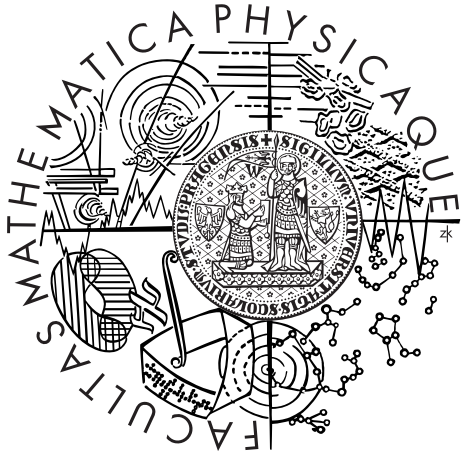
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



František Haas

Rozpoznávání znaků v digitalizovaných matematických výrazech

Katedra aplikované matematiky

Vedoucí bakalářské práce: RNDr. Tomáš Valla
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2011

Poděkování.
Chtěl bych poděkovat vedoucímu práce RNDr. Tomáši Vallovi za cenné přípomínky, podněty k zamyšlení a trpělivost během vypracování bakalářské práce. A svému otci RNDr. Františku Haasovi za dobré rady a podporu.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Rozpoznávání znaků v digitalizovaných matematických výrazech

Autor: František Haas

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: RNDr. Tomáš Valla, Katedra aplikované matematiky

Abstrakt: Cílem bakalářské práce je nalezení vhodných metod a algoritmů pro segmentaci textu a rozpoznávání symbolů pomocí umělých neuronových sítí. Nejdříve se práce věnuje základním principům umělého neuronu a umělých neuronových sítí, struktuře konvolučních neuronových sítí a zejména pak algoritmu zpětného šíření a stochastického algoritmu Levenberga-Marquardta. Dále se pak práce věnuje zpracování obrazu a jeho segmentaci na jednotlivé symboly pomocí grafových algoritmů. Součástí této práce je i implementace těchto metod a algoritmů v aplikaci, která zpracovává digitalizované matematické výrazy a převádí je do vektorového formátu.

Klíčová slova: neuronové sítě, segmentace obrazu, rozpoznávání znaků

Title: Recongition of symbols in digitalized mathematical expressions

Author: František Haas

Department: Department of Applied Mathematics

Supervisor: RNDr. Tomáš Valla, Department of Applied Mathematics

Abstract: This bachelor thesis focuses on finding suitable methods and algorithms for text segmentation and character recognition using artificial neural networks. Firstly, the thesis covers basic principles of artificial neuron and artificial neural networks, structure of convolutional neural networks and mainly backpropagation algorithm and stochastic Levenberg-Marquardt algorithm. Then the thesis describes image processing and image segmentation to single symbols using graph algorithms. This thesis also includes implementation of these methods and algorithms in an application which converts digital mathematical expressions to vector format.

Keywords: neural networks, image segmentation, character recognition

Obsah

Úvod

0.1 Motivace

V současné době existuje již řada nástrojů pro optické rozpoznávání textu, nicméně drtivá většina těchto programů je přistupná pouze komerčně a bez otevřeného zdrojového kódu. Encyklopedie Wikipedia [9] uvádí seznam 30 programů zaměřených na optické rozpoznávaní textu. Ze všech uvedených nástrojů, o kterých lze zjistit bližší informace a nejsou pouze nadstavbou nebo rozhraním k jiným programům, jsou v tomto seznamu jen čtyři s otevřeným zdrojovým kódem: CuneiForm [10], GOCR [11], OCrad [12] a Tesseract [13]. CuneiForm je specializovaný pro rozpoznávání tištěných textů a nerozeznává ručně psaný text. GOCR je určen pouze pro rozpoznávání tištěných znaků z rodiny sans-serif. OCrad rozpoznává tištěné znaky extrakcí charakteristik a vyhledáváním ve své databázi tištěných symbolů. Tesseract ve verzi 2 (stabilní verze v době tvorby této práce) nepracoval s jiným než pouze řádkově strukturovaným vstupem. Většinou se tedy aplikace zabývají pouze zpracováním tištěného textu a to především v řádkové struktuře.

Tématem této práce je prozkoumat a implementovat metody pro segmentaci a rozpoznání ručně psaných matematických výrazů. Cílem práce je udělat první krok pro vytvoření aplikace, převádějící ručně psané matematické rovnice do typografického formátu. Tím se rozumí návrh a implementace postupů pro zpracování a segmentaci obrazu a rozpoznání ručně psaných symbolů.

Tato práce se nezabývá samotným převodem do typografického formátu, ale přípravou dat pro tento proces a výsledná aplikace demonstruje své schopnosti převodem matematických výrazů do vektorového formátu.

První fází této přípravy je segmentace znaků z obrazu textu zaznamenaného například fotoaparátem, skenerem nebo elektronickou tužkou na tabletu. Tento proces je výpočetně náročný, algoritmy pracují nad velkým množstvím dat v podobě bitmap s vysokým rozlišením a počet dílčích kroků postupně zpracovávajících obraz je veliký. Důvodem je snaha dobře segmentovat nejen čisté záznamy z tabletů a elektronické tužky, ale i data méně ostrá a méně konstratní zachycená například kompaktním fotoaparátem.

Druhou fází je rozpoznávání jednotlivých znaků. Pro rozpoznávání jsou použity konvoluční neuronové sítě, protože dosahují velmi dobrých výsledků při rozpoznávání textu. Nicméně konvoluční neuronové sítě neměly být zpočátku v této práci použity. Prvním konceptem bylo použití pouze plně zapojených dopředných sítí spolu s algoritmy pro extrakci charakteristik symbolů, podle kterých měla síť znaky rozpoznávat. Tato koncepce však nepodávala přesvědčivé výsledky a proto byla použita ověřená metoda.

0.2 Neuronová síť

Za umělou neuronovou síť se obecně považuje struktura vzájemně propojených prvků, neuronů. Tento koncept je inspirovaný studiem biologického nervového systému.

Neuron, obdobně jako jeho biologická předloha, je funční prvek tvořený vstupem (dendrit), tělem (soma) a výstupem (axon). Základní funkcí neuronu je reakce na vnější podněty přicházející vstupy a přenos reakce výstupem dál.

Stejně jako biologické neuronové sítě, jsou umělé neuronové sítě velmi výkonným systémem. Hlavní funkcí neuronové sítě je reagovat výstupem na předložený vzor. Přitom pod pojmem předložený vzor, stejně jako pod pojmem výstup, je možno si představit široké spektrum objektů. Může se jednat o vstup matematické funkce a výstupem může být její aproximace, nebo mohou být síti předložena data o trhu, počasí a pod. a od sítě je pak požadována předpověď na základě historického vývoje.

0.3 Klasifikátor

Klasifikace, lze říci třídění, škatulkování, rozpoznávání, je člověku vlastní a běžná činnost, kterou užívá denně. Je tak samozřejmá, že o ní člověk zřídkakdy přemýšlí. Právě teď, s každým řádkem, který čtenář tohoto textu čte, jeho mozek rozpoznává písmena, slabiky a slova. Člověk běžně poznává místa, kde byl, byť během času došlo k jejich změně, poznává osoby, byť se stárnutím, usilovným cvičením či plastickými operacemi mění jejich vzhled, poznává zvuky nebo hlasy, které již někdy slyšel, byť se s časem mění jak tyto zvuky a hlasy, tak i citlivost smyslů posluchače. Vstupem jsou v tomto případě vjemy lidských smyslů, předzpracované orgány a rozpoznané velkou neuronovou sítí, lidským mozkem.

Toto ale nejsou schopnosti, se kterými se člověk rodí. Jsou to dovednosti, které si každý jednotlivý člověk dlouho osvojuje, kterým se velmi dlouho učí. Učení je přitom významná schopnost, která je od klasifikátoru vyžadována. Stejně jako děti listují slabikáři a obrazkovými knížkami a učí se rozpoznávat barvy, zvířata a věci, tak i od klasifikátoru je požadováno, aby se učil. Aby na základě informací o tom, zda předložený vzor rozpoznal nebo ne, modifikoval své postupy.

Další významnou vlastností lidského mozku je schopnost generalizovat a zobecňovat. Je to schopnost rozpoznat nejen identické kopie naučených vzorů, ale i vzory jim podobné. Tak, jako člověk umí rozeznat např. tvář, kterou předtím viděl jen na fotce. Tak, jako je člověk schopen číst text napsaný různými druhy (typy, fonty) písma, i když se ve škole učil jen jeden druh. To je schopnost, kterou by klasifikátor měl mít. Nemůže být schopen rozpoznávat jen identické vzory, neboť právě schopnost učení, zobecňovámí, tedy určitá úroveň inteligence, je to, co je od klasifikátoru hlavně požadováno.

Zásadním požadavkem při navrhování, tvorbě, klasifikátoru je možnost jeho praktického sestrojení a užívání na dostupných prostředcích. Například klasifikace deseti různých vzorů (geometrických tvarů, písmen, znaků, značek, atd.) na pixelové mapě o velikosti 32 na 32 polí s 256 barvami. Celkový počet všech možných podob takové mapy je doslova astronomický: $(32 \cdot 32)^{256} \approx 4,3 \cdot 10^{770}$. Na uložení jedné mapy je potřeba 256 B, na uložení všech map tedy více než $1.075 \cdot 10^{761}$ GB , což je na osobním počítači nemožné. Proto je třeba nalézt efektivnější nástroje a tato bakalářská práce se zabývá použitím neuronových sítích jako klasifikátoru.

1. Struktura neuronu a neuronové sítě

1.1 Neuron

Neuron je základní funkční prvek umělé neuronové sítě. Jeho předlohou je biologický neuron. Z tohoto pohledu nejpodstatnější částí biologického neuronu jsou dendrit, soma, axon a synapse. Dendrity jsou dostředivé, většinou relativně krátké, výběžky neuronu, které nepřímo přijímají a přenášejí vzruchy od jiných neuronů do vlastního těla (soma) neuronu. Axon je naopak relativně dlouhý odstředivý výběžek vycházející z těla neuronu. Samotné spojení mezi neurony je zajištěno synapsemi, které bez kontaktu a na chemické bázi přenášejí vzruchy.

