VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

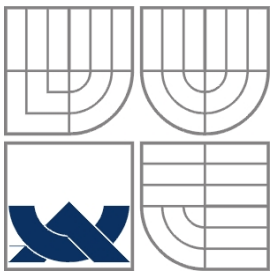
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

METODY DETEKCE A REPREZENTACE HRAN V OBRAZE

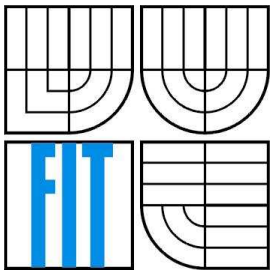
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JAN BERÁNEK
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

METODY DETEKCE A REPREZENTACE HRAN V OBRAZE

EDGE DETECTION AND REPRESENTATION ALGORITHMS FOR OBJECT RECOGNITION

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

JAN BERÁNEK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. MICHAL ŠPANĚL

BRNO 2007

Metody detekce a reprezentace hran v obraze

Edge Detection and Representation Algorithms for Object Recognition

Vedoucí:

Španěl Michal, Ing., UPGM FIT VUT

Oponent:

Herout Adam, Ing., Ph.D., UPGM FIT VUT

Zadání:

1. Prostudujte základy zpracování obrazu.
2. Zorientujte se v metodách detekce a reprezentace hran a hranic objektů (hranové detektory, kontury a další).
3. Vyberte několik zajímavých metod a zaměřte se na jejich použití v praxi (rozpoznávání objektů, OCR, apod.).
4. Experimentujte s implementací, případně navrhněte vlastní modifikace metod.
5. Diskutujte dosažené výsledky a možnosti budoucího vývoje. Zvažte další pokračování v rámci diplomové práce.
6. Vytvořte stručný plakát prezentující vaši bakalářskou práci, její cíle a výsledky.

Kategorie:

Zpracování obrazu

Implementační jazyk:

C/C++, C# nebo Python

Operační systém:

MS Windows, Linux (pokud možno přenositelný kód)

Literatura:

- Dle pokynů vedoucího.

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologii Vysokého učení technického v Brně.

Abstrakt

Tato práce se zabývá metodami hranové detekce a reprezentace hran v digitálním obraze. Shrnuje a popisuje některé používané metody. Dále se zaměřuje na návrh a popis implementace jednoduchého OCR systému založeného na hranové detekci a využívajícího umělou neuronovou síť pro rozpoznávání velkých strojově psaných písem anglické abecedy. Systém je napsán v jazyce C a využívá grafické knihovny OpenCV a knihovny FANN pro tvorbu umělé neuronové sítě. Aplikace je vytvořena a testována v prostředí operačního systému Windows XP.

Klíčová slova

OCR, hranový detektor, Freemanův řetězcový kód, Houghova transformace, Cannyho hranový detektor

Abstract

Main topic of this work are edge detection and representation algorithms used in an digital image. Some of typically used algorithms and methods are shown and described. Next aim of this work is design and description of implementation of a simple OCR system based on edge detection with use of an artificial neural net for a machine printed character recognition. The system was written in C programming language and uses OpenCV graphic library and FANN library for creating a neural net. Application was created and tested under Windows XP operating system.

Keywords

OCR, edge detector, Freeman chain code, Hough transformation, Canny edge detector

Citace

Jan Beránek: Metody detekce a reprezentace hran v obraze, bakalářská práce, Brno, FIT VUT v Brně, 2007

Metody detekce a reprezentace hran v obraze

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením ing. Michala Španěla
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jan Beránek
14.května 2007

Poděkování

Rád bych poděkoval vedoucímu práce za konstruktivní kritiku a připomínky při vývoji programu i při psaní této práce.

Obsah

1 Úvod

Počítačové zpracování obrazu je disciplína s dlouhodobou historií a širokým polem využití v praxi. V posledních letech nabývá na významu díky rozšíření digitálních techniky a s tím spojené nasazení v nejrůznějších odvětvích průmyslu, medicíny, počítačové grafiky i zábavy. Tato práce se zabývá přehledem metod užívaných při hranové detekci, způsoby reprezentace hran a praktickým využitím těchto metod v systémech OCR.

Cílem tohoto projektu bude vytvořit v jazyce C jednoduchý OCR systém založený na hranovém detektoru a reprezentaci nalezených kontur pomocí Freemanova řetězcového kódu. Samotné rozpoznávání bude řešeno použitím umělé neuronové sítě.

První kapitola má název Úvod a obsahuje stručné uvedení do problematiky práce, strukturu práce a vymezení projektu. Druhá kapitola teoreticky rozebírá problematiku hranové detekce, uvádí několik příkladů i ukázek použití. Jsou zde vysvětleny principy Cannyho hranového detektoru a Houghovy transformace. Dále se zabývá metodami reprezentace hran a podrobněji popisuje princip řetězcových kódů. Teorie neuronových sítí je vysvětlena v kapitole 3. Čtvrtá kapitola nastiňuje současný stav a užití systémů OCR. V další kapitole je návrh OCR systému založeného na hranové detekci. Podrobnější popis implementace a užitých technologií je obsažen v šesté kapitole. Výsledky práce a postupů pak v kapitole následující. Poslední kapitola je Závěr, zhodnocující výsledky práce a možnosti vývoje projektu do budoucna.

2 Detekce a reprezentace hran v obraze

Tato kapitola je teoretickým úvodem do celé obsáhlé problematiky hranové detekce a reprezentace hran. Postupně jsou v ní vysvětleny pojmy důležité pro pochopení principů užívaných v praxi. Na obrázcích jsou ukázky výsledků některých používaných hranových detektorů i schémata a grafy doplňující výklad.

2.1 Co je to digitální obraz

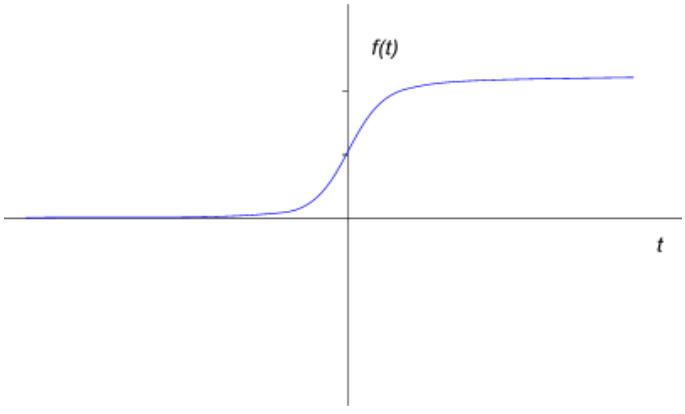
Dříve než se začne obraz zpracovávat, je nutno ho nějakým způsobem převést do digitální podoby (pomocí skeneru, digitálního fotoaparátu, kamery, atd.). Digitalizace obrazu je převod analogového signálu do diskrétního tvaru. Získaný obraz je pak uložen do počítače.

Vstupní signál je popsán funkcí *f(x,y)* dvou proměnných – souřadnic bodu v obraze. Funkční hodnota odpovídá například jasu nebo hodnotám spektrálních složek signálu při barevném snímání. Vstupním signál je kvantován a vzorkován. Výsledkem je matice čísel popisující obraz – *digitální obraz*. Jeden prvek matice představuje jeden obrazový element, nazývaný *pixel*.

Při hranové detekci nás bude zajímat hlavně černobílý obraz – informace o barvě nebude důležitá. Proto je i barevný vstupní obraz na počátku zpracování zbaven barev. K tomu lze použít hned několik technik. Funkční hodnota obrazové funkce bude tedy představovat hodnotu jasu. Bude proto nazývána jasová funkce nebo funkce intenzity.

2.2 Detekce hran

Detekce hran je postup v digitálním zpracování obrazu, sloužící k nalezení oblastí bodů, ve kterých se podstatně mění jas (viz obr. 2.1). Slouží k nalezení takových míst obrazu, které lidské oko vnímá jako hranice objektů nebo různých tvarů, např. rozhraní světla a stínu. K provedení hranové detekce se používají hranové detektory (*edge detector*).

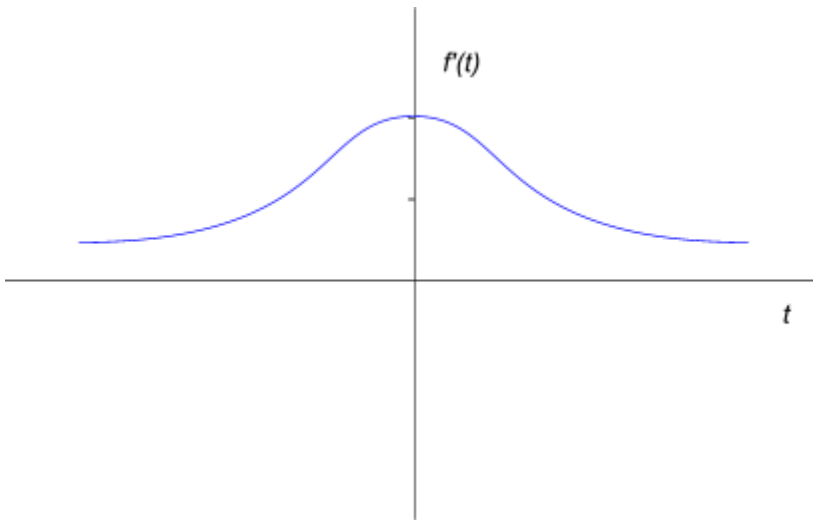


[obr. 2.1] Jednorozměrná funkce jasu v obraze – znázornění hrany

Hranových detektorů existuje celá řada, různící se jak v metodách nalezení hrany, její reprezentaci, tak v citlivosti rozpoznání hrany a s ní související odolnosti proti šumu a jiným nežádoucím efektům zpracovávaného obrazu.

V základu se hranové detektory dají rozdělit na detektory využívající *první derivaci* nebo *druhou derivaci* jasové funkce. Při použití první derivace je výsledný hranový gradient porovnán s prahem, který určuje, jedná-li se o hrany či nikoli. U metod druhé derivace je výskyt hrany detekován, je-li prostorová změna v polaritě druhé derivace dostatečně významná.

Pokud je hrana definována jako relativně velká změna hodnoty jasové funkce, bude v místě hrany velká hodnota derivace této funkce. Hodnota této derivace (gradientu) bude maximální v kolmém směru ku hraně. V praxi se však detekce provádí jen ve dvou, resp. ve čtyřech směrech. K výpočtu gradientu můžeme přistupovat jako ke konvolučnímu filtrování obrazu. Jednotlivé hranové detektory se pak liší jádrem konvolučního filtru. Jádro filtru - matice užitá ke konvoluci - udává, které body se pro výpočet gradientu použijí a jaké budou mít váhy. Velikost a vlastnosti filtru významně ovlivňují výsledek detekce. Obecně platí, že čím je matice větší, tím je detektor odolnější proti šumu, protože okolí použité k aproximaci je větší. Konvolucí mezi jádrem filtru a zpracovávaným obrazem získáme tzv. *gradientní obraz*. Konvoluce se provádí pro detekci v jednom směru (vodorovně nebo svisle) a pro druhý směr se použije matice transponovaná.



