samotné načtení dat pak již probíhá přes funkci `readXML`. Po stisku tohoto tlačítka je na rozdíl od vkládání vzorů do množin potřeba volit soubory s příponou *xml*. Při načtení dojde k vyplnění seznamu v okně TRAINING DATA. Dalším tlačítkem je tlačítko SAVE, které slouží k uložení trénovací množiny a klasifikátoru do souboru. Uložení těchto dat do souboru zajistí volání funkce `writeXML`. Posledním tlačítkem je tlačítko CLEAR ALL, po jehož stisknutí dojde k vyčištění proměnných. Dojde k vymazání trénovací a testovací množiny, vyčištění seznamu s trénovacími a testovacími vzory. Současně dojde k vymazání dat klasifikátoru a segmentovaných obrazů umístěných ve složce `segmentation`.

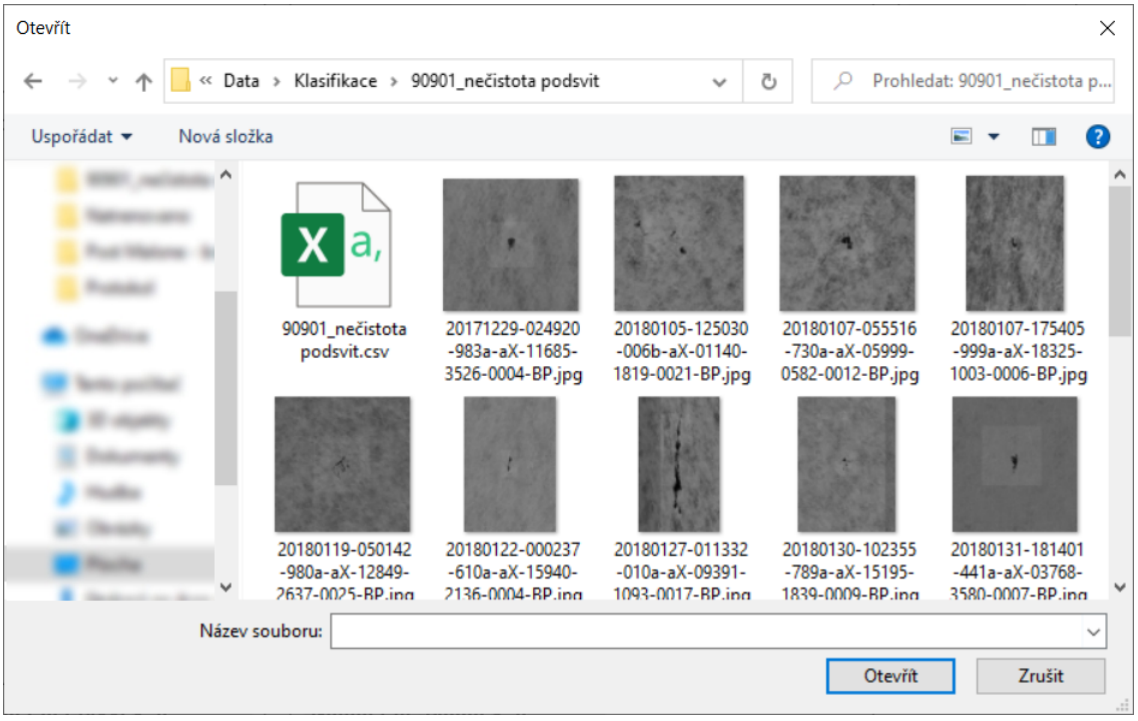
### Zobrazení výsledků predikce

V okně je zobrazen výsledek u toho vzoru, který je vybrán ze seznamu testovacích dat. Zobrazené predikce jsou vyčítány z vnitřní proměnné `iPredicitions` struktury `TestingSample`. V této proměnné jsou však uloženy ve formátu, který je zadáván do textového pole uvnitř bloku TRAINING DATA. Při zobrazení tak dojde k převodu na slovní popis třídy pomocí vyhledávací tabulky (funkce `LUTable`). Každý klasifikátor zařadí testovací vzor pouze do jedné třídy. Výjimku tvoří neuronová síť, která k jednomu testovacímu vzoru přiřadí tři nejvíce pravděpodobné třídy. Zobrazení výsledků neuronové sítě se tak liší od zobrazení výsledků ostatních klasifikátorů. U neuronové sítě je zobrazen název třídy spolu s příslušnou váhou, která je uvedena v závorce za názvem třídy.

### 3.2.3 Dialogové okno pro výběr souborů

V rámci aplikace potřebuji do proměnných programu načíst data ze souborů. Před načtením dat je nutné získat cestu k danému souboru. K zisku cesty využívám dialogové okno, se kterým pracuji pomocí `IFileOpenDialog` rozhraní. Nejprve jsem využíval `OPENFILENAMEA` strukturu, ale ukázalo se, že při pokusu o načtení více souborů naráz již tato struktura nevyhovuje požadavkům, neboť lze nahrát omezený počet souborů, který je definován délkou cesty k souborům. Proto jsem zvolil rozhraní `IFileOpenDialog`.

Celkově pracuji se dvěma typy souborů. Obrazy jsou uloženy v souborech typu *jpg*, data aplikace ukládám do souborů typu *xml*. Dále potřebuji umožnit situaci, kdy je vybrána složka a do programu jsou předány cesty ke všem souborům uvnitř této složky. Jelikož by fungoval stejný princip pro zisk cesty k *jpg*, *xml* souboru a ke složce, rozhodl jsem pro zisk cesty ke všem třem variantám použít stejnou funkci, která se pro dané případy liší pouze zadáním vstupních parametrů. Funkci jsem pojmenoval `OpenDialog`. Při vytváření funkce jsem využil již zmíněné rozhraní `IFileOpenDialog`. Nejprve je třeba vytvořit instanci dialogu pomocí funkce `CoCreateInstance`. Následně je již možné dialog otevřít. Otevřený dialog lze vidět na Obr. 3.7.




Obr. 3.7: Dialog pro výběr cesty k souboru

Před otevřením dialogového okna nastavuji výchozí příponu souboru. Výchozí

přípona bude přidána k souboru, u kterého uživatel příponu nezadal sám. Tohoto mechanismu je využito pouze pokud není funkce použita pro získání cesty ke složce. V takovém případě se přípona nenastavuje. Přípona je nastavena v závislosti na vstupním parametru funkce. K samotnému otevření okna slouží funkce `Show`. K volbě mezi možností výběru jednoho souboru a výběru více souborů dochází nastavením parametru `FOS_ALLOWMULTISELECT`. Parametr je nastavován spolu s dalšími parametry funkcí `SetOptions`. Funkce vrací proměnnou obsahující cesty k souborům, se kterými se pracuje dále v programu. Načítání samotných dat probíhá již v jiných funkcích (`readXML`, `writeXML` a `imread`).

# 4 Volba parametrů pro klasifikaci

Cílem této kapitoly je s pomocí již hotového rozhraní klasifikátoru (knihovny) a grafického rozhraní zvolit vhodné parametry k dosažení co možná nejvyšší spolehlivosti při klasifikaci vad. V současné době je na lince použit rychlý klasifikátor, který nedosahuje ve všech případech vysokých přesností, neboť jsou příznaky vypočítávány v rámci detekce vad a je zde kladen důraz na rychlost. Na jednom procesoru i5-8600K se zpracovává datový tok ze dvou kamer. Každá z kamer produkuje datový tok 1600 MB/s. Rozlišení každé z kamer je 8 kPix, snímková frekvence dosahuje hodnot 200 FPS. Procesor tak musí zpracovávat 3200 MB/s. Již jsem zmínil, že se v souboru s obrazem vady nacházejí i další uložená data. Mezi tato data patří právě hodnoty příznaků objektu dané vady vypočtené při detekci vady. Tyto příznaky však nelze použít a je třeba objekt vady znovu segmentovat a opětovně provést výpočet příznaků. Seznam příznaků, které vypočítávám, jsem již uvedl v tabulce 3.3.

K zaručení vysoké spolehlivosti při klasifikaci vad je třeba zvolit pro každou skupinu vad vhodnou sestavu příznaků do příznakového vektoru a vhodnou konfiguraci klasifikátorů. Nejprve jsem zkusil využít kombinaci příznaků, kterou jsem získal analýzou vlastností vad v kapitole 2. Zároveň jsem nejprve zkoušel klasifikaci s výchozí konfigurací klasifikátorů. Klasifikátory jsem pouze vytvořil a dále jsem nijak neměnil jejich parametry. Jelikož se ukázalo, že klasifikace nedosahuje vysokých spolehlivostí, rozhodl jsem se u jednotlivých skupin rozšířit příznakové vektory, k čemuž jsem využil funkci `fMaxSearch`, která pro množinu dat vypíše do souboru spolehlivosti všech možných kombinací příznaků. Složení příznakových vektorů jsou pro jednotlivé typy klasifikátorů odlišná. Po zvolení vhodných kombinací příznaků pro každou skupinu vad ještě vybírám vhodnou konfiguraci jednotlivých typů klasifikátorů. V následujících podkapitolách je pro každou skupinu vad volba příznaků a konfigurace uvedena zvlášť.

Z dodaného datasetu vytvořím trénovací a testovací množinu s rozdělením 50 % dat v trénovací množině a 50 % dat v testovací množině. Při vytváření rozhraní jsem používal jako klasifikátor pouze neuronovou síť. Poté jsem se rozhodl pro rozšíření rozhraní o další typy klasifikátorů. Nakonec jsem pracoval s těmito pěti typy klasifikátorů:

- Bayesův klasifikátor,
- klasifikace na základě nejbližšího souseda,
- metoda pomocných vektorů,
- náhodné rozhodovací stromy,
- umělá vícevrstvá neuronová síť.