Zjednodušeně řečeno, funkcí biologického neuronu jako celku je tlumit nebo zesilovat signály, které přicházejí dendrity nebo přímo do samotného soma a vysílat je dalším neuronům axonem. Neurony můžeme dle podoby a struktury rozdělit do tří skupin. Jedná se o unipolární, bipolární a multipolární neurony. Všechny typy neuronů mají právě jeden axon a soma, unipolární neuron je specifický absencí dendritů, bipolární má pouze jeden dendrit, multipolární pak mají více dendritů. Čili do neuronu přicházejí dendrity různé vzruchy od různých neuronů a odchází právě jeden vzruch jedním axonem, může být ale přenesen do libovolného počtu neuronů synapsemi.

Umělý neuron můžeme definovat jako matematickou funkci o n proměnných a n parametrech následovně:

$$s = \sum_{i=0}^{n} x_i \cdot w_i \tag{1.1}$$

$$y = \begin{cases} 1 & \text{pro } s \geq 0; \\ 0 & \text{pro } s < 0. \end{cases} \tag{1.2}$$

Například pro $n = 2$ se při výpočtu výstupu neuronu fakticky nejprve počítá skalární součet dvou dvourozměrných vektorů a následně porovnává úhel, který navzájem svírají. Množinu bodů v rovině lze správným nastavením vah neuronu rozdělit na dvě disjunktní množiny. Omezením v tomto případě je, že dělící přímka musí procházet středem souřadného systému, protože váhy neuronu určují vektor s počátkem právě v tomto středu. Aby bylo eliminováno toto omezení, je nutno upravit definici neuronu následovně.

$$s = \sum_{i=0}^{n} (x_i \cdot w_i) + \theta \tag{1.3}$$

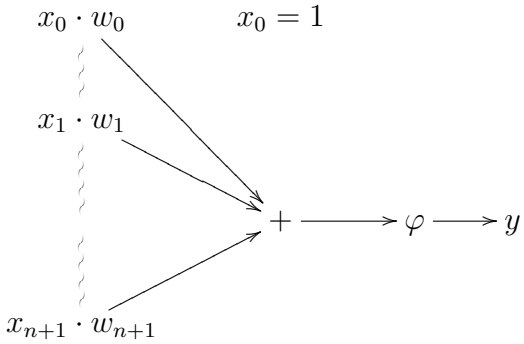
$$y = \begin{cases} 1 & \text{pro } s \geq 0; \\ 0 & \text{pro } s < 0. \end{cases} \tag{1.4}$$

Do definice neuronu je přidán tzv. práh, ten umožňuje posunout dělící přímku ze středu roviny. Pro ilustraci, je-li úkolem rozdělit body [2,4], [4,2] a [5,5] tak, aby body [2,4], [4,2] ležely v jedné polorovině a bod [5,5] ve druhé, je vidět, že

přímka procházející středem souřadného systému to neumožňuje, ale pro neuron kde $w_1 = 1, w_2 = 1, p = -8$, neuron body správně rozdělí.

Definici lze ještě trochu zobecnit a zjednodušit sjednocením vstupů a prahu, aby se zjednodušil proces učení. Místo prahu se přidá neuronu další vstup se stabilní hodnotou jedna a k němu příslušná váha. Na rozdíl od změny prahu při učení se tedy bude v neuronu měnit o jednu váhu navíc. A tak skokovou přenosovou funkci lze nahradit libovolnou funkci jedné proměnné. Obecná podoba neuronu vypadá takhle.

Obrázek 1.1:



$$y = \varphi(\sum_{i=0}^{n+1} (x_i \cdot w_i)) \tag{1.5}$$

Definice 1

- *Nechť $\vec{x} = (x_1, .., x_{n+1})$ je vstupní vektor neuronu, pak $\vec{x} = (x_0 = 1, x_1, .., x_{n+1})$ je rozšířený vstupní vektor neuronu, kde každé $x_i \in R$.*
- *Nechť $\vec{w} = (w_1, .., w_{n+1})$ je váhový vektor neuronu, pak $\vec{w} = (w_0, w_1, .., w_{n+1})$ je rozšířený váhový vektor neuronu, kde každé $w_i \in R$.*
- *Formální neuron (obrázek 1.1, výraz (1.5)) je výpočetní jednotka, jejíž vstup je rozšířený váhový vektor a jejíž výstup se počítá podle rovnice 1.5.*
- *Perceptron (výraz (1.3), výraz (1.4)) je speciální případ formálního neuronu, kde $w_0 = -\theta$ a přenosová funkce φ je rovna výrazu (1.4).*
  (podle [2])

1.2 Lineární separabilita

Výše popsaný model neuronu lze jako výpočetní jednotku použít například pro zpracování jednoduchých logických funkcí. Pro váhy $\vec{w} = (-2, 1, 1)$ a výše popsanou skokovou přenosovou funkci neuron řeší funkci AND, pro váhy $\vec{w} = (-1, 1, 1)$ pak funkci OR.

Tabulka 1.1:

AND

x_0	x_1	x_2	w_0	w_1	w_2	s	$=$
1	0	0	-2	1	1	-2	0
1	0	1	-2	1	1	-1	0
1	1	0	-2	1	1	-1	0
1	1	1	-2	1	1	0	1

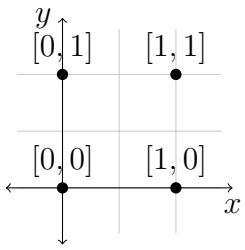
Tabulka 1.2:

OR

x_0	x_1	x_2	w_0	w_1	w_2	s	$=$
1	0	0	-1	1	1	-1	0
1	0	1	-1	1	1	0	1
1	1	0	-1	1	1	0	1
1	1	1	-1	1	1	1	1

Pokud by se podobným způsobem řešila logická funkce XOR, nebylo by možné nalézt řešení. Problémem je tzv. lineární separabilita. Pokud jsou vstupní hodnoty funkce XOR chápany jako souřadnice bodů v rovině a jejich příslušnost k množině dělené neuronem je označována hodnotou funkce XOR, patří body [0,1] a [1,0] do množiny označené 1 a body [0,0] a [1,1] do množiny označené 0. Je zřejmé, že tyto čtyři body není možné jednou přímkou rozdělit do zadaných množin (oddělit přímkou) a tedy není možné tuto úlohu řešit jedním výše definovaným neuronem.

Obrázek 1.2:

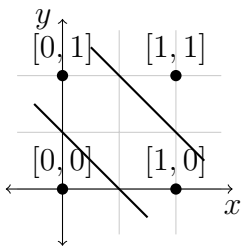


Pro řešení úlohy je vhodné nejprve ji rozdělit na dílčí části zvládnutelné samostatnými neurony. Spojením neuronů pak vytvořit síť řešící daný problém.

První neuron bude reagovat na vstup [0,0] výstupem 0 a výstupem 1 na zbylé vstupy. Druhý neuron bude reagovat na vstup [1,1] obdobně. Pro vstupy [0,1] a [1,0] tedy budou oba neurony reagovat výstupem 1, naopak pro vstupy [0,0] a [1,1] bude vždy alespoň jeden neuron reagovat výstupem 0. Do třetího neuronu proto jako vstupy budou zapojeny výstupy obou těchto neuronů a váhy budou

nastaveny stejně jako při řešení logické funkce AND. Je vidět, že síť řeší logickou funkci XOR.

Obrázek 1.3:



1.3 Neuronová síť

Po seznámení se s jednoduchým neuronem a problémem lineární separability je zřejmé, že pro praktické použití je potřeba silnějšího nástroje než jednotlivého neuronu. Proto se neurony využijí jako základní stavební jednotka a jejich spojením se bude tvořit neuronová síť.

Definice 2
Neuronová síť je čtveřice (I, O, N, E), kde $I \subset N$ je vstupní vrstva nebo-li vstupní vektor sítě, $O \subset N$ je výstupní vrstva sítě neboli výstupní vektor sítě, N jsou výpočetní jednotky (formální neurony) a E je množina orientovaných hran. Orientovaná hrana je trojice (u, v, w), kde $u \in I \cup N$, $v \in O \cup N$ a $w \in R$.
(podle [2])

Definice 3
Dopředná neuronová síť je neuronová síť jejíž výpočetní jednotky lze rozdělit do n disjunktních množin $N_0, \ldots, N_{n-1}$ takových, že $\bigcup_{i=0}^{n-1} N_i = N$. A pro všechny hrany $(u, v, w) \in E, u \in N_i \Rightarrow v \in N_{i+1}, I = N_0, O = N_{n-1}$.

Definice 4
Plně zapojená neuronová síť je dopředná neuronová síť, v níž pro každou dvojici vrstev N_i, N_{i+1} a pro všechny dvojice neuronů $u_i \in N_i, u_{i+1} \in N_{i+1}$ existuje hrana $(u_i, u_{i+1}) \in E$.

Definice dopředné neuronové sítě odpovídá jednoduché síti, která je uvedena jako příklad u logické funkce XOR. Je to zároveň typ sítě, kterým se dále tato práce zabývá, zejména pak učením a podrobnější strukturou zapojení vrstev a neuronů.

1.3.1 Výpočet výstupu neuronové sítě

Algoritmus výpočtu výstupu dopředné neuronové sítě (I, O, N, E) pro vzor $\vec{x}$.

Algoritmus 1

1. *Pro první vrstvu* N_0 *a její výstupy platí* $\vec{y_0} = \vec{x}$.
2. *Pro* $i \in 1 \dots n$, *kde* n *je počet vrstev. Pro* $j \in 0 \dots h$, *kde* $h = |N_i|$.

$$s_{i,j} = w_0 + \sum_{(N_{i-1,k}, N_{i,j}, w) \in E} (y_{i-1,k} \cdot w)$$

$$y_{i,j} = \varphi(s_{i,j})$$

Pro zjednodušení zápisu a implementace dopředných neuronových sítí se obvykle v každé vrstvě vytvoří jeden neuron bez vstupů, jehož stálým výstupem je hodnota 1 a do tohoto prahového neuronu se zapojí všechny neurony následující vrstvy.

