# Doprovodný dokument k aplikaci OCR Sudoku Solver

Lukáš Jendele – IPSS 1. Ročník MFF UK

Programování II – zápočtový program

## Stručné zadání:

Vytvořit aplikaci, která rozpozná číslice z tabulky a vyřeší sudoku.

## Přesné zadání:

Vytvořit aplikaci s uživatelským rozhraním pomocí WinForms, která bude umět rozpoznat číslice z tabulky (z jednoho zdroje, jedním fontem, jednou velikostí) a pomocí metody *Backtracking* vyřeší sudoku v rozumné době.

## Mé rozšíření zadání:

Aplikace bude zpracovávat více fontů, velikostí a zdrojů.

## Zvolený algoritmus:

V aplikaci se používají tři základní algoritmy. První předzpracuje obrázek pro metodu rozpoznávání číslic, druhý rozpoznává číslice a třetí nakonec vyřeší samotné zadání sudoku. Metodu předzpracování obrázku jsem nazval segmentace. Ta probíhá následovně: Vstupní obrázek sudoku se převede na černobílý (v závislosti na nastavitelné hranici jasu barvy), poté se najdou všechny nepřerušované vodorovné a svislé čáry a vyfiltrují se ty, které spolu sousedí. Na vlastní rozpoznání čísel jsem vyzkoušel jednoduchý genetický algoritmus. Jeho princip spočíval v hledání náhodné n-tice bodů, tak aby program v zadaném obrázku našel 9 rozdílných číslic. S tímto postupem jsem bohužel nedosáhnul dostatečně uspokojivých výsledků, jelikož algoritmus předpokládal, že všech 9 číslic se objeví v zadání a nezvládal šum v obrázcích, a proto jsem sáhnul po mocnějším nástroji: neuronové síti s algoritmem učení zpětné propagace. O algoritmu se o něco podrobněji zmiňuji v programátorské dokumentaci. Většinu informací jsem čerpal z http://neuralnetworksanddeeplearning.com/. Jako alternativu jsem vyvinul metodu Digit Patterns, která sice není tak úspěšná jako neuronové sítě, ale přesto poskytuje jednoduchou a účinnou cestu, která vede k rozumným výsledkům. Konečně vlastní řešení sudoku se provádí backtrackingem s ořezáváním optimalizovaným pomocí chytrého bitového pole.

## Uživatelská dokumentace:

### O aplikaci:

Aplikace slouží k vyřešení zadání úloh sudoku. Aby bylo zadávání jednodušší, zvládá OCR Sudoku Solver načtení obrázku, který sudoku obsahuje.

Použití:

Nejprve spustíme aplikaci OCR Sudoku Solver a měli bychom vidět úvodní obrazovku, která v pravém dolním rohu obsahuje dvě tlačítka. Klikneme na tlačítko *Next* a vybereme si metodu zadávání sudoku. Pokud máme sudoku k dispozici jako bitmapový obrázek (případně ho můžeme jednoduše vytvořit pomocí Windows funkce *PrintScreen* a následného oříznutí) zvolíme způsob *From image file*. Následně zmáčkneme tlačítko *Browse* a příslušný soubor nalezneme na disku a otevřeme. Soubor se načte a aplikace sama rozpozná číslice v tabulce, které můžeme následně podle obrázkové předlohy doopravit. Pokud se to aplikaci nepovede, oznámí to uživateli a v tom případě doporučuji druhou manuální metodu. Pokud obrázkem nedisponujeme, zvolíme metodu manuálního zadávání *From table* a vyplníme tabulku v levém dolním rohu ručně. Když je tabulka správně vyplněná, stiskneme tlačítko *Solve*. Pokud má sudoku zadané do tabulky řešení, pak se zobrazí na následující obrazovce v přehledné tabulce, kde jsou tučně vyobrazeny původně uživatelem zadané číslice. Pokud řešení neexistuje, oznámí to Sudoku Solver uživateli.