## 4.1 Tmavé vady - podsvit

Nejprve jsem zjišťoval spolehlivost při složení příznakového vektoru podle kapitoly 2.1. Pro skupinu tmavých vad detekovaných při podsvícení se jednalo o následující sadu příznaků:

- konvexnost (Solidity),
- rozsah (Extent),
- poměr stran (Aspect ratio),
- minimální jasová úroveň uvnitř vady (QMIN),
- maximální jasová úroveň uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

S takto sestaveným příznakovým vektorem dosahovaly klasifikátory s výchozí konfigurací spolehlivostí zobrazených v prvním řádku tabulky 4.1. První řádek v tabulce (Původní spolehlivost) odpovídá spolehlivosti, kdy byly zvoleny příznaky na základě analýzy vlastností a byly zvoleny výchozí konfigurace klasifikátorů. Druhý řádek tabulky (Finální spolehlivost) odpovídá spolehlivosti při zvolení upravených příznakových vektorů a zvolení konfigurace klasifikátorů s nejvyšší spolehlivostí.

Tab. 4.1: Spolehlivosti klasifikátorů pro tmavé vady - podsvit

| 9 | Bayesův klasifikátor | Nejbližší soused | Metoda podpůrných vektorů | Náhodné stromy | Neuronová síť |
|---|---|---|---|---|---|
| Původní spolehlivost [%] | 81,57 | 88,09 | 64,87 | 89,74 | 86,14 |
| Finální spolehlivost [%] | 86,97 | 88,24 | 83,52 | 90,79 | 90,56 |

### Bayesův klasifikátor

U Bayesova klasifikátoru nelze nastavovat žádnou další konfiguraci [11]. U tohoto typu klasifikátoru se ukázalo, že ke zvyšování spolehlivosti dochází zejména přidáváním dalších příznaků. Do příznakového vektoru jsem přidal maximální šířku a výšku objektu. Ke zvýšení spolehlivosti pomohlo také odebrání 75. kvantilu jasové úrovně uvnitř objektu vady. Výsledné složení příznakového vektoru, se kterým dosahoval klasifikátor nejvyšší spolehlivosti, konkrétně 86,97 %, je:

- konvexnost (Solidity),
- úhel (Angle),
- rozsah (Extent),
- maximální šířka (Max width),
- maximální výška (Max height),
- poměr stran (Aspect ratio),
- obsah (Area),
- minimální jasová úroveň uvnitř vady (QMIN),
- maximální jasová úroveň uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- homogenita (Homogenity).

### Metoda nejbližšího souseda

Při metodě nejbližšího souseda umožňuje knihovna OpenCV volbu typu algoritmu a výchozí hodnotu parametru K, který má být použit k predikci [12]. U tohoto klasifikátoru lze nastavit typ algoritmu na hrubou sílu (Brute force) nebo využití struktury k-d stromů. Ke zvýšení spolehlivosti pomohlo odebrání minimální jasové úrovně z příznakového vektoru. Výsledný vektor obsahuje tyto příznaky:

- konvexnost (Solidity),
- rozsah (Extent),
- poměr stran (Aspect ratio),
- maximální jasová úroveň (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

Po zvolení příznakového vektoru jsem volil vhodnou konfiguraci parametrů. Jako nejvhodnější nastavení parametrů se jeví volba typu algoritmu využívajícího k-d struktury s výchozím (nulovým) nastavením parametru K. Při této konfiguraci dosahuje úspěšnost 88,24 %.

### Metoda podpůrných vektorů

U metody podpůrných vektorů nabízí knihovna OpenCV nastavení parametrů v závislosti na zvoleném typu klasifikace [13]. Ve výchozím nastavení je nastavena klasifikace pomocí podpůrných vektorů s parametrem C. Parametr C slouží jako

multiplikátor postihu odlehlých vzorů. Ve výchozím nastavení je však hodnota parametru nulová. Jako funkce jádra je ve výchozím nastavení zvolena radiální funkce (Radial basis function). Ke zvýšení spolehlivosti u této metody dochází přidáním hodnoty úhlu do příznakového vektoru a vynecháním většiny radiometrických příznaků. Jediný radiometrický příznak, který je ponechán, je homogenita. Konečná sestava příznakového vektoru je následující:

- konvexnost (Solidity),
- úhel (Angle),
- rozsah (Extent),
- poměr stran (Aspect ratio),
- homogenita (Homogenity).

Klasifikátor využívající metodu podpůrných vektorů dosahuje nejvyšší spolehlivosti volbou výchozího typu klasifikace (klasifikace s parametrem C) a nastavení jádra využívajícího průsečíku histogramů (Histogram intersection). Hodnota parametru C je nastavována v závislosti na trénovacích datech, neboť k trénování využívám funkci `trainAuto`, která tento parametr vnitřně upravuje. S touto konfigurací dosahuje klasifikátor spolehlivosti 83,52 %.

### Náhodné rozhodovací stromy

Knihovna umožňuje nastavení počtu proměnných (příznaků), které budou pro každý z náhodných stromů vybrány [14]. Při zachování výchozí hodnoty funkce volí do každého náhodného stromu počet příznaků úměrný odmocnině celkovému počtu všech použitých příznaků. Pro zvýšení spolehlivosti klasifikace jsem z příznakového vektoru odebral rozsah a naopak do vektoru přidal maximální šířku objektu vady a obsah objektu vady. V příznakovém vektoru se nacházejí tyto příznaky:

- konvexnost (Solidity),
- maximální šířka (Max width),
- poměr stran (Aspect ratio),
- obsah (Area),
- minimální jasová hodnota uvnitř vady (QMIN),
- maximální jasová hodnota uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

Klasifikátor dosahuje nejvyšších přesností, když je zvoleno porovnávání se čtyřmi nejbližšími vzory. Při této konfiguraci dosahuje spolehlivosti 90,79 %.

### Neuronová síť

Mírně odlišným způsobem jsem získával vhodnou konfiguraci u klasifikátoru typu neuronová síť. U tohoto typu klasifikátoru lze nastavit větší množství parametrů [15]. Knihovna OpenCV nenabízí obdobnou možnost jako u SVM klasifikátoru, tedy že si sama při trénování zvolí vhodné parametry pomocí funkce `trainAuto`. Musel jsem vyzkoušet větší množství možných kombinací, než jsem získal kombinaci s nejvyšší spolehlivostí, kterou používám napříč všemi skupinami vad. Jedná se o pětivrstvou síť, kde se v každé vrstvě nachází 15 buněk. Výjimku tvoří první vrstva, kde počet buněk odpovídá počtu příznaků, které vstupují do klasifikace, a poslední vrstva, ve které počet buněk odpovídá počtu tříd, do kterých klasifikátor vady klasifikuje. Jako aktivační funkci jsem zvolil symetrický sigmoid. Jako trénovací metodu jsem zvolil algoritmus zpětného šíření chyby (Back-propagation). Příznakový vektor jsem rozšířil o hodnoty maximální šířky a maximální výšky objektu vady. Ke zvýšení spolehlivosti také přispělo odebrání 25. kvantilu jasových úrovní objektu vady. Výsledný příznakový vektor obsahuje tyto příznaky:

- konvexnost (Solidity),
- rozsah (Extent),
- maximální šířka (Max width),
- maximální výška (Max height),
- poměr stran (Aspect ratio),
- minimální jasová úroveň uvnitř vady (QMIN),
- maximální jasová úroveň uvnitř vady (QMAX),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

S výše zmíněnou konfigurací a příznakovým vektorem dosahuji u neuronové sítě u tmavých vad pořízených při podsvícení spolehlivost 90,56 %.

## 4.2 Světlé vady - podsvit

Jako v předešlé podkapitole, opět nejprve získám úspěšnost klasifikace při složení vektorového příznaku podle kapitoly 2.2. Pro skupinu světlých vad detekovaných při podsvícení se jedná o následující příznaky:

- rozsah (Extent),
- maximální šířka (Max width),
- maximální výška (Max height),
- maximální jasová úroveň uvnitř vady (QMAX),

- 50. kvantil jasových hodnot mimo vadu (Q50_out).

S takto sestaveným příznakovým vektorem dosahovaly klasifikátory s výchozí konfigurací spolehlivostí zobrazených v prvním řádku tabulky 4.2. První řádek v tabulce (Původní spolehlivost) odpovídá spolehlivosti při použití příznaků z kapitoly 2.2 a výchozích konfiguracích klasifikátorů. Druhý řádek tabulky (Finální spolehlivost) odpovídá spolehlivosti při zvolení upravených příznakových vektorů a zvolení konfigurace klasifikátorů s nejvyšší spolehlivostí.

Tab. 4.2: Spolehlivosti klasifikátorů pro světlé vady - podsvit

| 10 | Bayesův klasifikátor | Nejbližší soused | Metoda podpůrných vektorů | Náhodné stromy | Neuronová síť |
|---|---|---|---|---|---|
| Původní spolehlivost [%] | 66,67 | 57,58 | 42,42 | 69,70 | 65,15 |
| Finální spolehlivost [%] | 77,27 | 74,24 | 68,18 | 72,73 | 77,27 |

### Bayesův klasifikátor

Ke zvýšení úspěšnosti Bayesova klasifikátoru jsem do příznakového vektoru přidal parametr obsahu. Složení příznakového vektoru pro Bayesův klasifikátor je:

- rozsah (Extent),
- maximální šířka (Max width),
- maximální výška (Max height),
- obsah (Area),
- maximální jasová úroveň uvnitř vady (QMAX),
- 50. kvantil jasových hodnot mimo vadu (Q50_out).

