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

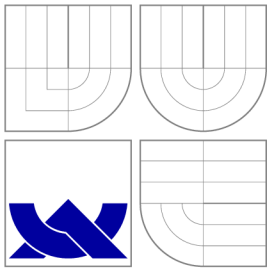
ROZPOZNÁVÁNÍ RUČNĚ PSANÉHO PÍSMA POMOCÍ NEURONOVÝCH SÍTÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

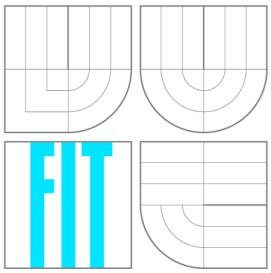
AUTOR PRÁCE
AUTHOR

Bc. VLADIMÍR HORKÝ

BRNO 2012



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

ROZPOZNÁVÁNÍ RUČNĚ PSANÉHO PÍSMA POMOCÍ NEURONOVÝCH SÍTÍ

HANDWRITTEN CHARACTER RECOGNITION USING ARTIFICIAL NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VLADIMÍR HORKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. OLDŘICH PLCHOT

BRNO 2012

Abstrakt

V této práci budou představeny neuronové sítě, konkrétně algoritmus zpětného šíření chyby. Bude vyloženo teoretické pozadí algoritmu a budou zde řešeny problémy, se kterými se můžete setkat při učení takovéto sítě. Práce se také zabývá předzpracováním obrazu a obrazovými příznaky, které jsou hlavním stavebním kamenem klasifikace. Část práce se také zabývá experimenty s neuronovou sítí nad zvolenými obrazovými příznaky. Součástí práce je také vytvoření demo-aplikace pro experimenty s neuronovými sítěmi a pro převod textu v obraze na text elektronický.

Abstract

Neural networks with algorithm back-propagation will be presented in this work. Theoretical background of the algorithm will be explained. The problems with training neural nets will be solving there. The work discuss some techniques of image preprocessing and image extraction features, which is one of main part in classification. Some part of work discuss few experiments with neural nets with chosen image features.

Klíčová slova

Neuronové sítě, OCR, ICR, Zpracování obrazu, Analýza hlavních komponent, PCA, Gradient Structural and Concavity, GSC, Klasifikace, Zpětné šíření chyby, Freemanův řetězcový kód, Segmentace, Houghova transformace, Detekce hran.

Keywords

Neural nets, OCR, ICR, Image processing, Principal component analysis, PCA, Gradient Structural and Concavity, GSC, Classification, Back-propagation, Freeman's chain code, Segmentation, Hough transformation, Edge detection.

Citace

Vladimír Horký: Rozpoznávání ručně psaného písma pomocí neuronových sítí, diplomová práce, Brno, FIT VUT v Brně, 2012

Rozpoznávání ručně psaného písma pomocí neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana ing. Oldřicha Plchota

.....
Vladimír Horký
23. května 2012

Poděkování

Za odbornou pomoc, trpělivé konzultace a jiné cenné rady při zpracování této práce děkuji vedoucímu diplomové práce Ing. Oldřichu Plchotovi.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V dnešní moderní době si život bez elektronického textu nedokážeme ani představit, ať už jej používáme k posílání e-mailů, textových zpráv(SMS) či při vývoji softwaru...

To vede společnosti čím dál více k nahrazení starých papírových kartoték za kartotéky elektronické. Jejich nespornou výhodou oproti textu „na papíře"je totiž možnost použití automatizace, schopnost rychlého a efektivního zpracování dokumentu, snadný převod do papírové podoby, možnost sdílet a využívat informace, které jsou v dokumentech obsaženy apod. Přes tyto výhody má však elektronický text i jednu nespornou nevýhodu, k jeho čtení nám nestačí pouze oči, ale potřebujeme »mašinku« na jeho zobrazení.

Naučit počítače rozpoznávat znaky se experti pokouší již od poloviny padesátých let. V současnosti je rozpoznávání tištěného textu na vysoké úrovni. Výbornou ukázkou stavu je společnost Google, která se v současnosti probírá stohy knihoven a převádí je do elektronické podoby. Jejich cílem je umožnit vyhledávání v knihách stejně tak, jak je dnes možné vyhledávat v internetu [9].