[obr. 2.2] Derivace funkce intenzity

Následují ukázky některých významných filtrů:

	G_x	G_y
• Robertsův	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
• Prewittové	$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
• Sobelův	$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
• Robinsonův	$\begin{bmatrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
• Kirschův	$\begin{bmatrix} 3 & 3 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & -5 \end{bmatrix}$	$\begin{bmatrix} -5 & -5 & -5 \\ 3 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}$
• Frei-Chenův	$\begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$
• Prewittové(5X5)	$\begin{bmatrix} 2 & 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 & -2 \end{bmatrix}$	$\begin{bmatrix} -2 & -2 & -2 & -2 & -2 \\ -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$

Z výsledků detekce hran v horizontálním a vertikálním směru G_x a G_y lze spočítat sílu hrany:

$$G = \sqrt{G_x^2 + G_y^2} \qquad [1]$$

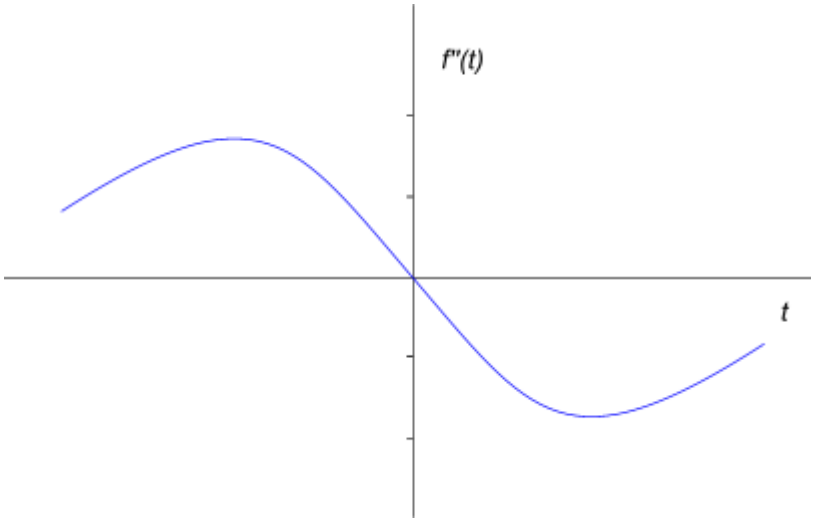
nebo můžeme hodnotu následně aproximovat:

$$|G| = |G_x| + |G_y| \qquad [2]$$

Orientace gradientu vzhledem k vodorovné ose lze vypočíst jako:

$$\theta = \arctan\frac{G_x}{G_y} \qquad [3]$$

Metody založené na druhé derivaci jasové funkce se snaží nacházet průchody této derivace nulou (viz obr. 2.3). Využívají toho, že je snazší nalézt průchod nulou, nežli extrém funkce. Bohužel druhá derivace je ještě citlivější na šum, než první, proto je vhodné její výpočet kombinovat s takovým vyhlazením, které odstraní maximální množství šumu a při tom nepoškodí hrany. Tyto metody často pracují s Laplaceovým operátorem – Laplaciánem, který aproximuje druhou všesměrovou derivaci jasové funkce. K výpočtu se používá tedy pouze jedna matice. K jeho nevýhodám patří, že má dvojité odezvy na hrany odpovídající tenkým liniím v obraze.



[obr. 2.3] Druhá derivace funkce intenzity

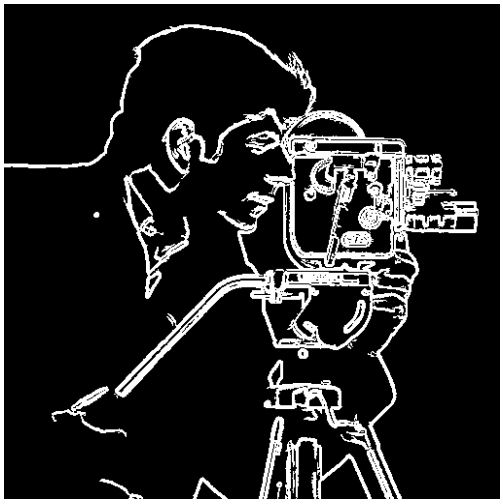
Konvoluční masky mohou vypadat takto:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{bmatrix} \qquad \begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

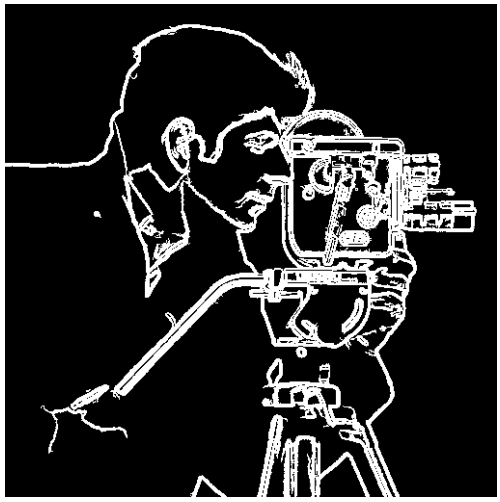
Marrův filtr se snaží odstranit šum použitím Gaussovské funkce (rozmazání) a následnou aplikací Laplaciánu (LoG = *Laplacian of Gaussian*).

Aplikace některých hranových detektorů je vidět na následujících obrázcích:

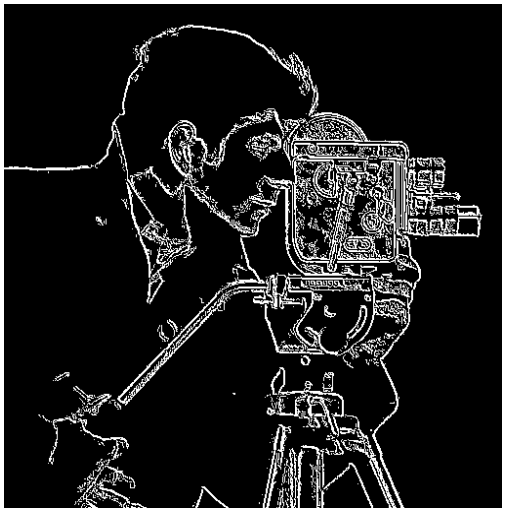
[obr. 2.4 a] Originální obrázek



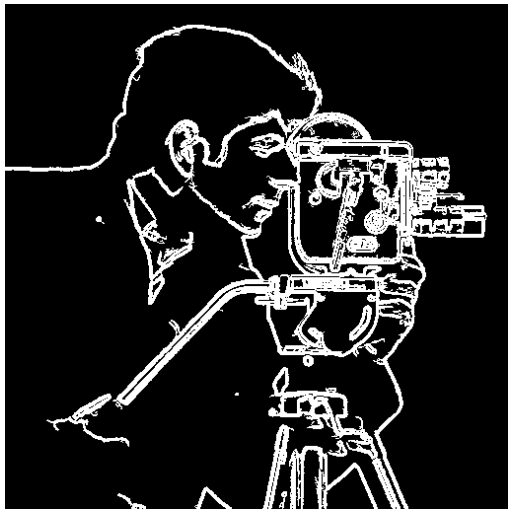
[b] Frei-Chen



[c] Operátor Prewittové



[d] Roberts



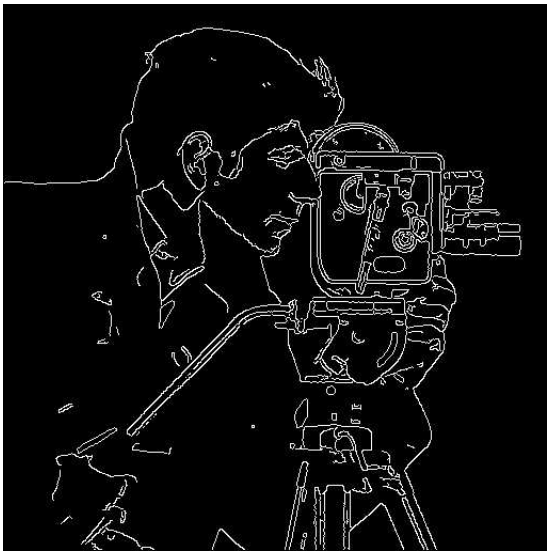
[e] Sobel

2.3 Cannyho hranový detektor

Cannyho algoritmus je obecně znám jako optimální hranový detektor. Je navržen tak, aby splňoval tři základní požadavky, které dříve zmíněné detektory zajistit nedokázaly. Tyto požadavky jsou:

- minimální počet chyb – musí být detekovány všechny hrany a nesmí být žádná odezva na místa, která hranami nejsou
- přesnost – poloha detekované hrany musí být určena co nejpřesněji
- jednoznačnost – každá hrana může být detekována pouze jednou

Postup detekce sestává z několika kroků. Prvním je eliminace šumu Gaussovým filtrem. Gaussův filtr může být realizován pomocí konvoluce užívající masku, jejíž výpočet je relativně snadný. Poté následuje aplikace Sobelova operátoru pro nalezení velikostí gradientů a jejich směrů (viz výše). Následuje proces ztenčení (*thinning*), spočívající ve výběru lokálních maxim nalezených gradientů. Princip této fáze probíhá tak, že jako bod hrany je označen jen takový bod, jehož sousední body v okolí kolmém na směr gradientu (směr gradientu je znám – vrací jej Sobelův detektor) mají hodnotu gradientu nižší. Posledním krokem je prahování s hysterezí (*tresholding*). Jeho účel je v ohodnocení významu nalezených hran – splnění kritéria jediné odezvy. Hrany pocházející ze šumu mají obvykle nižší hodnoty gradientu než hrany skutečně hledané. Cannyho detektor má mezi vstupními parametry hodnoty dvou prahů T_1 a T_2 . Hodnoty nalezených gradientů jsou potom porovnávány s těmito prahy. Pokud je hodnota gradientu vyšší než hodnota prahu T_2 , je označen jako hrana přímo. Pokud je jeho hodnota nižší než T_1, je chápán jako nehranový. A konečně, leží-li jeho hodnota v rozmezí těchto dvou prahů, je bod označen jako hrana pouze v případě, že sousedí s bodem, který už za hranu uznán byl.



[obr. 2.5] Cannyho detektor

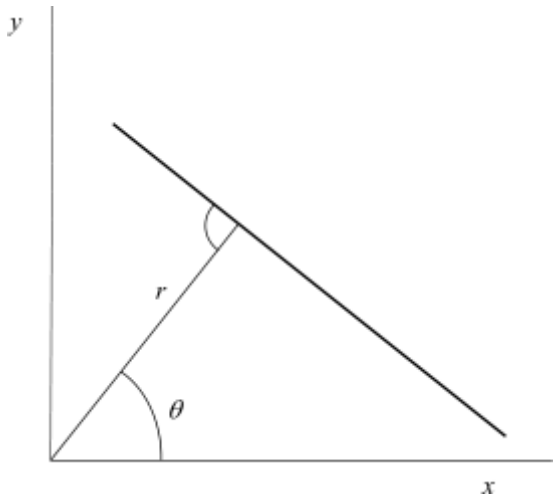
2.4 Houghova transformace

Houghova transformace je metoda pro nalezení parametrického popisu objektů v obraze. Při implementaci je třeba znát analytický popis tvaru hledaného objektu. Proto je tato metoda používána pro detekci jednoduchých objektů v obraze jakou jsou přímky, kružnice, elipsy, atd. Obrovskou výhodou této metody je její robustnost, kdy segmentace není příliš citlivá na porušená data nebo šum. Jde o transformaci z Kartézského souřadnicového systému do polárního. Houghova transformace bude sloužit ke zjištění orientace textu – naklonění řádků. Budeme proto hledat přímky odpovídající jednotlivým řádkům.

Parametrické vyjádření přímky je:

$$r = x \cdot \cos\theta + y \cdot \sin\theta \qquad [4]$$

kde *r* je délka normály od přímky k počátku souřadnic a θ je úhel mezi normálou a osou *x* (viz obr. 2.6).



[obr. 2.6] Parametrický popis přímky

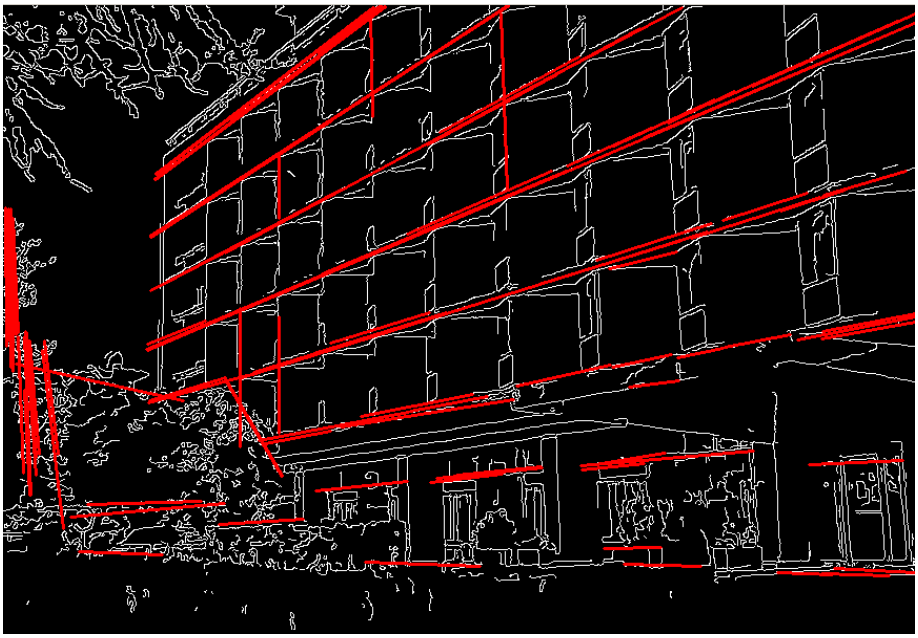
Jednotlivé body vstupního obrazu mají parametry *x* a *y* odpovídající jejich poloze v obraze. Tyto body se dají prezentovat křivkami v parametrickém prostoru. Parametrický prostor je obecně *n*-rozměrný prostor, kde *n* odpovídá počtu hledaných parametrů. V našem případě se jedná o dvourozměrný prostor o parametrech *r* a θ.

