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

Bakalářská práce

Prohledávání dokumentů podle automaticky extrahovaných vzorů

Plzeň 2018 Josef Baloun

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 29. dubna 2018

Josef Baloun

Poděkování

Tato práce vznikla za podpory projektů CERIT Scientific Cloud (LM2015085) a CESNET (LM2015042) financovaných z programu MŠMT Projekty velkých infrastruktur pro VaVaI.

Poděkování patří také doc. Ing. Pavlu Královi, Ph.D za ochotu a cenné rady během vedení práce.

Abstract

This bachelor thesis deals with the design and implementation of methods that allow the search of handwritten words. A prerequisite is the ready segmentation of the document to the word images. In the first part of the thesis, the available data collections are mapped for searching in handwritten texts, followed by a list of possible solutions among which neural networks are selected. In the second part, experiments are performed on the Parzival data collection. The purpose of the experiments is to verify the functionality of the methods, a search for an appropriate network architecture and to determine the effect of the individual parts of the methods on the result. Three methods based on neural networks are designed to deal with the different sizes of input images in three ways. These methods are tested and compared on the Parzival database where they have achieved excellent results. The best results of 92,62 % MAP QbS and 90,01 % MAP QbE were achieved with a method based on the convolutional neural network and the PHOC representation of the output vector.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací metod, které umožní vyhledávání slov v ručně psaném textu. Předpokladem je již hotová segmentace dokumentu na obrázky slov. V první části práce jsou zmapovány dostupné datové kolekce pro vyhledávání v ručně psaných textech. Následuje seznámení s možnými řešeními, mezi kterými jsou zvoleny neuronové sítě. V druhé části jsou provedeny experimenty na datové kolekci Parzival. Účelem experimentů je ověření funkčnosti metod, hledání vhodné architektury sítě a zjištění vlivu jednotlivých částí metod na výsledek. Navrženy jsou tři metody založené na neuronových sítích, které se dokáží vypořádat s rozdílnou velikostí vstupních obrázků třemi způsoby. Tyto metody jsou otestovány a porovnány na databázi Parzival, kde dosáhli výborných výsledků. Nejlepších výsledků 92,62 % MAP QbS a 90,01 % MAP QbE bylo dosaženo s metodou, která je založena na konvoluční neuronové síti a PHOC reprezentaci výstupního vektoru.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Prohledávání dokumentů podle automaticky extrahovaných vzorů lze také nalézt pod pojmy jako *rozpoznávání* nebo *vyhledávání slov v ručně psaných dokumentech*. V anglické literatuře je nejčastěji použit termín *word spotting*.

Toto prohledávání je děleno podle vstupního kritéria, kterým může být obrazový vzor (dále jako *QbE*) nebo textový řetězec (dále jako *QbS*). Během prohledávání se snažíme na základě vstupního kritéria nalézt obrázky s odpovídajícím slovem. Představme si situaci, kdy má historik najít v kronice, která obsahuje 1 000 stran textu, všechny zmínky o klášteře v Teplé.

Vyřešení tohoto problému by usnadnilo práci při vyhledávání slov ve velkém množství ručně psaných dokumentů, ve kterých historici hledají důležité zmínky o určitém ručně psaném textovém řetězci. Místo toho, aby člověk prohledával dlouhé hodiny svazky dokumentů, by stačilo nechat těžkou práci vykonat počítač a zkontrolovat pouze pravděpodobné výskyty hledaného řetězce.

Cílem této práce je návrh a implementace dvou metod, které umožní vyhledávání slov. Předpokladem je předem provedená segmentace dokumentu na obrázky slov. K dosažení tohoto cíle jsme se mezi možnými řešeními rozhodli použít neuronové sítě, které v dnešní době dosahují skvělých výsledků napříč různými oblastmi umělé inteligence.

Obsah práce je následující. Kapitola 2 studuje dostupné datové kolekce pro vyhledávání v ručně psaných textech. V další kapitole 3 se seznámíme s možnými řešeními a základem neuronových sítí. Kapitola 4 popisuje zvolená řešení a jejich princip. V kapitole 5 následuje seznámení s použitými knihovnami a systémem. Poslední kapitola 6 prezentuje provedené experimenty, dosažené výsledky a jejich rozbor.

2 Databáze

Pro vyhledávání podle vzoru (dále jako *QbE*) i řetězce (dále jako *QbS*) potřebujeme data pro testování konkrétního řešení a také pro trénování vytvořeného modelu.

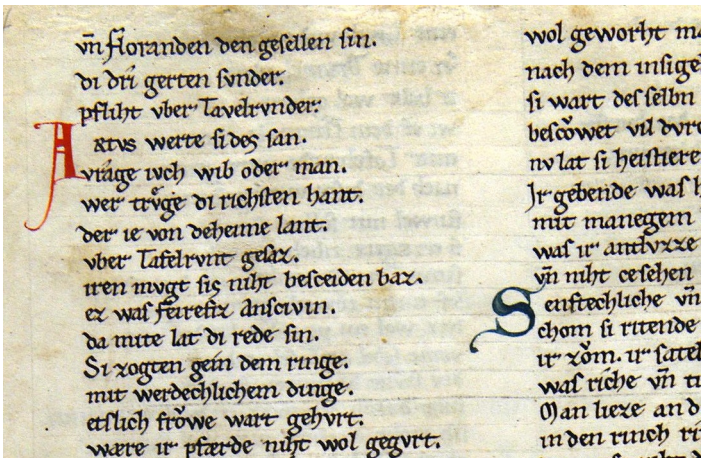
Za tímto účelem je třeba v případě QbE vědět, které obrázky obsahují stejnou informaci, a v případě QbS je nutné znát, jakou informaci (co je napsáno) má konkrétní obrázek. Proto požadujeme, aby data obsahovala obrázky a jejich přepisy v textové podobě. Takto připravená data jsou nutná pro trénování modelů i pro vyhodnocení úspěšnosti vyhledávání.

Takovou databázi můžeme vytvořit, nebo použít některou z již vytvořených databází pro tento problém. Použití vytvořené databáze nám kromě ušetřené práce přinese i další výhody. Protože tyto databáze obvykle obsahují nadefinované sady trénovacích, validačních a testovacích dat, můžeme výsledky našeho řešení porovnat s jinými dostupnými řešeními, které používají totožnou databázi.

Následující databáze splňují naše podmínky a zároveň jsou často používány v článcích, které řeší tento problém.

2.1 Databáze Parzival

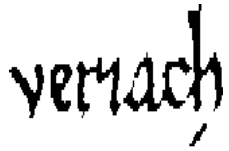
Tato databáze je vytvořena z dokumentu pocházejícího z 13. století, který byl napsán třemi spisovateli, středověkým německým jazykem a gotickým skriptem (náhled viz obrázek 2.1).



Obrázek 2.1: Náhled původní stránky z databáze Parzival

Databázi tvoří 47 obrázků původních stran ve formátu JPEG s rozlišením 300dpi, normalizované obrázky řádek a slov s přepisy. Normalizace se týká sklonu a výšky obrázků a je blíže popsána v [13]. Součástí jsou také dvě nadefinované sady trénovacích, validačních a testovacích dat.

Pro naše potřeby obsahuje 23 478 černobílých normalizovaných obrázků slov (příklad viz obrázek 2.2) s přepisy. Udáváno je 93 znaků a 4 934 unikátních slov. Použití je možné pouze pro nekomerční účely a výuku. Vyžadována je registrace a citování [13].



Obrázek 2.2: Normalizovaný obrázek slova *veriach* z databáze Parzival

2.2 Databáze Washington

Databáze byla vytvořena ze sbírky dokumentů prvního amerického prezidenta George Washingtona. Pochází tedy z 18. století, je napsána anglickým jazykem a dvěma spisovateli.

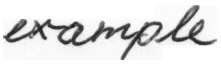
Obrázek 2.3: Normalizovaný obrázek slova *within* z databáze Washington

Databáze je vytvořena z 20 stránek a obsahuje normalizované obrázky řádek a slov s přepisy podobně jako u databáze Parzival. Normalizace je provedena stejně jako v případě databáze Parzival. Součástí je jedna nadefinovaná sada pro trénování, validaci a testování.

Obsahuje 4 894 normalizovaných obrázků slov (příklad viz obrázek 2.3) s přepisy, které tvoří 1 471 unikátních slov. Udáváno je 82 znaků. Použití je možné pouze pro nekomerční účely a výuku. Vyžadována je registrace a citování [13], kde je tato databáze použita a blíže popsána.

2.3 Databáze IAM Handwriting

Databáze pochází ze současnosti. Obsahuje formuláře ručně přepsaného anglického textu, které vyplnilo 657 lidí. Automatickou segmentací těchto formulářů jsou získány obrázky slov, která jsou následně ručně ověřena.



Obrázek 2.4: Obrázek slova *example* z databáze IAM Handwriting

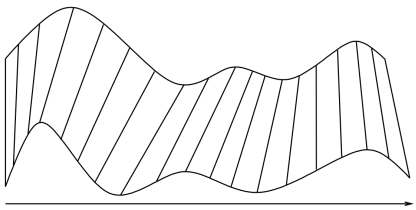
Databáze obsahuje 115 320 obrázků slov (příklad viz obrázek 2.4). Použití je možné pouze pro nekomerční účely. Vyžadována je registrace a citování [15], kde se touto databází blíže zabývají.

3 Možná řešení

Ke starším řešením patří např. Dynamic Time Warping a skrytý Markovův model. V současnosti je tento problém řešen zejména pomocí neuronových sítí.

3.1 Dynamic Time Warping

Dynamic Time Warping (dále jako DTW) je algoritmus používaný obvykle pro rozpoznávání řeči. Jeho aplikace je možná v podstatě na libovolné signály, přičemž výsledek je závislý na tvaru signálu a nikoliv na jeho délce. Vrátíme-li se pro představu zpět k řeči, tak výsledný rozdíl dvou stejných slov řečených jinou rychlostí by měl být nízký a pro rozdílná slova vysoký.

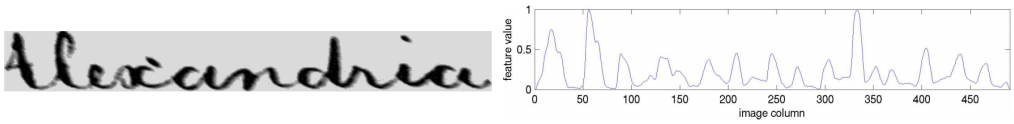


Obrázek 3.1: Princip porovnávání signálů pomocí DTW: horní a dolní signál jsou mezi sebou porovnávány na místech podle vyznačených přímek [16]

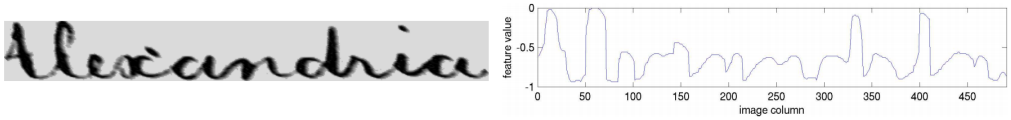
Z toho vychází myšlenka článku [16] porovnávat dva obrázky jako dva signály. Následně na základě jejich rozdílu vyhodnotit, jestli obsahují podobnou informaci. Na obrázku 3.1 je vidět, že dva signály $X = (x_1, ..., x_M)$ a $Y = (y_1, ..., y_N)$ jsou porovnávány na různých místech. K tomu je vytvořena matice D o rozměru M x N, kde $D(i, j)$ představuje rozdíl částí signálů $X_{1:i}$ a $Y_{1:j}$. Matice D je vypočtena pomocí vzorce 3.1, kde $d(x_i, y_j)$ představuje rozdíl vzorků x_i a x_i. Výsledný rozdíl signálů potom představuje $D(M, N)$.

$$D(i,j) = min \left\{ \begin{array}{l} D(i, j-1) \\ D(i-1, j) \\ D(i-1, j-1) \end{array} \right\} + d(x_i, y_j) \tag{3.1}$$

Otázkou zůstává jak z obrázku vytvořit signál. To lze například vytvořením histogramu celého obrázku, ale to pravděpodobně nepřinese žádné výhody. Rozumnou možností je rozdělit obrázek do sloupců. Hodnota ve sloupci



Obrázek 3.2: Vytvoření signálu z obrázku pomocí obsahu inkoustu ve sloupci [16]



Obrázek 3.3: Vytvoření signálu z obrázku nejvyšší hodnotou ve sloupci [16]

bude představovat obsah inkoustu (viz obrázek 3.2). Dále můžeme vzít jako hodnotu sloupce nejvyšší (viz obrázek 3.3) a nejnižší pozici inkoustu. Dále budou uvedeny výhody a nevýhody tohoto algoritmu.

Výhody

- Poměrně jednoduchý algoritmus, který spočívá v převedení obrázků na signály a jejich následném porovnání.
- Nevyžaduje trénování, a proto je možné ho použít na libovolnou sadu dat. Lze očekávat podobnou úspěšnost, ale v žádném případě nemůžeme očekávat stejnou, protože je závislý na vytvořeném signálu, který může být pro různé styly písma zcela odlišný.
- Nevadí mu různá šířka znaků. Z podstaty algoritmu DTW nevadí rozdílná šířka signálu (tedy i jednotlivých písmen), pokud je tvar signálu (písmen) shodný.

Nevýhody

- Záměna znaků je velmi snadná. Snadno si představíme tvar signálu pro písmena Q a O, který bude prakticky totožný.
- Je závislý na stavu dat. Pro špatně zachovalé obrázky bude pravděpodobně docházet k snazším záměnám znaků.
- Má poměrně nízkou úspěšnost (viz [16]), která na nezachovalých datech může být ještě snížena.

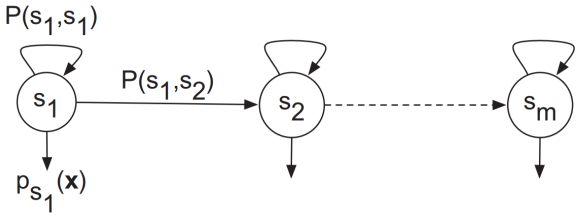
3.2 Skrytý Markovův model

Skrytý Markovův model (dále jako *HMM*) je statistický model, který předpokládá, že modelovaný systém popisuje posloupnost možných stavů, kde pravděpodobnost každého dalšího stavu závisí pouze na stavu předchozím.

Aby bylo možné porovnávat různé styly písma a dosáhnout tak lepší úspěšnosti, jsou obrázky řádků nejprve normalizovány. Řeší se např. převod na černobílý obrázek, sklon písma a odstranění šumu.

HMM je použit v [13], kde jsou kontrolovány rovnou celé řádky. Z normalizovaného obrázku řádky jsou následně pomocí posuvného okna šířky jeden pixel extrahovány lokální příznakové vektory. Okno se pohybuje zleva doprava a v každém místě je extrahováno devět lokálních příznakových vektorů, které jsou tvořeny např. na základě přechodů černá-bílá nebo horního a dolního obrysu (více viz [13]).

HMM jsou použity pro reprezentaci každého znaku z množiny rozpoznávaných znaků a jsou naučeny na obrázcích řádků s přepisy. Tyto modely mají lineární topologii a určitý počet m skrytých stavů S (viz obrázek 3.4). Model může zůstat v současném stavu (bere ohled na rozdílné šířky stejného znaku), nebo přejde do dalšího stavu. $P(S_i, S_j)$ značí pravděpodobnost přechodu ze stavu S_i do stavu S_j. V každém stavu se rozhoduje podle vstupu v podobě dříve extrahovaných lokálních příznakových vektorů. Výstupem stavu je pozorovatelný příznakový vektor x s pravděpodobností $P_{S_i}(x)$, podle kterého lze vyhodnotit jestli byl znak rozpoznán.