2. Algoritmy učení neuronu a neuronové sítě

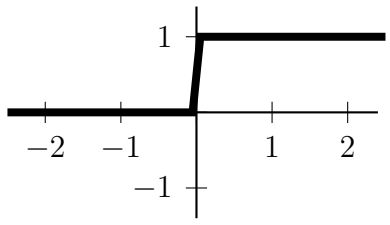
2.1 Problém učení

V předešlé kapitole byla nadefinována obecná podobu neuronu a popsán problém lineární separability. Na příkladech jednoduchých logických funkcí bylo demonstrováno použití umělého neuronu se skokovou přenosovou funkcí s výstupy 0 a 1. Určení vah neuronu v takto jednoduchých příkladech je často triviální problém, pro složitější nebo ne zcela zřejmé úlohy však bude lepší postup hledání vah podrobně popsat. Cílem učení je tedy nalézt takový vektor vah, aby neuron na předkládané vstupy odpovídal správnými výstupy

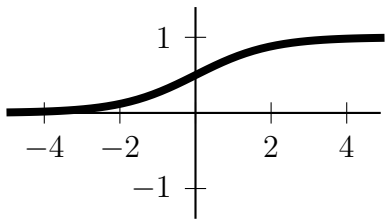
2.2 Přenosové funkce

Výběr vhodné přenosové funkce neuronu je zásadní pro použití a učení neuronu. Vzhledem k tomu, že vstupem i výstupem neuronu jsou reálná čísla, je škála možných funkcí prakticky neomezená. Jako vhodné kandidátky pro přenosovou funkci se nabízí např. následující funkce.

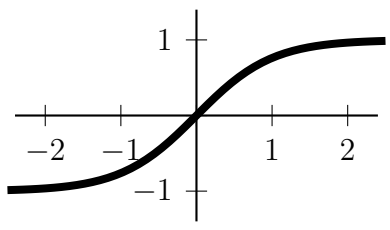
Obrázek 2.1:



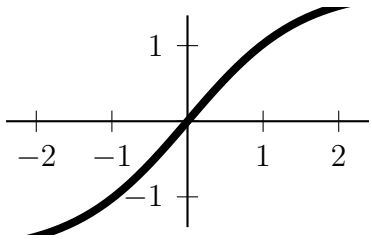
(a)
$y = 1$ pro $x >= 0$;
$y = 0$ pro $x < 0$.



(b) $y = \frac{1}{1+\exp(-x)}$



(c) $y = \tanh(x)$



(d) $y = 1.7519 \cdot \tanh(\frac{2}{3} \cdot x)$

2.3 Algoritmus delta pravidla

Algoritmus delta pravidla slouží k učení perceptronu. Lze říci, že slouží i k učení jednovrstevných perceptronových sítí, ale fakticky se jedná pouze o nepropojenou množinu perceptronů, algoritmus tedy stačí popsat na jednom perceptronu. Perceptron se bude učit, aby na vstupní vektor $(-1, 1)$ reagoval výstupem -1 a na vstupní vektor $(1, 1)$ reagoval výstupem 1. Je zřejmé, že není podstatné, zda neuron pracuje s dvojicí hodnot $0, 1$ nebo $1, -1$, protože před vstupem hodnot do neuronu a po výstupu lze dvojice jednoduše zaměnit. Váhový vektor neuronu na začátku procesu učení je zvolen náhodně.

Tabulka 2.1:

x_0	x_1	x_2	w_0	w_1	w_2	s	=	d
1	-1	1	-0.8	-1.0	0.9	1.1	1	-1
1	1	1	-0.8	-1.0	0.9	-0.9	-1	1

Je zřejmé, že v tuto chvíli perceptron na oba vstupy odpovídá nesprávně. Intuitivně, změna váhy w_i by měla být uměrná tomu, jaká hodnota vstupu odpovídajícímu váze w_i do perceptron vstoupila, tedy x_i, a jak moc perceptron chyboval u daného vzoru. V tomto případě je problém se skokovou přenosovou funkcí. Na výstupu neuronu -1 nebo 1 lze pouze poznat, zda perceptron odpověděl správně nebo špatně, není však zřejmé, jak blízko nebo daleko byl perceptron od opačné odpovědi. Proto se bude chyba perceptronu měřit jako rozdíl požadovaného výstupu a excitace perceptronu ($\delta = d - s$). Perceptron nicméně není trénován pouze pro jeden vzor, ale pro větší počet vzorů a proto váhy budou upravovány průměrem těchto hodnot. Významným prvkem tohoto algoritmu je volba a užití parametru η, který určuje jakou měrou budou aplikovány změny vah. V tomto případě je zvoleno $\eta = 1.0$ pro zjednodušení příkladu, aby se neuron naučil správně odpovídat už po první změně vah. Pokud je η příliš malé, je třeba více iterací pro nalezení správných vah. Pro η větší než 1 by naopak hodnoty vah neúměrně rostly.

Tabulka 2.2:

x_0	x_1	x_2	w_0	w_1	w_2	s	=	d	δ	$\eta\delta x_0$	$\eta\delta x_1$	$\eta\delta x_2$
1	-1	1	-0.8	-1.0	0.9	1.1	1	-1	-2.1	-2.1	2.1	-2.1
1	1	1	-0.8	-1.0	0.9	-0.9	-1	1	1.9	1.9	1.9	1.9
celkem										-0.2	4.0	-0.2
průměr										-0.1	2.0	-0.1

Po aplikaci úprav vah se perceptron přibližuje požadovanému řešení. V tomto příkladě řeší perceptron problém správně už po první iteraci. Většinou je iterací třeba mnohem více.

Tabulka 2.3:

x_0	x_1	x_2	w_0	w_1	w_2	s	=	d
1	−1	1	−0.9	1.0	0.8	−1.1	−1	−1
1	1	1	−0.9	1.0	0.8	0.9	1	1

Proces učení perceptronu je ve své podstatě minimalizace chybové funkce:

$$\sum^{P} (f(\vec{x_p}, \vec{w}) - d_p)^2 \tag{2.1}$$

kde X je množina trénovacích vzorů, D je množina správných výstupů neuronu na vstupy z X a $|X| = |D| = P$. Vektor $\vec{w}$ je váhový vektor neuronu a funkce f znázorňuje výpočet výstupu perceptronu se vstupem $\vec{x_p}$ a váhami $\vec{w}$.

Algortimus popisuje např. [1].

2.4 Algoritmus zpětného šíření

Při popisu struktury a vlastností neuronu bylo ukázáno, že existují lineárně neseparabilní problémy, které lze řešit jen vzájemně zapojenými neurony, tedy neuronovou sítí. A pokud mají být neuronové sítě užívány pro řešení složitějších problémů jako jsou kupříkladu již zmíněné logické funkce, je nutný i efektivní způsob, jak neuronovou síť naučit daný problém správně řešit a tím je právě algoritmus zpětného šíření.

Má-li být algoritmem zpětného šíření učena dopředná neuronová síť, je třeba, aby tato síť byla vytvořena z formálních neuronů, jejichž přenosová funkce je spojitá na R (např. obrázek 2.1 (b), (c) a (d)).

Stejně jako u perceptronového učení i u algoritmu zpětného šíření je principem minimalizace chybové funkce dopředné neuronové sítě o m výstupech pro P trénovacích vzorů

$$E = (1/2) \cdot \sum_{p=0}^{P} \sum_{i=0}^{m} (y_{i,p} - d_{i,p})^2 \tag{2.2}$$

kde $y_{i,p}$ je i-tý prvek skutečného výstupu sítě na vzor $\vec{x_p}$ a $d_{i,p}$ je i-tý prvek trénovaného výstupu sítě na vzor $\vec{x_p}$.

Přenosovou funkci neuronů v síti lze zvolit například následovně (obrázek 2.1 (c))

$$y = \varphi(x) = \tanh(x) \tag{2.3}$$

Tedy změna váhy z i-tého neuronu do j-tého neuronu následující vrstvy se určí následovně

$$\Delta w_{i,j} = -\frac{\partial E}{\partial w_{i,j}} \tag{2.4}$$

Pro srozumitelnost následujících rovnic je uvedeno co označuje jednotlivý index či člen. Vrstvy jsou číslovány vzestupně od vstupní k výstupní.

Tabulka 2.4:

i	neuron z vrstvy i
j	neuron z vrstvy i+1
k	neuron z vrstvy i+2
z	neuron z poslední vrstvy
l	obecný index
p	index označuje vstupní vzory, resp. právě trénovaný
$w_{l,ll}$	váha mezi neurony l a ll
s_l	excitace neuronu l
y_l	výstup neuronu l
φ	přenosová funkce neuronu

Následujícími úpravami a derivacemi výrazu (2.4) se získají vzorce pro úpravu vah poslední vrstvy:

$$-\frac{\partial E}{\partial w_{i,j}} = -\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial s_j} \cdot \frac{\partial s_j}{\partial w_{i,j}} \tag{2.5}$$

$$\begin{aligned}
\frac{\partial s_j}{\partial w_{i,j}} &= \frac{\partial \sum_{ii} y_{ii} \cdot w_{ii,j}}{\partial w_{i,j}} = \\
&= \frac{\partial (\sum_{ii \neq i} (y_{ii} \cdot w_{ii,j}) + y_i \cdot w_{i,j})}{\partial w_{i,j}} = \\
&= \frac{\partial \sum_{ii \neq i} y_{ii} \cdot w_{ii,j}}{\partial w_{i,j}} + \frac{\partial (y_i \cdot w_{i,j})}{\partial w_{i,j}} = \\
&= 0 + y_i = \\
&= y_i
\end{aligned}$$

$$\begin{aligned}
\frac{\partial y_j}{\partial s_j} &= \frac{\partial \varphi(s_j)}{\partial s_j} = \\
&= \frac{\partial \tanh(s_j)}{\partial s_j} = \\
&= \frac{1}{\cosh(y_j)^2}
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial E}{\partial y_j} &= \frac{\partial((1/2)\cdot\sum_p\sum_{jj}(y_{jj,p}-d_{jj,p})^2)}{\partial y_j} = \\
&= \frac{\partial((1/2)\cdot\sum_p(\sum_{jj\neq j}(y_{jj,p}-d_{jj,p})^2)+(y_{j,p}-d_{j,p})^2)}{\partial y_j} = \\
&= \frac{\partial((1/2)\cdot\sum_p\sum_{jj\neq j}(y_{jj,p}-d_{jj,p})^2)}{\partial y_j} + \frac{\partial((1/2)\cdot\sum_p(y_{j,p}-d_{j,p})^2)}{\partial y_j} = \\
&= 0 + \frac{\partial((1/2)\cdot\sum_{p,(\text{pouze pro jeden vzor})}(y_{j,p}-d_{j,p})^2)}{\partial y_j} = \\
&= \frac{\partial((1/2)\cdot(y_j-d_j)^2)}{\partial y_j} \\
&= (y_j-d_j)
\end{aligned}
$$

$$\Delta w_{i,j} = -\frac{\partial E}{\partial w_{i,j}} = -(y_j-d_j)\cdot\frac{1}{\cosh(y_j)^2}\cdot y_i = \delta_j\cdot y_i \tag{2.6}$$

