Přírodovědecká fakulta Jihočeské univerzity

Bakalářská práce:

Systém pro rozpoznávání tištěného písma

Vypracoval: Ondřej Hanzlík

Vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

České Budějovice 2011

Bibliografické údaje

Hanzlík Ondřej, 2011: Systém pro rozpoznávání tištěného písma.

[Character recognition system. Bc. Thesis, in Czech.] – 30 p. (počet stran), Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Abstract:

The thesis proposes a system for recognition of printed text (OCR), which uses neural network for recognizing letters. The neural network is implemented using the program RapidMiner. To control the neural network is using the processes created by program RapidMiner. These processes are run directly from a Java application. RapidMiner is implemented into the java application and using its libraries is started directly from java application.

Abstrakt:

Práce se zabývá návrhem systému pro rozpoznání tištěného textu (OCR), který pro rozpoznávání písmen používá neuronovou síť. Neuronová síť je implementována za pomoci programu RapidMiner. Pro řízení neuronové sítě jsou použity procesy vytvořené programem RapidMiner. Tyto procesy jsou spouštěny přímo z java aplikace. RapidMiner je implementován do java aplikace a za pomoci jeho knihoven je z java aplikace přímo spouštěn.

Keywords:

OCR, optical, character, recognition, neural network, RapidMiner

Klíčová slova:

OCR, optický, znak, rozpoznání, neuronová síť, RapidMiner

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské, a to v nezkrácené podobě, elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích, dne 26. 4. 2011

Podpis: .....

Poděkování

Děkuji Ing. Miroslavu Skrbkovi, Ph.D. za pomoc při zpracování bakalářské práce a pomoc při řešení problémů vzniklých v průběhu jejího vypracovávání.

Obsah

1 Úvod a cíle práce

1.1 Úvod do problematiky

Práce se zabývá sestavením zjednodušeného, ale kompletního systému pro rozpoznáním tištěného textu, tak aby bylo možné rozpoznávat text různé velikosti, psaný různými fonty a různým typem písma. K samotnému rozpoznávání jednotlivých znaků (písmena, čísla a další znaky, které se vyskytují v textu) se používá neuronová síť, která má schopnost naučit se definované znaky, a v závislosti na takto naučené neuronové síti rozpoznávat znaky, které jsou síti předkládány jako neznámé. Výsledky rozpoznání písmen neuronovou sítí jsou ve správném pořadí zapisovány do textového souboru a tím je dokončen výstup celého systému a převod textu z obrázku do textové podoby.

Celý systém má za úkol demonstrovat možnost využití neuronové sítě pro účely rozpoznávání textu, a sloužit jako ukázka při výuce neuronových sítí a inteligentních systémů. Hlavní důraz není tedy kladen ani tak na předzpracování dat (obrázků), ale na samotný způsob, jakým jsou rozpoznávány jednotlivé znaky. Součástí práce je i vytvoření webových stránek popisujících postup a řešení jednotlivých částí úkolu. Tyto stránky slouží také jako materiál použitelný při výuce, a jako příručka popisující fungování celého systému.

Jedná se tedy o kompletaci zjednodušeného systému OCR, na bázi neuronové sítě, jehož výstupem jsou textové soubory vzniklé z předložených obrázků (například oskenované stránky textu). Spojením všech výše zmiňovaných metod je kompletace systému dosaženo.

1.2 Cíle práce

Seznámit se se systémy, metodami a programovou podporou pro rozpoznávání tištěného písma ze skenovaných podkladů a toto téma zpracovat. Dále navrhnout a realizovat zjednodušený rozpoznávací systém na bázi neuronové sítě pro demonstrační účely ve výuce. Pro realizaci systému se snažit co nejvíce využívat existujících softwarů a knihovních komponent. Součástí práce je vytvoření uživatelská příručka ve formě WWW stránek.

2 Části systému

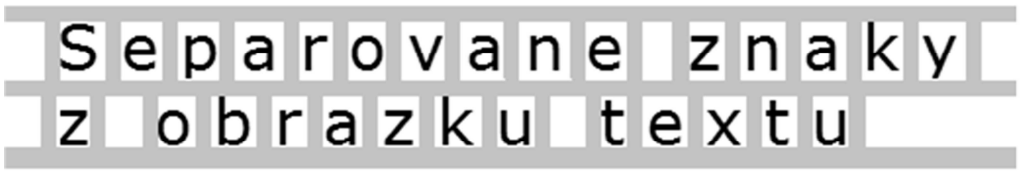
Pro kompletaci celého systému je třeba spojení následujících funkcionalit a jednotlivých částí systému.

2.1 Předzpracování

- Separování znaků:

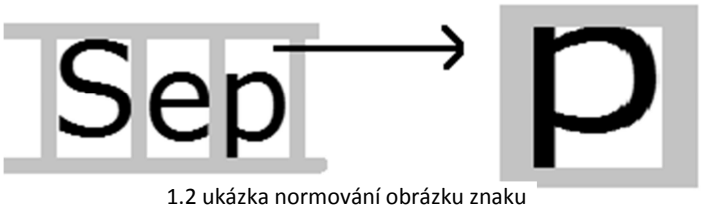
Nalezení řádků textu – aby bylo možné neuronovou sítí rozpoznat text, který je v obrázku, je třeba separovat každý obrázek znaku, který se v textu vyskytuje, a to ve správném pořadí tak, jak jdou jednotlivé znaky za sebou. Aby bylo dosaženo separace obrázků znaků, je třeba nejprve nalézt pozici, na které se vyskytují jednotlivé řádky textu. Z řádků textu je poté možno separovat obrázky znaků.

Nalezení znaků v řádku – ve chvíli kdy jsou známi pozice řádků v textu, můžeme přistoupit k separaci obrázků znaků z daných řádků. Tímto dosáhneme separace všech obrázků znaků vyskytujících se v obrázku obsahujícím text pro rozpoznání a to ve správném pořadí. Náznačení zůpůsobu vyhledávání prázdných prostor mezi řádky a mezi znaky je k vidění na obrázku 1.1.



1.1 Ukázka techniky pro separaci znaků

Normalizace separovaných obrázků znaků – z každého obrázku znaku bude třeba získat jeho atributy. Aby to bylo možné, musí dojít k normalizaci obrázků tak, aby velikost všech obrázků znaků byla stejná, a znak vykreslený v obrázku se ze všech stran dotýkal krajů obrázku (vršek znaku se musí dotýkat vrchního okraje obrázku, spodek spodního okraje obrázku, stejně tak se znak musí zprava dotýkat pravého a zleva levého okraje obrázku). Kdyby tomu tak nebylo, mohlo by docházet ke značným odchylkám při získávání atributů z obrázků, v nichž je vepsán stejný znak, a tedy je předpokládáno i získání stejných atributů. Neuronová síť by pak mohla mít značný problém tyto různé hodnoty atributů přiřadit ke správným znakům. Ukázka normalizace obrázku znaku je naznačena na obrázku 1.2.

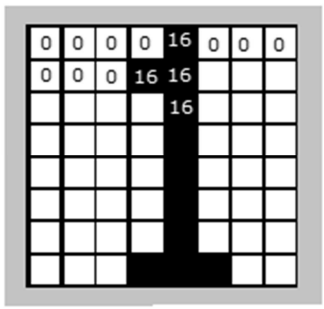


1.2 ukázka normování obrázku znaku

- Atributy obrázků

Získání atributů – poté co dojde k normalizaci obrázků, jsou obrázky připraveny k analýze a tím získání jejich atributů. Sada atributů obrázku je reprezentantem znaku, který je v obrázku nakreslen. Získání atributů je věc nutná proto, aby bylo možné neuronové síti předložit údaje reprezentující dané obrázky znaků. Na obrázku 1.3 je naznačen postup získávání atributů z obrázku znaku.

Uložení atributů – získané atributy musejí být uloženy ve formě takové, aby je bylo možné předložit neuronové síti a ta s nimi byla schopna správně pracovat. Je třeba navrhnout a dodržovat formát uložení získaných atributů obrázku.



1.3 ukázka získání atributů znaku obrázku

2.2 Neuronová síť

- Neuronová síť

Samotná neuronová síť – dalo by se říci, že neuronová síť zajišťuje převod atributů obrázku znaku na textovou podobu tohoto znaku. Neuronová síť tedy díky atributům obrázku znaku predikuje, o jaký znak se jedná.

Vytvoření trénovací množiny atributů – aby bylo možné neuronovou sítí rozpoznávat jednotlivé znaky, musí být síť nejprve naučena všem znakům, které chceme, aby uměla rozpoznat. Právě k tomuto účelu slouží trénovací množina, která obsahuje sady všech atributů obrázků znaků, které požadujeme, aby bylo možno rozpoznávat.

Vytvoření množiny atributů pro rozpoznání – aby neuronová síť mohla rozpoznávat znaky, předkládáme jí množinu atributů reprezentujících obrázky znaků. Tuto množinu předkládáme na vstup neuronové sítě, a výstupem potom jsou predikované hodnoty, určující jaký znak se nejvíce hodí ke každé sadě atributů.

Uložení výstupu neuronové sítě – hodnoty, které neuronová síť na výstupu predikuje (přiřazené znaky k daným množinám atributů obrázků znaků) je nutno uložit do čitelné podoby. Vytvoříme tedy textový soubor, v němž budou z výstupu neuronové sítě uloženy pouze predikované znaky.

2.3 Funkcionality aplikace

- **Načtení a rozpoznání obrázku –** je zapotřebí vytvořit rozhraní, schopné vybrat obrázek, z něhož chceme rozpoznat text a spustit proces rozpoznání vybraného obrázku.

- **Ovládání učení sítě –** učení neuronové sítě musí být také uživatelem ovladatelné. Za tímto účelem je třeba vytvořit rozhraní, ve kterém je možno vybrat fonty, které chceme neuronovou síť naučit a z těchto fontů vytvořit trénovací množinu. Nesmí chybět možnost spuštění učení neuronové sítě s předložením vytvořené trénovací množiny.

- **Nastavení programu –** možnost nastavení programu tak, aby bylo možné upravovat průběh procesu rozpoznávání obrázků, zvolit zda chci ukládat jednotlivé separované obrázky znaků a další možnosti nastavení systému.

3 Přehled možných řešení

Vyřešení jednotlivých částí systému je možné následujícími technikami.

3.1 Předzpracování

Separování znaků – k separování obrázků znaků z obrázku textu lze použít různých technik. Existují patenty zabývající se touto problematikou. Ukázku takového patentu můžeme vidět na stránce, viz: použitá literatura [5]. Další ukázka patentu tohoto typu je například zde, viz: použitá literatur [7].