Obrázek 3.4: Skrytý Markovovův model pro reprezentaci znaku použitý v [13]

Model klíčového slova je vytvořen jako sekvence modelů znaků. Na základě výstupu modelu klíčového slova je vypočteno skóre pro řádek, které představuje pravděpodobnost, že řádek obsahuje klíčové slovo. Pokud pravděpodobnost řádku přesahuje určitou hranici, je řádek vyhodnocen jako odpovídající a vrácen spolu se souřadnicemi nalezeného slova. Následují výhody a nevýhody tohoto postupu.

Výhody

- V porovnání s DTW má lepší úspěšnost.
- Bere v úvahu rozdílné šířky znaků.

Nevýhody

- Vyžaduje trénování.
- V porovnání s DTW je mnohem složitější.

3.3 Neuronové sítě

Použití neuronových sítí je v současné době populární. Jsou používány stále častěji a to i v oblasti rozpoznávání vzorů.

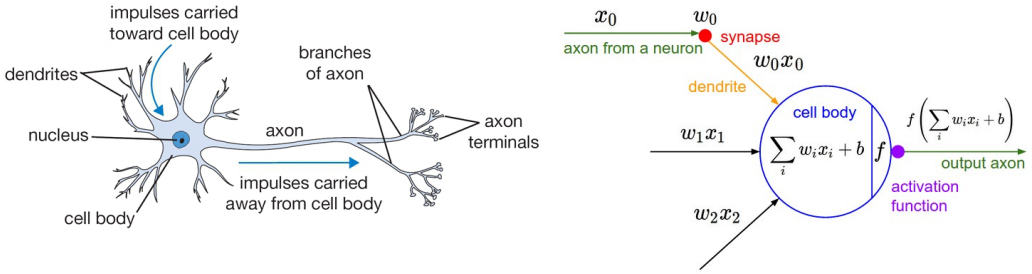
Výhody

- Dosahuje nejlepší úspěšnosti z prostudovaných článků.
- Je možné vytvořit více architektur sítě.

Nevýhody

- Vyžaduje trénování.
- Trénování je časově náročné.
- Správné nastavení parametrů sítě není známo.

Použití neuronových sítí pro vyhledávání vzorů může být různorodé. Prakticky neexistuje jediný správný postup, ale obvykle je jich hned několik. Zajímavou možností pro vyhledávání podle vzoru je SpottingNet [19], kde jsou na vstup sítě přivedeny dva obrázky a výstupem je, zda si obrázky odpovídají. PHOCNet [18] je zástupcem další možnosti, která spočívá v převedení obrázku na jinou reprezentaci, kterou je možné jednodušeji porovnávat. Tento postup umožňuje vyhledávání podle vzoru i řetězce a vychází z něho zvolená řešení, která jsou popsána v kapitole 4. Následuje seznámení se základem neuronových sítí.



Obrázek 3.5: Neuron (vlevo) a jeho matematický model (vpravo) [3]

3.3.1 Základ neuronových sítí

Cílem této části je seznámení s principem neuronových sítí a základními topologiemi. Podsekce ve značné míře čerpá z CS231n: Convolutional Neural Networks for Visual Recognition [3], kam zároveň odkazujeme pro hlubší pochopení dané problematiky.

Biologická analogie

Oblast neuronových sítí byla inspirována cílem modelování biologických nervových systémů.

Neuron Neuron je základní výpočetní jednotkou mozku. Lidský nervový systém obsahuje přibližně 86 miliard neuronů, které propojuje přibližně $10^{14} - 10^{15}$ synapsí. V obrázku 3.5 můžeme vidět nákres neuronu a jeho matematický model.

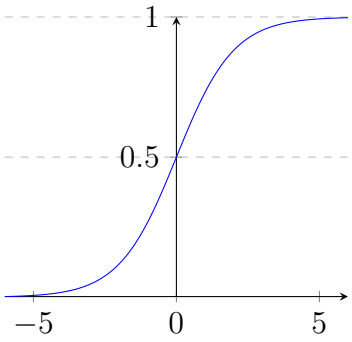
Neuron ze vstupních signálů na svých dendritech produkuje výstupní signál, který je veden na jediný axon. Axon (výstup) může být propojen synapsemi na dendrity (vstup) dalších neuronů.

Model neuronu V matematickém modelu (viz obrázek 3.5) jsou vstupy neuronu značeny jako $x_0, x_1, ..., x_n$. Sílu jednotlivých spojení představují váhy $w_0, w_1, ..., w_n$. Váhy ovlivňují vliv jednoho neuronu na druhý, toho je docíleno násobením vstupu jeho váhou $(w_i \cdot x_i)$.

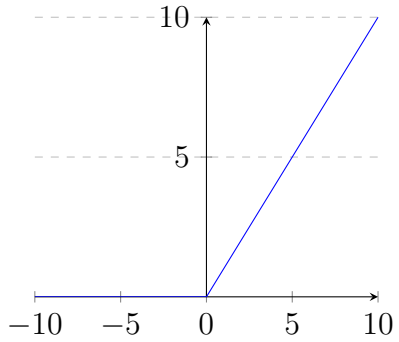
Celkový vstup poté dostaneme sečtením $\sum_{i=0}^{n} w_i \cdot x_i + b$, kde b představuje bias, což je parametr neuronu, který lze použít pro posunutí aktivační funkce doleva nebo doprava.

Vhodnou úpravou vah a parametru bias během trénování se snažíme dosáhnout požadovaného chování sítě.

V reálném neuronu musí celkový vstup z dendritů přesáhnout určitou mez, aby mohl neuron vystřelit signál přes axon. Zde si povšimněme neli-



Obrázek 3.6: Sigmoida jako aktivační funkce: $f(x) = \frac{1}{1+e^{-x}}$



Obrázek 3.7: ReLU aktivační funkce: $f(x) = \max(0, x)$

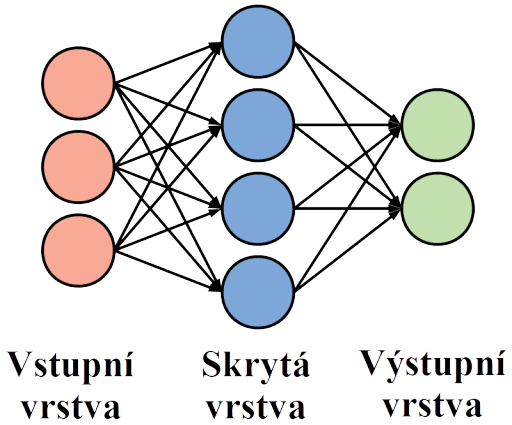
nearity mezi vstupem a výstupem neuronu. V matematickém modelu toto chování modelujeme pomocí *aktivační funkce*.

Pro aktivační funkci je důležité, aby nebyla lineární. Příkladem aktivační funkce je *sigmoida* (viz obrázek 3.6) nebo *ReLU* (viz obrázek 3.7). Výstup neuronu vypočteme jako $f(\sum_{i=0}^{n} w_i \cdot x_i + b)$, kde f je aktivační funkce.

Architektura neuronové sítě

Neuronovou síť tvoří propojení jednotlivých modelů neuronu, ty jsou obvykle rozmístěny do vrstev (viz obrázek 3.8).

Vstupní vrstva Je první vrstva sítě, která představuje vstupní data. Její velikost je závislá na velikosti vstupních dat. Na obrázku 3.8 se jedná o vektor se 3 hodnotami.



Obrázek 3.8: Model neuronové sítě (Vrcholy grafu představují jednotlivé neurony. Hrany představují jednotlivé synapse.)

Skrytá vrstva Nachází se mezi vstupní a výstupní vrstvou. Množství skrytých vrstev a počet neuronů v nich zásadně ovlivňuje kapacitu neuronové sítě a schopnost se učit složitá data.

Výstupní vrstva Je poslední vrstva, z které získáme výstup celé sítě. V tomto případě se jedná o vektor s dvěma hodnotami, který může představovat dvě pravděpodobnosti (první značí pravděpodobnost první sledované vlastnosti, druhá značí pravděpodobnost druhé sledované vlastnosti).

Na základě počtu neuronů ve výstupní vrstvě a zvolené aktivační funkci může být výsledek různě interpretován. Např. pro klasifikaci ANO - NE bychom mohli zvolit jeden výstupní neuron a sigmoidu jako aktivační funkci. Potom by hodnota $\geq 0,5$ mohla být interpretována jako ANO a hodnota $< 0,5$ jako NE.

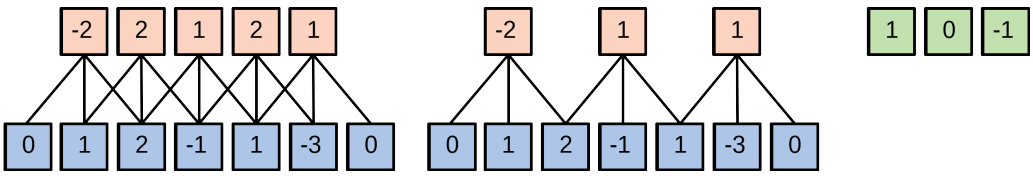
Vrstvy neuronové sítě

Předpokládejme, že síť je složena z vrstev jako na obrázku 3.8 a vzniklý graf z neuronů a synapsí je acyklický.

Neuronová síť potom může být složena z různého počtu a různých typů vrstev.

Fully-connected vrstva Je nejběžnějším typem vrstvy (použita v obrázku 3.8 jako skrytá a výstupní vrstva), kdy je každý neuron propojen s každým neuronem v předchozí vrstvě. Lze si všimnout značného množství synapsí a tedy vah, které je potřeba se učit a uchovávat v paměti počítače.

Konvoluční vrstva Tvoří základ konvolučních neuronových sítí, které se často používají pro rozpoznávání a klasifikaci obrázků. Oproti předchozí fully-connected vrstvě je neuron propojen pouze s malou oblastí (*velikost filtru*) a jeho naučené parametry (tvoří *filtr*) jsou sdíleny mezi neurony napříč celou vrstvou (viz obrázek 3.9).

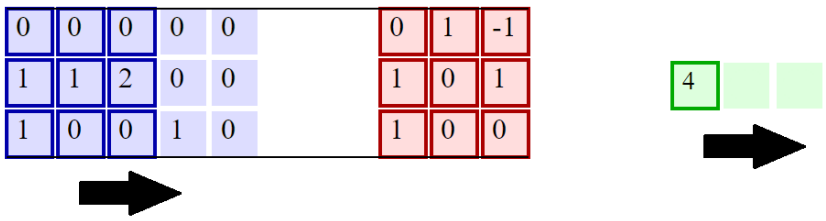


Obrázek 3.9: Možné propojení neuronů v konvoluční vrstvě (vlevo a uprostřed) a sdílené váhy neuronů tvořící filtr (vpravo) [3]

Vzdálenost o jakou je filtr posouván lze nastavit parametrem *stride*, který je obvykle 1 x 1, což znamená posun o jeden pixel ve vertikálním i horizontálním směru.

Dále se používá parametr *padding*, který doplní okraje nulami. Lze tak kontrolovat rozměry výstupu.

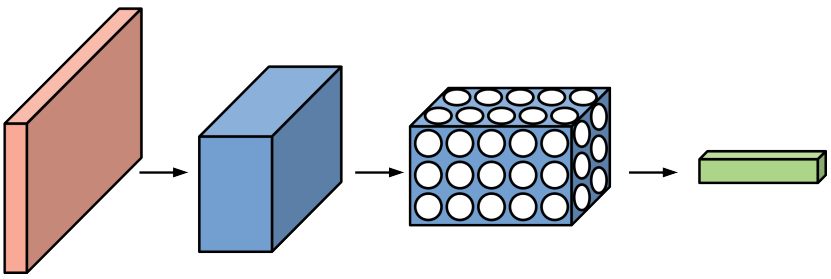
Je-li vstupem černobílý obrázek o šířce 5 pixelů a výšce 3 pixely a zvolíme-li jeden filtr velikosti 3 x 3, bude činnost konvoluční vrstvy odpovídat obrázku 3.10.



Obrázek 3.10: Zjednodušený princip konvoluční vrstvy: Na vstup (vlevo) aplikujeme filtr (uprostřed) a získáme výstup (vpravo). Filtr posunujeme po vstupu dokud není výstup kompletní. [3]

Obecně má konvoluční vrstva své neurony rozmístěné ve třech dimenzích a podle obrázku 3.11 převádí 3D vstupní objekt na jiný 3D objekt, který představuje aktivace neuronů. K tomu používá množinu filtrů. Filtr je také 3D objektem a představuje váhy pro oblast, ze které se pomocí konvoluce získá výstup neuronu. Hloubka filtru odpovídá hloubce vstupu. Konvoluce se tedy provádí v celé hloubce vstupu. Hloubka výstupu jednoho filtru je potom rovna jedné.

Použijeme-li v konvoluční vrstvě více filtrů, získáme z každého filtru výstup s hloubkou jedna o stejných rozměrech. Pokud tyto jednotlivé výstupy poskládáme na sebe, získáme výstup konvoluční vrstvy, kde je hloubka rovna počtu filtrů.



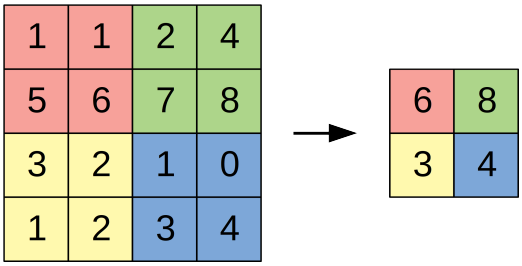
Obrázek 3.11: Transformace objektu v konvoluční síti (vstupní objekt vlevo) [3]

Značnou výhodou této vrstvy je *sdílení parametrů* mezi neurony. První konvoluční vrstva obvykle detekuje hrany v obrázku, proto není nutné, aby se každý neuron znovu učil jak hranu detekovat. Další vrstvy mohou detekovat komplexnější objekty jako např. kruh, obličej nebo stůl, kde je sdílení opět vhodné. Díky sdílení parametrů dosáhneme značné úspory paměti.

Max pooling vrstva Jejím účelem je snížení počtu parametrů a množství následných výpočtů. Toho je dosaženo tak, že se vezme maximální hodnota z oblasti určené parametrem *poolsize* (viz obrázek 3.12).

Tato operace je prováděna nezávisle pro každý řez hloubky vstupního objektu. Vstup a výstup mají tedy shodnou hloubku.

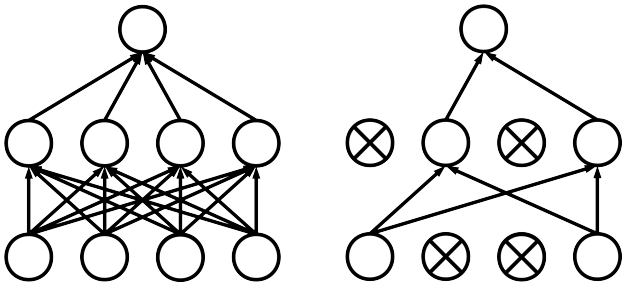
Posun oblasti je určen parametrem *stride* podobně jako u konvoluční vrstvy.



Obrázek 3.12: Princip max pooling vrstvy: vstup hloubky 1 (vlevo) a výstup s 2 x 2 poolsize a 2 x 2 stride (vpravo) [3]

Dropout vrstva Použití dropout vrstvy je technika, která dokáže zabránit přílišnému upnutí sítě na vzory během trénování modelu.

Při trénování je nastaven parametr p, který značí pravděpodobnost, že neuron v dané vrstvě zůstane aktivní. Žádný neuron se tak nemůže spoléhat na



Obrázek 3.13: Princip dropout vrstvy: model sítě před (vlevo) a po (vpravo) aplikování dropout (kruhy představují aktivní neurony, kruhy s křížem neaktivní neurony a šipky aktivní spojení mezi neurony)

konkrétní hodnotu z konkrétního neuronu. Princip je naznačen na obrázku 3.13. Po úpravě vah během tréninku je dropout aplikován v jiné podobě.