Samotná transformace pak probíhá tak, že parametry *x* a *y* každého vstupního bodu obrazu, který představuje bod úsečky nebo přímky, jsou dosazeny do rovnice [4]. Za hodnotu θ je postupně dosazována každá možná hodnota – interval $\langle 0°,360° \rangle$ (matematicky by se jednalo o nekonečnou množinu hodnot reálných čísel představujících úhel θ, ale v případě implementace jsou dosazovány hodnoty podle nastavení citlivosti Houghovy transformace – například po jednom stupni) a hodnota *r*

se dopočítává. Tak v parametrickém prostoru vznikají křivky, pro každý vstupní bod jedna (v případě úseček se jedná o sinusoidy). Pokud jsou vstupní body kolineární (body ležící na jedné přímce), křivky parametrického prostoru se protínají nejčastěji v bodě představujícím parametry hledané přímky.

Příklad aplikace Houghovy transformace v *OpenCV*:

[obr. 2.7 a] Vstupní obrázek



[b] Obrázek po zpracování Cannyho algoritmem a Houghovou transformací

2.5 Reprezentace hran

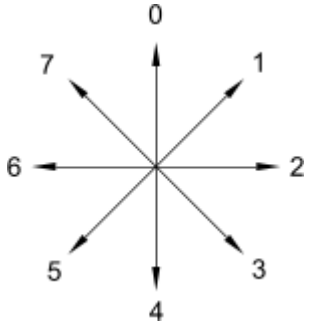
Reprezentace hran řeší otázku, jakým způsobem ukládat a prezentovat data, která nám poskytují hranové detektory. Prosté zobrazení nalezených hran je v řadě případů nedostačující, často je pro další práci s obrazem nutné detekované hrany nějakým způsobem ukládat a zpracovávat pro další práci s nimi.

Předpokládejme, že máme funkční a spolehlivý hranový detektor, který nám dokáže detekovat souvislé kontury objektů v obraze (například Cannyho detektor). Nyní stojíme před otázkou, jak budou tyto kontury uloženy a reprezentovány z hlediska počítačových dat. Máme celou řadu technik, mezi kterými můžeme vybírat. Kritériem pro výběr správného přístupu je v první řadě převažující typ křivek, které chceme reprezentovat. Například při zpracování kartografických údajů, kde očekáváme převážně dlouhé zaoblené křivky, jako jsou například řeky nebo železnice, s minimem ostrých rohů a častých změn směru, je výhodnější reprezentovat tyto křivky pomocí kombinace oblouků a rovných segmentů nebo přímo pomocí křivek – matematických funkcí. Naproti tomu při zpracování rastrových dat ze skeneru nebo digitálních fotografií, kde převládají kontury s množstvím ostrých změn směru, často navíc zatížené šumem, bude výhodnější užít řetězcových kódů.

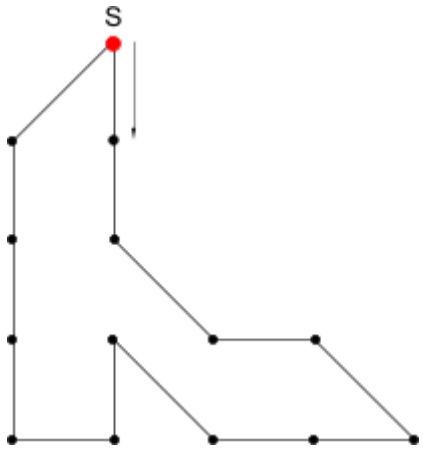
2.6 Řetězcové kódy

Princip řetězcových kódů spočívá v tom, že cestu z jednoho bodu do sousedního bodu popisované kontury lze označit – například číslicí. Pokud uvažujeme, že se pohybujeme na úrovni jednotlivých pixelů, musíme podle zvolené provázanosti (*connectivity*) takto označit 4 resp. 8 směrů.

Freemanův řetězcový kód pracuje s 8-mi směry a s jejich označením číslicemi od 0 do 7. Tyto směry si můžeme představit jako možnosti tahu s dámou na šachovnici (viz obr. 2.8). Potom řetězcový kód kontury z obrázku 2.9 s počátkem v bodě S bude vypadat takto {4,4,3,2,3,6,6,7,4,6,0,0,0,1}.



[obr. 2.8] Označení směrů Freemanova kódu



[obr. 2.9] Příklad popisované kontury

3 Neuronové sítě

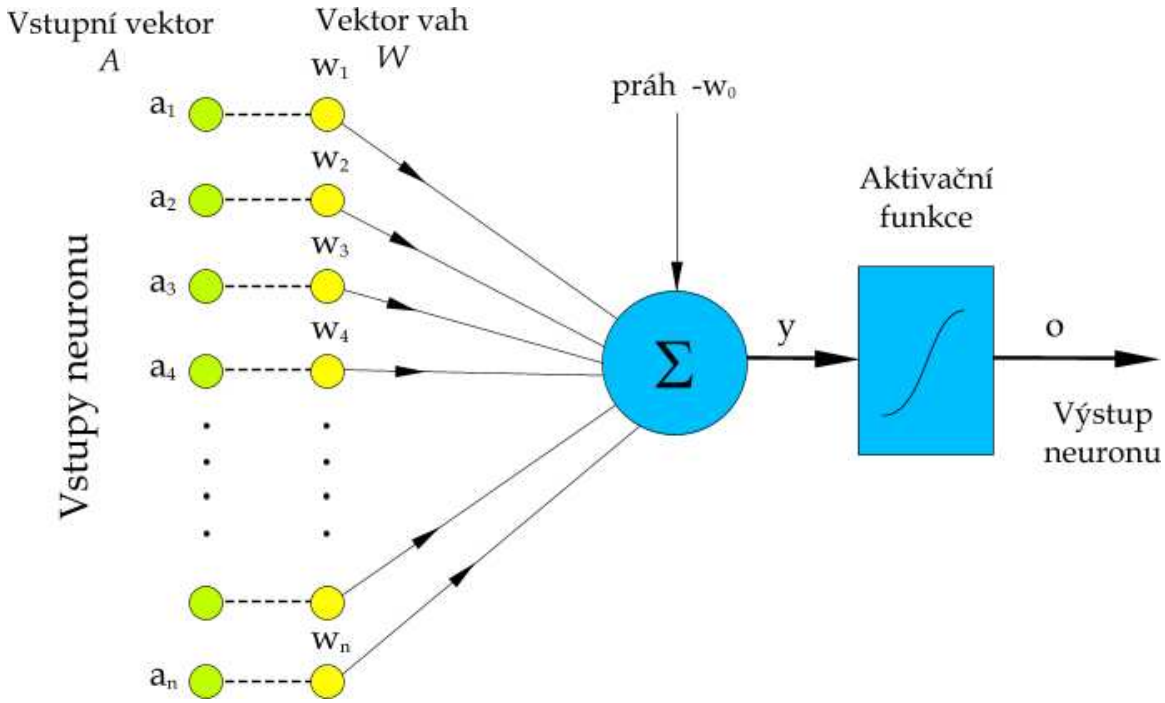
Umělá neuronová síť je v projektu použita k vlastnímu rozpoznání znaku podle histogramu Freemanova řetězcového kódu.

Umělá neuronová síť napodobuje chování biologické neuronové sítě, ovšem v menším měřítku. Lidský mozek například disponuje množstvím přibližně 100 miliard propojených neuronů. Umělá neuronová síť takových měřítek by byla nemyslitelná. Narozdíl od lidského mozku umělé neuronové sítě nemají za úkol řídit chování člověka, jeho učení a rozhodování se. Slouží k řešení problémů užšího zaměření, jako například rozpoznávání písma, geometrických tvarů, předpovídání vývoje burzy, rozpoznávání hlasu, filtrace a dalších.

3.1 Neuron

Umělý neuron pracuje na stejném principu jako neuron biologický. Skládá se ze tří částí: tělo, axon (výstup) a množství vstupů (dendrity). Výstup neuronu je propojen se vstupy jiných neuronů pomocí synapsí. Tím se výstupní signál jednoho neuronu přivádí na vstup jiného. V lidském mozku je neuron propojen s přibližně 10 000 až 100 000 jinými neurony.

Přenášené signály mohou působit jako budící (excitátory) nebo tlumící (inhibitory). Neuron reaguje na součet signálů na svých vstupech. Aktivace neuronu nastane, překročí-li součet excitátorů hodnotu součtu inhibitorů o určitou hodnotu (práh).



[obr. 3.1] Matematický model neuronu

Protože sílu signálů lze reprezentovat reálnými čísly, je možné popsat neuronovou síť matematicky. Chování neuronu můžeme popsat jako násobení vektorů. Vstupní vektor neuronu (dendrity) označme jako vektor $A = (a_1, a_2, a_3, a_4, ..., a_n,)$. Vektor vah představující buď buzení nebo tlumení signálu označíme jako vektor $W = (w_1, w_2, w_3, w_4, ..., w_n,)$. Vynásobením těchto dvou vektorů a odečtením hodnoty prahu od výsledku získáme tzv. vnitřní potenciál neuronu. Tento potenciál poté projde aktivační funkcí, čímž získáme výstup neuronu. Učení neuronu potom spočívá v upravování hodnot vah tak, aby výstup neuronu byl co nejblíže požadovanému učícímu vzoru.

Vnitřní potenciál neuronu:

$$y = \sum_{i=1}^{n} a_i \cdot w_i - w_0 \qquad [5]$$

Výstup neuronu:

$$o = f(y) \qquad [6]$$

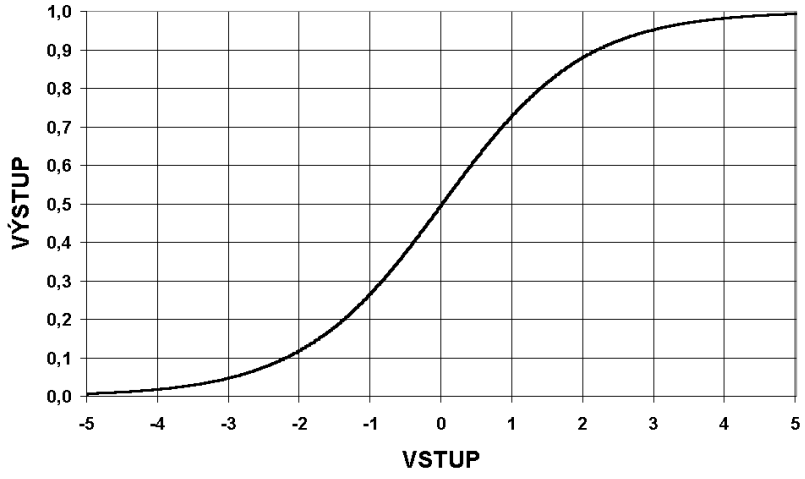
kde *f* je aktivační funkce neuronu. Nejběžnější aktivační funkcí bývá sigmoidální funkce ve tvaru

$$f(x) = \frac{1}{1+e^{-x}} \qquad [7]$$

Výstup neuronu má tedy konečnou podobu ve vzorci

$$o = \frac{1}{1+e^{-\left(\sum_{i=1}^{n} a_i \cdot w_i - w_0\right)}} \qquad [8]$$

Sigmoidální funkce vrací hodnoty v intervalu $\langle 0;1 \rangle$ a jako aktivační funkce je používána mimo jiné proto, že je diferencovatelná, což je požadavek u neuronových sítí s backpropagation učícím algoritmem. Její průběh je patrný z obrázku 3.2.