Pro separování znaků lze využít i různých technik využívajících početních operací v závislosti na barvách v obrázku s textem, jak je zmiňováno například v dokumentu od autorů Megan Elmore a Margaret Martonosi, viz: použitá literatura [10].

Další technikou používanou při předzpracování obrázku s textem je úprava obrázku tak, aby bylo nalezení znaků jednodušší. Například odstranění čar tabulky, v níž je text (tato metoda se používá například při rozpoznání textu ze skenovaných faktur). Těmito metodami a následným nalezením znaků textu se zabývá tato stránka, viz: použitá literatura [6].

Obrázky znaků lze separovat i za pomoci techniky histogramů. Tento způsob funguje na principu nalezení bílých prostor mezi řádky a mezi znaky v řádku. Tato technika je naznačena na stránkách, viz: použitá literatura [2].

Z hlediska dostupnosti a vysokého poměru kvality vůči nenáročnosti této techniky, a zejména z důvodu, že hlavním cílem práce není předzpracování obrázku, ale demonstrace využití neuronové sítě, byl po konzultaci s vedoucím práce tento způsob vybrán jako vhodný k využití pro tento systém.

Získání atributů obrázků – ve chvíli kdy máme připraveny obrázky znaků, separované z textu v obrázku, můžeme po procesu normování obrázků znaků přistoupit k získání jejich atributů. Atributy z obrázků lze získat tak, že obrázek rozdělíme na malé čtverečky o stejné velikosti, a u každého čtverečku zjistíme počet jeho tmavých bodů, které se v něm vyskytují. Tato technika je uváděna i na více již dříve uváděných stránkách.

Konkrétní technika normalizace i získávání atributů z obrázků, která byla i vybrána k použití v systému, se nachází na stránkách: „*http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits*“. Tato technika byla zvolena jako vhodná i z toho důvodu, že data z těchto stránek jsou využívána při výuce.

3.2 Neuronová síť

Neuronová síť – jsou tři způsoby získání neuronové sítě pro zapojení do tohoto systému. První z nich je navrhnout a vytvořit neuronovou síť vlastní silou, což znamená vlastní naprogramování sítě.

Druhou možností je využití existujících knihoven pro práci s neuronovou sítí. Takových knihoven existuje celá řada. Příkladem může být java knihovna jménem *NNLib*, jejíž popis nalezneme v dokumentu, viz: použitá literatur [11], nebo knihovna v jazyce C, jménem *FANN*, s popisem na stránkách, vit: použitá literatur [9].

Třetí způsob je využití jednoho z mnoha softwarů implementujících neuronovou síť. Příkladem jsou programy jako: *JavaNSS*, *SNNS*, *NeuralLab*, *EasyNN-plus* či *RapidMiner*.

Právě poslední zmiňovaná varianta, využití existujícího softwaru, byla vybrána jako nejvhodnější řešení pro získání neuronové sítě. Konkrétní vybraný program byl RapidMiner, pomocí něhož bude práce s neuronovou sítí probíhat, a který bude v systému využit. RapidMiner umožňuje implementování do vlastní vytvořené aplikace, což je možnost, která je pro začlenění do celého systému velice vhodná. RapidMiner byl používán i při výuce neuronových sítí, a proto jeho zvolení je nejvhodnějším řešením.

3.3 Funkcionality aplikace

Správa funkcionalit systému – k propojení a správě všech jednotlivých součástí systému je třeba vytvoření programového vybavení, které zajistí pospolitost a kompletaci celého systému.

Možností v jakém programovacím jazyku navrhnout takovéto vybavení je mnoho. Například C##, C++, PHP, ASP, Java. Programovací jazyk vybraný k sestrojení aplikace byl programovací jazyk Java, a to jak už z důvodu jeho multiplatformnosti, tak kvůli faktu, že knihovny programu RapidMiner jsou také v jazyce Java. Tato volba je tedy nejrozumnějším řešením.

4 Návrh řešení

Návrh propojení a využití jednotlivých částí systému. Popis jakým způsobem je realizováno použití vybraných možností řešení.

4.1 Předzpracování

Separování znaků – algoritmus pro separaci znaků, který byl vybrán, pracuje, jak již bylo řečeno, na bázi histogramů. Díky analýze řádků sloupců pixelů, zajistí nalezení řádků textu a zjištění jejich polohy. Při nalezení pozice každého z řádků dojde k průchodu daného řádku a nalezení pozic začátku a konce každého znaku v řádku od prvního do posledního.

Aby bylo možné zajistit zjištění začátků a konců řádků, stejně tak jako zjištění začátků a konců znaků v řádcích, musí být ověřena tmavost každého pixelu v daném prostoru mezi znaky a řádky, který je tvořen po sobě jdoucími řádky či sloupci bílých pixelů. Toto ověření tmavosti pixelu a jeho klasifikace jako světlý, nebo tmavý pixel nemůže probíhat pouhým ověřením, zda je každý pixel bílý (v barevném modelu RGB to znamená červená = 255, zelená = 255, modrá = 255), nebo má jinou barvu. Klasifikace tmavosti pixelu nezávisí tedy na barvě pixelu, ale na jeho jasové složce.

Jasovou složku z barvy pixelu získáme zjištěním y-ové hodnoty z barevného modelu YUV. Vzorec pro spočítání jasové složky barevného modelu YUV za pomoci barevného modelu RGB je Y = 0.299*R + 0.587*G + 0.114*B, viz.: *„http://cs.wikipedia.org/wiki/YUV“*. Po spočítání jasové složky zbývá už jen zvolit práh, určující kdy je pixel ještě klasifikován jako světlý, a kdy jako tmavý. Rozsah jasové složky je 0 až 255, přičemž 0 je nejtmavší hodnota (černá) a 255 je nejsvětlejší hodnota (bílá).

Získání atributů obrázků – normovaný obrázek znaku má velikost 32 x 32 pixelů. Takto velký obrázek je rozdělen na malé čtverečky o velikosti 4 x 4 pixely. Jednoduchým počtem tedy zjistíme, že počet čtverečků v obrázku je 32 / 4 = 8 čtverečků na výšku a stejný počet čtverečků na šířku. Pro představu se dá říci, že přes celý obrázek je vytvořena mřížka o šedesáti čtyřech stejně velkých čtvercích. Poté zjišťujeme počet tmavých bodů v každém ze čtverců. Velikost čtverce je 4 x 4 pixely, což znamená, že hodnoty, kterých můžeme pro každý čtverec dosáhnout (počet černých bodů v daném čtverci) jsou 0 až 16 (4 x 4 = 16).

Tímto dosáhneme získání šedesáti čtyř číselných atributů obrázku (8 x 8 = 64). Těchto 64 atributů tvoří základ sady atributů každého obrázku, a jsou jednou z hlavních částí množiny dat předkládaných neuronové síti. Druhou a poslední část množiny tvoří údaj o tom, zda znak v obrázku je nám známý, nebo není, respektive jestli je známo, jakému znaku vytvořená sada atributů náleží. Tato informace je šedesátý pátý atribut obrázku znaku, a jeho hodnoty jsou následující. Pro znak, který známe a definujeme, o jaký znak se jedná, nabyde šedesátý pátý atribut hodnoty znaku samotného. Pro příklad: písmeno „A“ je vykresleno v obrázku, z něhož získáváme hodnoty atributů. Šedesátý pátý atribut u takového obrázku nabyde hodnoty „A“. V opačném případě, když neznáme znak vykreslený

v obrázku, jehož atributy získáváme, nabyde tento atribut hodnoty, která značí, že jde o znak neznámý.

Tímto způsobem tedy získáme 64 + 1 = 65 atributů, které jsou reprezentanty obrázku znaku, který jsme analyzovali.

4.2 Neuronová síť

Neuronová síť – neuronová síť je samotným jádrem programu. Její funkce je naprosto jasná – predikovat, jaké znaky náleží atributům, jež jí byly předloženy v množině pro rozpoznání. Aby ale neuronová síť uměla jakýkoli znak rozpoznat, musí být nejprve znakům naučena. Jedná se o takzvané učení s učitelem, kdy je síti nejprve předložena sada atributů s definovaným atributem nesoucím informaci o tom, jaký znak náleží sadě předložených atributů získaných z obrázku znaku. Tato množina sad atributů s definovaným znakem první části sady náležejícím, se jmenuje učící množina.

Je zřejmé, že vytvoření učící množiny i trénovací množiny, znamená uložení atributů obrázku v takové formě, aby je bylo možno předložit neuronové síti. V případě rozpoznávání znaků musí atribut nesoucí hodnotu, zda jde o známý znak, či znak určený k rozpoznání, být síti předložen a označen jako atribut, který chceme predikovat. V programu RapidMiner dostane tedy tento atribut označení „label“. Neuronová síť ho pak predikuje na základě k němu náležících šedesáti čtyřech atributech. Při učení neuronové sítě je tento atribut označen také jako „label“, ale neuronová síť ho nepredikuje, nýbrž si ho přiřadí k předloženým šedesáti čtyřem atributům. Tato popsaná metoda je právě ona metoda učení neuronové sítě s učitelem.

Model naučené neuronové sítě se ukládá, aby mohla být neuronová síť později načtena a použita při procesu rozpoznávání písmen (predikce atributu označujícího jakému znaku náleží sada šedesáti čtyř atributů).

Poslední věc, která si z hlediska neuronové sítě žádá vyřešení je ukládání výsledků výstupu neuronové sítě. Konkrétně uložení výsledků v čitelné podobě (do textového souboru). To znamená, že z celého výstupu se ukládána pouze hodnota predikovaného atributu s označením „label“. To se řeší za použití RapidMiner knihoven a vytvořením vybavení pro uložení požadovaných hodnot.

Celé ovládání neuronové sítě z vlastní aplikace probíhá díky implementovaným knihovnám programu RapidMiner. Aby bylo možné proces rozpoznání a učení spustit, je třeba nejdříve vytvořit proces, spustitelný programem RapidMiner. Vytvoření procesů je možné přímo v grafickém rozhraní RapidMineru, což je praktický a přehledný způsob. Potřebné procesy, které je nutno vytvořit jsou dva. Jeden, který umožňuje načtení trénovací množiny a její předložení neuronové síti pro učení, a poté následné uložení naučeného modelu neuronové sítě. Druhý, který umožňuje načtení modelu neuronové sítě a rozpoznání předložené množiny pro rozpoznání.

Tímto máme připravenou další část systému pro implementaci do vlastní aplikace a zbývá vyřešit spuštění procesů v požadovanou chvíli, s předložením správné trénovací množiny, či množiny k rozpoznání. Poslední věcí je samotné

uložení výsledků predikce neuronové sítě, které je řešeno automaticky aplikací poté, kdy je dokončení proces rozpoznání.