Po natrénování sítě se už dropout nepoužívá a všechny neurony zůstávají aktivní.

Trénování neuronové sítě

Správné nastavení vah neuronové sítě se provádí trénováním, které je nejčastěji řešeno algoritmem *backpropagation*.

Pro trénování potřebujeme trénovací sadu, kterou tvoří dostatečně velké množství dvojic vstupu a požadovaného výstupu. Při začátku trénování nejdříve váhy nastavíme náhodně.

Následně se podle vstupu z trénovací sady vyhodnotí výstup sítě, který porovnáme s požadovaným výstupem z trénovací sady. Na základě tohoto rozdílu lze zpětným chodem dopočítat, jaký vliv mají jednotlivé neurony na výstup sítě (viz [3] nebo [8]) a tedy i jak změnit jejich váhy, aby se odchylka od správného výstupu co nejvíce snížila. Cílem trénování je tedy dosáhnout minimální možné chyby neuronové sítě přes celou sadu anotovaných dat.

4 Zvolená řešení

Pro řešení tohoto problému jsme zvolili neuronové sítě, které na základě prostudované literatury dosahují nejlepších výsledků. Pomocí neuronových sítí budou generovány příznakové vektory pro reprezentaci hledaných slov. Na základě porovnávání těchto příznakových vektorů potom budeme hledat vhodné odpovědi pro jednotlivé dotazy.

Aby bylo možné provádět QbS i QbE dotazy, zvolili jsme PHOC reprezentaci výstupního vektoru, se kterou je možné dle prostudované literatury dosáhnout výborných výsledků. PHOC reprezentace bude popsána v následující sekci.

Poté následují tři zvolená řešení, kterými jsou konvoluční síť (dále jako *cnn*), konvoluční síť se Spatial Pyramid Pooling vrstvou (dále jako *spp*) a síť s konvoluční LSTM (dále jako *lstm*).

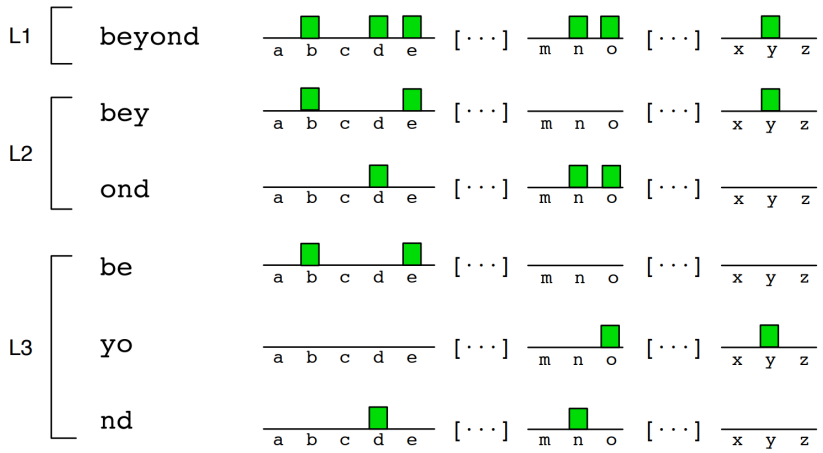
4.1 PHOC reprezentace výstupního vektoru

PHOC (z anglického *Pyramidal Histogram of Characters*) reprezentaci je vhodné použít při modelování slov různých délek vektorem o stejné dimenzi. PHOC vektor zjednodušeně reprezentuje slovo tak, že popisuje výskyt znaků v jeho částech. Poskytuje informaci o tom, zda se konkrétní znak nachází v první polovině, v druhé třetině slova apod. Výhodou tohoto postupu je konstantní velikost vektoru.

Mějme množinu znaků o velikosti n. Pro slovo vytvoříme binární histogram, kde 1 značí přítomnost znaku a 0 značí nepřítomnost znaku. Každý histogram má tedy velikost n.

Pouze znalost znaků, které se vyskytují ovšem nestačí pro identifikaci slova. Např. slova *spát* a *psát* obsahují stejné znaky, a tak by jejich binární histogramy byly totožné. Proto slovo rozdělíme na poloviny a v další úrovni přidáme dva histogramy, kde první značí výskyt znaků v první polovině slova a druhý značí výskyt znaků v druhé polovině slova. Slova *spát* a *psát* takto ovšem stále nerozlišíme, proto přidáváme další úrovně (viz obrázek 4.1) a slovo dělíme na třetiny, čtvrtiny apod.

Dále si všimněme, že velikost vektoru roste s každou další úrovní. První úroveň bere v úvahu celé slovo a má tedy velikost n. Pátá úroveň dělí slovo na pětiny a velikost je tedy $5n$. Výsledná dimenze výstupního vektoru je tedy součet velikostí všech úrovní. Jednotlivé úrovně jsou ve vektoru poskládány za sebou.



Obrázek 4.1: Příklad PHOC reprezentace slova *beyond* pro první (L1), druhou (L2) a třetí (L3) úroveň [11]

V [11] doporučují používat druhou až pátou úroveň. Výsledná dimenze by byla $(2 + 3 + 4 + 5) \cdot n$.

4.2 Konvoluční síť

Konvoluční síť je tvořena konvolučními a max pooling vrstvami. Důvodem použití je velké množství vstupní informace v případě obrázků, kde vzhledem k velkému počtu parametrů není vhodné použití plně propojené sítě.

Např. pro obrázek ve stupních šedi 120 x 120 pixelů by měl neuron 14 400 vah. Zde se velmi snadno narazí na paměťové limity. Ve [3] navíc zmiňují, že je toto plné propojení neuronů zbytečné a velké množství parametrů vede k rychlému upnutí na trénovací vzory a přetrénování sítě.

Proto je vhodné použití konvolučních vrstev, které byly popsány výše. Ve spojení s max pooling vrstvou je možné redukovat výstup konvoluční vrstvy, který díky max pooling vrstvě obsahuje pouze důležité informace. Následně je možné připojit fully-connected vrstvu.

Zamyslíme-li se nad výstupem konvoluční vrstvy, na který aplikujeme max pooling, může se síť díky výběru nejdůležitější informace z určené oblasti vypořádat i s lehkým sklonem a deformací písma. Dejme tomu, že konvoluční vrstva detekuje oblouk a výstup je zredukován max pooling vrstvou. Potom je jedno, kde přesně byl oblouk v oblasti *poolsize* detekován. Potom pro dva vstupy rozdílné pouze v malém posunu tohoto oblouku budou vypočteny stejné výstupy.

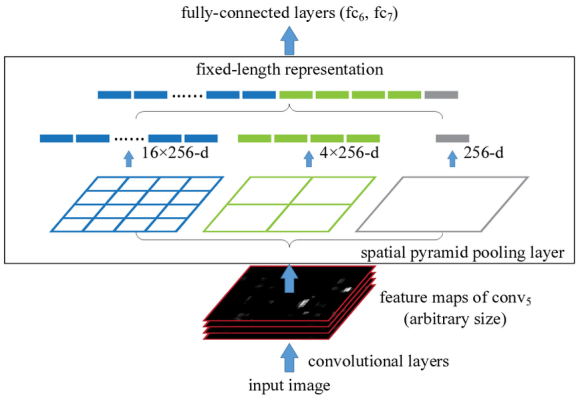
Nevýhodou je stále nutnost vstupu o konstantních rozměrech, proto je nutné obrázek před přivedením na vstup sítě upravit.

4.3 Konvoluční síť se Spatial Pyramid Pooling vrstvou

Problémem konvoluční sítě je potřeba stejně velkého vstupu. S tímto problémem se hodláme vypořádat podobně jako u sítě PHOCNet (viz [18]) použitím Spatial Pyramid Pooling vrstvy, která nám umožní na vstup sítě přivést obrázky s libovolným rozměrem bez potřeby úprav.

4.3.1 Spatial Pyramid Pooling vrstva

Spatial Pyramid Pooling vrstva (viz [14]) řeší problém spojení mezi konvoluční a fully-conected vrstvou tak, že výstup konvoluční vrstvy převede na vektor konstantní délky podle obrázku 4.2.



Obrázek 4.2: Princip Spatial Pyramid Pooling vrstvy [14]

Na výstup konvoluční vrstvy nejprve aplikuje mřížku 1 x 1 (to odpovídá celému výstupu) a v této oblasti provede max pooling přes celou oblast. Max pooling je proveden pro všech n filtrů a získá se prvních n hodnot vektoru (počet filtrů na obrázku 4.2 je 256). Následně je aplikována další mřížka 2 x 2, kde je výstup rozdělen na 4 oblasti a v nich opět proveden max pooling, tím se získá dalších $4n$ hodnot. Podobně se pokračuje s mřížkou 4 x 4.

Výstupní vektor Spatial Pyramid Pooling vrstvy je tedy konstantní délky nezávisle na velikosti vstupu a fully-conected vrstva může být připojena.

4.4 Síť s konvoluční LSTM

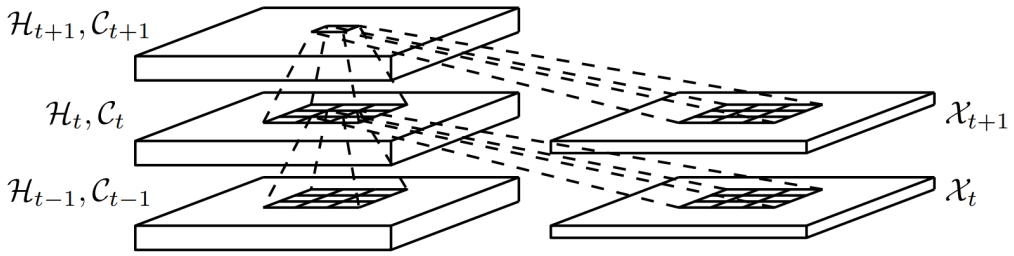
Síť s konvoluční LSTM je dalším způsobem, jak vyřešit rozdílné rozměry vstupních obrázků. V tomto případě k tomu hodláme použít konvoluční LSTM (z anglického Long Short Term Memory).

Protože vstup konvoluční LSTM tvoří sekvence snímků, je opět nutné vstupní obrázek vhodně zpracovat. Na výstup konvoluční LSTM můžeme napojit fully-connected vrstvu a výstupní vektor může odpovídat *cnn* a *spp*.

4.4.1 Konvoluční LSTM

Konvoluční LSTM je podrobně rozebrána v [17], kde je používána pro předpověď počasí ze snímků oblohy.

Konvoluční LSTM si uchovává stav, který na základě vstupu upravuje. Úpravy jsou konvoluční a dochází tedy ke sdílení vah. Princip je naznačen na obrázku 4.3, kde H_t představují skryté stavy, C_t představují výstupy buněk a X_t představují vstupy.



Obrázek 4.3: Princip konvoluční LSTM [17]

Na základě předchozího stavu H_{t-1}, C_{t-1} a vstupu X_t je vypočten další stav H_t, C_t (viz [17]). Výstupem mohou být jednotlivé stavy nebo poslední stav.

5 Prostředí pro vývoj

Pro zpracování obrázků je použita knihovna OpenCV. Pro vývoj a testování neuronových sítí jsme se rozhodli použít knihovnu Keras, která používá doporučenou knihovnu TensorFlow. Důvodem je rychlá a efektivní implementace architektur neuronových sítí při experimentování s možností výpočtu na grafické kartě. Z důvodu použitých knihoven je programovacím jazykem Python.

5.1 Knihovna OpenCV

OpenCV [1] je open source knihovna počítačového vidění, která je volně dostupná pro akademické i komerční použití. Knihovna je navržena s ohledem na výpočetní efektivitu a soustředí se také na aplikace reálného času.

Podporované operační systémy jsou Windows, Linux, Android a Mac OS. Knihovna je napsána v C++ a má rozhraní pro C, C++, Python, Java a MATLAB.

OpenCV obsahuje více než 2 500 optimalizovaných algoritmů z oblasti počítačového vidění. Tyto algoritmy mohou být použity například pro detekování a rozpoznávání tváří, identifikaci objektů, klasifikaci lidských pohybů ve videích nebo sledování pohybujících se objektů. Často se používá pro zpracování obrazu v reálném čase.

Podle oficiálních stránek OpenCV [1] používá více než 47 tisíc uživatelů z různých oblastí. Jejich aktivita na fórech spolu s dobrou dokumentací nabízí řešení pro obrovské množství problémů v oblasti počítačového vidění. Spolu se snadnou použitelností je to důvod, proč jsme se rozhodli použít tuto knihovnu pro manipulaci s obrázky.

V této práci bude knihovna použita převážně pro načítání obrázků a jejich úpravě pro vstup neuronové sítě. To obnáší např. změnu velikosti obrázků. Další využití bude pro vyhlazení obrázků nebo rozšíření množiny dat, kde lze použít afinní transformace. V následující části se seznámíme s používanými funkcemi knihovny.

5.1.1 Změna velikosti obrázku

Pro změnu velikosti obrázku lze použít funkci *resize*:

```
res = cv.resize(img, (width, height), interpolation)
```

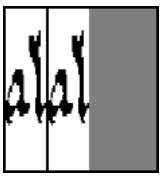
Zde *res* je obrázek se změněnou velikostí, *img* je původní obrázek, *width* je zvolená šířka, *height* je zvolená výška a *interpolation* je zvolená interpolace. Jako interpolaci používáme výchozí bilineární interpolaci. Výsledek může odpovídat obrázku 5.1.

Obrázek 5.1: Příklad změny velikosti obrázku funkcí resize (vlevo původní, vpravo po roztažení)

Další možností je zvětšení obrázku pomocí funkce *copyMakeBorder*:

```
res = cv.copyMakeBorder(img, top, bottom, left, right,
                        borderType, value)
```

Zde *top, bottom, left, right* jsou velikosti okrajů v daných směrech, *borderType* definuje způsob tvorby okrajů a *value* je barva okraje, pokud je nastaven borderType na konstantní okraj. Výsledek může odpovídat obrázku 5.2.

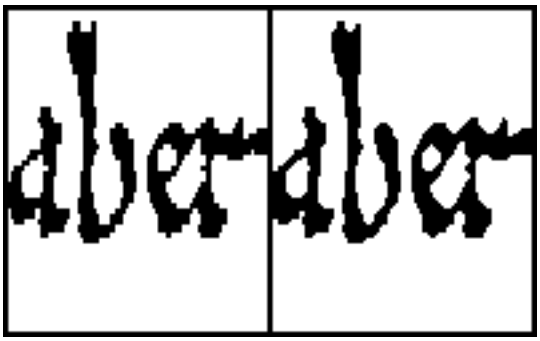


Obrázek 5.2: Příklad změny velikosti obrázku funkcí copyMakeBorder (vlevo původní, vpravo po změně)

Protože OpenCV používá jako vnitřní reprezentaci obrázku pole z balíku NumPy [7], je možné pro potřebu oříznutí obrázku jednoduše použít prostředky tohoto balíku.

5.1.2 Vyhlazení obrázků

Vyhlazení hran písmen lze dosáhnout nejprve rozmazáním funkcí *GaussianBlur*, pomocí které získáme obrázek ve stupních šedi. Obrázek následně převedeme zpět na černobílý pomocí funkce *threshold*. Výsledek potom odpovídá obrázku 5.3.



Obrázek 5.3: Příklad vyhlazení obrázku (vlevo původní, vpravo vyhlazený)

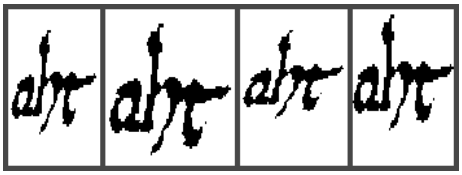
5.1.3 Afinní transformace

Afinní transformace podle dokumentace OpenCV představuje jakoukoli transformaci, kterou lze vyjádřit pomocí násobení matic a přičtení vektoru. Pomocí afinních transformací můžeme u obrázku vyjádřit rotaci, posun a změnu velikosti.