[obr. 3.2] Průběh sigmoidální funkce

3.2 Neuronová síť

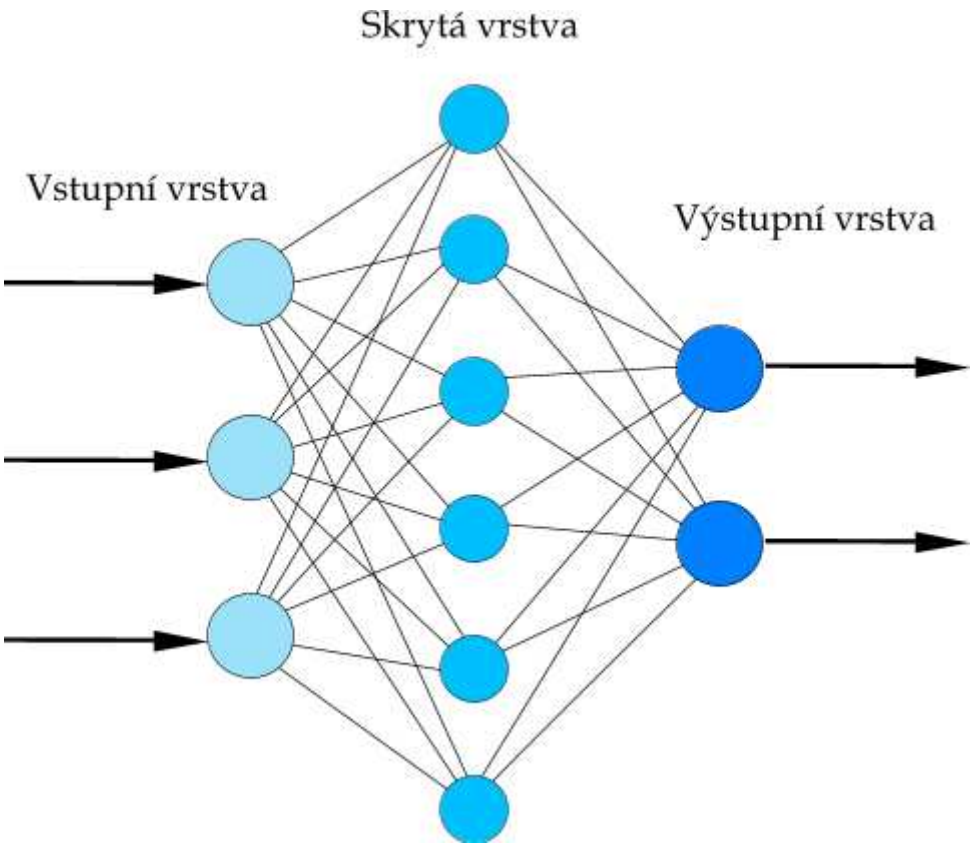
Jedním z nejvýznamnějších rysů neuronové sítě je její schopnost učit se a to jak z charakteru samotných dat, tak i z vlastních chyb při snaze zachytit vztah mezi nezávisle i závisle proměnnými. V praxi to vypadá tak, že když biologická neuronová síť dostane na vstupy signály určité síly, které na výstupu způsobí určitou reakci (například když sáhneme do ohně → reakce bude, že to pálí), nastaví se váhy na vstupech příslušných neuronů (v těch, kterými se signál šířil) a příště už budeme vědět, že oheň pálí i bez toho, abychom na něj museli sahat. Neuronová síť se naučila.

Učení umělé neuronové sítě spočívá v jejím naučení správné reakce na vstupy. To probíhá tak, že síti je předložen učící vzorek obsahující vstupy a jim odpovídající správné výstupy. Síť se snaží nastavit příslušné váhy vstupů neuronů (jako u biologické sítě) tak, aby její výstup byl co nejblíže požadované hodnotě. To se děje pomocí učícího algoritmu, jehož výběr je na nás. Příklad takového algoritmu je hledání řešení ve směru největšího gradientu – změní se hodnota některé váhy a zjistí se, zda se výstup přiblížil nebo oddálil požadované hodnotě. Logicky pak změna hodnoty váhy pokračuje ve stejném, resp. opačném směru. Nevýhodou je možnost uváznutí v nějakém lokálním minimu. Učící algoritmus pak nastaví náhodnou (lépe řečeno úplně jinou) hodnotu váhy a „doufá“, že se z uváznutí vyprostil.

Rychlost i správnost učení neuronové sítě závisí hlavně na její architektuře. Ta spočívá v navržení počtu vrstev a počtu neuronů v každé vrstvě sítě. Bylo dokázáno, že síť, která je tvořena nejméně třemi vrstvami neuronů, je schopna aproximovat libovolnou matematickou funkci, přičemž přesnost této aproximace je tím větší, čím více neuronů síť obsahuje. Pokud má ovšem síť příliš mnoho neuronů ve skryté vrstvě, učí se pomalu a také se může stát, že se bude učit příliš specifické a podrobné vztahy mezi vstupními a výstupními daty, které vedou ke nesprávným výsledkům. Nicméně neexistuje univerzální způsob jak určit, kolik neuronů ve skryté vrstvě by pro řešení určitého problému síť měla mít. Počet neuronů ve vstupní a výstupní vrstvě sítě je dán přímo návrhem řešení problému: pokud se má síť naučit funkci XOR, bude mít ve vstupní vrstvě dva neurony pro dva vstupy funkce a jeden neuron výstupní pro výstup funkce. Neuronová síť organizovaná do vrstev, kde každý neuron je spojen s každým neuronem s předchozí i následující vrstvy, se nazývá síť s dopředným šířením (*feedforward neural network*). Taková síť má konečnou délku odezvy a neumí se učit.

Sítě se zpětnou vazbou (*feedback neural network*), ve kterých se signál může šířit stále dokola, zvětšovat nebo zmenšovat se, se učit umí a mají paměť.

V projektu je použita umělá neuronová síť knihovny *FANN*, o třech vrstvách, s 260 neurony ve skryté vrstvě, 8 neurony ve vstupní a 26 ve výstupní vrstvě. Učící algoritmus je standardní *back-propagation* a požadovaná maximální chyba je 0,001. Aktivační funkce je tansigmoidální, její obor hodnot je $\langle -1;1 \rangle$.



[obr. 3.3] Model neuronové sítě s jednou skrytou vrstvou

4 OCR – Optical Character Recognition

4.1 Současný stav

Praktické užití výše zmíněných metod je úzce spjato s aplikacemi zpracování obrazu a počítačového vidění. Své uplatnění nachází v průmyslových aplikacích, zpracování medicínských dat i úpravě digitálních fotografií a videa.

Co se využití hranových detektorů a následné zpracovávání nalezených hran týče, mají svoje využití například v programech pro rozpoznání textu (OCR), detektorech objektů, detekce lidského obličeje a dalších aplikacích počítačového vidění.

4.2 Aplikace

Optical Charakter Recognition – optické rozpoznání textu je odvětví počítačového zpracování obrazu, které se zabývá rozpoznáváním tištěného nebo ručně psaného textu a jeho převodem do textu digitálního, např. ASCII znaky.

Jako vstup pro OCR programy slouží obraz s textem (například digitální fotografie nebo výstup skeneru). Výstupem bývá text, který je počítačem snadno editovatelný a dále použitelný. Výhody takto zpracovaného textu jsou nasnadě: rychlejší vyhledávání (nerozpoznaný text je nutno prohledávat ručně - čtenářem), možnost čtení textu pomocí software, kontrola a oprava chyb a překlepů, sdílení pomocí internetu a podobně.

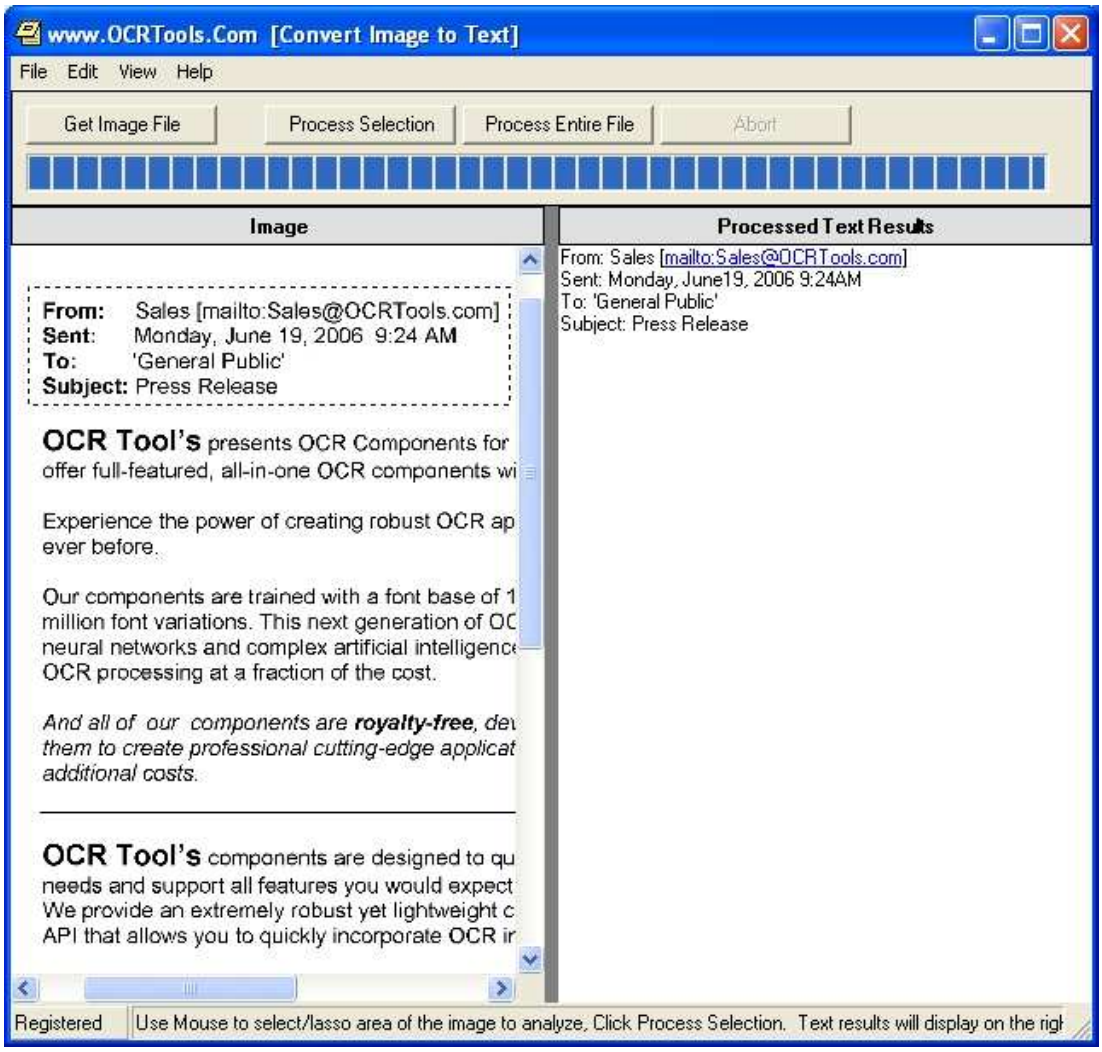
Historie systémů OCR sahá hluboko do historie samotných počítačů, do 50. let 20. století. Vývoj těchto systémů lze rozdělit do tří generací, podle jejich vlastností a schopností rozpoznání různých druhů textu.

V současnosti je na trhu dostupná spousta komerčních i nekomerčních OCR systémů. Komerční produkty většinou přesahují kvalitu zdarma dostupných produktů, a krom samotného rozpoznání textu umí třeba číst čárové kódy, rozpoznávat jazyk textu, zachovávat formátování textu, exportovat získaný text do populárních formátů pro editaci textu nebo jejich další šíření.

Kromě zmíněného využití, dají se OCR systémy v různých podobách využívat také v dalších odvětvích. Kupříkladu ze záběrů průmyslových kamer snímající vozidla na silnicích a křižovatkách jsou schopny rozpoznat SPZ automobilu, jehož řidič porušil pravidla provozu, a automaticky generovat pokuty nebo hlásit tuto skutečnost na příslušné místo. Digitální fotoaparáty s vyšším rozlišením zabudované v mobilních telefonech mohou analyzovat vyfocený nápis a poté jej uživateli přečíst nebo přeložit, případně dále použít jako prostý text.

Detektory objektů slouží spíše než k detekci obecných objektů ke specifickým účelům. Detekce lidského obličeje potažmo směru pohledu v obraze nebo ve videosekvenci slouží mnoha různým

účelům. Kamera uvnitř automobilu snímající řidiče předává obraz na zpracování a systém může kontrolovat pozornost řidiče, směr jeho pohledu a případně upozorňovat na nezpozorovaná nebezpečí. Úplně jinou aplikací jsou potom systémy pro automatický střih videa, které se podle pohledu a grimas ve tvářích snímaných osob snaží vybírat záběry kde se „něco děje“. Dalším využití představují systémy pro rozpoznávání specifických geometrických tvarů, například dopravních značek, čárových kódů, atd. Aplikace v medicíně tvoří celou samostatnou kapitolu se systémy analyzujícími lékařská data nejrůznější povahy – od detekce rakovinového nádoru až po analýzu rentgenových snímků.



