VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Zdeněk Ševčík

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

AUTOMATICKÁ KLASIFIKACE OBRAZŮ
AUTOMATIC IMAGE CLASSIFICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Zdeněk Ševčík**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Pavel Sikora**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Audio inženýrství**

Ústav telekomunikací

Student: Bc. Zdeněk Ševčík ***ID:*** 186635
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Automatická klasifikace obrazů

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je nastudovat a teoreticky popsat současné metody strojového učení bez učitele umožňující klasifikaci digitálních obrazů na základě obrazové podobnosti a metody pro extrakci příznaků z obrazových dat. Následně vybrat vhodnou metodu pro extrakci příznaků z obrazových dat a vhodné metody strojového učení bez učitele. Funkčnost metod poté otestovat ve vlastní aplikaci. Jazyk pro tvorbu této aplikace bude Python.

DOPORUČENÁ LITERATURA:

[1] GONZALEZ, R. C.; WOODS R. E.: Digital Image Processing, Prentice Hall, New Jersey, 2002,

[2] Krizhevsky, A., Sutskever, I., Hinton G. E., ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems 25, 2012. p. 1097-1105

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Pavel Sikora

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Cílem této práce je prozkoumat shlukovací algoritmy strojového učení bez učitele, které lze použít pro klasifikaci databáze obrazů podle podobnosti. Pro vybrané shlukovací algoritmy je sepsán teoretický základ. Pro zlepšení klasifikace použité databáze se diplomová práce zabývá různými metodami předzpracování obrazů. Těmito metodami jsou z obrazu extrahovány příznaky. Dále práce řeší implementaci metod předzpracování a praktickou aplikaci shlukovacích algoritmů. V praktické části je naprogramována aplikace v programovacím jazyce Python, která klasifikuje databázi obrazů do tříd podle podobnosti. Diplomová práce testuje všechny použité metody a ke konci práce je zpracována rešerše výsledků.

KLÍČOVÁ SLOVA

Aglomerativní shlukování, automatická klasifikace obrazů, BIRCH, BoVW, BRISK, extrakce příznaků, K-majority, K-means, konvoluční autoenkodér, OBR, strojové učení, učení bez učitele

ABSTRACT

The aim of this thesis is to explore clustering algorithms of machine unsupervised learning, which can be used for image database classification by similarity. For chosen clustering algorithms is written up a theoretical basis. For better classification of used database this thesis deals with different methods of image preprocessing. With these methods the features from image are extracted. Next the thesis solves of implementation of preprocessing methods and practical application of clustering algorithms. In practical part is programmed aplication in Python programming language, which classifies the database of images into classes by similarity. The thesis tests all of used methods and at the end of the thesis is processed searches of results.

KEYWORDS

Aglomerativ clustering, automatic image classification, BIRCH, BoVW, BRISK, convolution autoencoder, feature extraction, machine learning, K-majority, K-means, ORB, unsupervised learning

ŠEVČÍK, Zdeněk. *Automatická klasifikace obrazů*. Brno, 2020, 83 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Pavel Sikora

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Automatická klasifikace obrazů“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové 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

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Pavlu Sikorovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

1 Úvod

V dnešní době vzniká požadavek na roztřídění velkých databází obrazů, které již člověk není schopen sám třídit. Proto se využívá strojového učení, které je schopno tuto operaci udělat za nás. Strojové učení lze rozdělit do dvou hlavních skupin. První skupinou je učení s učitelem, které řeší problematiku klasifikace a regrese. Druhou skupinou je učení bez učitele, které se zabývá shlukováním a asociací.

Klasifikace databází se běžně provádí pomocí strojového učení s učitelem, kdy se využívá nejrůznějších metod, především neuronových sítí. Pro vysokou přesnost vyžaduje metoda učení s učitelem velké množství anotovaných dat. Tento proces je výpočetně náročný. Důvodem výběru metody učení bez učitele pro aplikaci na danou problematiku je skutečnost, že tato metoda je schopná pracovat bez anotovaných dat a dokáže se učit sama. Práce se zabývá shlukovacími algoritmy aplikovanými na problematiku klasifikace obrazů.

Pro vyšší úspěšnost klasifikace použité databáze řeší tato práce vhodné předzpracování obrazů před samotnou klasifikací pomocí shlukovacích algoritmů. Práce popisuje metody předzpracování obrazu, které jsou schopny z obrazu vyextrahovat příznaky. Obrazové příznaky definují obraz a nesou obrazovou informaci. Shlukovacím algoritmům jsou poté ke klasifikaci předány pixely nebo právě tyto příznaky.

Práce teoreticky popisuje strojové učení a jeho podskupiny. Dále se blíže zabývá shlukovou analýzou, kterou lze rozdělit na hierarchické a nehierarchické metody. Teoreticky popisuje algoritmy shlukové analýzy vhodné pro klasifikaci obrazů.

Praktická část práce se zabývá tvorbou aplikace v programovacím jazyce Python, která používá logiku strojového učení bez učitele sloužící ke klasifikaci obrazů. Práce vysvětluje použití implementovaných metod předzpracování obrazu a shlukovacích algoritmů. Popisuje příkazy a vysvětluje parametry, které jsou zde využívány. Dále se také stručně zabývá postupem a příkazy využívanými při programování aplikace.

Diplomová práce provádí testování použitých metod předzpracování a použitých shlukovacích algoritmů. Vyhodnocuje výsledky zatřídění použité databáze pro různý počet vybraných tříd.

2 Extrakce příznaků

Extrakce příznaků je operace, při které jsou z obrazu vyextrahována pouze ta data, která mají pro další zpracování smysl. Tato operace je úzce spojena s problematikou redukce dimenze. V tomto kontextu lze říci, že extrakce příznaků redukuje dimenzi obrazu tak, že nepodstatné části obrazu jsou odstraněny a zůstanou pouze důležitá data. Extrakci příznaků lze rozdělit na detekci příznaků a deskripci příznaků [5]. Před samotným popisem extrakce příznaků je nejprve nutné definovat rastrový digitální obraz.

2.1 Rastrový digitální obraz

Digitální obraz je v počítačové technice obraz, který je reprezentován binární formou. Rastrový obraz je tvořen pixely, což jsou nejmenší elementární jednotky rastrové grafiky. Pixely jsou uspořádány ve dvojrozměrné či trojrozměrné matici. Matice pixelů tedy definuje obraz a jeden pixel odpovídá jedné buňce matice. Každý pixel má přesně určenou polohu v matici. Rozměr neboli rozlišení obrazu určuje počet pixelů ve vodorovné a svislé ose. Nevýhodou rastrové grafiky je, že při zvětšení velikosti obrazu se zhorší obrazová kvalita obrazu. Při velkém zvětšení jsou v obraze patrné rastry [1].

Každý pixel je charakterizován jasem a barvou. Jas odpovídá svítivosti plošky reálného obrazu, která je promítnutá do daného pixelu. Černá barva tedy odpovídá nulové svítivosti a v počítačové grafice je zapsána číslem 0, kdežto bílá barva představuje maximální svítivost a je zapsána maximálním použitelným číslem.

Výsledná barva pixelu je dána kombinací barev. Informace o barvě bývá obvykle reprezentována jako bod v barevném modelu. Existuje více barevných modelů jako RGB, CMYK, HSV nebo HSL, z nichž nejpoužívanější je barevný model RGB. Barvu tedy určuje vektor hodnot tří barev, který nazýváme barevnou hloubkou. Nejčastěji se každé barvě přiřazuje hodnota 0 až 255, což představuje 8 bitů. Složením těchto tří barev se získá barevná hloubka 24 bitů [2]. Tato diplomová práce pracuje s barevným a šedotónovým obrazem. Barevný obraz je definován třemi maticemi. Obvykle první matice reprezentuje červenou barvu, druhá modrou barvu a třetí zelenou barvu. Nutno ale zmínit, že pořadí jednotlivých barevných matic se může lišit.

2.1.1 Šedotónový obraz

Šedotónový obraz je definován pouze jednou maticí pixelů. Při převodu barevného obrazu na šedotónový jsou zredukovány dimenze, což zmenší výpočetní nároky extrakce příznaků a shlukování, za cenu ztráty informace o barevnosti obrazu. Lidské

oko je nedokonalé a není citlivé na všechny základní barvy stejně. Nejvíce je citlivé na zelenou barvu, potom na červenou a nejméně na modrou. Proto se pro přepočet intenzity barvy z barevného obrazu na šedotónový obraz používá vztah [2]

$$I = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B. \tag{2.1}$$

Poté je pixel definován pouze jedním číslem, které má obvykle osm bitů a v tom případě dosahuje hodnot od 0 do 255.

2.2 Příznaky

Obrazové příznaky jsou data, která nesou informaci obrazu a používají se v aplikacích souvisejících se zpracování obrazu. Příznaky lze chápat jako malé části obrazu, které vynikají oproti svému okolí a obvykle zaujmou divákovu pozornost. V místě příznaku se zpravidla výrazně mění intenzita obrazu[4].

Obrazové příznaky lze rozdělit do dvou typů. Prvním typem příznaků jsou bodové příznaky (rohy). Ty se nachází na specifických místech v obraze a často je popisují pixely, které se nachází kolem nich. Mohou to být například: rohy budov, rohy objektů, vrcholky hor, černá tečka na bílé zdi, hvězda na noční obloze a mnohé další.

Příznaky druhého typu nejsou jen jedním bodem, ale jsou to delší textury, u kterých velmi záleží na jejich směru a orientaci. Často bývají takové příznaky označovány jako hrany. Jsou to například: hrany budov, okraje objektů a další [5].

Příznaky mají obvykle tyto vlastnosti [4]:

- pevně definovanou pozici v obraze,
- matematickou definici,
- jejich okolí obsahuje velké množství užitečných informací,
- disponují vysokou odolností proti působení lokálních a globálních deformací.

2.3 Detekce příznaků

Operace detekce příznaků má za úkol detekovat příznaky v obraze. To znamená, že hledá umístění příznaků v obrazové matici. V ideálním případě by měl být příznak nalezen co nejpřesněji a měl by co nejlépe splňovat všechny vlastnosti příznaků uvedené výše. Většinou je kladen požadavek na nalezení co největšího počtu příznaků. Nutno ale poznamenat, že čím více příznaků je detekováno, tím více narůstají výpočetní nároky. Navíc nejdůležitější informace obrazu bývají v prvních desítkách či stovkách detekovaných příznaků a s další detekcí příznaků důležité informace přibývají jen minimálně. Proto je důležité najít kompromis a detekovat takový počet

příznaků, aby byla nalezena valná většina důležitých informací z obrazu a zbytečně nenarůstaly výpočetní nároky.

Existuje spousta algoritmů, které se zabývají problematikou detekce příznaků. Tyto algoritmy se nazývají detektory (operátory) příznaků. Nejznámější detektory příznaků jsou: Moravcův, Harrisův, Shi-Tomasi, FAST [7], SIFT, SURF a ORB [9]. Nutno říci, že algoritmy SIFT, SURF a ORB nejsou jen detektory, ale zároveň i deskriptory příznaků. Takovéto algoritmy bývají nazývány extraktory příznaků a nejjednodušším z nich je diskrétní konvoluce [4].

2.3.1 FAST

Algoritmus FAST (Features From Accelerated Segment Test) je detektor bodových příznaků vyvinutý Edwardem Rostenem a Tomem Drummondem v roce 2006 [7]. Detekce je provedena na základě rozdílu intenzit testovaného pixelu a sousedních pixelů. Jedná se o velmi rychlou a jednoduchou metodu, která ale dosahuje vysoké úspěšnosti.

Algoritmus pracuje následujícím způsobem:

1. Nejprve je vytvořena kružnice o poloměru tří pixelů se středem v testovaném pixelu. Na kružnici okolo testovaného pixelu leží 16 pixelů [6].
2. Následně je porovnávána intenzita testovaného pixelu s pixely ležícími na kružnici. U každého z 16 pixelů na kružnici je určen rozdíl intenzity oproti testovanému pixelu [4]. Testovaný pixel je považován za příznak, pokud je u n nebo více pixelů ležících na kružnici určen větší rozdíl intenzity, než je předem nastavený práh rozdílu intenzity. Nutno poznamenat, že všechny pixely, které splňují tuto podmínku, spolu musí sousedit (musí být vedle sebe), jinak testovaný pixel nebude jako příznak určen. Ve většině případů bývá počet pixelů splňujících podmínku většího rozdílu intezity nastaven na 12 (n=12). Podmínka, kterou musí pixel splňovat, aby mohl být algoritmem označen za příznak, lze vyjádřit vztahem [6]
$$|I_t - I_k| > T, \tag{2.2}$$
kde I_t je intenzita testovaného pixelu, I_k je intenzita pixelu ležícího na kružnici ($k \in (1, 16)$) a T je práh rozdílu intenzity. Tímto způsobem algoritmus projde každý pixel v obraze a určí, zda je příznakem nebo ne.
3. Pokud bylo jako příznak určeno více pixelů, které spolu sousedí, tak algoritmus z každé této oblasti sousedících příznaků vybere jeden příznak s největší či nejmenší intenzitou a ostatní příznaky zahodí. Tato operace se nazývá potlačení nemaximálních hodnot [6].

Metoda bývá zrychlena tím, že není porovnávána intenzita u všech 16 bodů na kružnici, ale pouze u čtyř z nich. Postupně se určí rozdíl intenzity pixelů 1, 9, 5 a 13

od testovaného pixelu. Testovaný pixel je považován za příznak, pokud alespoň tři ze čtyř pixelů mají větší rozdíl intenzity, než je nastavený práh rozdílu intenzity [4].

Podle autorů [7] je FAST detektor dvakrát rychlejší než detektor SIFT a dokonce rychlejší než Harrisův operátor. Autoři dále uvádějí srovnatelnou úspěšnost s algoritmem SIFT. Výhodou algoritmu FAST je tedy jeho rychlost oproti ostatním detektorům a zároveň jeho relativně vysoká úspěšnost. Nevýhodou je nízká odolnost vůči šumu a závislost na nastavení vhodné hodnoty prahu rozdílu intenzit [7].

2.4 Deskripce příznaků

Poté, co jsou detektorem příznaků nalezeny příznaky v obraze, je nutné tyto příznaky matematicky popsat. Touto problematikou se zabývá proces deskripce příznaků. Vhodně popsané příznaky disponují těmito vlastnostmi:

- robustnost,
- opakovatelnost,
- invariance vůči rotaci,
- invariance vůči posunutí,
- odolnost vůči změně světelných podmínek.

Proces deskripce příznaků zakóduje každý příznak detekovaný detektorem příznaků do série číslic (deskriptorového vektoru). Výstupem deskripce příznaků jsou pak deskriptorové vektory popisující příznaky.

Algoritmy zabývající se popisem příznaků se nazývají deskriptory příznaků. Většina z nich jsou relativně moderní metody a fungují na různých principech. Liší se také tím, zda je výstup decimální či binární. Mezi nejznámější deskriptory příznaků patří: SIFT, SURF, HOG, BRISK, MSER, BRIEF [8] a ORB [9].

2.4.1 BRIEF

Algoritmus BRIEF (Binary Robust Independent Elementary Features) je deskriptor příznaků založený na porovnávání pixelů, které byly detektorem příznaků určeny jako příznaky, s jejich okolními pixely [4].

Princip deskriptoru je následující [6]:

1. Nejprve je oblast kolem každého příznaku kolvována čtvercovým gaussovským oknem o velikosti $9 \cdot 9$ pixelů s rozptylem 2.
2. Poté jsou z oblastí kolvovaných gaussovským oknem vybrány pixely podle gaussovského rozložení $G(0, \frac{1}{25} \cdot 9^2)$.
3. V každé oblasti jsou takto vybrané pixely porovnány s pixelem, který byl určen jako příznak. Porovnání provádí funkce [6]

$$\tau = \left\{ \begin{array}{lcc} 1 & : & p(x) > p_i(y) \\ 0 & : & jinak \end{array} \right\}, \tag{2.3}$$

kde $p(x)$ je intenzita pixelu určeného za příznak a $p_i(y)$ jsou intenzity pixelů vybraných pomocí gaussovského rozložení.

4. Nakonec je každému příznaku přiřazen deskriptorový vektor. Tento vektor má obvykle velikost n_d 126, 256 nebo 512 prvků a je definován funkcí [6]

$$f_{d_n}(p) = \sum_{i=1}^{n_d} 2^{i-1} \cdot \tau(p, x_i, y_i). \tag{2.4}$$

Ze vztahu 2.3 lze vyčíst, že výstup deskriptoru je binární. Deskriptorové vektory jednotlivých příznaků jsou tedy binárními řetězci. To je velmi důležité pro další zpracování. Podle autorů je deskriptor BRIEF rychlejší než deskriptor SURF. Dále uvádí, že jeho efektivita je s tímto deskriptorem srovnatelná [8]. Výhodou algoritmu je tedy rychlost a nízká výpočetní náročnost. Nevýhodou naopak je závislost na rotaci, kdy deskriptor BRIEF dokáže pracovat jen s málo pootočenými obrazy [4].

2.5 ORB

ORB (Oriented FAST and Rotated BRIEF) je algoritmus určený pro extrakci příznaků z obrazu. Byl vytvořen ve snaze poskytnout méně výpočetně náročnou alternativu pro komerční využití k algoritmům SIFT a SURF, které jsou licencované. Publikoval ho Ethanom Rublee se svými kolegy na Mezinárodní konferenci o počítačovém vidění v roce 2011. Autoři uvádí, že úspěšnost algoritmu ORB je srovnatelná s algoritmem SIFT a je o dva řády rychlejší [9].

Dokáže v obraze najít i matematicky popsat příznaky. Jedná se tedy o detektor a deskriptor příznaků v jednom. Je složen z příznakového detektoru FAST a deskriptoru BRIEF, které jsou modifikovány tak, aby byly odolné vůči změně rotace příznaků.

2.5.1 ORB detektor

Detekce příznaků algoritmem ORB probíhá v následující krocích [6]:

1. Nejprve je zkonstruována scale space pyramida metodou DoG (Difference of Gaussians) používanou v algoritmech SIFT a SURF, díky které je detekce příznaků méně závislá na velikosti příznakové oblasti. Pomocí metody DoG je vstupní obraz rozdělen do pyramidy, jejíž vrstvy (gausiány) jsou postupně více rozostřené verze vstupního obrazu. Rozostření je provedeno aplikací gaussovského filtru s postupně rostoucím rozplytem. Obvykle je pyramida sestavena

z pěti gausiánů. První gausián je vytvořen gaussovským filtrem s rozptylem $\sigma_0 = 1.6$ a rozdíly mezi jednotlivými rozptyly jsou $k = \sqrt{2}$. Poté je každý gausián ještě rozdělen do několika oktáv (obvykle čtyři). V každé oktávě je zredukováno rozlišení obrazu na polovinu.

2. Dále jsou detektorem FAST, který byl popsán v podkapitole 2.3.1, ve scale space pyramidě nalezeny příznaky. Algoritmus detekuje stejné příznaky v různých oktávách jednotlivých gausiánů a tím je dosažena invariance vůči velikosti příznakové oblasti.
3. Poté se Harrisovým kritériem $\mathbf{R}$ vybírají nejlepší příznaky. Harrisův operátor je vylepšením Moravcova operátoru, který říká, že pokud je pixel příznakem, musí oproti okolním pixelům vynikat (musí být ve všech směrech od něj výrazná změna jasu). Kritérum Harrisova operátoru lze vypočítat vztahem [6]

$$\mathbf{R} = det(\mathbf{M}) - k \cdot stopa(\mathbf{M})^2, \tag{2.5}$$

kde $det(\mathbf{M})$ je determinant matice $\mathbf{M}$, k je konstanta určují citlivost na hrany a $stopa(\mathbf{M})$ je součet prvků na hlavní diagonále čtvercové matice. Matice $\mathbf{M}$ je vypočtena vztahem [6]

$$\mathbf{M} = \sum_{u,v} \mathbf{w}_{u,v} \cdot \begin{bmatrix} X^2 & X \cdot Y \\ X \cdot Y & Y^2 \end{bmatrix}, \tag{2.6}$$

kde $\mathbf{w_{u,v}}$ je váhovací okno a X a Y jsou aproximované parciální derivace obrazu ve směru osy x a y. Následně se Harrisova kritéria $\mathbf{R}$ vypočítaná pro všechny příznaky porovnají se zvoleným prahem. Je vybráno pouze N příznaků, které mají největší rozdíl hodnoty Harrisova kritéria $\mathbf{R}$ od nastaveného prahu.
4. Posledním krokem je určení směru všech příznaků vybraných kritériem $\mathbf{R}$. Díky tomu jsou příznaky invariantní vůči rotaci obrazu. K určení směru příznaku je neprve nutné vytvořit oblast se středem v daném příznaku. V této oblasti jsou spočítány momenty podle rovnice [9]

$$m_{pq} = \sum_{x,y} x^p \cdot y^q \cdot I(x,y), \tag{2.7}$$

kde $I(x,y)$ je intenzita pixelu v místě obrazu x, y. Následně je z těchto momentů vypočten centroid pomocí vztahu [9]

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}). \tag{2.8}$$

Orientaci příznaku určuje vektor z místa daného příznaku do místa centroidu. Směr příznaku určuje úhel θ, který je vypočítán podle rovnice [9]

$$\theta = atan2(m_{01}, m_{10}). \tag{2.9}$$

Obr. 2.1: Detekované příznaky s rotací a velikostí detektorem ORB [59]

2.5.2 ORB deskriptor

Po detekci příznaků je provedena deskripce příznaků. Algoritmus ORB popisuje příznaky následujícím způsobem [6]:

1. Nejprve je celý obraz vyhlazen integrálem na okně $5 \cdot 5$.
2. Následně je proveden první a druhý krok algoritmu BRIEF, který byl popsán v podkapitole 2.4.1. Z pixelů, které byly vybrané v prvním a druhém kroku algoritmu BRIEF, se vytvoří matice [6]

$$\mathbf{S} = \begin{pmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{pmatrix}. \tag{2.10}$$

   Tato matice $\mathbf{S}$ se přenásobí maticí rotace s úhlem θ, který byl vypočten při detekci algoritmu ORB. Násobení matic ilustruje vztah [6]

$$\mathbf{S}_\theta = \mathbf{R}_\theta \cdot \mathbf{S}, \tag{2.11}$$

   kde $\mathbf{R}_\theta$ je matice rotace. Pixely zapsané v matici $\mathbf{S}_\theta$ jsou pak invariantní vůči rotaci obrazu.
3. Nakonec je proveden třetí a čtvrtý krok algoritmu BRIEF, který by popsán v kapitole 2.4.1.

Výstupem algoritmu ORB je tedy binární deskriptorový vektor **d** pro každý příznak. Ve výchozím nastavení deskriptorový vektor algoritmu ORB obsahuje 32 osmibitových čísel (d(0) až d(31)). Příklad takového vektoru ilustruje vztah

$$\mathbf{d} = (15, 20, 180, 240, 51, 49, 57, 128, 126, 12, 36, 67, 59, 98, 51, 241, 163, 16, 23,$$
$$28, 65, 48, 37, 159, 91, 84, 141, 201, 139, 77, 51, 8). \tag{2.12}$$

Celý obraz je pak popsán maticí **O**, kdy jeden řádek matice je jeden deskriptorový vektor a počet řádků určuje počet deskriptorových vektorů (příznaků). Ve výchozím nastavení algoritmus ORB detekuje a popíše 500 příznaků, tento případ ilustruje matice

$$\mathbf{O} = \begin{pmatrix} d_1(0) & d_1(1) & \dots & d_1(31) \\ d_2(0) & d_2(1) & \dots & d_2(31) \\ \vdots & \vdots & \vdots & \vdots \\ d_{500}(0) & d_{500}(1) & \dots & d_{500}(31) \end{pmatrix}. \tag{2.13}$$

2.6 BRISK

Algoritmus BRISK (Binary Robust Invariant Scalable Keypoints) je dalším extraktorem příznaků publikovaným v roce 2011. Autoři uvádí, že jeho úspěšnost je srovnatelná s algoritmem SURF, ale má nižší výpočetní nároky [10]. Jedná se o kombinaci detektoru a deskriptoru příznaků, které ale mohou být použity i samostatně. Příznaky, které jsou detekované a popsané algoritmem BRISK, jsou do určité míry invariantní vůči rotaci a změně rozměru obrazu. Autoři připouští, že při těchto transformacích ve velkém měřítku, dosahuje algoritmus horších výsledků než algoritmus SURF [4].

2.6.1 BRISK detektor

Algoritmus BRISK při detekci příznaků vychází ze scale space pyramidy algoritmu SIFT a používá detektor příznaků FAST. Detekce příznaků probíhá následujícím způsobem [10]:

1. Nejprve je zkonstruována scale space pyramida metodou DoG, která byla popsána v kapitole 2.5.1.
2. V dalším kroku jsou detekovány příznaky algoritmem FAST, který byl popsán v kapitole 2.3.1. Příznaky jsou detekovány ve všech oktávách se stejnou hodnotou prahu T.
3. Následně je použita operace potlačení nemaximálních hodnot, která je ovšem rozšířena následujícím způsobem. Aby byl pixel označen za příznak, musí splňovat dvě podmínky. První podmínkou je, že okolních osm pixelů musí mít

menší intenzitu. Druhou podmínkou je, že daný pixel musí mít větší intenzitu, než pixel na stejné pozici v sousedních oktávách (v oktávách nad a pod touto oktávou).

4. V dalším kroku je vytvořena parabola z těchto pixelů:
   - pixel označený jako příznak,
   - pixel na pozici příznaku o oktávu výš,
   - pixel na pozici příznaku o oktávu níž.

   Skutečná hodnota příznaku je pak určena jako maximum této paraboly.
5. Posledním krokem je interpolace souřadnic pixelů ze sousedních oktáv, ze kterých byla počítána parabola.