Nastavení:

V levém dolním rohu se nachází tlačítko *Settings* a po jeho kliknutí se zobrazí nový dialog, který umožňuje upravit chování celé aplikace. V prvé řadě je možné nastavit OCR metodu tj. metodu rozpoznávání číslic. Nabízí se jednodušší metoda *Digit Patterns* a složitější, ale přesnější metoda *Neural Network*. Neuronovou síť si můžeme sami vytvořit a vycvičit v tomtéž dialogu. Pomocí tlačítka *Browse* nastavíme složku se soubory obrázků číslic o rozměrech 30x30 pixelů, které mají na první pozici ve jméně souboru číslici, jež obsahují. Poté nastavíme složku, kam se mají v jednotlivých učících epochách ukládat neuronové sítě. Zbývající parametry specifikují, jak se má neuronová síť učit (jejich správné nastavení vyžaduje základní znalost neuronových sítí a trpělivost). Méně zkušeným uživatelům nedoporučuji parametry měnit. Nakonec klikneme na tlačítko *Start* a výuka neuronové sítě započne. Vytvořenou neuronovou síť je možno nastavit jako výchozí tak, že změníme *Default neural network* na *Neural network from file* a zvolíme cestu k souboru. Tlačítko *Stop* učení sítě přeruší. Poslední prvek, který jsem nezmínil je parametr *Brightness to be regarded as white*, který určuje maximální jas barvy obrázku, aby byla považovaná za bílou a nikoliv za černou. Nastavení se uloží pomocí *Ok*, zruší pomocí *Cancel* a resetuje tlačítkem *Restore defaults*.

## Reprezentace vstupních dat:

Aplikace bere jako vstup zadání Sudoku a nabízí dvě metody zadávání, ruční zadávání a rozpoznávání zadání z bitmapového obrázku pomocí OCR čtečky. Soubory pro rozpoznávání musí být formátu png, jpg nebo bmp a smí obsahovat pouze jednoduchou

tabulku. V případě, že bychom chtěli rozpoznávat číslice pomocí vlastní neuronové sítě (ze souboru), tak pouze pomocí takové, kterou jsme vytvořili touto aplikací v dialogu nastavení.

## Reprezentace výstupních dat:

Výstup aplikace, tedy řešení sudoku, je přehledně zobrazeno na poslední kartě v tabulce, jež má tučně zvýrazněné číslice, které byly součástí zadání.

## Programátorská dokumentace:

### Segmentace obrázku:

Než na obrázku můžeme identifikovat číslice, musíme ho nejdříve předpřipravit a číslice nalézt (tuto fázi jsem nazval segmentace obrázku). To obstarává třída OCRReader a dialog LearnWindow. Obrázek se nejdříve převede na černobílý podle toho, jestli jas barvy je větší než nastavená mez či nikoliv. Poté se v obrázku vyhledají všechny vodorovné a svislé čáry, jejichž délka je větší než 80% výšky/šířky obrázku. Následně se sousední čáry stejné délky vyfiltrují (berou se jako jedna tlustá čára). Tímto máme obrázek rozdělený na mřížku. Pak už stačí projít všechny vyplněné buňky mřížky (obsahují více jak 1% černé barvy), v nich najít číslici a tu zvětšit/zmenšit na velikost 30x30 pixelů s danými okraji. Poznámka: LearnWindow (název z historických důvodů – v tomto dialogu se měl učit genetický algoritmus) je třída, která by se mohla sjednotit s OCRReader v jednu. Třídy jsou rozděleny, protože jsem používal dialog LearnWindow k ladění aplikace. Veškerou provedenou práci s obrázkem jsem si zobrazoval do prvku typu PictureBox, kde se dala snadno zkontrolovat korektnost operací.