Spolehlivost tohoto typu klasifikátoru se tak zvedla na 77,27 %.

### Metoda nejbližšího souseda

Pro zvýšení spolehlivosti klasifikátoru využívajícího metodu nejbližšího souseda jsem z vektoru odebral parametr maximální šířky a maximální výšky. Naopak jsem do příznakového vektoru pro zvýšení spolehlivosti vložil parametr konvexnosti. Příznakový vektor u tohoto typu klasifikátoru obsahuje následující příznaky:

- konvexnost (Solidity),
- rozsah (Extent),
- maximální jasová úroveň uvnitř vady (QMAX),
- 50. kvantil jasových hodnot mimo vadu (Q50_out).

Algoritmus klasifikace je typu hrubá síla (Brute force), parametr K je nastaven na hodnotu 2. S touto konfigurací dosahuje klasifikace spolehlivosti 74,24 %.

### Metoda podpůrných vektorů

Klasifikátor využívající ke klasifikaci metodu podpůrných vektorů dosahuje nejvyšší spolehlivosti při příznakovém vektoru složeném pouze z těchto tří příznaků:

- konvexnost (Solidity),
- rozsah (Extent),
- 50. kvantil jasových hodnot mimo vadu (Q50_out).

Opět jsem tedy do příznakového vektoru vložil parametr konvexnosti a odebral maximální šířku a maximální výšku. Ke zvýšení spolehlivosti vedlo také odebrání parametru maximální jasové úrovně. Klasifikátor využívající metodu podpůrných vektorů dosahuje opět nejvyšší spolehlivosti volbou výchozího typu klasifikace (klasifikace s parametrem C) a nastavení jádra využívajícího průsečíku histogramů. Hodnota parametru C je ponechána na výchozí nulové hodnotě. Tato konfigurace dosahuje spolehlivosti 68,18 %.

### Náhodné rozhodovací stromy

Pro zvýšení spolehlivosti u náhodných rozhodovací stromů jsem rozšířil příznakový vektor o parametr úhlu a průměrné jasové úrovně uvnitř objektu vad. Příznakový vektor se tak skládá z těchto příznaků:

- úhel (Angle),
- rozsah (Extent),
- maximální šířka (Max width),
- maximální výška (Max height),
- maximální jasová úroveň uvnitř vady (QMAX),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out).

Nejvyšší spolehlivosti dosahuje výchozí nastavení konfigurace, při vytváření náhodných stromů se počet příznaků pro každý strom volí jako odmocnina celkového počtu příznaků. S touto konfigurací dosahuje klasifikátor spolehlivosti 72,73 %.

### Neuronová síť

Konfigurace neuronové sítě je shodná s konfigurací uvedenou v předchozí podkapitole. Příznakový vektor jsem pro neuronovou síť rozšířil o několik příznaků a její výsledné složení obsahuje tyto příznaky:

- konvexnost (Solidity),
- úhel (Angle),

- rozsah (Extent),
- maximální šířka (Max width),
- maximální výška (Max height),
- poměr stran (Aspect ratio),
- obsah (Area),
- minimální jasová úroveň uvnitř vady (QMIN),
- maximální jasová úroveň uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

Pro světlé vady v podsvitu dosahuje neuronová síť s uvedeným složením příznakového vektoru spolehlivosti 77,27 %. Konfigurace je stejná jako u předešlé skupiny vad. Jedná se o pětivrstvou neuronovou síť s 15 buňkami v prostředních třech vrstvách. Krajní dvě vrstvy mají počet buněk odpovídající počtu použitých příznaků a počtu klasifikačních tříd. Aktivační funkcí je opět symetrický sigmoid a trénovací metodou je algoritmus zpětného šíření chyby.

## 4.3 Tmavé vady - nadsvit

Jako v předchozích podkapitolách opět nejprve získám úspěšnost klasifikace při složení příznakového vektoru podle kapitoly 2.3. Pro skupinu tmavých vad detekovaných při nadsvícení se jedná o následující příznaky:

- konvexnost (Solidity),
- rozsah (Extent),
- poměr stran (Aspect ratio),
- obsah (Area),
- maximální jasová úroveň uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

S takto sestaveným příznakovým vektorem dosahovaly klasifikátory s výchozí konfigurací spolehlivostí zobrazených v prvním řádku tabulky 4.3. Druhý řádek odpovídá spolehlivostem, kterých dosahovaly klasifikátory po úpravách příznakového vektoru a pro výběru vhodné konfigurace.

Tab. 4.3: Spolehlivosti klasifikátorů pro tmavé vady - nadsvit

| 11 | Bayesův klasifikátor | Nejbližší soused | Metoda podpůrných vektorů | Náhodné stromy | Neuronová síť |
|---|---|---|---|---|---|
| Původní spolehlivost [%] | 94,65 | 89,01 | 79,72 | 89,86 | 92,11 |
| Finální spolehlivost [%] | 96,06 | 93,52 | 90,14 | 92,40 | 96,06 |

### Bayesův klasifikátor

U Bayesova klasifikátoru dochází ke zvýšení spolehlivosti klasifikace přidáním parametru maximální výšky. Kompletní sestava příznakového vektoru pro Bayesův klasifikátor obsahuje tyto příznaky:

- konvexnost (Solidity),
- maximální výška (Max height),
- poměr stran (Aspect ratio),
- obsah (Area),
- maximální jasová úroveň uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

S takto sestaveným příznakovým vektorem dosahuje Bayesův klasifikátor spolehlivosti 96,06 %.

### Metoda nejbližšího souseda

U metody nejbližšího souseda dochází k dosažení vyšší spolehlivosti odebráním několika příznaků z příznakového vektoru. Pro zvýšení spolehlivosti jsem odebral parametr rozsahu, obsahu, maximální jasové hodnoty a průměrnou jasovou hodnotu pixelů uvnitř objektu vady. V příznakovém vektoru se nacházejí tyto příznaky:

- konvexnost (Solidity),
- poměr stran (Aspect ratio),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

Po nastavení parametru K na hodnotu 5 a zvolení typu algoritmu využívajícího k-d struktury jsem u tohoto typu klasifikátoru dosáhl spolehlivosti 93,52 %.

### Metoda podpůrných vektorů

Při použití metody podpůrných vektorů pro klasifikaci tmavých vad detekovaných při nadsvícení dochází opět ke zvýšení spolehlivosti při odebrání několika příznaků. Konkrétně jsem odebral parametr poměru stran, obsahu, maximální a průměrnou jasovou úroveň pixelů uvnitř objektu vady a 25. kvantil jasových hodnot uvnitř vady. Příznakový vektor tak obsahuje tyto příznaky:

- konvexnost (Solidity),
- rozsah (Extent),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

Konfigurace klasifikátoru se shoduje se stejným typem klasifikátoru u předchozích dvou skupin vad. Zvolil jsem opět výchozí typ klasifikace (klasifikace s parametrem C) a nastavení jádra využívajícího průsečíku histogramů. Hodnota parametru C je nastavována v závislosti na trénovacích datech. S touto konfigurací dosahuji spolehlivosti 90,14 %.

### Náhodné rozhodovací stromy

U metody náhodných rozhodovacích stromů je sestavení příznakového vektoru následující:

- konvexnost (Solidity),
- rozsah (Extent),
- maximální šířka (Max width),
- obsah (Area),
- minimální jasová úroveň uvnitř vady (QMIN),
- maximální jasová úroveň uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out).

Konfiguraci klasifikátoru nechávám ve výchozím nastavení. Velikost náhodně vybrané podmnožiny příznaků v každém uzlu stromu je tedy nastavena jako odmocnina hodnoty celkového počtu příznaků. S tímto nastavením dosahuji u náhodných stromů spolehlivosti 92,40 %.

### Neuronová síť

Sestavení příznakového vektoru pro neuronovou síť odpovídá původnímu složení vektoru z kapitoly 2.3. Jedinou výjimku tvoří parametr maximální šířky, po jehož přidání dochází ke zvýšení spolehlivosti. Sestavení příznakového vektoru pro neuronovou síť klasifikující tmavé vady pořízené při nadsvícení je následující:

- konvexnost (Solidity),
- rozsah (Extent),
- maximální šířka (Max width),
- poměr stran (Aspect ratio),
- obsah (Area),
- maximální jasová úroveň uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

Se stejnou konfigurací, která byla zmíněna v předchozích dvou kapitolách, a s tímto sestavením příznakového vektoru dosahuji spolehlivosti 96,06 %.

## 4.4 Světlé vady - nadsvit

Stejně jako v předešlých podkapitolách i zde nejprve zjistím úspěšnost klasifikace při složení příznakového vektoru podle kapitoly 2.4. Pro skupinu světlých vad detekovaných při nadsvícení se jedná o následující příznaky:

- konvexnost (Solidity),
- úhel (Angle),
- maximální šířka (Max width),
- maximální výška (Max height),
- maximální jasová úroveň uvnitř vady (QMAX),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- homogenita (Homogenity).

S takto sestaveným příznakovým vektorem dosahovaly klasifikátory s výchozí konfigurací spolehlivostí zobrazených v prvním řádku tabulky 4.4. Význam hodnot v tabulce je obdobný jako v předchozích případech. První řádek (Původní spolehlivost) odpovídá spolehlivosti s prvotním sestavením příznakového vektoru a ponechanou výchozí konfigurací klasifikátorů. Druhý řádek tabulky (Finální spolehlivost) odpovídá spolehlivosti při zvolení upravených příznakových vektorů a zvolení konfigurace klasifikátorů s nejvyšší úspěšností.