[Obr. 4.1] – Ukázka volně dostupného OCR programu

OCR Image to Ascii Desktop Converter 1.0.0

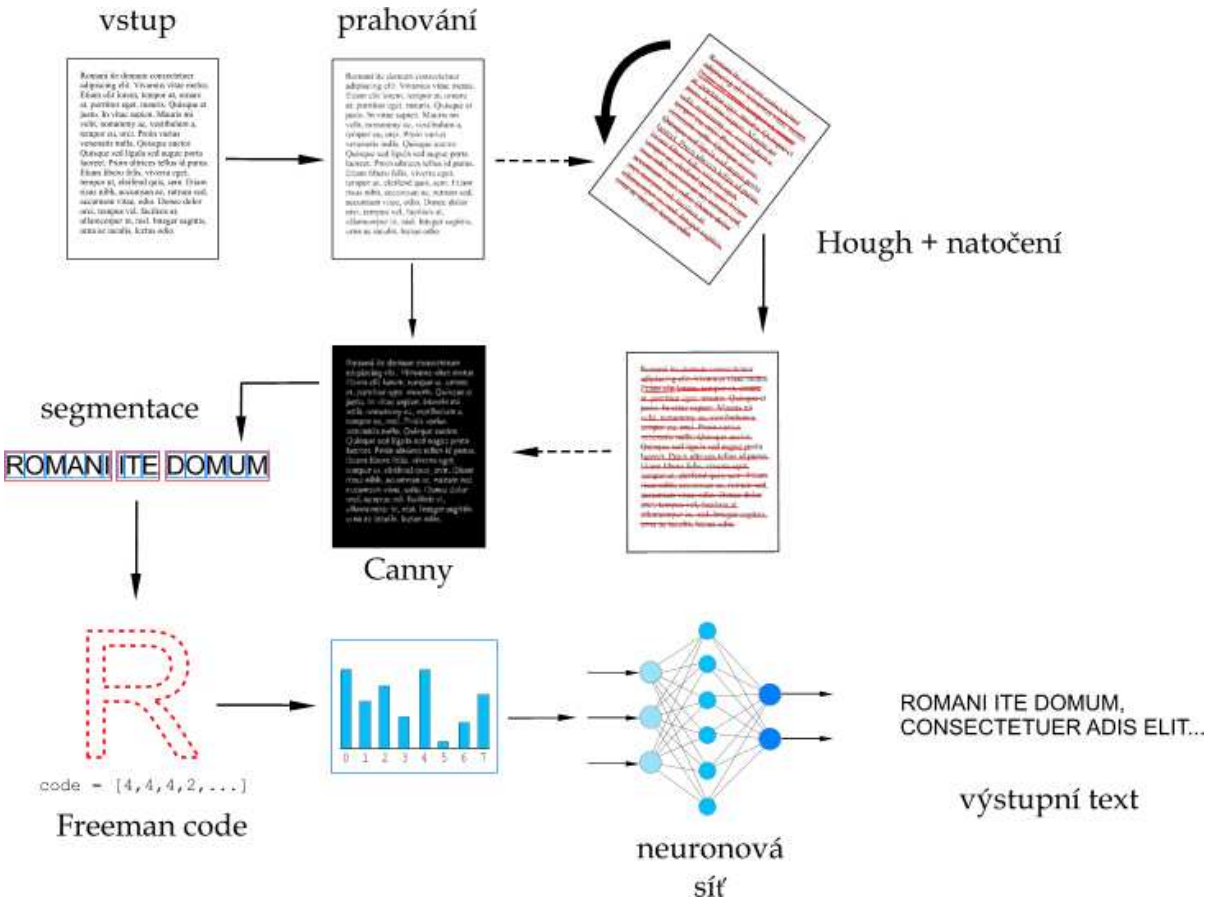
5 Návrh řešení

Výsledná aplikace bude na vstupu požadovat obrázek s textem v dostatečném rozlišení a velikostí rozpoznávaných znaků. Výstupem potom bude text v ASCII.

Aplikace bude pracovat v několika krocích:

- načtení vstupního obrázku obsahujícího text k rozpoznání
- prahování – převede vstupní obrázek na binární a provede segmentaci obrazu
- použití Houghovy transformace pro zjištění orientace textu – případné natočení textu do vodorovné polohy
- hranová detekce pomocí Cannyho algoritmu
- segmentace jednotlivých řádků, slov a znaků
- zpracování Freemanova řetězcového kódu
- vyhodnocení neuronovou sítí
- výstupní text

K realizaci aplikace jsem se rozhodl využít volně dostupných knihoven *OpenCV* a *FANN*. Blokové schéma na obrázku 5.1 znázorňuje princip práce programu.



[obr. 5.1] Blokové schéma návrhu OCR systému

Implementační detaily jednotlivých kroků jsou popsány dále v kapitole 6. Nyní bude následovat hrubý náčrt principu těchto kroků.

Načtení obrazu je nutné před každým dalším zpracováním obrazu. Obraz je načten většinou podle jména souboru obsahujícího obraz.

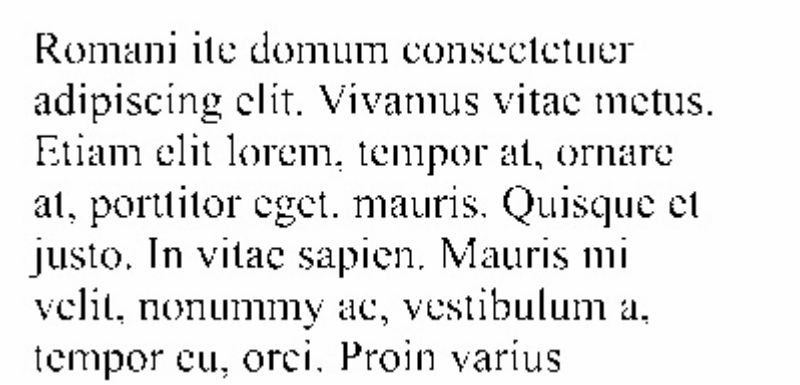
Prahování zbaví obraz barevné informace, která pro další kroky zbytečná, a vstupní obraz převede na binární – to znamená takový, že jednotlivé body nesou pouze informaci 1 nebo 0 (většinou reprezentováno jako bílá a černá). Prahování funguje na jednoduchém principu, kdy hodnota jasu jednotlivých bodů je srovnávána se zvolenou hodnotou prahu. V případě, že hodnota jasu je menší než hodnota prahu je tento bod označen jako 0 (černý), v opačném případě jako 1 (bílý). Pokud je vstupní obraz barevný, provede se nejdříve jeho převedení pomocí rovnice $y = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B$, kde R,G,B jsou jednotlivé barevné složky (červená, zelená, modrá) a y je výsledná hodnota jasu. Prahování slouží k segmentaci obrazu; rozděluje obraz na části, které jsou text a které jsou pozadí (viz obr. 5.2).

Dále je pomocí Houghovy transformace zjišťováno natočení textu – pokud vstupní text není vodorovný, je natočen o potřebný úhel, tak, aby vodorovný byl. Houghova transformace nalézá úsečky tvořené řádky textu (viz obr. 5.3), z jejichž náklonu je úhel vypočítáván. Řetězcové kódy z nakloněných znaků by se lišily od kódů znaků vodorovných a nedošlo by ke správnému rozpoznání znaku.

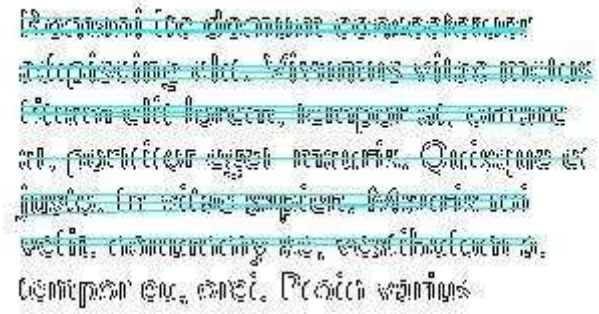
Na vodorovný text je poté aplikována hranová detekce – Cannyho hranový detektor. Jeho výstupem je seznam nalezených kontur. Uspořádání tohoto seznamu je popsány v kapitole 6.5. Toto uspořádání úzce souvisí se segmentací textu na řádky, slova a znaky. Nejde jen o to vstupní znak správně rozpoznat, ale také zařadit na správné místo ve výstupním textu. Jednotlivé znaky jsou reprezentovány konturami z Cannyho detektoru (jedna kontura představuje jeden znak – viz obr. 5.4). Tyto kontury je nutné číst ve správném pořadí a dávat je na vstup neuronové síti. Nebo je možnost seznam kontur správně přeuspořádat, aby jeho sekvenční čtení odpovídalo přirozenému čtení textu. Problém segmentace i jeho řešení jsou detailně popsány v kapitole 6.5.

Zpracování řetězcového kódu pracuje na úrovni znaku – nestará se o jeho umístění v textu. Kontura nalezená detektorem je zpracován do Freemanova řetězcového kódu podle postupu vysvětleného v kapitole 2. Spočítáním výskytů jednotlivých směrů získáme histogram Freemaenova kódu a vydělením položek histogramu celkovým počtem kroků v kontuře normalizovaný histogram. Je představován osmi čísly z intervalu $\langle 0{,}1 \rangle$. Normalizovaný histogram je předán jako vstup neuronové síti, která se podle jeho hodnot pokouší rozpoznat znak. Výstupem sítě je pole 26-ti čísel typu *float*. Index pole, na kterém se objeví nejvyšší hodnota odpovídá indexu (pořadí) rozpoznaného znaku v anglické abecedě.

Výstupní text je tedy získán tak, že správně uspořádaný seznam kontur je sekvenčně čten od počátku do konce, každá přečtená kontura je zpracována, předána síti a při úspěšném rozpoznání vypsána jako znak.



[Obr. 5.2] Text po použití prahování s vysokým prahem – došlo ke ztenčení znaků. Takto vysoký práh není vhodný



[Obr. 5.3] Úsečky nalezené Houghovou transformací (obrázek byl invertován)

[Obr. 5.4] Kontury nalezené Cannyho hranovým detektorem (obrázek byl invertován)

6 Implementace

V této kapitole je podrobněji popsána vlastní implementace programu. Budou zde vysvětleny použité postupy i algoritmy.

Celá aplikace se skládá ze tří programů, každý z nich má svůj zdrojový soubor. Program *sampler* slouží k vytváření trénovacích souborů pro neuronovou síť, jejíž trénování probíhá za pomoci programu *training*. Oba tyto programy nemusí koncového uživatele prakticky zajímat; ten bude používat hlavně program *ocr*, který slouží k vlastnímu rozpoznání textu z obrázku, který uživatel zadá jako vstupní parametr při spuštění programu.

6.1 Knihovna OpenCV

OpenCV (Open Computer Vision library) je volně dostupná grafická knihovna původně vyvíjená společností Intel. Má bohatou zásobu funkcí zaměřených převážně na zpracování obrazu, ale i tvorbu uživatelského rozhraní, práci s grafickými formáty a mnoho dalších.

V projektu je využívána právě kvůli funkcím zpracování obrazu. Konkrétně Cannyho hranový detektor, Houghova transformace, prahování, rotování obrazu a práce s vstupním formátem obrazu.

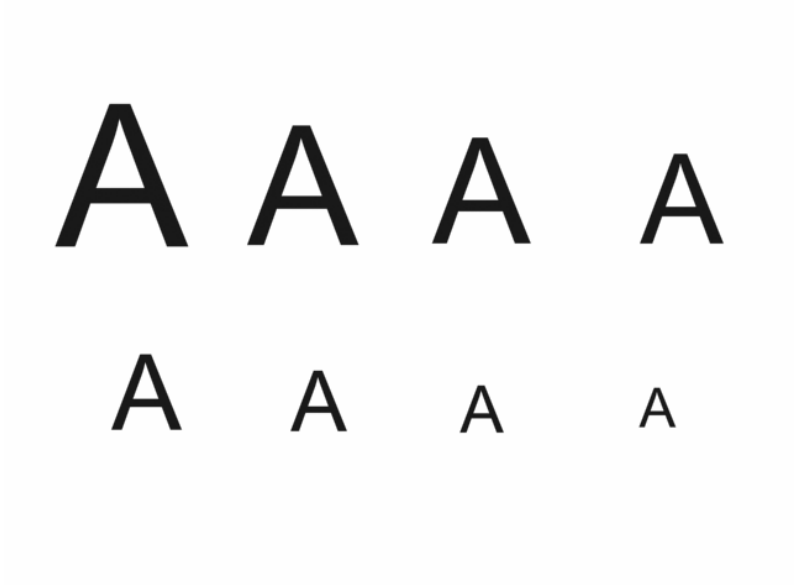
6.2 Knihovna FANN

Tato knihovna je použita pro vytvoření a natrénování umělé neuronové sítě. *FANN* (Fast Artificial Neural Net) je také volně dostupná *open source* knihovna implementující vícevrstvé umělé neuronové sítě jak plně, tak částečně propojené. Knihovna je jednoduchá k používání a dobře dokumentovaná.