### Neuronová síť:

Pro vysvětlení, co přesně neuronová síť je a proč a jak funguje, se odkazuji na knihu pana Nielsena na webu: http://neuralnetworksanddeeplearning.com/ Pro svou aplikaci jsem zvolil neuronovou síť, která se učí algoritmem zpětné propagace (backpropagation). Váhy jednotlivých spojení mezi neurony jsou uložené v maticích a bias hodnoty jednotlivých neuronů ve vektorech. Všechny hodnoty jsou inicializovány pomocí náhodných čísel s Gaussovou distribucí o průměru 0 a standardní odchylkou $\frac{1}{\sqrt{n}}$, kde n je počet vstupních neuronů. Implementovaná Cost funkce je CrossEntropyCostFunction (má rychlou schopnost učení neuron. sítě) a je regularizovaná pomocí metody L2-regularization, která spočívá v preferování učení se nízkých vah spojení v síti. Regularizace pomáhá zabránit tzn. přeučení neuronové sítě (například naučení se šumu obsaženého v obrázku).

Učící algoritmus nazývaný také jako Backpropagation se řídí následujícími základními instrukcemi. ($w^l$ je matice ohodnocení a $b^l$ je vektor bias hodnot pro vrstvu l)

1) Nastav vstupní hodnoty do neuronů na vstupní vrstvě
2) Pro všechny vrstvy l=2,3...L spočítej:$z^l = w^l a^{l-1} + b^l$ a $a^l = \sigma(z^l)$, *kde* $\sigma(z) = \frac{1}{1+e^{-z}}$
3) Spočítej chybu ve výstupní vrstvě: $\delta^l = \nabla C \odot \sigma'(z^L)$, kde $\odot$ znamená násobení po složkách
4) Chybu propaguj zpět vrstvami l=L, L-1, ..., 2:$(w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)$
5) Aktualizuj hodnoty w a b následovně: $w^l \to w^l - \frac{\eta}{n} \sum \delta^l (a^{l-1})^T$ a $b^l \to b^l - \frac{\eta}{n} \sum \delta^l$

Pro podrobnosti a důkaz se opět odkazuji na web p. Nielsena, který vše dokonale popisuje.

Použití neuronové sítě:

Když už máme implementaci neuronové sítě hotovou, tak je potřeba si svou vlastní síť na trénujících datech vytvořit. To se dělá pomocí funkce StochasticGradientDescent. Pro účely rozpoznávání čísel jsem použil neuronovou síť o třech vrstvách s parametry: η=0,3 λ=2 miniBatchSize=20, počet neutronů v mezivrstvě=30. Síť, kterou jsem uložil jako výchozí, se vytvořila v 10. epoše a správně vyhodnotila 10111/10162 číslic. Pokud máme síť již vytvořenou, voláme na ní funkci FeedForward a z navraceného výsledného vektoru vybíráme složku s nejvyšší aktivační hodnotou (funkce MaxArg).

Metoda Digit Patterns:

Jedná se o jednoduchou metodu, kterou jsem vyvinul jako alternativu sloužící k rozpoznávání číslic. V aplikaci je uložených 9 obrázků, každý s tučně napsanou číslicí. Při rozpoznávání se vezme obrázek s neznámou číslicí a porovná se pixel po pixelu proti 9 uloženým vzorům. Ten, který se shoduje v nejvíce černých pixelech, se bere jako výsledek. Metoda je velmi jednoduchá a přitom dává poměrně dobré výsledky. Problém jsem zpozoroval u několika vstupů u číslic 6 a 8 nebo také u 2 a 7. Nicméně ten by se dal vyřešit například dalším porovnáním pouze jedné poloviny obrázku.

Zbývající třídy a jejich použití:

Třída *MyBitArray*:

Třída, jak název napovídá, implementuje pole pro nastavování bitů. Vlastní hodnota je reprezentovaná pomocí typu unsigned interger. Třída poskytuje základní bitové operace jako je OR, XOR, AND a některé další šikovné funkce. Používá se v projektu pro práci a vyhodnocování sudoku pravidel pro řádky, sloupce a sektory.

