Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Vyhledávání obrázků podle obsahu

Plzeň 2020 Martin Hamet

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 21. července 2020

Martin Hamet

Abstract

This bachelor thesis focuses on content-based image retrieval. The purpose of this thesis is to survey available datasets and methods for the task. For comparison, two interesting methods were selected, one representing a classical approach using histograms and the other method based on convolutional neural networks. Evaluation of the both approaches was done on the two most often used datasets `Corel-10k` and `Caltech-101`, which are general enough to provide a wide range of different images.

Abstrakt

Bakalářská práce je zaměřená na vyhledávání podobných obrázků na základě jejich obsahu. Cílem bylo seznámit se s dostupnými datovými sadami a metodami pro tento úkol a otestování vybraných metod. Pro porovnání jsem vybral dvě zajímavé metody, reprezentující klasický přístup k problému, s využitím histogramů a metodu založenou na konvolučních neuronových sítích. Testování obou metod proběhlo na dvou nejčastěji využívaných datových sadách `Corel-10k` a `Caltech-101`, které jsou velmi obecné a poskytují široké spektrum rozdílných obrázků.

Obsah

1 Úvod

S popularitou digitálních zařízení, která jsou schopna pořizovat fotografie, značně roste sdílení informací ve formě obrázků a to především prostřednictvím internetu. Přestože je, již od roku 1990[15], vyhledávání snímků dobře prozkoumanou disciplínou stále se objevují nová použití s technologiemi jako například počítačové vidění, expertní systémy, atd. Obvykle bylo vyhledávání založené na datech spojených s obrázkem jako název, klíčová slova a podobně. Kvalita těchto přidaných dat se značně liší a nemusí být konzistentní s obsahem obrázků, což vede k využití vyhledávání podle skutečného obsahu obrázku.

Problém při vyhledávání obrázků podle jejich obsahu vzniká na dvou hlavních místech. Schopnost uživatele exaktně vyjádřit svůj požadavek a schopnost systému rozpoznat vlastnosti objektu snímku z jeho reprezentace[19, 22]. Jinými slovy, systém musí být schopen rozpoznat například vlastnosti auta z jednotlivých pixelů snímku. Jedná se tedy o problém rozpoznání záměru ze strany uživatele a vyšší sémantiky vzhledem ke snímku.

Kolem roku 2000 se výzkum zaměřil na dvě nové metody. Využití popisu pomocí lokáních invariantních vlastností *SIFT* a popisu pomocí *bag of words*. První zmíněná metoda se snaží nalézt příznaky invariantní vůči transformacím, které jsou dále použity pro popis obsahu obrázku. Přístup metody *Bag of words* spočívá v klasifikaci prvků snímku. Tedy slovním popisem obsahu obrázku. Jedná se v podstatě o klíčová slova, která slouží pro vyhledávání.

K posledním trendům patří *deep learning*. Jedná se o soubor algoritmů, které se snaží modelovat vyšší úroveň abstrakce obsahu obrázku pomocí mnohavrstvé hierarchické struktury nelineárních transformací [1]. Tato struktura se snaží napodobovat funkci lidského vnímání. Dobrým příkladem *deep learning* jsou neuronové sítě s mnoha skrytými vrstvami.

V této práci se čtenář seznámí se základními principy vyhledávání obrázků na základě jejich obsahu. Zvolené metody budou implementovány a otestovány na různých datových sadách. A v poslední řadě zhodnotíme jejich výsledky.

2 Vyhledávání obrázků na základě obsahu

Takzvaný `Content Based Image Retrieval` (`CBIR`) je úloha vyhledávání, která se prolíná s úlohami počítačového vidění, umělé inteligence a dalšími. První fází je analýza obsahu obrázku a získání informací o jeho obsahu na základě textury, tvarů, nebo jiných charakteristik ,které dokážeme získat (viz 2.1). Obecně není vytváření popisu snímku limitováno konkrétní metodou, ale naopak se často jedná o kombinaci různých přístupů počítačového vidění. Vlastnosti obrázku nemusejí být vždy globální charakteru, ale mohou být lokální na základě nějakého rozdělení, nebo segmentace snímku. Dále se budeme zabývat pouze globálními vlastnostmi. Z těchto vlastností je vytvořen popis snímku, který se dále používá při jeho vyhledávání. Při vyhledávání relevantních obrázků systém hledá v modelu vytvořeném z indexovaných popisů snímků na základně podobnosti popisů a vstupu. Pro tento účel musí systém definovat způsob určení podobnosti, jako například výpočet odlišnosti dvou popisů.

Jedním z hlavních úkolů `CBIR` je zajištění efektivity při získávání a zpracování obrázků a eliminace lidské práce při indexování (vytváření popisů) snímků [3].

2.1 Metody CBIR

2.1.1 Barva

Barva je jednou ze základních a snadno získatelných vlastností snímku a mnoho metod je na ní postaveno. Často jsou tyto metody založené na vytváření histogramu. Při analýze jednotlivých obrázků vytváříme barevné spektrum každého obrázku. Při získávání obrázků z datasetu můžeme vyhledávat na základě vstupů jako poměru barev, nebo stejným způsobem vytvoříme otisk obrázku a vyhledáme podobné. Pro nalezení podobných obrázků přímo podle jejich histogramu můžeme využít různých populárních metod jako `histogram intersection`[17], nebo nějakou z jejích variant.

2.1.2 Textura

Obrázek 2.1: Ukázka Gaborových filtrů[1]

Rozpoznávání na základě textury se nemusí zdát jako příliš užitečná vlastnost, ale za texturu lze považovat například i obloha, les, listí, atd. Při zkoumání textury se často využívá tzv. statistik druhého řádu `second-order statistics`. Jinými slovy zkoumáme dvojice pixelů a jejich relativní rozdíly jasu. Z těchto hodnot můžeme vypočítat vlastnosti jako směrovost, hrubost, pravidelnost atd. [18] Můžeme také využít Gaborových filtrů. Jedná se o lineární filtry v různých orientacích, které si lze za určitých podmínek představit jako detekci hran v určitém směru (viz obr. 2.1). Samozřejmě existují další způsoby popisu a analýzy textur jako fraktály, vlnová transformace (`wavelet transformation`), atd.

2.1.3 Tvar

Tvar objektů na snímku je velmi silnou popisovou vlastností a jak naznačuje studie `Recognition-by-component`[2], jedná se o jeden z hlavních způsobů člověka při rozpoznávání objektů. Pokud je možná segmentace objektů můžeme určovat jejich vlastnosti jako poměr šířky a výšky, momentové charakteristiky, kruhovost atd. Případně vytvoření histogramu hran může být také dobrý popis snímku[8].

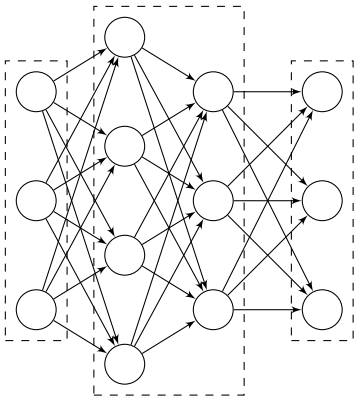
2.1.4 Ostatní

Při hledání podobných obrázků můžeme také využít další informace spojené s obrázkem jako jeho metadata. Například informace o poloze mohou být klíčová v závislosti na účelu koncového systému. Další možnosti se nabízí s využitím různých transformací, nebo zařazení komplexnějších rozpoznávacích prvků jako `SIFT` (`Scale Invariant Feature Transformation`)[13]. Jak bylo zmíněno dříve, obvykle využíváme kombinace výše zmíněných popisů a technik.

[1] Převzatý obrázek z https://mc.ai/improving-convolutional-neural-network-accuracy-using-gabor-filter-and-progressive-resizing/

3 Neuronová síť

Jedná se o prostředek počítačového učení, kde učíme počítač zadanou úlohu na připravených trénovacích datech. Například v případě rozpoznávání obrázků, kdy se systému předkládá velké množství snímků s předem známým obsahem. Systém tyto vstupy zpracovává a snaží se učit datové vzory odpovídající známým výsledkům.



Obrázek 3.1: Vrtvy sítě.

Samotná síť se skládá z mnoha uzlů (neuronů), které jsou mezi sebou hustě propojeny a tvoří tak vlastní neuronovou síť. Většinou jsou neurony organizovány do vrstev. Síť se obvykle skládá z jedné vstupní, výstupní a jedné nebo více skrytých vrstev. Data procházejí tedy sítí pouze jedním směrem od vstupní k výstupní vrstvě (viz obr.3.1).

Všechny vstupy neuronu mají přidělené váhy a produkt jejich hodnot je vstupem aktivační funkce neuronu (například sigmoida). Výstupem jednoho neuronu je tedy hodnota aktivační funkce. Tuto hodnotu O spočítáme jako:

$$O = S\left(\sum_{v}{}_i \cdot w_i\right) \tag{3.1}$$

kde v_i je hodnotou vstupu i neuronu, w_i je váha příslušná vstupu i a S je aktivační funkce.

Na počátku jsou obvykle váhy všech vstupů jednotlivých neuronů nastaveny na náhodné hodnoty. V procesu trénování, kdy jsou do sítě zavedena vstupní data (na neurony ve vstupní vrstvě), transformací vstupních hodnot (zatím náhodnými vahami) získáme hodnoty aktivačních funkcí ve výstupní vrstvě. Dále je třeba provést samotný akt učení, kdy pomocí zpětné propagace[20] upravujeme váhy neuronů tak, aby se výstup více blížil našemu požadovanému. Po dostatečném počtu opakování tohoto procesu bude síť schopná vrátit požadovaný výstup na podobný vstup.

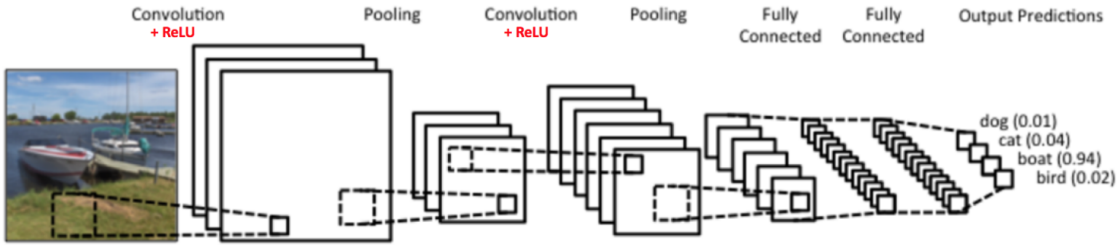
Vzhledem k tomu že hledáme podobné snímky a nesnažíme se o klasifikaci do tříd, je možné využít hodnoty jednotlivých neuronů v plně propojené vrstvě `FCL` jako příznakový vektor, který lze srovnávat s ostatními z databáze modelu, pro získání podobných obrázků. Plně propojenou vrstvou myslíme

vrstvu, kde každý neuron je navázaný na každý neuron z vrstvy předchozí. Jedná se tedy o vrstvu, která vytváří velmi hustou síť s velkým počtem parametrů.

3.1 Konvoluční neuronová síť

Princip konvoluční neuronové sítě *CNN*[1] spočívá ve schopnosti sítě učit se různě nahlížet na snímky pomocí konvolučních filtrů, různé pohledy mohou být jednoduché jako například barevné filtry, detekce hran, nebo komplexní v hlubších vrstvách sítě jako detekce rohů, textur, atd. Informace získané tímto způsobem můžeme rovnou zpracovávat nebo použít jako vstup klasické neuronové sítě.