Tab. 4.4: Spolehlivosti klasifikátorů pro světlé vady - nadsvit

| 12 | Bayesův klasifikátor | Nejbližší soused | Metoda podpůrných vektorů | Náhodné stromy | Neuronová síť |
|---|---|---|---|---|---|
| Původní spolehlivost [%] | 62,79 | 59,32 | 37,15 | 64,37 | 63,08 |
| Finální spolehlivost [%] | 66,39 | 59,71 | 54,77 | 66,22 | 66,27 |

### Bayesův klasifikátor

Pro zvýšení spolehlivosti Bayesova klasifikátoru jsem do příznakového vektoru přidal některé příznaky. Jmenovitě jsem přidal rozsah, poměr stran a minimální jasovou hodnotu, 25., 50. a 75. kvantil jasových úrovní uvnitř objektu vady. Příznakový vektor obsahuje tyto parametry:

- konvexnost (Solidity),
- úhel (Angle),
- rozsah (Extent),
- maximální šířka (Max width),
- maximální výška (Max height),
- poměr stran (Aspect ratio),
- minimální jasová úroveň uvnitř vady (QMIN),
- maximální jasová úroveň uvnitř vady (QMAX),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

S takto zvoleným příznakovým vektorem dosahuje Bayesův klasifikátor spolehlivosti 66,39 %.

### Metoda nejbližšího souseda

Ke zvýšení spolehlivosti vedlo přidání několika radiometrických příznaků a parametru rozsahu. Výsledný příznakový vektor se tak skládá z těchto příznaků:

- konvexnost (Solidity),
- úhel (Angle),
- rozsah (Extent),
- maximální šířka (Max width),
- maximální výška (Max height),

- minimální jasová úroveň uvnitř vady (QMIN),
- maximální jasová úroveň uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- homogenita (Homogenity).

Nejvyšší spolehlivosti dosahuje klasifikátor při nastavení parametru K na hodnotu 10 a typu algoritmu na hrubou sílu (Brute force). Při této konfigurace dosahuje klasifikátor pro světlé vady detekované při nadsvícení spolehlivosti 59,71 %.

### Metoda podpůrných vektorů

Na rozdíl od ostatních klasifikátorů, klasifikátor využívající metodu podpůrných vektorů dosahuje vyšší spolehlivosti při odebrání příznaků z příznakového vektoru. Nejvyšší spolehlivosti dosahoval klasifikátor s příznakovým vektorem složeným z těchto příznaků:

- konvexnost (Solidity),
- rozsah (Extent),
- homogenita (Homogenity).

Konfigurace klasifikátoru je stejná jako u předchozích klasifikátorů tohoto typu. Opět jsem zvolil výchozí typ klasifikace, tedy klasifikaci s parametrem C. Volil jsem opět jádro využívající průsečíku histogramů. Hodnota parametru C je opět nastavována v závislosti na trénovacích datech. Klasifikátor dosahuje spolehlivosti 54,77 % při nastavení této konfigurace.

### Náhodné rozhodovací stromy

Zvýšení spolehlivost u tohoto typu klasifikátoru jsem dosáhl přidáním parametru obsahu, 25. kvantilu jasových hodnot uvnitř objektu vady a homogenity. Příznakový vektor je pro náhodné rozhodovací stromy složen z těchto příznaků:

- konvexnost (Solidity),
- úhel (Angle),
- maximální šířka (Max width),
- maximální výška (Max height),
- obsah (Area),
- minimální jasová úroveň uvnitř vady (QMIN),
- maximální jasová úroveň uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot mimo vadu (Q50_out).

Vyšší spolehlivosti jsem dosahoval se zvolením parametru, který určuje počet příznaků použitých v každém ze stromů nastaveným do výchozí hodnoty. Počet parametrů je tedy roven druhé odmocnině počtu vstupních příznaků. S tímto nastavením a výše zmíněným příznakovým vektorem jsem u náhodných rozhodovacích stromů dosahoval spolehlivosti 66,22 %.

### Neuronová síť

Neuronová síť použitá pro světlé vady pořízené při nadsvícení má vyšší spolehlivost po přidání poměru stran, obsahu a několika radiometrických příznaků do příznakového vektoru. Výsledný příznakový vektor je poté složen z následujících příznaků:

- konvexnost (Solidity),
- úhel (Angle),
- rozsah (Extent),
- maximální šířka (Max width),
- maximální výška (Max height),
- poměr stran (Aspect ratio),
- obsah (Area),
- minimální jasová úroveň uvnitř vady (QMIN),
- maximální jasová úroveň uvnitř vady (QMAX),
- 25. kvantil jasových hodnot uvnitř vady (Q25),
- 50. kvantil jasových hodnot uvnitř vady (Q50),
- 50. kvantil jasových hodnot mimo vadu (Q50_out),
- 75. kvantil jasových hodnot uvnitř vady (Q75),
- homogenita (Homogenity).

Konfigurace neuronové sítě je obdobná jako konfigurace neuronových sítích použitých pro ostatní skupiny vad. Neuronová síť dosahuje pro skupinu světlých vad detekovaných za nadsvitu spolehlivosti 66,27 %.

# 5 Zhodnocení spolehlivosti klasifikace

V této kapitole provedu shrnutí spolehlivostí, kterých jsem při klasifikaci vad dosahoval. Spolehlivosti jednotlivých typů klasifikátorů byly uvedeny v předchozí kapitole. Pro tmavé vady v podsvitu jsou spolehlivosti jednotlivých klasifikátorů uvedeny v tabulce 4.1. Z tabulky je zřejmé, že nejvyšší spolehlivosti dosahují náhodné rozhodovací stromy, resp. neuronová síť, jejichž spolehlivost dosahuje hodnot 90,79 %, resp. 90,56 %. U světlých vad pořízených za podsvícení dosahuje opět jednu z nejvyšších spolehlivostí neuronová síť. Spolu s neuronovou sítí dosahuje vysokých spolehlivostí také Bayesův klasifikátor. Oba typy klasifikátoru dosahují spolehlivosti 77,27 %, což je možné vidět v tabulce 4.2. Nejvyšší spolehlivosti jsem dosahoval při klasifikaci tmavých vad v nadsvitu. V tabulce 4.3 lze vidět, že spolehlivosti u Bayesova klasifikátoru a neuronové sítě dosahovaly 96,06 %. Nejhůře, co se spolehlivosti týče, vychází skupina světlých vad pořízených v nadsvitu, u které dosahují nejvyšší spolehlivosti opět Bayesův klasifikátor a neuronová síť. Spolehlivost Bayesova klasifikátoru dosahuje hodnoty 66,39 %, spolehlivost neuronové sítě hodnoty 66,27 %. Spolehlivost klasifikátorů pro světlé vady pořízené za nadsvitu jsou zobrazeny v tabulce 4.4. V následujících podkapitolách shrnu spolehlivost klasifikátorů pro jednotlivé třídy vad.

## 5.1 Spolehlivost klasifikace do jednotlivých tříd

Uvedené spolehlivosti budou opět pro dataset rozdělený na poloviny, tedy 50 % vzorů v trénovací množině a 50 % vzorů v testovací množině. V následující sekci je pro každou skupinu vad zobrazena tabulka, která obsahuje spolehlivost při klasifikaci do jednotlivých tříd. Do tabulek jsem vložil i počet vzorů testovací množiny v daných třídách. Počty zde uvádím z toho důvodu, že jejich poměr odpovídá i poměrnému rozložení vzorů mezi třídami v trénovacích množinách. Obecně platí, že třídy s vyšším početním zastoupení vzorů dosahují vyšších spolehlivostí.

### 5.1.1 Tmavé vady - podsvit

K zobrazení spolehlivosti klasifikace vzorů z jednotlivých tříd slouží tabulka 5.1. Při pohledu na úspěšnost klasifikátorů u jednotlivých tříd se jako nejvhodnější typ klasifikátoru pro třídu 90901 (nečistota) jeví Bayesův klasifikátor. Ostatní typy klasifikátorů nedosahují u této třídy příliš vysokých spolehlivostí. Oproti tomu u třídy 90902 (průlet) dosahují prakticky všechny typy klasifikátorů vysokých úspěšností. Nízké spolehlivosti dosahují všechny typy klasifikátorů u třídy 90903 (úkap). Nejvyšších hodnot zde dosahuje klasifikátor klasifikující na základě nejbližšího souseda.

Nejvyšší spolehlivosti při klasifikaci vzorů z třídy 90904 (úkap s dírou) dosahuje neuronová síť. Opět nižších spolehlivostí, což je způsobeno malý početním zastoupením v trénovací množině, je dosahováno u třídy 90905 (úlet). Nejvyšší spolehlivosti u této třídy dosahuje Bayesův klasifikátor. Nejvíce početně zastoupenou třídou v trénovací množině je třída 90909 (zeslabené místo), u které je z tohoto důvodu dosahováno také nejvyšších spolehlivostí prakticky u všechny typů klasifikátorů. U poslední třídy 90911 (díra) dosahuje nejvyšší spolehlivosti Bayesův klasifikátor.