4.3 Funkcionality aplikace

Vlastní aplikace propojující a spravující všechny jednotlivé funkcionality systému je, jak již bylo řečeno, napsána v jazyce Java. Pro dosažení celistvosti systému musí být vytvořeno následující vybavení:

- **Načtení a rozpoznání obrázku –** vytvoření prostředí, ve kterém je možno zvolit cestu k obrázku, který chceme rozpoznávat, nebo cestu ke složce, ve které jsou uloženy obrázky určené k rozpoznání. V tomto prostředí nechybí možnost spuštění procesu rozpoznání, což sebou nese postupné spuštění následujících funkcionalit: separace znaků obrázků, normalizace obrázků znaků, získání atributů z normalizovaných obrázků, uložení sady atributů do množiny pro rozpoznání, předložení množiny pro rozpoznání neuronové síti a spuštění RapidMiner procesu pro rozpoznání, získání výstupu neuronové sítě a uložení predikovaných znaků do textového souboru.

Z tohoto prostředí je možno spustit také další prostředí programu, jako učení sítě nebo nastavení vlastností programu.

- **Ovládání učení sítě –** prostředí nabízí možnost zvolit font, či fonty, které chceme, aby se neuronová síť naučila a byla v závislosti na tom schopna rozpoznávat znaky. Tato možnost učení sítě s sebou nese spuštění následujících funkcionalit: výběr fontů k naučení, vykreslení všech učených znaků do obrázku, oříznutí obrázku a jeho normalizace tak, aby byl obrázek v normované velikosti (32 x 32 pixelů) stejně jako znaky, které se separují při rozpoznávání textu, dále získání sady atributů obrázku znaku, jejíž hodnota definovaného atributu je rovna samotnému znaku, uložení sady atributů do trénovací množiny a spuštění RapidMiner procesu pro učení neuronové sítě s předložením vytvořené trénovací množiny

- **Nastavení programu –** v nastavení aplikace je možno zvolit, zda chceme při procesu rozpoznání ukládat separované obrázky znaků, zda chceme k rozpoznání použít výchozí neuronové sítě a zda chceme při procesu učení neuronovou síť naučit i znaky obsahující čárky a háčky. Další možnost v nastavení je volba prahu světlosti, což je procentuální nastavení tolerance tmavosti pixelu. Jinými slovy nám toto nastavení dává možnost určit práh označující, od jaké světlosti barvy pixelu bude pixel označen za tmavý. Poslední možností nastavení je zvolení procentuální velikosti mezery vůči výšce řádku, jehož znaky separujeme.

5 Implementace zvolených nástrojů

Popis samotné implementace všech nástrojů, které byly výše popsány a vybrány k řešení problematiky

5.1 Předzpracování

- Separování znaků

Pro separaci znaků z textu, vykresleném v obrázku a pro zajištění všech potřebných funkcí souvisejících se separací, jsem vytvořil třídu `CharCutting`. Popis třídy je následující.

Instanci třídy vytvoříme zavoláním konstruktoru s parametrem typu `String`. Tento parametr udává cestu k obrázku, z něhož chceme jednotlivé znaky separovat a rozpoznat, nebo cestu ke složce takovýchto obrázků. V případě, že vybereme cestu ke složce, je možné že složka obsahuje ještě další složky se soubory, v nichž jsou obrázky uloženy. Aby byly možné všechny přístupy k souborům obrázků, vytvořil jsem metodu pro průchod stromové struktury zadané kořenové složky. Při průchodu stromu složky probíhá test, zda jsou všechny procházené objekty složka nebo soubor. U složek dochází k průchodu jejich obsahu a jeho opětovného testování. Jedná se o rekurzivní funkci. V případě, že byl kdykoli při průchodu stromovou strukturou složek nalezen soubor, dojde k jeho přidání do kolekce všech souborů vyskytujících se v kořenové složce nebo v jakékoli z jejích podsložek. Tím vytvoříme kolekci všech obrázků, ze kterých budeme chtít rozpoznávat text. V případě, že cesta předaná parametrem vede přímo k jednomu konkrétnímu obrázku, je do kolekce souborů obrázků přidán tento jediný soubor. Jestliže je zadaná cesta špatná, znamená to, že soubor či složka neexistuje nebo se nejedná o obrázek. V takovém případě je uživatel upozorněn výpisem a je umožněno zadat cestu znova.

Ve chvíli kdy je připravena kolekce souborů obrázků, procházíme ji, a z každého souboru vytvoříme pomocí Java knihovny `ImageIO` obrázek. Po vytvoření každého obrázku probíhá hned jeho upravení a separování znaků z textu vyskytujícím se v obrázku. Upravení obrázku znamená přikreslení jeho okrajů po všech stranách. Okraje mají bílou barvu a jsou po všech stranách obrázku. Toto je nutné kvůli správné funkčnosti algoritmu, který zjišťuje začátky a konce řádků a začátky a konce znaků. Popis tohoto algoritmu následuje níže, ovšem musím již zde upozornit na fakt, že bez této úpravy okrajů obrázku by mohlo docházet ke špatné klasifikaci začátku prvního řádku textu a klasifikaci prvního znaku v řádku. Stejně tak by mohlo docházet k chybnému nalezení posledního řádku textu a posledního znaku řádku. To by se stávalo v případě, kdy by se text vykreslený na obrázku dotýkal okrajů obrázku. Tím, že přikreslíme okolo celého obrázku okraje o šířce jeden pixel, dosáhneme toho, aby k chybě nedocházelo.

Ve chvíli kdy je obrázek připraven, můžeme přistoupit k procesu separace jeho znaků, který se skládá z částí:

- Nalezení řádků textu

Pro nalezení začátků a konců řádků textu v obrázku jsem vytvořil metodu implementující algoritmus zajišťující zjištění pozic řádků textu za pomoci histogramů. Začátek řádku hledáme tím způsobem, že procházíme všechny pixely v každé řádce obrázku (řádkou obrázku se myslí řádek pixelů o výšce jeden pixel a šířce rovnající se šířce obrázku. Projdou se tedy všechny pixely zleva doprava). U každého pixelu testujeme, zda je pixel světlý nebo tmavý. V případě, že počet světlých pixelů v řádku je roven celkovému počtu pixelů daného řádku, je řádek vyhodnocen jako světlý. Jestliže najdeme světlý řádek, jedná se o prázdný prostor, což značí, že neprocházíme řádek pixelů protínající text. Takto procházíme postupně další řádky obrázku (vždy pokračujeme bezprostředně následujícím řádkem). Ve chvíli kdy v procházeném řádku najdeme tmavé pixely, může se jednat o začátek řádku textu. Abychom byli schopni určit, kdy byl nalezen začátek řádku textu, a kdy konec řádku textu, využíváme k tomu dvě proměnné. Jednu, která nese informaci o tom, zda je řádek, který jsme právě prošli, celý světlý či nikoli. Druhou, která nese informaci o tom, jestli byl předchozí procházený řádek světlý nebo ne. Obě proměnné jsou typu `boolean` a inicializovány s hodnotou „false“. Ve chvíli nalezení bílého řádku se hodnota proměnné nesoucí informaci o aktuálním procházeném řádku změní na „true“. Při začátku průchodu dalšího řádku obrázku nabyde hodnota proměnné nesoucí informaci o světlosti předchozího procházeného řádku stejnou hodnotu, jako má proměnná nesoucí informaci o světlosti předchozího řádku. Za pomoci těchto dvou proměnných jsme schopni klasifikovat výskyt začátků a konců řádků.

Na konci průchodu každého řádku ověříme, zda jsme právě nenalezli začátek, nebo konec řádku textu. O začátek řádku se jedná ve chvíli, kdy proměnná s informací o světlosti předchozího řádku má hodnotu „true“, a proměnná s informací o světlosti aktuálního řádku má hodnotu „false“. Jedná se tedy o nalezení prvního nesvětlého řádku po průchodu řádků světlého prostoru. Konec řádku textu je nalezen v případě, že proměnná nesoucí informaci o světlosti předchozího procházeného řádku obrázku má hodnotu „false“ a současně proměnná s informací o světlosti aktuálně projitého řádku má hodnotu „true“.

Ve chvíli kdy je nalezen začátek nebo konec řádku, je třeba zaznamenat pozici začátku nebo konce řádku. V souřadnicovém systému, pomocí něhož se přistupuje k pixelům obrázku, se jedná o hodnotu y-ové souřadnice aktuálně projitého řádku obrázku. Hodnotu pozice začátku řádku textu uchováváme v proměnné a ve chvíli kdy nalezneme i konec řádku, uložíme pozici konce řádku textu také do proměnné. Tímto máme získány informace potřebné pro separaci znaků z právě nalezeného řádku textu. Přistoupíme tedy k separaci jednotlivých znaků obrázků.

- Nalezení znaků v řádku

Obrázky znaků separujeme z řádku textu stejnou technikou, jakou jsme získávali jednotlivé řádky textu. K určení začátků a konců znaků v řádku využíváme opět dvě proměnné typu `boolean`, za pomocí nichž poznáme, zda jsme nalezli začátky nebo konce znaků. Rozdíl je pouze ten, že neprocházíme celý obrázek, ale pouze prostor mezi začátkem a koncem nalezeného řádku textu. Procházení neprobíhá horizontálně (jako tomu bylo u hledání začátků a konců řádků), ale vertikálně. Prochází se tedy každý

sloupeček řádku (sloupečkem řádku se rozumí šloupeček pixelů o šířce jeden pixel a výšce rovnající se výšce řádku textu, ze kterého separujeme znaky) od začátku řádku textu (od levého okraje celého obrázku) po konec řádku (po pravý okraj celého obrázku). U každého pixelu sloupečku řádku proběhne test, zda se jedná o pixel světlý nebo pixel tmavý. Když se počet světlých pixelů procházeného sloupečku (sloupečku řádku textu) rovná celkovému počtu pixelů daného sloupečku, je sloupeček vyhodnocen jako světlý prostor.

Stejně jako hledání začátků a konců řádků textu, probíhá i hledání začátků a konců znaků. Za pomoci proměnných nesoucích informace o světlosti aktuálně prošlého sloupečku a předchozího prošlého sloupečku určíme, zda byl nalezen začátek znaku. V případě že ano, uložíme si tuto pozici do proměnné (v souřadnicovém systému je to hodnota x-ové souřadnice obrázku). Procházení pokračuje dál až do chvíle nalezení konce znaku. Opět si pozici konce znaku uložíme do proměnné. Kromě uložení pozice konce znaku je třeba současně zaznamenat i informaci o tom, že byl nalezen celý znak. Využijeme tedy proměnné, jejíž datový typ je opět `boolean` a její inicializace proběhla s hodnotou „false“. Tato proměnná nabývá hodnoty „false“ při každém začátku průchodu nového sloupečku řádku. Hodnotu „true“ jí nastavíme pouze v případě nalezení konce znaku. Když je tedy hodnota této proměnné „true“ a víme tedy, že jsme našli celý znak, je třeba ho uložit. Je připravena kolekce obrázku, do níž se všechny nalezené obrázky znaků budou ukládat.