Název *CNN* je odvozen od konvolučního operátoru, který použijeme jako prostředek pro získání výše zmíněných pohledů na snímek viz 3.1.1. Obecná CNN je založená na získávání informací z obrázku pomocí těchto konvolučních operátorů (fltrů), které si však sama vytvoří. Kombinací jednotlivých filtrů je síť schopná detekovat netriviální vlastnosti snímku. Jako příklad lze uvést kombinaci filtrů pro detekci horizontálních a vertikálních hran, která by umožnila síti rozpoznávat *rohy* například u dveří. Samozřejmě většinou filtrům vytvořeným vlastní sítí, a jejich kombinacím nelze snadno přiřadit význam srozumitelný člověku, avšak princip zůstává stejný.



Obrázek 3.2: Struktura CNN[2]

Celková struktura CNN je znázorněna na obr.3.2, kde tato síť obsahuje dvě konvoluční vrstvy za kterými vždy následuje *ReLU*[3] a *pooling* a tři plně propojené vrstvy *FCL*[4]. Tyto poslední vrstvy představují klasickou neuronovou síť, kde je každý neuron propojen s každým neuronem v následující i předchozí vrstvě, jak bylo popsáno výše (kapitola 3 a na obr. 3.1).

[1] Convolutional Neural Network

[2] Převzatý obrázek CNN `https://ujwlkarn.files.wordpress.com/2016/08/screen-shot-2016-08-07-at-4-59-29-pm.png?w=1493`

[3] Rectified Linear Unit

[4] Fully Connected Layer

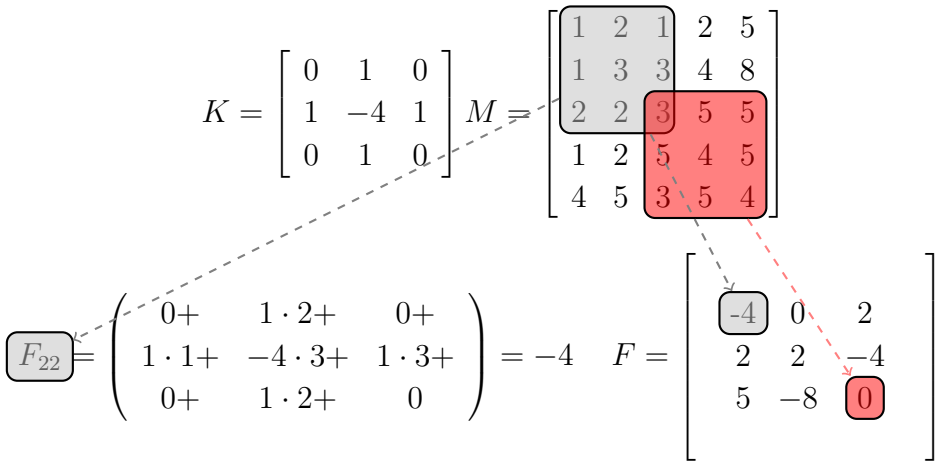
3.1.1 Konvoluční operátor

Konvoluce je matematická operace nad maticí s použitím matice druhé (jádra). První maticí je v našem případě samotný snímek respektive jeho pixely uspořádané do matice a jádrem bude samotný operátor. Výslednou hodnotu matice získáme ze vztahu:

$$F_{x,y} = \sum_{i=-k}^{k} \sum_{j=-k}^{k} M_{x-i,y-j} \cdot K_{i,j} \tag{3.2}$$

kde $F_{x,y}$ je hodnota výsledné matice na pozici $[x, y]$, k je dimenze jádra (jádro je obvykle čtvercovou maticí), M je zdrojová matice a K je matice jádra.

Samotný výpočet operace je lépe vidět na příkladu (viz obr.3.3).



Obrázek 3.3: Příklad výpočtu konvoluce

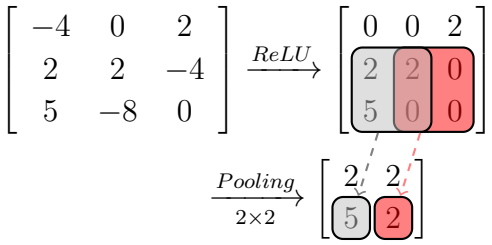
$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Obrázek 3.4: Konvoluční filtr pro detekci horizontálních hran

Vhodnou volbou hodnot jádra lze vytvořit filtr například pro detekci horizontálních hran (viz obr.3.4). Při použití takového filtru je třeba ošetřit kraje zdrojové matice, protože filtr nemůže být aplikován na okraji kvůli chybějícím hodnotám za hranou snímku. Buď bude výsledek menší v závislosti na velikosti filtru, nebo lze doplnit původní matici například o nulové hodnoty na okrajích. Tato problematika je obvykle nazývána `padding`.

Nelinearita a *pooling*

Jednotlivé operace konvoluce jsou pouze lineární operace a proto s využitím $ReLU$ zavedeme do procesu zpracování nelinearitu, kdy nahradíme všechny záporné hodnoty ve výsledku hodnotou nulovou (viz obr.3.5). Nelinearita nám poskytne lepší schopnost zobecňovat, kdy různé (podobné) mohou produkovat stejný výsledek. Toto nahrazení také zrychluje učení. V případě nulové váhy daného neuronu tento neuron přestává mít jakýkoliv vliv na výstup (není potřeba počítat úpravu jeho váhy).

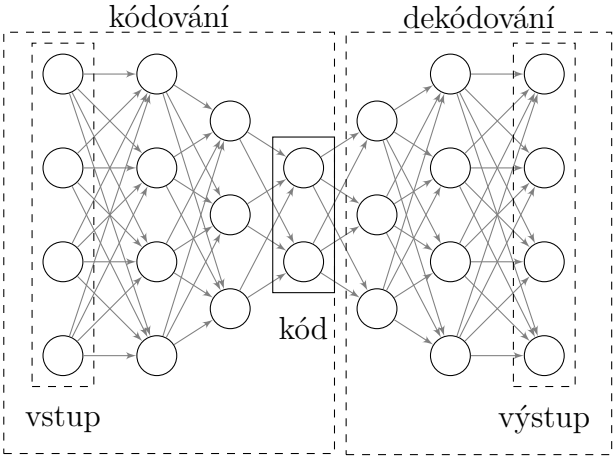


Obrázek 3.5: Příklad $ReLU$ a následný *pooling*

Proces *pooling* za každou konvolucí se stará o snížení dimenze jednotlivých výstupů z filtrů se zachováním jejich nejdůležitějších součástí. Jedná se tedy o zmenšení výsledné matice, kde definujeme okolí pro každý prvek a z daného okolí získáme jednu hodnotu. V případě `MaxPooling` se jedná o maximální hodnotu v okolí (viz obr.3.5).

3.2 Autoenkoder

Jedná se o neuronovou síť určenou pro učení bez učitele. Není tedy nutné předem znát správný výsledek při trénování. Princip funkce autoenkoderu spočívá v učení sítě co nejlépe replikovat vstupní hodnoty na svém výstupu. Tedy požadovaný výstup je vstupní vektor sítě a proto vstupní i výstupní vrstva musí mít stejný počet neuronů. Struktura autoenkoderu se obecně skládá z kódovací a dekódovací části, jak je vidět na obr. 3.6. Zpracováním vstupu v kódovací části získáme kódové slovo, které se dekódovací část snaží převést zpět na původní vstup. V našem případě je tedy výstupem vzniklý



Obrázek 3.6: Struktura autoenkoderu

kód a nikoliv výstupní vrstva, která slouží pouze jako nástroj pro učení. V případě jiných autoenkoderů tomu tak být nemusí. Například `denoising autoecoder` se snaží vyčistit snímek od šumu a výstupem potom je skutečně výstupní vrstva s rekonstruovaným snímkem (bez šumu).

Vytvářením *úzkého hrdla* (vrstvy s menším počtem neuronů) zajistíme ztrátovou kompresi vstupu a nutíme tím síť k vytvoření robustního kódování (odtud název autoenkoder) ze kterého se bude síť snažit co možná nejlépe sestavit původní vstup.

V kombinaci s konvolucí resp. `CNN` se autoenkoder snaží najít kódování snímku pomocí jednotlivých filtrů, tedy popis snímku pomocí jeho význačných vlastností. Takto vzniklý kód by měl obsahovat zobecněný popis snímku, který bude možné použít pro porovnání a výběr podobných snímků.

4 Datové sady

K dispozici je mnoho rozdílných datových sad, obvykle vytvářených za účelem klasifikace. Snímky jsou zařazeny do tříd a to často bývá pomocí souborové struktury, kdy jsou snímky rozděleny do složek podle jejich příslušnosti. Úlohy založené na `CBIR` jsou obvykle specifikovány konkrétněji a vyžadují speciální datové sady např. pro diagnostické nebo expertní systémy. Protože vytvoření obecného datasetu pro `CBIR` je velmi náročné obvykle využíváme existujících datasetů určených pro klasifikaci.

Z následujících datasetů jsem vybrali dva často používané, které nejsou příliš velké, snadno se s nimi manipuluje a bude je možné využít k porovnání (`Corel-10k` a `Caltech-101`).

Caltech 101(256)[6][7] Jedná se o databázi čítající 9 000(30 000) obrázků dělených do 101(256) kategorií. Obrázky byly pořízeny s využitím *Google Image Search* a ručně roztříděny do kategorií. Kategorie průměrně obsahují 90 (119) prvků. Jednotlivé snímky jsou průměrně rozměrů 300×200pixelů.

Obrázek 4.1: Příklad obrázků z `Caltech-101`

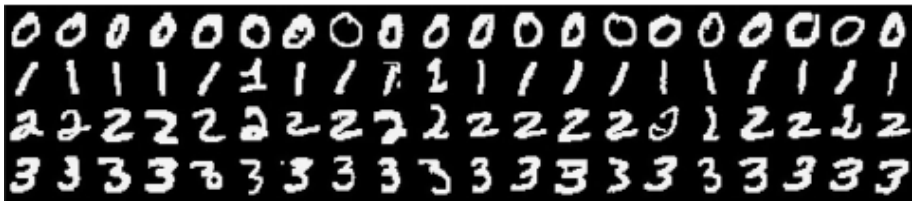
Corel[4] Dataset obsahuje 10 000 obrázků rozdělených do 80 kategorií. Obrázky mají formát 192×128 nebo 128×192, který pevně daným počtem pixelů usnadňuje zpracování.

Obrázek 4.2: Příklad obrázků z `Corel-10k`

ImageNet Velmi rozsáhlá databáze s více než 15 miliony obrázků s vysokým rozlišením dělených do 22 000 kategorií. Snímky byly získány z webu a ručně řazeny do tříd. Pro zpracování je možné využít podmnožin určených pro pro *ImageNet Large-Scale Visual Recognition Challenge*(ILSVRC)[16]. Nevýhodou jsou různé formáty rozlišení.

CIFAR10 Soubor se skládá z 60 000 obrázků rozdělených do 10 tříd. Každý snímek je ve formátu 32×32 pixelů. Set obrázků je rozdělen do trénovacích a testovacích podmnožin pro každou třídu. Bylo by možné použít rozšířenou verzi *CIFAR100*.