Obr. 2.2: Detekované příznaky s rotací a velikostí algoritmem BRISK [59]

2.6.2 BRISK deskriptor

Algoritmus BRISK popisuje příznaky následovně [10]:

1. Nejprve je z obrazu kolem příznaku vytvořen vzorkovací obrazec o velikosti $40 \cdot 40$ pixelů. V tomto obrazci jsou vytvořeny čtyři rovnoměrně rozložené soustředné kružnice se středem v daném příznaku. Na těchto kružnicích je vytvořeno N bodů. Aby se předešlo při vzorkování intenzity bodů aliasingu, obraz je filtrován gaussovským filtrem s rozptylem σ_i úměrným vzdálenosti jednotlivých bodů nacházejících se na stejné kružnici [4].

2. Z těchto N bodů jsou vybrány dva body p_i a p_j, pro jejichž intenzitu platí $I(p_i, \sigma_i)$ a $I(p_j, \sigma_j)$. Poté je vypočítán gradient mezi těmito body jako [10]

$$g(p_i, p_j) = (p_i - p_j) \cdot \frac{I(p_j, \sigma_j) - I(p_i, \sigma_i)}{\|p_j - p_i\|^2}. \tag{2.14}$$

3. Následně se rozdělí všechny N body na dvě množiny. Pokud jsou body menší než δ_{max}, patří do množiny blízkých párů. Pokud jsou body větší než δ_{min}, patří do množiny vzdálených párů. Hodnota δ_{max} se vypočítá jako $\delta_{max} = 9,75 \cdot t$ a hodnota δ_{min} jako $\delta_{min} = 13,67 \cdot t$, kde t je rozměr příznaku.
4. Poté se pomocí bodů v množině vzdálených párů spočítá celkový směr příznaku rovnicí [10]

$$g = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \frac{1}{L} \cdot \sum_{(p_i, p_j) \in L} g(p_i, p_j). \tag{2.15}$$

5. Aby byl příznak invariantní vůči rotaci, je vzorkovací obrazec rotován kolem daného příznaku o úhel $\alpha = arctan2(g_y, g_x)$.
6. Posledním krokem je porovnání intenzit bodů z množiny blízkých párů (p_i^α, p_i^α). Pokud je intenzita bodu p_j^α větší než intenzita bodu p_i^α, je bitu přiřazena hodnota jedna (v opačném případě hodnota nula). Pro každý bit je tedy počítána rovnice [10]

$$b = \left\{ \begin{array}{rcc} 1 & : & I(p_j^\alpha, \sigma_j) > I(p_i^\alpha, \sigma_i) \\ 0 & : & jinak \end{array} \right\}. \tag{2.16}$$

Příznak je tedy popsán binárním deskriptorovým vektorem $\mathbf{d}$, který se skládá z osmibitových čísel. Každé osmibitové číslo je složeno z bitů vypočítaných v posledním kroku popisu BRISK deskriptoru. Ve výchozím nastavení tvoří deskriptorový vektor 64 osmibitových čísel. Deskriptorový vektor algoritmu BRISK je tedy dvakrát delší než deskriptorový vektor algoritmu ORB. Obdobně jako u algoritmu ORB by šel zapsat příklad deskriptorového vektoru $\mathbf{d}$ 2.12. Deskriptorové vektory jsou v matici uspořádany stejným způsobem jako v případě algortitmu ORB 2.13.

2.7 BoVW model

2.7.1 Důvody k použití BoVW modelu

Pokud jsou příznaky extrahované výše uvedenými algoritmy předloženy přímo shlukovacím algoritmům, bude výsledná klasifikace obrazů dosahovat velmi nízké úspěšnosti, a to ze dvou důvodů [11]. Prvním a závažnějším důvodem je skutečnost, že extrahované příznaky se v obraze nachází na různých místech. Pokud jsou takovéto příznaky předloženy shlukovacímu algoritmu K-means, bude měřit vzdálenosti mezi příznaky, které spolu nesouvisí.

Tuto problematiku lze demonstrovat na následujícím případu: Jsou dva obrazy, které patří do třídy orel. Na prvním obraze se orel nachází vlevo nahoře a na druhém vpravo dole. Z obou obrazů jsou extrahovány příznaky. V prvním obraze 10. deskriptorový vektor popisuje příznak charakterizující zobák orla a 480. deskriptorový vektor popisuje příznak listu na stromě. Na druhém obraze ale 10. deskriptorový vektor popisuje příznak mraku na obloze a 480. deskriptorový vektor popisuje příznak zobáku orla. Po shlukovacím algoritmu k-means je tedy požadováno, aby porovnával vzdálenosti dvou deskriptorových vektorů, z nichž každý popisuje úplně jiný příznak.

Druhým důvodem je, že extraktory příznaků nejsou schopny z každého obrazu extrahovat stejné množství příznaků. Shlukovací algoritmy potřebují ke svému fungování, aby každý obraz, který je převeden na vektor a je definovaný deskriptorovými vektory jednotlivých přínaků, byl stejně dlouhý. Existuje více možností, jak tento problém řešit. Tato práce testuje následující možnosti:

1. Nastavit počet extrahovaných příznaků ze všech obrazů podle obrazu, kde jich bylo nalezeno nejméně. Každý obraz pak bude popsán stejným počtem deskriptorových vektorů.
2. U všech obrazů, u kterých byl nalezen nižší počet příznaků než nastavený, doplnit deskriptorové vektory nulami.

Obě možnosti přináší do systému určité zkreslení, které má vliv na výslednou klasifikaci obrazů.

Z výše uvedených důvodů je zřejmé, že před použitím shlukovacích algoritmů je nutné deskriptorové vektory určitým způsobem poskládát. K tomu slouží matematické modely. Tato práce se zabývá matematickým modelem BoVW (Bag of Visual Words) [11].

2.7.2 Princip BoVW modelu

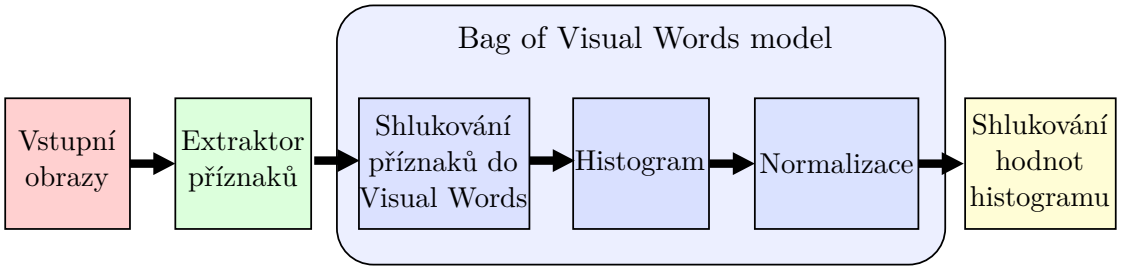
Model BoVW vychází z modelu BoW (Bag of Words), který slouží k analýze textu. Principem BoVW modelu je zařadit podobné příznaky do skupin, které bývají nazývané jako Visual Words (codewords). Následně vytvořit histogram pro každý obraz, který udává, kolik příznaků vyextrahovaných z každého obrazu patří do každého Visual Word. Tento histogram pro každý obraz se nazývá Bag of Visual Words, odtud pochází také název matematického modelu. BoVW model pracuje v následujících krocích [11]:

1. Nejprve jsou všechny deskriptorové vektory všech obrazů, které budou klasifikovány, seřazeny do skupin Visual Words podle podobnosti. To lze provést jakoukoliv metodou strojového učení s učitelem či bez učitele. Tato práce pro

tuto problematiku používá modifikaci algoritmu K-means zvanou K-majority [12]. Počet Visual Words udává počet shluků v algoritmu K-majority.

2. Dále je pro každý obraz vytvořen histogram. Histogram udává, kolik deskriptorových vektorů z celkového počtu patří do každého Visual Word. Jinými slovy ukazuje četnost výskytu deskriptorových vektorů pro každé Visual Word.
3. Posledním krokem je normalizace histogramu každého obrazu.

Výstupem BoVW modelu jsou normalizované hodnoty histogramu pro každý obraz, tedy normalizovaná četnost výskytu příznaků v každém Visual Word pro každý obraz [11]. Na obrázku 2.3 je zobrazeno blokové schéma celého procesu klasifikace obrazů s použím BoVW modelu.



Obr. 2.3: Klasifikace obrazů s použítím BoVW modelu

Problémem prvního kroku BoVW modelu je, že algoritmy ORB a BRISK popisují příznaky v binární formě, tedy binárními deskriptorovými vektory. Jelikož se tato práce zabývá strojovým učením bez učitele, je vhodné deskriptorové příznaky třídit shlukovacími algoritmy. Pro binární klasifikaci lze použít tři algoritmy, K-medoids, Aglomerativní shlukování a K-majority. Algoritmy K-medoids a Aglomerativní shlukování jsou schopné provést shlukování s binárními čísly. Problémem ale je, že pro výpočet potřebují vytvořit matici obsahující vzdálenosti mezi všemi deskriptorovými vektory tzv. distance matrix [12]. Oba algoritmy jsou tedy pro tuto problematiku nepoužitelné, protože algoritmus ORB ve výchozím nastavení detekuje 500 příznaků a každý z těchto příznaků je popsán 32 osmibitovými čísly. Pro třídění již mírně větší databáze bude tedy matice deskriptorových vektorů nabývat obrovských rozměrů a algoritmy K-medoids a Aglomerativní shlukování by vyžadovaly enormní výpočetní nároky [12]. Práce tedy využívá poslední ze zmíněných možností, a tou je algoritmus K-majority, který bude popsán v kapitole 3.7.6.

Práce dále řeší normalizaci histogramu dvěma metodami. Při první metodě je každá hodnota histogramu dělena maximální hodnotou histogramu. Pro jeden obraz

lze zapsat rovnici, podle které jsou hodnoty v histogramu normalizovány

$$f_{i_{norm}} = \sum_{i=1}^{k} \frac{f_i}{f_{max}}, \tag{2.17}$$

kde $f_{i_{norm}}$ je normalizovaná hodnota četnosti výskytu příznaku v daném Visual Word, f_i je nenormalizovaná hodnota četnosti výskytu příznaku v daném Visual Word, f_{max} je maximální nenormalizovaná hodnota četnosti výskytu příznaku v celém histogramu a k je počet Visual Words. Po tomto typu normalizace se hodnoty histogramu nachází v intervalu $(0, 1)$.

Druhou metodou při normalizaci histogramu je vydělení každé hodnoty histogramu počtem všech deskriptorových vektorů získaných z aktuálního obrazu. Pro jeden obraz lze zapsat rovnici, podle které jsou hodnoty v histogramu normalizovány

$$f_{i_{norm}} = \sum_{i=1}^{k} \frac{f_i}{n}, \tag{2.18}$$

kde n je počet deskriptorových vektorů daného obrazu. Při této metodě je součet všech normalizovaných hodnot histogramu roven 1.

2.8 Diskrétní 2D konvoluce

Diskrétní 2D konvoluce je oproti ostatním extraktorům příznaků jednoduchá operace, kterou lze extrahovat příznaky z obrazu. Výsledek konvoluce závisí na konvolučním jádru k, které je definováno maticí o rozměrech $i \cdot j$. Výstupní obraz $P_{výst}(x, y)$ je vypočítán tak, že každý pixel vstupního obrazu $P(x, y)$ je vynásoben konvolučním jádrem $k(i, j)$ a všechny dílčí součiny jsou sečteny [13].

Rozměr konvoluce určuje, v kolika osách se konvoluční jádro pohybuje a jaký rozměr bude mít výstupní obraz po konvoluci. Rozměrem konvoluce je myšleno, zda se jedná o 1D, 2D nebo 3D konvoluci. Pro 2D konvoluci se tedy jádro bude pohybovat po dvou osách (x, y) a výstupní obraz bude dvojrozměrný. Pro šedotónový obraz lze použít dvourozměrné konvoluční jádro, ale pro barevný obraz, který je trojrozměrný, je možné použít trojrozměrné jádro. To se sice pohybuje pouze ve dvou směrech, ale pokud má jádro stejnou hloubku jako barevný obraz, není nutné, aby se po hloubkové ose pohybovalo.

Diskrétní 2D konvoluci lze zapsat rovnicí [13]:

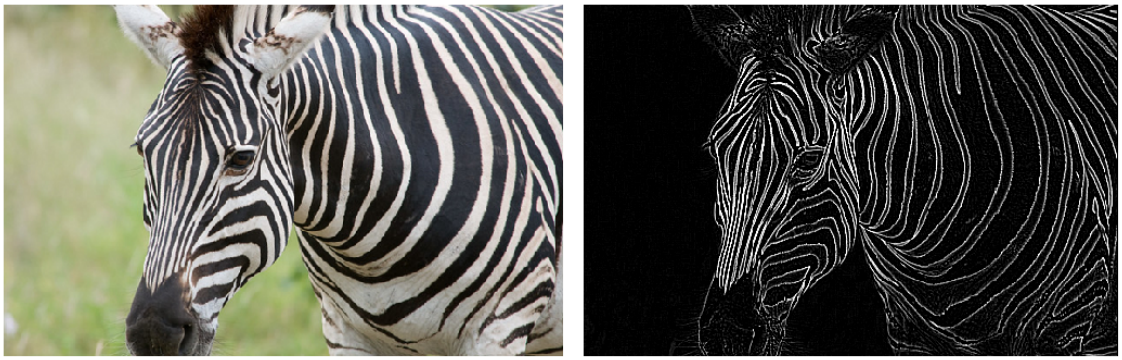
$$P_{výst}(x, y) = P(x, y) * k(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} P(x - i, y - j) \cdot k(i, j), \tag{2.19}$$

kde $P(x, y)$ je původní obraz, k je konvoluční jádro o rozměrech $i \cdot j$ a $P_{výst}(x, y)$ je výsledný obraz.

Použití 2D konvoluce na obrazu lze chápat jako filtrování obrazu, proto konvoluční jádro bývá často označováno jako filtr. Díky 2D konvoluci je možno obraz vyhlazovat (redukovat šum v obrazu), rozostřit nebo zaostřit. Pro klasifikaci obrazu se často používá konvoluční jádro, které je schopno zvýraznit hrany v obraze. Konvoluční jádro použité v procesu detekce hran je definováno maticí [13]:

$$k = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.20}$$

Na obraze 2.4 je zobrazen původní obraz a obraz vzniklý diskrétní 2D konvolucí s konvolučním jádrem použitým v procesu detekce hran.



Obr. 2.4: Ukázka diskrétní 2D konvoluce [59]

Ve výjimečných případech je možné při konvoluci určitý počet pixelů vstupního obrazu přeskočit. Konvoluční jádro tedy nemusí projít obraz pixel po pixelu. Tento proces bývá nazýván velikost kroku (stride). U většiny funkcí provádějících diskrétní konvoluci bývá stride implementován jako nastavitelný parametr. Výstupní obraz, který vznikl tak, že všechny pixely vstupního obrazu nebyly vynásobeny konvolučním jádrem podle rovnice 2.19, ztratí velkou část užitečné informace, ale jeho rozměr bude menší. Proto se ve valné většině případů provádí konvoluce s krokem jedna, tedy s každým pixelem vstupního obrazu [14].

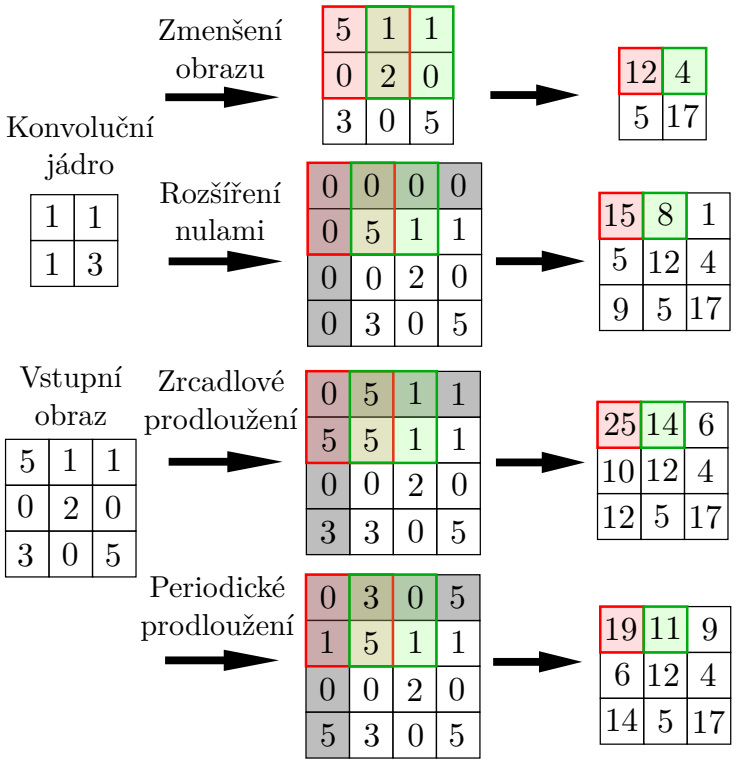
Dalším parametrem funkcí zabývajících se konvolucí bývá okrajový jev (padding). Ten udává, jak bude konvoluce řešena u krajních pixelů vstupního obrazu, u kterých se konvoluční jádro z části nachází mimo vstupní obraz. Například pokud bude konvoluce počítat výstupní hodnotu intenzity prvního pixelu vstupního obrazu s konvolučním jádrem o rozměrech $3 \cdot 3$, bude toto jádro z velké části mimo vstupní obraz (pět z devíti pixelů tohoto jádra nebude určeno). Existují následující přístupy, jak tento problém vyřešit:

1. **Zmenšení obrazu**: provede konvoluci pouze u pixelů vstupního obrazu, u kterých se celé konvoluční jádro nachází uvnitř obrazu. U krajních pixelů se

konvoluce neprovádí. Rozměr výstupního obrazu je zmenšen právě o krajní pixely [15].

2. **Rozšíření nulami**: rozšíří obraz pixely s nulovými hodnotami intezity. To znamená, že při výpočtu konvoluce krajních pixelů jsou pro části konvolučního jádra, které se nachází mimo původní obraz, dosazeny nulové hodnoty [15].
3. **Zrcadlové prodloužení**: rozšíří obraz svou zrcadlovou verzí. Konvoluce krajních pixelů se počítá z pixelů konvolučního jádra nacházejících se v obraze a z pixelů konvolučního jádra mimo obraz, které jsou zrcadlově otočeny oproti vstupnímu obrazu [16].
4. **Periodické prodloužení**: je velmi podobné přístupu zrcadlového prodloužení s tím rozdílem, že pro rozšíření obrazu není použit obraz zrcadlově otočený, ale obraz totožný se vstupním obrazem [16].

Pro lepší pochopení je vytvořen obrázek 2.5 ilustrující všechna nastavení parametru padding.



Obr. 2.5: Vliv parametru padding na diskrétní 2D konvoluci [17]

Všechny přístupy, až na první, zachovávají stejný rozměr obrazu, což je ve většině případů výhodné. Jakékoliv řešení okrajového jevu přináší do diskrétní konvoluce určité zkreslení. Toto zkreslení je ale zanedbatelné ze dvou důvodů. Prvním je, že většina obrazů se skládá ze stovek, tisíců i miliónů pixelů a počet krajních pixelů je velmi malý oproti celkovému počtu pixelů v obraze. Druhým důvodem je skutečnost,

že většina důležité informace v obrazech z použité databáze se nachází spíše ve středu obrazu než na jeho okrajích. Při rozšíření vstupu jsou tedy pro použitý dataset většinou zkresleny pouze nedůležité informace obrazu.

2.9 Diskrétní 3D konvoluce

V případě diskrétní 3D konvoluce se konvoluční jádro pohybuje ve třech osách (vodorovná osa x, svislá osa y a hloubková osa z) a výstup je také trojrozměrný. 3D konvoluce funguje na stejném principu jako 2D konvoluce, tedy výstupní obraz $P_{výst}(x, y, z)$ je vypočítán tak, že každý pixel vstupního obrazu $P(x, y, z)$ je vynásoben konvolučním jádrem $k(i, j, u)$ a všechny dílčí součiny jsou sečteny. Diskrétní 3D konvoluci lze zapsat rovnicí [18]

$$P_{výst}(x, y, z) = P(x, y, z) * k(i, j, u) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} P(x-i, y-j, z-u) \cdot k(i, j, u), \tag{2.21}$$

kde $P(x, y, z)$ je původní obraz, k je konvoluční jádro o rozměrech a $P_{výst}(x, y, z)$ je výsledný obraz. Tato práce pro diskrétní 3D konvoluci využívá trojrozměrné konvoluční jádro použité v procesu detekce hran.

2.10 Konvoluční autoenkodér

Autoenkodéry patří do oblasti umělých neuronových sítí. Umělé neuronové sítě jsou matematickými modely, které byly inspirovány neuronovou sítí lidského mozku. Stejně jako mozek se skládají z neuronů, které jsou matematickými funkcemi. Autoenkodéry jsou speciálním případem dopředných neuronových sítí a jako jedinné patří ke strojovému učení bez učitele. Autoenkodéry bývají označovány jako samoučící a snaží se dosáhnou toho, aby jejich výstup byl roven jejich vstupu [14].

Autoenkodér se skládá ze dvou hlavních částí, z enkodéru a dekodéru. Enkodér má za úkol zakódovat vstupní data a lze ho popsat funkcí [14] $f_1(x) = z$, kde x jsou vstupní obrazová data a z je komprimovaný výstup enkodéru. Dekodér se snaží takto zkomprimovaná data dekódovat a jeho výstupem jsou rekonstruovaná vstupní data enkodéru. Dekodér lze popsat funkcí [14] $f_2(z) = y$, kde y jsou rekonstruovaná vstupní data. Úkolem celého autoenkodéru je rekonstruovat vstupní obraz s co nejmenší chybou. Celý tento proces lze popsat složení rovnic enkódéru a dekodéru do rovnice

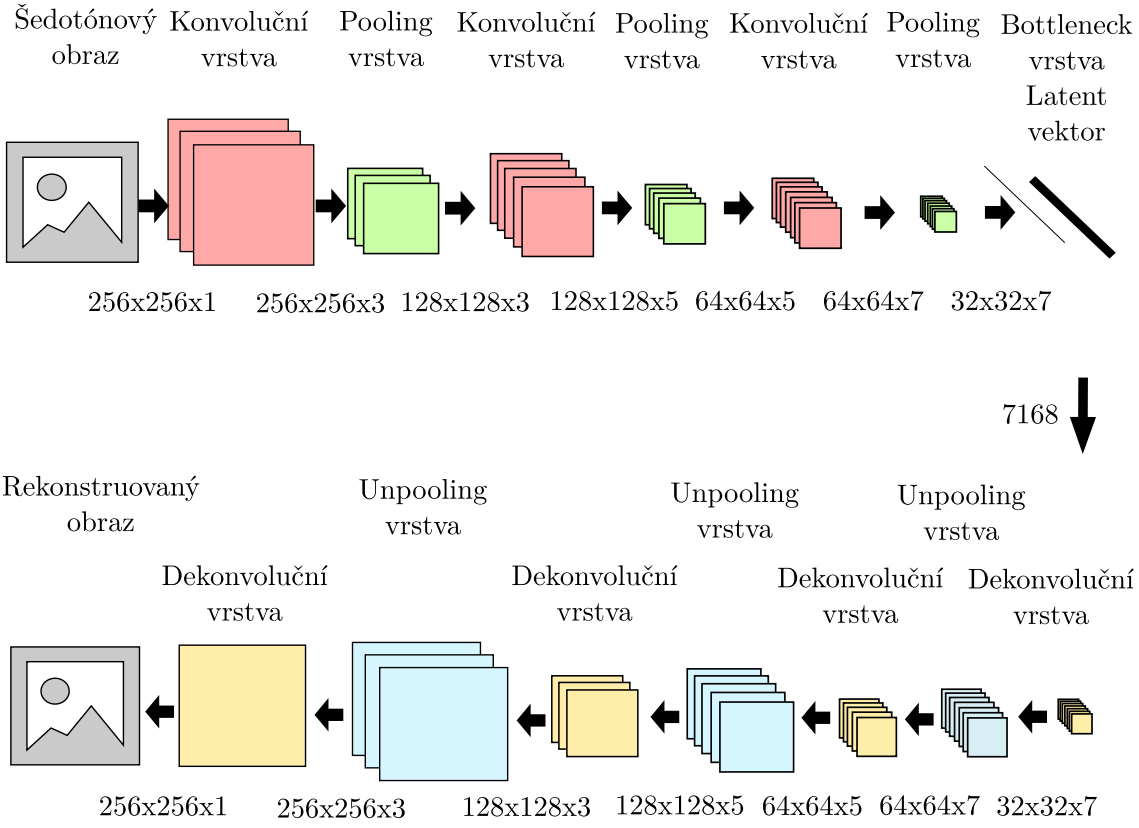
$$f_2(f_1(x)) = y. \tag{2.22}$$

Struktura a parametry autoenkodéru bývají nazývány modelem. Kódování pomocí autoenkodéru je metoda komprimace dat, která má následující vlastnosti [14]:

- ztrátovost,
- pevně daný vstupní formát dat,
- natrénování parametrů je určeno daty.

Vstupem i výstupem autoenkodéru jsou matice obrazových dat, jejichž rozměr musí být jednoznačně určen. Rozměr všech vstupních a výstupních obrazů musí být stejný.

Autoenkodér je složen z vrstev. Vrstvy se dělí podle účelu, matematické funkce a zapojení neuronů [14]. Práce se bude blíže zabývat pouze vrstvami určenými pro zpracování obrazu, tedy konvoluční a pooling vrstvou a jejich inverzními verzemi. Složení modelu autoenkodéru zobrazuje obrázek 2.6.



Obr. 2.6: Složení autoenkodéru

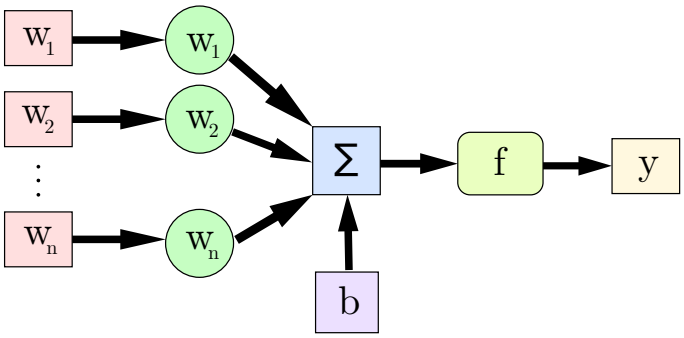
Existuje řada autoenkodéru. Tato práce se zabývá pouze konvolučním autoenkodérem, který je použit k extrakci příznaků z obrazu, které jsou následně předloženy shlukovacím algoritmům. Extrahované příznaky jsou komprimovaným výstupem enkodéru. Pro větší přehlednost jsou použity v konvoluční vrstvě pouze jednotky kanálů, v praxi se používají desítky až stovky.