Aby mohl být znak do kolekce uložen, musí nejprve být z procházeného obrázku překreslen do nového obrázku. Za tímto účelem jsem vytvořil metodu, se třemi parametry. První i druhý parametr je dvousložkové pole typu `int`. V prvním parametru jsou uchovány souřadnice začátku znaku v obrázku, tedy x-ová a y-ová souřadnice levého horního rohu znaku. Druhý parametr v sobě uchovává pozici konce nalezeného znaku, to je x-ovou a y-ovou pozici pravého dolního rohu znaku. Y-ové hodnoty se u všech znaků daného řádku rovnají (jsou totiž shodné se začátkem a koncem řádku). X-ové souřadnice mohou být u každého nalezeného znaku různé (v závislosti na šířce znaků). Třetí parametr je typu `String` a má tu informační hodnotu, že nám definuje, o jaký znak se u nalezeného obrázku znaku jedná. Při separace znaků z obrázku textu určenému k rozpoznání nevíme, jaký znak byl právě nalezen. Je však speciální případ, kdy víme, jaký znak byl nalezen a je vykreslován. Tento případ je vykreslování mezery. Popis vykreslení a zjištění mezery nechme ale na později. Nastavíme tedy hodnotu třetího parametru při volání funkce překreslení obrázku znaku z obrázku s textem pro rozpoznání na „null“.

Samotné překreslení obrázku znaku proběhne tak, že vytvoříme nový prázdný obrázek o velikosti nalezeného obrázku znaku (výška obrázku je y-ová souřadnice konce znaku od které odečteme y-ovou souřadnici začátku znaku, šířka obrázku je x-ová souřadnice konce znaku od které odečteme x-ovou souřadnici konce znaku). Poté procházíme všechny pixely v obrázku s textem pro rozpoznání, vyskytující se mezi souřadnicemi začátku a konce znaku. Pixely procházíme po řádcích a zjišťujeme jejich barvu. Stejnou barvu jako mají pixely v obrázku s textem pro rozpoznání, nastavíme pixelům v připraveném prázdném obrázku znaku. Tím dojde k překreslení nalezeného znaku do nového obrázku.

Zbývá ještě zjistit, kdy se v textu vyskytují mezery. Jelikož mezera je prázdný prostor, nelze ji zjišťovat také pomocí jejího prvního a posledního tmavého sloupce pixelů. Pro zjištění mezery jsem vytvořil metodu, která se zavolá vždy po separování každého jednoho znaku. Metoda je zavolána s parametry určujícími souřadnice vršku

a spodku řádku textu a souřadnicemi konce posledního vyseparovaného znaku z daného řádku. Metoda stanoví, jestli za znakem následuje mezera nebo ne a podle toho vrátí hodnotu „true“ nebo „false“. Stanovené existence mezery probíhá tím způsobem, že od posledního separovaného znaku proběhne kontrola tmavosti určitého počtu sloupců řádku, které následují bezprostředně za nalezeným znakem. Počet procházených sloupců za znakem je závislý na výšce řádku, ve kterém je mezera hledána.

V programu lze nastavit procentuální hodnotu z výšky řádku a tím zvolit, kolik světlých sloupců musí za znakem následovat, aby bylo vyhodnoceno, že je za ním mezera. Prakticky to znamená, že je-li výška řádku 10 pixelů a nastavíme, že pro pozitivní klasifikaci nálezu mezery je třeba nalézt prázdný prostor za znakem o velikosti 80% výšky řádku, bude vyhodnocena existence mezery v tom případě, že za separovaným znakem bude nelezeno 8 nebo více světlých sloupců (světlý prostor o šířce 8 pixelů). V případě že podmínka bude splněna a potřebný počet světlých sloupců bude nalezen, metoda vrátí hodnotu „true“. V takovém případě přijde na řadu opět metoda pro vykreslení obrázku znaku. Její parametry budou ale jiné, než při překreslování znaku z obrázku. První dva parametry, určující začátek a konec znaku v obrázku, budou mít hodnotu „null“. Třetí parametr určující jaký znak vykreslujeme, který je typu `String`, má hodnotu „mezera“. Takto se vykreslí prázdný obrázek velikosti 32 x 32 pixelů a každý jeho pixel je bílý. Obrázek je pak uložen do kolekce znaků.

Stejným způsobem se pokračuje dál až do konce textu. Tímto je tedy zajištěno procházení řádků textu v obrázku od prvního do posledního a v každém řádku procházení jednotlivých znaků od prvního do posledního, tak jak následují za sebou. Toto správné pořadí průchodu a překreslení znaků je nutné pro možnost rozpoznávání znaků textu ve správném pořadí.

Aby bylo možné získat z obrázku znaku atributy, musí ještě před přidáním obrázku znaku do kolekce všech obrázků znaků dojít k jeho normalizování.

- Normalizace separovaných obrázků znaků

Normalizace obrázku se skládá ze dvou částí. První je očištění obrázku od světlých okrajových prostor. Druhá část je transformace obrázku na velikost 32 x 32 pixelů. Očištění obrázku je věc nutná k tomu, aby bylo zaručeno, že každý znak vykreslený v obrázku, bude ležet na stejném místě. Separované obrázky znaků v závislosti na výšce řádku, z něhož byly separovány, můžou mít různě velký prostor světlých pixelů vyskytujících se nad a pod vykresleným znakem. To je následek způsobu separace obrázku znaku z řádku.

Jelikož normalizace znaků je využíváno i při procesu vytváření trénovací množiny pro neuronovou síť, vytvořil jsem statickou třídu `CharImageNormating`, obsahující metody pro očištění obrázku, transformaci obrázku se změnou velikosti a metodu ověřující tmavost pixelu.

Metoda pro očištění obrázku se volá s parametrem typu `BufferedImage` a její návratová hodnota je také typu `BufferedImage`. Obrázek předaný parametrem je procházen po řádcích od shora dolů a u každého řádku je vyhodnocováno, zda je řádek celý světlý či nikoli. Řádek je vyhodnocen jako světlý opět ve chvíli, kdy počet světlých pixelů řádku, se rovná celkovému počtu pixelů v řádku. Při nalezení světlého řádku se nic neděje. Naopak při nalezení prvního tmavého řádku je třeba zapsat polohu tohoto

řádku. Zapíšeme tedy do dvouprvkového pole, na první pozici, hodnotu y-ové souřadnice, na níž se první tmavý řádek vyskytuje. Pokračujeme dále v procházení řádků a při každém nalezeném tmavém řádku přepisujeme hodnotu druhého prvku pole. Tím dosáhneme toho, že v druhém prvku pole bude uložena pozice posledního procházeného tmavého řádku obrázku. Pole pak obsahuje y-ové souřadnice toho, kde začíná vykreslený znak, a kde končí. Stejně projdeme obrázek po sloupcích zleva doprava a při nalezení prvního tmavého sloupce (sloupec obsahující alespoň jeden tmavý pixel) uložíme jeho pozici do první složky pole, které nese informace o pozici začátku a konce tmavých sloupců obrázku (x-ové souřadnice). S každým dalším nalezeným tmavým sloupcem, měníme hodnotu druhé složky pole na hodnotu x-ové souřadnice nalezeného tmavého sloupce. Tímto jsme dosáhli vytvoření dvou polí, přičemž jedno pole nese informace o hodnotách y-ových souřadnic prvního a posledního tmavého řádku, a druhé nese informaci o hodnotách x-ových souřadnic prvního a posledního tmavého sloupce.

Vykreslíme tedy nový obrázek, jehož výška je rovna rozdílu hodnoty koncové a počáteční souřadnice z pole nesoucí informaci o poloze začátku a konce tmavých řádků a šířka obrázku je rovna rozdílu hodnoty koncové počáteční souřadnice z pole nesoucí informaci o poloze začátku a konce tmavých sloupců. Pixely takto připraveného prázdného obrázku vykreslíme tak, aby se shodovaly s pixely neočištěného obrázku mezi souřadnicemi začátků a konců tmavých sloupců a řádků na ose x a y. Metoda pro očištění obrázku vrátí právě tento očištěný obrázek. S ním pak zbývá udělat pouze poslední část normalizace.

O poslední část normalizace (transformace a změna velikosti) se stará metoda, jejímž parametrem jsou obrázek typu `BufferedImage`, který představuje obrázek, jež chceme transformovat a číselná hodnota typu `int` představující výšku řádku, ze kterého byl znak vyseparován. Metoda vrací obrázek transformovaný na velikost 32 x 32 pixelů. Transformace může probíhat více způsoby a to v závislosti na výšce a šířce obrázku vůči řádku, ze kterého byl separován. Jeden průběh transformace je takový, že se připraví prázdný obrázek typu `BufferedImage` o šířce i výšce 32 pixelů a do tohoto obrázku je metodou `drawImege` z knihovny `Graphics2D` vykreslen obrázek určený k transformaci. Ještě před samotným vykreslením proběhne nastavení filtru pro vykreslení grafických komponent metodou `setRenderingHint`, která má dva parametry. Prvním parametrem je `RenderingHints.KEY_RENDERING` a druhým parametrem je `RenderingHints.VALUE_RENDER_QUALITY`. Tato metoda je také ze třídy `Graphics2D`. Nastavením dosáhneme lepší kvality výstupního transformovaného obrázku. Tím by byl celý obrázek znaku připraven pro přidání do kolekce obrázků znaků, kdyby neexistovali výjimky.

Písmena, která jsou úzká jako „l“ nebo „i“, nemůžou být tímto způsobem transformovány z důvodu toho, že by došlo k jejich roztažení přes celý obrázek a vypadaly by jako černý čtverec. Proto před transformací obrázku proběhne ověření, zda je obrázek 3,5 krát vyšší než je široký. Takovýto obrázek je považován za úzký a jsou mu zprava a zleva dokresleny sloupce bílých pixelů. Šířka doplněných sloupců zprava a zleva je rovna poloviční šířce ještě netransformovaného obrázku. Celkově tedy obrázek svoji šířku zvýší na dvojnásobek. Takto transformovaný obrázek pak už nebude vypadat jako černý čtverec. Ještě pořád by se ale vyskytoval problém se znaky jako „.“ Nebo „,“. Proto je zde ještě ověření, zda není obrázek znaku pětkrát nižší, než je výška řádku, ze kterého byl separován. V takovém případě mu budou dokresleny okraje (bílé řádky a sloupce pixelů) po všech stranách (to znamená zprava i zleva, shora i zdola). Šířka takto dokreslených prostor je zprava a leva opět polovina původní šířky obrázku, shora a zdola je to polovina výšky obrázku. Získáme tím obrázek se znakem

napsaným vprostřed a transformace takového obrázku nebude mít na znak zásadní deformační následky.