Tab. 5.1: Spolehlivosti klasifikátorů pro jednotlivé třídy pro skupinu 9

| | Třída 90901 [%] | Třída 90902 [%] | Třída 90903 [%] | Třída 90904 [%] | Třída 90905 [%] | Třída 90909 [%] | Třída 90911 [%] |
|---|---|---|---|---|---|---|---|
| BAYES | 81,82 | 79,21 | 33,33 | 66,67 | 64,58 | 94,92 | 47,92 |
| K-NEAREST | 4,55 | 90,76 | 40,00 | 57,58 | 16,67 | 98,38 | 35,42 |
| SVM | 0,00 | 90,10 | 0,00 | 6,06 | 2,08 | 95,50 | 25,00 |
| R-TREES | 13,64 | 97,03 | 33,33 | 60,61 | 33,33 | 99,31 | 29,17 |
| ANN-MLP | 0,00 | 97,36 | 0,00 | 81,82 | 56,25 | 99,08 | 4,17 |
| Počet vzorů | 22 | 303 | 15 | 33 | 48 | 866 | 48 |

### 5.1.2 Světlé vady - podsvit

Z tabulky 5.2 lze vidět, že všechny typy klasifikátoru dosahovaly u třídy 91003 (úkap) poměrně vysokých spolehlivostí. Nejvyšší spolehlivosti pak dosahovaly klasifikátory využívající metod nejbližšího souseda a pomocných vektorů. Nejvyšší spolehlivosti při klasifikaci do třídy 91006 (utržené vlákno) dosahoval Bayesův klasifikátor. Problém s klasifikací do této třídy spočívá v poměrně malém početním zastoupení vzorů v dodaném datasetu a podobnost vady s vadou typu úkap. Tento jev lze vyčíst z tabulek v kapitole A.2, kde je vidět, že vady 91006 (utržené vlákno) jsou chybně klasifikovány do třídy 91003 (úkap). Velice vysokých spolehlivostí dosahovaly všechny typy klasifikátory u třídy (91007) vlaštovka. Naopak malých úspěšností je dosahováno u vad ze tříd 91008 (zesílené místo) a 91010 (sklad), neboť vzory těchto vad nebyly v dodaném datasetu příliš zastoupeny. Při klasifikaci jsou tak prakticky vždy klasifikovány chybně. Na celkovou spolehlivost klasifikátorů to však nemá významný vliv, neboť se jedná o malé množství vzorů. Poslední třídou je třída 91015 (aviváž), u níž opět není dosahováno vysokých úspěšností. Nejvyšší úspěšnosti u této třídy dosahuje Bayesův klasifikátor, metoda nejbližšího souseda a náhodné rozhodovací stromy.

Tab. 5.2: Spolehlivosti klasifikátorů pro jednotlivé třídy pro skupinu 10

| | Třída 91003 [%] | Třída 91006 [%] | Třída 91007 [%] | Třída 91008 [%] | Třída 91010 [%] | Třída 91015 [%] |
|---|---|---|---|---|---|---|
| Bayes | 89,29 | 66,67 | 89,47 | 0,00 | 0,00 | 42,86 |
| K-Nearest | 96,43 | 11,11 | 94,74 | 0,00 | 0,00 | 42,86 |
| SVM | 96,43 | 0,00 | 94,74 | 0,00 | 0,00 | 0,00 |
| R-Trees | 89,29 | 22,22 | 94,74 | 0,00 | 0,00 | 42,86 |
| ANN-MLP | 92,86 | 33,33 | 94,74 | 0,00 | 100,00 | 28,57 |
| Počet vzorů [-] | 28 | 9 | 19 | 1 | 2 | 7 |

### 5.1.3 Tmavé vady - nadsvit

Z tabulky 5.3 lze vyčíst, že pro třídu 91101 (nečistota) se jako nejvhodnější typ klasifikátoru ukazuje Bayesův klasifikátor a neuronová síť, popřípadě klasifikátor využívající metodu nejbližšího souseda. Pro vady z třídy 91102 (průlet) dosahuje nejvyšší spolehlivosti opět Bayesův klasifikátor. Poměrně vysoké úspěšnosti dosahuje také neuronová síť. S největší spolehlivostí jsou ze skupiny tmavých vad vzniklých za nadsvícení klasifikovány vzory z třídy 91109 (zeslabené místo). Prakticky všechny klasifikátory dosahují vysokých spolehlivostí. Je to pravděpodobně způsobeno tím, že se v trénovací množině nacházely vzory z této třídy v nejvyšším počtu. Pro třídu 91112 (muška) se jako nejvhodnější klasifikátor jeví neuronová síť, neboť ostatní typy klasifikátorů zde dosahují nižších spolehlivostí, což může být způsobeno nedostatečným početním zastoupením vady v dodaném datasetu.

Tab. 5.3: Spolehlivosti klasifikátorů pro jednotlivé třídy pro skupinu 11

| | Třída 91101 [%] | Třída 91102 [%] | Třída 91109 [%] | Třída 91112 [%] |
|---|---|---|---|---|
| Bayes | 96,97 | 81,82 | 99,65 | 0,00 |
| K-Nearest | 87,88 | 54,55 | 100,00 | 33,33 |
| SVM | 66,67 | 39,39 | 99,65 | 50,00 |
| R-Trees | 75,76 | 60,61 | 98,94 | 50,00 |
| ANN-MLP | 90,91 | 72,73 | 100,00 | 66,67 |
| Počet vzorů [-] | 33 | 33 | 283 | 6 |

### 5.1.4 Světlé vady - nadsvit

Pro třídu 91203 (úkap) vychází nejvyšší spolehlivost u Bayesova klasifikátoru a náhodných rozhodovacích stromů. Pro vzory ze třídy 91204 (úkap s dírou) dosahuje nejvyšší spolehlivost Bayesův klasifikátor spolu s neuronovou sítí. Pro třídu 91205 (úlet) dosahují nejvyšší spolehlivosti náhodné rozhodovací stromy. Lze také použít Bayesův klasifikátor, ale ten dosahuje poněkud nižších hodnot spolehlivosti. U třídy 91206 (utržené vlákno) dosahují podobných hodnot spolehlivost náhodné rozhodovací stromy a neuronová síť. Pro vzory z třídy 91207 (vlaštovka) se jeví jako nejvhodnější použití klasifikátoru s metodu nejbližšího souseda. Dále by se pro tuto třídu dalo využít také neuronové sítě, popřípadě Bayesova klasifikátoru. Pro třídu 91208 (zesílené místo) lze s výhodou použít klasifikátor, který využívá metodu nejbližšího souseda nebo neuronovou síť. Vzory z třídy 91210 (sklad) nebyly v datasetu příliš zastoupené a také segmentace objektu vady z obrazu s tímto typem vady je obtížná. Z toho důvodu je klasifikace těchto vad nespolehlivá. Oproti tomu třídu 91213 (aviváž), jež měla v datasetu výrazné zastoupení, lze klasifikovat prakticky všemi typy klasifikátorů. Poslední třídu 91214 (dlouhý tenký úlet) lze nejlépe klasifikovat Bayesovým klasifikátorem.

Tab. 5.4: Spolehlivosti klasifikátorů pro jednotlivé třídy pro skupinu 12

| | Třída 91203 [%] | Třída 91204 [%] | Třída 91205 [%] | Třída 91206 [%] | Třída 91207 [%] | Třída 91208 [%] | Třída 91210 [%] | Třída 91213 [%] | Třída 91214 [%] |
|---|---|---|---|---|---|---|---|---|---|
| Bayes | 62,40 | 58,70 | 52,99 | 51,74 | 55,34 | 54,64 | 36,36 | 86,54 | 52,24 |
| K-Nearest | 40,00 | 36,96 | 22,39 | 43,91 | 63,11 | 62,20 | 0,00 | 83,51 | 4,48 |
| SVM | 40,00 | 0,00 | 33,58 | 54,78 | 29,13 | 41,58 | 0,00 | 86,84 | 0,00 |
| R-Trees | 61,60 | 30,43 | 59,70 | 61,30 | 54,85 | 52,92 | 0,00 | 90,47 | 4,48 |
| ANN -MLP | 52,80 | 47,83 | 46,27 | 62,61 | 58,25 | 57,39 | 4,55 | 87,14 | 34,33 |
| Počet vzorů[-] | 125 | 46 | 134 | 230 | 206 | 291 | 22 | 661 | 67 |

## 5.2 Různé rozdělení datasetu

Dále jsem se rozhodl pro srovnání spolehlivosti klasifikátorů pro různá rozdělení trénovací a testovací množiny. Vzory jsem mezi trénovací a testovací množinu rozdělil

třemi způsoby. Nejprve jsem do trénovací množiny vložit polovinu dat (Data_50 - slouží pro nastavení parametrů klasifikátorů), poté třetinu dat (Data_33) a nakonec dvě třetiny všech vzorů (Data_66). Výsledné úspěšnosti klasifikátorů jsou zobrazeny v tabulce 5.5.

Tab. 5.5: Srovnání spolehlivostí pro různé rozdělení datasetu

| | Bayesův klasifikátor | Nejbližší soused | Metoda podpůrných vektorů | Náhodné stromy | Neuronová síť |
|---|---|---|---|---|---|
| Data_33 Spolehlivost [%] | 75,61 | 72,54 | 69,76 | 75,08 | 71,61 |
| Data_50 Spolehlivost [%] | 77,33 | 74,14 | 69,42 | 78,24 | 78,63 |
| Data_66 Spolehlivost [%] | 75,57 | 74,26 | 70,66 | 78,83 | 76,97 |

Z tabulky je patrné, že další rozšiřování trénovací množiny se sebou nenese výrazné narůstání hodnot spolehlivosti. Naopak při snížení počtu vzorů v trénovací množině dochází k mírnému poklesu spolehlivosti s výjimkou klasifikátoru SVM (metoda podpůrných vektorů).