2.10.1 Umělý neuron

V kontextu umělých neuronových sítí je umělý neuron matematická funkce. U konvolučního autoenkodéru jsou vstupy neuronu výstupy neuronů předchozí vrstvy, které jsou násobeny svými váhami. Pokud se jedná o neurony vstupní vrstvy, je jejich vstup matice snímků. Váha je parametr, který mění svoji hodnotu trénováním autoenkodéru. Všechny vážené vstupy jsou sečteny, a poté je k nim přičten bias. To je konstanta, která je opět trénovatelná a slouží jako práh, ovlivňuje tedy hodnotu sumy vážených vstupů neuronu. Na rozdíl od výstupů neuronů předchozí vrstvy, může být bias pouze jeden pro každý neuron. Následně je na součet vážených vstupů a konstanty bias aplikována aktivační funkce. Výstup neuronu u autoenkodéru je tedy pouze jeden a je použit jako jeden ze vstupů pro neurony následující vrstvy. Výstup neuronu definuje vztah [19]

$$y = f(\sum_{i=1}^{n} w_i \cdot x_i + b), \tag{2.23}$$

kde x_i jsou výstupy neuronů předchozí vrstvy, w_i jsou váhy příslušných výstupů neuronů předchozí vrstvy, b je trénovatelná konstanta bias, f je aktivační funkce a y je výstup neuronu. Model neuronu ilustruje obrázek 2.7.



Obr. 2.7: Model umělého neuronu [19]

2.10.2 Aktivační funkce

Aktivační funkce moduluje výstupní signál neuronu. Existuje mnoho aktivačních funkcí z nichž nejznámější jsou: ReLU (Rectified Linear Unit), ELU (Exponential Linear Unit), SELU (Scaled Exponential Linear Unit), Sigmoid, Identity, TanH, ArcTan, Binary step a Softmax. Každá z těchto funkcí má jiný průběh a každá má své využití v různých typech neuronových sítí. Tato práce využívá pouze aktivační funkce ReLU a Sigmoid.

ReLU je definovaná rovnicí [20]

$$f(x) = \left\{ \begin{array}{lcl} 0 & : & x < 0 \\ x & : & x \geq 0 \end{array} \right\} \tag{2.24}$$

Sigmoid (Logistic) je definována rovnicí [20]

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.25}$$

a říká, že pokud bude mít vstup jakoukoliv hodnotu, výstup bude mít hodnotu v rozmezí nula a jedna.

2.10.3 Konvoluční vrstva

Konvoluční vrstva konvolučního autoenkodéru pracuje na principu diskrétní konvoluce popsané v kapitole 2.8. Konvoluční vrstva je rozdělena do kanálů. V každém kanálu je provedena konvoluce s jiným konvolučním jádrem (filtrem). Výstupní obraz každého kanálu je vypočítán podle rovnice 2.19, a poté je k němu ještě přičten bias, který byl vysvětlen v kapitole 2.10.1. Rozměr celé konvoluční vrstvy je $x \cdot y \cdot k$, kde x je počet pixelů výstupního obrazu ve vodorovné ose, y je počet pixelů obrazu výstupního obrazu ve svislé ose a k je počet kanálů (filtrů) konvoluční vrstvy. Pokud je jako vstup konvoluční vrstvy použit trojrozměrný (barevný) obraz, výstupní obraz celé konvoluční vrstvy je rozšířen o hloubkovou osu z. Rozměr konvoluční vrstvy potom bude $x \cdot y \cdot z \cdot k$.

Konvoluční vrstva má následující parametry:

1. Rozměr konvolučních jader x, y, kde x je rozměr konvolučního jádra ve vodorovné ose a y je rozměr konvolučního jádra ve svislé ose. Nutno poznamenat, že všechny konvoluční jádra jedné konvoluční vrstvy mají stejný rozměr.
2. Parametr počet konvoluční jader k definuje, kolik kanálů konvoluční vrstva obsahuje. V každém kanálu konvoluční vrstvy je vždy použito jedno konvoluční jádro.
3. Parametr velikost kroku určuje, po kolika pixelech se bude konvoluční jádro posouvat. Byl vysvětlen v kapitole 2.8.
4. Parametr okrajový jev (padding) říká, jak se bude chovat diskrétní konvoluce na krajních pixelech obrazu. Byl vysvětlen v kapitole 2.8 [14].

Pomocí těchto parametrů lze vypočítat rozměr výstupního obrazu ve vodorovné i svislé ose podle rovnice [14]

$$P_{výst} = \frac{P_{vst} - K + 2 \cdot O}{S} + 1, \tag{2.26}$$

kde $P_{výst}$ je rozměr výstupního obrazu v dané ose, P_{vst} je rozměr vstupního obrazu v dané ose, K je rozměr filtru v dané ose, O je velikost okraje v dané ose a S je velikost kroku posunu konvolučního jádra v dané ose.

2.10.4 Pooling vrstva

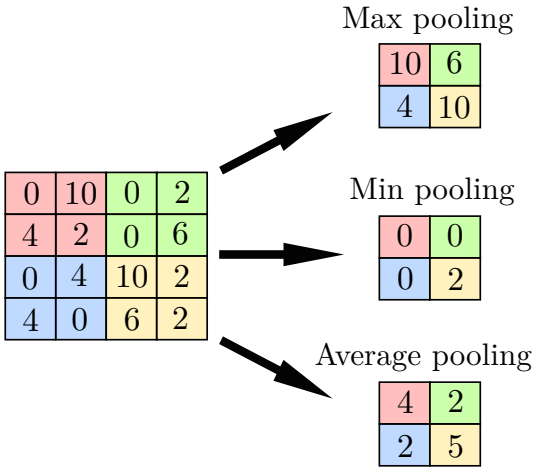
Při sestavování modelu konvolučního autoenkodéru je zvykem za každou konvoluční vrstvu aplikovat pooling (sdružovací) vrstvu. Operace pooling redukuje velikost dat, a tím redukuje množství parametrů předávaných další konvoluční vrstvě. Pooling redukuje rozměr obrazu ve svislé a vodorovné ose. Pooling vrstva má dva parametry, vertikální dělitel a horizontální dělitel, které určují velikost masky. Vertikální dělitel redukuje rozměr obrazu ve vodorovné ose a horizontální dělitel redukuje rozměr obrazu ve svislé ose. Rozměr výstupního obrazu lze vypočítat vztahem [14]

$$P_{výst}(x_{výst}, y_{výst}) = P_{vst}(\frac{x_{vst}}{v}, \frac{y_{vst}}{h}), \tag{2.27}$$

kde $P_{výst}$ je rozměr obrazu po aplikaci operace pooling, P_{vst} je rozměr vstupního obrazu, v je vertikální dělitel a h je horizontální dělitel.

Operace pooling posouvá masku po vstupním obraze vždy o velikost vstupní masky. Před každým posunutím vypočítá z pixelů vstupního obrazu nacházejících se uvnitř masky jeden pixel. Takto projde celý vstupní obraz a vypočítá pixely, které tvoří obraz výstupní. Nejčastějším případem je použití operace pooling s maskou $2 \cdot 2$ (vertikální dělitel a horizontální dělitel mají velikost 2). Uvnitř této masky se nachází 4 pixely. Operace pooling vypočítá z těchto pixelů jeden pixel podle dané metody. V tomto případě bude rozměr výstupního obrazu poloviční a rekuduje se tak 75% vstupních dat. Existuje více způsobů, jak lze vypočítat pixel z pixelů vstupního obrazu nacházejících se uvnitř masky [21]. Tyto pooling metody jsou zobrazeny na obrázku 2.8

- **Max pooling**: vybere z pixelů uvnitř masky pixel s největší hodnotou intenzity. Vybírá tedy z obrazu nejsvětlejší pixely. Tato metody je vhodná pro obrazy, které mají tmavší pozadí a důležité objekty jsou světlejší.
- **Min pooling**: naopak vybere z pixelů uvnitř masky pixel s nejmenší hodnotou intenzity, tedy nejtmavší pixel. Je vhodná pro obrazy, které mají světlejší pozadí a tmavší objekty.
- **Average pooling**: nevybírá konkrétní pixel z pixelů uvnitř masky, ale počítá aritmetický průměr jejich intenzit. Tento průměr je použit jako intenzita výstupního pixelu při každém posunu masky. Tato metoda vyhladí obraz a zmenší přechodové rozdíly intenzity jednotlivých pixelů obrazu [22].



Obr. 2.8: Pooling metody [14]

2.10.5 Bottleneck vrstva

Používají se dva přístupy, jak použít výstupní data enkodéru. Prvním způsobem je přímo použít výstup enkodéru jako vstup dekodéru. Výstup enkodéru představuje zkomprimovanou reprezentaci vstupních obrazů, čili enkodérem extrahované příznaky ze vstupních snímků. Tento výstup je obvykle více dimensionální, proto je nutné ho převést na vektor. Poté je již možné výstup enkodéru předat shlukovacím algoritmům, které mají za úkol samotnou klasifikaci databáze.

Při druhém přístupu se mezi výstup enkodéru a vstup dekodéru přidá plně propojená vrstva zvaná bottleneck, které jsou jako vstup předány zkomprimované reprezentace vstupních obrazů. Místo, kde jsou takto zakódovaná data uložena, bývá nazýváno Latent Space Representation. Výstup bottleneck vrstvy je poté použit jako vstup dekodéru. Shlukovacím algoritmům jsou pak předkládána data, která se nachází v Latent Space Representation. Tato data je opět nutné před samotným shlukováním převést na vektor [23].

Často se data převádí na vektor hned po výstupu z enkodéru. Pokud jsou zakódovaná data v Latent Space Representation uložena jako vektor, bývají nazývána Latent vektor. Před použitím dekodéru je nutné Latent vektor znovu převést na matici, protože dekodér očekává vstup ve stejném rozměru, jako výstup enkodéru. Shlukovacím algoritmům lze poté předat přímo Latent vektor [23].

2.10.6 Dekonvoluční vrstva

Dekonvoluční vrstva provádí inverzní operaci konvoluční vrstvy. Konvoluce, vynásobením pixelu vstupního obrazu konvolučním jádrem, vytváří jednu výstupní hod-

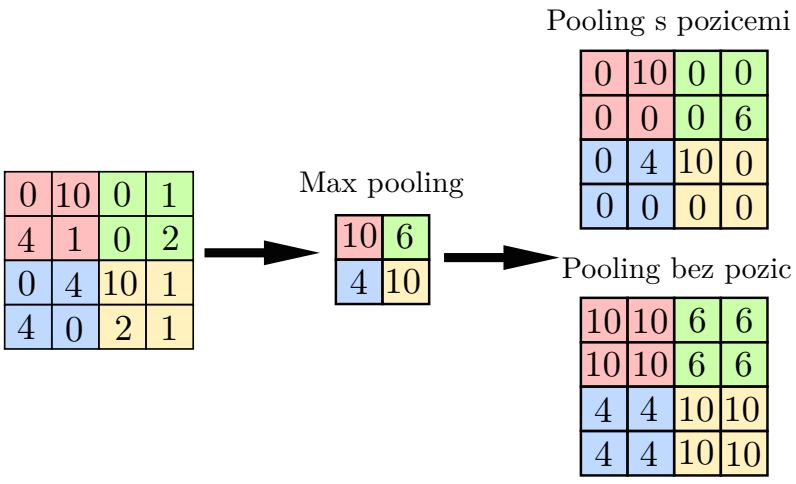
notu. Dekonvoluce naopak, vynásobením konvolučního jádra jedním pixelem vstupního obrazu, vytváří více pixelů výstupního obrazu. Dekonvoluce má stejné parametry jako konvoluce [24].

Pokud jsou na sebe napojeny konvoluční a dekonvoluční vrstva se stejnými parametry, bude mít výstup dekonvoluce stejný rozměr jako vstup konvoluce. S tímto principem pracuje autoenkodér. V enkodéru se používá většinou více konvolučních vrstev s určitými parametry. V dekodéru se poté používají dekonvoluční vrsty se stejnými parametry, ale opačným pořadím [24].

2.10.7 Unpooling vrstva

Jelikož je nutné zachovat stejné rozměry enkodéru i dekodéru, musí být použita inverzní operace k operaci pooling. Taková operace se nazývá unpooling. Pokud bylo v enkodéru použito několik pooling vrstev s určitými parametry, musí být v dekodéru použit stejný počet unpooling vrstev se stejnými parametry. Opět jako v případě dekonvolučních vrstev platí použití opačného pořadí.

Unpooling lze provádět dvěma způsoby, které demonstruje obrázek 2.9. První způsob využívá informaci o pozici pixelu, ze kterého byl pooling počítán. Pixel na této pozici se přepíše do výstupního obrazu a zbylé pixely v masce zůstanou nulové. Při druhém způsobu se celá oblast masky výstupního obrazu vyplní daným pixelem vstupního obrazu [24].



Obr. 2.9: Unpooling metody [14]

2.10.8 Trénování

Trénování konvolučního autoenkodéru znamená nastavení vah a biasů jednotlivých neuronů v autoenkodéru tak, aby výstup autoenkodéru byl co nejpodobnější vstupu. Odlišnost výstupu od vstupu konvolučního autoenkodéru je definována parametrem chyby sítě (loss). Funkce, která definuje tento parametr, se nazývá objektivní funkce. Lze tedy říci, že trénování autoenkodéru je minimalizace objektivní funkce [19]. Extistuje řada objektivních funkcí a nejpoužívanější z nich je: střední kvadratická chyba (mean squared error), categorical crossentropy a binary crossentropy.

Autoenkodér je trénován bez učitele a nepotřebuje tedy pojmenovaná data. Při trénování konvolučního autoenkodéru se využívá algoritmu zpětné propagace chyb (Backpropagation), který je schopný efektivně získat gradienty vah autoenkodéru. Tyto gradienty následně využívá optimalizační algoritmus, který při trénování autoenkodéru nastavuje hodnoty vah a biasů tak, aby minimilizoval objektivní funkci [19]. Tato diplomová práce používá optimalizační algoritmus Adam, který je blíže popsán ve zdroji literatury ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION [25]. Práce se blíže optimalizačními algoritmy, objektivní funkcí a algoritmem zpětné propagace chyb nezabývá.

K trénování se využívá trénovacích a validačních dat. Trénovací množina dat je sada dat, na kterých se autoenkodér učí požadovanou činnost. Nelze použít pouze trénovací data, protože autoenkodér se může přeučit. Tím je myšleno naučení autoenkodéru na jednu určitou sadu dat. Autoenkodér poté není schopen požadovanou činnost provádět na jiné sadě dat, která je odlišná od trénovací. K předejití problému přeučení se využívá validační množina dat. Ta bývá menší než množina trénovací (obvykle 10 až 20 % z trénovací množiny) a musí obsahovat jiná data než trénovací množina [19].

Učení probíhá v epochách. Epocha je opakování procesu učení. V každé epoše je autoenkodér trénován na trénovacích datech a následně je použita validační množina. Pomocí validační množiny se vyhodnocuje úspěšnost naučení autoenkodéru. V kontextu extrakce příznaků lze říci, že autoenkodér je správně naučen, pokud s vysokou úspěšností extrahuje příznaky při použití trénovací množiny dat, a zároveň pokud úspěšnost extrakce příznaků neklesá při použití validační množiny dat [19].

3 Strojové učení

Strojové učení (machine learning) je podskupina umělé inteligence, která má ve světě dosud největší vliv. Umožňuje systému „učit se“ ze souboru vstupních dat. Učením je myšleno reagovat na změny okolního prostředí změnou vnitřního stavu systému. Pomocí různých metod a algoritmů lze tedy vytvořit model, který dokáže předpovídat požadované vlastnosti pro nové pozorování [26]. Podle zpětné vazby lze strojové učení rozdělit do tří hlavních kategorií.

3.1 Učení s učitelem

Prvním a zároveň nejčastěji používaným typem strojového učení je učení s učitelem (supervised learning). Základní princip spočívá v učení, často označovaném jako trénování, daného algoritmu pomocí učitele, tedy za pomoci někoho či něčeho, co bude algoritmu pomáhat s rozpoznáním, jak má danou informaci interpretovat. Je nutno systému tedy poskytnou nejen vstupní, ale i výstupní data, která jsou od něj očekávána. Tato výstupní data může algoritmus využít ke svému učení. Nejzákladnější možností je porovnávání výstupních dat zpracovaných algoritmem s očekávanými výstupními daty. Na základě těchto rozdílů algoritmus změní výpočty (poučí se z chyb). Tento proces se opakuje až do doby, kdy rozdíl výstupních dat zpracovaných algoritmem a očekávaných výstupních dat neklesne pod nastavenou hodnotu (threshold). Data určená k trénování algoritmu se nazývají trénovací množina dat.

Matematický popis: Nechť N je trénovací množina vzorků dvojic vstup-výstup

$$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N), \tag{3.1}$$

kde x_1 až x_N jsou vstupní vzorky a y_1 až y_N jsou výstupní vzorky. Pokud každý vzorek y je generován funkcí $f = f(x)$, potom úkolem algoritmu je najít funkci h, která aproximuje pravou funkci f [27, 28].

3.2 Učení bez učitele

Dalším typem strojového učení je učení bez učitele (unsupervised learning), na které se tato práce specializuje. Při tomto typu učení má algoritmus k dispozici pouze vstupní data. Algoritmus nemá k dispozici žádná doplňující data a parametry, podle kterých by se mohl učit. To lze zapsat matematicky jako

$$(x_1, x_2, \ldots x_N). \tag{3.2}$$

Většinou budou vstupní data ve tvaru vstupních vektorů v n-rozměrném prostoru, tedy $\mathbf{x}_N$ bude vstupní N-tý n-dimenzionální vstupní vektor.

Učení bez učitele pracuje na principu shlukování, kdy hledá ve vstupních datech společné nebo velmi podobné elementy. Třídí tedy data do skupin, které mají podobné vlastnosti. Pokud má algoritmus k dispozici pouze vstupní data, je možné ho použít i pro aktivně dynamické sítě a neustále se měnící problémy, kdy může na základě každých nových vstupních dat upravit i parametry a měnit tak svůj výstup. Učení může mít tedy trvalý charakter.

Skupiny, do kterých se data třídí (shlukují), se nazývají shluky (clusters). Většině algoritmům je nutno zadat počet shluků, do kterých se mají vstupní data shlukovat. Podle těchto shluků se nazývá i nejzákladnější analýza, pomocí které se shluky vytváří, tvz. shluková analýza (cluster analysis). Shluková analýza je sice podoblast učení bez učitele, ale používá se mnohem častěji než ostatní typy analýz, proto se někdy nesprávně strojovému učení bez učitele říká shlukování (clustering) [27].

3.3 Posilované učení

Posilované strojové učení (reinforcement learning) je typem strojového učení, kterému je jako vstup předložen stavový prostor, ve kterém probíhá kompletní činnost algoritmu. Výstupem je posloupnost akcí, kterou daný algoritmus udělal, aby se dostal ze startovního stavu do stavu cílového. Algoritmus zpracovává data a dostává průběžnou odezvu, zda se jeho výstup shoduje s očekávaným výstupem či nikoli. Systém pak pokračuje v používání procesů, které vedly ke správnému výstupu a opouští či pozměňuje procesy, které produkovaly nežádoucí výstupy [27].

V praxi nejsou mantinely rozdělení přesně dodržovány a dochází k promíchání typů. Často dochází ke spojení strojového učení s učitelem a bez učitele, kdy vzniká tvz. semi-supervised learning [27]. Například nastane situace, kdy některá námi zvolená výstupní data nejsou správná (parametr, podle kterého se algoritmus učí, není správný). Potom nelze použít pouze učení s učitelem. Nejprve jsou tedy pomocí učení bez učitele zredukována výstupní data, aby zůstala pouze ta správná, a poté aplikováno učení s učitelem.

3.4 Fáze strojového učení

Strojové učení je obsáhlé téma a různé metody probíhají různými způsoby, ale lze obecně říci, že proces strojového učení probíhá v těchto fázích [29]:

1. **Příprava vstupních dat do potřebné podoby**: Jedná se o přípravnou fázi. Všechna vstupní data musí být v podobě, ve které je konkrétní algoritmus

dokáže zpracovat [29].

2. **Trénování**: Systému je poskytnuta množina trénovacích dat, na které se systém učí požadovanou činnost [29].
3. **Testování a validace**: Systém provede naučenou činnost na datech, u kterých je již předem znám výstup. Testovaní a validace se provádí z důvodu vyhodnocení úspěšnosti trénování [29].
4. **Aplikace nebo přeučování**: Shodují-li se výstupní data získaná algoritmem strojového učení s očekávanými výstupními daty, může algoritmus začít zpracovávat „ostrá“ data [29].

3.5 Problematiky řešitelné strojovým učením

Problematiky, které je schopno strojové učení řešit, lze rozdělit do několika skupin. Zde jsou nejvýznamnější z nich [29]:

1. **Klasifikace (classification)**: rozděluje vstupní množinu dat skupin. Jedná se o algoritmy založené na učení s učitelem. Touto problematikou se blíže práce zabývat nebude [28].
2. **Regrese (regression)**: předpovídá další chování systému. Algoritmy využívající regresi náleží také do skupiny učení s učitelem [29].
3. **Shlukování (clustering)**: shlukuje vstupní data do shluků na základě podobnosti nebo stejných (velice podobných) vlastností. Patří do skupiny učení bez učitele. Tato práce se blíže věnuje právě této úloze.

3.6 Shluková analýza

Shluková analýza je souhrn matematických operací, které se používají k rozkladu dat. Tento pojem jako první použil profesor R. C. Tryon v roce 1939 [30]. Jejím cílem je v dané množině vstupních dat najít podmnožiny tak, aby data v jedné podmnožině byla podobná a zároveň byla dostatečně odlišná od dat v jiné podmnožině. Podmnožiny, do kterých shluková analýza rozkládá data, se nazývají shluky.

Nechť X je množina, ve které je N objektů. Rozklad množiny X je množina disjunktních, neprázdných podmnožin C_m, které dohromady tvoří X. To lze matematicky zapsat rovnicí [31]

$$\Omega = \{C_1, C_2, \ldots C_m\}, \tag{3.3}$$

kde Ω značí rozklad a C_m jsou podmnožiny rozkladu. Potom

$$C_i \cap C_j = \emptyset, \qquad C_1 \cup C_2 \cup \ldots \cup C_m = X, \tag{3.4}$$

kde $i \neq j$. Každá množina C_i se nazývá komponentou rozkladu [31]. Shlukování je pak takový rozklad množiny X, který maximalizuje vzájemnou mezishlukovou nepodobnost nebo minimalizuje podobnost.

Shluková analýza dokáže pracovat pouze ve dvourozměrném prostoru. Při klasifikaci obrazů bude každý šedotónový obraz ve tvaru matice a každý barevný obraz bude definován třemi maticemi. Pokud nejsou obrazy předzpracovány, je nutné každý obraz převést na vektor. Pokud budou v obrazech extrahovány příznaky, je nutné všechny příznaky každého obrazu převést na vektor (jedinou výjimkou je použití BoVW modelu). V případě shlukové analýzy bývá často obraz nazýván objektem, který odpovídá jednomu řádku ve vstupní matici [30].

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1k} \\ x_{21} & x_{22} & \dots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nk} \end{bmatrix} \begin{matrix} objekt & 1 \\ objekt & 2 \\ \vdots & \\ objekt & n \end{matrix}, \tag{3.5}$$

kde X je vstupní množina dat, x_{nk} je pixel nebo vyextrahovaný příznak z obrazu, n značí počet objektů (obrazů) a k značí počet pixelů nebo počet vyextrahovaných příznaků v jednom objektu (obrazu).

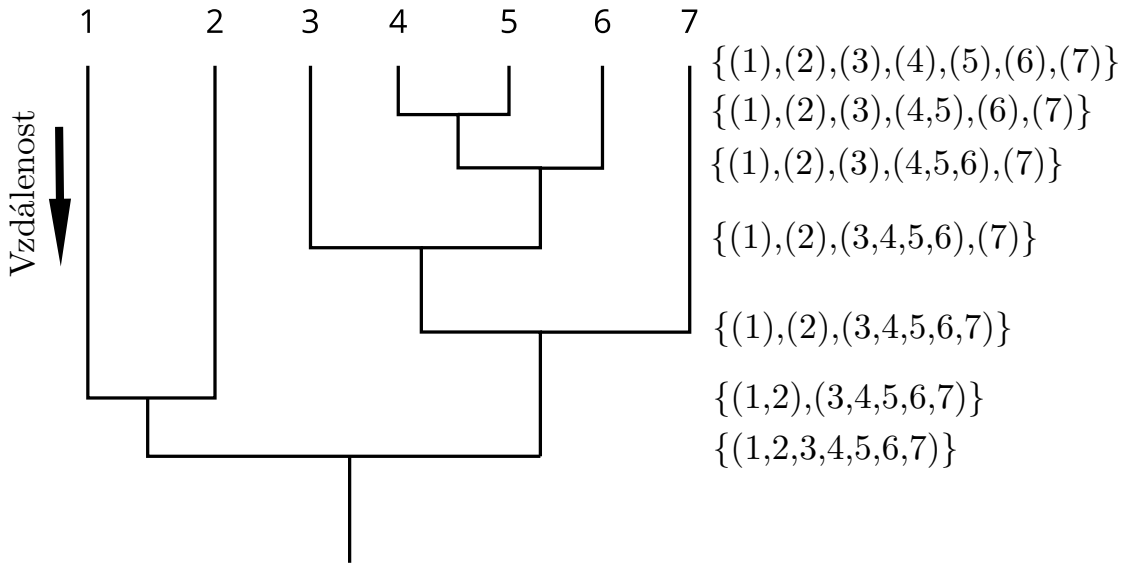
Podle cíle, ke kterému metody shlukové analýzy směřují, je lze rozdělit na hierarchické a nehierarchické [31].