Podobnými úpravami a derivacemi výrazu 2.4 se získají vzorce pro úpravu vah ostatních vrstvech:

$$
\begin{aligned}
-\frac{\partial E}{\partial w_{i,j}} &= -(\sum_k \frac{\partial E}{\partial s_k}\cdot\frac{\partial s_k}{\partial y_j})\cdot\frac{\partial y_j}{\partial s_j}\cdot\frac{\partial s_j}{\partial w_{i,j}} = &(2.7)\\
&= -(\sum_k \frac{\partial E}{\partial s_k}\cdot\frac{\partial s_k}{\partial y_j})\cdot\frac{1}{\cosh(y_j)^2}\cdot y_i &(2.8)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial s_k}{\partial y_j} &= \frac{\partial\sum_{jj} y_{jj}\cdot w_{jj,k}}{\partial y_j} = \\
&= \frac{\partial(\sum_{jj\neq j}(y_{jj}\cdot w_{jj,k})+y_j\cdot w_{j,k})}{\partial y_j} = \\
&= \frac{\partial\sum_{jj\neq j} y_{jj}\cdot w_{jj,k}}{\partial y_j} + \frac{\partial(y_j\cdot w_{j,k})}{\partial y_j} = \\
&= 0 + w_{j,k} = \\
&= w_{j,k}
\end{aligned}
$$

$$-\frac{\partial E}{\partial w_{i,j}} = -(\sum_k \frac{\partial E}{\partial s_k}\cdot w_{j,k})\cdot\frac{1}{\cosh(y_j)^2}\cdot y_i$$

Z odvození pro výstupní vrstvu vyplývá

$$\frac{\partial E}{\partial y_z} = (y_z - d_z)$$

$$\frac{\partial y_l}{\partial s_l} = \frac{1}{\cosh(y_l)^2}$$

$$\delta_z = -(y_z - d_z) \cdot \frac{1}{\cosh(y_z)^2}$$

Zřejmě platí

$$\frac{\partial E}{\partial s_l} = \frac{\partial E}{\partial y_l} \cdot \frac{\partial y_l}{\partial s_l}$$

Z toho plyne

$$\begin{aligned} \frac{\partial E}{\partial s_k} &= \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial s_k} = \\ &= (y_k - d_k) \cdot \frac{1}{\cosh(y_k)^2} = \\ &= \delta_k \end{aligned}$$

A konečně

$$\Delta w_{i,j} = -\frac{\partial E}{\partial w_{i,j}} = (\sum_k \delta_k \cdot w_{j,k}) \cdot \frac{1}{\cosh(y_j)^2} \cdot y_i = \delta_j \cdot y_i \tag{2.9}$$

Shrnutí:

$$\delta_j = \begin{cases} (y_j - d_j) \cdot \frac{1}{\cosh(y_j)^2} & \text{pro výstupní neuron;} \\ (\sum_k \delta_k \cdot w_{j,k}) \cdot \frac{1}{\cosh(y_j)^2} & \text{pro skrytý neuron.} \end{cases} \tag{2.10}$$

Algoritmus 2

1. *Inicilalizuj náhodně všechny váhy neuronů a zvol parametr η.*
2. *Předlož síti vzor $\vec{x}$ z trénovací množiny X.*
3. *Vypočítej výstup $\vec{y}$ pro předložený vzor $\vec{x}$, podle algoritmu (1).*
4. *Vypočítej dle rovnic (2.10) δ_j pro všechny neurony (postupně od poslední vrstvy k první, na pořadí ve vrstvě nezáleží).*
5. *Aktualizuj váhy dle $nw_{i,j} = w_{i,j} + \eta \cdot \delta_j \cdot y_i$, kde $nw_{i,j}$ je aktualizovaná hodnota váhy $w_{i,j}$.*
6. *Pokud není stále trénovací množina prázdná, předlož jiný vzor $\vec{x} \in X$ a přejdi na bod 3.*

Toto postup popisuje první epochu algoritmu, resp. první průchod trénovací množinou. Pokud síť stále není dostatečně naučená, opakuje se proces se začátkem ve 2. bodě. Některé úlohy vyžadují řádově stovky až tisíce epoch o tisicích vzorů, než se je síť naučí dobře řešit.

Algoritmus zpětného šíření je popsaný v [1] a [2].

2.5 Stochastický algoritmus Levenberga-Marquardta

Při řešení složitých a výpočetně náročných úloh mohou metody prvního řádu hledat správné řešení příliš dlouho. Při použití algoritmu prvního řádu u náročnějších úloh je zpravidla pro nalezení dostatečně dobrých vah sítě třeba vykonat řádově stovky až tisíce průchodů (epoch) trénovací množinou podobně velké mohutnosti. Pro rychlejší proces učení se proto používají tzv. algoritmy druhého řádu. Algoritmy prvního řádu jsou založeny na první derivaci chybové funkce dle vah, algoritmy druhého řádu pak na druhé.

Tato práce se bude dále zabývat stochastickým algoritmem Levenberga-Marquardta popsaným v [3] a [4].

U algoritmu zpětného šíření je těsně před změnou váhy použit parametr η ovlivňující velikost změny. Tento parametr je pro celou síť a všechny váhy stejný. Principem algoritmu Levenberga-Marquardta je vylepšení zpětného šíření o individuální parametry učení pro každou váhu, zjištěné z druhé derivace chybové funkce.

$$\eta_{i,j} = \frac{\eta}{(\frac{\partial^2 E}{\partial w_{i,j}^2}) + \mu} \tag{2.11}$$

Konstanta μ je důležitá pro udržení $\eta_{i,j}$ v rozumných mezích i pro velmi nízké hodnoty derivace. Neznámou tedy zůstává pouze druhá derivace chybové funkce.

Postup pro aproximaci matice druhých derivací chybové funkce je uveden v [4]. Druhou derivaci chybové funkce pro tuto metodu lze získat aproximací hessovské matice druhých derivací několika zjednodušeními. Předně místo celé matice lze uvažovat pouze její diagonálu, což je patrné už z 2.11. Výraz $\frac{\partial^2 E}{\partial w_{i,j}^2}$ je průměrem druhých derivací chybové funkce přes trénovací množinu.

$$\frac{\partial^2 E}{\partial w_{i,j}^2} = \frac{1}{P} \sum_{p}^{P} \frac{\partial^2 E^p}{\partial w_{i,j}^2} \tag{2.12}$$

$$\frac{\partial^2 E^p}{\partial w_{i,j}^2} = \frac{\partial^2 E^p}{\partial s_i^2} \cdot y_j^2 \tag{2.13}$$

Postup výpočtu těchto derivací je velmi podobný samotnému algoritmu zpětného šíření.

$$\frac{\partial^2 E^p}{\partial s_j^2} = \varphi'(s_j)^2 \cdot \sum_i w_{i,j}^2 \cdot \frac{\partial^2 E^p}{\partial s_i^2} + \varphi''(s_j) \cdot \frac{\partial E^p}{\partial y_j} \tag{2.14}$$

Tyto derivace vedou k problému spojenému s každou Newtonovskou metodou, výrazy mohou být totiž záporné a mohou způsobit pohyb $\eta_{i,j}$ nesprávným směrem. Proto je použita Gaussova-Newtonova aproximace, která zaručí nezápornost druhých derivací. Výraz zpětné propagace druhých derivací je pak následující.

$$\frac{\partial^2 E^p}{\partial s_j^2} = \varphi'(s_j)^2 \sum_i w_{i,j}^2 \cdot \frac{\partial^2 E^p}{\partial {s_i}^2} \tag{2.15}$$

Algoritmus 3

1. *Předlož síti vzor* $\vec{x}$ *z trénovací množiny* $|X|$*, polož* $P := |X|$.
2. *Vypočítej výstup* $\vec{y}$ *pro předložený vzor* $\vec{x}$*, podle algoritmu (1) a polož* $p := |X|$.
3. *Pro každou vrstvu* j*, postupně od výstupní k vstupní.*

$$a_j = \begin{cases} 1 & \text{pro neurony poslední vrstvy;} \\ \sum\limits_{w_{j,k}} b_j \cdot w_{j,k}^2 & \text{pro ostatní vrstvy.} \end{cases} \tag{2.16}$$

$$b_j = a_j \cdot (\frac{1}{\cosh(s_j)^2})^2 \tag{2.17}$$

$$c_{i,j,p} = \sum_j b_j \cdot y_j^2 \tag{2.18}$$

4. *Pokud není stále trénovací množina prázdná, předlož jiný vzor, a přejdi na bod 2.*
5. *Pro každou váhu v síti spočítej novou hodnotu* $\eta_{i,j}$ *následovně:*

$$\eta_{i,j} = \frac{\eta}{((\frac{1}{P}) \cdot \sum\limits_p^P c_{i,j,p}) + \mu} \tag{2.19}$$

3. Rozpoznávání symbolů neuronovou sítí

3.1 Vstup, výstup a přenosová funkce

V této práci jsou pro rozpoznávání symbolů použity neuronové sítě. Pro jejich použití je nutné nejdříve specifikovat, jakým způsobem budou neuronové síti předávány vstupní symboly a jakým způsobem bude síť odpovídat.