Použití afinních transformací v knihovně OpenCV vypadá následovně:

```
pts1 = np.float32([[50, 50], [200, 50], [50, 200]])
pts2 = np.float32([[10, 100], [200, 50], [100, 250]])
M = cv.getAffineTransform(pts1, pts2)
res = cv.warpAffine(img, M, (width, height))
```

Zde *pts1* představuje souřadnice bodů v původním obrázku a *pts2* umístění těchto bodů ve výstupním obrázku. Z těchto bodů je vytvořena matice *M*, která je spolu s původním obrázkem předána funkci *warpAffine*. Afinní transformace jsou použity při rozšíření datové množiny. Výsledek může vypadat např. podle obrázku 5.4.



Obrázek 5.4: Rozšíření sady pomocí afinních transformací (původní obrázek je vlevo, následují obrázky vytvořené z původního)

5.2 TensorFlow

TensorFlow [9] je open source knihovna pro numerické výpočty, která poskytuje rozhraní pro jazyky Java, C++, Python a další. Umožňuje výpočty

provádět na více procesorech nebo grafických kartách, což je při náročných výpočtech velice výhodné.

Knihovna TensorFlow byla vyvinuta výzkumnými pracovníky a inženýry z týmu Google Brain Team pro účely výzkumu v oblasti strojového učení a hlubokých neuronových sítí. Díky její obecnosti je však možné použití i pro jiné problémy.

Knihovnu TensorFlow používají společnosti jako Google, ebay nebo airbnb. Lze ji tedy považovat za funkční a ověřenou.

5.3 Keras

Keras [4] je vysokoúrovňové rozhraní pro neuronové sítě napsané v jazyce Python. Může pracovat nad knihovnami TensorFlow, CNTK [2] nebo Theano [10].

Umožňuje běh na procesoru i grafické kartě. Použití na grafické kartě je výhodné pro zvýšení rychlosti, kde nám zejména při trénování neuronových sítí ušetří stovky hodin. Další výhodou může být dávkové zpracování při trénování sítě. V normálním zpracování je úprava vah sítě prováděna po jednotlivých obrázcích. Oproti tomu u dávkového zpracování je úprava vah spočtena pro *dávku* obrázků. Zásahy do vah sítě potom mohou být přesnější. Keras podporuje velké množství neuronových sítí včetně konvoluční, která je pro nás zásadní.

Keras se snaží umožnit rychlé experimentování. K tomu mu pomáhá modularita, kde je model chápán jako spojení modulů, které lze konfigurovat a jednoduše spojovat. Jako moduly jsou brány např. vrstvy neuronové sítě, aktivační funkce a optimalizátory trénování. Navíc je možné si vytvořit vlastní moduly, podle již implementovaných.

V dnešní době existuje řada frameworků pro neuronové sítě, mezi kterými jsme si Keras vybrali pro svou efektivitu a schopnost rychle přejít od nápadu k výsledkům. Tuto schopnost je možné prezentovat následujícím úsekem kódu, který představuje celý proces vytvoření modelu a jeho trénování.

```
# Vytvoření sekvenčního modelu sítě
model = Sequential()
# Přidání konvoluční vrstvy s 32 filtry velikosti 3 x 3
model.add(Conv2D(32, (3, 3), activation='relu',
          input_shape=input_shape))
# Přidání max pooling vrstvy s 2 x 2 poolsize
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
```

```
# Přidání fully-connected vrstvy s 2048 neurony
model.add(Dense(2048, activation='relu'))
# Přidání dropout vrstvy
model.add(Dropout(0.5))
model.add(Dense(1000, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam')

# Trénování sítě na 10 epoch s dávkou 16
# trainX jsou trénovací vstupní data
# trainY jsou požadované výstupy
model.fit(trainX, trainY, batch_size=16, epochs=10)
```

Dávkou je myšlen počet vstupů, po kterém je vyhodnocena úprava vah sítě.

Epocha znamená, že jsou jedenkrát zpracována všechna trénovací data. Trénování na deset epoch potom znamená, že každý obrázek je zpracován sítí během trénování desetkrát. Vzhledem k dávkovému zpracování se může stát, že je obrázků nedostatek pro naplnění poslední dávky a budou vynechány. Tento problém částečně řeší promíchání dat před každou epochou.

5.4 Systém

Trénování neuronových sítí probíhalo v první fázi na MetaCentru [6]. V pozdější fázi se trénování přesunulo na stolní počítač s SSD diskem, procesorem Intel Core i5-4670K, 16 GB RAM a grafickou kartou GTX 770 s 2 GB GDDR5 paměti.

Důvodem přesunutí výpočtu na stolní počítač byla velikost trénovacích dat. Pro MetaCentrum je kvůli výkonu vhodné tato data předzpracovat a nahrát v podobě jednoho souboru. Problémem byla velikost tohoto souboru. Oproti tomu je na stolním počítači možné trénovací obrázky načítat a zpracovávat průběžně během trénování a ušetřit tak čas během experimentů.

6 Experimenty

Jako první experimentální síť jsme zvolili jednoduchou konvoluční síť s cílem ověření funkčnosti řešení, porovnání metod pro extrakci vzorů a hledání vhodného nastavení parametrů sítě.

Dále si popíšeme použité vzdálenosti pro porovnávání vektorů a metriky pro vyhodnocení úspěšnosti sítě, které jsou použité v této práci. Dále bude popsán způsob přípravy dat a jejich rozšíření.

6.1 Porovnávání vektorů

Jelikož výstupem sítě je vektor, je potřeba vektory porovnávat. Předpokládáme, že podobné vektory nesou podobnou informaci. Potom chceme, aby podobné vektory měly malou vzdálenost a rozdílné vektory naopak velkou.

Na základě vzdálenosti vektorů od vzorového vektoru můžeme určit, která data nejvíce odpovídají vzoru.

Mějme vektory u a v se složkami u_i a v_i.

Eukleidovská vzdálenost Eukleidovská vzdálenost měří přímou vzdálenost mezi dvěma vektory. Spočteme ji podle vzorce 6.1.

$$D_{euk}(u, v) = \sqrt{\sum_{i=1}^{n}(u_i - v_i)^2} \tag{6.1}$$

Kosinová vzdálenost Kosinová vzdálenost měří vzdálenost na základě velikosti úhlu mezi vektory. Spočteme ji podle vzorce 6.2.

$$D_{kos}(u, v) = 1 - \frac{\sum_{i=1}^{n}(u_i) \cdot (u_i)}{\sqrt{\sum_{i=1}^{n}(u_i)^2} \cdot \sqrt{\sum_{i=1}^{n}(v_i)^2}} \tag{6.2}$$

Bray-Curtis podobnost Bray-Curtis podobnost nabývá hodnot od nuly do jedné. Udává kolik společné informace sdílejí dva vektory a spočteme ji podle vzorce 6.3.

$$S_{b-c}(u, v) = \frac{\sum_{i=1}^{n}|u_i - v_i|}{\sum_{i=1}^{n}|u_i + v_i|} \tag{6.3}$$

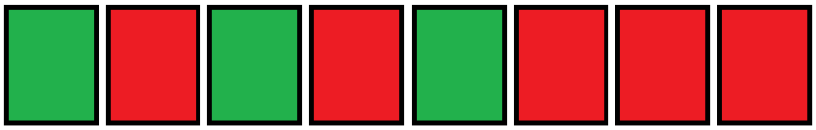
6.2 Evaluační metriky

Tato sekce se týká používaných metrik, které specifikují postup vyhodnocení úspěšnosti vyhledávání obrázků.

6.2.1 Mean Average Precision

Mean Average Precision (dále jako *MAP*) je metrika popsaná v [5], která je vhodná pro oblast získávání informací (angl. information retrieval).

Představme si dotaz na zelený obrázek, kde odpověď vypadá podle obrázku 6.1. MAP potom udává jak relevantní odpovědi budou uživateli poskytnuty, pokud budou procházeny od nejpravděpodobnějších.



Obrázek 6.1: Příklad odpovědi pro dotaz na zelený obrázek: Odpověď tvoří množina zelených a červených obrázků, které jsou v odpovědi seřazeny podle pravděpodobnosti, že se jedná o zelený obrázek. Vlevo je první odpovězený obrázek, který je zelený s největší vypočtenou pravděpodobností.

Výpočet

Pro výpočet MAP nejprve musíme spočítat Average Precision jednotlivých dotazů. Na příkladu (viz obrázek 6.1) známe správné odpovědi. Pro každou správnou odpověď spočteme zleva do její pozice počet odpovědí n a počet správných odpovědí s a spočteme jejich podíl $\frac{s}{n}$. Z těchto hodnot následně spočteme aritmetický průměr. Výpočet Average Precision pro tento příklad tedy je: $(\frac{1}{1} + \frac{2}{3} + \frac{3}{5}) \cdot \frac{1}{3} \doteq 0,76$

Pro tento příklad nás zajímá pouze prvních pět odpovědí, protože poslední správná odpověď je na páté pozici a mezi prvními pěti jsou tedy všechny správné. Další irelevantní dokumenty už nás nezajímají a proto s nimi dále nepočítáme. MAP je potom aritmetickým průměrem Average Precision všech dotazů.

Vyhodnocení pro dotaz podle řetězce (QbS)

Provedeme dotazy na všechna slova validační nebo testovací sady a z výsledků spočteme MAP.

Vyhodnocení pro dotaz podle obrázku (QbE)

Pro každé slovo validační nebo testovací sady je vybrán jeden obrázek jako dotaz. Tento obrázek je pro výpočet dotazu vyřazen. Pokud je pro konkrétní slovo v dané části sady pouze jeden obrázek, je dotaz přeskočen. Z výsledků dotazů spočteme MAP.

6.2.2 Přesnost (Accuracy)

Metrika přesnot (dále jako *acc*) oproti MAP není příliš vhodná pro oblast získávání informací, je ale běžně používána v řadě klasifikačních úloh.

Výpočet

Ve validační nebo testovací části sady je n slov, které dělí obrázky do n tříd. K těmto třídám (slovům) vytvoříme správné vektory. Poté pro každý obrázek pomocí sítě odhadneme vektor, který porovnáváme se správnými vektory tříd. Správný vektor s nejmenší vzdáleností od odhadnutého vektoru říká do jaké třídy obrázek zařadíme a tedy jaké slovo představuje.

Výslednou úspěšnost *acc* vypočteme podle vzorce 6.4 tak, že počet správně přiřazených obrázků *ok* vydělíme celkovým počtem obrázků *total*.

$$acc = \frac{ok}{total} \tag{6.4}$$

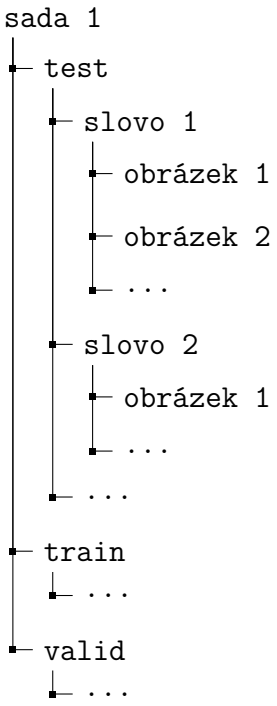
6.3 Příprava dat

Pro tento problém byla zvolena databáze Parzival, protože obsahuje velké množství dat a z prostudovaných databází nejlépe odpovídá historickým dokumentům, na které by mohlo být toto vyhledávání slov použito.

6.3.1 Volba a vlastnosti sady

Databáze Parzival obsahuje dvě oficiální sady. Zvolena byla první, protože je větší a je navíc použita i v [13].

Trénovací část sady obsahuje 3 165 slov a je určena pro trénování. Validační část má 1 724 slov a slouží pro ověřování funkčnosti a správné nastavení parametrů neuronové sítě. Testovací část je složena z 2 271 slov a je určena pro závěrečné vyhodnocení výsledků. Všechny obrázky jsou černobílé a mají výšku 120 pixelů. Nevýhodou je nerovnoměrné rozložení obrázků pro jednotlivá slova. Na jedné straně nalezneme jediného zástupce pro slovo a na druhé straně jich je 282.



Obrázek 6.2: Struktura sady po zpracování

Sada byla pro ulehčení práce zpracována do struktury (viz obrázek 6.2) tak, že je sada rozdělena do testovací, trénovací a validační části podle složek. V těchto složkách jsou složky pojmenovány podle přepisu slova a obsahují odpovídající obrázky.

6.3.2 Rozšíření datové sady

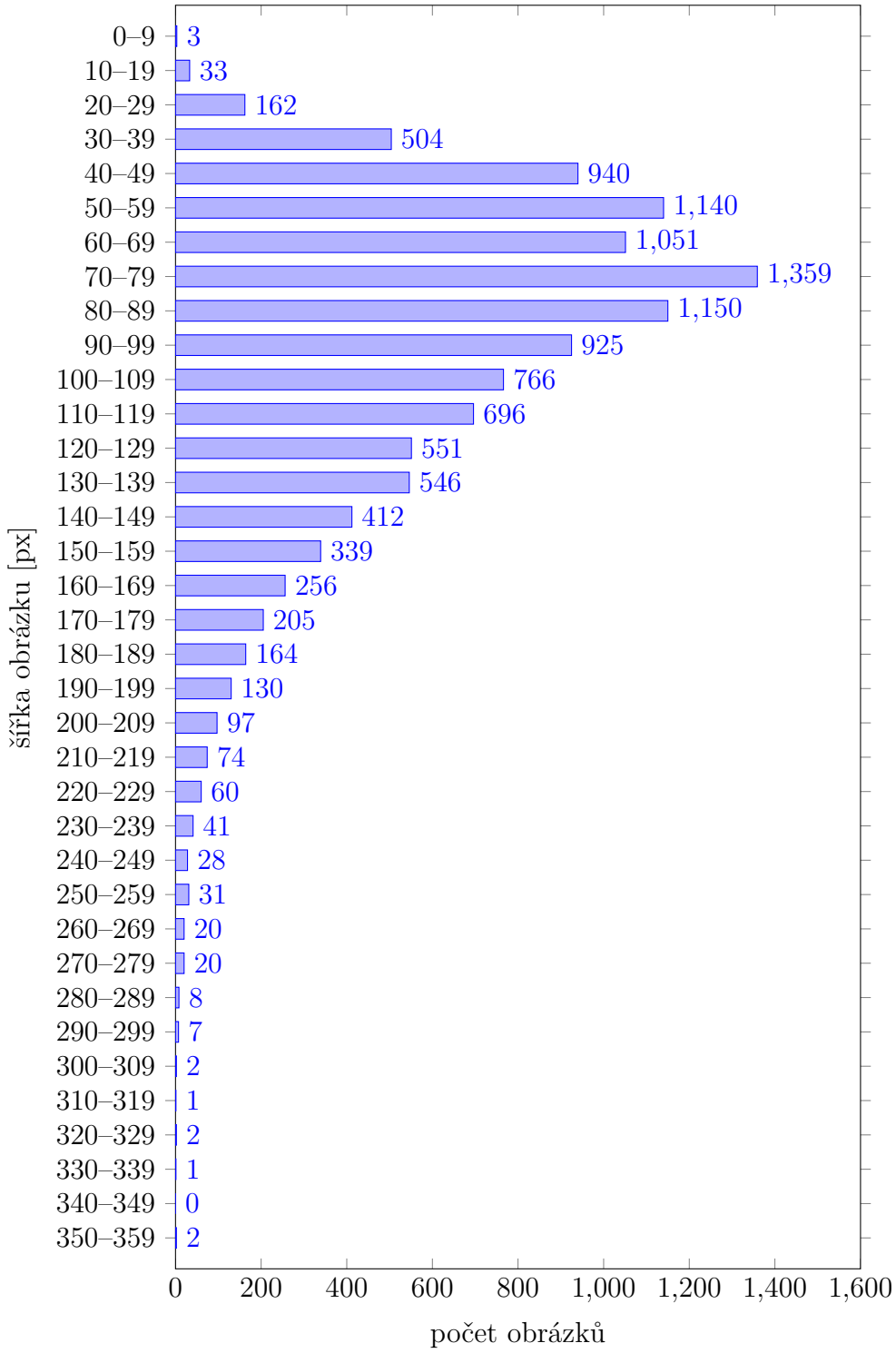
Rozšíření trénovací části bylo provedeno pomocí afinních transformací knihovny OpenCV (viz podsekce 5.1.3).