3.6.1 Hierarchické metody shlukové analýzy

Hierarchické metody pracují s dříve nalezenými shluky a vytváří z nich shluky nové. Nové shluky vytváří z podmnožin, kdy průnikem každých dvou podmnožin je buď prázdná množina, nebo jedna z původních podmnožin. Jedná se tedy o sekvenci vnořených rozkladů, která na jedné straně začíná triviálním rozkladem, kdy každý objekt dané množiny objektů tvoří jednoprvkový shluk a na druhé straně končí triviálním rozkladem s jedním shlukem obsahujícím všechny objekty [31].

Hierarchické shlukování se dělí na dva přístupy. Aglomerativní přístup považuje každý objekt zpracovávané množiny za shluk, ty dále shlukuje, až vznikne shluk obsahující všechny objekty dané množiny. U divizního přístupu všechny objekty tvoří jeden shluk. Tento shluk se dále rozděluje do shluků až do doby, kdy každý shluk obsahuje pouze jeden objekt [30].

Hierarchické shlukování lze graficky znázornit pomocí dendogramu, který je zobrazen na obrázku 3.1. Dendogram je binární strom. Každý uzel tohoto stromu představuje shluk. Horizontální řezy dendrogramem jsou rozklady ze shlukovací sekvence. Vertikální směr v dendrogramu představuje „vzdálenost“ mezi shluky (rozklady) [31].



Obr. 3.1: Dendogram [30]

Nevýhodou hierarchického shlukování je jeho výpočetní náročnost, takže je použitelný pouze pro malý počet vstupních objektů. Pro n objektů vzniká $2^{n-1} - 1$ možností. Další nevýhodou je, že metody hierarchického shlukování se snaží dosáhnou nejlepšího výsledku v daný čas a neberou v potaz další postup. Také nelze vlastnosti rozkladu vylepšit, protože spojené shluky při aglomerativním principu nelze rozdělit a rozdělené shluky při divizním principu nelze znovu spojit [32].

3.6.2 Nehierarchické metody shlukové analýzy

Metody nehierarchického shlukování nevytváří hierarchickou strukturu, ale objekty jsou rozděleny do shluků, jejichž počet je obvykle předem znám. Prvotní rozklad je dále nedělitelný, pouze se upravuje tak, aby byla optimalizována vzájemná vzdálenost a odlišnost shluků. V prvotním rozkladu můžou být centroidy určeny náhodně, nebo způsobem, jakým je daný algoritmus implementován. V ideálním případě by měla být hodnota funkcionálu kvality rozkladu již v počátečním stádiu shlukování co nejlepší [31].

Velká oblast nehierarchického shlukování se zabývá nalezením optimálního počtu shluků. Tato práce se touto oblastí zabývat nebude, protože v tomto případě počet shluků určuje počet vybraných kategorií z databáze.

3.7 Shlukovací algoritmy

3.7.1 Aglomerativní shlukování

Jak bylo řečeno v kapitole 3.6.1, aglomerativní hierarchické shlukování chápe každý objekt jako shluk. Tyto prvotní shluky se následně spojují do větších shluků až do doby, kdy nezůstane jeden jediný shluk, nebo nevznikne předem zadaný počet shluků. Shluky se spojují za základě podobnosti. Metoda pracuje iterativně a v každé iteraci se spojí dva nejvíce podobné shluky do jednoho shluku. Podobnost shluků je určována podle vzdálenosti mezi shluky. Čím více jsou si shluky podobné, tím méně jsou od sebe vzdálené. Metoda tedy v každé iteraci spojí dva shluky, které jsou od sebe nejméně vzdálené [33].

Existuje více metod, jak počítat vzdálenost mezi shluky a tím určit, jak jsou si shluky podobné. Výsledek aglomerativního shlukování se bude lišit právě podle toho, jakým způsobem je vzdálenost počítána. Pro všechny metody je zapsána rovnice, podle které je počítána vzdálenost dané metody. Rovnice popisují vzdálenost mezi g-tým shlukem a sjednocením shluků h, h', přičemž $D_{g\langle h,h'\rangle}$ značí vzdálenost mezi g-tým shlukem a sjednocením shluků h, h', D_{gh} představuje vzdálenost mezi shluky g a h, $D_{gh'}$ je vzdálenost mezi shluky g a h', $D_{hh'}$ představuje vzdálenost mezi shluky h a h', n_h je počet objektů ve shluku h, $n_{h'}$ představuje počet objektů ve shluku h' a n_g je počet objektů ve shluku g.

Metoda nejbližšího souseda

Podobnost mezi dvěma shluky se vypočítá jako vzdálenost dvou nejbližších objektů, které se nachází v těchto shlucích. Je tedy vytvořen shluk ze dvou shluků, jejichž objekty měly mezi sebou nejmenší vzdálenost. Výpočet vzdálenosti metodou nejbližšího souseda [34]:

$$D_{g\langle h,h'\rangle} = \frac{1}{2}(D_{gh} + D_{gh'} - |D_{gh} - D_{gh'}|). \tag{3.6}$$

U metody nastává problém v případě, kdy existují objekty se stejnou vzdáleností od již existujících shluků. Pak může dojít k řetězení. Takže větší počet objektů může vytvořit most mezi vzdálenými shluky, které se poté nesprávně spojí do jednoho shluku [33]. Metoda nejbližšího souseda je vhodná pro neeliptické tvary a shluky různých velikostí. Výhodou je, že výsledné shluky nemají sférický charakter. Metoda je bohužel velmi citlivá na šum a odlehlé objekty [30], které jsou blíže popsány v kapitole 3.7.3.

Metoda nejvzdálenějšího souseda

Podobnost mezi dvěma shluky se vypočítá jako vzdálenost dvou nejvzdálenějších objektů, které se nachází v těchto shlucích. Je tedy vytvořen shluk ze dvou shluků, jejichž objekty měly mezi sebou největší vzdálenost [33]. Výpočet vzdálenosti metodou nejvzdálenějšího souseda [34]:

$$D_{g\langle h,h'\rangle} = \frac{1}{2}(D_{gh} + D_{gh'} - |D_{gh} - D_{gh'}|). \tag{3.7}$$

Výhodou je, že metoda nejvzdálenějšího souseda není citlivá na šum a odlehlé objekty. Nevýhodou na druhou stranu je, že tvoří sférické (kulovité) shluky a velké shluky rozděluje [30].

Metoda průměrné vazby

Podobnost dvou shluků je určena porovnáním průměrných vzdáleností všech objektů v daných dvou shlucích. Vytvoří se tedy shluk spojením dvou shluků, u kterých je aritmetický průměr vzdáleností nejmenší. Výpočet vzdálenosti metodou průměrné vazby [34]:

$$D_{g\langle h,h'\rangle} = \frac{n_h}{n_h + n_{h'}} D_{gh} + \frac{n_{h'}}{n_h + n_{h'}} D_{gh'}. \tag{3.8}$$

Nevýhodou metody je předpojatost vůči kulovitým shlukům, naopak výhodou je menší citlivost na šum a odlehlé objekty [30].

Metoda centroidní

Metoda funguje na podobném principu jako nehierarchický shlukovací algoritmus K-means, který je blíže popsán v kapitole 3.7.4. Nejprve se pomocí všech objektů ve shluku určí jeho těžiště, které je nazýváno centroid. Vzdálenost centroidu od objektů ve shluku je počítána vztahem 3.16 a poloha centoridu je určena rovnicí 3.17. Poté se podobnost shluků určuje vzdáleností centroidů jednotlivých shluků. Nejpodobnější jsou si shluky, které mají nejmenší vzdálenost centroidů [33]. Výpočet vzdálenosti centroidní metodou [34]:

$$D_{g\langle h,h'\rangle} = \frac{n_h}{n_h + n_{h'}} D_{gh} + \frac{n_{h'}}{n_h + n_{h'}} D_{gh'} - \frac{n_h n_{h'}}{n_h + n_{h'}} D_{hh'}. \tag{3.9}$$

Metoda Wardova

Metoda říká, že nejpodobnější jsou si shluky, jejichž sloučením je minimalizováno kritérium euklidovského součtu vzdáleností mezi centroidy a jednotlivými shluky. Kritérium, které musí být splněno, aby podle Wardovy metody byly shluky dostatečně podobné a spojily se, lze zapsat vztahem [34]

$$\Delta C = \frac{n_h n_{h'}}{n_h + n_{h'}} \sum_{j=1}^{p} (\mathbf{x}_{hj} - \mathbf{x}_{h'j})^2, \tag{3.10}$$

kde $\mathbf{x}_{hj}$ jsou objekty ve shluku h a $\mathbf{x}_{h'j}$ jsou objekty ve shluku h'. Toto kritérium je velmi podobné minimalizovanému kritériu součtu kvadratických chyb nehierarchickému algoritmu K-means definovanému rovnicí 3.18. Výpočet vzdálenosti Wardovou metodou [34]:

$$D_{g\langle h,h'\rangle} = \frac{(n_h + n_g)D_{gh} + (n_{h'} + n_g)D_{gh'} - n_g D_{hh'}}{n_h + n_{h'} + n_g}. \tag{3.11}$$

3.7.2 BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) je shlukovací hierarchický algoritmus, který se nejčastěji používá pro velké skupiny dat. Pracuje na hierarchickém principu, kdy ukládá Clustering Features hodnoty do hierarchického systému Clustering Feature Tree. Clustering Features představují statistické informace o podshlucích a jsou ve tvaru [30]

$$CF = (N, LS, SS), \tag{3.12}$$

kde N je počet datových bodů v podshluku, LS je součet datových bodů v podshluku (linear sum), SS je součet kvadrátů datových bodů v podshluku (square sum).

Nechť jsou v podshluku 3 datové body a jejich pozice v prostoru jsou: $(1, 1), (2, 3), (5, 4)$. Součet datových bodů je popsán rovnicí

$$LS = \sum_{i=1}^{N} \mathbf{x}_i = (1, 1) + (2, 3) + (5, 4) = (8, 8). \tag{3.13}$$

Součet kvadrátů datových bodů je definován vztahem

$$SS = \sum_{i=1}^{N} \mathbf{x}_i^2 = (1, 1)^2 + (2, 3)^2 + (5, 4)^2 = (30, 26). \tag{3.14}$$

Clustering Feature tohoto podshluku je tedy

$$CF = (3, (8, 8), (30, 26)). \tag{3.15}$$

Algoritmus nejprve vytvoří Clustering Feature Tree zařazením všech vstupních datových bodů. Uloží ho do paměti a zkoumá, jestli je překročena maximální velikost bufferu paměti. Pokud není, hierarchickým shlukováním přiřadí Clustering Features do k shluků a následně přiřadí všechny datové body nejbližším shlukům. Pokud je překročena maximální velikost bufferu paměti, algoritmus upraví uzly Clustering Feature Tree a zvýší toleranci. To opakuje až do doby, kdy není překročena maximální velikost bufferu paměti [30].

Algoritmus má dva parametry. Parametr Threshold určuje maximální vzdálenost mezi podshlukem a novým datovým bodem. Pokud je vzdálenost větší, je vytvořen nový podshluk. Parametr Branching faktor určuje maximální počet podshluků v jednom uzlu [35].

3.7.3 K-means

Nejzákladnějším algoritmem nehierarchického shlukování je K-means. I když má své nedostatky, je to nejznámější shlukovací algoritmus. Nejznámější je právě proto, že z něj spousta dalších algoritmů vychází a pomocí různých vylepšení se snaží tyto nedostatky eliminovat. V základu také přesně vystihuje podstatu optimalizačních nehierarchických metod.

Úkolem algoritmu je rozdělit vstupní soubor dat do k shluků. V tomto případě musí být počet shluků vždy předem dán. Pracuje v euklidovském prostoru, kde v prvním kroku náhodně vytvoří virtuální body $\mathbf{m}_j$ nazývané centroidy. Jejich počet je roven počtu předem určených shluků [32]. Ve druhém kroku počítá euklidovskou vzdálenost mezi datovými body $\mathbf{x}_i$ a centroidy $\mathbf{m}_j$ následujícím způsobem [36]:

$$dist(\mathbf{x}_i, \mathbf{m}_j) = \|\mathbf{x}_i - \mathbf{m}_j\| = \sqrt{(x_{i1} - m_{j1})^2 + (x_{i2} - m_{j2})^2 + \ldots + (x_{ir} - m_{jr})^2}. \tag{3.16}$$

Poté přiřadí body k centroidům, ke kterým jsou vzdálenostně nejblíže. Ve třetím kroku přepočítá polohu všech centroidů a přesune každý centroid do těžiště shluku, ve kterém se nachází. V Euklidovském prostoru je těžiště shluku počítáno jako [36]:

$$\mathbf{m}_j = \frac{1}{|C_j| \cdot \sum_{x \in C_j} \mathbf{x}_i}, \tag{3.17}$$

kde $|C_j|$ je počet datových bodů v daném shluku C_j.

Dále je opakován druhý a třetí krok do té doby, než nastane jeden z následujících případů:

1. Pouze minimální nastavené množství datových bodů se již přesouvá mezi shluky.
2. Ustálí se poloha centroidů, nebo klesne změna polohy centroidů méně, než je nastavená minimální úroveň.
3. Je splněno minimalizované kritérium součtu kvadratických chyb [36]

$$SSE = \sum_{j=1}^{k} \sum_{x \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2, \tag{3.18}$$

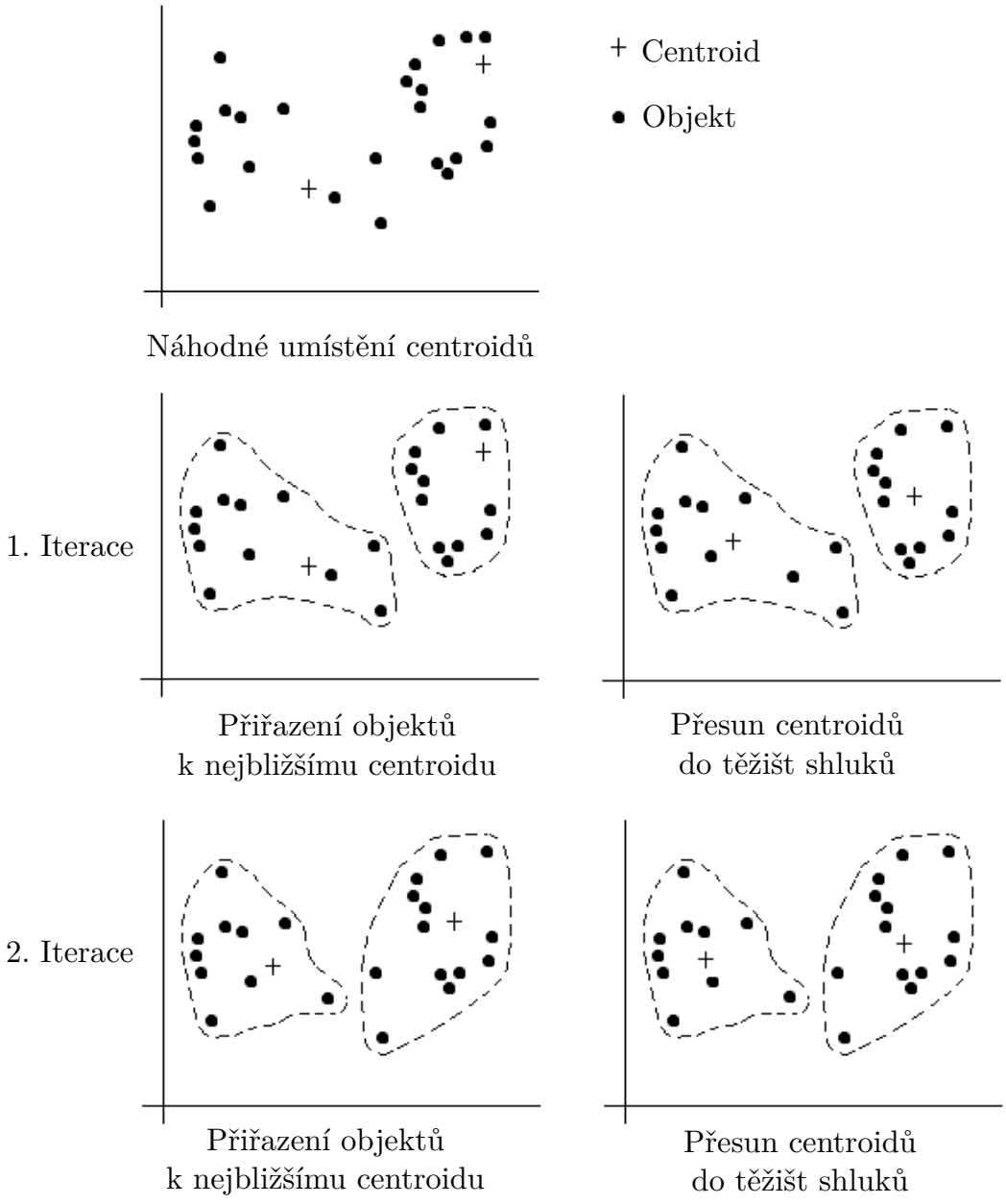
kde k je počet shluků.

Algoritmus tedy pracuje iterativně. Výstupem algoritmu K-means je rozdělení vstupních dat k shluků [32].

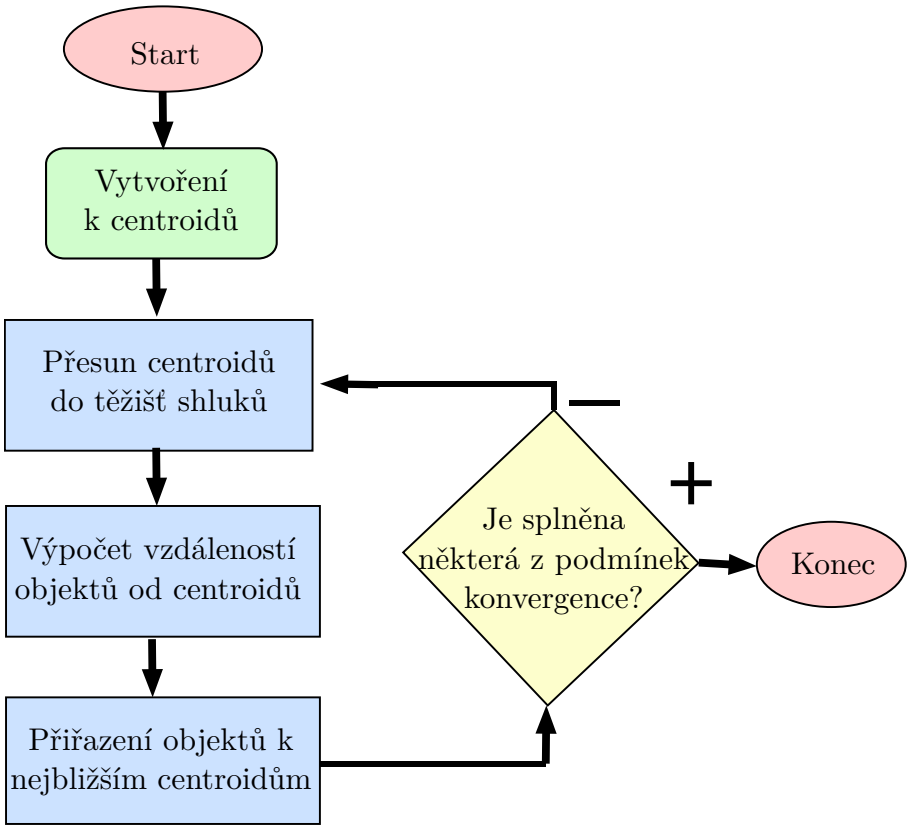
Existují dva základní typy algoritmu K-means: offline a online. Online K-means, který v roce 1967 popsal James Mac Queen [31], byl právě vysvětlen výše. Algoritmus v každé iteraci znovu počítá polohu těžiště a pozici centroidů. Offline K-means (batch K-means) byla první verze algoritmu a počítá novou polohu těžiště až po přiřazení

všech datových bodů všem shlukům [30]. Nevýhodou offline K-means je možnost vzniku prázdných shluků.

Posloupnost úkolů, které algoritmus online K-means provádí, je naznačena ve vývojovém diagramu na obrázku 3.3. Jeho princip zobrazuje obrázek 3.2.



Obr. 3.2: Princip algoritmu K-means [36]



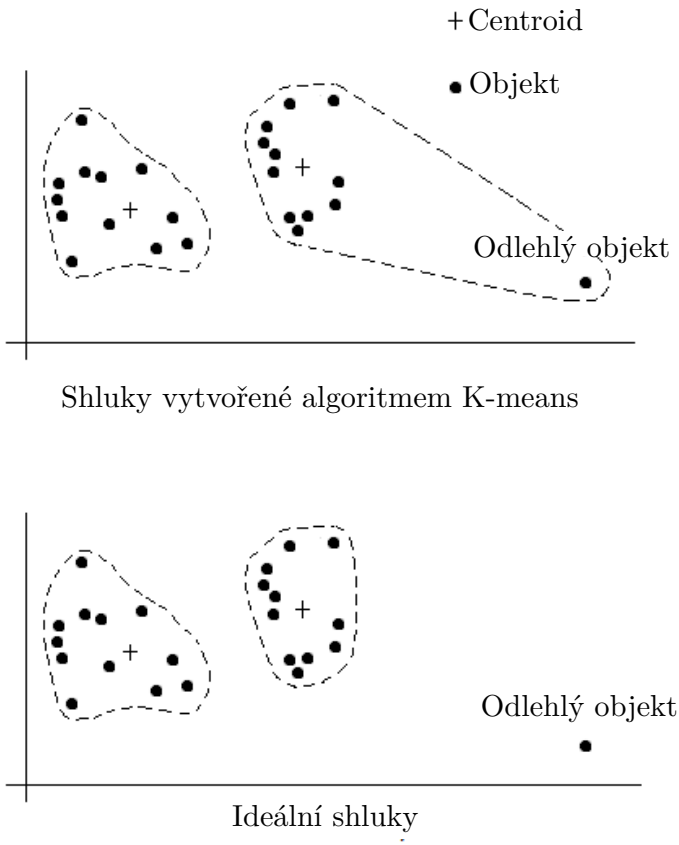
Obr. 3.3: Vývojový diagram algoritmu K-means [30]

Výhody a nevýhody K-means

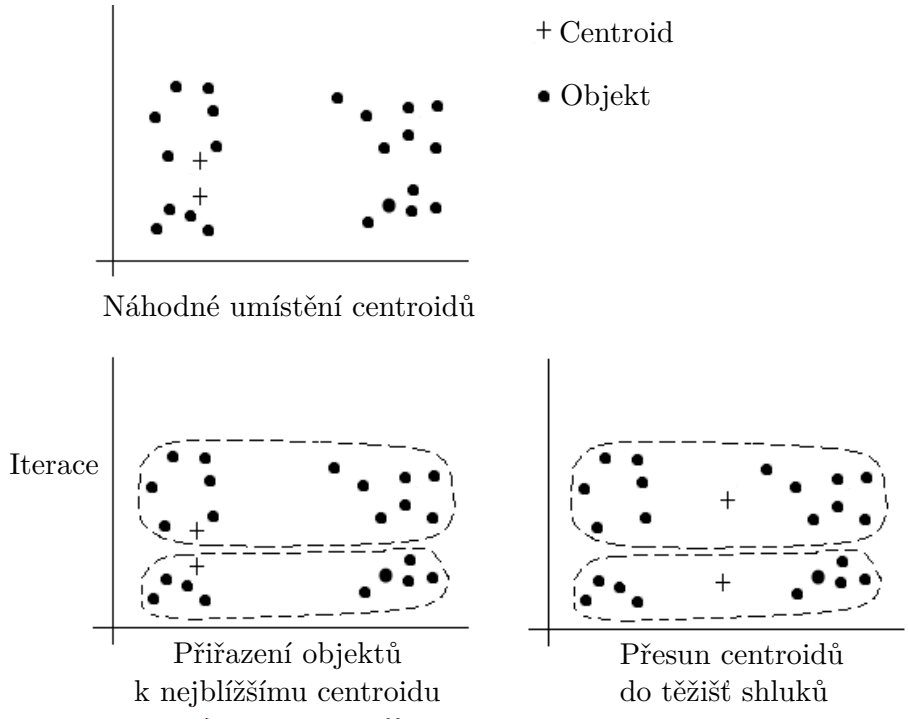
Výhodou algoritmu je jeho jednoduchost a relativně nízká výpočetní náročnost oproti ostatním metodám. Dále možnost přesouvat prvky mezi shluky. Přínosné také je, že algoritmus vždy v určitém počtu iterací dojde k alespoň částečnému řešení.

Nevýhodou je, že algoritmus K-means se velmi špatně vypořádá s odlehlými objekty (outliners), kvůli kterým dochází k nepřesnému utvoření shluků. Odlehlé objekty jsou data, která se nachází daleko od ostatních. Mohou to být chyby v načítání vstupních dat, hodnoty šumu, které nepatří do souboru vstupních dat nebo speciální objekty, které mají velice odlišné hodnoty od ostatních objektů [31]. Problematiku odlehlých objektů v algoritmu K-means demonstruje obrázek 3.4.

Další nevýhodou je skutečnost, že algoritmus vybírá prvotní pozici centroidů náhodně. To může vést ke špatnému vytvoření shluků. Špatné utvoření shluků zobrazuje obrázek 3.5. Pro různé prvnotní podmínky mohou tedy vznikou různé shluky.



Obr. 3.4: Odlehlé objekty v algoritmu K-means [36]



Obr. 3.5: Špatné utvoření shluků algoritmem K-means [36]

3.7.4 K-means++

Algoritmus K-means++ je vylepšená verze algoritmu K-means, která řeší problém náhodného umístění centroidů v prostoru. Umístí centroidy do prostoru tak, aby byly dostatečně daleko od sebe, což může vést k lepšímu utvoření shluků. Díky lepší pozici centroidů může algoritmus K-means++ konvergovat rychleji než algoritmus K-means a je zde menší šance pro vytvoření prázdných shluků. [35].