Síti je jako vstup předávána bitmapa obsahující symbol. Převod dvourozměrné matice pixelů do jednorozměrného vstupního vektoru je triviální, vstupní vektor se vytvoří postupným spojením řádků matice. Barvu bitmap je třeba sjednotit na šedou škálu a velikost palety sjednotit mapováním na pevně stanovený interval $[-1.0, 1.0]$.

Jako výstup sítě je použit vektor indikující míru podobnosti k dané třídě. Rozpoznává-li síť kupříkladu 5 symbolů, pak bude výstupní vektor velikosti 5 a pro symbol z první třídy bude ideálním a zároveň trénovaným výstupem vektor $(1.0, -1.0, -1.0, -1.0, -1.0)$.

Výběr vstupních intervalů hodnot a trénovaných výstupních vektorů je nutno dobře koordinovat s použitou přenosovou funkcí. V této práci je použita následující funkce (obrázek 2.1 (d)) zmíněná v [3].

$$\varphi(x) = 1.7519 \cdot \tanh(\frac{2}{3} \cdot x) \tag{3.1}$$

Kombinace takto tvořených vstupů, výstupů a přenosové funkce se ukázala jako lepší ze zkoumaných možností.

První varianta používala jako přenosovou funkci sigmoidu (obrázek 2.1 (b)):

$$\varphi(x) = \frac{1}{(1 + \exp(-\lambda \cdot x))} \tag{3.2}$$

Jedná se o často uváděnou základní přenosovou funkci neuronových sítí a proto od ní původně vycházela i tato práce. Obor hodnot této funkce pro $\lambda = 1$ je definován intervalem $(0, 1)$. Z toho vyplývá podoba vstupů a výstupů v podobných mezích. Neuronové sítě řešící tuto úlohu s touto přenosovou funkcí a daty v této podobě se velmi špatně učily rozpoznávat znaky a proto použití těchto parametrů nelze doporučit.

3.2 Struktura neuronové sítě

Nejdříve byly pro systém rozpoznávání použity plně zapojené dopředné sítě. Tyto sítě nepodávaly dobré výsledky zejména při rozpoznávání vzorů na kterých nebyly přímo trénovány.

Z tohoto důvodu byla snaha sítím místo bitmap předkládat extrahované charakteristiky symbolů, které by byly nezávislé na otočení, posunutí a jiných nedostatcích zaznamenaných znaků. Tento postup byl částečně úspěšný pro dostatečný počet charakteristik u malého počtu tříd symbolů, ale pro rostoucí

počet tříd ztrácel na účinnosti a extrahování většího množství možných charakteristik bylo velmi náročné.

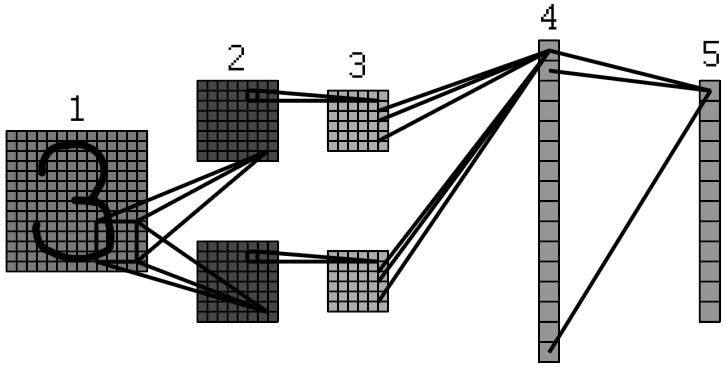
Kombinace těchto variant, tedy předávání plně zapojené dopředné síti na vstupu vektorizovanou bitmapu spolu s vektorem charakteristik podávalo lepší výsledky při klasifikaci znaků, nicméně ani to nebylo dostatečně dobré pro velký počet tříd symbolů.

Z těchto důvodů byla vybrána tzv. konvoluční neuronová síť.

3.2.1 Konvoluční neuronová síť

Konvoluční neuronová síť je dopředná neuronová síť specifická svým propojením neuronů mezi vrstvami.

Obrázek 3.1:



Její vrstvy lze, dle způsobu zapojení neuronů do předchozí vrstvy, dělit na tři typy.

- Prvním typem je konvoluční vrstva (2. na obrázku 3.1). Konvoluční vrstva se dále dělí na menší celky, lze je nazvat mapy. Mapa je charakteristická tím, že všechny její neurony zapojené do předchozí vrstvy sdílejí stejné váhy. Na obrázku je neuron z konvoluční vrstvy zapojen do čtverce ve vstupní vrstvě, sousední neuron z konvoluční vrstvy bude zapojen do stejně velkého čtverce ve vstupní vrstvě, pouze posunutého. Tímto zapojením se docílí efektu, že každá mapa z konvoluční vrstvy bude reagovat na stejné charakteristiky po celé ploše vstupního obrazu.
- Druhým typem je vzorkovací vrstva (3. na obrázku 3.1). Vzorkovací vrstvu lze opět dělit na menší celky, mapy. Všechny neurony jedné vzorkovací mapy jsou zapojeny do jedné konvoluční vrstvy obdobným způsobem jako jsou neurony jedné konvoluční mapy zapojeny do vstupní vrstvy. Neurony této vrstvy zpracovávají shodné typy charakteristik sousedících oblastí.
- Třetím typem je plně zapojená vrstva (4. a 5. na obrázku 3.1), každý její neuron je zapojen do všech neuronů předchozí vrstvy.

Toto rozdělení není striktní, nicméně dobře charakterizuje význam a funkci jednotlivých vrstev. Například za vzorkovací vrstvu lze napojit opět konvoluční

vrstvu s libovolným počtem map a neurony každé z map lze zapojit nikoliv pouze do jedné předchozí mapy, ale do libovolného počtu map.

Použitá struktura neuronové sítě je následující.

Vstupní vrstva obsahuje $28 \times 28 = 784$ neuronů, tedy zpracovává bitmapy široké a vysoké 28 pixelů.

Druhá vrstva je konvoluční vrstva o 8 mapách velikosti 12×12 neuronů napojených na čtverce velikosti 6×6 ve vstupní vrstvě. Sousední neurony v mapě konvoluční vrstvy jsou napojeny na překrývající se čtverce ve vstupní vrstvě o 4 pixely v šířce nebo výšce.

Třetí vrstva obsahuje 30 konvolučních map velikosti 6×6, kde každý neuron je napojen do všech 8 konvolučních map. Čtverec zapojení je velikosti 2×2 a sousední čtverce přiléhají těsně, nepřekrývají se.

Čtvrtá vrstva je plně zapojená o velikosti 100 neuronů a každý její neuron je zapojen do všech neuronů předchozí vrstvy.

Poslední vrstva je velká v závislosti na počtu rozpoznávaných symbolů a je opět plně zapojená do předchozí vrstvy.

3.3 Inicializace

Počáteční inicializace vah sítě je sice náhodná, ale váhy by měly být zvoleny takovým způsobem, aby jejich hodnoty nebránily procesu učení. Podstatné je, aby střední hodnota excitace neuronu byla rovna nule. Velikost ideální směrodatné odchylky je určená použitou přenosovou funkcí, resp. její derivací a střední hodnotou vstupních vektorů. Pokud jsou vstupy normalizované na střední hodnotu 0, je dle [3] vhodné generovat hodnoty vah pro neuron z rozdělení se střední hodnotou 0 a směrodatnou odchylkou $\delta = \frac{1}{\sqrt[2]{m}}$, kde m je počet vstupujících spojení. Díky tomu budou mít i výstupní hodnoty střední hodnotu rovnou 0, stejně jako vstupní.

3.4 Učení v praxi

Pro učení sítě byl použit stochastický algoritmus Levenberga-Marquardta. Tento algoritmus vyžaduje krom již popsaných detailů ještě několik parametrů. Velmi důležitým, nicméně velmi těžko odvoditelným jinak než empirickým způsobem je parametr rychlosti učení η. V některých publikacích je parametr η odvozován z velikosti trénovací množiny, jindy náhodně osciluje kolem zvolené konstanty v závislosti na vývoji chybové funkce sítě. Vzhledem k menšímu počtu vrstev než je obvykle běžné pro konvoluční sítě a k použitém algoritmu, který nastavuje parametr učení $\eta_{i,j}$ idividuálně pro každou váhu, nebylo třeba pro rychlou konvergenci sítě k řešení, složitého určování globalního parametru η, ale byl nejčastěji volen z intervalu $(0.01, 0.00001)$ a byl v každé epoše postupně snižován.

Počet vzorů použitých pro výpočet aproximace matice druhé derivace chybové funkce byl volen jako 10% velikosti trénovací množiny. Navíc nebyla tato matice a příslušné koeficienty přepočítávány každou epochu, nýbrž pouze jednou za několik epoch. Toto zjednodušení velmi zrychlí výpočet aniž by zhoršovalo výsledky učení.

3.5 Výsledky

Schopnosti sítě byly ověřeny na množině ručně psaných číslic MNIST (viz. [8]). Síť se učila číslice od 0 do 9 na 60000 vzorech. Výsledkem učení byla $97,4\%$ úspěšnost na validační množině číslic o 10000 vzorech.

Pro praktické použití byla síť trénována pro rozpoznání 49 symbolů

$$0-9, A-Z, +-*/ <>= ()[]\Pi\Sigma$$

za účelem rozpoznávání ručně psaných matematických výrazů. Obtížným bylo obstarání vhodných dat. Nakonec bylo síti předloženo 10 vzorů od každého symbolu, z nichž se níže popsaným způsobem vytvořilo 1000 vzorů od každého z nich. Validační množina byla tvořena z jiných 10 vzorů, z nichž se poškozením vytvořilo 100 vzorů od každého. Tedy síť byla trénována na 49000 vzorech a validována na 4900 vzorech. Bylo dosaženo 96% úspěšnosti na validační množině. Velmi podstatným prvkem takové úspěšnosti je fakt, že ačkoliv byl počet vzorů pětinásobný oproti databázi MNIST, byly symboly psané jednou osobou.