Afinní transformace byla vypočtena ze tří párů bodů, kde první z páru byly body (1, 1), (2, 1), (1, 2). Souřadnice druhého bodu z páru tvořily souřadnice prvního bodu vynásobené náhodným číslem z rovnoměrného rozdělení (0.9, 1.1).

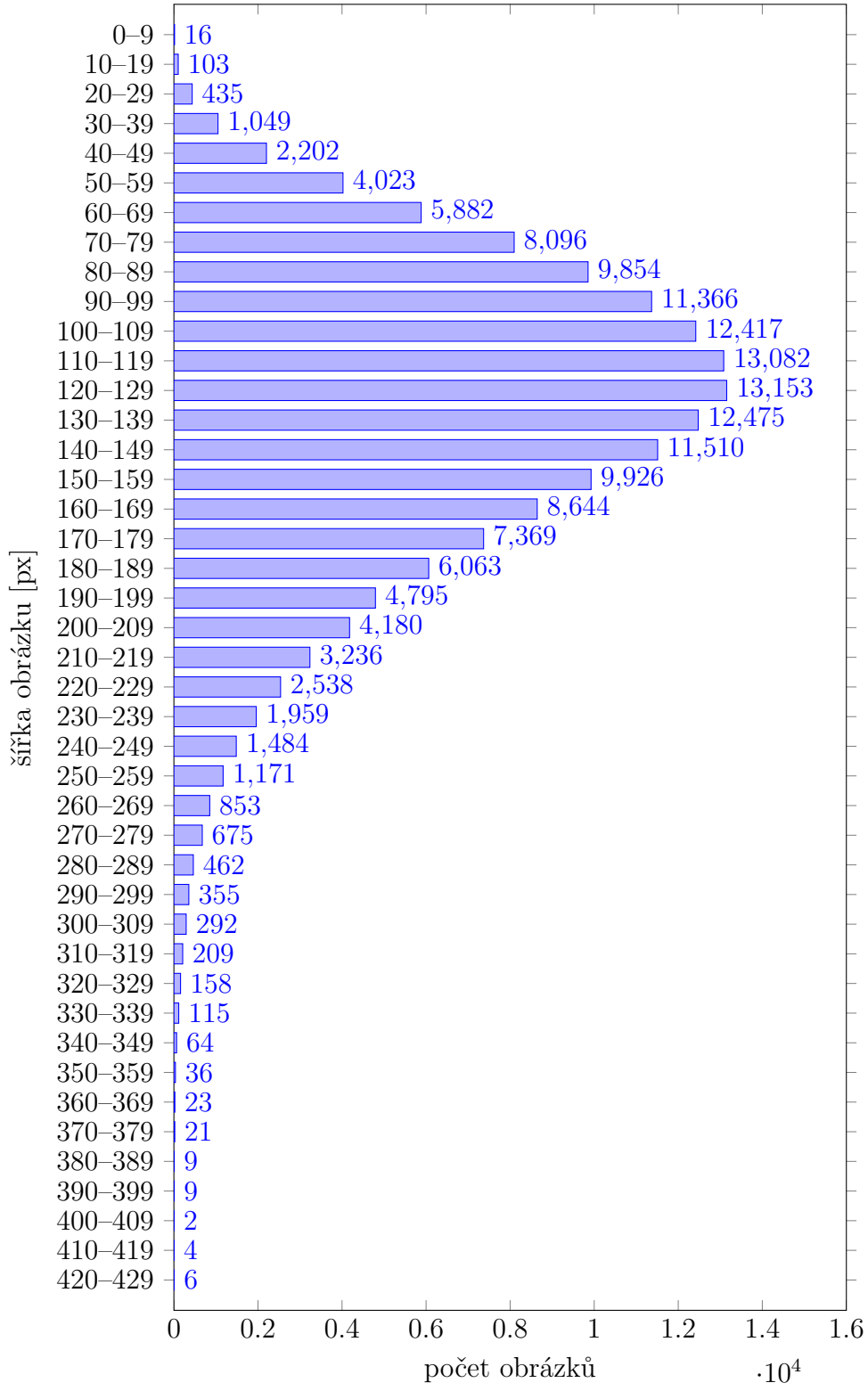
Takto vytvořené obrázky jsou přidány do trénovací části sady. Kromě původní sady byly jejím upravením vytvořeny ještě tři další pro porovnání.

Původní sada

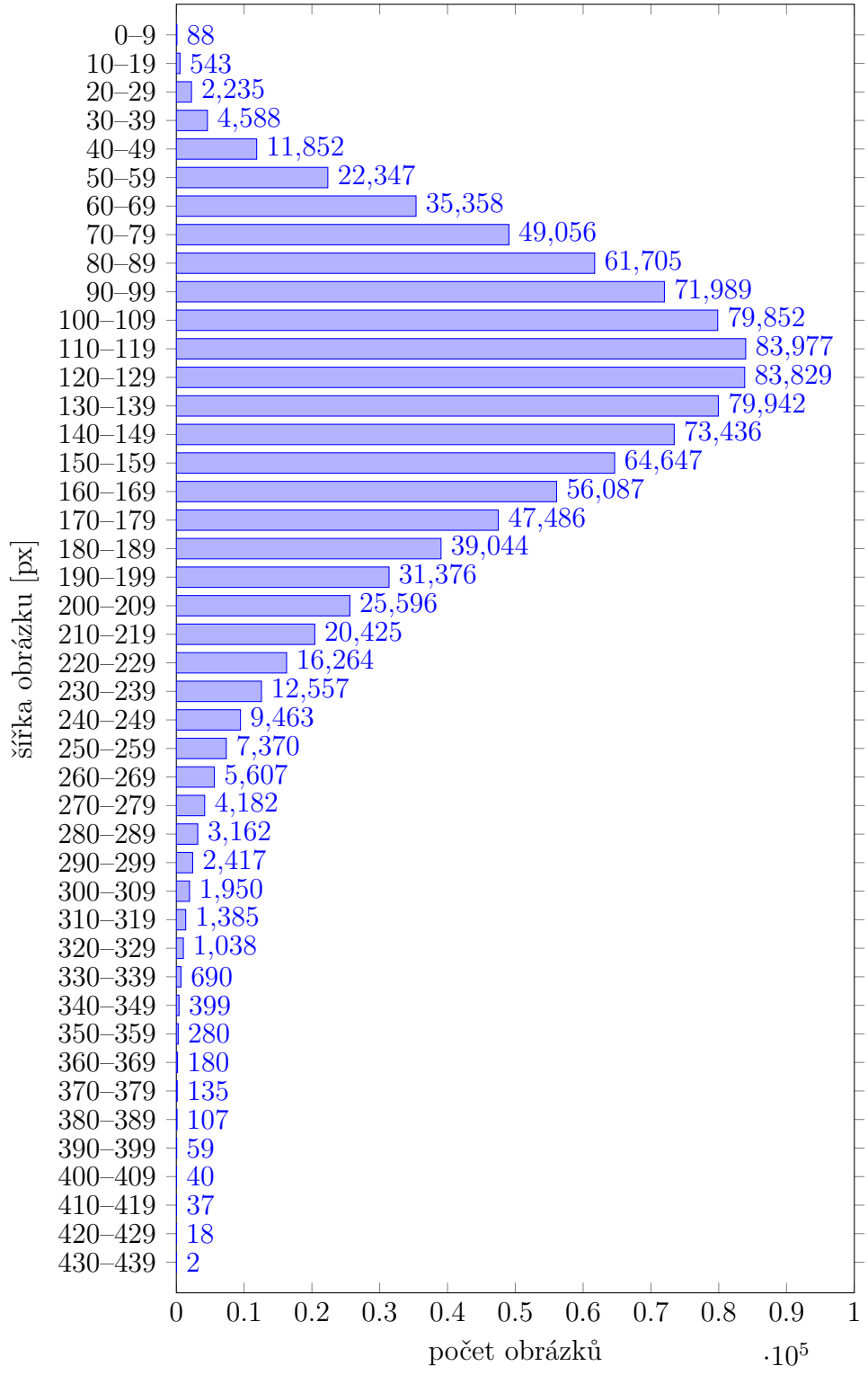
Původní sada (dále jako *orig*) je ponechána bez úprav. V této sadě nalezneme obrázky různých šířek, kde jejich rozložení v trénovací části odpovídá obrázku 6.3.



Obrázek 6.3: Rozložení trénovací části původní sady (*orig*) v závislosti na šířce obrázků



Obrázek 6.4: Rozložení trénovací části rozšířené sady (*aug50*) v závislosti na šířce obrázků



Obrázek 6.5: Rozložení trénovací části plně rozšířené sady (*aug320*) v závislosti na šířce obrázků

Rozšířená sada

Rozšířená sada (dále jako *aug50*) vznikla z původní výše popsaným rozšířením, kdy bylo každé slovo trénovací části doplněno na 50 příkladů. Distribuci trénovací části podle šířky obrázků popisuje obrázek 6.4.

Plně rozšířená sada

Plně rozšířená sada (dále jako *aug320*) vznikla z původní výše popsaným rozšířením, kdy bylo každé slovo v trénovací části doplněno na 320 příkladů, čímž bylo dosaženo stejného počtu vzorových příkladů pro každé slovo. Rozložení šířek obrázků této části popisuje obrázek 6.5.

Vyhlazená sada

Vyhlazená sada (dále jako *fix*) vznikla vyhlazením obrázků (viz podsekce 5.1.2) testovací, trénovací i validační části původní sady. Rozložení trénovací části je shodné s rozložením původní sady (viz obrázek 6.3).

6.4 Experimentální neuronová síť

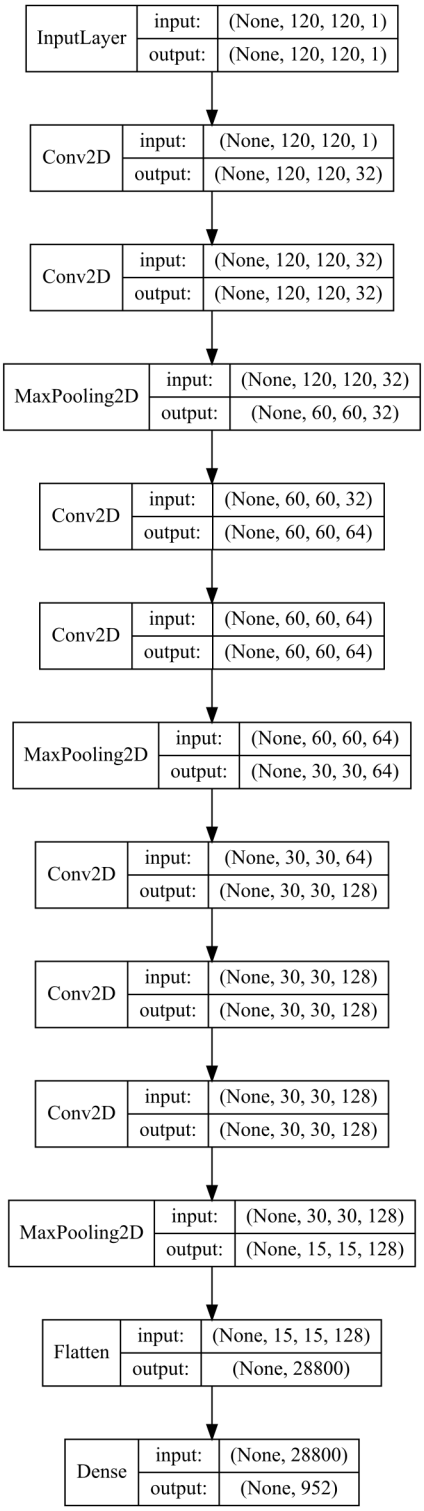
Architektura experimentální neuronové sítě (dále jako *conv128*) vychází z příkladu konvoluční sítě uvedeného v [3]. Pro zvolení parametrů sítě čerpáme dále inspiraci v [18]. Důvodem použití této architektury je ověření funkčnosti řešení a zjištění vlivu jednotlivých částí řešení na výsledky.

Vstup sítě

Na vstupu experimentální neuronové sítě (viz obrázek 6.6) je použit obrázek v rozlišení 120 x 120 bodů. Obrázek má jeden barevný kanál. Obrázky s jiným rozlišením byly pomocí knihovny OpenCV převedeny na toto rozlišení.

Výstup sítě

Výstupem je PHOC vektor s doporučenými úrovněmi 2, 3, 4 a 5 a výslednou dimenzí 952. Velikost znakové sady je 68, tvoří ji malé znaky anglické abecedy a 42 speciálních znaků, které se vyskytují v trénovací části sady. Jiné znaky jsou ignorovány, protože se nenachází v trénovací části a tedy není možné se je naučit.



Obrázek 6.6: Architektura experimentální sítě (*conv128*), tvar vstupu a výstupu vrstvy lze interpretovat jako: (velikost dávky, výška, šířka, hloubka) nebo (velikost dávky, počet neuronů)

Architektura

Architektura sítě se skládá ze sedmi konvolučních vrstev (3 x 3 filtr, 1 x 1 stride a padding), které na obrázku 6.6 představují Conv2D vrstvy. Aktivační funkcí je funkce ReLU (viz obrázek 3.7). Počet filtrů je hodnota u poslední dimenze výstupního tvaru na obrázku 6.6, jsou to hodnoty 32, 64 a 128.

Za druhou, čtvrtou a sedmou konvoluční vrstvou následuje vždy jedna max pooling vrstva (2 x 2 poolsize, 2 x 2 stride), která je na obrázku jako MaxPooling2D.

Jako poslední je jedna fully-connected vrstva tvořená Flatten a Dense. Aktivační funkcí je sigmoida (viz obrázek 3.6), protože nabývá hodnot od nuly do jedné a je tedy ideální pro predikci PHOC vektoru. Počet neuronů odpovídá dimenzi výstupního vektoru.

6.5 Vliv počtu filtrů konvoluční vrstvy

Tento experiment analyzuje přínos vyššího počtu filtrů v konvolučních vrstvách. Se základní *conv128* architekturou porovnáváme tři další architektury, z nichž dvě (*conv64* a *conv32*) se liší pouze v menším počtu filtrů.

První porovnávanou je *conv64*, kde je počet filtrů v konvolučních vrstvách poloviční oproti *conv128*, tzn. 16, 32 a 64 filtrů.

Druhou porovnávanou je *conv32*, kde počet filtrů tvoří čtvrtinu filtrů *conv128*.