Třída *Settings*:

Dialog pro správu nastavení aplikace. Nastavení se ukládá pomocí C# třídy Properties.Settings, která se sama stará o ukládání a načítání xml dat. Místo uložení na disku je v "AppData\Local\OCRSudokuSolver". Dialog mimo jiné dokáže vytvářet (učit) nové neuronové sítě. Volání téměř kopíruje volání funkce *StochasticGradientDescent* třídy *NeuralNetwork,* pouze počet vrstev neuronových sítě a počet vstupních a výstupních neuronů je pevný. Průběžná úspěšnost se testuje na trénujících datech.

Třída *SudokuSolver*:

Slouží pro hledání vlastního řešení sudoku. V konstruktoru se jí předá pole čísel se zadáním. Nejzajímavější je metoda *SolveSudoku*, která pomocí klasického backtrackingu s ořezáváním nachází řešení. Algoritmus postupuje jako člověk, nejdříve si pro každé prázdné políčko spočte počet možností na vyplnění a vyplní to políčko, které nabízí nejméně možností a zavolá se tatáž funkce. Pokud najde řešení, ta metoda skončí, v opačném případě zkusí do políčka doplnit další možnosti. Na algoritmu je zajímavá práce s bitovými poli MyBitArray, které rychlost výpočtu velmi vylepšují.

Třídy *MyVector, My2DMatrix*:

Má vlastní implementace lineární algebry potřebná k fungování neuronové sítě. Podporují základní operace: maticový součin, skalární součin a operace po složkách. Třídy také obsahují serializační metody, které volá třída NeuralNetwork.

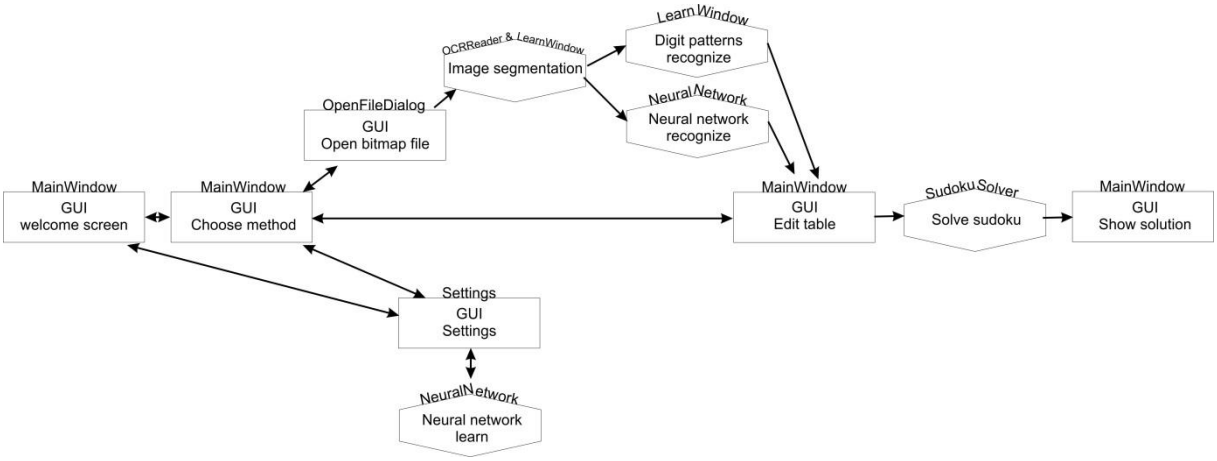
Soubor *Constants.cs*:

Soubor, ve kterém se udržují konstantní parametry pro celý projekt. Aby mohly být konstanty součástí tříd, ve kterých se používají, jsou třídy označeny pomocí klíčového slova *partial*.

Ostatní podprogramy:

Všechny ostatní funkce, které jsem zde vynechal, jsou buď součástí uživatelského rozhraní, nebo se jedná pouze o pomocné podprogramy, jejichž funkce je z názvu, komentáře nebo obsahu očividná (například XMLSerializace třídy *NeuralNetwork*).