Symboly jsou před trénováním i praktickým rozpoznáváním zmenšeny se zachováním poměru výšky a šířky na velikost 20×20 pixelů a orámovány do čtverce 28×28. Nakonec jsou symboly v bitmapě vycentrovány

3.5.1 Příprava dat

Zásadní prvek při učení neuronové sítě jsou použitá trénovací data. Motivace u rozpoznávání vzorů je zřejmá. Síť by měla správně klasifikovat jak vzory předložené při procesu učení, tak i vzory nepředložené avšak podobné těm z trénovací množiny. Měla by tedy umět dobře zobecňovat. Z tohoto důvodu je třeba, aby síť rozeznávala vzory podle skutečně charakteristických vlasností cizích jiným třídám vzorů. Takovému naučení velmi pomůže dostatečně velká a rozmanitá trénovací množina.

Je tedy potřeba velká množina symbolů, řádově tisíce, spolu s jejich označením. Získat takovou množinu se specifickými symboly, navíc ve vhodném formátu, není snadné. Vytvořit množinu takového objemu ručně je ještě obtížnější, avšak tento proces lze z velké části automatizovat.

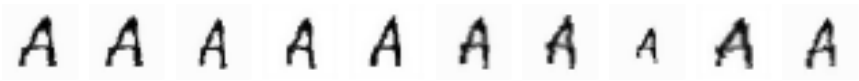
Použité řešení v této práci je inspirované [7]. Prvním krokem je získání alespoň omezené sady vstupních symbolů. Druhým krokem je generování náhodně deformovaných symbolů z této omezené sady. Symboly se deformují následovně.

- Globálním horizontálním a vertikálním posunem.
- Horizontálním a vertikálním posunem různým pro každý pixel a globálně vyhlazeným Gaussovským filtrem, díky čemuž poškození působí elasticky.
- Horizontální a vertikální změnou rozměrů.
- Rotací kolem středu s náhodnou odchylkou.

Principem Gaussovského filtrování je výpočet nové hodnoty pixelu v závislosti na průměru hodnot okolních pixelů, váženém vzdáleností pixelů dle Gaussovské křivky a rozdílu hodnot pixelů opět dle Gaussovské křivky. Algoritmus je bohužel výpočetně relativně náročný, z tohoto důvodu není použit dvourozměrný filtr, ale obraz je filtrován nejdříve horizontálně jednorozměrným filtrem a následně

stejným filtrem vertikálně. Výsledky jsou srovnatelné s původním postupem, náročnost běhu je řádově menší.

Obrázek 3.2:



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j)

Na obrázku 3.2 jsou příklady náhodně deformovaných symbolů získaných z jednoho vzoru.

4. Segmentace

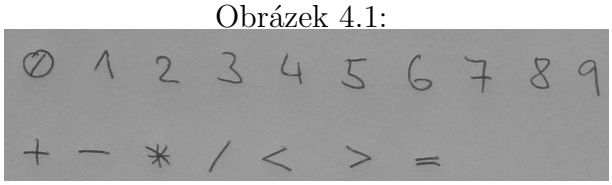
4.1 Problém segmentace

Tato práce se zabývá rozpoznáváním jednotlivých ručně psaných symbolů pomocí neuronových sítí. A vzhledem k tomu, že výstupem práce má být i software řešící stejný úkol, budou zřejmě neuronové síti předkládána data z různých zdrojů v rozdílném stavu. K tomu je zapotřebí univerzální postup, jakým data pro síť předzpracovat. Vstupem se budou rozumět data se zaznamenanými symboly v běžně používaných obrazových formátech (png, jpg, bmp). Úkolem segmentace bude získat z obrazu jednotlivé symboly i přes přítomný šum, nedokonalý kontrast a ostrost.

4.2 Algoritmus segmentace

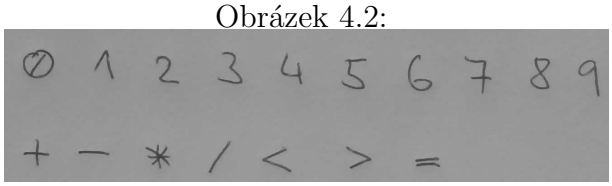
Celý algoritmus bude znázorněn kompletním provedením na vzorovém vstupu.

Obrázek 4.1:



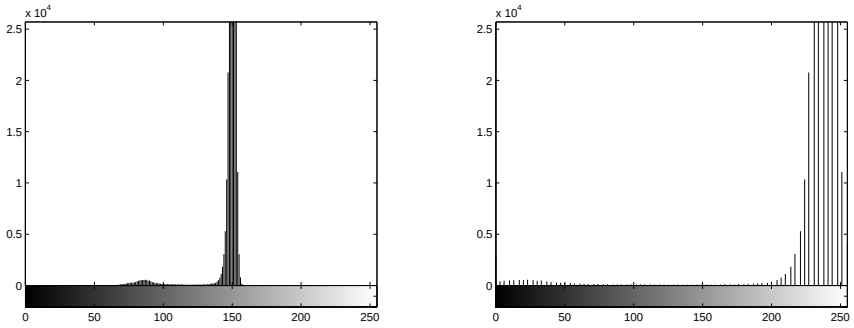
Prvním krokem je odstranění bílého šumu při zachování ostrosti hran. Bylo použito Gaussovské filtrování obrazu.

Obrázek 4.2:



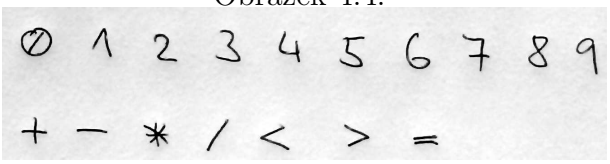
Druhým krokem je zlepšení kontrastu obrazu. V tomto kroku je třeba blíže se podívat na histogram barev a upravit ho následovně.

Obrázek 4.3:



(a) (b)

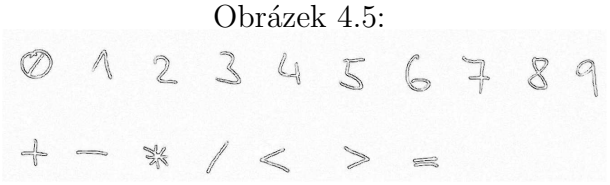
Obrázek 4.4:



Na obrázku 4.3 (a) je zobrazen histogram obrazu před úpravou, na obrázku 4.3 (b) po úpravě. Je vidět, že nový obraz lépe využívá celé barevné palety a mezi obrázky 4.2 a 4.4 došlo ke znatelnému zlepšení kontrastu.

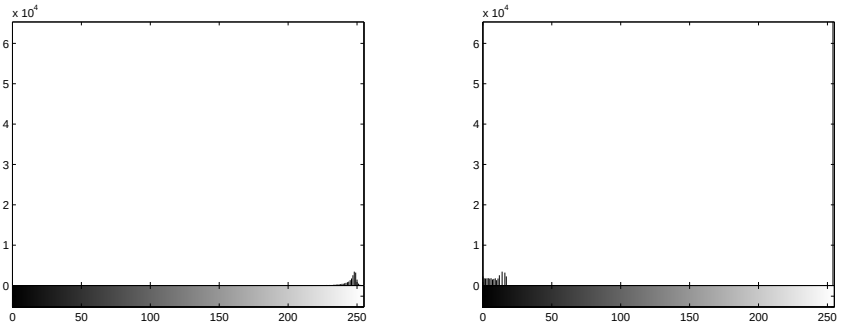
Třetím krokem je detekování hran v obrazu, tedy velkých rozdílu v kontrastu blízkých pixelů. Zde je použito filtrování podle [5].

Obrázek 4.5:



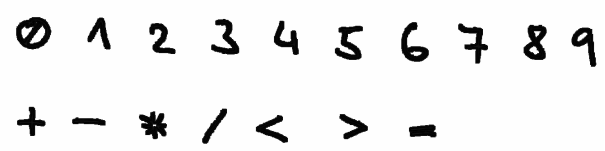
Čtvrtým krokem je ekvalizace histogramu hran. Ještě předtím je ale třeba opět filtrovat šum a velmi málo výrazné hrany, zhruba spodních pět procent škály. Tím se odstraní například šum způsobený kazy papíru nebo špatným záznamem.

Obrázek 4.6:



(a) (b)

Obrázek 4.7:



Na obrázku 4.6 (a) je histogram obrazu před úpravou, na obrázku 4.6 (b) pak histogram po úpravě.

Patým krokem je segmetace souvislých symbolů z celku. Pro tento proces je použit modifikovaný algoritmus [6] počítačového vidění.

Definice 5

- *Hodnota hrany* $e = (u_i, u_j) \in E$ *grafu* $G = (V, E)$ *je funkce* $w(e) = |pixel(u_i) - pixel(u_j)|$.
- *Komponenta grafu* $G = (V, E)$ *je podgraf* $C = (V', E')$ *grafu* G, *kde* $V' \subset V$ *a* E' *tvoří minimální kostru na* V' *podle hodnot hran.*
- *Interní diference komponenty* C *je funkce* $indif(C) = \max_{e \in E'} w(e)$.

Algoritmus 4

1. *Postav graf* $G = (V, E)$ *nad obrazem tak, že vrcholem je každý pixel obrazu, a množinu MC všech komponent incializuj tak, že každá komponenta obsahuje exkluzivně právě jeden vrchol. Hrany obecně spojují sousední vrcholy.*
2. *Seřad hrany vzestupně podle rozdílu intenzit pixelů jimi spojených.*
3. *Vyber první hranu* $e = (u_i, u_j), u_i \in C_i, u_j \in C_j$ *a pokud platí.*

$$C_i \neq C_j \tag{4.1}$$

$$w(e) <= \min\left(indif(C_i) + \tau, indif(C_j) + \tau\right) \tag{4.2}$$

   Pak $C_i = (V_i \cup V_j, E_i \cup E_j \cup \{e\})$ *a* C_j *vyřaď z MC a pokračuj.*
4. *Pokud existuje stále nezpracovaná hrana, přejdi na 3.*

5. Vrať množinu MC disjunktních komponent obrazu.

Nejdůležitějším prvkem algoritmu je postup pro tvorbu hran grafu a volba parametru tolerance τ.