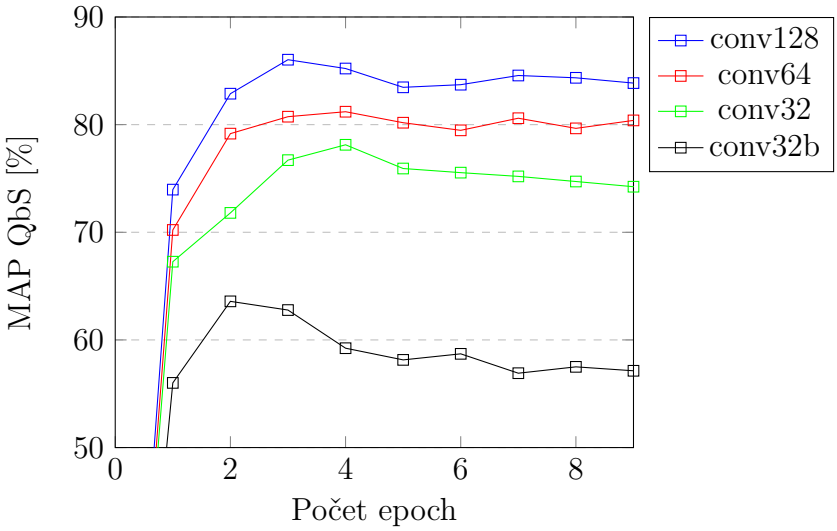
Vzhledem k tomu, že díky menšímu počtu filtrů se zmenšuje i počet parametrů sítě, je poslední porovnávanou sítí *conv32b*. Tato architektura je podobná *conv32*, ale chybí poslední max pooling vrstva, čímž se dosáhne podobného počtu parametrů sítě *conv128*.

Sítě jsou trénovány na trénovací části původní sady (*orig*) s dávkou 16 obrázků. Optimalizátorem je doporučený Adam (název odvozen z angl. adaptive moment estimation) [12]. Trénování je ukončeno po devíti epochách.

Vzdálenost vektorů je porovnávána pomocí eukleidovské vzdálenosti. Vyhodnocení probíhá na validační části sady po každé epoše trénování.

Závěrem je, že více filtrů přinese lepší výsledky, jak je vidět na obrázku 6.7, kde jsou jednotlivé sítě natrénovány a vyhodnoceny po každé epoše trénování.

Překvapením může být vynechání poslední max pooling vrstvy u *conv32b*, které přineslo oproti *conv32* neočekávané zhoršení, přestože počet parametrů *conv32b* je přibližně čtyřikrát větší. Velkou roli tedy hraje i architektura sítě.



Obrázek 6.7: Vliv počtu filtrů v konvoluční vrstvě na úspěšnost jednotlivých architektur během trénování sítě

6.6 Vliv počtu konvolučních vrstev

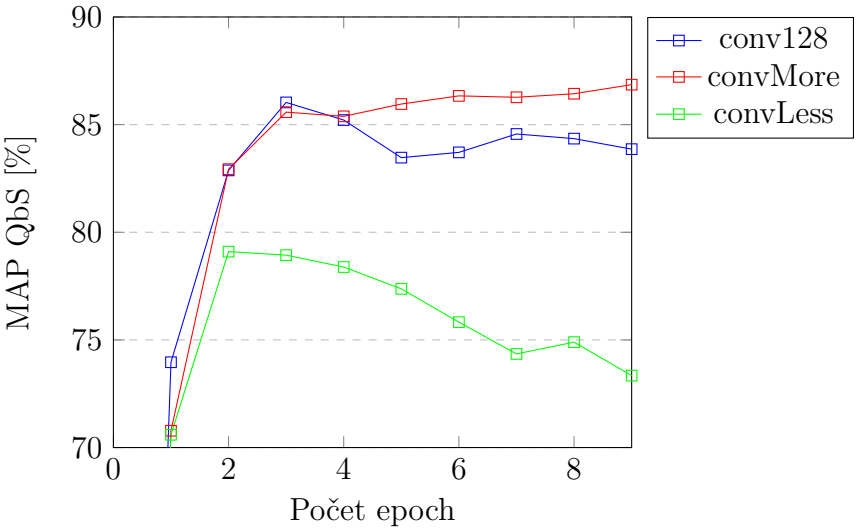
Tento experiment hledá vhodný počet konvolučních vrstev. Se základní *conv128* jsou porovnány další tři architektury.

První je *convMore*, která má oproti *conv128* o dvě konvoluční vrstvy více. Před poslední max pooling vrstvou jsou dvě konvoluční vrstvy místo tří a za poslední max pooling vrstvou jsou přidány tři konvoluční vrstvy se 128 filtry.

Další je *convLess*, kde jsou oproti *conv128* odstraněny dvě poslední konvoluční vrstvy.

Poslední *convHalf* architektura má pouze tři konvoluční vrstvy. Oproti *conv128* je před max pooling vrstvou vždy pouze jedna konvoluční vrstva. Tato architektura se však ukázala jako nedostatečná s nulovou úspěšností, proto ve výsledcích (viz obrázek 6.8) není zahrnuta.

Trénování a vyhodnocení probíhá stejně jako u předchozího experimentu. Výsledky (viz obrázek 6.8) ukazují, že vyšší počet konvolučních vrstev zvyšuje kapacitu sítě. Z rozdílu mezi *convMore* a *conv128* lze považovat tyto počty vrstev za dostatečné. Nejlepší úspěšnosti dosáhla *convMore*, kde je patrný také lepší průběh během trénování.

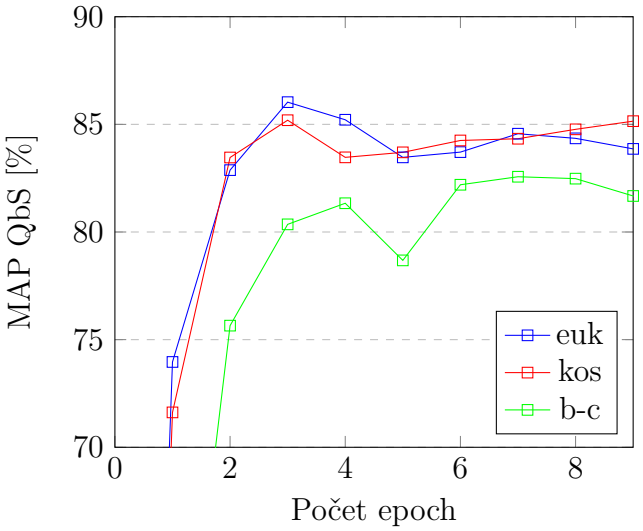


Obrázek 6.8: Vliv počtu konvolučních vrstev na úspěšnost jednotlivých sítí během trénování

6.7 Vliv porovnávání vektorů

V tomto experimentu se chystáme porovnat eukleidovskou (*euk*), kosinovou (*kos*) vzdálenost a Bray-Curtis podobnost (*b-c*). Architektura je totožná s *conv128*. Pro natrénovanou síť jsou na validační části vypočteny úspěšnosti za použití *euk*, *kos* a *b-c* pro porovnávání vektorů.

Obrázek 6.9 ukazuje, že Bray-Curtis podobnost přináší horší výsledky.

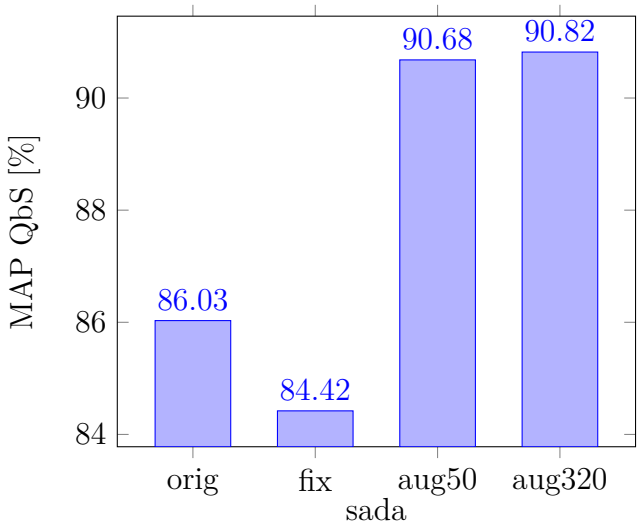


Obrázek 6.9: Úspěšnost *conv128* na validační části sady při použití eukleidovské (*euk*), kosinové (*kos*) vzdálenosti a Bray-Curtis (*b-c*) podobnosti

Kosinová vzdálenost dosáhla zajímavých výsledků, které jsou s eukleidovskou srovnatelné. Výsledkem experimentu je zvolení eukleidovské vzdálenosti pro porovnávání vektorů.

6.8 Vliv rozšíření datové sady

Zde nás zajímá, jaký vliv mají různé úpravy dat. Základní architektura *conv128* je natrénována a vyhodnocena na původní (*orig*), rozšířené (*aug50* a *aug320*) a vyhlazené (*fix*) sadě.



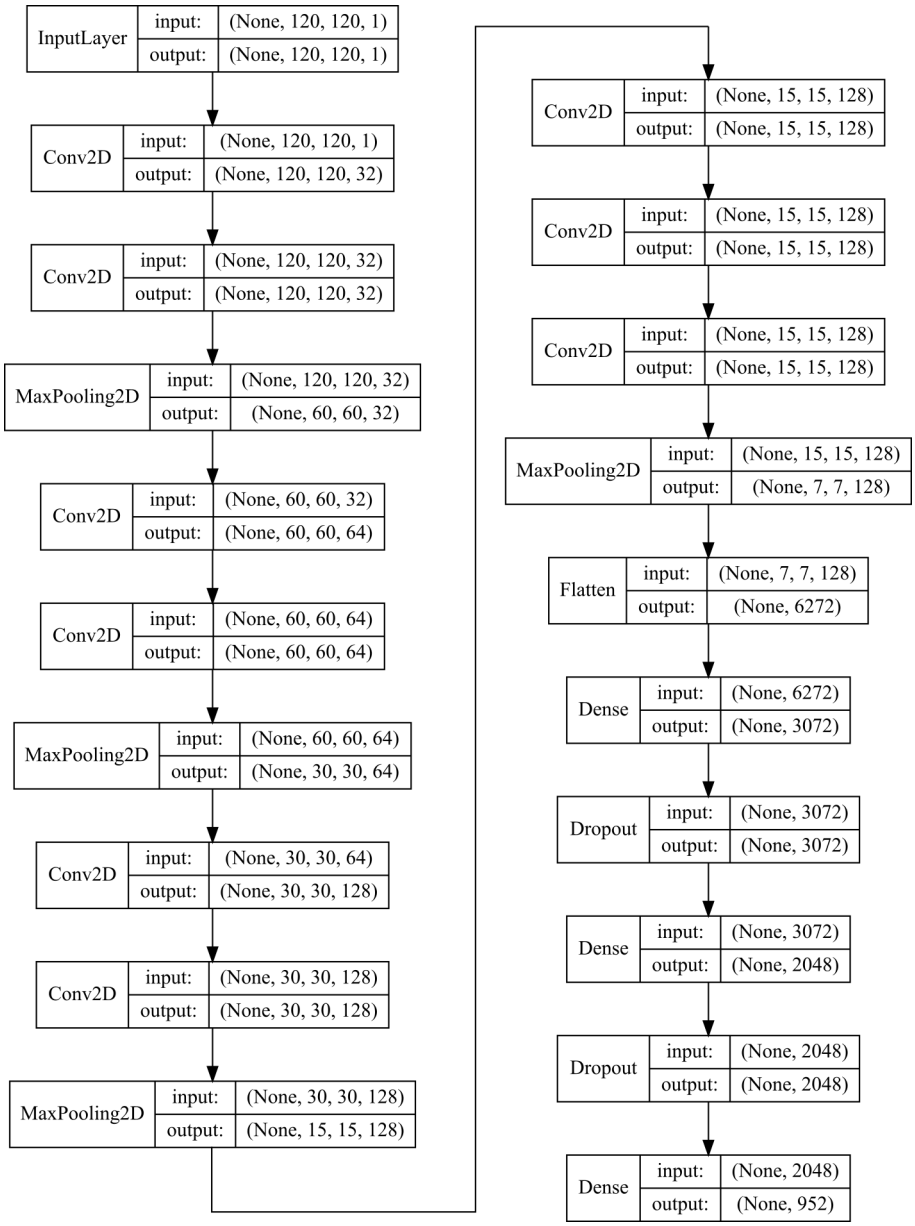
Obrázek 6.10: Úspěšnost *conv128* na jednotlivých sadách

Jelikož jsou sady složeny z různého počtu obrázků, je i počet úprav vah sítě během jedné epochy různý. Proto nás zajímá pouze maximální dosažená úspěšnost.

Z výsledků (viz obrázek 6.10) můžeme usoudit, že vyhlazení obrázků není vhodné. Naopak rozšíření datové sady pomocí afinních transformací lze doporučit.

6.9 Konvoluční síť

Konvoluční síť (*cnn*) na základě provedených experimentů rozšiřuje experimentální architekturu, která si vedla poměrně dobře.



Obrázek 6.11: Architektura konvoluční sítě (*cnn*), tvar vstupu a výstupu vrstvy lze interpretovat jako: (velikost dávky, výška, šířka, hloubka) nebo (velikost dávky, počet neuronů)

Vstup sítě

Vstupem sítě je obrázek s původní výškou 120 pixelů, jeho šířka je převedena na 120 pixelů pomocí výše popsané funkce *resize* knihovny OpenCV. Tato hodnota je zvolena na základě analýzy distribuce rozšířených sad (viz obrázky 6.4 a 6.5), kde je patrné, že nejvíce obrázků a tedy i slov má přibližně tuto šířku a navíc dělí menší a větší obrázky v podobném poměru.

Výstup sítě

Výstupem sítě je PHOC vektor o velikosti 952, který je shodný s výstupem experimentální sítě (viz sekce 6.4).

Architektura

Architektura sítě (viz obrázek 6.11) je složena z devíti konvolučních vrstev (32, 64 nebo 128 filtrů o velikosti 3 x 3, 1 x 1 stride, padding a ReLU aktivační funkce). Za druhou, čtvrtou, šestou a devátou konvoluční vrstvou následuje vždy jedna max pooling vrstva (2 x 2 poolsize, 2 x 2 stride). Za nimi následují tři fully-connected vrstvy (první dvě mají 3072 a 2048 neuronů s ReLU aktivační funkcí, poslední výstupní má 952 neuronů a sigmoidu jako aktivační funkci). Mezi fully-connected vrstvami jsou dvě dropout vrstvy (pravděpodobnost, že neuron zůstane aktivní, je 0,5).

Trénování

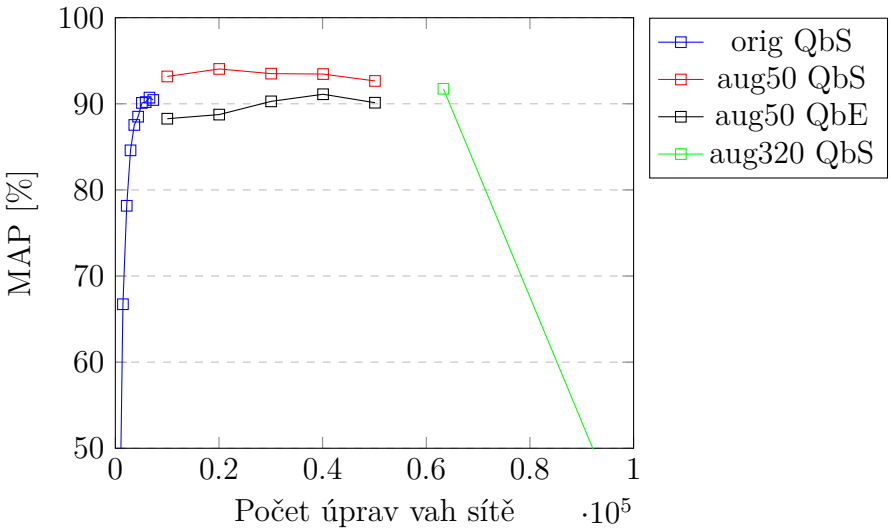
Trénování sítě je provedeno s dávkou 16 obrázků, jako optimalizátor je použit doporučený Adam a výstupní vektory jsou porovnávány pomocí eukleidovské vzdálenosti.