Takto transformovaný a očištěný obrázek znaku je uložen do kolekce všech obrázků znaků separovaných z obrázku s textem pro rozpoznání.

Přehled a příklady ovládání třídy `CuttingChar:`

- Vytvoření instance třídy: **`new CuttingChar(String cesta)`**
- Šířka aktuálně procházeného obrázku: **`getSirka()`**
- Výška aktuálně procházeného obrázku: **`getVyska()`**
- Kolekce všech souborů ze stromu souborů od kořenového adresáře / kořenového souboru: **`getVsechnySoubory()`**
- Kolekce všech separovaných normovaných obrázků znaků: **`getList_znaku()`**
- Kolekce všech stránek s textem k rozpoznání (všechny obrázky s textem pro rozpoznání): **`getKolekceStranek()`**
- Uložení separovaných obrázků znaků: **`uloz_obrazky_znaku(ArrayList <BufferedImage> obrazky)`**

- Atributy obrázků

Získání atributů – pro získání atributů z obrázků znaků jsem vytvořil třídu `CharAttributes`. Konstruktorů této třídy je více a jsou popsány v dokumentaci, která je součástí přílohy [A]. Hlavní a systémem využívané konstruktory jsou dva. Jeden se používá při vytváření trénovací množiny neuronové sítě, druhý se používá při vytváření množiny k rozpoznání. Získání první části sady atributů probíhá v obou případech stejným způsobem. Jak bylo řečeno, sada atributů má dvě části. První část se skládá z šedesáti čtyř číselných hodnot v rozmezí 0 až 16. Druhá část je jeden atribut udávající o jaký znak se jedná (jakému znaku sada šedesáti čtyř atributů náleží). V případě, že neznáme znak, kterému atributy náleží, je označeno, že jde o neznámý znak.

První část sady atributů získáme z obrázku znaku tak, že projdeme všechny jeho pixely ve čtvercích velikosti 4 x 4 pixely, na které si obrázek rozdělíme a spočteme počet tmavých pixelů v každém čtverci. Jelikož čtverců v obrázku je 64, budeme mít i 64 atributů, které je třeba uchovávat. Před analýzou pixelů obrázku si tedy připravíme dvourozměrné pole o velikosti 8 x 8. Do tohoto pole budou ukládány výsledky spočtení tmavých pixelů ve čtvercích obrázku. Přístup k výsledkům pak bude shodný s přístupem ke čtvercům v obrázku, k nimž by se dalo přistoupit pomocí souřadnicového systému. Například počet tmavých pixelů pátého čtverce z prvního řádku čtverců bude v poli uložen na první pozici prvního rozměru pole a na páté pozici druhého rozměru pole. Pro přehlednost ideální. Když máme připraveno pole, do kterého se

budou ukládat počty tmavých pixelů ve čtvercích, je třeba mít připravenou i proměnnou, pomocí níž budeme počet tmavých pixelů ve čtverci počítat. Tato proměnná je inicializována s hodnotou „0“ a je typu `int`.

Máme připraveny proměnné, stačí tedy začít ukládat atributy obrázku. Procházíme dvourozměrné pole, do kterého budeme ukládat počty tmavých pixelů každého čtverce. Průchod začne u prvního prvku prvního rozměru pole a pokračuje průchodem všech jeho prvků druhého rozměru pole (průběh průchodu je tedy: 0 x 1, 0 x 2, 0 x 3, 0 x 4, …). Při průchodu každého prvku pole počítáme počet tmavých pixelů v daném čtverci obrázku. Abychom počítaly pixely správného čtverce obrázku, je třeba zadat souřadnice začátku každého čtverce.

Pro spočítání tmavých pixelů čtverce se volá metoda, jejímž parametrem jsou dvě hodnoty typu `int`. První udává x-ovou souřadnici začátku čtverce, u kterého počítáme počet tmavých pixelů, druhá udává y-ovou souřadnici začátku tohoto čtverce. Metoda projde všechny pixely ve čtverci od bodu začátku do bodu, který se nachází o čtyři pixely dál než x-ová souřadnice začátku čtverce a o čtyři pixely pod y-ovou souřadnicí začátku čtverce. Při průchodu každého dalšího prvku v druhém rozměru pole, stoupá hodnota y-ové souřadnice začátku čtverce o čtyři. Při průchodu každého dalšího prvku z prvního rozměru pole, stoupá hodnota x-ové souřadnice začátku čtverce o čtyři. Tímto způsobem je projit celý obrázek znaku a počet tmavých pixelů v každém jeho čtverci velikosti 4 x 4 pixely je uložen v dvourozměrném poli.

Druhou část sady atributů, což je jeden atribut, definující jakému znaku náleží sada šedesáti čtyř atributů, získáme v závislosti na tom, jaký konstruktor třídy `CharAttributes` použijeme. V případě, že předáme konstruktoru parametrem pouze obrázky znaků bez definování toho, jaký znak se v obrázku vyskytuje, nebude se tento atribut u první části sady atributů vůbec vyskytovat. K jeho vytvoření dojde až při ukládání sady atributů. Příkladem může být vytvoření instance třídy `CharAttributes` s parametrem typu `ArrayList <BufferedImage>.` Naopak při použití konstruktoru třídy `CharAttributes`, jehož parametr s sebou nese i definici toho, jaký znak je v obrázku znaku vykreslen, je tento atribut spjat s první částí sady atributů. Příklad je použití konstruktoru s parametrem typu `HashMap<BufferedImage, String>,` kde každý obrázek znaku uložený v hashmapě má přiřazenu hodnotu, udávající jaký znak je v obrázku znaku nakreslen. I v tomto případě se definující atribut k sadě šedesáti čtyř atributů připisuje až při ukládání celé sady atributů ale díky hashmapě je definující znak spjat s šedesáti čtyřmi atributy obrázku znaku.

Když máme uchovány všechny hodnoty potřebné k uložení celé sady atributů, přistupme k samotnému procesu uložení atributů.

Uložení atributů – uložení atributů probíhá tak, že je vytvořena proměnná typu `String`, do které je zapsána nejprve druhá část sady atributů (definující atribut je pro přehlednost zapisován jako první) a poté je zapsána první část sady atributů (šedesát čtyři atributů). Každý atribut je oddělen čárkou. V případě, že definující atribut neexistuje (což nastane při použití konstruktoru třídy `CharAttributes`, jehož parametr s sebou nese pouze obrázky bez definice, jaký znak je v nich vykreslen), je jako hodnota definujícího znaku zapsáno „unknown“.

Touto metodou lze uložit sadu atributů všech obrázků znaků do textového souboru a tím vytvořit trénovací množinu pro neuronovou síť i množinu pro rozpoznání.

Přehled a příklady ovládání třídy `CharAttributes`:

- Vytvoření instance třídy (bez definujícího atributu):
  `CharAttributes(ArrayList<BufferedImage> obrazkyZnaku)`

- Vytvoření instance třídy (s definujícím atributem):
  `CharAttributes(HashMap<BufferedImage, String> znaky)`

- Kolekce první části všech polí s atributy obrázků:
  `getSeznamPoliPrevodu()`

- Zápis kolekce polí atributů do souboru:
  `zapisTxt(ArrayList<int[][]> kolekceAtributu)`

5.2 Neuronová síť

- Neuronová síť

Samotná neuronová síť je využívána pomocí programu RapidMiner a implementována pomocí vytvoření procesu, který se z vlastní aplikace spustí za pomoci implementovaných knihoven RapidMineru. K java aplikaci se tedy připojí RapidMiner knihovny, které jsou umístěny v domovském adresáři programu RapidMiner ve složce „lib“. Tímto máme otevřen prostor k práci s RapidMinerem a volání jeho funkcí přímo z naší aplikace.

Vytvoření trénovací množiny atributů – neuronovou síť je třeba naučit základním znakům vyskytujícím se v textu. Obsah trénovací množiny je tedy tvořen sadou atributů znaků, které chceme, aby neuronová síť dokázala rozpoznávat. Konkrétně jsou to znaky z ASCII tabulky od třicátého třetího do stodvacátého druhého znaku. Všechny tyto znaky je třeba vykreslit na připravený bílý obrázek, provést jejich normování, získat jejich atributy a přitom definovat jaký znak je v obrázku vykreslen.

Vytvořil jsem tedy třídu `UceniSite` a v ní metoda pro vykreslení znaku do připraveného bílého obrázku (výška i šířka obrázku je nastavena na 100 pixelů a všechny jeho pixely mají nastavenou barvu na bílou). Vykreslení znaku proběhne pomocí metody `drawString` ze třídy `Graphics2D`. Před vykreslením znaku do obrázku je barva vykreslení nastavena na černou a je použito filtru antialiasing a vysoké kvality renderování. Výstup obrázku s použitím těchto filtrů se více blíží znakům ze skenovaného textu. Takto připravený vykreslený obrázek znaku projde procesem normování (očištění a transformace za pomoci třídy `CharImageNormating`). Poté jsou z každého obrázku znaku získány atributy a uloženy do hashmapy, kde je ke každému obrázku přiřazena hodnota typu `String`, definující jaký znak je v obrázku znaku vykreslen.

Uložením všech sad atributů obrázků znaků z hashmapy získáme trénovací množinu pro neuronovou síť. K tomu slouží metoda zapisující sady atributů do textového souboru pro pozdější načtení a využití neuronovou sítí.

Konstruktor třídy `UceniSite` má tři parametry. První parametr je typu `String` a jeho hodnota představuje jméno fontu, jaký bude použit při vykreslování znaků do obrázků. To je nutné proto, aby bylo možné rozpoznávat znaky psané různými fonty. Neuronovou síť je pak tedy možno naučit všechny fonty jaké si vybereme. Druhý parametr je typu `boolean` a udává, zda se mají ukládat obrázky učených znaků. Třetí parametr je také typu `boolean` a udává, zda chceme neuronovou síť učit i znakům obsahujícím čárky a háčky. V případě, že má tento parametr hodnotu „true“, přibydou k vykreslovaným znakům z ascii tabulky, ještě ostatní české znaky, které obsahují čárky a háčky.