Elektronické rozpoznávání znaků je velice účinný způsob, jak zvýšit užitnou hodnotu skenování. Šetří čas a umožňuje rychlé zadání textu, který byl vytištěn. Elektronické rozpoznávání lze jednoduchým způsobem rozdělit podle specifik vytěžování (automatizovaného čtení) dokumentu. Nejrozšířenější metodou je využití technologie OCR (*z anglického slova Optical Character Recognition – Optické rozpoznávání znaků*), která se používá k vytěžování tištěného písma, k následné analýze textu a přenosu rozpoznaných znaků do příslušného kódování. Rozšířením o technologii ICR (*z anglického slova Inteligent Charakter Recognition – Inteligentní rozpoznávání znaků*) získáme možnost vytěžovat také rukou psaný text.

Tato práce se zaměřuje právě na oblast zpracování ručně psaného textu. Je zaměřena zejména na použití neuronových sítí.

V druhé kapitole budou přiblíženy základy zpracování obrazu. Podstatná část bude věnována morfologickým operátorům a hranovým detektorům, sloužících pro předzpracování obrazu před následnou extrakcí obrazových charakteristik, kterým je věnována 3. kapitola. Jsou zde detailně popsány jednotlivé metody pro získání co nejlepší obrazové informace. Ve 4. kapitole naleznete obecné informace o klasifikátorech. Navazující kapitola je věnována neuronovým sítím. Popisuje její fungování a zmiňuje některé její typy. Je zde také detailně rozebrán trénovací algoritmus zpětného šíření chyby, problémy při učení a jejich řešení. Obsahem 6. kapitoly je návrh výsledné aplikace. Sedmá kapitola popisuje jednotlivé kroky programu při převodu textu v obrázku na text elektronický. Předposlední kapitola diskutuje experimenty při hledání optimálního řešení. Celou práci završuje 9. kapitola, ve které naleznete souhrn této práce. Jsou zde diskutovány výsledky a možný další budoucí vývoj aplikace.

Kapitola 2

Zpracování obrazu

Prvním krokem v rozpoznávání vzorů je „naučit"počítač, jaké jsou třídy vzorů. Učení probíhá formou ukázek příkladů všech tříd, kdy si stroj vytvoří svůj prototyp všech tříd. V průběhu rozpoznávání jsou znaky porovnávány s naučenými daty a podle jejich nejbližší shody jsou přiřazeny do odpovídající třídy.
Toto se děje na základě nějaké znalosti či ze statistických informací. Obecné klasifikační systémy se skládají z několika komponent – získání dat, předzpracování, extrahování příznaků, klasifikace a vyhodnocení rozpoznaných dat. Výsledek klasifikace pak nejvíce ovlivňuje kvalita získaných dat a příznaky z nich získaných. Proto je vhodné nebo naopak žádoucí si tato data předzpracovat.

2.1 Předzpracování obrazu

Po úspěšném získání obrazu a jeho digitalizaci máme k dispozici digitální obraz pozorované scény. Díky způsobu snímání nebo nevhodných podmínkách při jeho průběhu může obraz obsahovat různé nežádoucí efekty, jako je např. šum, rozmazání... Typickým příkladem preprocesních algoritmů je odstraňování šumu, vzorů na pozadí, vypreparování textu, srovnání šikmo psaného textu, korekce velikosti a sklonu znaků. K redukci šumu se používají tzv. *vyhlazovací* filtry. Pro vyhlazování se používají např. binomiální, box, mediánové, Kuwahara filtry, které využívají odlišné algoritmy.

Obecně spočívá filtrace v modifikaci obsahu obrazového bodu (pixelu) s ohledem na nejbližší okolí. Úprava se nejčastěji provádí pomocí čtvercové matice, tzv. **filtrační matice**, kdy se s prvky nejbližšího okolí pixelu a prvky filtrační matice provádí určité operace. Nejjednodušší filtrací je násobení prvků filtrační matice s prvky nejbližšího okolí upravovaného pixelu.

V následujících podkapitolách se budeme věnovat některým přístupům k segmentaci obrazu, neboť úspěšná detekce a segmentace rozpoznávaného vzoru tvoří půl úspěchu.