Tento algoritmus byl navržen pro účely počítačového vidění nad vstupy malého rozlišení. Na vstupech jsou obvykle propojovány všechny sousední hrany a výstupem jsou komponenty barev plynulých přechodů.

V této implementaci však není graf stavěn nad původním obrazem ale nad obrazem s detekovanými hranami a výstupem by měly být komponenty tvořené výraznými hranami, tedy tmavé fragmenty obrazu. Proto jsou v grafu propojeny pouze vrcholy s dostatečnou tmavostí (ostrostí hran), tím je i snížena výpočetní náročnost. Nejsou propojovány pouze sousední vrcholy, ale i vrcholy ve větší vzdálenosti. Účelem takového zapojení je segmentace vstupů se sníženou ostrostí, kdy je linie hrany okolo symbolu přerušovaná. Parametr τ je v původní implementaci nerostoucí funkcí velikosti komponenty, v tomto případě se osvědčila konstanta v hodnotě menší než pět procent šířky barevné škály.

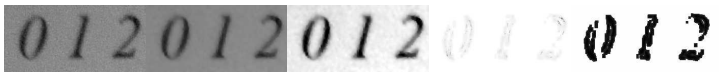
Posledním krokem algoritmu je tvorba samotných výřezů se symboly z původního obrazu. Jejich polohu určují komponenty grafu o více než dvou vrcholech.

4.3 Výsledky algoritmu

Při vývoji algoritmu bylo cílem navrhnout zcela autonomní postup pro segmentaci obrazu bez dalších parametrů a výsledky pro data zaznamenaná kompaktním fotoaparátem nebo skenerem jsou dobré.

Přesto, pokud by algoritmus znal další parametry ohledně vstupu, poradil by si i s více zašuměnými nebo rozmazanými daty, zde je příklad.

Obrázek 4.8:



(a) (b) (c) (d) (e)

Kdyby měl algoritmus určité informace o relativní velikosti symbolů nebo jejich mezer, mohl by v grafu spojit vrcholy na větší vzdálenost a i přes rozmazaný vstup by správně segmentoval číslice v celku.

Algoritmus by si mohl poradit i s tištěným textem, kde jsou často jen velmi malé, občas prakticky nulové, mezery mezi znaky. V takovém případě se při testech jako dobrý postup ukázalo několikásobné zvětšování obrazu spolu s opakovaným Gaussovským filtrováním. Vzhledem k velké náročnosti výpočtu by však bylo vhodné takovéto zvětšování a segmentaci na zvětšeném obrazu provádět pouze lokálně a po částech obrazu.

Tento algoritmus je výsledkem výzkumu této práce. Algoritmus dobře segmentuje ručně psané symboly zaznamenané fotoaparátem nebo skenerem. Avšak je zde i spoustu oblastí ke zdokonalení. Předně, jedná se o rychlost a

paměťovou náročnost pro vstupy s velkým rošlišením a velkým počtem symbolů. V takové případě je velké množství operací a budování grafu nad velkým počtem pixelů opravdu náročné, ale na konečný výsledek algoritmu toto nemá negativní vliv. Zároveň tento algoritmus není určen pro zpracování dat v reálném čase, jeho užití je spíše v dávkovém zpracování většího množství dat a zde se větší výpočetní náročnost dá tolerovat.

5. Aplikace

5.1 Implementace

Implementace této práce je napsaná v jazyce C++. Použita je knihovna OpenCV pro práci s obrazovými formáty a knihovna Boost pro manipulaci se soubory a pro generování pseudonáhodných čisel. Níže jsou v bodech uvedeny hlavní implementované části.

- Obecnou strukturu dopředných neuronových sítí s plně zapojenými, konvolučními a vzorkovacími vrstvami a algoritmy pro tvoření takto propojených vrstev.
- Algoritmus zpětného šíření.
- Stochastický algoritmus Levenberga-Marquardta.
- Algoritmus pro generování elasticky poškozených obrazů a datasetů pro učení sítí.
- Všechny funkce a algoritmy popsané v kapitole segmentace zpracovávající obrazová data.

5.2 Knihovna

Přiložený program v podobě zdrojových kódů je možné využít jako knihovnu nebo základ pro segmentaci a rozpoznávání symbolů nebo jako knihovnu pro tvorbu a učení dopředných neuronových sítí. Rozpoznávání postavené na této aplikaci by se díky dobrému generátoru mohlo například učit písmo konkrétních lidí a o to lépe rozpoznávat jejich rukopis. Ve spojení s analyzátorem matematických výrazů a jejich převodem do typografického formátu vznikl velmi užitečný nástroj.

5.3 Program

Zároveň je možné program použít jako nástroj pro převod naskenovaných nebo vyfocených ručně psaných matematických výrazů do vektorového formátu v latexu a následně například do formátu pdf. Program je dodán s dvěmi naučenými neuronovými sítěmi. První z nich je naučena na rozpoznání pouze ručně psaných číslic, druhá je naučena pro již popsanou množinu 49 symbolů pro matematické výrazy. Uživatelská a programátorská dokumentace v příloze 7.1 a 7.2.

6. Závěr

Cílem práce bylo navrhnout postupy a algoritmy pro segmentaci a rozponávání textu.

Rozpoznávání textu mělo být od začátku řešeno neuronovými sítěmi, ale ani jejich struktura nebo algoritmy učení nebyly pevně stanoveny. V této práci se postupně přešlo od použití plně zapojených dopředných sítí s rozpoznáváním přímo z bitmap, přes exktrakci charakteristik ze znaků, až po kombinaci těchto metod. Nakonec byly tyto postupy nahrazeny použitím konvolučních neuronových sítí. Důvodem byly špatné výsledky rozpoznávání takto použitých neuronových sítí. V případě exktrace charakteristik se jednalo o velmi náročnou metodu, jejíž výsledky jsou dosažitelné konvolučními neuronovými sítěmi s mnohem menším úsilím.

Algoritmus segmentace textu prošel také značným vývojem. Zpočátku byla použita primitivní metoda binarizace pouze podle intenzity barvy. Postupně byly přidávány metody potlačující šum obrazu a zvýrazňování kontrastu. Dalším krokem bylo přidání metody detekce hran. A nakonec zpracování takto předpřipraveného obrazu takovým způsobem, aby bylo možné celistvě segmentovat i méně ostré symboly bez souvislé ohraničující hrany.

Výsledná implementace především demonstruje schopnosti a potenciál těchto metod a lze ji využít jako základní stavební kámen aplikace zpracovávající matematické výrazy do typografického formátu.

Seznam použité literatury

[1] Piction, Phil. *Neural Networks*. Second Edition. PALGRAVE: 2000. ISBN 13 : 987 0333 8028 78

[2] Rojas, Raúl. *Neural Networks A Systematic Introduction*. Springer-Verlag: Berlin 1996. ISBN 978-3540605058

[3] LeCun, Yann. Bottou, Leon. Orr, B. Genevieve. Müller, Klaus-Robert. *Efficient BackProp*. Springer: Berlin 1998.

[4] LeCun, Yann. Bottou, Leon. Bengio, Yoshua. Haffner, Patrick. *Gradient-Based Learning Applied to Document Recognition*. 1998.

[5] Lindeberg, Tony. *Discrete derivative approximations with scale-space properties*. 1993.

[6] Felzenszwalb, F. Pedro. Huttenlocher, P. Daniel. *Efficient Graph-Based Image Segmentation*. 2004.

[7] Simard, Y. Patrice. Steinkraus, Dave. Platt, C. John. *Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis*. 2003.

[8] `http://yann.lecun.com/exdb/mnist/`

[9] `http://en.wikipedia.org/wiki/`
`/List_of_optical_character_recognition_software`

[10] `http://en.openocr.org/`

[11] `http://jocr.sourceforge.net/`

[12] `http://www.gnu.org/software/ocrad/`

[13] `http://code.google.com/p/tesseract-ocr/`

7. Přílohy

7.1 Uživatelská dokumentace

Program se spouští v příkazové řádce s těmito argumenty:

```
./ocr.exe ocr arg1 arg2 arg3 arg4 arg5 arg6
```

$arg1$	uložená neuronová sít
$arg2$	obraz pro rozpoznání
$arg3$	cílový souboru se souřadnicemi a kódy znaků, při hodnotě 0 se nic neukládá
$arg4$	soubor mapující kódy znaků na latexové symboly
$arg5$	vektorový výstup v latexu, při hodnotě 0 se nic neukládá
$arg6$	adresář pro uložení procesu segmentace, při hodnotě 0 se nic neukládá

- Formát souboru se souřadnicemi znaků je následující:

  ```
  [výška obrazu] [šířka obrazu]
  [počet symbolů]
  [x1] [y1] [x2] [y2] [kód symbolu]
  ...
  [x1] [y1] [x2] [y2] [kód symbolu]
  ```

- Vektorový výstup do formátu latex využívá balíčku „textpos“. Pro převod souboru do formátu pdf je třeba spustit:

  ```
  pdflatex vystup.tex
  ```

7.2 Programátorská dokumentace

Aplikaci lze rozdělit na tři funkční části.

- Neuronové sítě – struktura, vytváření a algoritmy
- Zpracování obrazu
- Segmentace obrazu

7.2.1 Neuronové sítě - struktura

V implementaci neuronových sítí byl kladen důraz na jejich čistý návrh, kdy struktura neuronové sítě je pouze nosičem své topologie, tedy svých vrstev, spojeních mezi nimi a jejich vahami. Všechy algoritmy pracující s neuronovou sítí jsou postaveny na jejím vnějším rozhraní.