MNIST Jedná se o datovou sadu ručně psaných číslic rozdělených do 9 kategorií. Tato sada se mimo jiné často využívá pro ověření základní funkčnosti sítí. Sada obsahuje 70 000 černobílých obrázků z toho 10 000 obrázků je určeno k testování. Každá třída je tedy velmi dobře reprezentovaná svoji množinou.



Obrázek 4.3: Příklad obrázků z `MNIST`

5 Metriky

Systémy v oblasti klasifikace a rozpoznávání je třeba nějakým způsobem hodnotit, aby bylo možné vyjádřit jejich úspěšnost. Takové hodnocení umožňuje systém porovnat s konkurencí, nebo zjistit efekt jeho modifikací při vývoji.

Hlavní otázkou je jakým způsobem systém hodnotit. Z principu rozpoznávání vyplývají dva hlavní měřitelné atributy, úplnost a přesnost (viz 5.2 resp. 5.1).

Protože systémy popsané dvěma atributy není tak snadné porovnávat, v praxi je vhodnější využít popisu pouze jednou hodnotou. Takovou hodnotu je třeba přizpůsobit konkrétní úloze. Pro popis se nabízí několik možností jako užití *F-measure*, průměrná přesnost, střední geometrická přesnost (GMAP), střední průměrná přesnost (MAP), atd.

V našem případě bude výsledkem dotazu seznam obrázků seřazených podle relevance určené systémem. Bude tedy vhodné využít průměrné přesnosti (viz 5.4), která zohledňuje pořadí nalezených výsledků. Jak je vidět na obr. 5.1, průměrná přesnost značně zvýhodňuje prvky podle jejich pořadí v odpovědi na dotaz. Pro popis použijeme tedy metriku střední průměrné přesnosti, která zhodnotí výsledky všech použitých dotazů při testování s velkým důrazem na uspořádání získaného výsledku. Bylo by možné použít modifikaci GMAP, která vyžaduje dostatečný výkon (úspěšnost) ve všech dotazech. Náš systém nebude mít omezení na nejhorší povolenou úspěšnost a nebudeme tedy tuto metriku uvažovat.

5.1 Přesnost

Přesnost, neboli *precision*, hodnotí kolik relevantních výsledků systém poskytl. Jedná se o podíl relevantních a všech získaných výsledků (viz rovnice č. 5.1). Procentuální přesnost intuitivně ukazuje úspěšnost rozpoznávání pro daný dotaz. V případě že systém vrátil všechny relevantní výsledky hodnota přesnosti bude 100%. Výpočet přesnosti $P@n$ provedeme dle vzorce:

$$P@n = \frac{R_Q}{Q} \tag{5.1}$$

kde Q je počet získaných výsledků na dotaz a R_Q je počet relevantních výsledků ze všech získaných. Samotnou přesnost značíme jako $P@n$, kde n udává počet požadovaných výsledků dotazu. Pokud by v datech existoval

pouze jeden relevantní výsledek přesnost pro velké n by se značně snížila a bude tedy třeba tento fakt zohlednit.

5.2 Úplnost

Samotná přesnost nevypovídá o úplnosti dotazu. Jinými slovy chtěli bychom zohlednit počet všech relevantních prvků ve zdroji. Pro tyto potřeby se používá hodnota úplnosti neboli *recall*. jedná se o poměr získaných relevantních výsledků a všech relevantních prvků v prohledávaném zdroji. Výpočet úplnosti $R@n$ pro dotaz na n prvků provedeme dle vzorce:

$$R@n = \frac{R_Q}{R} \tag{5.2}$$

kde R_Q je počet relevantních výsledků ze všech získaných, podobně jako v případě výpočtu *precision*(viz 5.1), a R je počet všech relevantních prvků v databázi.

Hlavní nevýhodou tohoto atributu je nutnost znalosti všech relevantních prvků v databázi na daný dotaz a její často velmi nízké hodnoty v případě, že nepožadujeme velké množství obrázků na jeden dotaz. Úplnost je tedy omezena počtem vrácených snímků vůči počtu všech v databázi.

5.3 F-míra

Hodnota *F-measure*[14] udává harmonický průměr úplnosti a přesnosti (viz rovnice č.5.3). Vztah pro výpočet by bylo možné dále upravit, pokud by jsme chtěli klást větší důraz na přesnost nebo úplnost. Přestože *f-measure* zahrnuje oba výše zmíněné atributy, její hodnotu nelze intuitivně interpretovat a sloužila by pouze jako prostředek pro porovnání s případnou předpojatostí vůči jednomu z atributů.

$$F_1 = 2 \cdot \frac{P \cdot R_Q}{P + R_Q} \tag{5.3}$$

5.4 Průměrná přesnost

Tato hodnota kombinuje úplnost a přesnost pro daný počet dotazovaných prvků. Průměrná přesnost[21] $AP@n$ je průměrem hodnot přesností $P@n$ ze všech relevantních výsledků n. Hodnotu vypočítáme ze vztahu:

$$AP@n = \frac{\sum_n P@n}{R_Q} \tag{5.4}$$

kde $P@n$ je přesnost pro n požadovaných prvků (viz vzorec č. 5.1) a R_Q je počet získaných relevantních prvků. Příklad výpočtu je vidět na obr. 5.1.

5.5 Střední průměrná přesnost

Hodnotou mAP rozumíme pouze průměr jednotlivých průměrných přesností AP. Výpočet je znázorněn v tabulce 5.1.

Předpokládáme že máme celkem 4 relevantní prvky pro oba dotazy.

Query #1	F	T	T	F	T	F
$R@n$	0	0.25	0.50	0.50	0.75	0.75
$P@n$	0	0.50	0.67	0.50	0.60	0.50

Query #2	T	F	T	F	F
$R@n$	0.25	0.25	0.50	0.50	0.50
$P@n$	1.00	0.50	0.67	0.50	0.40

střední průměrnou hodnotu mAP spočítáme jako:

$$AP_{Q1} = (0.50 + 0.67 + 0.60)/3 = 0.59$$
$$AP_{Q2} = (1.00 + 0.67)/2 = 0.83$$
$$mAP = (0.59 + 0.83)/2 = 0.71$$

Tabulka 5.1: Příklad dotazu

T značí relevantní prvek
F značí irelevantní prvek

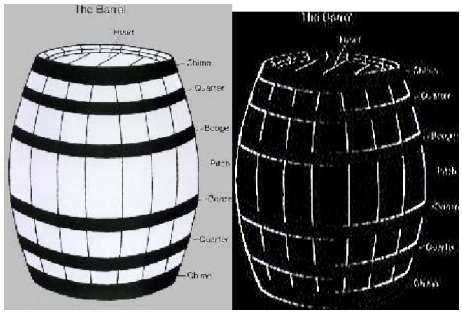
6 Zvolené metody

6.1 Color difference histogram

Jedná se o klasický přístup k rozpoznávání s použitím histogramu. Vzhledem k faktu, že vyhledáváme obecné obrázky, nemůžeme usuzovat žádné konkrétní vlastnosti a použití histogramu pro vytvoření příznakového vektoru je tedy vhodnou volbou. Histogram jako takový je invariantní vůči mnoha transformacím a je tedy dostatečně obecný.

Přestože, samotný histogram barev snímku může tvořit příznakový vektor, jednalo by se o poměrně slabý a příliš obecný popis. Histogram barev nezohledňuje objekty, které se na snímku skutečně nachází. Existují různé způsoby jak rozšířit histogram o další vlastnosti, které by přispěly k celkovému zachycení obsahu snímku. Nejjednodušším by mohlo být například rozdělení snímku na sektory, kde by pro každý sektor byl vytvořen vlastní histogram. Tato metoda zachovává alespoň nějakou informaci o rozložení barev na snímku.

Vybraná metoda `CDH` přistupuje k tomuto problému z pohledu lidského vnímání. Člověk při rozpoznávání nevnímá pouze barvy, ale především tvary objektů (viz 2.1.3). Například pomocí Sobelova operátoru můžeme snadno detektovat hrany, které nám poskytnou hodnotnou informaci o objektech na snímku.



Obrázek 6.1: Sobelův vertikální operátor

Problém je však se získáním a reprezentací informace o tvaru, nikoliv pouze o hranách. Tento problém se snaží řešit metoda Edge Histogram Descriptor (`EDH`)[10]. `EDH` detekuje hrany v devíti pevně daných směrech a tvořením histogramu takto nalezených hran pro jednotlivé segmenty zkoumaného obrázku. Detekce hran v různých směrech se dá snadno docílit nastavením operátoru viz obr (viz obr. 6.1), kde je ukázka detekce hran ve vertikálním směru.

6.1.1 Princip

Zvolená metoda Color Difference Histogram (CDH)[12] využívá kombinace detekce hran a změny barvy. Běžně se s detekcí hran setkáme na černobílých obrázcích. Tím bychom ale zbytečně ztráceli cenné informace, proto je využitá detekce hran s ohledem na jednotlivé barevné kanály snímku. Detekce hrany respektive výpočet gradientu probíhá obdobně s tím rozdílem, že je gradient získáme z diferenciálů jasu v jednotlivých kanálech. Tímto přístupem získáváme informaci i o hranách, které vznikají při chromatické změně a nikoliv pouze změně jasu.

Dále je histogram tvořen tak že, zaznamenáváme změnu barvy pixelů na stejné hraně a změnu orientace hrany pro pixely stejné barvy v daném okolí. Tvoříme tedy dvojici histogramů. Jeden pro barevné rozdíly a druhý pro rozdílné orientace. Hodnoty přidávané do histogramu jsou úměrné velikosti rozdílů. Získáme tím specifičtější informaci o vlastnostech objektů na snímku.

Barevný prostor

Při použití reprezentace barvy jako složek *RGB* a porovnáváním její změny předpokládáme že člověk přikládá každému z těchto kanálů stejnou váhu. To ovšem není pravda. Ve snaze přiblížit se lidskému vnímání vlastního snímku je využitý barevný prostor *Lab*. Oproti ostatním reprezentacím barvy má *Lab* výhodu především pro počítačové vidění a to právě proto že z pohledu vnímání člověka se jedná o uniformní reprezentaci. Jinými slovy v *Lab* je barva daná souřadnicemi *x, y, z* v prostoru a v našem případě dvě různé barvy, tak jak je vnímá člověk, budou v tomto prostoru od sebe vzdálené úměrně jejich rozdílu.

Tento poznatek nemusí nutně zajistit lepší rozpoznávací schopnost systému, ale v tomto případě tomu experimentální výsledky v článku naznačují [12].

6.2 Konvoluční autoenkoder

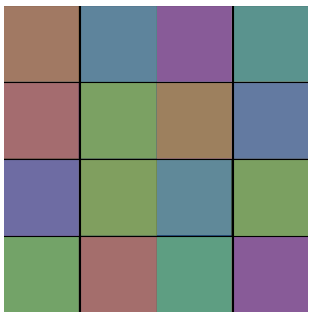
Obecně jsme již popsali funkci autoenkoderu (viz kapitola 3.2) a jeho konvoluční verze využívá stejného principu. Hlavní rozdíl spočívá v nahrazení plně propojených vrstev konvolučními. Autoenkoder nemusí obsahovat pouze vrstvy jednoho typu. V závislosti na řešeném problému můžeme vyzkoušet různé kombinace. Podobně jako při rozpoznávání často bývá jedna, nebo několik posledních vrstev plně propojených. V našem případě můžeme využít stej-

ného principu, kde se plně propojená vrstva bude učit, v jaké míře předchozí filtry přispívají celkové reprezentaci.