Na rozdíl od algoritmu K-means, algoritmus K-means++ nejprve vytvoří jeden cetroid $\mathbf{c}_1$ v náhodném místě prostoru X. Poté vytvoří další centroid $\mathbf{c}_i$ tím způsobem, že vybere bod $\mathbf{x}$ v prostoru X s pravděpodobností [37]

$$\frac{D(x_k)^2}{\sum \mathbf{x} \in X^{D(x_k)^2}}, \tag{3.19}$$

kde $D(x)$ značí nejkratší vzdálenost datového bodu a nejbližšího centroidu.

3.7.5 Mini Batch K-means++

Mini Batch K-means++ algoritmus je upravená varianta algoritmu K-means++, která pracuje s malými skupinami datových bodů mini-batches, díky kterým se zredukuje čas nutný k výpočtům [35]. Algoritmus nejprve náhodně použije b datových bodů ze vstupního souboru dat a vytvoří z nich malé skupiny mini-batches o stejné dané velikosti, které uloží do paměti. V dalším kroku přiřadí mini-batches k nejbližším centroidům a vytvoří shluky. Následně vybere nový náhodný bod v prostoru ze souboru vstupních dat a použije ho ke změně polohy centroidů a upravení shluků. To se provádí tak dlouho, dokud se centroidy neustálí a nedojde ke konvergenci [38].

Při použití algoritmu Mini Batch K-means dochází ke konvergenci rychleji, než v případě algoritmu K-means++, ale dochází k mírnému snížení kvality výsledků [35].

3.7.6 K-majority

Algoritmus K-majority je modifikací algoritmus K-means. Autory byl prezentován na konferenci SPIE (The International Society for Optical Engineering) v únoru 2013 [12]. Byl vytvořen z důvodu poskytnutí shlukovací metody, která je schopná shlukovat binární deskriptorové vektory algoritmu ORB popsaného v kapitole 2.5. Algortimus K-majority tedy shlukuje data definovaná binární formou do shluků. Vychází z algoritmu K-means, který shlukuje data na základě a euklidovské vzdálenosti a centroidů. Algoritmu K-means byl popsán v kapitole 3.7.4. Algoritmus K-majority je modifikován tak, že místo euklidovské vzdálenosti používá Hammingovu vzdálenost a místo centroidu používá majoritní vektor složený z majoritních čísel. Všechny ostatní kroky jsou totožné s algoritmem K-means [12].

Hammingova vzdálenost (Hamming distance) dvou řetězců značí, na kolika pozicích se mezi sebou řetězce liší. Při aplikaci na binární čísla Hammingova vzdálenost určuje, v kolika bitech se binární čísla liší. Hammingovu vzdálenost d pro dvě binární čísla i a j složená z n bitů lze definovat rovnicí [39]

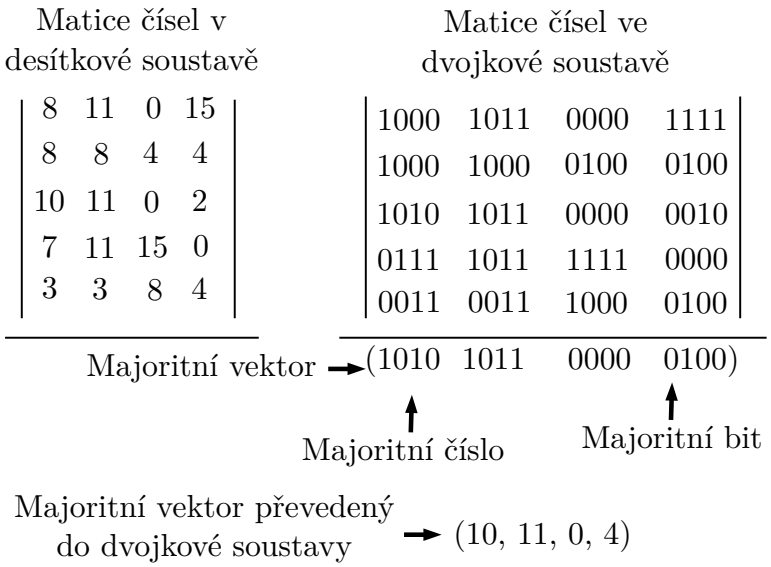
$$d(i,j) = \sum_{k=1}^{n} i_k \neq j_k, \tag{3.20}$$

kde k je index aktuálního bitu. Příklad výpočtu Hammingovy vzdálenosti pro dvě osmi bitová binární čísla je ukázán na obrázku 3.6. Obrázek také demonstruje skutečnost, že rozdíl výpočtu vzdáleností podle Hamminga a podle Euklida je velký. Je tedy neefektivní shlukovat binární čísla pomocí euklidovské vzdálenosti.

Desítková soustava	Dvojková soustava
195	11000011
15	00001111
	11001100
Euklidovská vzdálenost	Hammingova vzdálenost
d = $\sqrt{(195^2 - 15^2)}$ = 194	d = 4

Obr. 3.6: Příklad výpočtu euklidovské a Hammingovy vzdálenosti

Majoritní číslo ze skupiny binárních čísel lze zjistit tím způsobem, že se porovnává hodnota bitů na stejné pozici ze všech čísel skupiny. Pokud se v bitech na dané pozici v čísle častěji vyskytuje hodnota 1, zapíše se hodnota 1, pokud se častěji vyskytuje hodnota 0, zapíše se hodnota 0. Pro danou pozici se tedy zapíše majoritní hodnota bitů. Takto jsou zjištěny majoritní hodnoty pro všechny bity a majoritní číslo je složeno z majoritních bitů. Majoritní vektor vznikne složením majoritních čísel, které obsahuje [12]. Příklad výpočtu majoritního vektoru je na obrázku 3.7.



Obr. 3.7: Příklad výpočtu majoritního vektoru

3.7.7 Mean-shift

Dalším nehierarchickým shlukovacím algoritmem je algoritmus Mean-shift, který pracuje na principu sliding-windows, které hledají místo v prostoru s lokálně největší hustotou pravděpodobnosti datových bodů.

Algoritmus Mean-shift pracuje ve 2D prostoru, ve kterém jsou rozmístěny body. V této práci jsou datovými body jasové hodnoty pixelů jednotlivých obrazů. Vytvoří v náhodných místech sliding-windows, což jsou okna ve tvaru kruhu tak, aby všechny datové body ležely uvnitř těchto oken. Algoritmus sám velikost sliding-windows neurčuje a je nutné ji algoritmu zadat. Poté algoritmus spočítá vážený průměr ze všech datových bodů v každém sliding-window. Střed sliding-window a místo váženého průměru nejsou stejné [40]. Vzdálenost mezi středem kruhu a místa váženého průměru se nazývá mean-shift vektor.

V dalším kroku posune střed sliding-window do místa váženého průměru právě o vzdálenost mean-shift vektoru. Tuto operaci algoritmus provede pro všechny sliding-windows. Nyní je sliding-window v jiném místě ve 2D prostoru. Takže algoritmus opět spočítá vážený průměr ze všech bodů uvnitř sliding-window a posune sliding-window ze středu do místa váženého průměru o vzdálenost Mean-Shift vektoru.

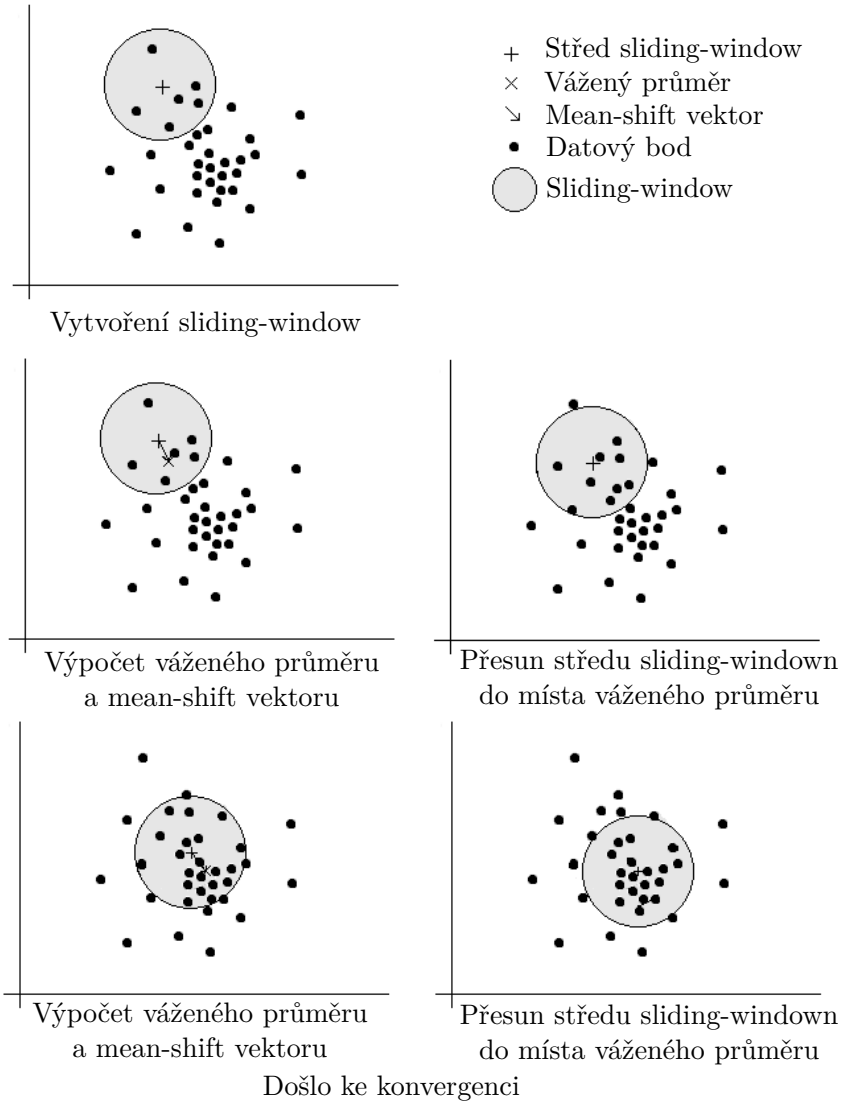
Tyto operace se stále opakují. Algoritmus tedy pracuje v iteracích, kdy v každé iteraci spočítá nový vážený průměr všech sliding-windows a všechny sliding-windows posune v prostoru o vzdálenost Mean-Shift vektoru, až do doby, kdy se sliding-windows nepohybují, nebo se pohybují méně, než je nastavená hodnota [41]. Mean-shift vektor směřuje k místu s lokálně největší hustotou pravděpodobnosti a pomocí

něj se sliding-windows do tohoto místa přesouvají. Toto místo lze chápat jako střed shluku [40].

Pro každý obrazový datový bod lze napsat rovnici mean-shift vektoru jako: [41]

$$\mathbf{y}_{i,j+1} = \frac{\sum_{k=1}^{n} \mathbf{x}_k \cdot g\left[\left\|\frac{(\mathbf{y}_{i,j}-\mathbf{x}_k)}{h}\right\|_2^2\right]}{\sum_{k=1}^{n} g\left[\left\|\frac{(\mathbf{y}_{i,j}-\mathbf{x}_k)}{h}\right\|_2^2\right]}, \tag{3.21}$$

kde $\mathbf{x}_k$ je obrazový datový bod z daného obrazu, $\mathbf{y}_{i,j}$ je mean-shift vektor, h je šířka jádra a g je odhad hustoty pravděpodobnosti. Princip hledání místa s lokálně největší hustotou pravděpodobnosti pomocí sliding-window demonstruje obrázek 3.8. Pro přehlednost je použito pouze jedno sliding-window.



Obr. 3.8: Princip algoritmu Mean-shift [40]

Jakmile dojde ke konvergenci (sliding-windows se ustálí a nachází se v místě s lokálně největší hustotou pravděpodobnosti), algoritmus utvoří shluky. Do jednoho shluku jsou přiřazeny všechny obrazové datové body, jejichž sliding-windows se přemístily ze své původní pozice do jednoho stejného místa s lokálně největší hustotou pravděpodobnosti. Je vytvořeno tolik shluků, kolik je míst s největší hustotou pravděpodobnosti [40].

Výhodou algoritmu Mean-Shift je, že algoritmus sám vyhodnotí počet shluků, takže není nutné jejich počet předem znát. Nevýhodou je skutečnost, že algortimus nedokáže sám určit optimální velikost sliding-window [40].

4 Aplikace

Výstupem praktické části této práce je aplikace, která je schopná klasifikovat databáze obrazů pomocí různých metod předzpracování obrazu a různých metod strojového učení bez učitele podle podobnosti. Aplikace je naprogramovaná v programovacím jazyce Python. Na naprogramování aplikace byl použit program Visual Studio Code [42]. V této kapitole budou popsány pouze nejdůležitější příkazy volané při implementaci aplikace. Celou tvorbu aplikace tato práce nepopisuje. Kompletní implementaci aplikace, všechny použité knihovny, moduly, funkce a příkazy lze najít ve složce „Zdrojový kód“ v příloze této diplomové práce.

Zdrojový kód aplikace je rozdělen do více souborů. Zde je výčet a stručný popis jednotlivých souborů. Podrobnější popis implementace se nachází níže v příslušných kapitolách.

- Hlavní soubor, kde se provádí většina výpočtů, je nazván „Preprocessing_and _clustering“. Jak napovídá název, v tomto souboru jsou implementovány téměř všechny metody předzpracování obrazu a téměř všechny shlukovací algoritmy. Jsou zde funkce pro načítání obrazu, změnu velikosti obrazu, převedení matice na vektor, ukládání klasifikovaných obrazů do složek a mnohé další.
- V souboru „App“ je naprogramovaná samotná aplikace. Jsou zde implementována všechna tlačítka, výběrové seznamy, textové popisy a vzhled celé aplikace.
- Dalším souborem je soubor „K_majority“, ve kterém je vlastní implementace algoritmu K-majority. Tento algoritmus je umístěn do speciálního souboru z toho důvodu, že zatím není v žádné knihovně implementován. K-majority byl tedy v této práci kompletně implementován a lze ho z tohoto souboru použít pro další vědecké studie.
- Posledními dvěma soubory jsou „Pretrained_autoencoder“ a „Pretrained_au toencoder_MNIST“, ve kterých jsou implementovány modely konvolučního autoenkodéru pro databázi barevných obrazů a databázi MNIST [43]. Modely jsou naučeny a uloženy. Při klasifikaci databáze jsou pouze načteny.

4.1 Knihovny

Aplikace využívá řadu knihoven. Nejdůležitější knihovnou pro tuto práci je knihovna `Scikit-learn` z [35]. Zde je využit modul, ve kterém jsou implementovány shlukovací algoritmy, `sklearn.cluster`. Tato knihovna pro svoje fungování vyžaduje další podpůrné knihovny jako: `NumPy` [44] (knihovna pro práci s maticemi), `SciPy` [45] (matematická knihovna založená na knihovně NumPy) a `Matplotlib` [46] (knihovna pro vykreslování grafů a obrazů). Knihovna `Scikit-image` [47] byla využita pro úpravu obrazu.

Dále je importována knihovna `Tkinter` [48], která je použita pro tvorbu tlačítek a dalších ovládacích prvků aplikace. Modul této knihovny `Tkinter.ttk` slouží k novodobějšímu vzhledu těchto ovládacích prvků a lepšímu vzhledu celé aplikace. Dále jsou použity knihovny `Queue` [49] a `Threading` [50], díky kterým lze s aplikací interagovat, když provádí výpočty předzpracování obrazu a shlukování.

K vytvoření modelu konvolučního autoenkodéru byla použita knihovna `Keras` [51] a její moduly `Model`, `Load_model`, `Layers`, `Backend`, `TensorBoard`. Knihovna `Keras` pro svoje fungování vyžaduje podpůrnou knihovnu `TensorFlow` [52]. Pro získání datasetu Mnist byl ze stejné knihovny použit modul `Mnist`. Aplikace také využívá knihovnu `OpenCV` [53], ve které jsou implementovány algoritmy pro extrakci příznaků BRISK a ORB.

4.2 Programování a popis aplikace

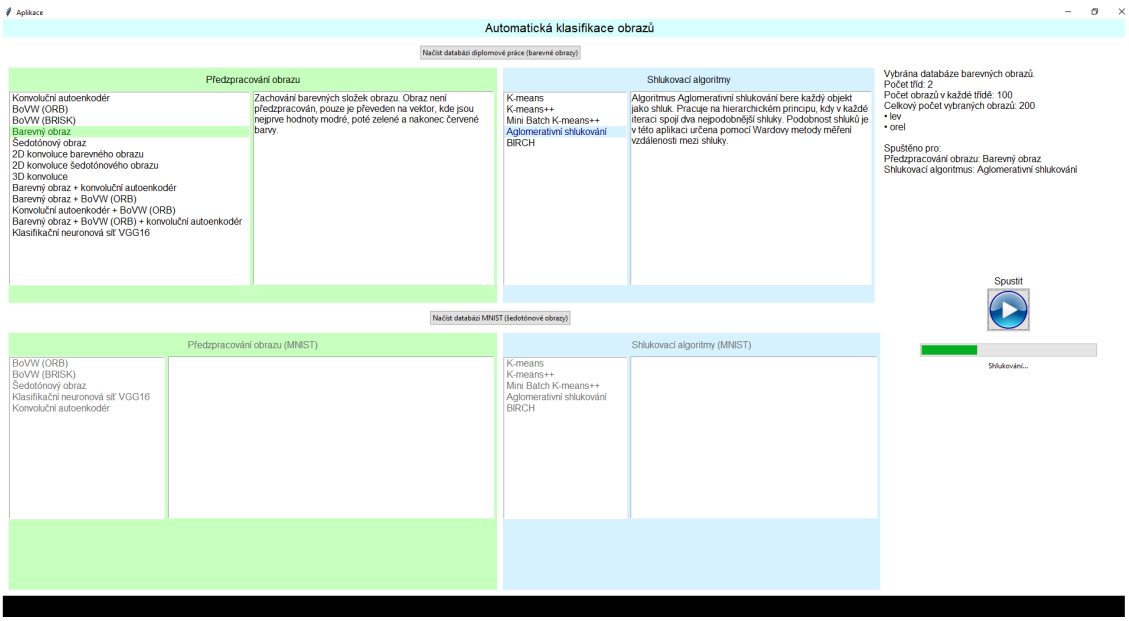
Vzhled aplikace je vytvořen pomocí knihovny `Tkinter` [48] a modulu pro modernější vzhled `Tkinter.ttk`. Okno, ve kterém se aplikace zobrazuje, je voláno příkazem `window=tk.`
`Tk()`. Nadpis aplikace je volán příkazem `heading = tk.Label`. Pro implementaci tlačítek, kterými je aplikace ovládána, slouží příkaz `button = ttk.Button`.

Pod tlačítkem načíst databázi je vytvořen velký sloupec, který je rozdělen na tři menší sloupce. Ty jsou volány příkazem `column = tk.Frame`. V prvním je umístěn výběr z více způsobů předzpracování obrazu. Ve druhém sloupci je naprogramován výběr z různých shlukovací algoritmů. Seznam pro výběr předzpracování obrazu a shlukovacího algoritmu je volán příkazem `tk.Listbox()`.

Třetí sloupec slouží ke spuštění aplikace. Tlačítko ke spuštění aplikace je vytvořeno stejně jako ostatní tlačítka, ale pro svůj vzhled používá obraz ikony volané jako: `playIcon = tk.PhotoImage(file="playIcon.png", master= playButton)`. Pod tlačítkem je umístěn ukazatel průběhu. Ten se spustí, když aplikace začne výpočet a přestane ukazovat průběh, jakmile aplikace výpočet ukončí. Ukazatel průběhu je volán příkazem `progressbar = ttk.Progressbar`. Při spuštění aplikace se pod ukazatelem průběhu zobrazuje, jakou aktuální operaci aplikace provádí.

Výpočet předzpracování a shlukování probíhá na vlákně, které běží na pozadí. Díky tomu lze s aplikací manipulovat, i když je spuštěn výpočet. Vlákno je voláno příkazem `thread = threading.Thread(target=backgroundThread, args=())`.

Většina parametrů (například: `length, mode, style, bg, font`) všech tlačítek, nadpisů a sloupců slouží k úpravě velikosti, okrajů, písma, barvy, umístění a celkovému vzhledu aplikace. Aplikace je vyobrazena na obrázku 4.1



Obr. 4.1: Aplikace

4.2.1 Spouštění aplikace

Aplikace se spouští souborem „Aplikace“. Aby byla aplikace spustitelná z tohoto jednoho souboru, je zabalena pomocí knihovny `pyinstaller` [58]. Uživatel aplikace tak nemusí instalovat Python a importovat knihovny, které aplikace využívá.

Aplikaci lze spustit přímo ze zdrojového kódu. K tomu je nutné naistalovat Python verzi 3.6.8, knihovnu `Scikit-learn` verzi 0.21.3 a Keras verzi 2.3.1. Tyto knihovny využívají pro svoje fungování další podpůrné knihovny. Návod ke kompletní instalaci knihovny `Scikit-learn` lze najít na stránce Scikit-learn [35] a návod k instalaci knihovny `Keras` na stránce Keras [51]. Jsou zde popsány i instalace podpůrných knihoven, které jsou nutné pro fungování aplikace.

Po úspěšném spuštění aplikace a dokončení výpočtů, je klasifikovaná databáze uložena do složky „vysledek“.

4.3 Implementace předzpracování obrazu

4.3.1 Načtení obrazu

Pro načítání obrazu aplikace využívá dvě knihovny. Z knihovny `OpenCV` [53] je obraz načten příkazem `cv2.imread(path, cv2.IMREAD_UNCHANGED)`, kde parametr `path` určuje cestu k obrazu a parametr `cv2.IMREAD_UNCHANGED` určuje, že se má obraz načíst nezměněn.

Z knihovny `Scikit-image` [47] je obraz načten příkazem `io.imread`. Parametrem `as_gray` je určeno, zda se má obraz načíst barevně nebo šedotónově. Pokud je zadán parametr `as_gray=False`, obraz se načte jako barevný, takže je trojrozměrný a definují ho tři matice jasových hodnot R, G a B. Pokud je zadán parametr `as_gray=True`, obraz se načte jako šedotónový. To znamená, že ještě před načtením, je vstupní barevný obraz přepočítán na šedotónový podle rovnice 2.1. Obraz je dvojrozměrný a definuje ho jedna matice stupňů šedi.

4.3.2 Změna velikosti obrazu

Pro téměř všechny metody předzracování obrazu je provedena změna velikosti všech obrazů použité databáze barevných obrazů na velikost $300 \cdot 300$ pixelů. Tím je dosažena stejná velikost všech obrazů a zároveň to usnadní výpočty a sníží výpočetní čas. Změna velikosti obrazů je provedena příkazem `transform.resize(image, size)`.

Při použití konvolučního autoenkodéru je zvětšena velikost obrazů na velikost, se kterou byl model naučen, tedy na $512 \cdot 512$ pixelů. Obrazy databáze MNIST mají velikost pouze $28 \cdot 28$ pixelů. Autoenkodér naučený na databázi MNIST byl trénován pro obrazy velikosti $32 \cdot 32$ pixelů z důvodu nastavení parametru konvolučních vrstev `padding` na `same`. Při použití konvolučního autoenkodéru pro databázi MNIST je tedy změněna velikost obrazů na $32 \cdot 32$.

4.3.3 Převedení matice na vektor

Jak bylo blíže popsáno v kapitole 2.1, v případě šedotónového obrazu je rastrový obraz definován jednou maticí jasových složek pixelů a v případě barevného obrazu třemi maticemi jasových složek RGB. Shlukovací algoritmy pracují pouze ve 2D prostoru. Je tedy nutné převést matici na vektor. K tomu aplikace využívá příkaz `np.reshape` z knihovny `NumPy` [44]. Pokud se jedná o barevný obraz, aplikace nejprve uloží do proměnných b, g, r barevné složky obrazu. Poté použitím stejného příkazu převede matici na vektor a to tím způsobem, že ve vektoru jsou nejprve zapsány hodnoty pixelů modré (proměnná b), zelené (proměnná g) a červené (proměnná r) barvy.

4.3.4 Použití diskrétní 2D konvoluce

Jako jedna z možností předzpracování obrazu před aplikací shlukovacího algoritmu je diskrétní 2D konvoluce. Ta je provedena pomocí příkazu `scipy.signal.convolve 2d(img, Kernel, mode="same")` z knihovny `SciPy` [45] podle rovnice 2.19.

Diskrétní 2D konvoluci lze provést pro šedotónový obraz, ale i pro barevný. To je implementováno tak, že aplikace uloží barevný obraz do tří proměnných b, g, r,

z nichž každá reprezentuje jasového hodnoty jedné barevné složky B,G,R. Následně provede 2D konvoluci pro každou proměnou b, g, r a výsledky sečte.

Uživatel aplikace může před shlukováním použít i diskrétní 3D konvoluci volanou příkazem `scipy.ndimage.convolve(img, edgeKernel3d)` z knihovny `SciPy`.

4.3.5 Použití algoritmu ORB

Extraktor příznaků, který byl popsán v kapitole 2.5, je importován z knihovny `OpenCV` [53]. Pro extrakci příznaků algoritmem ORB je neprve nutné extraktor vytvořit příkazem `cv2.ORB_create()`. Extraktor obsahuje několik parametrů, které jsou nastaveny defaultně na optimální úroveň. Z těchto parametrů jsou nejdůležitějšími parametry `nfeatures` a `patchSize`. Parametr `nfeatures` určuje maximální počet příznaků extrahovaných algoritmem a jejich počet je defaultně nastaven na 500. Parametr `patchSize` určuje počet binárních osmibitovými čisel, kterými je popsán každý deskriptorový vektor. Tento parametr je defaultně nastaven na hodnotu 31. Je nutné zmínit, že první bitové číslo zabírá nultou pozici, takže při výchozím nastavení parametru na hodnotu 31, obsahuje deskriptorový vektor 32 číslic.

Detekci a deskripci příznaků algoritmem ORB lze provést jedním společným příkazem `keypoints, descriptors = orb.detectAndCompute(img, None)`, kde parametr `img` obsahuje obraz, pro který je detekce a deskripce provedena a parametr `None` říká, že nemá být použita žádná maska. Obrazové přínaky jsou uloženy do proměnné `keypoints` a deskriptorové vektory jsou uloženy do proměnné `descriptors`.