Workflow programu:




## Průběh práce:

Nejprve jsem si napsal základní segmentaci obrázku a vyzkoušel jednoduchý genetický algoritmus (popsaný výše), ale bohužel bez uspokojivých výsledků. Na internetu jsem se dozvěděl o neuronových sítích, a tak jsem začal s jejich samostudiem. Po pročtení několika článků jsem narazil na http://neuralnetworksanddeeplearning.com/, kde jsou sítě velmi srozumitelně a přehledně popsané a také přiloženou ukázku v jazyku Python jsem shledal užitečnou. Pak už stačilo jen napsat a odladit vlastní implementaci v jazyce C# včetně základních funkcí lineární algebry, najít na internetu vhodný training set a nastavit parametry neuronové sítě, tak aby se síť efektivně učila, což se ukázalo jako celkem obtížný úkol. Následovala třída SudokuSolver a posledním krokem bylo UI a tato dokumentace.

## Další rozšíření a vylepšení:

1) Aplikace by mohla místo jednoho řešení zobrazit (nalézt) všechna, a tak i ověřit unikátnost řešení zadávaného sudoku.
2) Alternativní metoda rozpoznávání číslic (Digit Patterns) by mohla být rozpracovanější, tak aby poskytovala přesnější identifikaci číslic
3) Při načítání obrázku by se mohl objevit nástroj na oříznutí, případně navrhnout oblast k ořezání
4) Segmentace obrázku by mohla být sofistikovanější (podpora čárkované/křivé čáry, natáčení obrázku atp.)
5) Grafické označení porušení pravidel Sudoku hned při zadávání
6) Pokud by výstupy metody rozpoznávání číslic naznačovaly nejistotu identifikace, tak by to aplikace oznámila uživateli
7) Učení neuronové sítě, analýza obrázku i řešení sudoku by mohlo běžet na vlastním vlákně – momentálně vše běží na jednom hlavním vlákně
8) Podrobnější dokumentace

## Závěr:

Použité metody pro rozlišení číslic se ukázaly jako vhodně vybrané. Neuronové sítě fungují téměř bezchybně a metoda DigitPatterns má úspěšnost okolo 80%, Ta by se dala ještě vylepšit (viz výše). Výpočet vlastního řešení Sudoku je velmi rychlý a efektivní. Od zápočtového programu jsem požadoval, abych se na něm mnoho naučil. V mém případě to byly hlavně neuronové sítě. Kromě sítí jsem se naučil práci s bitmapami, souřadnicemi a grafikou GDI+, XML serializaci a mnoho jiných užitečných prvků jazyka C#. Celkově projekt splnil má očekávání, odnáším si z něho mnoho cenných informací, které jsem při jeho tvorbě načerpal.

## Sada testovacích příkladů:

Testovací obrázky se zadáním sudoku jsou přiloženy k projektu v adresáři test_pictures\. Sada číslic pro vyučení neuronové sítě se nachází v adresáři test_pictures\trainingResized\

## Použité zdroje:


http://neuralnetworksanddeeplearning.com/ - veškeré potřebné informace o neuronových sítích

https://github.com/mnielsen/neural-networks-and-deep-learning - ukázka v Pythonu (hlavně network2.py)

http://stackoverflow.com/questions/218060/random-gaussian-variables – algoritmus pro generování náhodných proměnných s Gaussovou distribucí

http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

a

http://www.boyter.org/2013/09/collection-letters-training/ - zdroj trénující sady číslic

https://github.com/pasnox/oxygen-icons-png - balíček ikon, z nichž některé jsou v UI

Ostatní články o neuronových sítích, které jsem neshledal tak užitečnými:

http://www.codeproject.com/Articles/4392/Neural-Dot-Net-Pt-Introduction?fid=15954&fr=29

http://www.codeproject.com/Articles/4392/Neural-Dot-Net-Pt-Introduction?fid=15954&fr=29

http://link.springer.com/chapter/10.1007%2F978-3-642-30223-7_87#page-1

https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf

https://en.wikipedia.org/wiki/Delta_rule

a další…