Pro návrh neuronových sítí neexistuje žádný sjednocený postup a vše často závisí na experimentech. Nicméně některé vlastnosti sítě lze intuitivně odhadovat. Některé články na internetu představují příklad autoenkoderu, kde klesá počtem filtrů s rostoucí hloubkou sítě. To sice dává smysl pokud uvažujeme o klasickém autoenkoderu, kdy se snažíme snížit dimensionalitu vstupu při kódování. Pokud ovšem uvážíme funkci konvolučních vrstev dospějeme k opačnému názoru (viz 6.2.1).

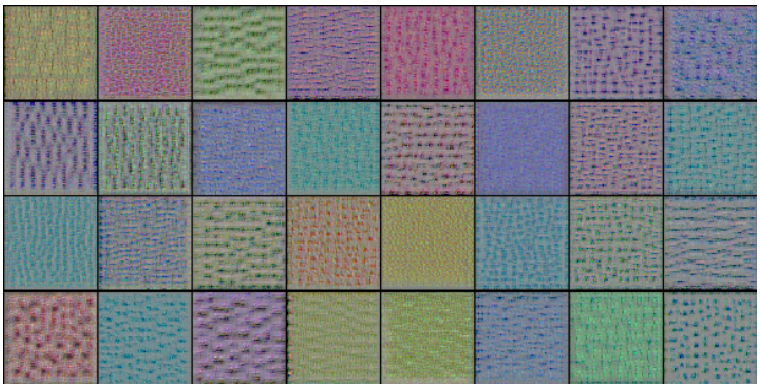
6.2.1 Počty filtrů

První konvoluční vrstvy detekují vlastnosti snímku na nízké úrovni. Jejich vstupem jsou skutečné hodnoty barevných kanálů obrázku. Bude se často jednat o rozpoznávání barvy, jasu a jejích změn jako například hrany, vše co se dá zjistit jednoduchou konvolucí. Jak je vidět na ukázce (viz obr. 6.2) první vrstva se naučila používat pouze barevné filtry nad snímkem. Vstupem následující vrstvy jsou mapy vytvořené vrstvou předchozí. Tato vrstva využívá již získaných informací z vrstev předchozích. Můžeme si například představit filtr, který využije detekce vodorovných a svislých hran z vrstvy předchozí a bude detekovat jejich průsečík (roh). Síť se ale váhy jednotlivých filtrů učí sama, proto ne všechny filtry nemusí mít takto intuitivní a zřejmý význam.



Obrázek 6.2: Ukázka obrázků, které nejvíce aktivují filtry první konvoluční vrstvy

S hloubkou sítě roste komplexnost rozpoznávaných vlastností (viz obr. 6.3) a abychom využili potenciál každé vrstvy, je vhodné zvyšovat počet také počet filtrů s rostoucí hloubkou.



Obrázek 6.3: Ukázka obrázků, které nejvíce aktivují filtry třetí konvoluční vrstvy

6.2.2 Předzpracování

Obecně pro jakékoliv rozpoznávání je vhodné snímky upravit před samotným zpracováním. V našem případě použijeme pouze základní způsoby předzpracování dat.

Škálování

Vstupem neuronové sítě je fixní počet parametrů, proto budeme chtít mít všechny vstupní obrázky ve stejné velikosti. Nabízí se obrázek škálovat na požadované rozměry. Porušením poměru mezi délkou a výškou rozměrů obrázku by docházelo k deformaci vlastního obsahu a to by mělo negativní vliv na schopnost rozpoznávání obsahu. Proto ještě před škálováním provedeme normalizaci rozměrů (viz centrování snímku). Následně škálujeme obrázek na požadovanou velikost. Pro zachování co nejvíce informací jsme použili maximální velikost v datasetu, kde ke škálování nedojde. V případě datasetu `Corel-10k` jsme použili velikost $80 \times 80px$ (nejmenší strana obrázku).

Centrování snímku

Abychom získali vždy stejný poměr délky a výšky snímku, je potřeba vstupní obrázek oříznout, nebo doplnit. Protože by doplňování přineslo velké prázdné sekce na okrajích obrázku, především při srovnání horizontálně a vertikálně orientovaných obrázků na jeden rozměr, a tím by mohlo dojít k ovlivnění rozpoznávání, rozhodl jsem se pro oříznutí na čtverec.

Obrázek 6.4: Centrování obrázku

Samozřejmě by bylo lepší nejprve určit jakou část obrázku ponechat, ale z důvodu jednoduchosti pouze ořízneme střed obrázku dle jeho kratší strany, jak se znázorněno na obr (viz 6.4). Obsah obrázku mimo zvýrazněnou oblast zahazujeme. Zjišťování co je na obrázku důležité je velmi náročná úloha a tímto způsobem budeme alespoň konzistentní.

Centrování pixelu

Preferované hodnoty pro učení, zejména pro námi použitý framework `Keras` (viz 7.1), jsou v intervalu $< 0, 1 >$. Provedeme tedy nejprve normalizaci do tohoto intervalu. V ideálním případě by průměrná hodnota každé barevné složky byla $0, 5$ (polovina intervalu). To ale obecně nemusí být pravda. Je možné normalizovat pomocí vztahu:

$$S = S - S_{avg} + 0, 5 \tag{6.1}$$

kde S je barevná složka pixelu a S_{avg} je střední hodnota složky. Normalizovat můžeme v rámci každého snímku, dávky při trénování, nebo celého datasetu. V našem případě jsme se rozhodli pro normalizaci podle datasetu. Je tedy potřeba nejprve zjistit střední hodnoty každé složky ze všech obrázků. Tyto hodnoty budou použité jako konstanty v nastavení programu.

Dále je možné vypočítat kovariační matici a provést dekorelaci. Tento proces se nazývá `pixel whitenning`. Tímto procesem bychom zajistili efektivnější učení pomocí zpětné propagace[11]. V našem případě se spokojíme pouze s vycentrováním barevných složek.

Běžně se centrování dělá na průměrnou hodnotu 0. To ovšem v našem případě nelze pokud používáme aktivační funkci `ReLu` (viz 3.1.1) a ztrátovou funkci `Binary cross entropy` (viz 7.4.3). `ReLu` záporné hodnoty ořezává a `BCE` pro ně není určená.

6.2.3 Trénování

V první fázi je třeba natrénovat sestavený autoenkoder. Je vhodné rozdělit trénování na dávky. Není to nutné, pokud se celý dataset vejde do paměti RAM najednou, ale obecně je vždy lepší s tímto problémem počítat a buď využít jíž implementovaný generátor z použitého framworku, nebo implementovat vlastní. Námi použitý framework `Keras` poskytuje generátor, který snadno načítá soubory dle souborové struktury na disku a poskytuje základní možnosti předzpracování a augmentace snímků.

Pokud by bylo třeba provádět náročnější operace, bylo by vhodnější nejprve předzpracovat celý dataset a načítat jíž připravené obrázky. V závislosti

na předzpracování by mohlo dojít k značnému urychlení při trénování. Protože neprovádíme žádné náročné operace, předzpracování probíhá vždy při načtení obrázku. V našem případě jsem implementoval generátor vlastní pro snazší kontrolu předzpracování. Při trénování je vhodné procházet dataset náhodně a nikoliv postupně. Postupný průchod by mohl ovlivnit schopnost učení, kdy by se jednotlivé třídy procházely postupně a nikoliv najednou. Z důvodu testování jsem zavedl *seed* pro generování náhodných čísel. Je tedy možné dospět ke stejným výsledkům, co se generování pořadí týče.

Vlastní proces trénování tedy probíhá vždy po dávkách. Dále jsem zajistil, že pro každou epochu trénování se projde celý dataset. Pomůžeme tím konvergenci k nějakému výsledku. Trénování na části datasetu je možné, ale obecně se nedoporučuje. Každá epocha by se mohla učit na jiných vstupech a mohla by se tím zhoršit konvergence sítě. Celkově u trénování nám jde o co možná nejvěrnější vykreslení vstupního obrázku na výstupu (není to samozřejmě jedinou podmínkou vzhledem k potřebě zobecňovat). V takovém případě se síť skutečně učí rozpoznávat vlastnosti obrázku.

6.2.4 Vytvoření modelu

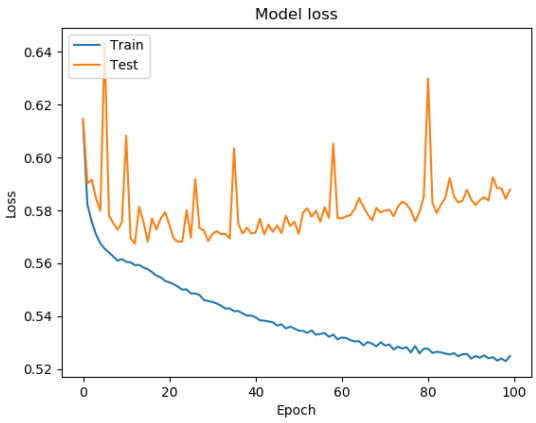
Po dostatečném natrénování autoenkoderu budeme využívat pouze kódovací část celé sítě (viz obr. 3.6 - kódování). Je třeba získat příznakové vektory ke všem položkám datasetu a vytvořit tak jeho model, který využijeme při hledání podobných snímků. Příznakový vektor každého snímku získáme pomocí kódovací části autoenkoderu. Takový model je možné v praxi dále rozšiřovat o další reprezentace s rostoucím datasetem.

6.2.5 Testování

Po vytvoření modelu můžeme přejít k testování. Zde budeme chtít projít celou testovací část datasetu a vyhledat a zhodnotit podobné obrázky ke každé testovací položce. Nejprve vytvoříme příznakový vektor k testované položce a z modelu získáme všechny podobné obrázky tak, že seřadíme všechny uložené vektory modelu podle vzdálenosti k testovacímu. K výpočtu vzdálenosti lze použít různé způsoby. V našem případě jsem použil Euklidovu vzdálenost dvou vektorů. Dále provedeme hodnocení seřazeného seznamu, zda jsou obrázky ze stejné třídy a provedeme výpočet metrik (viz 5).

Pro průchod testovací části datasetu můžeme opět využít generátor. Protože už ale nedochází k učení (ovlivnění modelu) na pořadí nezáleží. Projdeme tedy celou testovací množinu a zobrazíme výsledky.

V průběhu trénování dochází také k testování po každé epoše. Opět nedochází k ovlivnění modelu a jedná se pouze o informaci, která nám pomůže například určit jestli už nedochází k přetrénování, nebo špatnému zobecňování problému. Například v případě přetrénování, kdy autoenkoder velmi dobře replikuje trénovací obrázky, ale nedokáže zobecňovat a úspěšnost na testovací množině zůstává stejná, nebo se zhoršuje. Taková situace je vidět v extrémním případě na obrázku 6.5.



Obrázek 6.5: Ukázka extrémního případu přetrénování

Faktor učení

Zásadní vliv na rychlost a úspěšnost fáze trénování má rychlost učení tzv. `learning rate`. Tato hodnota udává jak moc se budou upravovat váhy neuronů v síti. Při vyšších hodnotách se bude síť učit rychleji, avšak kvůli velkým změnám nemusí nutně konvergovat k požadovanému výsledku. Naopak s malými hodnotami bude učení probíhat značně déle a může se stát že síť uvízne v lokálním minimu a opět se nedostaneme ke správnému výsledku. Obvykle se tedy nepoužívá statická hodnota v rámci celého trénování, ale hodnotu upravujeme za běhu. Obecně budeme chtít začít s hodnotami většími a postupně je budeme snižovat a tím zpřesňovat celé nastavení sítě (váhy neuronů).