## 5.3 Srovnání s klasifikátorem použitým na lince

Na lince použitý klasifikátor dosahuje celkové spolehlivosti 77,0 %. Pro tmavé vady - podsvit (ID = 9) dosahuje 85% spolehlivosti, pro světlé vady - podsvit (ID = 10) dosahuje 67,5% spolehlivosti, pro tmavé vady - nadsvit (ID = 11) dosahuje spolehlivosti 88,8 % a pro světlé vady - nadsvit (ID = 12) dosahuje použitý klasifikátor spolehlivosti 66,3 %. Uvedené hodnoty spolehlivosti byly získány z datasetu, ze kterého bylo 70% vzorů vloženo do trénovací množiny a 30 % vzorů do testovací množiny.

Při rozložení dat, kdy se v trénovací a testovací množině nachází shodně 50 % vzorů, dosahují mnou použité klasifikátory: Bayesův klasifikátor, náhodné rozhodovací stromy a neuronová síť mírně vyšší celkové spolehlivosti. Jednotlivé hodnoty spolehlivostí lze vidět v tabulce 5.5.

Pro klasifikaci tmavých vad pořízených při podsvícení dosahují vyšší spolehlivosti dva typy mnou použitých klasifikátorů. Konkrétně se jedná o náhodné rozhodovací stromy a neuronovou síť, viz tabulka 4.1. Při klasifikaci světlých vad vzniklých za podsvitu dosahují všechny mnou použité typy klasifikátorů vyšší spolehlivost, viz

tabulka 4.2. Obdobně i pro klasifikaci tmavých vad pořízených při nadsvícení dosahují všechny mnou použité klasifikátory vyšší spolehlivosti, viz tabulka 4.3. Hůře se jeví použití mých klasifikátorů pro klasifikaci poslední skupiny vad, tedy světlých vad vzniklých při nadsvícení. Pro tuto skupinu nedošlo při použití žádného z mnou vytvořených klasifikátorů k výraznému zvýšení spolehlivosti. Klasifikátory typu: Bayesův klasifikátor, náhodné rozhodovací stromy a neuronová síť dosahovaly pro tuto skupinu vad srovnatelných spolehlivostí, což lze vidět srovnáním hodnoty spolehlivosti klasifikátoru použitém na lince s tabulkou 4.4.

# Závěr

Cílem diplomové práce bylo vytvoření vhodného rozhraní formou knihovny pro použití klasifikátoru, grafického rozhraní pro práci s klasifikátorem a vytvoření klasifikátoru, který vylepší dosavadní klasifikaci vad na lince kontinuální výroby.

Nejprve byla provedena literární rešerše používaných metod pro klasifikaci obrazu, do které byl zařazen klasifikátor s diskriminační funkcí, neuronové sítě, rozhodovací stromy a klasifikátor sloužící k rozpoznávání vzorů. V literární rešerši byly také zmíněny segmentační metody na základě prahování, segmentace z obrazu hran, segmentace založená na regionech a srovnávání se vzorem. Další částí literární rešerše bylo uvedení možných typů příznaků pro popis objektu vady.

K práci byly dodány vzory vad různých tříd. Vady byly rozděleny do čtyř hlavních skupin, které se dále dělí na jednotlivé třídy. V práci byly na příkladech vad analyzovány vlastnosti tříd. Avšak i vady jedné třídy vykazují poněkud odlišné vlastnosti a naopak některé vady různých třídy vykazují podobné vlastnosti, což má za následek chybné klasifikace vzorů a snížení celkové spolehlivosti klasifikace.

Hlavním úkolem diplomové práce bylo vytvoření knihovny pro použití klasifikátoru a vytvoření grafického rozhraní pro práci s klasifikátorem. Byla vytvořena knihovna umožňující přidávání a odebírání jednotlivých vzorů do trénovací a testovací množiny. Pro trénovací množinu byly implementovány funkce sloužící pro přidání a odebrání celé třídy vzorů. Dále bylo vytvořeno grafické rozhraní, které umožňuje práci s knihovnou. Aplikace umožňuje použití výše zmíněných funkcí, navíc nabízí ukládání dat do souboru a následné načtení zpět do programu.

Dále byla na dodaných vzorech vad natrénována sada klasifikátorů. U tří skupin vad se podařilo výrazně zvýšit spolehlivost klasifikace oproti používanému klasifikátoru, zejména pokud by se každý typ posuzoval klasifikátorem, který vykazuje nejlepší výsledky pro daný typ vad. Čtvrtou skupinu vad (světlé vady při nadsvitu) se již nepodařilo klasifikovat s vyšší spolehlivostí. Pro zvýšení úspěšnosti klasifikace bylo implementováno pět typů klasifikátorů, mezi které patří Bayesův klasifikátor, klasifikátor využívající metodu nejbližšího souseda, metodu pomocných vektorů, náhodné rozhodovací stromy a neuronovou síť. Metoda pomocných vektorů se ukázala jako nejméně vhodná, neboť již použitý klasifikátor předčila pouze v klasifikaci jedné ze čtyř skupin vad, a to tmavých vad vzniklých při nadsvícení. Jako nejvhodnější se jeví použití neuronové sítě, při jejímž použití dojde ke zvýšení celkové spolehlivosti klasifikace ze 77,0 % na 78,63 %.

Rychlost klasifikace obrazů vad je 1,58 vad/s. Tato rychlost byla dosažena na 1000 vzorcích ze všech 4 skupin vad. Poté byla změřena rychlost pro jednotlivé skupiny vad zvlášť. Skupina vad s ID 9 dosahovala rychlosti 1,2 vad/s, skupina vad s ID 10 rychlosti 2,4 vad/s, skupina s ID 11 rychlosti 1,9 vad/s a skupina

s ID 12 rychlosti 1,5 vad/s. Rychlost je velmi ovlivněna segmentací obrazů. Dalším rozšířením práce by bylo zrychlení segmentace a tím i celého klasifikačního procesu.

V současné době jsou výsledky práce (DLL knihovna) implementovány pro nasazení a otestování v pilotním provozu ve výrobním závodě PFNonwovens v Příměticích.

# Literatura


[1] SONKA, Michal, Vaclav HLAVAC a Roger BOYLE. *Image Processing, Analysis, and Machine Vision*. 4th edition. United States of America: Cengage Learning, 2015. ISBN 978-1-133-59369-0.

[2] JANÁKOVÁ, Ilona. *Segmentace: Úvod do segmentace - segmentační metody* [online]. VUT V BRNĚ. Brno, , 5 [cit. 2020-11-27]. Dostupné z URL: <http://vision.uamt.feec.vutbr.cz/POV/lectures/05_Segmentace.pdf>.

[3] FRIEDMAN, J. H., J. L. BENTLEY a R. A. FINKEL. *An algorithm for finding best matches in logarithmic expected time* [online]. , 211 [cit. 2020-12-03]. Dostupné z URL: <https://dl.acm.org/doi/pdf/10.1145/355744.355745>.

[4] ANN_MLP: ActivationFunctions. *OpenCV* [online]. [cit. 2021-03-05]. Dostupné z URL: <https://docs.opencv.org/master/d0/dce/classcv_1_1ml_1_1ANN__MLP.html#ade71470ec8814021728f8b31b09773b0>.

[5] BERNÁT, Gábor. How to build applications with OpenCV inside the "Microsoft Visual Studio."OpenCV [online]. [cit. 2020-12-09]. Dostupné z URL: <https://docs.opencv.org/master/dd/d6e/tutorial_windows_visual_studio_opencv.html>.

[6] Windows and Messages: CreateWindowW macro (winuser.h). *Microsoft Docs* [online]. 12/05/2018 [cit. 2020-12-09]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-createwindoww/>.

[7] Windows and Messages: GetMessage function (winuser.h). *Microsoft Docs* [online]. 12/05/2018 [cit. 2020-12-09]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getmessage>.

[8] Windows and Messages: TranslateMessage function (winuser.h). *Microsoft Docs* [online]. 12/05/2018 [cit. 2020-12-08]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/win32/inputdev/using-keyboard-input>.

[9] Windows and Messages: DispatchMessage function (winuser.h). *Microsoft Docs* [online]. 12/05/2018 [cit. 2020-12-09]. Dostupné z URL: <https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-dispatchmessage>.

[10] Machine Learning Overview. *OpenCV* [online]. Neuvedeno [cit. 2021-03-02]. Dostupné z URL: <https://docs.opencv.org/master/dc/dd6/ml_intro.html>.

[11] Cv::ml::NormalBayesClassifier Class Reference. *OpenCV* [online]. [cit. 2021-04-16]. Dostupné z URL: <https://docs.opencv.org/3.4/d4/d8e/classcv_1_1ml_1_1NormalBayesClassifier.html>.

[12] Cv::ml::KNearest Class Reference. *OpenCV* [online]. [cit. 2021-04-16]. Dostupné z URL: <https://docs.opencv.org/master/dd/de1/classcv_1_1ml_1_1KNearest.html>.

[13] Cv::ml::SVM Class Reference. *OpenCV* [online]. [cit. 2021-04-16]. Dostupné z URL: <https://docs.opencv.org/3.4/d1/d2d/classcv_1_1ml_1_1SVM.html>.

[14] Cv::ml::RTrees Class Reference. *OpenCV* [online]. [cit. 2021-04-16]. Dostupné z URL: <https://docs.opencv.org/master/d0/d65/classcv_1_1ml_1_1RTrees.html#a4f833672045ab3be645ca0cba0fc7a89>.

[15] Cv::ml::ANN_MLP Class Reference. *OpenCV* [online]. [cit. 2021-04-16]. Dostupné z URL: <https://docs.opencv.org/master/d0/dce/classcv_1_1ml_1_1ANN__MLP.html>.