6.3 Program *sampler*

Program s*ampler* slouží, jak již bylo řečeno, k vytváření trénovacích souborů pro neuronovou síť. Jeho užití vypadá tak, že uživatel při spuštění zadá jméno souboru s obrázkem a „jméno“ znaku, který je trénován.

Program pracuje na stejném principu jako *ocr*, co se týče užití Cannyho hranového detektoru a práce s Freemanovým řetězcovým kódem. Po nalezení kontur a jejich popisu, vytvoří se histogram Freemanova kódu (podrobněji popsán níže) a zapíše se do trénovacího souboru. Příklad vstupního obrázku je na obrázku 6.1.



[obr. 6.1] Příklad obrázku pro trénování písmene „A“

Jako trénovaní obrázky je doporučeno používat formáty bez komprese. Ta způsobuje často rozmazání kontur a jejich nepřesnou detekci a výsledky programu jsou horší. U trénovacích obrázků je důležité, aby nalezené kontury byly co nejpřesnější s minimem šumu.

Trénovaní soubor je psán podle požadavků knihovny *FANN*, jež určuje jejich přesný formát takto:

```
x y z
vstup11 vstup12 … vstup1y
výstup11 výstup12 … výstup1z
.
.
.
vstupx1 vstupx2 … vstupxy
výstupx1 výstupx1 … výstupxz
```

kde *x* je počet trénovacích vzorků, *y* počet vstupů a *z* počet výstupů neuronové sítě. Potom trénovací soubor vytvořený z obrázku pro trénování znaku „A“ vypadá takto:

```
8 8 26
0.224299 0.056075 0.168224 0.046729 0.233645 0.112150 0.037383 0.112150
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0.220472 0.039370 0.196850 0.039370 0.228346 0.118110 0.023622 0.125984
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0.228916 0.036145 0.192771 0.042169 0.228916 0.114458 0.030120 0.120482
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

```
0.225962 0.033654 0.201923 0.033654 0.235577 0.110577 0.033654 0.120192
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0.225806 0.036290 0.197581 0.036290 0.233871 0.108871 0.040323 0.116935
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0.226027 0.037671 0.195205 0.037671 0.232877 0.109589 0.041096 0.116438
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0.227273 0.042424 0.184848 0.042424 0.233333 0.109091 0.042424 0.115152
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
0.228426 0.035533 0.195431 0.038071 0.233503 0.111675 0.035533 0.119289
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

hodnota 1 v řádku, kde jsou samé -1 znamená, že chceme, aby neuronová síť při rozpoznání tohoto znaku měla na výstupu na prvním místě hodnotu 1, což značí písmeno A, a na ostatních hodnotu -1. Trénovací soubor pro písmeno B má jedničku na výstupu až na druhém místě, C na třetím atd. Program vytvoří trénovaní soubor a pojmenuje jej `ocr_char_jménoznaku.data.`

6.4 Program *training*

Tento program potřebuje ke své správné funkčnosti soubor `ocr_english.data` (jeho obsah je možno prohlížet na přiloženém CD), jenž tvoří hlavní trénovací soubor pro neuronovou síť. Parametry programu, ovlivňující průběh trénování, jsou tyto:

- požadovaná maximální chyba
- počet neuronů ve skryté vrstvě
- trénovací algoritmus
- aktivační funkce skryté a výstupní vsrtvy
- maximální počet epoch (trénovacích cyklů)
- počet epoch mezi výpisy

Požadovaná maximální chyba určuje, jaký maximální rozdíl mezi výstupem sítě a požadovaným výstupem je povolen. Je to zásadní parametr, který ovlivňuje délku trénování sítě. Při velké chybě je sice trénování velmi rychlé, ale výsledky použití sítě jsou pochopitelně zatížené větši chybou. V projektu je chyba nastavena na hodnotu 0,001.

Počet neuronů ve skryté vrstvě je snad nejvýznamnějším parametrem sítě. Závisí na něm jak průběh, tak délka trénování, ale i výsledné chování celé sítě. Při malém počtu neuronů ve skryté vrstvě není síť schopna se naučit složitější vztahy mezi předkládanými vzorky a požadovanými výstupy – je zkrátka hloupá. Příliš vysoký počet zase vede k prodloužení doby trénování a také možnému přetrénování sítě.

Volba trénovacího algoritmu závisí na konkrétním problému. V projektu síť pracuje se standardním *back-propagation* algoritmem, jehož výsledky byly uspokojivé. Podrobnější rozbor problému těchto algoritmů přesahuje rámec této práce. Rovněž volba aktivační funkce nebude podrobněji rozebírána.

Maximální počet epoch určuje, kolik trénovacích cyklů maximálně proběhne při učení sítě. Může se stát, že síť nebude schopna dosáhnout požadované maximální chyby a trénování skončí neúspěšně po proběhnutí určeného počtu cyklů.

Po spuštění trénování je možno sledovat jeho průběh na standardním výstupu, kde jsou zobrazovány informace o počtu trénovacích cyklů a aktuální chybě trénování, vypisované podle určeného počtu epoch mezi výpisy (viz příloha č.2).

Po úspěšném natrénování sítě je tato uložena do souboru `ocr_trained_float.net` (k nahlédnutí na přiloženém CD).

6.5 Program *ocr*

Toto je hlavní program celé aplikace. Pokud je k dispozici správně natrénovaná neuronová síť, není potřeba oba ostatní programy používat.

Program pracuje ve několika fázích:

- Houghova transformace pro zjištění orientace řádků
- Hranová detekce pomocí Cannyho hranového detektoru
- Segmentace znaků
- Zpracování kontur a Freemanova řetězcového kódu
- Rozpoznání znaku

Program má jediný vstupní parametr a to soubor obsahující obrázek s textem. Stejně jako u trénování sítě platí, že obrázky s minimem šumu mají mnohem lepší výsledky.

Zjištění orientace řádků

Teorie Houghovy transformace je popsána podrobně v kapitole 2. V programu je použita ke zjištění orientace řádků textu, tedy natočení vstupního obrázku. Houghova transformace je realizovány pomocí funkce *cvHoughLines2* knihovny *OpenCV*. Funkce ukládá nalezené úsečky do sekvence *lines*.

Hranová detekce by u šikmého textu proběhla bez problémů, ale Freemanův kód by vypadal jinak a rozpoznání textu by neproběhlo správně. Pomocí Houghovy transformace jsou tedy vyhledány úsečky podle jednotlivých řádků a z váženého průměru takto nalezených úseček je vypočítán úhel naklonění. Váženého průměru bylo použito proto, že ne všechny úsečky jsou stejně dlouhé a mohou

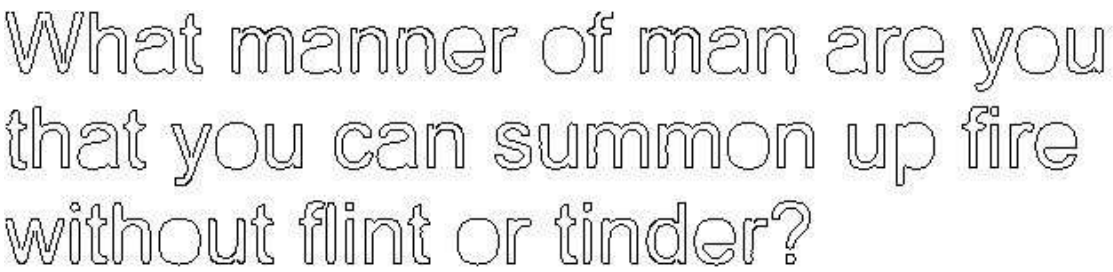
být detekovány úsečky, které vůbec sklonu řádku neodpovídají. Váhou v tomto průměru je délka úsečky.

Pokud je zjištěný úhel v rozmezí -3° až 3° (výsledky u vodorovného textu se pohybují zhruba v tomto rozmezí), neděje se nic. Při přesáhnutí této meze je celý obrázek rotován o příslušný počet stupňů. Rotace je dosaženo použitím funkce *cvGetQuadrangleSubPix*.

Hranová detekce

Program používá k provedení hranové detekce Cannyho hranový detektor pomocí funkce *cvCanny* z knihovny *OpenCV*. Prahy Cannyho detektoru jsou nastaveny na 120 a 230, jejichž hodnota byla zjištěna experimentálně.

Poté je volána funkce *cvFindContours*. Funkce prochází postupně celý obrázek a nalezené kontury ukládá do sekvence *contour*. Jedná se o obousměrně vázaný lineární seznam, v němž jsou uloženy mimo jiné všechny body kontury a několik dalších příznaků. Příklad nalezení kontur detektorem je vidět na obrázku 6.2.



[obr. 6.2] Příklad detekce kontur písma (obrázek byl negován)

Vstupní obrázek byl formátu JPEG. Ačkoli nebyl znehodnocen šumem, na obrázku jde vidět znepřesnění některých kontur kompresí. Problém také vzniká pokud jsou nalezené kontury přerušované. Tento jev lze dobře sledovat u kontury písmene „W“ hned na začátku textu. Problém spočívá v tom, že takto přerušené kontury jsou brány jako samostatné znaky – program předpokládá, že každá kontura představuje jeden znak. Tento problém je řešen v segmentaci textu.

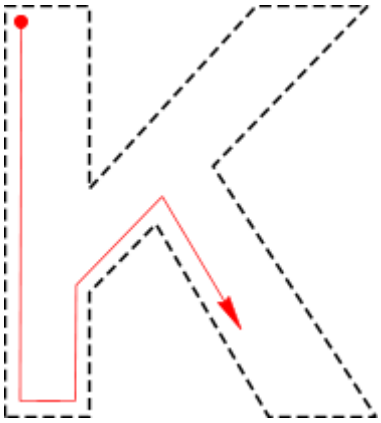
Segmentace textu

V této fázi se řeší problémy a komplikace na první pohled neočekávané. Jsou to především problémy spojené s správným rozdělením textu na řádky, slova a znaky.

Segmentace znaků je provedena již v předchozí fázi, když funkce *cvFindContours* uloží jednotlivé kontury do seznamu *contours*. Nyní vyvstává otázka, jak roztřídit jednotlivé kontury do slov a ta do řádků. Funkce *cvFindContours* pracuje systematicky a jednotlivé kontury ukládá podle jejich počátečního bodu a to tak, že začíná s konturou (s počátečním bodem) s nejvyššími souřadnicemi – tedy vpravo dole a pokračuje snižováním souřadnice *x*, tedy po řádku zprava doleva.

Poté přeskočí o řádek výš a znovu. Prakticky čte text opačným směrem než jsme zvyklí. Navíc dělá i normalizaci kontur tak, že startovní bod je vždy takový, který má nejnižší obě souřadnice, tedy levý vrchní bod. Pro přehlednost je vše znázorněno na obrázcích 6.3 a 6.4.

[obr. 6.3] Schéma ukládání kontur do seznamu



[obr. 6.4] Způsob čtení řetězcového kódu a umístění počátečního bodu kontury

Teoreticky už zbývá jen číst postupně počáteční body kontur a podle rozdílů jejich souřadnic je třídit do slov a řádků. Praxe je bohužel složitější. Jeden problém byl již zmíněn v popisu předchozí fáze a to přerušené kontury. Další problém spočívá v tom, že pokud jsou počáteční body kontur znaků na jednom řádku, mají teoreticky všechny stejnou souřadnici *y*. To neplatí, pokud je kontura nepřesně detekována a počáteční bod je potom posunut třeba jen o jeden pixel ve svislém směru nebo při rozdílu výšky velkých a malých písmen. Funkce ji pak uloží do jiného řádku než sousední znak, i když rozdíl těchto řádků bude malý (třeba jen jeden pixel). Je třeba projít celý seznam kontur a tyto přeskládat tak, aby řádky vznikaly jen pokud je rozdíl souřadnice *y* opravdu rozdíl řádku.