Přehled a příklady ovládání třídy `UceniSite`:

- Vytvoření instance třídy:
  `UceniSite(String jmenoFontu, boolean ulozImg, boolean carkyHacky)`

- Uložit kolekci definovaných obrázků znaků do souboru:
  `ulozitDefinovaneObrazky()`

- Zápis kolekce polí atributů do souboru:
  `zapisTxt(ArrayList<int[][]> kolekceAtributu)`

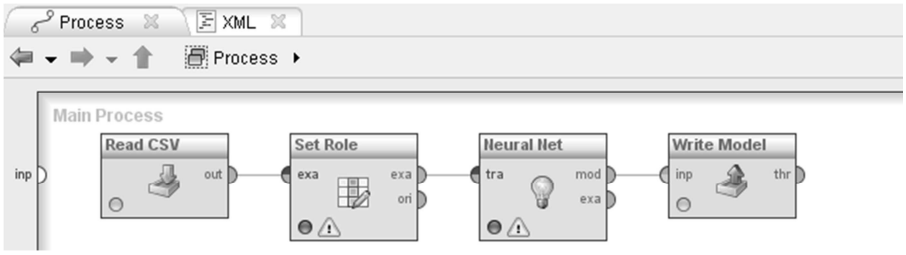
Vytvoření množiny pro rozpoznání atributů – množina znaků pro rozpoznání vzniká ihned při separaci znaků z obrázku s textem pro rozpoznání. Při separaci a normalizaci každého znaku z textu se získají ihned jeho atributy a k vytvoření množiny pro rozpoznání stačí pouze zavolat metodu pro uložení všech získaných atributů znaků. Atributy jsou poté uloženy do souboru a připraveny pro pozdější načtení a použití pro rozpoznání neuronovou sítí. Popis vytvoření této množiny je shodný s popisem vytvoření trénovací množiny (viz str. 21).

RapidMiner procesy – první proces, který jsem v RapidMineru vytvořil je proces učení neuronové sítě. Proces obsahuje operátor pro načtení csv souboru (textový soubor s uloženou trénovací množinou), operátor pro nastavení rolí atributů, neuronovou síť a operátor zapisující model neuronové sítě pro pozdější načtení procesem pro rozpoznání znaků. Operátoru sloužícímu pro načtení trénovací množiny je jako cesta k souboru, který má být načten, zvolena cesta k uložené množině sad definovaných atributů (konkrétně je to „\..\definovane.csv“). Nastavíme ještě oddělovač jednotlivých atributů v souboru trénovací množiny, což je „,“ a u operátoru pro nastavení rolí atributů vybereme definujícímu atributu roly „label“. V našem případě je to „attribute_0“, který má roly nastavenou na „label“. Zbývá už jen hlavní část procesu, a tou je nastavení neuronové sítě.

Po testování a nastavování různých počtů neuronů, učících cyklů, chybovosti a počtu skrytých vrstev, byla neuronová síť nastavena následovně. V neuronové síti je jedna skrytá vrstva. Tato vrstva obsahuje 45 neuronů. Počet učících cyklů sítě je 500

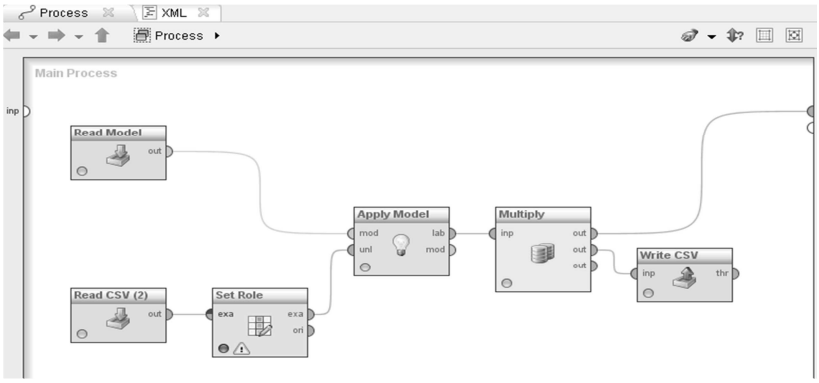
a chybovost byla nastavena na 0,001. Nastavení parametru momentum zůstalo na hodnotě 0,2. Tyto hodnoty vyšly po testech jako vhodné.

Posledním krokem v nastavení procesu učení sítě je nastavení cesty pro uložení modelu neuronové sítě. V našem případě je tato cesta nastavena na „vystupy_projektu\sit.mod“. Grafickou podobu proces učení znázorňuje obrázek 1.4.



1.4 RapidMiner proces pro učení sítě

Druhý proces RapidMineru slouží k rozpoznávání znaků. Jeho operátory jsou načtení modelu naučené neuronové sítě, načtení množiny pro rozpoznání, nastavení rolí, aplikátor modelu neuronové sítě a operátor pro zapsání výstupu neuronové sítě do csv souboru. Výstup neuronové sítě je napojen k výstupu procesu. Toto připojení je vytvořeno z toho důvodu, aby mohl být zachytáván výstup neuronové sítě i vlastní aplikace. Dále nastavíme všechny operátory. Operátor pro načtení neuronové sítě má nastavenou cestu shodně s operátorem zapisujícím model naučené sítě v procesu pro učení, což je „\..\definovane.csv“. Operátor pro načtení množiny pro rozpoznání má nastavenou cestu k souboru „..\atributy.csv“ a oddělovač jednotlivých atributů je opět nastaven na hodnotu „,“. Role atributu s názvem „attribute_0“ je nastavena na „label“ a operátoru pro zapsání výstupu neurnové sítě je nastavena cesta pro zapsání souboru na „vystupy_projektu\rozpoznane.csv“. Grafické zobrazení RapidMiner procesu pro rozpoznání je vidět na obrázku 1.5.



1.5 RapidMiner proces pro rozpoznání znaků

Takto jsou procesy připraveny pro spuštění z naší aplikace a je třeba je jen uložit do správné složky. Tou je složka „\DATA\rapidminer_process“ nacházející se v kořenovém adresáři vlastní aplikace.

Uložení výstupu neuronové sítě – protože ze souboru v němž je uložen výstup neuronové sítě, není pro člověka jednoduše, jasně a stručně čitelný predikovaný text, uloží se znaky, tak jak byly za sebou predikovány, do proměnné typu *String* a ta je posléze uložena do textového souboru. I za tímto účelem jsem vytvořil statickou třídu `RapidMinerControl`, ve které je metoda `getVystupProjektu()`, jejíž návratová hodnota je typu `String`. Metoda vrací rozpoznaný text, skládající se ze všech predikovaných znaků na výstupu neuronové sítě, řazených tak jak byly postupně za sebou separovány. Popis jak je toho dosaženo i s popisem ostatních metod a účelu této třídy následuje na dalších řádcích.

Přehled a příklady ovládání třídy `RapidMinerControl`:

Třída `RapidMinerControl` je statická a její metody slouží k obsluze funkcionalit programu RapidMiner. Za pomoci metod této třídy lze inicializovat program RapidMiner bez grafického rozhraní, spustit proces připravený pro spuštění programem RapidMiner, vrátit text skládající se ze znaků predikovaných neuronovou sítí, které byly získány z výstupu procesu nebo ukončit instanci RapidMineru a celou aplikaci. Popis toho, jak je RapidMiner pomocí těchto metod ovládán je zde:

- **Inicializace RapidMineru** – `initRapidm()`:

  Inicializace RapidMineru se provádí pomocí statické třídy RapidMiner, která se nachází v balíku com.rapidminer. Použijeme metodu `RapidMiner.init()` a tím RapidMiner inicializujeme. Před samotnou inicializací je možno nastavit parametry inicializace a například nezapínat komponenty RapidMineru, které nepotřebujeme pro naše účely spouštět. Tyto parametry lze nastavit metodou `System.setProperty` s parametry`(String parametr, String hodnotaParametru)`. Popis možných parametrů a jejich hodnot je k nahlédnutí v dokumentaci programu RapidMiner, která je součástí příloh.

- **Spuštění RapidMiner procesu** - `startProcess(String cesta)`:

  Ke spuštění procesu je třeba nejprve vytvořit instanci třídy `com.rapidminer.Process`. Je použit konstruktor třídy s parametrem typu `File`, který obsahuje soubor procesu RapidMineru. K samotnému spuštění a vykonání procesu dojde po použití metody `run()` z instance typu `com.rapidminer.Process`. Abychom mohli zachytit výstup procesu, vytvoříme se spuštěním procesu i instanci třídy `IOContainer` z balíku `operator` nacházející se v knihovnách RapidMineru a metodu `run()` spustíme s parametrem typu `IOContainer`. Příkaz tedy má následující tvar: `IOContainer io = run(new IOContainer())`. Tímto dojde ke spuštění a vykonání procesu tak, aby byly jeho výstupy zachyceny a bylo s nimi možno pracovat.

- **Získání predikovaného textu** - `getVystupProjektu()`:

  Abychom z výstupu procesu získali predikované znaky, je nutné nejprve z instance typu `IOContainer` odvodit instanci typu `ExampleSet`. Toho docílíme použitím metody `get(ExampleSet.class)` z instance typu `IOContainer`. Instance typu `ExampleSet` představuje celý výstup procesu. Procházením všech záznamů výstupu a přístupem k hodnotám daných atributů docílíme získání predikovaných hodnot. Záznamy výstupu procesu jsou typu `Example`. Projdeme tedy celou instanci typu `ExampleSet` a v každém průchodu vytvoříme instanci typu `Example` pomocí metody `getExample(int index)` z instance typu `ExampleSet`.

  Požadovaný sítí predikovaný znak z dané instance typu `Example` získáme pomocí metody `getValueAsString()` z této instance. Metodu `getValueAsString` je třeba zavolat s parametrem, který představuje atribut z množiny atributů výstupu procesu, ke kterému chceme přistoupit. Vytvoříme tedy instanci typu `Attributes` metodou `getAttributes()` z instance třídy `ExampleSet`. Jako parametr metody výše zmiňované metody `getValueAsString` použijeme návratovou hodnotu třídy `getPredictedLabel()` z instance typu `Attributes`. Pro upřesnění - příkaz, kterým přistoupíme k predikovanému znaku z daného záznamu ve výstupu procesu má tvar `example.getValueAsString(attributes.getPredicted-Label())`.