Existuje mnoho způsobů jak se vypořádat se správnou volbou `learning rate` jako stochastický přístup, naplánování předvolených hodnot, pokročilé algoritmy, které tuto hodnotu upravují za běhu a mnoho dalších. Použitý framework `Keras` podporuje všechny zmíněné s velkým množstvím nastavení. Velmi oblíbený a často používaný je algoritmus optimalizace nazvaný `Adam` (Adaptive Movement estimation)[9]. Tento algoritmus kombinuje vlastnosti `AdaGrad` (Adaptive Gradient algorithm) a `RMSProp` (Root Mean Square Propagation).

V rámci experimentů, v domnění že by síť mohla uvízlá v lokálním minimu, jsem vyzkoušel ručně upravovat rychlost učení. Ve všech pokusech jsem nikdy nedosáhl lepších výsledků, konvergence. Dobu trénování jsem

pouze zhoršil. `Adam` tedy vždy poskytoval lepší výsledek za kratší dobu. Ve všech dalších experimentech jsem využil výhradně tento algoritmus s výchozím nastavením.

7 Experimenty

Experimentováno bylo na dvou strojích a to osobní notebook a desktopa a proběhly neúspěšné pokusy o zprovoznění aplikace na metacentru (`metavo.metacentrum.cz`).

7.1 SW vybavení

Pro aplikaci jsme využili více běžně používaných knihoven, ale nejdůležitější součástí je framework pro tvorbu a práci s neuoronovými sítěmi `Keras`, který je volně dostupný jako open-source[1]. A jako *backend* pro `Keras` jsme použili `Tensorflow`[2], který je také open-source. Projekt je tedy vytvořený v programovacím jazyce `python`[3] a všechny součásti jsou volně k dispozici. Neměl by tedy být problém s tímto projektem dále pracovat.

Původní záměr byl využít knihovnu `Microsoft CNTK 2.0` a projekt psát v jazyce `C#` se kterým mám více zkušeností. Bohužel k této knihovně neexistuje tolik dostupných materiálů jako k výše zmíněnému frameworku `Keras` a protože se jedná o první projekt tohoto typu, zvolil jsem lépe dokumentovanou a ověřenou volbu tvorby neuronových sítí.

Aplikace byla vyvíjena a je určená primárně pro platformu `Windows`, ale například v případě autoenkoderu je možné, po úpravě cest, aplikaci spouštět i na platformě `Linux` (viz A). Stolní počítač s výkonnější grafickou kartou, který běží na operačním systému `Linux`, byl použitý primárně pro samotný výpočet s využitím vzdáleného připojení `SSH`.

7.2 HW vybavení

Původní plán byl pro trénování využít metacentrum. Bohužel při zprovozňování aplikace na metacentru jsem narazil na spoustu problémů s různými verzemi knihoven a ani po instalaci vlastních se nepodařilo všechny problémy vyřešit. Využití metacentra je samozřejmě možné, ale bylo by dobré s tím počítat od začátku a sestavit softwarové vybavení tak, aby odpovídalo připravenému prostředí ve výpočetním centru.

[1] https://keras.io/
[2] https://www.tensorflow.org/
[3] https://www.python.org/

Osobní notebook byl vybaven CPU `Intel Core i5-6198 DU 2,8GHz`, grafickou kartou `GeForce 940MX 2048 MB VRAM` a `11,9 GB RAM`.

Stolní počítač byl vybaven CPU `i7-4930K CPU 3.40GHz`, grafickou kartou `GeForce RTX 2080 Ti 11GB VRAM` a 64 GB RAM.

7.3 Color difference histogram

Pro srovnání jsem využil tento nový způsob popisu obrázku a vyzkoušel ho na obou používaných datasetech. Jasnou výhodou této metody je absence jakéhokoliv učení. Stačí tedy pouze vytvořit model (sadu popisů) nad datasetem a je možné začít rovnou s modelem pracovat.

7.3.1 Dosažené výsledky

Přesto, že se nám nepodařilo replikovat vynikající výsledky z původního článku[12], kde autoři představují hodnoty přesnosti na datasetu `Corel-10k` až 42% a `recall` 5%. V našich experimentech jsme dosáhli přesnosti $31,4\%$ a `recall` $3,344\%$. Metoda přesto poskytuje kvalitní popis a odpovídá tomu i vysoká `Mean Average Precision` viz tabulka 7.1.

Dataset	MAP	AP	Recall
$Corel-10k$	$88,7\%$	$31,4\%$	$3,344\%$
$Caltech-101$	$30,1\%$	$17,1\%$	$1,890\%$

Tabulka 7.1: Dosažené výsledky

Tato metoda byla vytvořena pro sadu `Corel-10k` a navzdory dobrým výsledkům je její efektivita na druhém datasetu značně nižší. Domnívám se, že to je způsobeno větším rozměrem obrázků a náročnějším obsahem (viz 7.5).

7.3.2 Srovnání barevných prostorů

Důležitou součástí metody je převod do barevného prostoru `Lab`, který by měl lépe reprezentovat vnímání člověka a tím i podobnost obrázků a proto jsem chtěl tento prostor srovnat s klasickým `RGB`. Překvapivě se ukázalo, že `RGB` poskytuje lepší přesnost, jak je vidět v tabulce 7.2. To ovšem odporuje výsledkům, které jsou v článku uvedeny.

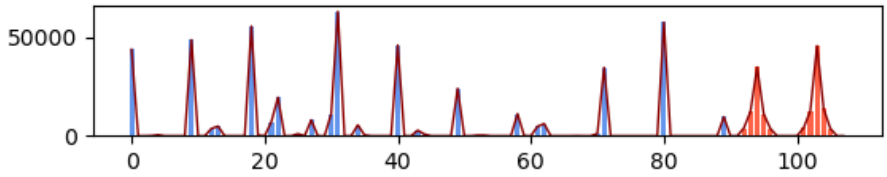
Při zkoumání vytvářených histogramů jsem si již při implementaci všiml, že se nevyužívá celé spektrum hodnot pro prostor `Lab`. `Lab` je značně větší

Dataset	Prostor	MAP	AP	Recall
$Corel-10k$	Lab	$42,7\%$	$23,1\%$	$2,431\%$
$Corel-10k$	RGB	$44,0\%$	$28,7\%$	$3,146\%$

Tabulka 7.2: Srovnání RGB a Lab

prostor než `RGB`. Převodem tedy vznikají *prázdná místa*. Zastoupení hodnot jednotlivých pixelů snímků z použitých datasetů se po převodu do `Lab`, řídí přibližně normálním rozdělením. Ostatně to je jeden z principů používaný při komprimaci (např. jpg), kdy *ořezáváme* hodnoty, kterým člověk přikládá menší váhu.

Domnívám se, že omezením barevného prostoru před přiřazováním by došlo k lepšímu využití příznakového vektoru. Tato skutečnost by mohla být jednou z příčin lepších výsledků s využitím `RGB`. V článku ale takové omezení zmíněné není. Při vytváření příznakového vektoru používáme indexy *bucketů* jednotlivých barevných složek (`L, a, b`), jako souřadnice krychle tvořené devadesáti ($10 \times 3 \times 3$) hodnotami (barevná část příznakového vektoru). Zde se projevuje nerovnoměrné využití barevného spektra periodickými poklesy hodnot histogramu, jak je vidět na obrázku 7.1.



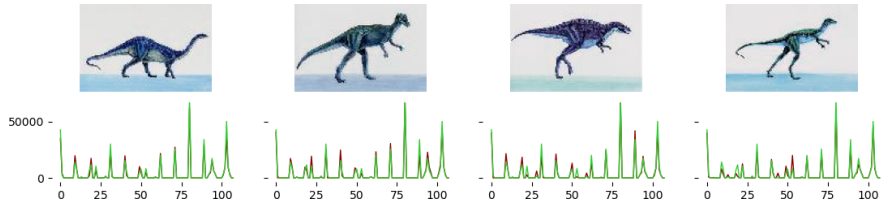
modrá - chromatická část, červená - hranová část

Obrázek 7.1: Reprezentace snímku `CDH`

Oproti článku, kde byla použita vlastní metrika vytvořená přímo pro tuto metodu, jsme použili k hledání nejpodobnějších obrázků Euklidovu vzdálenost a to právě kvůli srovnání s ostatními metodami. Při porovnání histogramů vrácených snímků je však vidět, že jsou si skutečně podobné viz obrázek 7.2.

7.4 Konvoluční autoenkoder

Naměřené hodnoty a experimenty s neuronovou sítí probíhaly především nad datasetem `Corel-10k`, který jako menší z dvou použitých umožňoval rychlejší trénování.



zelená - požadovaný snímek, červená - získaný snímek

Obrázek 7.2: Porovnání histogramů získaných snímků

7.4.1 Ověření konceptu

První testování tzv. *proof of concept* probíhalo na osobním notebooku s procesorem `Intel Core i5-6198 DU 2,8GHz` na ukázkovém modelu autoenkoderu popsaném přímo na stránkách `Kerasu`[4]. Model se trénoval nad datasetem `MNIST` (viz 4). Podle očekávání retrieval zde fungoval velmi dobře. Je zde jen 9 velmi jasně definovaných tříd s jedním barevným kanálem s velkým počtem snímků.

7.4.2 Využití CPU a GPU

Testování na `MNIST` datasetu ukázalo, že pouze s využitím procesoru při trénování jedné epochy (průchod všemi položkami trénovací množiny) zabere průměrně $5:41s$ při použití velmi malé sítě inspirované ukázkou z dokumentace `Kerasu`. Síť měla $1,6M$ parametrů. Stejná síť s využitím GPU, které `Keras` i `Tensorflow` samozřejmě podporuje dokáže zpracovat jednu epochu pouze za $1:29s$. Mluvíme tedy o téměř čtyřnásobném urychlení s využitím nepříliš výkonné mobilní GPU (GTX 940MX).

Rychlost trénování se dále značně zvýšila při testování na zvolených datových sadách `Corel` a `Caltech`, které obsahují obrázky s větším rozlišením, ale nejsou tak rozsáhlé počtem položek. S využitím stolního počítače s moderní grafickou kartou se dostáváme na čas trénování jedné epochy do řádu sekund. Trénování epochy sítě s $6,3M$ parametrů nad datovou sadou `Corel` průměrně zabralo $4s$. Pro všechny následující experimenty jsem tedy využíval pouze GPU desktopu.

Při testování sítě je potřeba zjistit vzdálenosti ke všem vektorům v modelu a vyhodnotit výsledky. Tento proces už běží čistě na CPU a kvůli velikosti našich sítí a rozsahu datových sad často trval podobně dlouho jako samotné trénování sítě. Zde se ještě nabízí možnost urychlení procesu použitím dávkového získání příznakových vektorů z kodéru, ale obecně neoče-

[4] https://blog.keras.io/building-autoencoders-in-keras.html

káváme zpracovávání rozsáhlých testovacích dat najednou. Tato funkce tedy není implementovaná.