Algoritmus pracuje tak, že prochází seznam kontur, sleduje rozdíly souřadnice *y* a počítá jednotlivé znaky. Pokud narazí na rozdíl větší než je výška jednoho znaku (výška se zjišťuje pomocí

jednoduché funkce *get_height* vytvořené pro tyto účely), označí následující znak jako znak dalšího řádku. Poté se vrací zpět o počet znaků, které zatím byly napočítány (měl by odpovídat počtu znaků na řádku) a třídí znaky podle souřadnice *x*. Po setřídění všech znaků lze považovat řádek za správně setříděný se všemi znaky, které v něm mají být. Algoritmus pokračuje na místě, které si označil jako další řádek, vynuluje počítadlo znaků a vše se opakuje. Pokud narazí na konec seznamu, proběhne poslední setřídění a algoritmus skončí.

Problém přerušených kontur bohužel uspokojivě vyřešen není. Jeho úspěšné odstranění spočívá ve správném rozpoznání přerušených kontur – což se v implementaci ukázalo jako nelehký úkol. Byl navržen a implementován algoritmus, který kontroluje, zda řetězcový kód neprochází vícekrát počátečním bodem, což je příznak přerušené kontury. Pokud je taková kontura zjištěna, její řetězcový kód je uložen, ale není dále zpracováván. Pokud je další kontura opět přerušená, je její kód přičten k uloženému kódu. Až v případě, že další kontura je již neporušená, je uložený kód zpracován (viz níže) a znak předán k rozpoznání. Tato metoda předchází víceméně úspěšně zdvojování znaků a výsledky rozpoznaných znaků ze sčítaných kódů jsou také relativně dobré (ve výstupu jsou však tištěny za znakem „*“ označující, že rozpoznaný znak může být chybný). Nezabraňuje ale chybám, které fragmenty přerušených kontur způsobují v segmentaci textu a znaky.

Zpracování kontur

Ve chvíli, kdy je vstupní text správně detekován a seřazen, zbývá přečíst jednotlivé kontury a zpracovat je. Čtení kontur je prováděno funkcemi *cvStartReadChainPoints* a *cvReadChainPoint*.

Zpracování kontur znamená tvorbu Freemanova řetězcového kódu z přečtených bodů kontur podle schématu na obrázku 2.8 v kapitole 1. Z tohoto kódu je dále vytvářen histogram a normalizovaný histogram. Následuje příklad zpracování jednoduchého kódu.

```
    Freeman code = [5,4,4,5,4,5,4,4,5,4,4,5,4,5,4,4,5,4,4,5,4,5,
4,4,5,4,2,2,2,2,0,0,0,1,0,1,1,2,1,2,2,2,3,1,3,2,2,3,3,4,4,3,4,4,2,
2,2,2,0,7,0,0,7,0,7,0,0,7,0,7,0,0,7,0,7,0,0,7,0,0,7,0,7,0,6,6,6,]
    Počet kroků: 87
    Freeman histogram:
    [0]:20      ********************
    [1]:5       *****
    [2]:14      **************
    [3]:5       *****
    [4]:20      ********************
    [5]:10      **********
    [6]:3       ***
    [7]:10      **********
```

```
Normalizovaný histogram:
[0]: 0.227273
[1]: 0.056818
[2]: 0.159091
[3]: 0.056818
[4]: 0.227273
[5]: 0.113636
[6]: 0.034091
[7]: 0.113636
```

Normalizovaný histogram vzniká tak, že každou hodnotu histogramu dělíme celkovým počtem prvků ve Freemanově kódu a výsledný podíl zapíšeme na příslušné místo. Právě hodnoty normalizovaného Freemanova histogramu předáváme umělé neuronové síti na rozpoznání znaku. Výhodou normalizovaného histogramu je relativní nezávislost na velikosti zpracovávaného znaku (lze pozorovat na příkladu trénovacího souboru pro znak „A“ výše v této kapitole, kde je takto popsáno osm znaků A každý jiné velikosti - přesto jsou hodnoty u každého znaku téměř shodné). Pokud je znak správně rozpoznán, objeví se na výstupu neuronové sítě hodnota 1.000 nebo jen o něco málo nižší právě na místě odpovídající pořadí znaku v abecedě.

Rozpoznání znaku

Toto je finální fáze programu. Je velmi jednoduchá a krátká. Neuronové síti je na vstupu funkce *fann_run* předán normalizovaný histogram rozpoznávaného znaku. Výstupem funkce je pole *calc_out* hodnot typu *float*.

Pokud rozpoznání proběhlo úspěšně, objeví se hodnota 1.000 pouze na jedné a správné pozici pole *calc_out*. Pokud nastaly při rozpoznávání problémy, mohou se projevit jako několik hodnot 1.000, většinou na místech znaků, které jsou si velmi podobné (nebo se tak alespoň jeví neuronové síti), nízké hodnoty (nižší než 0,8) nebo žádná kladná hodnota. To záleží na konkrétním případě. Pokud je znak úplně nerozpoznán, jsou většinou všechny hodnoty pole záporné, nebo blízké 0. V případě nejistoty rozpoznání znaku (maximální vrácený hodnota je nižší než 0.8) je místo znaku tištěn znak „ . “ (tečka).

7 Výsledky

Tato kapitola obsahuje výsledky získané při vytváření, a testování aplikace. Krom hodnocení výstupů rozpoznávání testu a znaků, je zde také průběh trénování neuronové sítě a celkové shrnutí výsledků aplikace.

7.1 Rozpoznávání textu

Úspěšnost rozpoznávání znaků jsou samozřejmě jedním z hlavních faktorů, který nás na OCR systému bude zajímat. Je na místě přiznat, že výsledky vyvíjeného programu nejsou nijak oslnivé. Kamenem úrazu je již samotný koncept založený na hranové detekci.

Kontury znehodnocené šumem nebo nízkým detailem jsou rozpoznávány chybně a často negativně ovlivňují další faktory jako správná segmentace textu a podobně. Ukázalo se, že pouhý normalizovaný histogram Freemanova řetězcového kódu na spolehlivé rozpoznání běžného počtu znaků není dostačující. Program v nynější fázi vývoje rozpoznává pouze velká písmena abecedy a doučení dalších znaků by výsledky ještě zhoršilo. Častými chybami jsou záměna znaků „W“ a „M“, „A“ a „V“, „Y“ a „V“ a další. Ke zlepšení by bylo zapotřebí přidat další charakteristiky znaku, z nichž by neuronová síť mohla lépe rozpoznat o jaký znak se jedná.

Nicméně, pokud jsou předkládané znaky dostatečně velké a hranová detekce nalezne nepřerušené čisté kontury, probíhá rozpoznání znaku bez problémů.

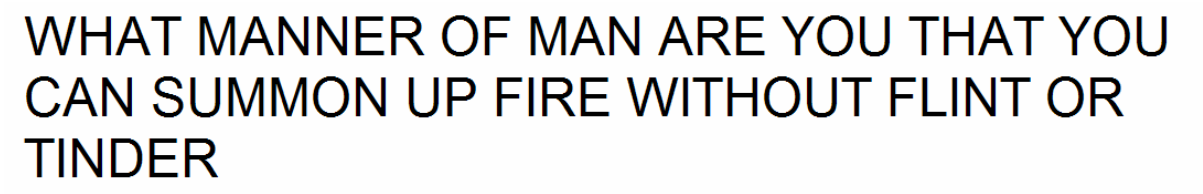
Co se týče účinnosti Houghovy transformace - nakloněný text je úspěšně zjištěn, rotací dojde k jeho narovnání do vodorovné polohy, ale výsledky rozpoznání jsou rovněž slabé. Problémem zůstávají přerušené kontury, jež vedou k chybám segmentace textu a naivní algoritmus dělení slov. Oba problémy spolu souvisí, chybně dělení slov je navíc umocněno proporcionálním písmem (znaky jsou různě široké)

Následují ukázky testovacích obrázků a výstupu programu.

Test 1

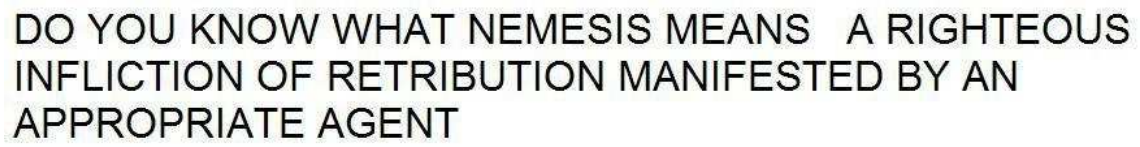
Výstup: `AHOJTOH LE JE TEXT AN IC VIC`

Test 2



Výstup:
```
*WHAT*MANNER M OF*MAN ARE VOU THAT VOU M
CAN SU*MM*MON UP F IRE*WITHOUT FL INTOR M
T INDER
```

Test 3



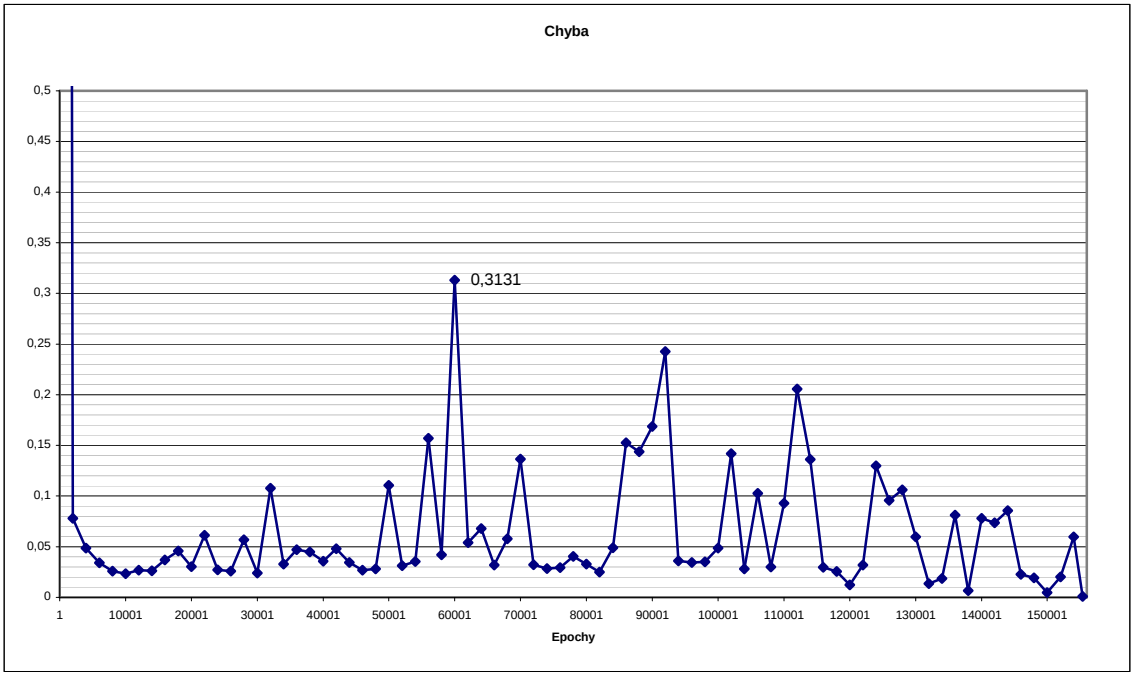
Výstup:
```
D.V.U*KN.*K HAT NE*MM*MES IS*MEANS AR IGHTE.US M
IN FL ICT I.N .F RETR IBUT I.N*MAN IFESTED BV M
AN APPR.PR IATE AGENT
```

7.2 Trénování neuronové sítě

Trénování sítě byla uskutečněna celá řada, avšak uveden je pouze výstup trénování finálního, použitého v programu. Ostatní dílčí výsledky byly víceméně experimentální a jejich hodnoty nemají hlubší význam.