# Seznam symbolů a zkratek

| | |
|---|---|
| **ANN-MLP** | umělá multivrstvá neuronová síť – Artifical Neural Network Multilayer Perceptron |
| **API** | rozhraní pro programování aplikací – Application Programming Interface |
| **CSV** | hodnoty oddělené čárkami – Comma-separated Values |
| **FPS** | počet snímků za sekundu – Frames per Second |
| **GUI** | grafické uživatelské rozhraní – Graphical User Interface |
| **int** | celočíselná proměnná – Integer |
| **JPG (JPEG)** | metoda pro uložení obrazu – Joint Photographic Experts Group |
| **K-D Tree** | K-rozměrný strom – K-Dimensional Tree |
| **kPix** | kilo pixel – kilo pixel |
| **LUTable** | vyhledávací tabulka – Look up Table |
| **MB/s** | megabyte/sekundu – Megabyte/Second |
| **Q25** | 25. kvantil – 25th Quantile |
| **Q50** | 50. kvantil – 50th Quantile |
| **Q75** | 75. kvantil – 75th Quantile |
| **QMAX** | maximální kvantil – Maximum Quantile |
| **QMIN** | minimální kvantil – Minimum Quantile |
| **ReLU** | usměrněná lineární jednotka – Rectified Linear Unit |
| **RPROP** | odolná zpětná propagace – Resilient Backpropagation |
| **SVM** | metoda podpůrných vektorů – Support Vector Machine |
| **vad/s** | počet vad za jednotku sekundy |
| **XML** | obecný značkovací jazyk – Extensible Markup Language |

# A Zhodnocení spolehlivosti

V této kapitole se nacházejí tabulky, jež obsahují údaje o spolehlivosti jednotlivých typů klasifikátorů rozdělené v příslušnosti k jednotlivým skupinám vad. Názvy sloupců tabulky odpovídají třídám, do kterých měl být vkládaný vzor přiřazen. Názvy řádků odpovídají třídám, do který daný klasifikátor vzor zařadil. V posledním řádku tabulky se tak nachází úspěšnost klasifikace vždy do dané třídy.

## A.1 Tmavé vady - podsvit

Tab. A.1: Spolehlivost Bayesova klasifikátoru pro skupinu 9

| | 90901 | 90902 | 90903 | 90904 | 90905 | 90909 | 90911 |
|---|---|---|---|---|---|---|---|
| 90901 | 18 | 38 | 0 | 0 | 11 | 1 | 3 |
| 90902 | 1 | 240 | 3 | 0 | 3 | 5 | 0 |
| 90903 | 0 | 1 | 5 | 0 | 0 | 0 | 0 |
| 90904 | 0 | 0 | 0 | 22 | 0 | 3 | 15 |
| 90905 | 2 | 19 | 5 | 2 | 31 | 0 | 0 |
| 90909 | 1 | 4 | 2 | 1 | 1 | 822 | 7 |
| 90911 | 0 | 1 | 0 | 8 | 2 | 35 | 23 |
| Spolehlivost [%] | 81,82 | 79,21 | 33,33 | 66,67 | 64,58 | 94,92 | 47,92 |

Tab. A.2: Spolehlivost nejbližšího souseda pro skupinu 9

| | 90901 | 90902 | 90903 | 90904 | 90905 | 90909 | 90911 |
|---|---|---|---|---|---|---|---|
| 90901 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 90902 | 17 | 275 | 5 | 2 | 23 | 0 | 4 |
| 90903 | 0 | 1 | 6 | 1 | 5 | 0 | 0 |
| 90904 | 0 | 0 | 0 | 19 | 0 | 5 | 12 |
| 90905 | 1 | 1 | 0 | 0 | 8 | 0 | 0 |
| 90909 | 3 | 25 | 4 | 4 | 10 | 852 | 15 |
| 90911 | 0 | 0 | 0 | 7 | 1 | 9 | 17 |
| Spolehlivost [%] | 4,55 | 90,76 | 40,00 | 57,58 | 16,67 | 98,38 | 35,42 |

Tab. A.3: Spolehlivost pro metodu pomocných vektorů pro skupinu 9

| | 90901 | 90902 | 90903 | 90904 | 90905 | 90909 | 90911 |
|---|---|---|---|---|---|---|---|
| 90901 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 90902 | 7 | 273 | 9 | 2 | 21 | 38 | 2 |
| 90903 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 90904 | 0 | 0 | 0 | 2 | 0 | 0 | 1 |
| 90905 | 0 | 6 | 0 | 0 | 1 | 1 | 0 |
| 90909 | 15 | 24 | 6 | 20 | 26 | 827 | 33 |
| 90911 | 0 | 0 | 0 | 9 | 0 | 0 | 12 |
| Spolehlivost [%] | 0,00 | 90,10 | 0,00 | 6,06 | 2,08 | 95,50 | 25,00 |

Tab. A.4: Spolehlivost náhodných rozhodovacích stromů pro skupinu 9

| | 90901 | 90902 | 90903 | 90904 | 90905 | 90909 | 90911 |
|---|---|---|---|---|---|---|---|
| 90901 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 90902 | 10 | 294 | 3 | 2 | 11 | 6 | 10 |
| 90903 | 0 | 0 | 5 | 0 | 1 | 0 | 0 |
| 90904 | 0 | 0 | 1 | 20 | 1 | 0 | 8 |
| 90905 | 0 | 0 | 2 | 0 | 16 | 0 | 0 |
| 90909 | 9 | 9 | 4 | 4 | 19 | 860 | 16 |
| 90911 | 0 | 0 | 0 | 7 | 0 | 0 | 14 |
| Spolehlivost [%] | 13,64 | 97,03 | 33,33 | 60,61 | 33,33 | 99,31 | 29,17 |

Tab. A.5: Spolehlivost neuronové sítě pro skupinu 9

| | 90901 | 90902 | 90903 | 90904 | 90905 | 90909 | 90911 |
|---|---|---|---|---|---|---|---|
| 90901 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 90902 | 9 | 295 | 11 | 1 | 14 | 4 | 2 |
| 90903 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 90904 | 0 | 0 | 0 | 27 | 1 | 0 | 23 |
| 90905 | 8 | 0 | 1 | 1 | 27 | 0 | 3 |
| 90909 | 5 | 8 | 3 | 3 | 6 | 858 | 18 |
| 90911 | 0 | 0 | 0 | 1 | 0 | 4 | 2 |
| Spolehlivost [%] | 0,00 | 97,36 | 0,00 | 81,82 | 56,25 | 99,08 | 4,17 |

## A.2 Světlé vady - podsvit

Tab. A.6: Spolehlivost Bayesova klasifikátoru pro skupinu 10

| | 91003 | 91006 | 91007 | 91008 | 91010 | 91015 |
|---|---|---|---|---|---|---|
| 91003 | 25 | 2 | 1 | 0 | 1 | 0 |
| 91006 | 2 | 6 | 1 | 0 | 0 | 2 |
| 91007 | 1 | 1 | 17 | 0 | 1 | 2 |
| 91008 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91010 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91015 | 0 | 0 | 0 | 1 | 0 | 3 |
| Spolehlivost [%] | 89,29 | 66,67 | 89,47 | 0,00 | 0,00 | 42,86 |

Tab. A.7: Spolehlivost nejbližšího souseda pro skupinu 10

| | 91003 | 91006 | 91007 | 91008 | 91010 | 91015 |
|---|---|---|---|---|---|---|
| 91003 | 27 | 6 | 1 | 0 | 1 | 2 |
| 91006 | 1 | 1 | 0 | 1 | 0 | 0 |
| 91007 | 0 | 2 | 18 | 0 | 1 | 2 |
| 91008 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91010 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91015 | 0 | 0 | 0 | 0 | 0 | 3 |
| Spolehlivost [%] | 96,43 | 11,11 | 94,74 | 0,00 | 0,00 | 42,86 |

Tab. A.8: Spolehlivost pro metodu pomocných vektorů pro skupinu 10

| | 91003 | 91006 | 91007 | 91008 | 91010 | 91015 |
|---|---|---|---|---|---|---|
| 91003 | 27 | 8 | 1 | 0 | 1 | 2 |
| 91006 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91007 | 1 | 1 | 18 | 1 | 1 | 5 |
| 91008 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91010 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91015 | 0 | 0 | 0 | 0 | 0 | 0 |
| Spolehlivost [%] | 96,43 | 0,00 | 94,74 | 0,00 | 0,00 | 0,00 |

Tab. A.9: Spolehlivost náhodných rozhodovacích stromů pro skupinu 10

| | 91003 | 91006 | 91007 | 91008 | 91010 | 91015 |
|---|---|---|---|---|---|---|
| 91003 | 25 | 6 | 1 | 0 | 0 | 2 |
| 91006 | 3 | 2 | 0 | 1 | 1 | 0 |
| 91007 | 0 | 1 | 18 | 0 | 1 | 2 |
| 91008 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91010 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91015 | 0 | 0 | 0 | 0 | 0 | 3 |
| Spolehlivost [%] | 89,29 | 22,22 | 94,74 | 0,00 | 0,00 | 42,86 |

Tab. A.10: Spolehlivost neuronové sítě pro skupinu 10

| | 91003 | 91006 | 91007 | 91008 | 91010 | 91015 |
|---|---|---|---|---|---|---|
| 91003 | 26 | 4 | 1 | 1 | 0 | 2 |
| 91006 | 1 | 3 | 0 | 0 | 0 | 0 |
| 91007 | 1 | 2 | 18 | 0 | 0 | 3 |
| 91008 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91010 | 0 | 0 | 0 | 0 | 2 | 0 |
| 91015 | 0 | 0 | 0 | 0 | 0 | 2 |
| Spolehlivost [%] | 92,86 | 33,33 | 94,74 | 0,00 | 100,00 | 28,57 |