2.1.1 Prahování

Pro extrakci příznaků potřebujeme obraz v binární podobě, tedy takový obraz, v němž se vyskytují pouze dvě hodnoty pixelu. Pro další strojové zpracování je binární obraz velmi výhodný neboť je obraz rozdělen do dvou vzájemně se nepřekrývajících oblastí.

Jednou z možných metod, jak takový obraz získat, je tzv. *prahování*. Prahování je velmi hojně používaná jasová transformace a patří k nejjednodušším segmentačním postupům. Jedná se o algoritmus na získání prahové úrovně šedi, kdy každému pixelu na pozadí se

přiřadí »0« a pixelu na popředí hodnota »1«. Rozhodování, jakou přiřadit hodnotu danému pixelu, se děje na základě porovnání vlastní hodnoty s tzv. *prahem*. Hodnota tohoto prahu (případně prahů) a způsob porovnání se pak nastavuje podle způsobu prahování.

Při jednoduchém prahování je hodnota prahu fixní a všechny pixely s hodnotou menší než je daný práh jsou nastaveny na »0« a naopak všechny pixely s větší hodnotou jsou nastaveny na »1«.

$$nove(X,Y) \quad = \quad (stare(X,Y) > prah \quad ? \quad MAX : MIN)$$

Při dvojitém prahování se používá rozmezí definované dvěma prahy, v němž se musí hodnota pixelu nacházet, aby získal hodnotu »1«.

$$nove(X,Y) \quad = \quad (prah1 > stare(X,Y) > prah2 \quad ? \quad MAX : MIN)$$

Při těchto algoritmech prahování jsme však postaveni před jedno důležité rozhodnutí a to, jakou hodnotu prahu pro naši úlohu použít, abychom z obrazu získali maximum žádoucích informací.

Hodnotu prahu lze získat experimentálně nebo se používají metody, kdy se tato hodnota nastavuje relativně vzhledem k hodnotám pixelů právě zkoumaného okolí (např. *Adaptivní prahování*). U adaptivního prahování se hodnota prahu nastavuje v závislosti na hodnotách okolních obrazových bodů aktuálního pixelu, vyjádřených lokálním histogramem.

Protože jsem nechtěl porušovat zažitou konvenci při klasifikaci písmen a číslic, kdy číslice je bílá na černém pozadí, použil jsem tzv. *inverzní prahování* – hodnotě menší jak práh nastavíme hodnotu maximální a naopak. Tímto dosáhneme toho, že informaci pro další výpočty nese v obrazu napsaný znak (černá = 0, bílá = 255).

2.1.2 Morfologické transformace

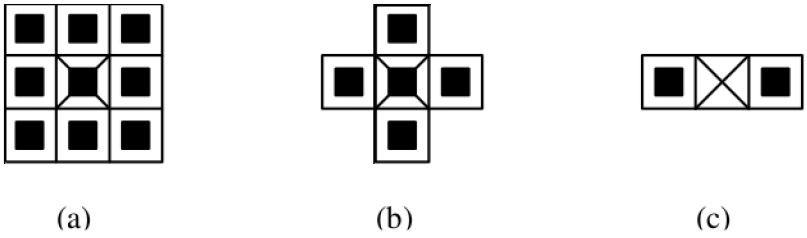
Informace zde popsané jsou čerpány z knih [5, 6, 18].

V názvosloví digitálního zpracování obrazu rozumíme morfologií matematický nástroj, s jehož pomocí lze provádět jak předzpracování, samotnou segmentaci obrazu s důrazem na tvar hledaných objektů nebo kvantitativní popis nalezených objektů. Matematická morfologie využívá vlastností bodových množin, výsledky z integrální geometrie a topologie. Výchozím předpokladem pak je představa, že reálné obrázky lze modelovat pomocí bodových množin libovolné dimenze.

Ve vstupním obraze se bohužel nezřídka setkáváme se šumem, představovaným nežádoucími pixely či jejich skupinami, které nemají návaznost na zobrazované objekty a nenesou obrazovou informaci. Pokud se od objektů dostatečně neodlišují svou jasovou složkou, je možné je odstraňovat na základě jejich velikosti a samostatnosti. Pro tento účel je výhodné využít právě binární morfologické operace.