7.4.3 Ztrátová funkce

Ztrátová funkce (`loss`) může být měřena různými způsoby. Obvykle se používá `mean squared error`, kterou lze snadno spočítat dle vztahu:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(A_i - B_i)^2 \tag{7.1}$$

kde A je vzorový vektor a B vektor rekonstruovaný. Trénováním se tedy snažíme minimalizovat tuto hodnotu. Další oblíbený způsob zjišťování odlišnosti dvou obrázků (vektorů) je použitím `binnary crossentropy`, která je založena na `Kullback-Leibler Divergence`[5]. Jedná se o míru rozdílnosti. Oba způsoby poskytují podobnou informaci a pracujeme s nimi identicky. Vždy se snažíme jejich hodnotu minimalizovat. Hodnoty `BCE` jsou výrazně vyšší. Neznamená to ale že by celkový model byl horší (nemůžeme přímo porovnávat hodnoty funkcí `MSE` a `BCE`).

Ztrátová funkce	Počet filtrů	MAP	AP	Recall
BCE	$16, 8, 8$	$30,1\%$	$17,1\%$	$1,890\%$
MSE	$16, 8, 8$	$30,1\%$	$16,5\%$	$1,825\%$
BCE	$8, 16, 32$	$31,6\%$	$17,9\%$	$1,983\%$
MSE	$8, 16, 32$	$31,2\%$	$17,8\%$	$1,971\%$

Tabulka 7.3: Ztrátové funkce

V našich experimentech obě míry poskytovali velmi podobný výsledek celkové sítě. Obě metriky zajistili konvergenci k velmi podobnému modelu. Avšak `BCE` poskytovala vždy nepatrně lepší výsledky, jak je vidět v tabulce 7.3.

7.4.4 Hloubka sítě

Naším předpokladem pro zobecnění sítě z rozpoznávání psaných číslic na obecné obrázky bylo prohloubení a rozšíření sítě. Experimentálně jsme tedy prohloubili strukturu sítě. Následující tabulka obsahuje přehled zkoumaných konfigurací. Všechny sítě mají šířku kódové části 128 a `stride` nastavený na

[5]https://deep-and-shallow.com/2020/01/09/deep-learning-and-information-theory/

2. Tedy všechny konvoluční filtry se posouvají o dva pixely v každém kroku. Není zde použitý `MaxPooling` (viz 7.4.5).

Každá vrstva využívá map z vrstev předchozích a tedy s hloubkou sítě se zvyšuje schopnost sítě učit se komplexnější charakteristiky snímku. Takový efekt je žádaný, ale pouze do nějaké míry. Síť se stává náchylnější k přetrénování. To znamená, že se při trénovací fázi síť učí lépe reprezentovat trénovací data, ale ztrácí schopnost zobecňování (viz obrázek 6.5). Přesto, že se tedy síť zlepšuje, její efektivita na neznámých snímcích (testovací množina) klesá. Pro natrénování komplexnějšího rozpoznávání je také potřeba větší množství dat. To je obecnou nevýhodou neuronových sítí.

Parametrů	Počet filtrů	MAP	AP	Recall
$200k$	$16, 8, 8$	$30,1\%$	$17,1\%$	$1,890\%$
$800k$	$128, 64, 32$	$31,2\%$	$17,7\%$	$1,963\%$
$2,4M$	$256, 128, 64$	$30,1\%$	$17,1\%$	$1,893\%$
$800k$	$8, 16, 32$	$31,6\%$	$17,9\%$	$1,983\%$
$1,7M$	$16, 32, 64$	$30,1\%$	$17,3\%$	$1,916\%$
$3,6M$	$32, 64, 128$	$29,5\%$	$16,7\%$	$1,846\%$
$1M$	$256, 128, 64, 32$	$31,7\%$	$17,9\%$	$1,983\%$
$500k$	$8, 16, 32, 64$	$30,4\%$	$17,2\%$	$1,911\%$
$2M$	$32, 32, 64, 128$	$30,5\%$	$17,2\%$	$1,902\%$

Tabulka 7.4: Hloubka a počet filtrů

Z úvahy o počtu filtrů (viz 6.2.1) bychom předpokládali, že rostoucí počet filtrů bude mít lepší výsledky, ale z našich experimentů je vidět, že ani jeden z přístupů není jasně lepší. Vzhledem k hloubce naší sítě a relativně slabším výsledkům, na zvoleném přístupu příliš nezáleží. Rozdíly by se mohly projevit na větších sítích.

7.4.5 Posun filtru

Při posunu filtrů o jeden pixel nedochází ke snižování dimensionality, pokud jsme nezařadili `MaxPooling` (viz 3.1.1), a počet parametrů tedy rapidně narůstá. Ke zúžení samozřejmě musí nakonec dojít v *úzkém hrdle* autoenkoderu. Výsledkem je sít která kvůli minimálním ztrátám informace má možnost věrněji replikovat vstupní snímek.

Posun filtru	Počet parametrů	Počet filtrů	MAP	AP	Recall
2	$800k$	$8, 16, 32$	$31,6\%$	$17,9\%$	$1,983\%$
1	$52M$	$8, 16, 32$	$27,2\%$	$15,3\%$	$1,697\%$

Tabulka 7.5: Posun filtru

Hlavní nevýhodou tohoto přístupu je velký nárůst parametrů u hrdla autoenkoderu (viz tabulka 7.5), kde se musí parametry z mnoha map (ze všech filtrů) navázat do, v našem případě, plně propojené vrstvy o délce požadovaného příznakového vektoru. K žádnému zlepšení schopnosti sítě nedošlo. Předpokládám, že pro tak velký počet vstupních parametrů máme k dispozici příliš malé datové sady.

7.4.6 Velikost filtru

S větší velikostí filtru má síť možnost reagovat na větší prvky snímku, nicméně nemá smysl velikost filtru příliš zvětšovat, uvážíme-li, že obrázky obvykle upravujeme na rozměr $80 \times 80px$. Po jedné vrstvě `MaxPooling`, nebo při posunu filtru o dva pixely už v další vrstvě pracujeme s polovičním rozměrem. Větší filtry tedy rychle ztrácí svoji funkci.

Ztrátová funkce	Počet filtrů	MAP	AP	Recall
$3x3$	$8, 16, 32$	$31,6\%$	$17,9\%$	$1,983\%$
$5x5$	$8, 16, 32$	$30,7\%$	$17,9\%$	$1,981\%$
$9x9$	$8, 16, 32$	$30,5\%$	$17,6\%$	$1,951\%$

Tabulka 7.6: Velikost filtru

V našich experimentech se žádné výrazné změny neprojevily, ale podle tabulky 7.6 je patrné že schopnost sítě s velikostí filtru klesá.

7.4.7 Velikost příznakového vektoru

V předchozích experimentech jsme zkoušeli kódovat obrázky pomocí příznakového vektoru o délce 128 jako nejmenšího rozumného popisu. Při použití menších se značně zhoršovala kvalita replikovaného obrázku z autoenkoderu. Použitím menšího popisu (užšího hrdla autoenkoderu) jsme se pokoušeli donutit síť k hledání skutečně důležitých informací ve snímku.

Důležité je nepřesáhnout délkou příznakového vektoru šířku vstupu. V případě přiblížení nebo překročení této hodnoty se síť samozřejmě bude pouze učit, jak hodnoty zkopírovat na výstup. Bez žádné snahy o hledání

důležitých vlastností bude autoenkoder výborně replikovat vstup, ale nebude schopen žádného zobecňování a tím i užitečné reprezentace.

Popisu	Parametrů	Počet filtrů	MAP	AP	Recall
128	$800k$	$128, 64, 32$	$31,2\%$	$17,7\%$	$1,963\%$
256	$1,6M$	$128, 64, 32$	$27,4\%$	$15,9\%$	$1,772\%$
128	$800k$	$8, 16, 32$	$31,6\%$	$17,9\%$	$1,983\%$
256	$1,6M$	$8, 16, 32$	$27,2\%$	$15,9\%$	$1,772\%$
128	$1M$	$256, 128, 64, 32$	$31,7\%$	$17,9\%$	$1,983\%$
256	$1,2M$	$256, 128, 64, 32$	$28,0\%$	$16,1\%$	$1,793\%$

Tabulka 7.7: Hloubka a počet filtrů

Při zdvojnásobení hrdla autoenkoderu, rychle klesá úspěšnost celé sítě viz tabulka 7.7. To je pravděpodobně způsobeno tím, že síť není nucena k vysoké kompresi vstupu a nepotřebuje se tedy učit důležitější vlastnosti snímku pro jeho rekonstrukci.

7.4.8 Další experimenty

MaxPooling a posun filtru

V podstatě se jedná o mechanizmus snižování dimensionality snímku. Využitím `MaxPooling`, jak bylo popsáno výše (viz 3.5)), docílíme zmenšení výběrem maximální hodnoty. V případě posunu filtru k výběru nedochází, ale protože při posunu o $2px$ přeskakujeme každý druhý pixel, dostáváme podobný efekt. Při porovnání obou způsobů jsem dosahoval velmi podobných výsledků. Snižování (v případě dekodéru zvyšování `UpScaling`) dimensionality má za následek ztrátu informace a tím i *rozmazanější* rekonstrukci obrázku. Na druhou stranu se zvyšuje schopnost zobecňování sítě.

Existují další alternativy těchto metod. Vyzkoušel jsem použití transponované konvoluční vrstvy v dekodéru místo kombinace konvoluční vrstvy následované vrstvou `UpScaling`. Nezaznamenal jsem ale žádné znatelné změny v celkové úspěšnosti sítě.

Plně propojená vrstva v autoenkoderu

Často v autoenkoderu zavádíme plně propojenou vrstvu na konci kódovací části po vzoru konvolučních klasifikačních sítí. Tato vrstva nám mimo jiné

usnadňuje práci při návrhu, kdy si můžeme přesně zvolit délku kódového slova počtem neuronů (viz Automatické generování struktury). Protože každý neuron v plně propojené vrstvě musí být navázaný na každý neuron z vrstvy předchozí, značně zde naroste počet parametrů sítě.

Popis	Parametrů	Počet filtrů	MAP	AP	Recall
800	$5k$	$16, 8, 8$	$33,3\%$	$20,1\%$	$2,222\%$
800	$100k$	$16, 8, 8$	$39,1\%$	$24,1\%$	$2,639\%$
400	$200k$	$128, 64, 32, 16$	$40,0\%$	$24,1\%$	$2,636\%$
128	$400k$	$32, 64, 128, 64, 32$	$38,5\%$	$23,2\%$	$2,532\%$
800	$800k$	$256, 128, 64, 32$	$38,5\%$	$23,2\%$	$2,532\%$

Tabulka 7.8: Sítě bez plně propojené vrstvy kodéru

Z našich experimentů se ukazuje že není vhodné mít většinu parametrů právě v této vrstvě. Parametry filtrů, které jsou v menšině, ztrácejí na svém významu a síť dosahuje nižších výsledků. Odebráním této plně propojené vrstvy získáváme lepší výsledky, jak je vidět v tabulce 7.8. Ztrácíme tím ale flexibilitu automatického generování struktury sítě (viz Automatické generování struktury).

Automatické generování struktury

Při vytváření autoenkoderu chceme dodržet symetrii kódovací a dekódovací části. Můžeme tedy síť snadno vytvářet zadáním parametrů vrstev a automaticky vygenerovat celou strukturu. To se velmi osvědčilo při experimentech. Problém vzniká při `Poolingu`, kdy například z mapy rozměrů 5×5 získáme mapu 3×3. V dekódovací části `UpScaling` vytvoří mapu 6×6. Tento problém vzniklý *zaokrouhlením* způsobí, že dekodér vrací snímek jiných rozměrů než na vstupu autoenkoderu.