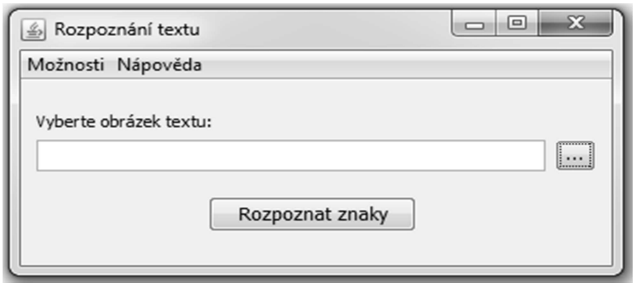
5.3 Funkcionality aplikace

Za účely ovládání funkcionalit programu jsem vytvořil statickou třídu `ProgramControler`, která obsahuje čtyři metody pro obsluhu celého systému. Pro uživatelsky příjemnější ovládání byla nad touto třídou vytvořena grafická rozhraní pro ovládání každé z metod třídy.

Přehled a příklady ovládání třídy `RapidMinerControl`:

- **Načtení a rozpoznání obrázku –** `rozpoznej(String cesta)`:

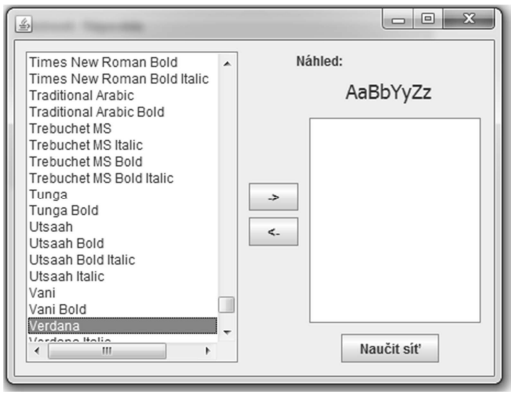
  Parametr cesta udává cestu k obrázku nebo složce obrázků. V metodě jsou postupně zapnuty funkce separování obrázků (instance třídy `CharCutting`), pro každý obrázek funkce pro získání atributů (instance třídy `CharAttributes`), zapnutí RapidMiner procesu pro rozpoznání, získání predikovaného textu z výstupu procesu a jeho uložení do textového souboru a následné otevření souboru. V závislosti na nastavení programu může metoda aktivovat funkci pro uložení separovaných znaků nebo použití výchozí neučené neuronové sítě při rozpoznání. Grafické rozhraní pro ovládání této metdy je ukázáno na obrázku 1.6.



1.6 GUI pro ovládání rozpoznání

- **Ovládání učení sítě** – `nauc(String[] fonty):`

Metoda vytvoří nejdříve instance třídy `UceniSite`, čímž dojde k vytvoření obrázků znaků za použití fontů předaných parametrem. V závislosti na nastavení programu proběhne nebo neproběhne i učení znaků obsahujících čárky a háčky. Tímto vznikne trénovací množina, která je použit a při následném spuštění RapidMiner procesu pro učení sítě. Tím je proces učení dokončen. Na obrázku 1.7 je vidět grafické rozhraní pro ovládání této metody.

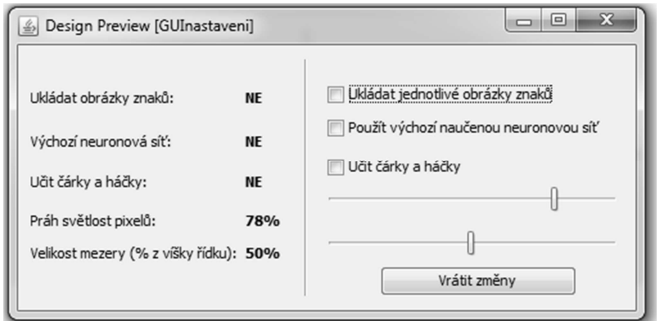


1.7 GUI pro učení sítě

- **Nastavení programu** – `nastaveni(boolean ukladejObrazky, boolean vychoziSit, boolean ucitCarkyHacky, double prahSvetlosti, double velikostMezery):`

Tato metoda slouží k nastavení funkcionalit běhu programu. Každý parametr třídy má svoji specifickou vlastnost. První parametr je typu `boolean` a v případě, že jeho hodnota bude „true“ jsou v průběhu procesu rozpoznání znaků a učení sítě ukládány obrázky znaků. Druhý parametr (`boolean vychoziSit`) určuje, zda pro rozpoznání použít výchozí naučenou neuronovou síť nebo neuronovou síť kterou si sami učíme. Další parametr (`boolean ucitCarkyHacky`) určuje, zda se mají při učení neuronové sítě učit i znaky obsahující čárky a háčky.

Další parametr je typu `double (double prahSvetlosti)` a pomocí něho lze nastavit práh světlosti pixelů, určující kdy bude pixel klasifikován jako tmavý a kdy jako světlý. Nastavení hodnot má smysl v intervalu mezi 0,1 a 0,99. Poslední parametr `(double velikostMezery)` slouží pro nastavení poměru velikosti světlého prostoru za separovaným znakem, nutným pro klasifikaci mezery. Jedná se o nastavení koeficientu, kterým bude násobena výška řádku, ze které byl znak separován. Hodnoty mají smysl v rozsahu mezi 0 až 1. Grafické rozhraní pro ovládání nastavení programu je vidět na obrázku 1.8.



1.8 GUI pro nastavení programu

Poslední metodou je metoda pro ukončení celé aplikace, která pouze zavolá metodu `exitRapidminer()`, ze třídy `RapidMinerControl`.

6 Testování a výsledky systému

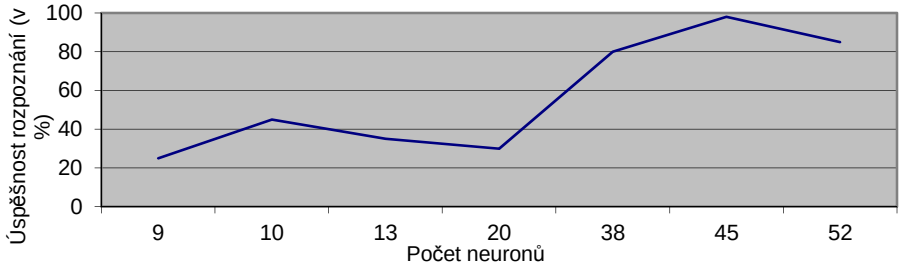
Otestování správnosti práce systému a spouštění požadovaných funkcionalit systému ve správnou chvíli ve správném pořadí. Volba optimálních hodnot nastavení systému. Vyhodnocení výstupů systému v závislosti na předložených vstupech.

6.1 Testování aplikace

Pro správný chod aplikace je třeba mít připravenou správnou strukturu adresářů a složek v kořenovém adresáři aplikace. Kořenový adresář, ze kterého je aplikace spouštěna musí tedy obsahovat složku „DATA“, v níž jsou složky „rapidminer_process“, „znaky_k_rozpoznani“ a „znaky_naucene“. Ve složce „rapidminer_process“ se musí nacházet soubory rozpoznani.rmp (RapidMiner proces pro rozpoznání), rozpoznani_default.rmp (RapidMiner proces pro rozpoznání s využitím defaultní neuronové sítě), uceni.rmp (RapidMiner proces pro učení neuronové sítě) a složka „vystupy_projektu“ obsahující soubor sit_default.mod (výchozí naučená neuronová síť). Toto je souborová struktura nutná pro bezchybnou práci aplikace, což jsem ověřil otestováním funkčnosti spuštění všech funkcionalit systému. Test správného běhu aplikace proběhl úspěšně.

6.2 Optimální nastavení

Podstatným nastavením, jehož optimální hodnoty je třeba nalézt, je nastavený hodnot neuronové sítě. Při měnění počtu neuronů sítě, počtu učících cyklů a chybovosti sítě, byla vyhodnocována správnost znaků predikovaných neuronovou sítí. Testování nastavení začalo za použití dvou neuronů a jejich počet byl postupně navyšován. Než se počet neuronů dostal na deset, pohybovala se úspěšnost rozpoznání v rozmezí mezi 16% a 25%. Při použití deseti neuronů stoupla úspěšnost rozpoznání na 45%. Tato hodnota zůstala podobná do použití dvanácti neuronů. Při použití třinácti a více neuronů opět schopnost sítě rozpoznávat znaky značně klesla a to na 35% s tím, že s přibývajícím počtem neuronů úspěšnost rozpoznání klesala. Dobré výsledky úspěšnosti rozpoznání se začaly objevovat při použití třiceti osmi neuronů. V tomto případě se úspěšnost pohybovala okolo 80%. Při pokračování v navyšování počtu neuronů úspěšnost rozpoznání stoupala. Nejvyšší úspěšnosti rozpoznání bylo dosaženo při použití čtyřiceti pěti neuronů. Zde se hodnota úspěšnosti rozpoznání pohybuje okolo 98,5%. Při dalším navyšování počtu neuronů nebylo dosaženo větší úspěšnosti. Naopak při překročení hranice padesáti dvou neuronů začala úspěšnost rozpoznání opět klesat. Graf závislosti počtu neuronů na úspěšnosti rozpoznání je zobrazen na obrázku 1.9.



1.9 Graf závislosti počtu neuronů na úspěšnosti rozpoznání

Nejvhodnější možné nastavení neuronové sítě je tedy čtyřicet pět neuronů v jedné skryté vrstvě a počet učících cyklů byl po testování v závislosti na chybovosti při učení sítě nastaven na pět set cyklů. Větší hodnota nemá smysl, jelikož pouze prodlužuje dobu učení neuronové sítě, ale výsledky rozpoznání se od této hodnoty již nemění.

Dále je třeba nalézt optimální hodnoty prahu světlosti, dle kterého jsou klasifikovány pixely jako tmavé nebo. Proběhl tedy test závislosti hodnoty prahu světlosti na správnosti separování znaků textu a jejich rozpoznání. Testování proběhlo na nejčastěji se vyskytujícím se textu, to znamená normálním textu o velikosti 12 bodů. Použitý font byl Verdana.

Při nastavení prahu světlosti na nejnižší možnou hodnotu (což je 1% světlosti) je separace znaků naprosto špatná. Důvodem je to, že jako světlé pixely, jsou klasifikovány ty pixely, jejichž světlost je 1% a více. U skenovaných textů jsou to tedy skoro všechny pixely a to i pixely vykreslující znak. Jako tmavé pixely jsou klasifikovány pouze opravdu tmavé pixely, dá se říci, že pouze černé (v RGB modelu to znamená červená = 0, modrá = 0, zelená = 0 a všechny barvy s o procento větší světlostí). Nedojde tedy k identifikaci znaků a nemusí dojít ani k identifikaci řádků. Pokračuje se tedy zvyšováním této hodnoty, dokud výsledky separace znaků nejsou pozitivní. Hodnota, na které začíná docházet ke správné separaci znaků, je 35% světlosti. Chyby se ale objevují u písmen jako „m“, „n“, „u, které jsou v jejich nejužším rozděleny a separovány jako dva znaky. Postupujeme tedy ve zvyšování hodnoty dál až do dosažení co nejlepších výsledků. Hodnota, při které dochází ke korektnímu separování znaků i s klasifikací mezer je 70%. Tato hodnota je zvolena jako výchozí hodnota prahu světlosti pro klasifikaci tmavosti bodů.