Binární obraz můžeme vyjádřit jako 2D bodovou množinu a to tak, že body objektů reprezentují množinu X, která odpovídá pixelům s hodnotou jedna, a body doplňku této množiny popisují pozadí a reprezentují se pixely s hodnotou nula. Základním popisným kamenem je tedy množina dvojic celých čísel ($\in Z^2$).

Morfologická transformace Ψ je pak dána relací mezi obrazem (bodová množina X) s typicky menší bodovou množinou tzv. strukturním elementem B (obr. 2.1). Aplikaci morfologické transformace $\Psi(X)$ na obraz X si pak můžeme představit, jako bychom strukturní element B systematicky posouvali po obraze.



Obrázek 2.1: Typické strukturní elementy

Měl bych ještě zmínit, že ke každé morfologické transformaci $\Psi(X)$ existuje transformace duální $(\Psi^*(X)$, která vyplývá z množinového doplňku (rov. 2.1).

$$\Psi(X) \quad = \quad (\Psi^*(X^c))^c \tag{2.1}$$

Základní morfologické transformace jsou *Dilatace*, *Eroze*, *Otevření* a *Uzavření*.

Dilatace a eroze

Dilatace $\oplus$ je operace, která vytváří tzv. Minkowského množinový součet (skládá body dvou množin pomocí vektorového součtu).

> „Dilatace $X \oplus B$ je bodovou množinou všech možných vektorových součtů pro dvojice pixelů, vždy pro jeden z množiny X a jeden z množiny B (rov. 2.2). [6]

$$X \oplus B \quad = \quad \{p \in \varepsilon^2 : \quad p = x + b, \quad x \in X, b \in B\} \tag{2.2}$$

Rovnici (2.2) můžeme také vyjádřit jako sjednocení posunutých bodových množin obrazu a strukturních elementů:

$$X \oplus B \quad = \quad \bigcup_{b \in B} X_b \tag{2.3}$$

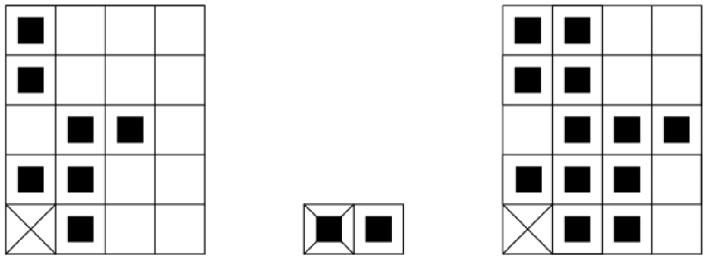
Samostatná dilatace eliminuje izolované díry v objektech a rozšiřuje obrysy objektů na úkor okolního pozadí.

Jestliže se pod jakoukoliv jedničkovou buňkou strukturního elementu nachází černý pixel, pak výsledkem operace je černý pixel. Má-li se zachovat původní rozměr objektů, kombinuje se dilatace společně s erozí, která bude popsána v dalším odstavci.

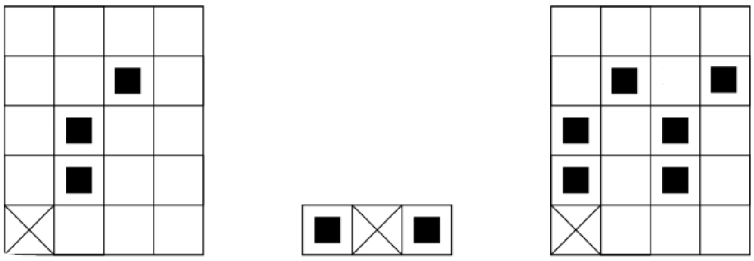
Eroze $\ominus$ je duální morfologickou transformací k dilataci. Jedná se o skládání dvou množin pomocí tzv. Minkowského množinového rozdílu. Eroze skládá množiny podle předpisu:

$$X \ominus B \quad = \quad \{p \in \varepsilon^2 : \quad p + b \in X, \quad \textit{pro každé } b \in B\} \tag{2.4}$$

Tato rovnice (2.4) může být vyjádřena jako průnik všech posunů obrazu X o vektory $-b \in B$ (rov. 2.5).