To samozřejmě není možné a pokud takovou síť chceme vytvořit, musíme ručně upravit strukturu například změnou parametrů některé vrstvy jako posun filtru, nastavením `padding` (chování filtru v krajních bodech mapy), atd. Nelze tedy takovou strukturu generovat a je třeba ji ručně upravovat.

Generování může tedy posloužit pouze pro hrubé vyzkoušení mnoha různých struktur autoenkoderu pro daný problém a další úpravy je třeba provádět ručně.

Porovnání s existující sítí

V článku `CBIR for Breast Ultrasound Images`[5] je použitý autoenkoder pro získání relevantních ultrazvukových snímků obrázků rakoviny prsu. Smyslem projektu bylo zlepšit diagnózu tohoto onemocnění. My jsme vyzkoušeli replikovat strukturu použité sítě pro náš obecný `CBIR` na datasetu `Corel-10k`.

Příznakový vektor	Parametrů	Počet filtrů	MAP	AP	Recall
32	$50k$	$43, 22, 11, 6$	$26,6\%$	$14,7\%$	$1,583\%$

Tabulka 7.9: Síť pro vyhledávání podobných ultrazvukových snímků

Jak je vidět v tabulce 7.9, síť nedosáhla příliš dobrých výsledků. Síť byla samozřejmě navržena pro černobílé snímky o větších rozměrech. Museli jsme tedy naše obrázky zvětšit, což také mohlo přispět k horším výsledkům.

7.5 Porovnání

Jedním z velkých problémů úloh `CBIR` je ověření správnosti algoritmu, nebo konkrétní specifikace co od úlohy očekáváme. V rámci datových sad se jedná především o určení podobnosti obrázků. Obvykle využíváme datasetů původně vytvořených pro klasifikaci a předpokládáme, že pouze obrázky ve stejné třídě si jsou podobné. Na tomto předpokladu byly založené i testované metody. Tento předpoklad často není pravdou. Naše metody a především neuronové sítě, se učí reflektovat toto rozdělení, nikoliv skutečné vnímání člověka.

Vytvoření dobré datové sady pro úlohy obecného `CBIR` je velmi obtížný problém. Snadněji získáme takový dataset, omezením úlohy na nějaký konkrétní problém, jako například hledání podobných rentgenů jednotlivých částí těla, vyhledávání podobného uměleckého stylu obrazů, znaků, číslic atd. To ovšem není možné pokud chceme obecně získávat podobné obrázky. Tento problém se silně projevuje u neuronových sítí, které vyžadují obrovské množství dat pro jejich správné natrénování.

Kvůli výše zmíněnému problému často vznikají různé verze jednotlivých datasetů, které lépe nahrávají aktuálně řešenému problému a je tedy velmi obtížné porovnávat skutečnou efektivitu různých metod, protože naměřené hodnoty mohou být velmi rozdílné od hodnot naměřených na zdánlivě podobných datech, jak tomu bylo například u metody `CDH`.

Obě metody mají svoje výhody, ale klasický přístup `CDH` překvapil svojí vysokou úspěšností. V původním článku jsou uvedeny mnohem lepší vý-

sledky, kterých se mi nepodařilo dosáhnout a to především přikládám faktu, že CDH byl testován na datasetu Corel-10k který, jak by se mohlo na první pohled zdát, ve skutečnosti není volně dostupným datasetem Corel-10k použitým v našich experimentech. V původním článku je poznámka, že se jedná o ručně vybranou podmnožinu původního datasetu Corel-200k o rozsahu $10k$ obrázků. Název je tedy mírně zavádějící.

Z pohledu použití je metoda CDH snazší na nasazení, protože nevyžaduje žádnou zvláštní přípravu. Avšak vytvoření modelu nad datasetem zabere nezanedbatelnou dobu. Samotné vytváření je pochopitelně z principu značně pomalejší, než získání popisu z natrénované sítě. Tento proces by se dal urychlit paralelizací. Interpret pythonu neumožňuje běh na více jádrech procesoru a přesto, že se dá dosáhnout paralelizace paralelním během různých procesů, nemusela by tato úloha být triviální. V našem případě na osobním notebooku (viz 7.2) se jedná o zpracování v řádu dnů. Například vytvoření modelu nad datasetem Caltech trvalo přibližně $50h$.

Trénování je jedna z nevýhod využití neuronových sítí, ale s využitím GPU se celý proces značně urychlil. V našem případě, kdy je síť relativně malá, trénování nebyl žádný problém. Po natrénování je naopak neuronová síť velice rychlá při vytváření popisu snímku, protože se jedná o lineární operaci.

Přístup pomocí neuronové sítě se zdá být více škálovatelný. Pro menší datové sady jsou stále klasické metody výhodnější především proto, že nepotřebují velké množství dat pro trénování.

Dataset	Metoda	MAP	AP	Recall
$Corel-10k$	CAE	$40,0\%$	$24,1\%$	$2,636\%$
$Corel-10k$	CDH	$42,7\%$	$23,1\%$	$2,430\%$
$Caltech-101$	CAE	$29,1\%$	$15,3\%$	$2,891\%$
$Caltech-101$	CDH	$20,2\%$	$10,9\%$	$1,885\%$

Tabulka 7.10: Porovnání metod

V tabulce je vidět že obě metody měly problém s náročnější datovou sadou Caltech-101, kde metoda CDH dosáhla relativně nízké přesnosti. A dokonce ji předstihla i naše síť. CDH byla navržena pro data prvního zmíněného datasetu. Pro využití na jiných by bylo pravděpodobně třeba upravit parametry metody, nebo použít obrázky stejných rozměrů.

Námi testovaný přístup s využitím autoenkoderu nedosáhl příliš dobrých výsledků. Očekával jsme výrazně vyšší úspěšnost než u výše zmíněné metody, i když se jedná se o velmi jednoduchou strukturu. Na datasetu MNIST

dosahoval autoenkoder téměř perfektní úspěšnosti. V našem případě se ale jedná o velmi obecný problém a ukazuje se, že není jednoduché takovou síť zobecnit a úspěšně natrénovat.

8 Závěr

Práce byla zaměřena na vyhledávání obrázků podle obsahu. Nejprve jsem prozkoumal dostupné datasety, které jsou často zaměřené na konkrétní problematiku jako například hrady, jídlo, rentgenové snímky, atd. Protože jsem ale chtěl pracovat s obecnými daty, zaměřil jsem se pouze na `Corel-10k` a `Caltech-101` vzhledem k tomu, že jsou často využívány pro testování a je snazší metody na nich porovnávat. Hlavním problémem pro úlohy `CBIR` je vytvoření datasetu, který by skutečně odpovídal podobnosti obrázků tak, jak je vnímá člověk. Často tedy využíváme datasety určené pro klasifikaci.

Nedílnou součástí `CBIR` je vyhodnocování úspěšnosti. Pro specializované problémy vznikají velmi konkrétní metriky, které se špatně srovnávají s ostatními modely, ale z těch běžně používaných je střední průměrná přesnost `MAP` vhodná právě pro úlohy typu `CBIR`, kde hraje velkou roli pořadí získaných obrázků. V kombinaci s ostatními hodnotami si dokážeme udělat poměrně dobrý obrázek o tom, jak naše řešení funguje.

Dále jsem prozkoumal dostupné metody a vybral dva zajímavé přístupy k problému. Metoda založená na histogramu vykazovala, podle autorů článku, velmi dobré výsledky. Teprve až při implementaci se ukázalo, že výsledky nebude možné replikovat kvůli randomizaci a ručně vytvořenému datasetu, který ve skutečnosti není veřejně dostupný. Přesto jsem ověřili, že metoda funguje dobře i na standardních datových sadách, i když ne na takové úrovni jako bychom čekali podle článku.

Druhá metoda s využitím neuronových sítí je zajímavá kvůli popularitě konvolučních sítí v oblasti rozpoznávání a také kvůli velmi dobrým výsledkům na jednoduchých datových sadách. Nepodařilo najít se mnoho `CBIR` systémů založených čistě na autoenkoderu pro obecnější problémy. Obvykle byl autoenkoder použitý jako součást mnohem většího systému. Vyzkoušel jsem tedy vlastní návrh a implementaci. Jak se ukázalo, návrh takové sítě není jednoduchý a zároveň není ani jednotný názor na to, jak k takovému problému přistupovat. Přesto, že jsem očekával výrazně lepší výsledky, autoenkoder nakonec první metodu překonal alespoň na datasetu `Caltech-101`.

Domnívám se, že hlubšímu porozumění snímku sítí brání především její velikost. Na zvolených datasetech se ale nepodařilo natrénovat síť větší. Hlavním důvodem by mohl být relativně malý dataset (v obou případech přibližně 10 000) v porovnání například s jednoduchým `MNIST` (70 000). Konkrétně tedy malý počet obrázků v každé kategorii. Síť nemá dostatek dat pro to, aby se dokázala naučit komplexnější informace z každé třídy. Jako

druhý problém vidím obecnost úkolu, kdy je třeba používat velké množství různorodých dat, aby byla síť schopná pracovat s jakýmkoliv obrázkem.

Pro lepší výsledky při získávání obecných obrázků by bylo vhodné kombinovat autoenkoder s dalšími metodami rozpoznávání, které by mohly odlehčit nároky kladené na vlastní síť a následným trénováním nad rozsáhlou databází obrázků.

Literatura

[1] Bengio, Y. – Courville, A. – Vincent, P. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2013, 35, 8, s. 1798–1828. Dostupné z: https://arxiv.org/abs/1206.5538.

[2] Biederman, I. Recognition-by-components: a theory of human image understanding. *Psychological review*. 1987, 94 2, s. 115–147.

[3] Chang, C. Image Information Systems. In *Proc. IEEE Pattern Recognition*, 73, s. 754–766, April 1985.

[4] Chen, Y. – Bi, J. – Wang, J. Z. MILES: Multiple-instance learning via embedded instance selection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. 2006, 28, 12, s. 1931–1947.

[5] Cho, H. et al. A similarity study of content-based image retrieval system for breast cancer using decision tree. *Medical physics*. 2013, 40 1.

[6] Fei-Fei, L. – Fergus, R. – Perona, P. One-Shot learning of object categories. In *Pattern Recognition and Machine Intelligence*, 2004.

[7] G., G. – AD., H. – P., P. The Caltech 256. Technical report, California Institute of Technology, 2006.

[8] Jain, A. K. – Vailaya, A. Image retrieval using color and shape. *Pattern Recognit.* 1996, 29, s. 1233–1244.

[9] Kingma, D. P. – Ba, J. Adam: A Method for Stochastic Optimization. *CoRR*. 2015, abs/1412.6980.

[10] Nandanwar, A. Edge Histogram Descriptor , Geometric Moment and Sobel Edge Detector Combined Features Based Object Recognition and Retrieval System. 2016.

[11] LeCun, Y. et al. Efficient BackProp. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, s. 9–50, Berlin, Heidelberg, 1998. Springer-Verlag. ISBN 3540653112.

[12] Liu, G.-H. – Yang, J.-Y. Content-Based Image Retrieval Using Color Difference Histogram. *Pattern Recogn.* January 2013, 46, 1, s. 188–198. ISSN 0031-3203. doi: 10.1016/j.patcog.2012.06.001. Dostupné z: https://doi.org/10.1016/j.patcog.2012.06.001.

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

[14] Powers – W., D. M. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*. 2011, s. 37–63.