Jelikož se v experimentech ukázalo jako výhodné trénování na rozšířených sadách, začali jsme s trénováním na *aug320*, kde jedna epocha odpovídá 63 300 úpravám vah sítě. Zde se ovšem ukázalo, že navzdory rozšíření a použití dropout vrstvy dochází k přetrénování a během druhé epochy úspěšnost spadla k nule. Důvodem může být vysoký počet úprav vah případně i velké množství rozšířených obrázků, které si mohou být velice podobné.

Proto byl model natrénován i na sadě *aug50*, kde jedna epocha odpovídá 10 020 úpravám vah sítě. Pro úplnost jsme síť natrénovali i na sadě *orig*, kde jedna epocha představuje 732 úprav vah.

Zobrazení výsledků je obtížné, protože jsou sady různé, ale na obrázku 6.12 můžeme vidět, že nejlepších výsledků lze dosáhnout na sadě *aug50*,

kde byl po druhé epoše výsledek na validační části sady 94 % MAP QbS a 88,7 % MAP QbE.



Obrázek 6.12: Úspěšnost *cnn* sítě na jednotlivých sadách v závislosti na počtu úprav vah sítě (kvůli přehlednosti je MAP QbE zobrazen pouze pro nejúspěšnější sadu *aug50*)

6.10 Konvoluční síť se Spatial Pyramid Pooling vrstvou

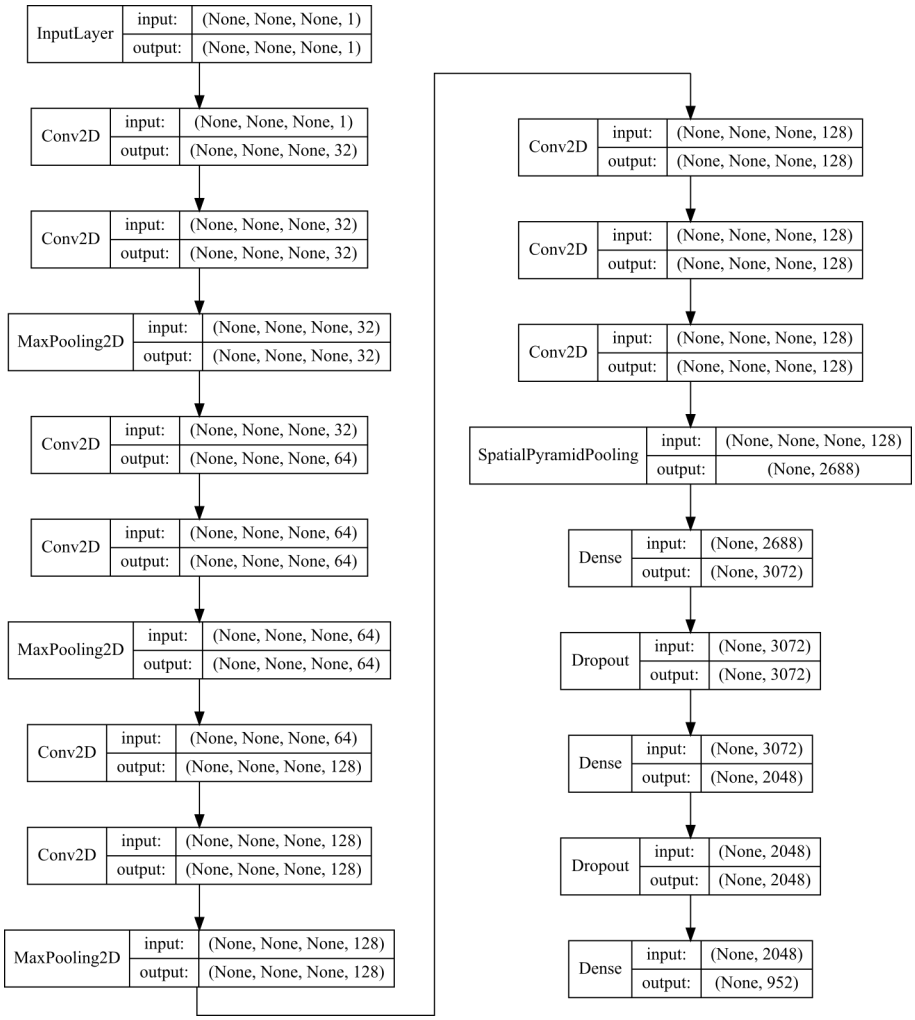
Konvoluční síť se Spatial Pyramid Pooling vrstvou (*spp*) vychází z *cnn* a PHOCNet (viz [18]). Vstupem této sítě mohou být obrázky s rozdílným rozlišením.

Vstup sítě

Díky použití Spatial Pyramid Pooling vrstvy je vstupem sítě obrázek s libovolnými rozměry. Jediná možná změna z důvodu později popsané architektury je zvětšení obrázku, pokud je obrázek menší než 32 x 32 pixelů.

Výstup sítě

Výstupem sítě je PHOC vektor o velikosti 952, který je shodný s výstupem experimentální sítě (viz sekce 6.4).



Obrázek 6.13: Architektura konvoluční sítě se Spatial Pyramid Pooling vrstvou (*spp*), tvar vstupu a výstupu vrstvy lze interpretovat jako: (velikost dávky, výška, šířka, hloubka) nebo (velikost dávky, počet neuronů)

Architektura

Architektura (viz obrázek 6.13) je parametry shodná s *cnn*. Rozdíl této architektury oproti *cnn* je v použití výše popsané Spatial Pyramid Pooling vrstvy (1 x 1, 2 x 2 a 4 x 4 mřížka) místo poslední max pooling vrstvy.

Trénování

Protože je nutné v knihovně Keras použít stejný rozměr dat v jedné dávce, byla trénovací část sady *aug50* rozdělena na 4 přibližně stejně velké části podle velikosti obrázků (viz tabulka 6.1). Obrázky v jednotlivých částech potom byly roztaženy na přibližně průměrnou hodnotu.

To může být nevýhodou, ale změna velikosti je málokdy větší než 30 pixelů. Navíc jsou velikosti zmenšovány i zvětšovány, což ve výsledku může přinést více vzorů pro jednotlivé znaky.

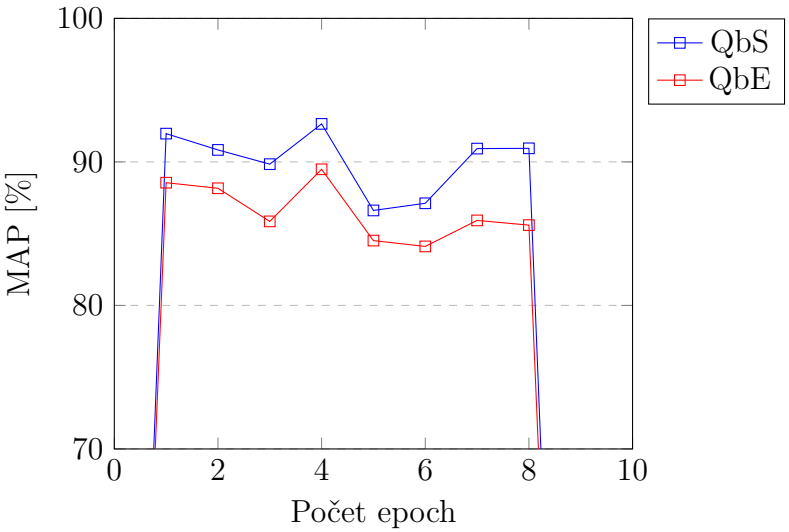
Šířky obrázků [px]	Počet obrázků	Šířka po roztažení [px]
0 - 99	43 026	70
100 - 129	38 652	110
130 - 169	42 555	140
170 a více	36 088	200

Tabulka 6.1: Rozdělení trénovací části sady pro síť se Spatial Pyramid Pooling vrstvou

Síť je natrénována na takto připravené sadě s dávkou 16 obrázků. V průběhu jedné epochy se postupně vystřídají všechny části. Jako optimalizátor je použit doporučený Adam a výstupní vektory jsou porovnávány pomocí eukleidovské vzdálenosti.

Vyhodnocení poté probíhá na základě obrázků, které jsou na vstup sítě přivedeny bez úprav s původním rozměrem (případně mohou být zvětšeny).

Na výsledcích (viz obrázek 6.14) je vidět, že během deváté epochy došlo k přetrénování sítě podobně jako u *cnn*. Nejlepší výsledek na validační sadě byl po čtvrté epoše s hodnotou 92,65 % MAP QbS a 89,48 % MAP QbE.



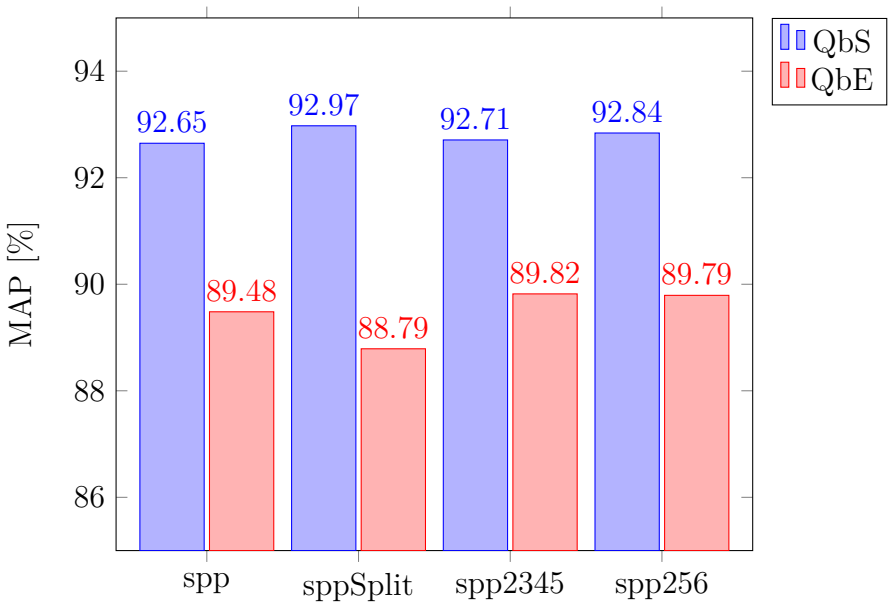
Obrázek 6.14: Úspěšnost *spp* sítě během trénování

Experimenty pro srovnání

Vzhledem k rozdělení trénovací části byl pro vyhodnocení úspěšnosti použit další způsob, kde byly obrázky pro vyhodnocení upraveny tak, aby odpovídaly úpravám pro trénovací část (viz tabulka 6.1). Výsledky na obrázku 6.15, kde je tento způsob pojmenován jako *sppSplit*, ovšem neukazují významné zlepšení.

Další pokusy o lepší výsledky byly provedeny s architekturami *spp256* a *spp2345*. Oproti *spp* je u *spp256* použito 256 filtrů v posledních třech konvolučních vrstvách. U *spp2345* je oproti *spp* pro Spatial Pyramid Pooling vrstvu použita mřížka 2 x 2, 3 x 3, 4 x 4 a 5 x 5.

Obě tyto architektury dosáhly mírného zlepšení (viz obrázek 6.15) za cenu zvýšení počtu parametrů na přibližně dvojnásobnou hodnotu oproti *spp*. Trénování a vyhodnocení však doprovázely problémy s nedostatkem paměti grafické karty. Proto je nadále použita síť *spp* jako paměťově úspornější.



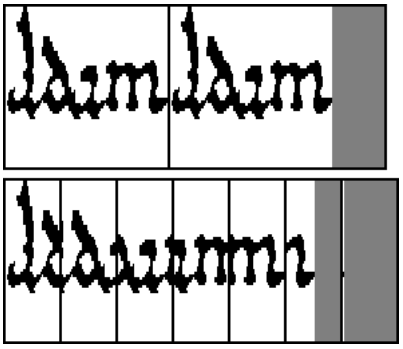
Obrázek 6.15: Nejlepší úspěšnosti architektur *spp*, *spp2345*, *spp256* a porovnání s *sppSplit* v případě načítání obrázků podle rozdělení trénovací části i při vyhodnocení

6.11 Síť s konvoluční LSTM

Síť s konvoluční LSTM je dalším způsobem, jak se vypořádat s rozdílnými rozměry vstupních obrázků. V tomto případě k tomu používáme konvoluční LSTM vrstvu implementovanou v knihovně Keras jako *ConvLSTM2D*.

Vstup sítě

Použití *ConvLSTM2D* vrstvy je možné pro sekvence snímků. To je ideální např. pro videa, ale naším vstupem je obrázek.



Obrázek 6.16: Příprava vstupu pro *convlstm* síť: původní obrázek (vlevo nahoře) slova Adam, jeho doplnění na požadovanou šířku (vpravo nahoře) a vytvoření sekvencí (dole)

Aby bylo možné tuto vrstvu použít pro naše potřeby, rozhodli jsme se pomocí OpenCV upravit vstupy dle obrázku 6.16. Zde můžeme vidět původní černobílý obrázek, který je oříznut nebo doplněn na šířku 160 pixelů. Pro případné doplnění obrázku je použita šedá barva (hodnota 127 pro RGB kanály) z důvodu, že nenese žádnou informaci. Síť by tak měla být schopna se tuto informaci lépe naučit. Z takto upraveného obrázku následně pomocí okénka o šířce 40 pixelů s posunem 20 pixelů získáme 7 sekvencí, které budou tvořit vstup sítě.

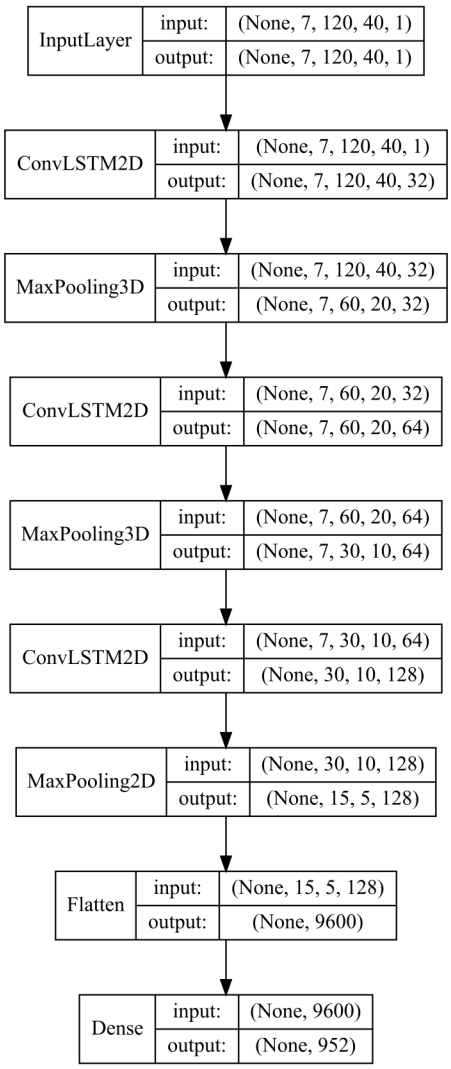
Výstup sítě

Výstupem sítě je PHOC vektor o velikosti 952. Výstup je shodný s výstupem experimentální sítě (viz sekce 6.4).

Architektura

Architektura sítě (viz obrázek 6.17) je složena ze tří *ConvLSTM2D* vrstev (32, 64 a 128 filtrů o velikosti 3 x 3, padding a výchozí aktivační funkce), které představují výše popsanou konvoluční LSTM.

Výstupem prvních dvou *ConvLSTM2D* vrstev jsou jednotlivé stavy. Tyto jednotlivé stavy jsou pomocí max pooling vrstvy (2 x 2 poolsize, 2 x 2 stride) zmenšeny a přivedeny na vstup další *ConvLSTM2D* vrstvě. Výstupem poslední *ConvLSTM2D* vrstvy je pouze poslední stav, na který opět navazuje



Obrázek 6.17: Architektura sítě s konvoluční LSTM (*lstm*), tvar vstupu a výstupu vrstvy lze interpretovat jako: (velikost dávky, počet sekvencí, výška, šířka, hloubka), (velikost dávky, výška, šířka, hloubka) nebo (velikost dávky, počet neuronů)

max pooling vrstva (2 x 2 poolsize, 2 x 2 stride). Následuje jedna fully-connected vrstva (952 neuronů a sigmoida jako aktivační funkce).