4.3.6 Použití algoritmu BRISK

Algoritmus BRISK je použit obdobným způsobem jako algoritmus ORB. Nejprve je příkazem `cv2.BRISK_create()` vytvořen extraktor a následně jsou příkazem `brisk.detectAndCompute(img, None)` detekovány příznaky a vytvořeny deskriptorové vektory. Algoritmus má opět své parametry, které jsou nastaveny na optimální úroveň. Oproti algoritmu ORB obsahují deskriptorové vektory algoritmu BRISK ve výchozím nastavení 64 číslic.

4.3.7 Implementace BoVW modelu

Jako další metodu předzpracování obrazu aplikace nabízí BoVW model, který byl popsán v kapitole 5.5. BoVW model není algoritmus a není ani implementovaný v žádné knihovně. Je to matematický model, který má za úkol poskládat vyextrahované příznaky tak, aby bylo možné je lépe třídit. BoVW model využívá extrakci příznaků algoritem ORB nebo BRISK. Takto Extrahované příznaky jsou popsány binárními deskriptorovými vektory, které jsou v aplikaci při použití BoVW modelu

shlukovány algoritmem K-majority do shluků zvaných Visual Words. Algoritmus K-majority byl popsán v kapitole 3.7.6 a jeho implementace je popsána v kapitole 4.4.6. Po vytvoření Visual Words je pro každý obraz zpracován histogram, který udává četnost výskytu příznaků v každém Visual Word. Po vytvoření histogramu jsou hodnoty normalizovány. Implementace histogramu a normalizace je provedena následujícím způsobem:

```
for imagePath in imagePaths:
    histogram = [0] * histogramLength
    for j, (descriptorPath, descriptor)
            in enumerate(allDescriptorsWithPath):
        if imagePath == descriptorPath:
            index = clustering.labels_[j]
            histogram[index] += 1
    maxFreq = max(histogram)
    histogram = [freq/maxFreq for freq in histogram]
    imageData.append(histogram)
return imageData
```

Výstupem BoVW modelu jsou normalizované hodnoty histogramu, které jsou předloženy některému ze shlukovacích algoritmů ke klasifikaci. Klafikace databáze s použitím BoVW modelu byla testována pro obě metody normalizace histogramu řešené v kapitole 2.7.2. Úspěšnost klasifikace databáze dosahuje vyšší hodnoty pro normalizaci první metodou, tedy podle rovnice 2.17.

4.3.8 Model konvolučního autoenkodéru

Pro sestavení modelu konvolučního autoenkodéru využívá práce modul knihovny `keras Model`. Konvoluční a dekonvoluční vrstva je volána příkazem `Conv2D((k, (x,y), activation, padding)`. Parametr `k` určuje počet kanálů, parametr `(x,y)` určuje rozměr konvolučních jader, parametr `activation` určuje aktivační funkci a parametr `padding` určuje, jak bude počítána konvoluce při okrajovém jevu. Všechny tyto parametry byly vysvětleny v kapitolách 2.10.3, 2.10.2 a 2.8. Pooling vrstva je volána příkazem `MaxPooling2D(x, y)`, kde parametr `(x, y)` určuje velikost masky. Tento příkaz provádí operaci pooling typu max pooling. Unpooling vrstva je volána příkazem `UpSampling2D(x, y)`. Parametr `(x, y)` má stejný význam jako v případě pooling vrstvy.

Model konvolučního autoenkodéru, který byl v této práci sestaven, je složen z enkodéru a dekodéru. Enkodér obsahuje tři konvoluční a tři pooling vrstvy. Dekodér obsahuje tři dekonvoluční a tři unpooling vrstvy, které jsou inverzně uspořádany.

Přesné sestavení autoenkodéru z vrstev a příkazy s nastavením parametrů jsou následující:

1. **Vstupní vrstva**
   `Input(shape=(512, 512, 3), name='encoder_input').`
2. **Konvoluční vrstva**
   `Conv2D(16, (3, 3), activation='relu', padding='same').`
3. **Pooling vrstva**
   `MaxPooling2D((2, 2)).`
4. **Konvoluční vrstva**
   `Conv2D(8, (3, 3), activation='relu', padding='same').`
5. **Pooling vrstva**
   `MaxPooling2D((2, 2)).`
6. **Konvoluční vrstva**
   `Conv2D(8, (3, 3), activation='relu', padding='same').`
7. **Pooling vrstva** (Výstup enkodéru)
   `MaxPooling2D((2, 2), name="encoded").`
8. **Dekonvoluční vrstva** (Vstup dekodéru)
   `Conv2D(8, (3, 3), activation='relu', padding='same').`
9. **Unpooling vrstva**
   `UpSampling2D((2, 2)).`
10. **Dekonvoluční vrstva**
    `Conv2D(8, (3, 3), activation='relu', padding='same').`
11. **Unpooling vrstva**
    `UpSampling2D((2, 2)).`
12. **Dekonvoluční vrstva**
    `Conv2D(16, (3, 3), activation='relu', padding='same').`
13. **Unpooling vrstva**
    `UpSampling2D((2, 2)).`
14. **Dekonvoluční vrstva** (výstup dekodéru/autoenkodéru)
    `Conv2D(3, (3, 3), activation='sigmoid', padding='same').`

Výstup sedmé vrstvy je zároveň výstupem enkodéru. Tento výstup představuje zkomprimovaná zakódovaná data (obrazové příznaky), která jsou po převodu na vektor předávána některému ze shlukovacích algoritmů ke klasifikaci. Poslední vrstva je dekonvoluční, ale lze ji chápat spíše jako výstupní vrstvu, tedy vrstvu inverzní ke vstupní vrstvě. Jejím úkolem je zredukovat počet kanálů, aby rozměr výstupních obrazů byl roven rozměru vstupních obrazů. V této vrstvě je použita aktivační funkce sigmoid, která docílí toho, že pixely výstupních obrazů jsou normalizovány v rozmezí hodnot 0 až 1.

Při hledání nejlepšího modelu bylo testováno mnoho variant. Nejlepších výsledků

pro použitou databázi dosahuje použití pouze tří konvolučních vrstev. Při použití více konvolučních vrstev se naučení modelu mírně zhoršilo. Bylo zjištěno, že pro použitý dataset je vhodné za každou konvoluční vrstvu použít pooling vrstvu. Pokud bylo použito více konvolučních vrstev za sebou, dosahoval model horších výsledků. Také byl testován ideální počet kanálů v jednotlivých konvolučních vrstvách. Při použití velkého počtu kanálů nastává zhoršení trénování modelu.

V práci bylo testováno více objektivních funkcí i optimalizačních algoritmů. Optimalizační algoritmus (optimizer) a objektivní funkce (loss) je parametrem při kompilaci modelu, která je zapsána příkazem `autoencoder.compile(optimizer='adam', loss='binary_crossentropy')`.

Nejlepších výsledků při závěrečné klasifikaci obrazů dosahuje nastavení autoenkodéru, které používá optimalizační algoritmus Adam a objektivní funkci binary crossentropy. Tuto objektivní funkci lze použít, pokud poslední vrstva modelu autoenkodéru používá aktivační funkci sigmoid nebo jinou aktivační fuknci, která normalizuje výstupní hodnoty od 0 do 1. Tento předpoklad model složený v této práci splňuje.

Druhým nejlepším nastavením modelu autoenkodéru bylo použití optimalizačního algoritmu Adam v kombinaci s objektivní funkcí střední kvadratická odchylka. Pro tuto objektivní funkci bylo nutné změnit aktivační funkci v poslední vrstvě na ReLU.

Pro obě nastavení byl model naučen a uložen. V Aplikaci je načteno první nastavení, které je použito ke klasifikaci databáze. Uložení modelu do souboru provádí jednoduchý příkaz `autoencoder.save("Název souboru.h5")`. Načtení modelu ze souboru je provedeno podobně jednoduchým příkazem `load_model("Název souboru.h5")`. K získání výstupu enkodéru, tedy obrazových příznaků, je nutné použít výstup sedmé vrstvy. To je provedeno pomocí jeho názvu. Z toho důvodu je navíc oproti ostatním vrstvám u sedmé vrstvy přídán název. Výstup enkodéru obsahující obrazové příznaky je získán příkazem `Model(inputs=autoencoder.input, outputs=autoencoder.get_layer("encoded").output)`.

V práci bylo opakovaným učením modelu autoenkodéru zjištěno, že model se nejlépe naučí, pokud trénovací množina obsahuje co největší počet obrazů, pokud mají obrazy co největší rozlišení, a pokud je použito co největší množství epoch. Všechny tyto předpoklady mají své omezení a jsou limitovány vypočetní náročností.

Pro lepší naučení modelu byla trénovací množina obrazů rozšířena pomocí následujících transformací:

- vertikální otočení,
- horizontální otočení,
- rotace o 15° a posun ve směru vodorovné osy o 15 pixelů.

Pro implementaci těchto transformací byl importován z knihovny `keras` modul

`ImageDataGenerator`. Transformace provádí generátor transformací, který je nutné nejprve příkazem `imgGen = ImageDataGenerator()` inicializovat, a poté lze provést trasformace obrazu příkazem `imgGen.apply_transform`. Pro výše zmíněné transformace jsou použity parametry:

- `'flip_horizontal': True`
- `'flip_vertical': True`
- `'theta': 15, 'tx': 30`.

Díky tomuto rozšíření trénovací množiny byl model naučen na 2 000 trénovacích a 500 validačních obrazech s deseti epochami. Rozlišení obrazů bylo $512 \cdot 512$ pixelů.

V této diplomové práci byl vytvořen i druhý model konvolučního autoenkodéru pro databázi MNIST. Tento model byl naučen na 60 000 obrazech, tedy na celé databázi MNIST. Velikost obrazů pro trénování byla $32 \cdot 32$ pixelů. Ostatní nastavení parametrů zůstalo zachováno.

4.3.9 Klasifikační neuronová síť VGG 16

Jako další možnost předzpracování obrazu aplikace nabízí extrakci příznaků pomocí klasifikační neuronové sítě VGG 16 implementované v knihovně `Keras` [51]. Diplomová práce používá tuto síť naučenou na databázi obrazů ImageNet [54]. Tato databáze obsahuje 14 miniónů barevných obrazů a je podobná použité datábazi v této práci. Pro implementaci modelu byla vytvořena následující funkce [55]:

```
def doAutoencoder_VGG16(imagePaths, X):
    tb._SYMBOLIC_SCOPE.value = True
    model = VGG16(weights='imagenet', include_top=False,
                  input_shape=(224, 224, 3))
    print(model.summary())
    X = preprocess_input(X)
    imageData = model.predict(X)
    outputShape = imageData.shape
    imageData = np.reshape(imageData, (len(X),) +
    (outputShape[1] * outputShape[2] * outputShape[3],))
return imageData,
```

kde jako výstup modelu je použita poslední pooling vrstva a tento výstup je přímo převeden na vektor. Před použitím tohoto modelu je nutné pro všechny použité obrazy provést změnu velikosti na $224 \cdot 224$ pixelů. Nutno zmínit, že extrakce příznaků pomocí klasifikační neuronové sítě spadá do skupiny strojového učení s učitelem. Jelikož byl využit předtrénovaný model k extrakci příznaků, lze tedy chápat tuto metodu jako učení bez učitele.

4.4 Použití shlukovacích algoritmů

Všechny shlukovací algoritmy, které aplikace používá, jsou importovány z knihovny `sklern.cluster`. Většina shlukovacích algoritmů implementovaných v této knihovně má spoustu parametrů, které jsou obvykle nastaveny na defaultní optimální úroveň. Pokud je defaultní nastavení parametru vhodné pro klasifikaci databáze, nebude jej tato práce dále zmiňovat. Práce se blíže zaměří pouze na ty parametry, které jsou nastaveny nevhodně a mají vliv na výslednou klasifikaci databáze.

4.4.1 Aglomerativní shlukování

Algoritmus je volán použitím příkazu `AgglomerativeClustering(n_clusters=k).fit(imageData)`, kde parametr `n_clusters` určuje počet shluků, do kterých se mají obrazy shlukovat a parametr `imageData` jsou hodnoty pixelů jednotlivých obrazů, které má algoritmus zpracovat. Tyto dva parametry mají pro všechny shlukovací algoritmy stejný význam, takže je tato práce dále popisovat nebude.

Metoda, jakou je měřena vzdálenost mezi shluky, je v algoritmu defaultně nastavena na Wardovu. Lze ji měnit parametrem `linkage:{"ward", "complete", "average", "single"}`, ale opětovným spouštěním aplikace pro různé metody bylo ověřeno, že pro klasifikaci použité databáze je Wardova metoda nejlepší.

4.4.2 BIRCH

Algoritmus je volán příkazem `Birch(n_clusters=k).fit(imageData)`. Parametry `threshold` a `branching_faktor` jsou defaultně nastaveny na hodnoty 0.5 a 50. Opětovným spouštěním aplikace pro různé hodnoty těchto parametrů bylo ověřeno, že defaultní hodnota je nejlepší pro třídění použité databáze obrazů.

4.4.3 K-means

Algoritmus je volán použitím příkazu `KMeans(n_clusters=K, init='random').fit(imageData, sample_weights)`, kde parametr `init='random'` určuje, že prvotní umístění centroidů je náhodné a parametr `sample_weights` určuje váhu jasových hodnot jednotlivých pixelů. Váha je defaultně pro všechny jasové hodnoty pixelů stejná. Stejná váha je vhodná pro třídění použitých obrazů.

Rozdílnou váhu jasových hodnot pixelů lze využít, pokud se jeden nebo malá skupina obrazů radikálně liší od ostatních obrazů ve své třídě. Jasové hodnoty pixelů těchto obrazů jsou odlehlými objekty (jak je blíže popsáno v podkapitole 3.7.3) a negativně působí na utváření shluků. Potom dojde také k horšímu rozdělení databáze obrazů do tříd. Pokud se těmto pixelům přiřadí menší váha (například 10 %),

dojde k redukci jejich negativního vlivu na vytváření shluků a lze tak dosáhnout lepších výsledků.

4.4.4 K-means++

Algoritmus K-means++ je volán stejným příkazem jako algoritmus K-means, pouze je změněn parametr `init='random'` na `init='k-means++'`. Tato změna způsobí, že prvotní rozložení centroidů není náhodné, ale že prvotní centroidy jsou rozloženy dostatečně daleko od sebe pomocí rovnice 3.19.

4.4.5 Mini Batch K-means++

Algoritmus je volán příkazem `MiniBatchKMeans(n_clusters=k).fit(imageData, sample_weights)`. Narozdíl od algoritmu K-means++, tento algoritmus vytváří malé skupiny mini batches, které byly blíže popsány v kapitole 3.7.5.

4.4.6 Vlastní implementace K-majority

Modifikace algoritmu K-means, zvaná K-majority, která byla popsána v kapitole 3.7.6, dosud není v žádné knihovně implementována. V této diplomové práci byl tedy algoritmus K-majority naprogramován. K naprogramování byla využita implementace algoritmu K-means [56]. Algoritmus K-majority počítá vzdálenost obrazových dat pomocí Hammingovy vzdálenosti a místo centroidu využívá majoritního vektoru.

Algoritmus je využit při metodě předzpracování obrazu pomocí BoVW modelu. V tomto modelu má algoritmus za úkol shlukovat deskriptorové vektory, které popisují extrahované příznaky extraktorem příznaků ORB nebo BRISK, do skupin zvaných Visual Worlds na základě podobnosti. Z důvodu toho, že jsou deskriptorové vektory algoritmů ORB a BRISK binární, je k měření podobnosti využita Hammingova vzdálenost.

Výpočet Hammingovy vzdálenosti pro tuto problematiku je výpočetně náročná operace. Je nutné si uvědomit, že algoritmus K-majority musí v každé iteraci vypočítat vzdálenost mezi všemi destriptorovými vektory a všemi majoritními vektory. Náročnost výpočtu je ukázána na následujícím příkladu.

V aplikaci byly vybrány pro klasifikaci dvě třídy, z nichž každá obsahuje 100 obrazů. Byla vybrána metoda předzpracování BoVW modelem s extraktorem příznaků ORB s počtem extrahovaných příznaků 200 a počtem Visual Words 25. Extraktorem příznaků bylo tedy z každého obrazu extrahováno 200 příznaků. Každý deskritprový vektor extraktoru ORB definuje 32 osmi bitových čísel. Při výpočtu algoritmu K-majority je nutné v každé iteraci spočítat Hammingovu vzdálenost mezi všemi

destriptorovými vektory a majoritními vektory, tedy u každého bitu zjistit, zda se rovná nebo nerovná. Pokud se rovná, je vzdálenost 0, pokud se nerovná, je vzdálenost 1. Hammingova vzdálenost tedy určí, v kolika bitech se vektory liší. Celkový počet výpočtu pro výpočet Hammingovy vzdálenosti při takovémto nastavení je: $200 \cdot 200 \cdot 32 \cdot 8 \cdot 25 = 256000000$ (200 obrazů, 200 deskriptorových vektorů z každého obrazu, 32 osmi bitových čísel z každého destriptorového vektoru, 8 bitů v každém osmi bitovém čísle a 25 majoritní vektorů).

Výpočet Hammingovy vzdálenosti byl nejprve naprogramován v jazyce Python, ale jedna iterace algoritmu K-majority trvala přibližně 200 sekund. Pro konvergenci algoritmu bylo nutné 20 až 40 iterací. Z toho důvodu byl výpočet Hammingovy vzdálenosti převeden do jazyka C, který je schopen provést tento výpočet za desetinu času. K tomu byl použit kompilátor GCC [57], který umožní volání funkcí jazyka C v jazyce Python. Bylo také nutné definovat typy proměnných. Funkce, která byla vytvořena pro výpočet Hammingovy vzdálenosti v jazyce C, vypadá takto:

```
uint8_t getDistance(uint8_t*p1, uint8_t*p2, size_t count)
{   int distance = 0;
    for (int i = 0; i < count; i++)
    {   uint8_t x = p1[i];
        uint8_t y = p2[i];
        distance += __builtin_popcount(x ^ y);}
    return distance;}
```

Po dalších menších úpravách se podařilo sníži výpočetní čas pro jednu iteraci z výše zmíněnými parametry z přibližně 200 sekund na 24 sekund. Metoda BoVW model byla razantně zrychlena, ale čas potřebný k výpočtu je stále relativně vysoký. Dále byla vytvořena funkce v jazyce Python pro výpočet majoritního bitu:

```
def bitMode2(numbers, b):
    setBit = (1 << b)
    bitSum = 0
    for num in numbers:
        bitSum += getBit(num, b)
    return setBit if bitSum > (len(numbers) // 2*setBit)
    else 0
```

Majoritní vektor počítá v jazyce Python funkce:

```
def binaryMajority(numbers):
    result = 0
    for b in range(0, 8):
        majorityBit = bitMode2(numbers, b)
```

```
            result = result | majorityBit
    return result
```

Toto jsou tři nejdůležitější funkce pro implementaci algoritmu K-majority. Tato práce se blíže programováním algoritmu K-majority zabývat nebude. Kompletní zdrojový kód algoritmu lze najít v souborech „K_majority.py“ a „hamming.c“. Z těchto souborů lze také implementovaný algoritmus K-majority využít pro budoucí vědecké studie.

4.4.7 Mean-shift

Algoritmus je volán příkazem `MeanShift(bandwidth=bandwidth).fit(imageData)`, kde parametr `bandwidth` určuje velikost sliding-windows, které byly popsány v kapitole 3.7.7. Velikost sliding-window je citlivý parametr, který je nutné přesně určit, aby algoritmus pracoval správně.

Parametr velikosti sliding-window je odhadován pomocí funkce, která je naprogramována v knihovně `sklern.cluster` a lze ho volat příkazem `bandwidth = estimate_bandwidth(imageData, quantile=0.2, n_samples=500)`, kde parametr `quantile` představuje medián dvojic vzdáleností mezi hodnotami pixelů a jeho hodnota by se měla pohybovat mezi 0 až 1. Parametr `n_samples` určuje počet hodnot pixelů, pro které je tato funkce počítána. Pokud není tento parametr zadán, je tato funkce počítána ze všech hodnot pixelů, které algoritmus zpracovává. To ale zvyšuje čas potřebný k výpočtu funkce a přesnost odhadu je téměř konstatní.

5 Testování a výsledky práce

V této kapitole budou prezentovány výsledky úspěšnosti klasifikace databází pro výše popsané metody předzpracování obrazu a shlukovací algoritmy při použití různého počtu tříd. Pokud nebude řečeno jinak, pro algoritmy K-means, K-means++ a Mini Batch K-means++ byly vždy spuštěny tři testy. Výsledek úspěšnosti klasifikace pomocí těchto algoritmů odpovídá průměru těchto testů. Více testů bylo prováděno z toho důvodu, že tyto shlukovací algoritmy pro opětovné spuštění se stejným nastavením dosahují různých výsledků. Úspěšnost klasifikace databáze se pro opětovné spuštění se stejným nastavením lišila až o 25 %. Pokud nebude v práci uvedeno jinak, testování bylo provedeno pro databázi Open Images Dataset V5 [59], která je blíže popsána níže.

5.1 Použité databáze

V této diplomové práci byla jako hlavní použita databáze barevných obrazů. Tato databáze obsahuje 1000 obrazů rozdělených do následujících tříd:

1. orel
2. lev
3. počítačová myš
4. zebra
5. červená růže
6. klaviatura
7. česká vlajka
8. okurky
9. počítačová klávesnice
10. hnědý medvěd

V každé třídě se nachází 100 obrazů. Témeř všechny třídy (až na jednu) jsou použity z databáze Open Images Dataset V5 [59]. Obrazy ve třídě „česká vlajka“ jsou staženy z volně šířitelných obrazů na Google [60]. Spouštěním aplikace pro různé třídy bylo dokázáno, že tato třída obrazů se chová podobně jako třídy stažené z databáze Open Images Dataset V5. Pokud nebude v práci uvedeno jinak, testování bylo provedeno s touto databází.

Jako vedlejší byla použita databáze MNIST [43], obsahující 60 000 šedotónových obrazů s číslicemi 0 až 9. Databáze byla importována z knihovny `Keras` [51] z modulu `datasets`. Pro klasifikaci bylo vybráno prvních 1000 obrazů. Nutno zmínit, že při výběru prvních 1000 obrazů, nejsou všechny třídy zastoupeny stejným počtem obrazů. Různý počet obrazů v jednotlivých třídách může způsobit změnu v úspěšnosti klasifikace. Proto porovnání úspěšnosti obou databází je spíše orientační.

5.2 Počet a podobnost obrazů

Pro všechny shlukovací algoritmy bylo s použitím daného datasetu zjištěno, že procentuální úspěšnost klasifikace neklesá s rostoucím počtem obrazů z 10 na 1000 v jednotlivých třídách, ale zůstává relativně konstantní. V tabulce 5.1 je zobrazena procentuální úspěšnost klasifikace databáze při zvyšování počtu obrazů z 10 na 1000 ve třídách „orel“ a „lev“. Při tomto testování byl zachován barevný obraz a použit shlukovací algoritmus K-means.

Počet obrazů ve třídě	10	20	30	40	50	70	90	110	130	150
Úspěšnost [%]	100,00	100,00	96,70	89,00	87,10	100,00	95,00	95,00	93,30	95,00
Počet obrazů ve třídě	200	250	300	400	500	600	700	800	900	1000
Úspěšnost [%]	98,00	97,50	94,82	92,00	93,50	95,61	92,74	94,14	95,88	94,56

Tab. 5.1: Závislost úspěšnosti algoritmu K-means na počtu obrazů ve třídě

Dalším zjištěním bylo, že úspěšnost shlukovacích algoritmů při klasifikaci použité databáze závisí na podobnosti a složitosti obrazů v jedné třídě a rozdílnosti obrazů mezi jednotlivými třídami. Shlukovací algorimy dokáží klasifikovat databázi lépe, pokud jsou v jedné třídě obrazy podobné. Opakovaným spouštěním aplikace bylo dokázáno, že největší procentuální úspěšnosti dosahuje třída „orel“, protože v ní jsou nejpodobnější obrazy. Téměř všechny obrazy v této třídě mají na pozadí oblohu, tedy modrou barvu. Obvykle je uprostřed obrazu orel, který je vždy relativně podobný. Pokud budou v jedné třídě obrazy, které jsou dosti odlišné a relativně složité, nebude úspěšnost této třídy při třídění databáze příliš úspěšná. Úspěšnost roztřídění datasetu je také vyšší, pokud jsou jednotlivé třídy od sebe dostatečně odlišné.

Pro algoritmus K-means je zpracována tabulka 5.2, která zobrazuje procentuální úspěšnost klasifikace databáze při použití různých kombinací dvou tříd. Tabulka obsahuje kombinace třídy „orel“ s ostatními třídami a kombinace třídy „zebra“ s ostatními třídami. Třída „zebra“ obsahuje rozdílnější a složitější obrazy než třída „orel“. Navíc obrazy třídy „zebra“ jsou podobnější s obrazy z ostatních tříd. Z těchto důvodů dosahuje třída „zebra“ výrazně nižší úspěšnosti při klasifikaci databáze.