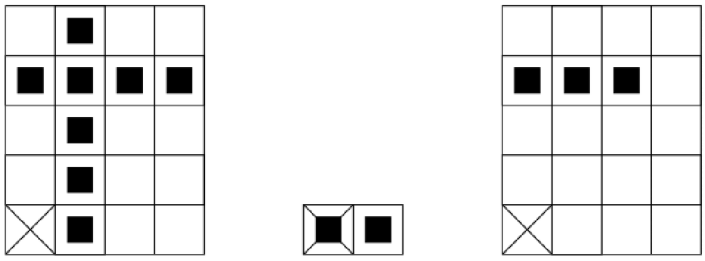


Obrázek 2.2: Dilatace



Obrázek 2.3: Dilatace pro případ, kdy počátek není prvkem strukturního elementu

$$X \ominus B = \bigcap_{b \in B} X_{-b} \tag{2.5}$$



Obrázek 2.4: Eroze

Eroze eliminuje izolované pixely na pozadí a ubírá obrysy objektů. Černý pixel je výsledkem operace jen tehdy, když jsou černé pixely současně pod všemi jedničkovými buňkami strukturního elementu. Ve všech ostatních případech je výsledkem bílý pixel.

Pozor: Dilatace a eroze nejsou navzájem inverzní operace!

Otevření a uzavření

Kombinace eroze a dilatace patří mezi další významné morfologické transformace – *otevření* a *uzavření*. Zde se právě využívá skutečnosti, že eroze a dilatace nejsou navzájem inverzními

transformacemi. **Otevření** mění pouze detaily obrazu a to tak, že přerušuje tenké spoje mezi objekty a zvětšuje mezi nimi v těchto místech mezery. Je to tedy eroze následována dilatací. Na obrázku 2.5 je vidět, že eroze odstraní tenká spojení, čímž se natolik změní tvar objektu, že následující dilatace již není schopna tyto odebrané části zpět přidat.

Obdobně dilataci následovanou erozí nazýváme **uzavřením**. Tato transformace mění opět pouze detaily obrazu, které jsou velikostně srovnatelné s velikostí strukturního elementu. Tvary větších objektů zůstávají přitom nezměněny. Pomocí uzavření můžeme potlačit díry uvnitř objektů, či propojit objekty, které se dříve nedotýkaly.

Obrázek 2.5: Ukázka morfologického otevření a uzavření

Transformace hit or miss

> „Transformace tref či miň (angl. hit or miss) je morfologický operátor, který indikuje shodu strukturního elementu a části obrazu."[6]

Zjednodušeně řečeno pro výsledek operace není důležité jen to, zda jsou na zadaných pozicích černé pixely, ale také to, že na jiných místech jsou naopak požadovány pixely bílé. Do strukturního elementu je tedy, kromě požadavku černých nebo bílých pixelů, nutné zadat, že na pixelech na určitých pozicích naopak vůbec nezáleží. Každá buňka strukturního elementu pak popisuje jeden ze tří možných stavů.

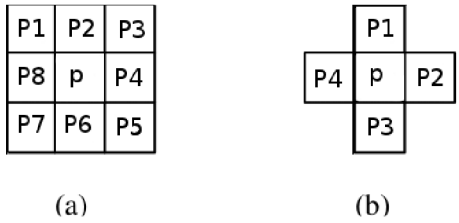
Skeletonizace – ztenčování

Skeletonizace je metoda, která se aplikuje na binární obraz. Cílem teto metody je nalézt skeleton (topologickou kostru) objektu. Skeletonem rozumíme zjednodušený tvar oblasti při zachování její tvarové charakteristiky, čímž redukujeme množství informací. V našem případě hledáme středovou osu objektu, jejíž šíře bude čítat 1 pixel. Při ztenčování se také okrajové pixely odečítají od objektů, podobně jako je to u eroze či otevření, ale nikoliv ty pixely, jejichž zmizení by způsobilo porušení souvislosti daného objektu.