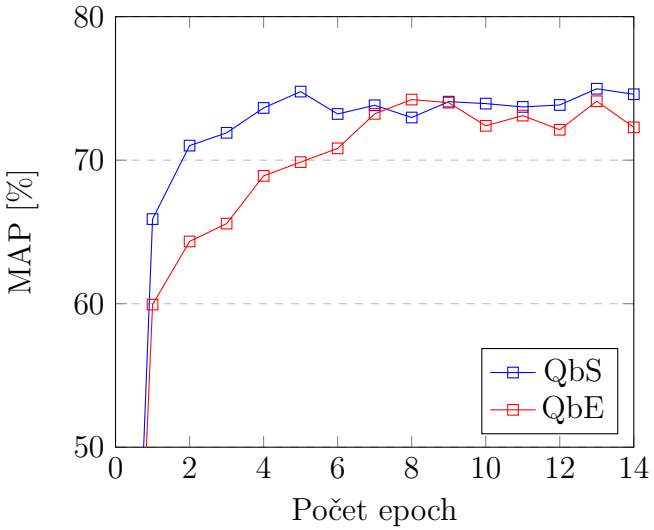
Trénování

Síť je trénována na sadě *aug50* s dávkou 8 obrázků kvůli paměťové náročnosti. Jedna epocha tedy představuje dvojnásobný počet úprav vah oproti *cnn* a *spp*. U optimalizátoru Adam je nastaven dvojnásobný learning rate oproti výchozí hodnotě. Vektory jsou porovnávány pomocí eukleidovské

vzdálenosti.

Naše očekávání bylo poměrně nízké, protože tato vrstva se používá většinou za účelem rozpoznávání pohybů. Navíc jsme nenarazili na použití konvoluční LSTM pro řešení tohoto problému.

Navzdory tomu nejsou výsledky (viz obrázek 6.18) až tak špatné a na validační sadě dosahují po třinácté epoše 74,9 % MAP QbS a 74,1 % MAP QbE.



Obrázek 6.18: Výsledky *convlstm* sítě během trénování

Nevýhodou této architektury oproti *cnn* a *spp* je vedle úspěšnosti také vyšší časová náročnost trénování.

6.12 Souhrn výsledků

Pro závěrečné vyhodnocení byly vybrány výše popsané modely *cnn* po druhé epoše, *spp* po čtvrté epoše a *lstm* po třináctém epoše trénování, protože po těchto epochách dosahovaly nejlepších výsledků na validační sadě.

Vyhodnocení probíhá na testovací části sady z databáze Parzival. Je použita eukleidovská vzdálenost. Vypočteny jsou metriky *MAP QbS*, *MAP QbE* a *acc*.

Protože v některých článcích vyhodnocují MAP pouze pro dotazy obsažené v trénovací části sady, přidali jsme i výsledky pro tento postup jako *T-MAP QbS* a *T-MAP QbE*.

Vzhledem k úspěšnostem na validační části dopadly výsledky (viz tabulka 6.2) podle očekávání i na datech, která model předtím neviděl. Jako nejlepší se ukázala síť *cnn*, která dokázala překonat sítě *spp* a *lstm*.

Pro srovnání v [13] zmiňují u HMM 88,15 % a u DTW 36,85 % MAP QbS. Pro přímé porovnání mezi neuronovými sítěmi na databázi Parzival se nám bohužel nepodařilo nalézt odpovídající článek.

	MAP QbS	MAP QbE	acc	T-MAP QbS	T-MAP QbE
cnn	92,62 %	90,01 %	90,51 %	95,28 %	90,63 %
spp	90,54 %	87,57 %	85,19 %	92,34 %	87,78 %
lstm	70,92 %	67,86 %	78,73 %	81,93 %	70,29 %

Tabulka 6.2: Výsledky naměřené na testovací části sady z databáze Parzival

Na výsledcích pro QbS dotazy (viz obrázky 6.19, 6.20 a 6.21) je patrné, že relevance odpovědí pro uživatele bude poměrně vysoká. Podobně je tomu i u QbE dotazu (viz obrázek 6.22).

Na druhou stranu u dalšího QbE dotazu pro stejné slovo (viz obrázek 6.23) jsou navráceny irelevantní odpovědi. Zde je však pro vyhledávání použit obrázek, kde může být srovnání oproti tomu na obrázku 6.22 problematické i pro člověka.

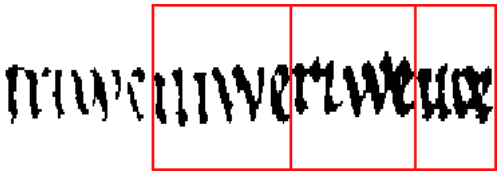
Tyto odlišné obrázky, které většinou nemají podobného zástupce v trénovací sadě, mohou být následně při vyhledávání v dokumentech vraceny jako nepravděpodobné.

Obrázek 6.19: Odpověď na QbS dotaz pro slovo *aventivre* seřazené zleva (vlevo nejpravděpodobnější, zelená je správně, červená je chybně)

Obrázek 6.20: Tři nejlepší odpovědi na QbS dotaz pro slovo *gein* seřazené zleva od nejlepší (zelená je správně)

Obrázek 6.21: Tři nejlepší odpovědi na QbS dotaz pro slovo *triwe* seřazené zleva od nejlepší (zelená je správně)

Obrázek 6.22: QbE dotaz pro obrázek slova *triwe*: vlevo je obrázek pro dotaz, následují tři nejlepší odpovědi seřazené zleva (zelená je správně)



Obrázek 6.23: QbE dotaz pro odlišný obrázek slova *triwe*: vlevo je obrázek pro dotaz, následují tři nejlepší odpovědi seřazené zleva (červená je chybně, odpovědi zleva *niwe*, *riwe* a *uor*)

7 Závěr

Cílem práce byl návrh a implementace dvou metod, které umožní vyhledávání slov v ručně psaném textu. Předpokladem je hotová segmentace dokumentu na obrázky slov.

V první části bakalářské práce byly zmapovány dostupné datové kolekce, které je možné použít pro tento problém. Následovalo seznámení s možnými řešeními, studium principu neuronových sítí a oblasti získávání informací (angl. information retrieval).

Byla navržena tři různá řešení založená na neuronových sítích. První metodu představuje konvoluční síť, kde jsou vstupní obrázky upraveny na stejný rozměr. Druhá metoda umožňuje vstup v podobě obrázku s různými rozměry díky použití Spatial Pyramid Pooling vrstvy. Nad rámec zadání byla navržena třetí metoda, kde je vstupní obrázek převeden pomocí posuvného okna na sekvence. Tyto sekvence jsou zpracovány sítí s konvoluční LSTM.

Dále byly provedeny experimenty, kde byl porovnán vliv na úspěšnost jednotlivých postupů týkajících se přípravy dat, architektury sítě a porovnávání výstupních vektorů. V některých případech bylo dosaženo překvapivých výsledků, které například ukazují, že snížení množství předávané informace může vést k výrazně lepší úspěšnosti sítě. Díky tomu byl vytvořen teoretický i praktický základ, který pomohl dosáhnout výborných výsledků pro vyhledávání vzorů v dokumentech.

Síť se Spatial Pyramid Pooling vrstvou dosáhla 90,54 % MAP QbS a 87,57 % MAP QbE. Další síť s konvoluční LSTM si vedla ve srovnání s ostatními hůře a dosáhla 70,92 % MAP QbS a 67,86 % MAP QbE. Nejlepší výsledky 92,62 % MAP QbS a 90,01 % MAP QbE byly dosaženy s konvoluční sítí. S touto architekturou se podařilo na dotazy podle vzoru i řetězce vrátit relevantní odpovědi v datové kolekci Parzival. Ukázalo se tak, že je možné tento systém použít v praxi, pokud bude splněna podmínka úspěšné segmentace dokumentu a připraveno dostatek trénovacích dat.

Tato práce může sloužit jako dobrý základ pro případné rozšíření. Zlepšení by mohlo přinést další rozšíření trénovací sady, protože má na úspěšnost neuronové sítě velký vliv, jak se potvrdilo i v provedených experimentech. Lepších výsledků by bylo možné také dosáhnout návrhem a použitím jiné reprezentace než PHOC vektor, spojením navržených klasifikátorů dohromady nebo použitím robustnějších architektur, které byly v této práci limitovány 2 GB paměti grafické karty. Zajímavé poznatky by mohlo přinést testování na dalších databázích a vyřešení segmentace dokumentu na obrázky slov.

Přehled zkratek

- acc - evaluační metrika přesnost (accuracy)
- aug50 - rozšířená sada z databáze Parzival, kde jsou slova v trénovací části doplněna na 50 příkladů
- aug320 - rozšířená sada z databáze Parzival, kde jsou slova v trénovací části doplněna na 320 příkladů
- b-c - Bray-Curtis podobnost
- cnn - konvoluční síť
- conv128 - experimentální síť
- euk - eukleidovská vzdálenost
- fix - vyhlazená sada z databáze Parzival
- kos - kosinová vzdálenost
- lstm - síť s konvoluční LSTM
- MAP - Mean Average Precision metrika
- orig - původní sada z databáze Parzival
- QbE - dotaz podle vzoru (z anglického Query by Example)
- QbS - dotaz podle řetězce (z anglického Query by String)
- spp - konvoluční síť se Spatial Pyramid Pooling vrstvou

Literatura

[1] *OpenCV (Open Source Computer Vision Library)* [online]. [cit. 2018/01/03]. About OpenCV. Dostupné z: `https://opencv.org/about.html`.

[2] *The Microsoft Cognitive Toolkit (CNTK)* [online]. [cit. 2018/04/24]. Dostupné z: `https://www.microsoft.com/en-us/cognitive-toolkit/`.

[3] *CS231n Convolutional Neural Networks for Visual Recognition* [online]. [cit. 2017/11/28]. Dostupné z: `http://cs231n.github.io/`.

[4] *Keras* [online]. [cit. 2018/04/14]. Dostupné z: `https://keras.io/`.

[5] *CS 276 / LING 286: Information Retrieval and Web Search* [online]. [cit. 2018/03/27]. Introduction to Information Retrieval Evaluation. Dostupné z: `https://web.stanford.edu/class/cs276/handouts/EvaluationNew-handout-1-per.pdf`.

[6] *MetaCentrum VO* [online]. [cit. 2018/03/26]. Dostupné z: `https://metavo.metacentrum.cz/`.

[7] *NumPy* [online]. [cit. 2018/04/24]. Dostupné z: `http://www.numpy.org/`.

[8] *Biologické algoritmy (5) - Neuronové sítě* [online]. [cit. 2018/01/03]. Učení – Backpropagation. Dostupné z: `https://www.root.cz/clanky/biologicke-algoritmy-5-neuronove-site/`.

[9] *TensorFlow* [online]. [cit. 2018/04/14]. Dostupné z: `https://www.tensorflow.org/`.

[10] *Theano* [online]. [cit. 2018/04/24]. Dostupné z: `http://deeplearning.net/software/theano/`.

[11] Almazán, J. et al. Word spotting and recognition with embedded attributes. *IEEE transactions on pattern analysis and machine intelligence*. 2014, 36, 12, s. 2552–2566.

[12] Brownlee, J. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning* [online]. [cit. 2018/04/24]. Dostupné z: `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/`.

[13] Fischer, A. et al. Lexicon-free handwritten word spotting using character HMMs. *Pattern Recognition Letters*. 2012, 33, 7, s. 934 – 942. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2011.09.009. Dostupné z:

http:
//www.sciencedirect.com/science/article/pii/S0167865511002820.
Special Issue on Awards from ICPR 2010.

[14] He, K. et al. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. 06 2014, 37.

[15] Marti, U.-V. – Bunke, H. The IAM-database: An English sentence database for offline handwriting recognition. 11 2002, 5, s. 39–46.

[16] Rath, T. M. – Manmatha, R. Word spotting for historical documents. *International Journal of Document Analysis and Recognition (IJDAR)*. Apr 2007, 9, 2, s. 139–152. ISSN 1433-2825. doi: 10.1007/s10032-006-0027-8. Dostupné z: https://doi.org/10.1007/s10032-006-0027-8.

[17] Shi, X. et al. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *CoRR*. 2015, abs/1506.04214. Dostupné z: http://arxiv.org/abs/1506.04214.

[18] Sudholt, S. – Fink, G. A. PHOCNet : A Deep Convolutional Neural Network for Word Spotting in Handwritten Documents. In *Proc. Int. Conf. on Frontiers in Handwriting Recognition*, 2016.

[19] Zhong, Z. et al. SpottingNet: Learning the Similarity of Word Images with Convolutional Neural Network for Word Spotting in Handwritten Historical Documents. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, s. 295–300, Oct 2016. doi: 10.1109/ICFHR.2016.0063.

Přílohy

Obsah přiloženého DVD

K bakalářské práci je přiložen DVD disk, který obsahuje archiv *bp.zip*. Tento archiv obsahuje následující položky:

- db - přiložené sady vytvořené z databáze Parzival
- sw - programové vybavení, skripty pro jednotlivé experimenty a soubory s nejúspěšnějšími váhami architektur *cnn*, *spp* a *lstm*
- text - zdrojové soubory LaTeX, obrázky a tato práce v PDF
- readme.txt - podrobný obsah DVD a uživatelská příručka

Uživatelská příručka

V práci bylo provedeno velké množství experimentů, proto jsou ve složce *sw* přiloženy skripty, pomocí kterých je možné výše zmíněné experimenty zopakovat. Výjimku tvoří experimenty provedené na sadě *aug320*, která není z důvodu velikosti přiložena. Dále je přiložen skript pro QbS a QbE dotaz a skript pro vyhodnocení nejúspěšnějších sítí.

Podmínky pro spuštění skriptů

Pro spuštění je vyžadován Python (doporučená verze 3.6.2) s balíky h5py, Keras, matplotlib, numpy, opencv-python, scipy a tensorflow.

Dále je nutné umístit obsah DVD (složky *db* a *sw*) na místo s povolením zápisu a zkontrolovat cesty v *sw/settings/paths.py*.

Spuštění skriptů

Spuštění skriptů se předpokládá z kořenové složky *sw*. Spuštění všech skriptů je bez parametrů. Výstup skriptů je ve složce *out*.

Skripty pro experimenty

Skripty pro opakování provedených experimentů provedou natrénování sítě na dané sadě a její vyhodnocení. Skripty jsou pojmenovány jako:

```
<síť>_<sada>.py
```

Výstupem skriptů jsou soubory s váhami sítě po jednotlivých epochách trénování:

```
<síť>_<sada>-<epocha>.hdf5
```

a logovací soubor s výsledky na validační sadě po jednotlivých epochách trénování:

```
<síť>_<sada>_<evaluační metrika>_<porovnávání vektorů>.log
```

Skript pro vyhodnocení nejúspěšnějších sítí

Skript *final_eval.py* vyhodnotí nejúspěšnější sítě *cnn*, *spp* a *lstm* na testovací sadě. Výstupem je vyhodnocení sítí v logovacím souboru s názvem *final_eval.log*.

Skript pro QbS a QbE dotaz

Skript *example_query.py* provede jeden vzorový QbS a QbE dotaz. Vrácené odpovědi jsou uloženy jako obrázky s názvem:

`<typ dotazu>_<hledaný řetězec>_<pořadí>_<řetězec odpovědi>.png`

V případě QbE je uložen i hledaný vzor jako:

`QbE_<hledaný řetězec>.png`