Název třídy	Orel	Orel	Orel	Orel	Orel	Orel	Orel	Orel	Orel
Název třídy	Lev	P. myš	Zebra	Č. růže	Klaviatura	Č. vlajka	Okurky	P. klávesnice	H. medvěd
Úspěšnost [%]	93,50	71,50	87,00	83,00	81,00	68,00	85,50	81,00	91,00
Název třídy	Zebra	Zebra	Zebra	Zebra	Zebra	Zebra	Zebra	Zebra	Zebra
Název třídy	Orel	Lev	P. myš	Č. růže	Klaviatura	Č. vlajka	Okurky	P. klávesnice	H. medvěd
Úspěšnost [%]	87,00	54,00	82,50	53,50	55,50	80,00	52,50	51,00	51,00

Tab. 5.2: Závislost úspěšnosti algoritmu K-means na kombinaci tříd

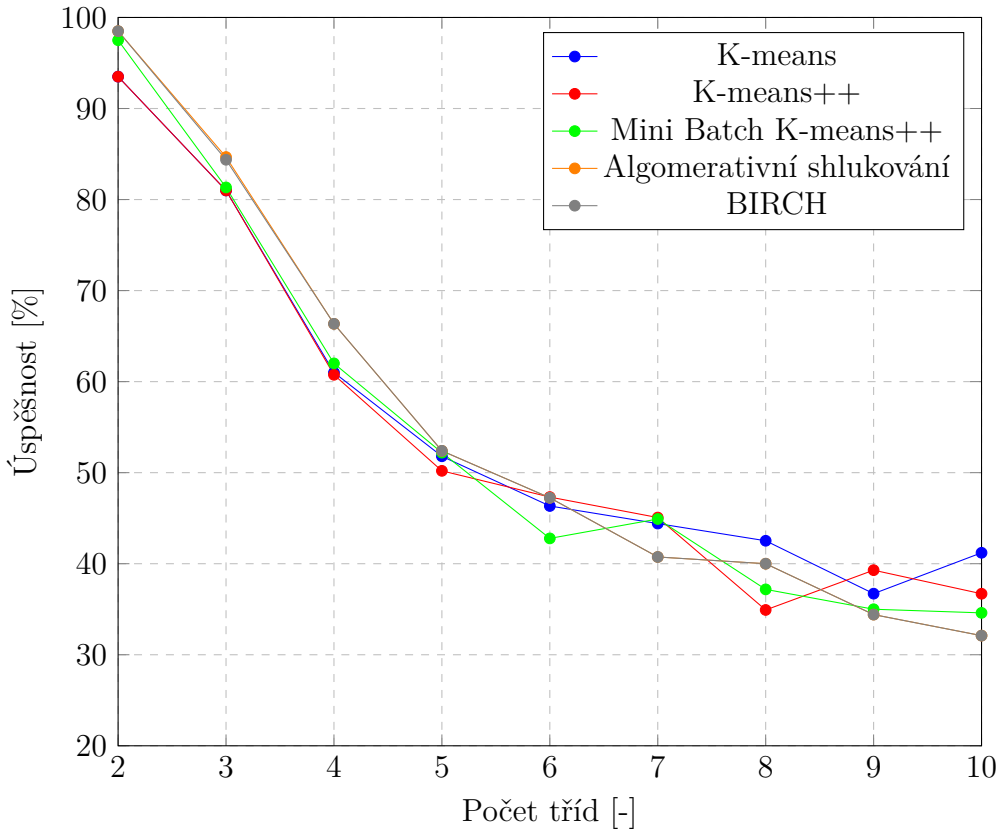
5.3 Zachování barevného obrazu

Pro všechny shlukovací algoritmy řešené v této práci bylo zjištěno, že největší vliv na úspěšnost klasifikace má počet tříd. Při všech metodách předzpracování obrazu dosahuje klasifikace databáze pro menší počet tříd mnohem vyšší úspěšnosti. Při zvyšování počtu tříd klesá úspěšnost zatřídění databáze.

Pro zachování barevného obrazu, čili pouze převedení barevných matic na vektor, dosahuje úspěšnost pro dvě třídy až 98,5 %. Při zvyšování počtu tříd od dvou do pěti, procentuální úspěšnost prudce klesá z 98, 5% na 50%. Pokud je dále zvyšován počet tříd od pěti do desíti, procentuální úspěšnost již klesá jen mírně, a to z 53% na 30, 8%. Toto tvrzení dokazuje tabulka 5.3.

Barevný obraz	2 třídy	3 třídy	4 třídy	5 tříd	6 tříd	7 tříd	8 tříd	9 tříd	10 tříd
K-means	93,50	81,00	61,00	51,80	46,34	44,41	42,52	36,71	41,20
K-means++	93,50	81,00	60,75	50,20	47,32	45,07	34,92	39,29	36,70
Mini Batch K-means++	97,50	81,33	62,00	52,20	42,78	44,91	37,18	35,00	34,60
Aglomerativní shlukování	98,50	79,67	66,25	53,20	44,82	41,14	40,90	35,00	30,80
BIRCH	98,50	84,67	66,35	52,40	47,24	40,74	40,00	34,41	32,10

Tab. 5.3: Úspěšnost klasifikace databáze bez předzpracování



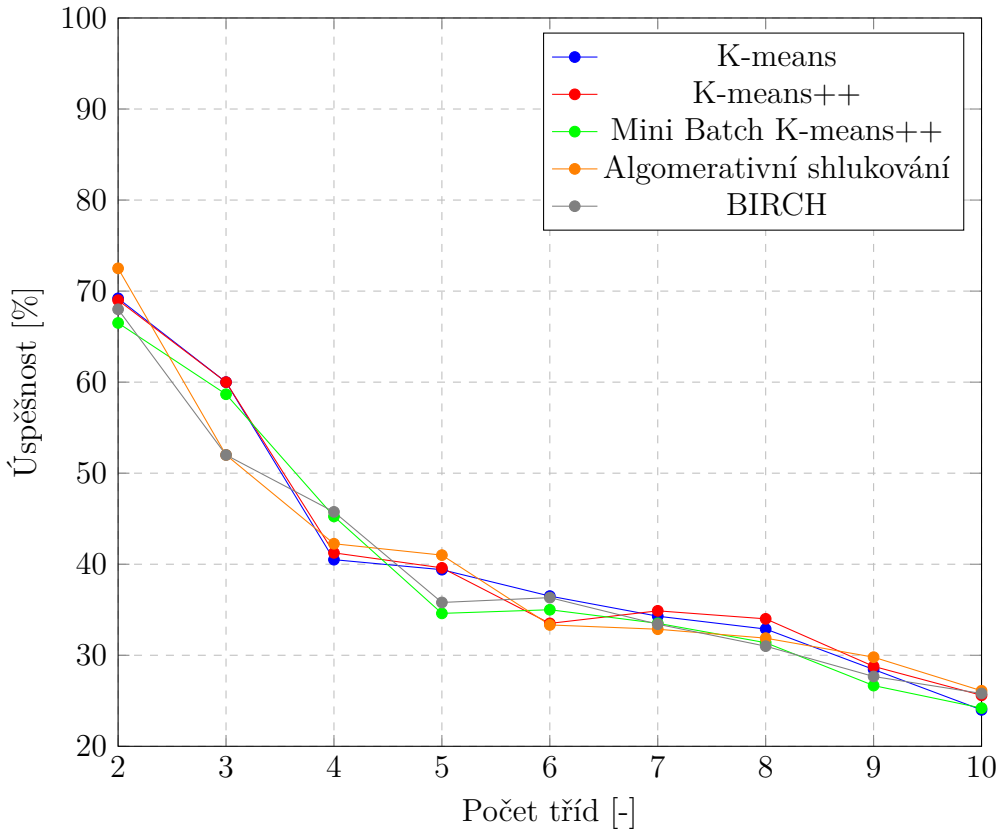
Graf 5.1: Úspěšnost klasifikace databáze bez předzpracování

5.4 Převod na šedotónový obraz

Při převodu barevného obrazu na šedotónový se sníží úspěšnost klasifikace databáze o 5 až 25 %. Důvodem nižší úspěšnosti je ztráta barevné informace obrazů. Shlukovací algoritmy tak ztratí část důležité informace pro klasifikaci. Při převodu na šedotónový obraz se úspěšnost klasifikace použitého datasetu pomocí shlukovacích algoritmů pro zvyšující počet tříd pohybuje od 72,5 % do 24 %. Úspěšnost klasifikace při převodu barevného obrazu na šedotónový dokládá tabulka 5.4.

Šedotónový obraz	2 třídy	3 třídy	4 třídy	5 tříd	6 tříd	7 tříd	8 tříd	9 tříd	10 tříd
K-means	69,50	60,00	40,50	39,40	36,50	34,29	32,88	28,44	24,00
K-means++	69,00	60,00	41,25	39,60	33,50	34,88	34,00	28,78	25,60
Mini Batch K-means++	66,50	58,67	45,25	34,60	35,00	33,59	31,38	26,67	24,20
Aglomerativní shlukování	72,50	52,00	42,25	41,00	33,33	32,86	31,88	29,79	26,10
BIRCH	68,00	52,00	45,75	35,80	36,33	33,41	31,00	27,67	25,80

Tab. 5.4: Úspěšnost klasifikace při převodu na šedotónový obraz



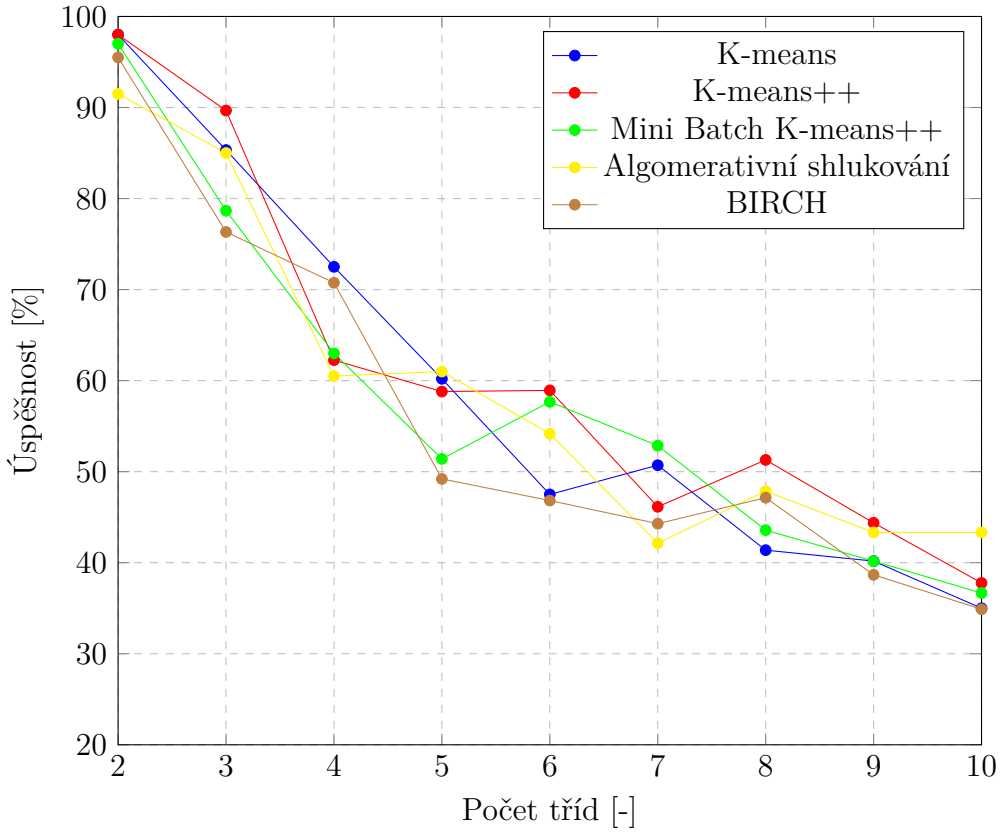
Graf 5.2: Úspěšnost klasifikace při převodu na šedotónový obraz

5.5 BoVW model

Při předzpracování obrazů pomocí BoVW se zvýší úspěšnost klasifikace o 0 až 15 %. Zvýšení úspěšnosti je tedy relativně malé. Pokud by BoVW model neměl tak vysoké výpočetní nároky a rychlost výpočtu by byla srovnatelná se zachováním barevného obrazu, tak by se tato metoda předzpracování dala považovat za vhodnou. Bohužel úspěšnost klasifikace se zvýší o příliš malou hodnotu, za cenu razantního zvýšení výpočetních nároků. Diplomová práce tedy považuje použití BoVW modelu pro použitou databázi za nevhodné. Úspěšnost klasifikace při použití BoVW modelu s extraktorem ORB je zobrazena v tabulce 5.5.

BoVW model	2 třídy	3 třídy	4 třídy	5 tříd	6 tříd	7 tříd	8 tříd	9 tříd	10 tříd
K-means	98,00	85,33	72,50	60,20	47,50	50,71	41,38	40,18	35,00
K-means++	98,00	89,67	62,25	58,80	58,93	46,14	51,14	44,38	37,78
Mini Batch K-means++	97,00	78,67	63,00	51,40	57,67	52,86	43,58	40,17	36,67
Aglomerativní shlukování	91,50	85,00	60,50	51,00	54,17	42,14	47,98	43,33	43,33
BIRCH	95,50	76,33	70,75	49,20	46,83	44,29	47,14	38,67	34,89

Tab. 5.5: Úspěšnost klasifikace při předzpracování BoVW modelem



Graf 5.3: Úspěšnost klasifikace při předzpracování BoVW modelem

5.5.1 Nastavení BoVW modelu a extraktoru ORB

Aplikace této diplomové práce umožňuje pro předzpracování BoVW modelem použít příznaky extrahované algoritmy ORB a BRISK. Opětovným spouštěním aplikace pro různé shlukovací algoritmy a různý počet tříd bylo otestováno, že metoda předzpracování BoVW modelem dosahuje průměrně o 10 % lepších výsledků s použitím extraktoru ORB než extraktoru BRISK.

Dále bylo v aplikaci testováno ideální nastavení počtu extrahovaných příznaků algoritmem ORB a ideální nastavení počtu Visual Words pro tvorbu histogramu v BoVW modelu. Největší úspěšnost klasifikace použité databáze dosahuje nastavení počtu extrahovaných příznaků extraktorem ORB na 500 v kombinaci s nastavením počtu Visual Words na 100. Toto tvrzení dokládá tabulka 5.6.

Počet extrahovaných příznaků	200	500	500	500	500	500	1000
Počet Visual Words v histogramu	25	25	50	100	150	200	200
Úspěšnost [%]	89,00	86,50	94,50	98,00	97,50	96,00	97,50

Tab. 5.6: Kombinace počtu extrahovaných příznaků a počtu Visual Words pro použité třídy „orel“ a „lev“

5.6 Diskrétní 2D a 3D konvoluce

Diplomová práce testuje použití diskrétní 2D a 3D konvoluce jako nejjednodušího extraktoru příznaků. Diskrétní 2D konvoluce barevného obrazu vykazuje průměrně pro všechny třídy o 10 % nižší úspěšnost než zachování barevného obrazu. 2D konvoluce šedotónového obrazu dosahuje nízké úspěšnosti, která je srovnatelná s předzpracováním převodu na šedotónový obraz. Metody převodu barevného obrazu na šedotónový a diskrétní 2D konvoluce šedotónového obrazu vykazují tedy nejnižší úspěšnost zatřídění použitého datasetu.

Diskrétní 3D konvoluce dosahuje nízké úspěšnosti pro malý počet tříd, ale od šesté třídy se úspěšnost ustálí na 40 %. Použití diskrétních konvolucí jako metodu předzpracování diplomová práce nedoporučuje použít, protože úspěšnost klasifikace dané databáze se sníží. Nízkou úspěšnost klasifikace dokládá tabulka 5.7.

Algoritmus K-means++	2 třídy	3 třídy	4 třídy	5 tříd	6 tříd	7 tříd	8 tříd	9 tříd	10 tříd
2D konvoluce barevného obrazu	89,50	78,33	54,63	42,80	38,67	30,43	26,00	30,00	22,10
2D konvoluce šedotónového obrazu	69,50	66,00	46,75	39,40	33,67	28,86	23,63	22,56	20,00
3D konvoluce	79,00	62,67	53,25	45,00	40,33	40,43	40,00	39,67	39,85

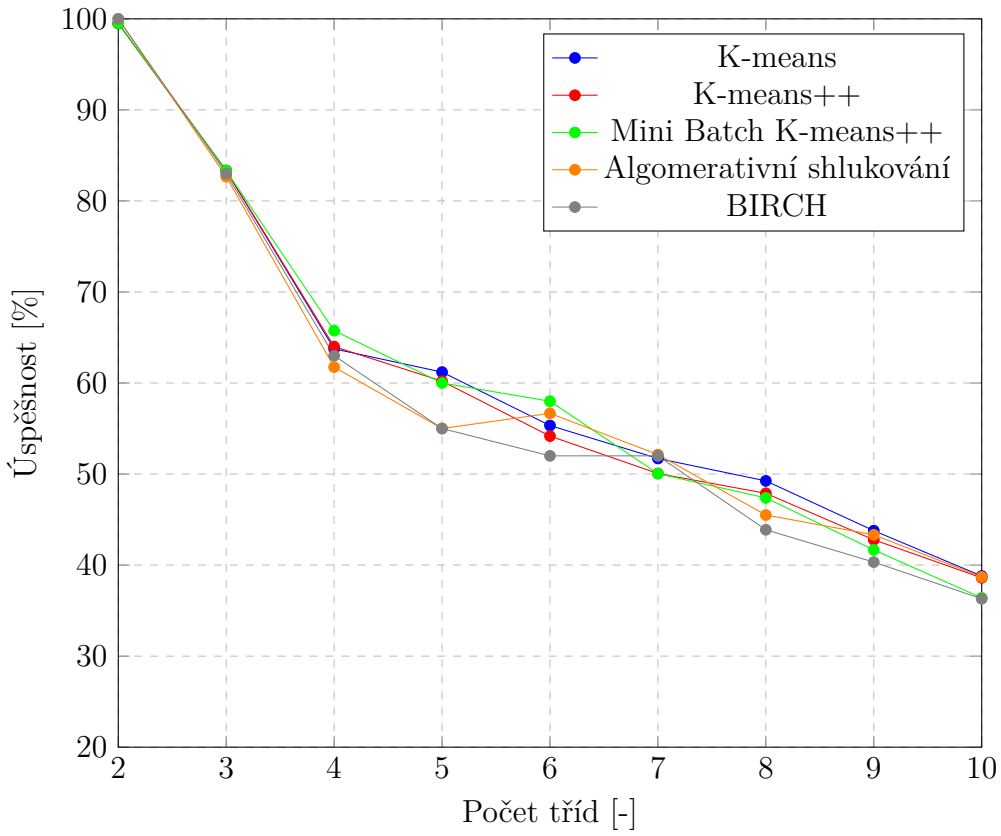
Tab. 5.7: Úspěšnost klasifikace algoritmem K-means++ při použití konvolucí

5.7 Konvoluční autoenkodér

Při předzpracování obrazu pomocí konvolučního autoenkodéru dosahuje úspěšnost klasifikace o 0 až 5 % výšší hodnoty než bez předzpracování. Výhodou této metody oproti BoVW modelu je možnost předtrénování konvolučního autoenkodéru, kdy při použití lze model pouze načíst. Díky tomu jsou výpočetní nároky ve srovnání s BoVW modelem velmi nízké. Model autoenkodéru je jediná metoda předzpracování obrazu pomocí učení bez učitele, která je schopná roztřídit dvě třídy použitého datasetu se 100 % úspěšností. Diplomová práce dochází k závěru, že kvůli pouze nízkému navýšení úspěšnosti není tato metoda předzpracování pro danou databázi příliš vhodná. Výsledky testování ukazuje tabulka 5.8.

Autoenkodér	2 třídy	3 třídy	4 třídy	5 tříd	6 tříd	7 tříd	8 tříd	9 tříd	10 tříd
K-means	99,50	83,33	65,75	61,20	55,33	51,71	49,25	43,78	38,80
K-means++	99,50	86,33	64,00	60,20	54,17	50,06	47,88	42,78	38,60
Mini Batch K-means++	99,50	81,33	65,75	60,00	58,00	50,04	47,38	41,67	36,40
Aglomerativní shlukování	100,00	82,67	61,75	55,00	56,67	52,14	45,50	43,33	38,70
BIRCH	100,00	83,00	63,00	55,00	52,00	52,00	43,88	40,33	36,30

Tab. 5.8: Úspěšnost klasifikace při předzpracování konvolučním autoenkodérem



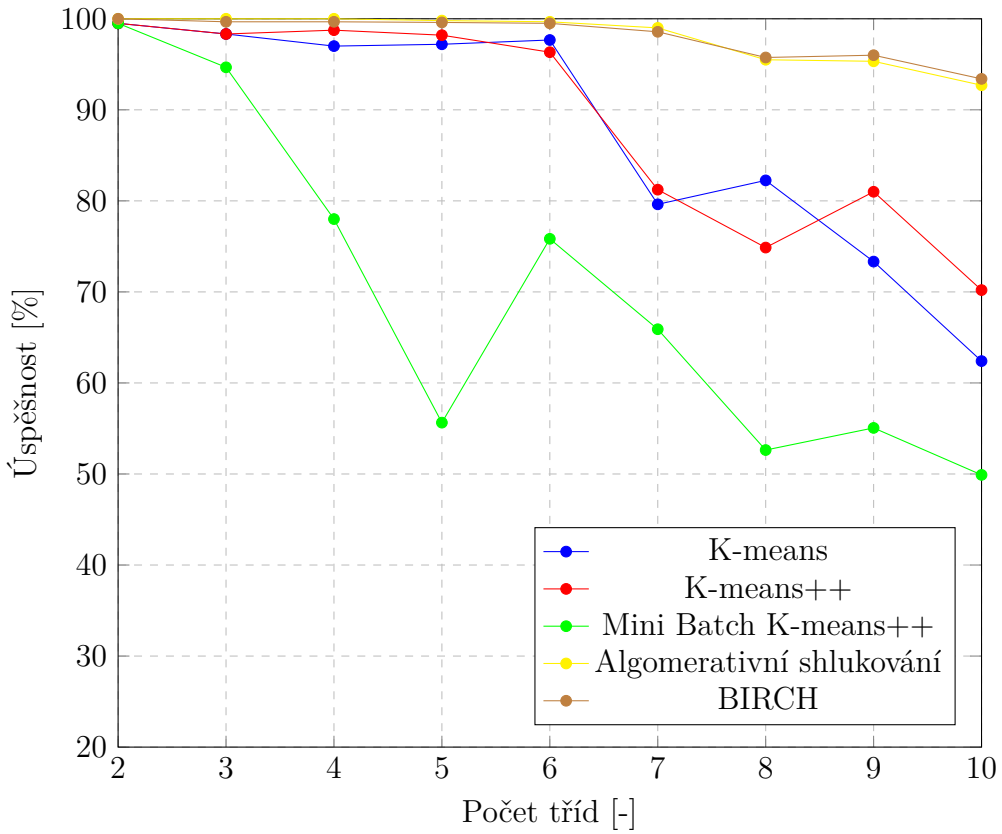
Graf 5.4: Úspěšnost klasifikace při předzpracování konvolučním autoenkodérem

5.8 Klasifikační neuronová síť VGG 16

Při předzpracování obrazu klasifikační neuronovu sítí VGG 16 [55] dosahuje klasifikace nejlepších výsledků. Pro algoritmy Aglometrativní shlukování a BIRCH dosahuje klasifikace výborné úspěšnosti 100 až 92,7 %. Při použití algoritmů K-means a K-means++ je klasifikace mírně horší, od 99,5 do 62,4 %. Nejhorších výsledků dosahuje klasifikace při použití algoritmu Mini Batch K-means++. Klasifikační neuronová síť VGG 16 sice spadá do skupiny strojového učení s učitelem, ale jelikož byl využit předtrénovaný model z internetu, dá se považovat aplikace této techniky za učení bez učitele. Úspěšnost klasifikace použité databáze pro toto předzpracování obrazu udává tabulka 5.9.

Síť VGG 16	2 třídy	3 třídy	4 třídy	5 tříd	6 tříd	7 tříd	8 tříd	9 tříd	10 tříd
K-means	99,50	98,33	97,00	97,20	97,67	79,62	82,25	73,33	62,40
K-means++	99,50	98,33	98,75	98,20	95,33	81,23	74,87	81,00	70,20
Mini Batch K-means++	91,50	94,67	78,00	55,64	75,83	65,83	52,63	55,06	49,90
Aglomerativní shlukování	100,00	100,00	100,00	99,80	99,67	99,00	95,50	95,33	92,70
BIRCH	100,00	99,67	99,67	99,60	99,50	99,57	95,75	96,00	93,40

Tab. 5.9: Úspěšnost klasifikace při předzpracování sítí VGG 16



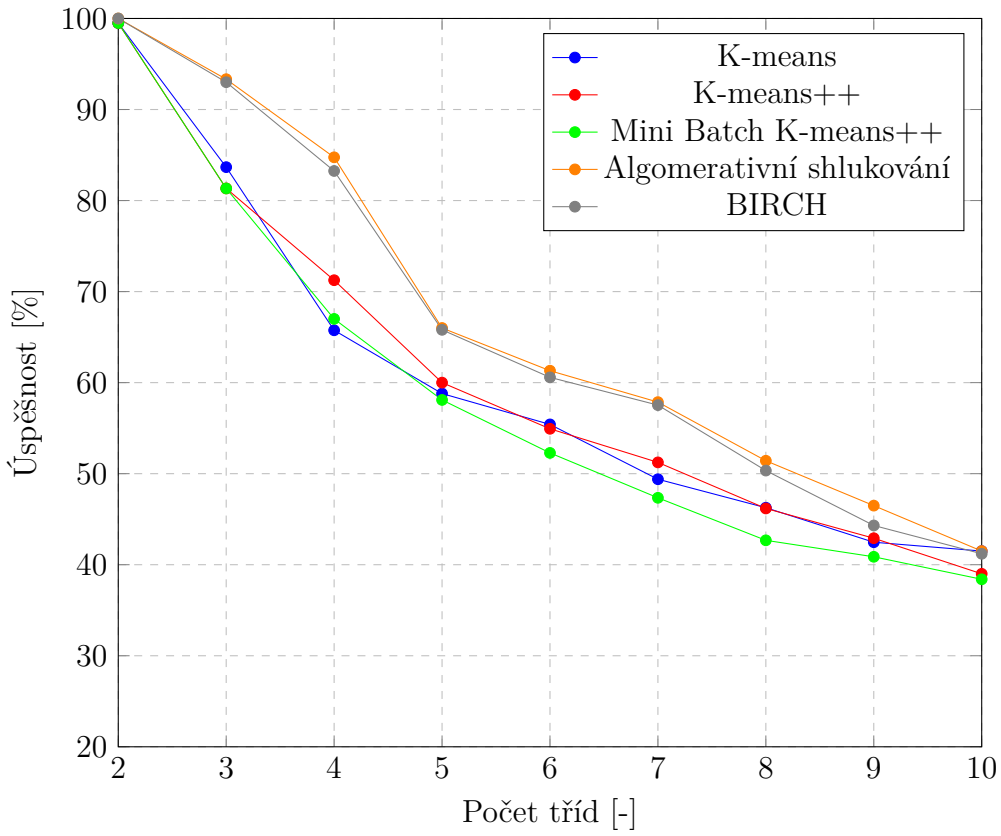
Graf 5.5: Úspěšnost klasifikace při předzpracování sítí VGG 16

5.9 Kombinace metod předzpracování

Tato práce dále testuje kombinaci použitých metod předzpracování obrazu. Kombinace metod není příliš účinná, protože přináší jen mírné zvýšení úspěšnosti zatřídění použitého datasetu. Z testovaných kombinací předzpracování obrazu je nejlepší kombinace BoVW modelu a konvolučního autoenkodéru. Tato kombinace přináší zvýšení úspěšnosti klasifikace databáze o 0 až 10 % v porovnání s použitím pouze jedné z těchto metod. Výsledky testování této kombinace zobrazuje tabulka 5.10. Všechny kombinace přezdpracování obrazu zvyšují výpočetní nároky, z důvodu nutnosti výpočtů více metod předzracování.