Druhou proměnnou, jejíž optimální hodnotu je třeba nalézt je hodnota velikosti mezery. Opět začneme na nejnižší hodnotě a postupným navyšováním a následnou kontrolou výstupu rozpoznání nalezneme hodnotu, při níž dochází k nejlepším výsledkům rozpoznání. Správná nalezená hodnota, při níž dochází ke korektní klasifikaci mezer, je 50%.

Tyto hodnoty jsou průměrně nejlepšími, ale u každého textu se můžou optimální hodnoty pohybovat o několik procent výše či níže. Pro dosažení co nejlepších výsledků je třeba hodnoty upravovat testovat. Nicméně výše popsané optimální hodnoty slouží jako směrodatné hodnoty, ze kterých lze vycházet.

6.3 Testování vstupů

Při testování rozpoznání na „čistém“ obrázku textu, bez výskytu šumů, se průměrná úspěšnost rozpoznání se sítí naučenou stejnému fontu, kterým je napsán rozpoznávaný text, pohybuje okolo 98%. Z obrázku textu, ve kterém se vyskytuje přijatelná hranice šumu, proběhne rozpoznán s úspěšností pouze o málo menší. Tato úspěšnost se pohybuje okolo 86%. Při pokrytí obrázku „silnějším“ šumem, značně klesá schopnost rozpoznat znaky a správnost rozpoznání se pohybuje pouze okolo 20%. Při použití výchozí neuronové sítě, která je naučena skupině všech často používaných fontů, se procento úspěšnosti rozpoznání textu z „čistého“ obrázku pohybuje okolo 85%. Při rozpoznávání obrázku textu s přijatelnou hranicí šumu je úspěšnost rozpoznání přibližně 73%.

U obrázku se „silnějším“ šumem je procento úspěšnosti rozpoznání opět o poznání nižší a to okolo 15%. Toto je dáno algoritmem pro separování znaků.

Separace znaků proběhne se špatnými výsledky jak kvůli tomu, že je šum na obrázku překreslen i do separovaného znaku a tím pádem proběhne i chybné získání atributů obrázku znaku, tak kvůli faktu, že tmavé body šumu v prázdném (světlém) prostoru obrázku jsou vyhodnocovány jako znak vždy, když jejich tmavost přesáhne práh světlosti nastavený uživatelem. Procento úspěšnosti rozpoznání se tedy velice mění v závislosti na nastavení parametrů procesu rozpoznání.

V důsledku použitého algoritmu pro separování znaků není možné rozpoznávat ani text v tabulkách, či text na obrázku, který je orámovaný, protože nedojde k detekci řádků, tím pádem ani písmen. Problém není pouze v orámování ale ve faktu, že se v textu vyskytuje tmavý sloupec pixelů, vykreslený přes více řádek (v případě okrajů přes všechny řádky). Stejný problém se naskytne v případě výskytu tmavého řádku pixelů v obrázku. Takový řádek nedovolí provést správnou separaci znaků z řádku.

Dalším problémem, který není možné řešit stávajícím algoritmem pro separování obrázků znaků, je separování znaků s diakritikou. Čárky, háčky a podobná znaménka nad znaky, jsou separovány samostatně bez znaku, kterému náleží. Diakritická znaménka jsou tedy programem vyhodnocena jako samostatné znaky k rozpoznání. Například písmeno „Ě“ je vyhodnoceno jako dva znaky. V prvním obrázku znaku bude v obrázku vykresleno „ˇ“, v druhém obrázku znaku bude vykresleno „E“. Dochází tedy k naprosté nemožnosti text rozpoznat.

Rozpoznání textu je tedy možné z obrázků, na kterých je vepsán tmavý text na světlém pozadí.

7 Návrh pro budoucí řešení

Hlavní částí systému, která by mohla být vylepšována, je technika předzpracování obrázku s textem k rozpoznání. Konkrétně jde o techniku separace znaků a získání obrázků znaků, určených k rozpoznání. **V tomto ohledu je možné vytvořit a zlepšit následující techniky:**

- Vytvoření techniky umožňující nalezení textu na obrázku obsahujícím čáry, jako například tabulky s textem.

- Vytvoření techniky umožňující nalezení textu a separaci jeho znaků z obrázků jako jsou například fotografie. Umožnit například separování znaků textu z náhrobků na fotografii.

- V případě výskytu šumů v obrázku s textem pro rozpoznání, použít techniku pro „vyčištění“ obrázku

- Pro zkvalitnění zobrazení znaku vepsaného v obrázku znaku, použít techniku, umožňující zvýraznění vykresleného znaku v obrázku.

- Zlepšit techniku pro identifikaci mezery v textu

Z části spravující funkcionality pro rozpoznání znaků obrázků je možné zlepšit následující:

- Vytvořit RapidMiner procesy přímo ve vlastní aplikaci, bez nutnosti jejich načítání z disku.

- Neukládat trénovací množinu neuronové sítě a množinu k rozpoznání, ale data odeslat přímo do vytvořeného RapidMiner procesu.

8 Závěr

Po shrnutí všech popsaných záležitostí lze říct následující:

Vytvoření zjednodušeného systému pro rozpoznávání písmen:

Toto bylo splněno**.** Systém je schopen rozpoznat tištěný text a v případě správného nastavení lze dosáhnout velice příznivých výsledků. Z části popisující možnosti vylepšení tohoto systému bylo zmíněno, jaké všechny techniky by bylo možné z hlediska separace znaků vylepšit. Fakt, že algoritmus pro separaci znaků má své popsané nedostatky, nemění nic na to, že systém jako celek je schopen tištěný text rozpoznávat a jelikož zadání znělo vytvořit zjednodušený systém pro rozpoznávání písma, hlavní důraz nebyl kladen na algoritmus pro separování znaků. Hlavním cílem byla demonstrace využití neuronové sítě, čímž se dostáváme k dalšímu splněnému cíli.

Využití neuronové sítě pro rozpoznávání znaků:

Tento cíl byl také splněn. Pro rozpoznání znaku vepsaného v obrázku separovaného znaku byla použita neuronová síť a výsledky demonstrují možnost jejího využití pro tento řešený problém.

Využití existujících dostupných softwarů a knihoven:

I tento cíl byl splněn**.** Neuronová síť je implementována pomocí programu RapidMiner a samotný proces rozpoznání i učení sítě je řízen a vzniknul ta pomocí programu RapidMiner. Dále byly využity RapidMiner knihovny. Po seznámení se s těmito knihovnami a jejich použitím, v kombinaci se seznámením se, se způsobem práce programu RapidMiner bylo dosaženo schopnosti učení neuronové sítě i rozpoznávání znaků neuronovou sítí za pomoci těchto komponent.

Webová uživatelská příručka:

Byl vytvořen webový průvodce systémem vysvětlující dané postupy a techniky pro dosažení požadovaných cílů. V příručce je ukázáno použití neuronové sítě pro řešený problém a využití programu RapidMiner a jeho knihoven. Toto jsou věci, jejichž využití je možné uplatnit ve výuce. Bylo tedy splněno jak vytvoření webového průvodce systémem, tak vytvoření prostředku demonstrujícího možnost využití neuronové sítě pro rozpoznávání textu ve výuce.

Během vytváření práce jsem se blíže seznámil jak s různými systémy, tak s metodami a programovou podporou pro rozpoznávání tištěného písma. Byly tedy splněny všechny body zadání.

9 Použitá literatura

[1] *Rapid-i* [online]. 18.10.2011 [cit. 2011-02-02]. Integrating RapidMiner into your application. Dostupné z WWW: <http://rapid-i.com/wiki/index.php?title=Integrating_RapidMiner_into_your_application>

[2] *Character Recognition* [online]. 1.3.1996 [cit. 2011-01-31]. Character Recognition Overview. Dostupné z WWW: <http://www.eecs.berkeley.edu/~fateman/kathey/char_recognition.html#compare>

[3] YUV. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, , last modified on 12. 9. 2010 [cit. 2011-02-02]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/YUV>.

[4] *RapidMiner 5.1 Class Documentation* [online]. 2009 [cit. 2011-03-20]. RapidMiner 5.1 Class Documentation. Dostupné z WWW: <http://rapid-i.com/api/rapidminer-5.1/overview-summary.html>

[5] *Google.com* [online]. 1997 [cit. 2011-04-03]. OCR image preprocessing method for image enhancement of scanned documents. Dostupné z WWW: <http://www.google.cz/patents?id=dp4hAAAAEBAJ&printsec=abstract#v=onepage&q&f=false>

[6] *Neurogy.com* [online]. 2004 [cit. 2011-01-14]. OCR Pre-Processing White Paper. Dostupné z WWW: <http://www.neurogy.com/ocrpreproc.html>

[7] *Freepatentsonline.com* [online]. 1997 [cit. 2011-04-26]. OCR image preprocessing method for image enhancement of scanned documents . Dostupné z WWW: <http://www.freepatentsonline.com/5594815.html>

[8] *UCI Machine Learning* [online]. 2010 [cit. 2011-04-26]. Optical Recognition of Handwritten Digits Data Set Download: Data Folder, Data Set Description Abstract: Two versions of this database available; see folder. Dostupné z WWW: <http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits>

[9] *Leenissen.dk* [online]. 2008 [cit. 2011-04-26]. FANN. Dostupné z WWW: <http://leenissen.dk/fann/wp/>

[10] *Cra.org* [online]. 2008 [cit. 2011-01-13]. A Morphological Image Preprocessing Suite for OCR on Natural Scene Images. Dostupné z WWW: <http://www.cra.org/Activities/craw_archive/dmp/awards/2008/Elmore/melmore_dmp.pdf>

[11] *Laral.istc.cnr.it* [online]. 2003 [cit. 2011-02-02]. Neural Networks Library In Java. Dostupné z WWW: <http://laral.istc.cnr.it/daniele/software/NNLibManual.pdf>

Přílohy

[A] CD s vlastní aplikací a jejími zdrojovými kódy, s dokumentací k vytvořené java aplikaci, s dokumentace ke knihovnám programu RapidMiner a webovým průvodce aplikací.

[B] CD s textem bakalářské práce v elektronické podobě a vlastní java aplikací včetně dokumentací.