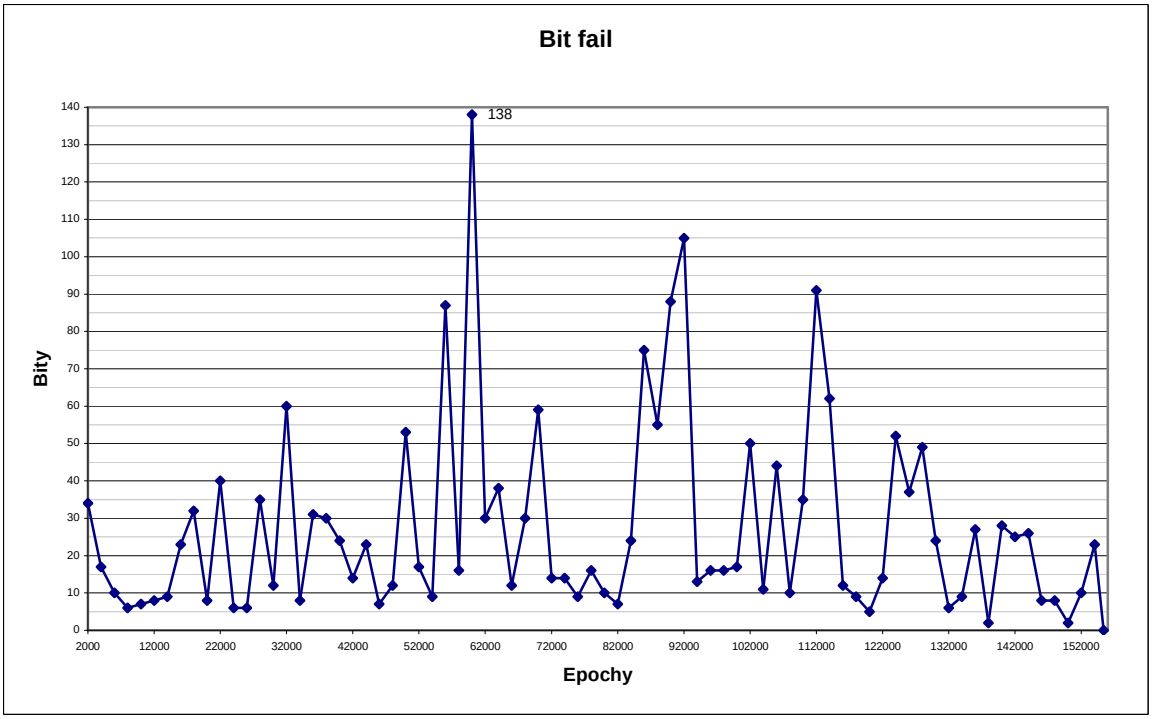
Trénování proběhlo na jediném trénovacím souboru `ocr_english.data`. Tento soubor obsahuje trénovací data získaná za pomoci *programu1* použitého na trénovaní obrázky zobrazující velká písmena anglické abecedy fontem Arial. Každý obrázek zobrazoval osm znaků různé velikosti – přibližně od velikosti 28 po 72. Neuronová síť měla 260 neuronů ve skryté vrstvě a požadovaná maximální chyba byla 0,001. Kompletní průběh trénování neuronové sítě je v příloze [2].

Na grafu č.1 můžeme sledovat průběh chyby v průběhu učení. Hodnota chyby po prvním cyklu je 6,4113, není tedy kvůli rozdílu v řádu na grafu znázorněna.



[Graf 1] Hodnota chyby v průběhu učení

Požadované hodnoty maximální chyby bylo dosaženo po 155 377 cyklech, její hodnota činí 0,000993. Na grafu č.2 je znázorněna *bit fail* – počet výstupních neuronů s výstupní chybou přesahující povolenou hranici. Je zřejmé, že oba grafy mají velmi podobný průběh.



[Graf č.2] Počet neuronů s větší než požadovanou chybou

Závěr

Zhodnocení výsledků programu je v kapitole 7. Ačkoli nyní nejsou výsledky na takové úrovni, aby se dal použít v praxi, program jako takový není navržen špatně a možnosti jeho nebyly vyčerpány. Stávající problémy s přerušenými konturami a chybným rozpoznáváním některých znaků lze vyřešit sofistikovanějšími algoritmy segmentace textu, resp. přidáním dalších charakteristik identifikujících znak přesněji a jednoznačněji. Rozpoznání většího množství znaků je možné prostým natrénováním neuronové sítě.

Během přípravy a návrhu projektu jsem se hlouběji seznámil s problematikou zpracování obrazu, hranové detekce a práce s neuronovými sítěmi. Vlastní přínos shledávám v pochopení postupů užívaných v dnešní praxi zpracování obrazu i principů sofistikovanějších metod segmentace obrazu, jež lze v budoucnu uplatnit při vývoji a návrhu dalších programů.

Navržený systém ještě není takové podobě, kdy by se dal prakticky použít. Krom odstranění stávajících problémů se v budoucnu dá rozšířit o rozpoznání většího množství znaků, včetně interpunkce a podpory fontů. K pohodlnějšímu užívání je možno navrhnout grafické uživatelské rozhraní umožňující přesnější nastavení programu i práci s rozpoznaným textem. Další možnosti skýtá implementace vícero přístupů k problematice rozpoznání textu, nejen hranové detekce.

Literatura

[1] Ing. Špatněl, M., Ing. Beran, V. *Obrazové segmentační techniky – Přehled existujících metod*,12. října 2005, Fakulta informačních technologií VUT, Božetěchova 2, Brno. Dostupný na URL: http://www.fit.vutbr.cz/~spanel/segmentace/.en (květen 2007)

[2] Øivind Due Trier, Anil K Jain, Torfinn Taxt, *Feature Extraction Methods For Character Recognition A Survey*, 19.července 1995. Dostupný na URL: http://citeseer.ist.psu.edu/trier95feature.html (květen 2007)

[3] Jukka Iivarinen, *Shape coding techniques*, 18. června 1997. Dostupný na URL: http://www.bmva.ac.uk/bmvc/1997/papers/062/node2.html (květen 2007)

[4] James Matthews, *An introduction to edge detection: The Sobel edge detector*, 2002. Dostupný na URL: http://www.generation5.org/content/2002/im01.asp (květen 2007)

[5] Steffen Nissen, *Artificial Neural Networks made easy with the FANN library*, březen 2005. Dostupný na URL: http://www.codeproject.com/library/Fann.asp (květen 2007)

[6] Bill Green, *Canny Edge Detection Tutorial*, 2002. Dostupný na URL: http://www.pages.drexel.edu/~weg22/can_tut.html (květen 2007)

[7] Bill Green, *Edge Detection Tutorial*, 2002. Dostupný na URL: http://www.pages.drexel.edu/~weg22/edge.html (květen 2007)

[8] Vladimír Myslík, *Referát z předmětu speciální architektury*, 1996. Dostupný na URL: http://aldebaran.feld.cvut.cz/~xmyslik/www/neural.html (květen 2007)

[9] Ladislav Metelka, *Empirická objektivní analýza pole slunečního svitu pomocí neuronových sítí*, 1999. Dostupný na URL: http://www.chmi.cz/poboc/HK/OK/PUBLIKACE/SSV/SSV.htm (květen 2007)

Seznam příloh

Příloha č.2 – Trénování neuronové sítě

Toto je kompletní výpis průběhu trénování neuronové sítě tak, jak jej vrací knihovna FANN.

```
Max epochs  5000000. Desired error: 0.0010000000.
Epochs            1. Current error: 6.4112842036. Bit fail 5304.
Epochs         2000. Current error: 0.0778804106. Bit fail 34.
Epochs         4000. Current error: 0.0487047177. Bit fail 17.
Epochs         6000. Current error: 0.0339574136. Bit fail 10.
Epochs         8000. Current error: 0.0257935150. Bit fail 6.
Epochs        10000. Current error: 0.0233613907. Bit fail 7.
Epochs        12000. Current error: 0.0267274613. Bit fail 8.
Epochs        14000. Current error: 0.0261899303. Bit fail 9.
Epochs        16000. Current error: 0.0368521027. Bit fail 23.
Epochs        18000. Current error: 0.0458873814. Bit fail 32.
Epochs        20000. Current error: 0.0304665028. Bit fail 8.
Epochs        22000. Current error: 0.0613334413. Bit fail 40.
Epochs        24000. Current error: 0.0270417695. Bit fail 6.
Epochs        26000. Current error: 0.0257543957. Bit fail 6.
Epochs        28000. Current error: 0.0569682449. Bit fail 35.
Epochs        30000. Current error: 0.0241223410. Bit fail 12.
Epochs        32000. Current error: 0.1076288504. Bit fail 60.
Epochs        34000. Current error: 0.0329395346. Bit fail 8.
Epochs        36000. Current error: 0.0469982157. Bit fail 31.
Epochs        38000. Current error: 0.0447146191. Bit fail 30.
Epochs        40000. Current error: 0.0357087780. Bit fail 24.
Epochs        42000. Current error: 0.0478691681. Bit fail 14.
Epochs        44000. Current error: 0.0344890403. Bit fail 23.
Epochs        46000. Current error: 0.0269362739. Bit fail 7.
Epochs        48000. Current error: 0.0282495396. Bit fail 12.
Epochs        50000. Current error: 0.1105281699. Bit fail 53.
Epochs        52000. Current error: 0.0313324063. Bit fail 17.
Epochs        54000. Current error: 0.0352442007. Bit fail 9.
Epochs        56000. Current error: 0.1569409090. Bit fail 87.
Epochs        58000. Current error: 0.0418723518. Bit fail 16.
Epochs        60000. Current error: 0.3131675533. Bit fail 138.
Epochs        62000. Current error: 0.0539633246. Bit fail 30.
Epochs        64000. Current error: 0.0678015176. Bit fail 38.
Epochs        66000. Current error: 0.0320224552. Bit fail 12.
Epochs        68000. Current error: 0.0577722390. Bit fail 30.
Epochs        70000. Current error: 0.1364379584. Bit fail 59.
Epochs        72000. Current error: 0.0323052991. Bit fail 14.
Epochs        74000. Current error: 0.0282958863. Bit fail 14.
Epochs        76000. Current error: 0.0292826166. Bit fail 9.
Epochs        78000. Current error: 0.0405816611. Bit fail 16.
Epochs        80000. Current error: 0.0328830971. Bit fail 10.
Epochs        82000. Current error: 0.0249444083. Bit fail 7.
Epochs        84000. Current error: 0.0489973461. Bit fail 24.
Epochs        86000. Current error: 0.1526841650. Bit fail 75.
Epochs        88000. Current error: 0.1438053542. Bit fail 55.
Epochs        90000. Current error: 0.1686698315. Bit fail 88.
Epochs        92000. Current error: 0.2426943872. Bit fail 105.
Epochs        94000. Current error: 0.0361553921. Bit fail 13.
Epochs        96000. Current error: 0.0344649039. Bit fail 16.
Epochs        98000. Current error: 0.0349419912. Bit fail 16.
Epochs       100000. Current error: 0.0485633168. Bit fail 17.
Epochs       102000. Current error: 0.1416617749. Bit fail 50.
Epochs       104000. Current error: 0.0280498056. Bit fail 11.
Epochs       106000. Current error: 0.1025376974. Bit fail 44.
```

```
Epochs       108000. Current error: 0.0299014204. Bit fail 10.
Epochs       110000. Current error: 0.0929193029. Bit fail 35.
Epochs       112000. Current error: 0.2055166843. Bit fail 91.
Epochs       114000. Current error: 0.1361749593. Bit fail 62.
Epochs       116000. Current error: 0.0298367552. Bit fail 12.
Epochs       118000. Current error: 0.0256130929. Bit fail 9.
Epochs       120000. Current error: 0.0123618689. Bit fail 5.
Epochs       122000. Current error: 0.0317693435. Bit fail 14.
Epochs       124000. Current error: 0.1299209595. Bit fail 52.
Epochs       126000. Current error: 0.0958157240. Bit fail 37.
Epochs       128000. Current error: 0.1059731315. Bit fail 49.
Epochs       130000. Current error: 0.0595485697. Bit fail 24.
Epochs       132000. Current error: 0.0134270892. Bit fail 6.
Epochs       134000. Current error: 0.0185814500. Bit fail 9.
Epochs       136000. Current error: 0.0810762013. Bit fail 27.
Epochs       138000. Current error: 0.0067818843. Bit fail 2.
Epochs       140000. Current error: 0.0778681016. Bit fail 28.
Epochs       142000. Current error: 0.0737450918. Bit fail 25.
Epochs       144000. Current error: 0.0855508973. Bit fail 26.
Epochs       146000. Current error: 0.0227192009. Bit fail 8.
Epochs       148000. Current error: 0.0192690024. Bit fail 8.
Epochs       150000. Current error: 0.0047411688. Bit fail 2.
Epochs       152000. Current error: 0.0200905730. Bit fail 10.
Epochs       154000. Current error: 0.0595614630. Bit fail 23.
Epochs       155377. Current error: 0.0009930544. Bit fail 0.
```