BoVW a autoenkodér	2 třídy	3 třídy	4 třídy	5 tříd	6 tříd	7 tříd	8 tříd	9 tříd	10 tříd
K-means	99,50	83,67	65,75	58,80	55,41	49,39	46,26	42,47	41,50
K-means++	99,50	86,67	71,25	60,00	54,93	51,24	46,18	42,91	39,00
Mini Batch K-means++	99,50	81,33	67,00	58,10	52,28	47,26	42,68	40,87	38,40
Aglomerativní shlukování	100,00	93,33	84,75	66,00	61,31	57,86	51,42	46,49	41,50
BIRCH	100,00	93,00	83,25	65,50	60,58	57,53	50,34	44,31	41,20

Tab. 5.10: Úspěšnost klasifikace při použití BoVW modelu a autoenkodéru



Graf 5.6: Úspěšnost klasifikace při použití BoVW modelu a autoenkodéru

Problémem při kombinaci metod předzpracování je, že různé metody jsou definovány jiným počtem obrazových dat. Z toho důvodu může při kombinaci metoda, která je definována mnohem větším počtem dat, převyšovat ostatní metody. Klasifikace je pak provedena především podle této metody a ostatní metody jsou téměř potlačeny.

Tato práce řeší vyvážení kombinace metod parametrem `sample_weights` u algoritmů K-means, K-means++ a Mini Batch K-means++. Tímto parametrem lze pro metody, které jsou definovány větším počtem obrazových dat, použít menší váhu a naopak pro metody, které obsahují menší počet dat, použít váhu větší. Tímto je v diplomové práci docíleno vyrovnání metod předzpracování při jejich kombinaci.

5.10 Testování databáze MNIST

Diplomová práce testuje použité metody předzpracování i pro databázi MNIST [43]. Tento dataset obsahuje šedotónové obrazy. Shlukováním bez předzpracování je myšleno zachování šedotónových obrazů a následný převod na vektor. Ostatní metody předzpracování jsou prováděny také pro šedotónové obrazy. Dataset MNIST obsahuje jednodušší obrazy než hlavní dataset barevných obrazů. Z toho důvodu by také úspěšnost klasifikace měla dosahovat vyšších hodnot. Tento předpoklad byl pro různé metody předzpracování potvrzen a dokládá ho tabulka 5.11.

Úspěšnost klasifikace při předzpracování sítí VGG 16 [55] je pro databázi MNIST mnohem nižší než pro databázi barevných obrazů. Důvodem je, že klasifikační síť je naučena na databázi ImageNet [54], která obsahuje odlišné obrazy než databáze MNIST. Tato síť je pro dataset MNIST nevhodně naučena. Diplomová práce tedy potvrzuje předpoklad, že lepší naučení sítě zlepší klasifikaci databáze.

Při zachování šedotónového obrazu nedokázal algoritmus BIRCH klasifikovat databázi MNIST. Pro opětovné spouštění vždy utvořil jen jeden shluk, který obsahoval všechny obrazy. Tato diplomová práce se domnívá, že důvodem je nedostatek obrazových dat, protože obrazy datábaze MNIST mají menší rozměr, jsou definovány jen jednou maticí a algoritmus BIRCH je navržen pro velké skupiny dat.

MNIST - 10 tříd	Šedotónový obraz	BoVW model	Síť VGG 16	Autoenkodér
K-means	49,10	38,10	50,10	49,40
K-means++	57,50	37,20	48,50	50,70
Mini Batch K-means++	41,20	36,90	44,20	43,60
Aglomerativní shlukování	55,30	36,50	61,30	51,30
BIRCH	-	38,00	52,50	45,40

Tab. 5.11: Úspěšnost klasifikace databáze MNIST

5.11 Výpočetní nároky metod přezdpracování obrazu

V diplomové práci byla sestavena tabulka 5.12, ve které je uveden výpočetní čas nutný k zatřídění kompletní databáze barevných obrazů při použití různých metod předzpracování obrazu. Pro toto testování byl vybrán shlukovací algoritmu Aglomerativní shlukování. Testování proběhlo na počítači s těmito parametry: procesor: Inter(R) Core(TM) i7-8700 CPU @ 3.20GHz, RAM paměť: 16 GB, typ systému: 64bitový operační systém. Z tabulky 5.12 je vidět, že BoVW model má mnohem vyšší výpočetní nároky, než ostatní metody předzpracování obrazu. Pro větší datasety je v této podobě a nastavení prakticky nepoužitelný.

Metoda předzpracování obrazu	Barevný obraz	Šedotónový obraz	BoWV model
Výpočetní čas [s]	158	53	7182
Metoda předzpracování obrazu	2D konvoluce	Konvoluční autoenkodér	Síť VGG 16
Výpočetní čas [s]	151	102	93

Tab. 5.12: Výpočetní nároky metod přezdracování obrazu

5.12 Zhodnocení shlukovacích algoritmů

Algoritmus Mean-shift ve výsledcích testování není uveden. Utváří totiž nesprávný počet shluků a roztřídí tak použité databáze do nesprávného počtu tříd. Z toho důvodu vykazuje mnohem nižší úspěšnost klasifikace v porovnání s ostatními shlukovacími algoritmy zkoumanými v této práci. Diplomová práce tedy dochází k závěru, že algoritmus Mean-shift nelze pro klasifikaci daných datasetů použít.

Při většině použitých metod předzpracování obrazu mají shlukovací algoritmy přibližně stejnou úspěšnost zatřídění databáze. Při použití klasifikační sítě VGG 16 a kombinace metod BoVW modelu a konvolučního autenkodéru dosahují algoritmy Aglomerativní shlukovací a BIRCH vyšší úspěšnosti než ostatní algoritmy.

Při testování použité databáze bylo dokázáno, že algoritmus K-means++ dosahuje srovnatelné úspěšnosti klasifikace jako algoritmus K-means. Vhodnější řešení utvoření prvotních centroidů algoritmu K-means++ tedy nezlepší výslednou klasifikaci použitého datasetu.

Při testování použitého datasetu byl potvrzen teoretický předpoklad, že shlukovací algoritmus Mini Batch K-means++ konverguje rychleji než algoritmy K-means a K-means++. Dle teorie by utvoření shluků algoritmem Mini batch K-means++ mělo být horší. Tento předpoklad může diplomová práce potvrdit pouze pro některé metody předzpracování obrazu. Při předzpracování obrazu pomocí kombinace BoVW modelu a konvolučního autoenkodéru dosahuje shlukovací algoritmus Mini

Batch K-means++ v průměru o 3 % nižší úspěšnosti než algoritmus K-means++. Při předzpracování sítí VGG 16 je rozdíl úspěšnosti klasifikace těchto algoritmů 20 %. Diplomová práce se domnívá, že horší utváření shluků algoritmu Mini Batch K-means++ se projeví pouze při přesné extrakci příznaků. Proto pro tyto typy předzpracování obrazu dosahuje klasifikace tímto algoritmem nižší úspěšnosti.

Při použití předzpracování obrazu pomocí technik strojového učení bez učitele dospěla diplomová práce k závěru, že shlukovací algoritmy jsou schopné s vysokou úspěšností klasifikovat použitou databázi pouze při dodržení následujících podmínek. První podmínkou je třídění pouze malého počtu tříd (ideálně dvou), protože práce zjistila, že úspěšnost klasifikace databáze klesá s rostoucím počtem tříd. Druhou podmínkou je, že obrazy v jedné třídě musí být dostatečně podobné a zároveň dostatečně odlišné od obrazů v jiných třídách.

Při požadavku klasifikace většího počtu tříd dosahují shlukovací algoritmy v kombinaci s metodami předzpracování pomocí učení bez učitele nízké úspěšnosti. Při klasifikaci většího počtu tříd je možnost pro předzpracování použít předtrénovanou klasifikační neuronovou síť. Tato práce extrahuje příznaky pomocí předrénované sítě VGG 16. Při extrakci příznaků pomocí učení s učitelem v kombinaci se shlukovacími algoritmy lze s velkou úspěšností klasifikovat větší počet tříd použité databáze.

Shlukovací algoritmy mají své výhody. První výhodou je skutečnost, že učení bez učitele nepotřebuje trénovací a validační skupinu obrazů, takže algoritmy není nutné učit, jak správně obrazy třídit. To přináší výrazné zjednodušení práce a redukuje počet obrazů i čas, který je potřeba, aby daný přístup správně fungoval. Druhou výhodou je snížení času nutného k výpočtu algoritmu, protože algoritmus není nutné trénovat, a také většina shlukovacích algoritmů má nižší výpočetní nároky než algoritmy, které spadají do oblasti učení s učitelem.

6 Závěr

Tato diplomová práce se zabývala klasifikací databáze obrazů na základě obrazové podobnosti pomocí metod předzpracování obrazu a technik strojového učení bez učitele. Dále byla vytvořena aplikace, ve které byly tyto metody otestovány.

Jako metody předzpracování obrazu byly použity extraktory příznaků. Tyto algoritmy dokáží v obraze detekovat a následně popsat obrazové příznaky. Příznaky jsou významné části obrazu nesoucí informaci obrazu. V diplomové práci jsou popsány jak jednoduché extraktory příznaků jako disktrétní 2D a 3D konvoluce, tak i komplexnější algoritmy pro extrakci příznaků jako konvoluční autoenkodér, ORB a BRISK. Je zde také řešeno uspořádání příznaků extrahovaných algoritmy ORB a BRISK do matematického modelu BoVW.

Klasifikace obrazů na základě obrazové podobnosti je provedena pomocí shlukovacích algoritmů. Ty pracují na principu shlukování, kdy podobné pixely jsou shlukovány do společných shluků na základě výpočtu vzdáleností. Tato práce řeší hierarchické i nehierarchické metody shlukové analýzy. Jako hierarchické shlukovací algoritmy byly použity Aglomerativní shlukování a BIRCH. Jako nehierarchické metody jsou řešeny algoritmy K-means, K-means++, Mini Batch K-means++ a Meanshift. V diplomové práci byl také kompletně naprogramován shlukovací algoritmus K-majority, který dokáže shlukovat data v binární formě.

Praktickým výstupem diplomové práce je naprogramovaná aplikace v programovacím jazyce Python, která umožňuje třídění databáze obrazů do jednotlivých tříd. Aplikace provádí klasifikaci pomocí kombinace metody předzpracování obrazu a shlukovacího algoritmu. Aplikace umožňuje zatřídění databáze barevných obrazů obsahujících 10 tříd a databáze MNIST.

Pro použitý dataset bylo zjištěno zhoršení přesnosti shlukovacích algoritmů použitím diskrétní 2D a 3D konvoluce. Při převodu barevného obrazu na šedotónový se úspěšnost zatřídění použité databáze snížila o 5 až 25 %. Extrakce příznaků algoritmem ORB a následné použití BoVW modelu zvýší úspěšnost klasifikace 0 až 15 % za cenu razantního zvýšení výpočetních nároků. Předzpracování konvolučním autoenkodérem zvýší úspěšnost klasifikace průměrně o 10 %.

Bylo zjištěno, že shlukovací algoritmy dokáží třídit použitou databázi obrazů pouze omezeně. Pokud je požadavek roztřídit databázi pouze do dvou tříd, lze algoritmy použít s velikou úspěšností (průměrně 95 %). Pokud je ale potřeba roztřídit databázi do více tříd, úspěšnost shlukovacích algoritmů prudce klesá. Výhodou shlukovacích algoritmů oproti většině metod strojového učení s učitelem je jejich relativně nízká výpočetní náročnost a schopnost správné klasifikace bez anotovaných dat.

Tato diplomová práce testuje použití klasifikační neuronové sítě VGG 16 pro extrakci příznaků, která spadá do oblasti strojového učení s učitelem. Bylo zjištěno, že

při extrakci příznaků pomocí této sítě lze použitou databázi shlukovacími algoritmy klasifikovat s vysokou přesností i pro větší počet tříd. Při této metodě předzpracování jsou při klasifikaci nejúspěšnější shlukovací algoritmy Aglomerativní shlukovaní a BIRCH a jejich úspěšnost klesá od 100 do 92,7 % pro různý počet tříd.

Dalším zjištěním bylo, že úspěšnost algoritmů zůstává relativně konstantní se změnou počtu obrazů v jednotlivých třídách. Práce ověřila, že pro shlukovací algoritmy je vhodné, aby obrazy v jedné třídě byly podobné a zároveň, aby obrazy v různých třídách byly dostatečně odlišné.

Literatura

[1] P. TOMEK: *Detekce ohně a kouře ve videosekvenci*. Bakalářská práce, Brno, FIT VUTv Brně, 2010, s.31.

[2] J. HAVELKA: *Geometrické transformace obrazu*. Bakalářská práce, Brno, FIT VUT v Brně, 2008, s.29.

[3] R. C. GONZALEZ, R. E. WOODS: *Digital Image Processing*. Prentice Hall, New Jersey, 2002, s.793.

[4] F. Žilka: *Detektory a deskriptory oblastí v obrazu*. Diplomová práce, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2016. s. 86.

[5] D. TYAGI: *Introduction To Feature Detection And Matching*. Analytics Vidhya, leden 2019. Dostupné z URL: <`https://medium.com/analytics-vidhya/introduction-to-feature-detection-and-matching-65e27179885d`>

[6] P. BARBORKA: *Analýza metod pro detekci příznaků u v digitalizovaném obraze*. Bakalářská práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra kybernetiky, 2016, s. 61.

[7] E. ROSTEN, T. DRUMMOND, V. BABAUD, K. KONOLIGE, G. BRADSKI: *Machine Learning for High-Speed Corner Detection*. Willow Garage, Menlo Park, California, 2011.

[8] V. LEPETIT, Ch. STRECHA, P. FUA: *BRIEF: Binary Robust Independent Elementary Features*. CVLab, EPFL, Lausanne, Switzerland, 2010.

[9] E. RUBLEE, : *ORB: an efficient alternative to SIFT or SUR*. Willow Garage, Menlo Park, California, 2011.

[10] S. LEUTENEGGER, M. CHLI, R. SIEGWART: *BRISK: Binary Robust Invariant Scalable Keypoints*. Autonomous Systems Lab, ETH Zurich, 2011, s. 8.

[11] A. OLAODE, G. NAGHDY, C. TODD: *Unsupervised Classification of images: A Review*. ResearchGate, 2014.

[12] C. GRANA, m. MANFREDL, D. BORGHESANI, R. CUCCHIARA: *A Fast Approach for Integrating ORB Descriptors in the Bag of Words Model*. ResearchGate The International Society for Optical Engineering, 2013.

[13] JAHIRUZZAMAN, S. SAHA, A. K. HAWLADER: *Dynamically Reconfigurable Parallel Architecture Implementation of 2D Convolution for Image Processing over FPG* . Jahangirnagar University, Bangladesh, 2015, s. 6.

[14] A. MYŠÁK: *Využití autoenkodérů pro detekci anomálií v obraze*. Bakalářská práce, Technická Univerzita v Liberci, Fakulta mechatroniky, informatiky a mezioborových studií, 2018, s. 42.

[15] JITENDER: *Types of padding in convolution layer* [online]. GeeksforGeeks, 2018, [cit. 9. 3. 2019]. Dostupné z URL: <`https://www.geeksforgeeks.org/types-of-padding-in-convolution-layer/`>

[16] B. ZITOVÁ: *Zpracování obrazu a rozpoznávání I*. Ústav teorie informace a automatizace s. 194.

[17] D. CORNELISSE: *An intuitive guide to Convolutional Neural Networks* [online]. FreeCodeCamp, Machine Learning, 2018, [cit. 10. 3. 2019]. Dostupné z URL: <`https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/`>

[18] Y. SEKIKAWA, K. ISHIKAWA, H. SAITO: *Constant Velocity 3D Convolution.* Denso IT Laboratory, Tokyo; Keio University, Yokohama, 2018 s. 12.

[19] M. LOHNISKÝ: *Využití autoenkodérů pro tvorbu hlubokých sítí*. Bakalářská práce, FIT VUT v Brně, 2012, s. 33.

[20] H. CASPER: *Activation Functions Explained - GELU, SELU, ELU, ReLU and more* [online]. Machine Learning From Scratch, Insight For Developers, 2019, [cit. 11. 3. 2019]. Dostupné z URL: <`https://mlfromscratch.com/activation-functions-explained/#/`>

[21] M. MITRENGA: *Konvoluční neuronová síť pro segmentaci obrazu*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018, s. 55.

[22] M. BASAVARAJAIAH: *Maxpooling vs minpooling vs average pooling* [online]. Medium,2019, [cit. 11. 3. 2019].

[23] J. DESPOIS: *Latent space visualization — Deep Learning bits #2* [online]. Hackernoon,2017, [cit. 12. 3. 2019]. Dostupné z URL: <`https://hackernoon.com/latent-space-visualization-deep-learning-bits-2-bd09a46920df`>

[24] M. KERHART: *Učení hlubokých neuronových sítí v Mathematica*. Diplomová práce, FEL ČVUT, katedra počítačů. 2015, s. 29.

[25] D. KINGMA, J. L. BA: *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*. ICLR, 2015, s. 15.

[26] B. BUŠO, M. KOLMAN, V. VACEK: *Porovnání metod machine learningu pro analýzu kreditního rizika*. Vysoká škola ekonomická v Praze.

[27] S. RUSSELL, P. NORVIQ: *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003, chapter 5 Learning, s. 693 - 757.

[28] M. HASAN, A. KOTOV, A. I. CARCONE, M. DONG, S. NAAR, K. B. HARTLIEB: *A study of the effectiveness of machine learning methods for classification of clinical interview fragments into a large number of categories*. Journal of Biomedical Informatics, 2016.

[29] J. KURFÜRSTOVÁ: *Strojové učení kouzla zbavené*. EDTECH KISK, Masarykova univerzita, 2018 [cit. 15. 10. 2019].

[30] J. KANTOR: *Učení bez učitele*. Diplomová práce, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008, s. 74.

[31] J. KELBEL, D. ŠILHÁN: *Shluková analýza*. CVUT, Fakulta Elektrotechniká, 2013, s.11.

[32] B. S. EVERITT, S. LANDA, M. LEESE: *Cluster Analysis*. Daniel Stahl King's College London, UK, Pátá edice.

[33] J. HAVLÍČEK: *Algoritmizace metod shlukování* . Bakalářská práce, VŠB - Technická univerzita Ostrava, Fakulta metalurgie a materiálového inženýrství, 2012, s.29

[34] P. KONČINSKÝ: *Učení bez učitele*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. s.46

[35] Pedregosa a další: *Scikit-learn: Machine Learning in Python* [online]. Journal of Machine Learning Research JMLR 12, s. 2825 - 2830, 2011, [cit. 25. 10. 2019]. Dostupné z URL: <`https://scikit-learn.org/stable/index.html`>

[36] L. BING: *Web Data Mining*. Springer London New York, druhé vydání, 2011, s. 643 ISBN 978-3-642-19459-7.

[37] D. ARTHUR, S. VASSILVITSKII: *k-means++: The Advantages of Careful Seeding*. Stanford theory group, Standford University, England, s. 11.

[38] S. R. FITRIYANI, H. MURFI: *The K-Means with Mini Batch Algorithm for Topics Detection on Online News*. Department of Mathematics University of Indonesia, Depok, Indonesia, 2016, s. 5.

[39] J. SCHULZ: *Hamming distance* [online]. 2008, [cit. 28. 10. 2019]. Dostupné z URL: <http://www.code10.info/index.php?option=com_content&view=article&id=59:hamming-distance&catid=38:cat_coding_algorithms_data-similarity&Itemid=57>

[40] G. SEIF: *The 5 Clustering Algorithms Data Scientists Need to Know* [online]. Towards Data Science, 2018, [cit. 28. 10. 2019]. Dostupné z URL: <https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68>

[41] H. CHO, S. J. KANG, Y. H. KIM: *Image Segmentation Using Linked Mean-Shift Vectors and Global/Local Attributes*. TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, 2017, s.2132 - 2140.

[42] *Visual Studio Code* [online].[cit. 10. 10. 2019]. Dostupné z URL: <https://code.visualstudio.com/>

[43] Y. LeCun, C. Cortes, Ch. J. C. Burges, *THE MNIST DATABASE of handwritten digits* [online]. [cit. 11. 3. 2020]. Dostupné z URL: <http://yann.lecun.com/exdb/mnist/>

[44] *NumPy* [online]. [cit. 20. 10. 2019]. Dostupné z URL: <https://numpy.org/>

[45] *SciPy* [online]. [cit. 20. 10. 2019]. Dostupné z URL: <https://www.scipy.org/>

[46] *Matplotlib: Visualization with Python* [online]. [cit. 20. 10. 2019]. Dostupné z URL: <https://matplotlib.org/>

[47] *scikit-image Image processing in Python* [online]. [cit. 25. 10. 2019]. Dostupné z URL: <https://scikit-image.org/>

[48] *tkinter — Python interface to Tcl/Tk* [online]. [cit. 15. 11. 2019]. Dostupné z URL: <https://docs.python.org/3/library/tkinter.html>

[49] *queue — A synchronized queue class* [online]. [cit. 11. 12. 2019]. Dostupné z URL: <https://docs.python.org/3/library/queue.html>

[50] *threading — Thread-based parallelism* [online]. [cit. 11. 12. 2019]. Dostupné z URL: <https://docs.python.org/3/library/threading.html>

[51] CHOLLET, FRANÇOIS a další: *Keras* [online]. Keras Documentation, [cit. 4. 3. 2020]. Dostupné z URL: <https://keras.io/>

[52] *TensorFlow* [online].[cit. 4. 3. 2020]. Dostupné z URL: <https://www.tensorflow.org/>

[53] *OpenCV* [online].[cit. 1. 3. 2020]. Dostupné z URL: <https://opencv.org/>

[54] *ImageNet* [online].[cit. 20. 4. 2020]. Dostupné z URL: <http://www.image-net.org/>

[55] CHOLLET, FRANÇOIS a další: *Keras Applications* [online].[cit. 20. 4. 2020]. Dostupné z URL: <https://keras.io/api/applications/>

[56] *5-MIN TUTORIAL: K-MEANS CLUSTERING IN PYTHON* [online].[cit. 10. 3. 2020]. Dostupné z URL: <http://konukoii.com/blog/2017/01/15/5-min-tutorial-k-means-clustering-in-python/>

[57] *GCC, the GNU Compiler Collection* [online].[cit. 12. 4. 2020]. Dostupné z URL: <https://gcc.gnu.org/>

[58] D. CORTESI: *Using PyInstaller* [online]. [cit. 22. 11. 2019]. Dostupné z URL: <https://pythonhosted.org/PyInstaller/usage.html>

[59] *Open Images Dataset V5* [online]. [cit. 20. 12. 2019]. Dostupné z URL: <https://storage.googleapis.com/openimages/web/index.html>

[60] *Česká vlajka* [online]. Google, [cit. 20. 12. 2019]. Dostupné z URL: <https://www.google.cz/search?q=%C4%8Desk%C3%A1+vlajka&source=lnms&tbm=isch&sa=X&ved=2ahUKEwj-s83YiMLpAhWMmBQKHdsFCOYQ_AUoAXoECBQQAw&biw=1920&bih=969>

[61] F. LUND: *Nauč se Python!* [online]. Creative Commons Attribution-ShareAlike 4.0 International [cit. 10. 10. 2019]. Dostupné z URL: <https://naucse.python.cz/course/pyladies/>

[62] F. LUND: *An Introduction To Tkinter* [online]. [cit. 15. 11. 2019]. Dostupné z URL: <https://effbot.org/tkinterbook/tkinter-index.htm>

[63] *Threading/Queue in Python* [online]. [cit. 16. 11. 2019]. Dostupné z URL: <https://stackoverflow.com/questions/7074984/threading-queue-in-python>

A Obsah přílohy

V příloze je umístěna zmiňovaná aplikace. Aplikace byla testována pro Windows 10. Pokud je aplikace spuštěna a jsou úspěšně dokočeny výpočty, v kořenovém adresáři je vytvořena složka „vysledek“, ve které je umístěna klasifikovaná databáze.

- `/` ..... kořenový adresář přílohy
  - `Databaze` ..... Použitá databáze obsahující 10 tříd
    - `červená růže`
    - `česká vlajka`
    - `hnědý medvěd`
    - `klaviatura`
    - `lev`
    - `okurky`
    - `orel`
    - `počítačová klávesnice`
    - `počítačová myš`
    - `zebra`
  - `icons` ..... ikony použité v aplikaci
    - `odIcon.png`
    - `playIcon.png`
  - `modely` ..... uložené přednaučené modely konvolučního autoenkodéru
    - `2020_03_18__13_01.h5`
    - `2020_03_21__03_50.h5`
    - `mnist_2020_04_03__01_02.h5`
  - `Zdrojový kód` ..... zdrojové kódy aplikace
    - `App.py`
    - `Preprocessing_and_clustering.py`
    - `Pretrained_autoencoder.py`
    - `Pretrained_autoencoder_MNIST.py`
    - `K_majority.py`
    - `hamming.c`
  - `Aplikace.exe` ..... soubor pro spuštění aplikace