## A.3 Tmavé vady - nadsvit

Tab. A.11: Spolehlivost Bayesova klasifikátoru pro skupinu 11

| | 91101 | 91102 | 91109 | 91112 |
|---|---|---|---|---|
| 91101 | 32 | 4 | 0 | 5 |
| 91102 | 1 | 27 | 1 | 1 |
| 91109 | 0 | 2 | 282 | 0 |
| 91112 | 0 | 0 | 0 | 0 |
| Spolehlivost [%] | 96,97 | 81,82 | 99,65 | 0,00 |

Tab. A.12: Spolehlivost nejbližšího souseda pro skupinu 11

| | 91101 | 91102 | 91109 | 91112 |
|---|---|---|---|---|
| 91101 | 29 | 5 | 0 | 4 |
| 91102 | 3 | 18 | 0 | 0 |
| 91109 | 0 | 10 | 283 | 0 |
| 91112 | 1 | 0 | 0 | 2 |
| Spolehlivost [%] | 87,88 | 54,55 | 100,00 | 33,33 |

Tab. A.13: Spolehlivost pro metodu pomocných vektorů pro skupinu 11

| | 91101 | 91102 | 91109 | 91112 |
|---|---|---|---|---|
| 91101 | 22 | 10 | 0 | 2 |
| 91102 | 9 | 13 | 1 | 1 |
| 91109 | 1 | 10 | 282 | 0 |
| 91112 | 1 | 0 | 0 | 3 |
| Spolehlivost [%] | 66,67 | 39,40 | 99,65 | 50,00 |

Tab. A.14: Spolehlivost náhodných rozhodovacích stromů pro skupinu 11

| | 91101 | 91102 | 91109 | 91112 |
|---|---|---|---|---|
| 91101 | 25 | 8 | 1 | 3 |
| 91102 | 8 | 20 | 2 | 0 |
| 91109 | 0 | 5 | 280 | 0 |
| 91112 | 0 | 0 | 0 | 3 |
| Spolehlivost [%] | 75,76 | 60,61 | 98,94 | 50,00 |

Tab. A.15: Spolehlivost neuronové sítě pro skupinu 11

| | 91101 | 91102 | 91109 | 91112 |
|---|---|---|---|---|
| 91101 | 30 | 4 | 0 | 2 |
| 91102 | 2 | 24 | 0 | 0 |
| 91109 | 1 | 5 | 283 | 0 |
| 91112 | 0 | 0 | 0 | 4 |
| Spolehlivost [%] | 90,91 | 72,73 | 100,00 | 66,67 |

## A.4 Světlé vady - nadsvit

Tab. A.16: Spolehlivost Bayesova klasifikátoru pro skupinu 12

| | 91203 | 91204 | 91205 | 91206 | 91207 | 91208 | 91210 | 91213 | 91214 |
|---|---|---|---|---|---|---|---|---|---|
| 91203 | 78 | 9 | 18 | 15 | 2 | 2 | 3 | 4 | 4 |
| 91204 | 11 | 27 | 10 | 3 | 6 | 3 | 2 | 10 | 1 |
| 91205 | 10 | 2 | 71 | 8 | 0 | 0 | 3 | 6 | 6 |
| 91206 | 13 | 2 | 9 | 119 | 6 | 7 | 2 | 17 | 13 |
| 91207 | 7 | 1 | 6 | 18 | 114 | 5 | 0 | 16 | 0 |
| 91208 | 0 | 2 | 3 | 7 | 36 | 159 | 3 | 35 | 1 |
| 91210 | 2 | 2 | 3 | 2 | 1 | 1 | 8 | 1 | 3 |
| 91213 | 2 | 1 | 0 | 49 | 41 | 114 | 0 | 572 | 4 |
| 91214 | 2 | 0 | 14 | 9 | 0 | 0 | 1 | 0 | 35 |
| Spolehlivost [%] | 62,40 | 58,70 | 52,99 | 51,74 | 55,34 | 54,64 | 36,36 | 86,54 | 52,24 |

Tab. A.17: Spolehlivost nejbližšího souseda pro skupinu 12

| | 91203 | 91204 | 91205 | 91206 | 91207 | 91208 | 91210 | 91213 | 91214 |
|---|---|---|---|---|---|---|---|---|---|
| 91203 | 50 | 14 | 27 | 18 | 3 | 2 | 10 | 2 | 6 |
| 91204 | 4 | 17 | 2 | 0 | 1 | 0 | 0 | 0 | 0 |
| 91205 | 23 | 1 | 30 | 12 | 4 | 4 | 2 | 5 | 11 |
| 91206 | 36 | 7 | 49 | 101 | 5 | 4 | 4 | 18 | 41 |
| 91207 | 4 | 1 | 9 | 3 | 130 | 4 | 0 | 12 | 0 |
| 91208 | 5 | 3 | 11 | 45 | 26 | 181 | 5 | 72 | 2 |
| 91210 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91213 | 3 | 2 | 5 | 47 | 37 | 96 | 0 | 552 | 4 |
| 91214 | 0 | 0 | 1 | 4 | 0 | 0 | 1 | 0 | 3 |
| Spolehlivost [%] | 40,00 | 36,96 | 22,39 | 43,91 | 63,11 | 62,20 | 0,00 | 83,51 | 4,48 |

Tab. A.18: Spolehlivost pro metodu pomocných vektorů pro skupinu 12

| | 91203 | 91204 | 91205 | 91206 | 91207 | 91208 | 91210 | 91213 | 91214 |
|---|---|---|---|---|---|---|---|---|---|
| 91203 | 50 | 17 | 14 | 10 | 1 | 0 | 2 | 1 | 4 |
| 91204 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91205 | 34 | 5 | 45 | 12 | 4 | 4 | 7 | 0 | 9 |
| 91206 | 29 | 20 | 12 | 126 | 34 | 15 | 5 | 32 | 21 |
| 91207 | 6 | 1 | 9 | 30 | 60 | 9 | 6 | 11 | 16 |
| 91208 | 4 | 2 | 53 | 6 | 19 | 121 | 1 | 43 | 8 |
| 91210 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91213 | 2 | 1 | 1 | 46 | 88 | 142 | 1 | 574 | 9 |
| 91214 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Spoleh-livost [%] | 40,00 | 0,00 | 33,58 | 54,78 | 29,13 | 41,58 | 0,00 | 86,84 | 0,00 |

Tab. A.19: Spolehlivost náhodných rozhodovacích stromů pro skupinu 12

| | 91203 | 91204 | 91205 | 91206 | 91207 | 91208 | 91210 | 91213 | 91214 |
|---|---|---|---|---|---|---|---|---|---|
| 91203 | 77 | 20 | 27 | 15 | 0 | 0 | 4 | 1 | 8 |
| 91204 | 0 | 14 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91205 | 17 | 5 | 80 | 17 | 4 | 6 | 7 | 1 | 14 |
| 91206 | 25 | 5 | 17 | 141 | 23 | 10 | 7 | 25 | 33 |
| 91207 | 2 | 1 | 2 | 1 | 113 | 6 | 0 | 3 | 3 |
| 91208 | 1 | 0 | 5 | 7 | 20 | 154 | 4 | 33 | 0 |
| 91210 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 91213 | 3 | 1 | 1 | 49 | 46 | 115 | 0 | 598 | 6 |
| 91214 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 3 |
| Spoleh-livost [%] | 61,60 | 30,43 | 59,70 | 61,30 | 54,85 | 52,92 | 0,00 | 90,47 | 4,48 |

Tab. A.20: Spolehlivost neuronové sítě pro skupinu 12

| | 91203 | 91204 | 91205 | 91206 | 91207 | 91208 | 91210 | 91213 | 91214 |
|---|---|---|---|---|---|---|---|---|---|
| 91203 | 66 | 10 | 31 | 10 | 0 | 0 | 2 | 1 | 6 |
| 91204 | 3 | 22 | 4 | 0 | 0 | 0 | 4 | 0 | 1 |
| 91205 | 15 | 1 | 62 | 3 | 1 | 0 | 4 | 2 | 5 |
| 91206 | 26 | 8 | 16 | 144 | 11 | 4 | 6 | 25 | 23 |
| 91207 | 8 | 1 | 3 | 8 | 120 | 9 | 0 | 12 | 0 |
| 91208 | 1 | 1 | 6 | 18 | 39 | 167 | 3 | 44 | 2 |
| 91210 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 91213 | 2 | 1 | 1 | 45 | 35 | 111 | 0 | 576 | 7 |
| 91214 | 4 | 2 | 10 | 2 | 0 | 0 | 2 | 1 | 23 |
| Spoleh-livost [%] | 52,80 | 47,83 | 46,27 | 62,61 | 58,25 | 57,39 | 4,55 | 87,14 | 34,33 |

# B Obsah přiloženého DVD

V elektronické verzi práce se nenachází celé projekty programového řešení práce, neboť je nahrávání elektronické práce omezeno velikostí (15 MB). Kompletní řešení práce je tak obsaženo na DVD přiloženém k tištěné verzi práci. V elektronické příloze se také nachází omezené množství vzorů v datasetu.

- `/` ..... kořenový adresář přiloženého archivu
  - `Textová část`
    - `Text práce.pdf`
    - `Textová část.zip`
  - `Programová část`
    - `ClassifierGUI` ..... zdrojové soubory grafického rozhraní
    - `Classifier` ..... zdrojové soubory rozhraní klasifikátoru
    - `opencv` ..... adresář s knihovnou OpenCV
  - `Dataset` ..... dataset vzorů vad