Návrh struktur tvořících neuronovou síť:

```
class NeuralNetwork
{
public:
 NeuralNetwork();
 NeuralNetwork( const NeuralNetwork& network);
 ~NeuralNetwork();
 void add( NNLayer* layer);
 void del( uint32_t index);
 layers_t::iterator begin();
 layers_t::iterator end();
 layers_t::const_iterator begin() const;
 layers_t::const_iterator end() const;
 layers_t::reverse_iterator rbegin();
 layers_t::reverse_iterator rend();
 layers_t::const_reverse_iterator rbegin() const;
 layers_t::const_reverse_iterator rend() const;
 uint32_t size() const;
private:
 layers_t layers_;
};
```

```
typedef std::vector< NNLayer*> layers_t;

class NNLayer
{
public:
 NNLayer();
 virtual ~NNLayer();
 neurons_t neurons_;
 weights_t weights_;
};
```

```
typedef std::vector< NNNeuron* > neurons_t;
typedef std::vector< NNWeight*> weights_t;
typedef std::vector< NNConnection*> connections_t;

struct NNNeuron
{
 NNNeuron();
 connections_t connections_;
};

struct NNWeight
{
 NNWeight();
 double value_;
};

struct NNConnection
{
 NNConnection( uint32_t neuronid, uint32_t weightid);
 uint32_t neuronid_;
 uint32_t weightid_;
};
```

Vrstvy jsou tvořeny takovým způsobem, aby na sobě nebyly fyzicky závislé, tedy neexistují mezi nimi přímé ukazatele a každá vrstva je technicky nahraditelná vrstvou o stejném počtu neuronů. Každá vrstva spravuje vektory svých neuronů a svých spojení. Každý neuron pak spravuje svá spojení do vyšší vrstvy, spojení je popsáno indexem neuronu ve vyšší vrstvě a indexem příslušné váhy tohoto spojení. Váha není přímo součastí spojení, protože zejména v konvolučních sítích několik spojení sdílí stejnou váhu.

7.2.2 Neuronové sítě - vytváření

Při vytváření neuronových sítí, zejména při tvorbě spojení v konvolučních neuronových sítích, je třeba krom počtu neuronů ve vyšší vrstvě znát i další věci týkající se struktury, například počet konvolučních map a jejich zapojení. To proto, aby bylo možné do těchto map zapojit další vzorkovací nebo jiné konvoluční vrstvy. Z toho důvodu jsou při budování neuronových sítí užívány třídy odvozené od základní třídy reprezentující vrstvu sítě. A tyto odvozené třídy obsahují právě potřebné údaje pro tvorbu složitých zapojení a dále implementují metody pro takovéto zapojení na základě jednoduchých parametrů popisujících topologii sítě.

Funkce budující strukturu sítě použitou v této práci:

```
NeuralNetwork* network = new NeuralNetwork();

InputLayer* input =
 new InputLayer( 28, 28);

ConvLayer* hidden1 =
```

```
 input->append_ConvLayer( 8, 12, 12, 6, 6, 2, 2);

std::vector< std::vector< uint32_t>> p;
 std::vector< uint32_t> tmp;
 for ( uint32_t i = 0; i < hidden1->map_count_; ++i {
  tmp.push_back( i);
 }
 uint32_t map_count = 30;
 for ( push_back i = 0; i < map_count; ++i) {
  p.push_back( tmp);
 }

ConvLayer* hidden2 =
 hidden1->append_ConvLayer( p, 6, 6, 2, 2, 2, 2);

FullLayer* hidden3 =
 hidden2->append_FullLayer( 100);

FullLayer* output =
 hidden3->append_FullLayer( 49);

network->add( input);
network->add( hidden1);
network->add( hidden2);
network->add( hidden3);
network->add( output);

return network;
```

7.2.3 Neuronové sítě - algoritmy

Vstupy a výstupy neuronové sítě ve všech algoritmech jsou vektory hodnot typu double. Všechny pomocné struktury pro odkládání dat během výpočtu výstupu a během učení jsou předem inicializované a proto do většiny funkcí vstupuje mnoho vektorů. Podstatné funkce implementující algoritmy popsané v této práci jsou:

```
typedef std::vector< double> vio_t;
typedef std::vector< vio_t> vvio_t;
```

Výpočet výstupu neuronové sítě:

```
void
 compute(
  NeuralNetwork& network,
  const vio_t& vinput,
  vvio_t& vvnet);
```

Struktura vvnet obsahuje výstupy všech neuronů po vrstvách.

Výpočet algoritmu zpětného šíření pro jeden vzor:

```
void
 backprop(
  NeuralNetwork& network,
  const vio_t& vinput,
  const vio_t& voutput,
  const vvio_t& vvnet,
  const vvio_t& vvwcoef,
  const double eta,
  vvio_t& vvdelta);
```

Struktura vvwcoef obsahuje $\eta_{i,j}$ individuální koeficienty učení algoritmu Levenberga-Marquardta, struktura vvdelta obsahuje změny pro daný vzor vinput a správný výstup voutput.

Výpočet aproximované matice druhých derivací pro algoritmus Levenberga-Marquardta:

```
void
 hessian(
  NeuralNetwork& network,
  const vvio_t& vvinput,
  const vsize_t vindex,
  const size_t size,
  const double mu,
  vvio_t& vvnet,
  vvio_t& vvwcoef,
  vvio_t& vvwder,
  vvio_t& vvxder,
  vvio_t& vvyder);
```

Struktury vvwder, vvxder a vvyder jsou pomocné struktury.

Funkce pro komplexní trénink sítě:

```
void
 train(
  NeuralNetwork& network,
  const vvio_t& vv_train_input,
  const vvio_t& vv_train_output,
  const vvio_t& vv_validation_input,
  const vvio_t& vv_validation_output,
  double eta,
  double eta_decay,
  double eta_min,
  double mse_stop,
  uint32_t epoch_max,
  uint32_t hessian_size,
  uint32_t hessian_max,
  uint32_t fails_max);
```

Tato funkce řídí proces učení, předává sítí vzory, v určených intervalech přepočítává koeficienty $\eta_{i,j}$, měří chybovou funkci na trénovací a validační

množině, zastavuje proces učení v případě dosažení maxima epoch nebo stanoveného minima chybové funkce.

7.2.4 Zpracování obrazového vstupu

Základní struktura pro reprezentaci černobílého obrazu v programu:

```
class image_t
{
public:
 const static uint8_t WHITE = 255;
 const static uint8_t BLACK = 0;
public:
 image_t( uint32_t height, uint32_t width, uint8_t val);
 uint8_t& operator[] ( uint32_t index);
 uint8_t operator[] ( uint32_t index) const;
 uint8_t& operator() ( uint32_t h, uint32_t w);
 uint8_t operator() ( uint32_t h, uint32_t w) const;
 bool in( uint32_t h, uint32_t w) const;
 uint32_t height() const;
 uint32_t width() const;
 uint32_t size() const;
 std::vector< uint8_t> data() const;
 void data( const std::vector< uint8_t>& val);
 uint32_t sum() const;
private:
 uint32_t height_;
 uint32_t width_;
 std::vector< uint8_t> data_;
};
```

Načítání a ukládání obrazu do formátů (png, jpg, bmp) je řešeno externí knihovnou OpenCV. Žádné algoritmy pro zpracování obrazu z externích knihoven nejsou použity. Nejvýznamější je funkce pro deformaci obrazu:

```
image_t
 distort(
  const image_t& image,
  const uint32_t distortion,
  const uint32_t movement,
  const uint32_t scalling,
  const uint32_t rotation,
  const uint32_t seed);
```

Distortion je limit náhodného posunutí pixelů, movement je limit pro celkové posunutí, scalling je limit pro změnu rozměrů a rotation limit pro rotaci ve stupních. Tato funkce nejdříve na základě argumentů a pseudonáhodných hodnot vytvoří mapu relativní deformace jednotlivých pixelů obrazu a nakonec dopočítá nové hodnoty pixelů.

7.2.5 Segmentace obrazu

Velmi podstatnou součástí implementace segmentace je struktura efektivně spravující příslušnost vrcholu ke komponentě:

```
class disjoint_set_forest
{
 struct member_t
 {
  member_t( uint32_t id, uint32_t parent_id);
  uint32_t id_;
  uint32_t parent_id_;
  uint32_t rank_;
 };

public:
 disjoint_set_forest();
 void create(uint32_t count);
 uint32_t find( uint32_t member_id);
 std::pair< uint32_t, uint32_t>
  merge( uint32_t set_id_a, uint32_t set_id_b);
 uint32_t size();
 void clear();

private:
 std::vector< member_t> member_v_;
 uint32_t count_;
};
```

Hlavička funkce segmentující obraz:

```
imagepos_vt
 segment(
  const image_t& source_image,
  const image_t& graph_image,
  const uint32_t min_size = 0,
  const uint32_t edge_range = 3,
  const uint8_t background = UINT8_MAX,
  const uint8_t min_darkness = UINT8_MAX,
  const uint8_t max_difference = UINT8_MAX);
```

Sourceimage je originální vstupní obraz, graphimage je obraz s detekovanými hranami, minsize je minimální přijímaná velikost komponenty, tedy symbolu, edgerange je vzdálenost, na kterou se spojují pixely.

7.2.6 Formát struktur v souborech

Neuronová síť je uložena takto:

```
["layers"] [počet vrstev]
```

```
... pro všechny vrstvy ...
["------------------------------"]
["neurons"] [počet neuronů ve vrstvě]

... pro všechny neurony ...
["connections"] [počet spojení neuronu]
... pro všechna spojení ...
[id neuronu ve vyšší vrstvě] ["w"] [id váhy] ["|"]

["weights"] [počet vah ve vrstvě]
... pro všechny váhy ...
[hodnota váhy]
```

7.2.7 Kompilace ze zdrojových kódů

U bakalářské práce jsou přibalené zdrojové kódy spolu se všemi potřebnými knihovnami nutnými pro sestavení.

Skript pro sestavení programu vyžaduje vývojové prostředí Visual Studio 2010. Sestavení se spustí příkazem.

```
devenv /build release ocr.sln
```