[15] Rui, Y. et al. Relevance feedback: A power tool for interactive content-based image retrieval. *IEEE Transactions on Circuits and Systems for Video Technology*. 1998, 8, 5, s. 644–655. ISSN 1051-8215. doi: 10.1109/76.718510.

[16] Russakovsky, O. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*. 2015, 115, 3, s. 211–252. doi: 10.1007/s11263-015-0816-y.

[17] Swain, M. J. – Ballard, D. H. Color indexing. *International Journal of Computer Vision*. Nov 1991, 7, 1, s. 11–32. ISSN 1573-1405. doi: 10.1007/BF00130487. Dostupné z: https://doi.org/10.1007/BF00130487.

[18] Tamura, H. – Mori, S. – Yamawaki, T. Textural Features Corresponding to Visual Perception. *IEEE Transactions on Systems, Man, and Cybernetics*. 1978, 8, s. 460–473.

[19] Wan, J. et al. Deep Learning for Content-Based Image Retrieval: A Comprehensive Study. In *ACM Multimedia*, 2014.

[20] Wythoff, B. J. Backpropagation neural networks: A tutorial. *Chemometrics and Intelligent Laboratory Systems*. 1993, 18, 2, s. 115 – 155. ISSN 0169-7439. doi: https://doi.org/10.1016/0169-7439(93)80052-J. Dostupné z: http://www.sciencedirect.com/science/article/pii/016974399380052J.

[21] Zhang, E. – Zhang, Y. *Average Precision*, s. 192–193. Springer US, Boston, MA, 2009. doi: 10.1007/978-0-387-39940-9˙482. Dostupné z: https://doi.org/10.1007/978-0-387-39940-9_482. ISBN 978-0-387-39940-9.

[22] Zhou, W. – Li, H. – Tian, Q. Recent Advance in Content-based Image Retrieval: A Literature Survey. *CoRR*. 2017, abs/1706.06064. Dostupné z: http://arxiv.org/abs/1706.06064.

A Uživatelská dokumentace

Pro oba programy je potřeba mít nainstalovaný `Python` verze `3.6`

A.1 Color difference histogram

Pro spuštění programu je potřeba mít nainstalované potřebné knihovny. Pro tento úkol je doporučené použít nástroj `Anakonda`, který je volně stažitelný[1], nebo jeho obdobu. Pomocí tohoto nástroje si můžeme vytvořit virtuální prostředí do kterého nainstalujeme všechny potřebné knihovny.

A.1.1 Anakonda

Jedná se o konzolový nástroj pro vytváření a správu virtuálních prostředí. Virtuálním prostředím odstíníme náš projekt od ostatních a zamezíme tak mnoha konfliktům.

Po instalaci je vhodné přidat cestu ke spustitelnému souboru do systémové proměnné `PATH`. Po instalaci můžeme ověřit verzi a funkčnost zadáním příkazu `conda info` do příkazové řádky. Dále pomocí následujících příkazů si připravíme prostředí pro `python`.

`conda info` Informace o instalaci

`conda create --name Histograms python=3.6` Vytvoření virtuálního prostředí `Histograms` s požadovanou verzí `pythonu`.

`conda env list` Výpis vytvořených prostředí.

`conda activate [env]` Aktivace prostředí, kde `[env]` je název prostředí.

`conda install [package]` Instalace balíčku, kde `[package]` je název požadovaného balíčku (např. `opencv`)

`conda env create -f Histograms.yml` Vytvoření prostředí ze souboru `Histograms.yml`

Vytvoříme tedy nové prostředí ze souboru, nebo ručním přidáváním potřebných balíčků. Prostředí je před použitím vždy nutné aktivovat.

[1] www.anaconda.com

A.1.2 Aplikace

Aplikace se skládá z několika částí jako `Main`, `Calc`, atd. Pro práci s programem jsou důležitá pouze `Main.py` jako vstupní bod programu a `Settings.py`, kde můžeme nastavit chování aplikace (viz A.1.3).

Nejprve je třeba upravit nastavení programu podle našich požadavků a následně program spouštíme příkazem `python Main.py`. Po spuštění program vytvoří dataset rozdělený na trénovací a testovací množinu, pokud neexistuje. Následně pokud neexistuje model dojde k jeho vytvoření. Tato operace je časově velmi náročná. V závislosti na nastavení dojde k testování celé testovací množiny, nebo pouze jednoho snímku a zobrazení výsledků. V případě testování jednoho snímku jsou vykresleny další informace a reprezentace získaných obrázků.

Aplikace umožňuje využívat *předpočítaný model*. Jedná se o model, který se vytvoří z celkového datasetu, tedy trénovací i testovací množiny. Smysl spočívá ve vytvoření příznakových vektorů pro všechny snímky. Při testování se tedy pouze porovnávají uložené hodnoty. To značně urychluje práci, protože vytvoření vektoru trvá nezanedbatelně dlouho. Z tohoto důvodu se při vykreslování načítají obrázky ze zdrojového datasetu. Pro samotnou logiku se samozřejmě použijí rozdělené trénovací a testovací množiny.

A.1.3 Nastavení

LEARNING Zda chceme, aby program ukládal zpracovávané snímky do modelu. Typicky při zpracování datasetu.

TEST_BULK Zda má testování proběhnout nad celým testovacím datasetem. Při vypnutí této možnosti dojde k otestování pouze jednoho snímku.

MODEL Název uloženého modelu

USE_LAB Zda se má použít převod do barevného prostoru `Lab`.

TARGET_INDEX Index testovaného obrázku (pouze pro `TEST_BULK = False`). Po načtení modelu proběhne `retrieval` pouze pro jeden obrázek s výpisem dalších informací

SOURCE_DIR Cesta do složky s původním (zdrojovým) datasetem

SOURCE_DIR_DATASET Cesta kam se vytvoří testovací a trénovací množina

PRECALCULATED_MODEL Používat dopředu vypočítaný model

A.2 Konvoluční autoenkoder

Podobně jako pro `CDH` je potřeba nainstalovat všechny potřebné knihovny. Navíc je třeba nainstalovat správné (navzájem kompatibilní) verze `CUDA toolkit`, `cuDNN`, `Keras` a `Tensorflow`. Prostředí je opět možné importovat ze souboru `AutoencoderConv_GPU`. Je nezbytné upravit verze výše zmíněných knihoven tak, aby odpovídali konfiguraci počítače, zejména grafické kartě, jejím ovladačům a podporované verzi `CUDA`. `CUDA toolkit` je třeba nainstalovat samostatně, nejlépe přímo ze stránek `nvidia`[2].

Soubor pro import prostředí je připraven obecně, ale v případě problémů je potřeba upravit verze jednotlivých balíčků. Některé příkazy nejsou v různých verzích podporovány a mohlo by být třeba je upravit přímo v kódu. Bohužel je obtížné připravit aplikaci nezávisle na konfiguraci počítače.

Nejsnazší je řídit se instrukcemi instalace na stránkách `tensorflow`[3] pro `Windows`, nebo `Linux`. Na těchto stránkách je i seznam otestovaných kombinací, ze kterých je dobré vycházet. Pokud bychom potřebovali jinou verzi například pro starší grafickou kartu, je nutné sestavit `tensorflow` ze zdrojových kódů, protože (`tensorflow` je dopředu sestavený pouze od určitě úrovně *compute capability* určené ovladačemi grafické karty. To je samozřejmě značně náročnější.

Pro stažení `cuDNN` (podpora neuronových sítí) je třeba založit bezplatný účet na stránkách `nvidia`. Při stahování dbáme na správnou verzi balíčku a verzi `CUDA`, kterou podporuje naše grafická karta. Po instalaci `cuDNN` je třeba ještě přidat cestu do systémové proměnné `path` podle instrukcí.

Dále je třeba nainstalovat `tensorflow-gpu` pomocí `Anakondy` (viz A.1.1). Pokud budeme chtít využívat pouze CPU stačí instalace `tensorflow` a můžeme ignorovat `CUDA toolkit` a `cuDNN`. Framework `keras` bychom měli instalovat až po `tensorflow`.

Novější verze `tensorflow` obsahují přímo odpovídající verzi `kerasu`. Můžeme také využít snazší možnost instalace kombinace `keras - tensorflow` pomocí příkazu `conda install keras-gpu=[version]` (pouze pro CPU: `conda install keras=[version]`), kde `[version]` je číslo verze.

[2] www.developer.nvidia.com/cuda-toolkit

[3] www.tensorflow.org/install

A.2.1 Aplikace

Obdobně jako pro `CDH` aktivujeme prostředí a nastavíme požadovanou funkčnost v souboru s nastavením `Settings.py`. V nastavení se nachází všechny důležité hodnoty a máme na výběr z několika možností spouštění včetně automatického generování struktury sítě pro číslo struktury `0` (s plně propojenou vrstvou) a `100` (bez plně propojené vrstvy s využitím `MaxPooling`). Struktury a jejich generování se nachází v souboru `CNNStructure.py`.

Program spouštíme příkazem `python Main.py`. Po spuštění se vytvoří testovací a trénovací datové sady ze zdrojového datasetu, pokud neexistují. Dále pokud není nalezen model dojde k jeho založení a spustí se trénovací fáze. Pokud je nastaveno ukládání *checpointů*, dochází k uložení modelů v případě, že se model zlepšil od předchozí nejlepší iterace. Je tedy možné pokračovat po přerušení. Případně je možné použít některý z požadovaných *checkopontů* jako výsledný smazáním přebytečných a úpravou počtu epoch trénování v nastavení. Po natrénování proběhne vytvoření modelu získáním příznakových vektorů všech snímků trénovací množiny.

Dále se podle nastavení vykreslí graf historie učení, zobrazí se ukázka rekonstruovaných obrázků a proběhne testování nad celou testovací množinou, nebo nad několika obrázky s ukázkou výsledků. Na konec pokud je povolené hledání obrázků pro maximální aktivaci filtrů, dojde k jejich generování a vykreslení. Výsledné metriky a obrázky se ukládají do složky s modelem.

A.3 Nastavení

SOURCE_DIR Cesta do složky s původním datasetem

SOURCE_DIR_DATASET Cesta kam se vytvoří testovací a trénovací

MODEL_FOLDER Složka kam bude model a jeho dodatečné soubory ukládány.

MODEL_NAME Název uloženého modelu

SAVE_CHECKPOINTS Zda se mají ukládat *checkpointy* pri učení.

DECODER_SHOW Zda se mají zobrazit rekonstruované obrázky.

RETRIEVAL Zda se má provést testování nad celou testovací množinou.

RETRIEVAL_SHOW Zda se má zobrazit okno s ukázkou vrácených obrázků.

TRAIN_SHOW_HISTORY Zda se má zobrazit historie trénování.

RANDOM_SEED Seed pro generátor náhodných čísel použitý pro rozdělení trénovací a testovací množiny.

PLOT_LAYER_CL_WEIGHT Zda se má program pokusit najít obrázky maximálně aktivující jednotlivé filtry.

STRUCTURE_INDEX Číslo struktury v souboru `CNN_Structure.py`

STRUCTURE_CL_ Nastavení pro automatické generování struktury. (Pouze pro `STRUCTURE_INDEX = 0`)

STRUCTURE_POOLING_KERNEL Nastavení velikosti jádra při `poolingu`. (Pouze pro `STRUCTURE_INDEX = 100`)

PLOT_LAYER_CL_WEIGHT Zda se má aplikace pokusit najít obrázky, které nejvíce aktivují jednotlivé filtry