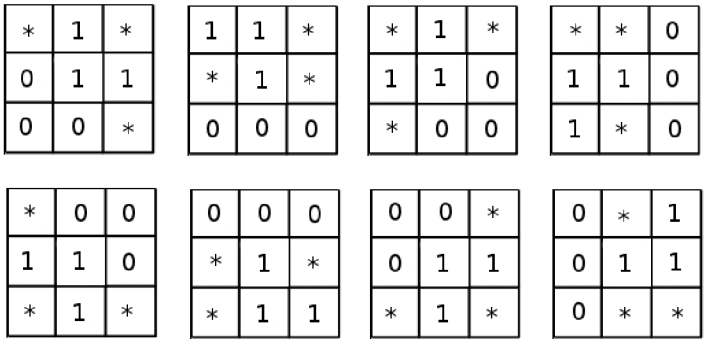
Nejdůležitějším krokem u metod skeletonizace je rozhodnout, který pixel v obrazu je redundantní. Při posuzování se využívá tzv. sousedství pixelu. O sousedství jde pouze tehdy, je-li vzdálenost mezi pixely rovna jedné. Ve většině algoritmů se využívá 8-okolí, ale můžeme využít i 4-okolí. Pojem sousedství demonstruje obrázek (obr. 2.6).

My se nadále budeme zabývat pouze algoritmy počítající s 8-okolím. V tomto případě máme tedy $2^8 = 256$ možných kombinací černých a bílých pixelů v sousedství. Tyto kombi-

nace jsou uvedeny v knize [1]. Z toho plyne, že nejdůležitějším a zároveň nejtěžším krokem je rozhodnout, kterou kombinaci vybrat pro odstranění redundantního pixelu.



Obrázek 2.6: Ukázka 8-okolí a 4-okolí. $P1 - P8$ jsou sousedy pixelu p



Obrázek 2.7: Strukturní elementy Golayovi abecedy, * označuje, že na daném pixelu nezáleží

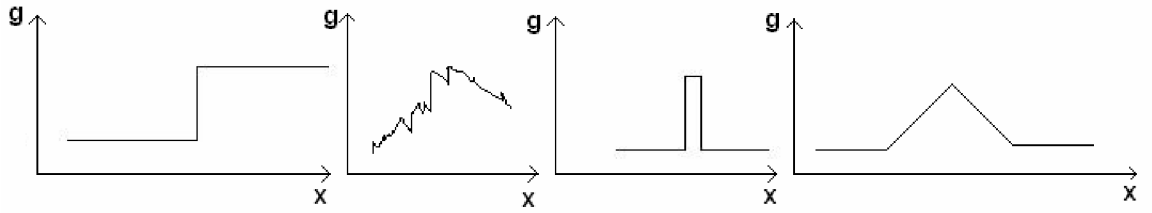
Ztenčování provádíme opakováním transformace hit or miss (kap. 2.1.2) při současném pootáčení příslušného strukturního elementu do všech osmi směrů. Existuje několik posloupností strukturních elementů. Pro tuto transformaci je používán strukturní element, který bývá obvykle označován jako písmeno L tzv. Golayovi abecedy (viz obr. 2.7). Pro každý potenciálně odebratelný pixel je nutno provést všech těchto osm transformací.

Operace ztenčování je idempotentní, tzn. že po určitém kroku se přestanou pixely měnit. Pokud po sobě následujících iteracích zůstává obraz nezměněn, je algoritmus ukončen.

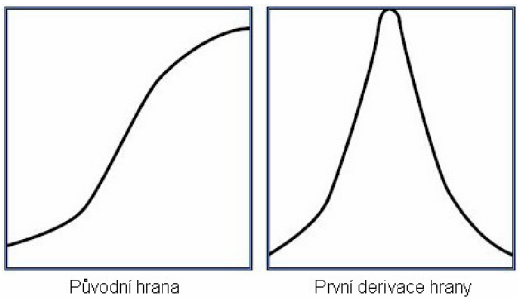
2.1.3 Detekce hran

Pro vnímání člověka jsou velice důležitá místa v obraze, kde se náhle mění hodnota jasu. Těmto odpovídajícím pixelům říkáme hrany a k jejich nalezení se používají hranové detektory (angl. edge detector). Hranu si lze také představit jako přechod nízkofrekvenčního pásma do pásma vysokofrekvenčního. Tento přechod má určitou šířku = *strmost hrany*, která také určuje ostrost hrany.

Hranových detektorů existuje celá řada, lišící se jak ve způsobu nalezení hrany, tak v její citlivosti rozpoznání a s ní související odolnosti proti šumu a jiným nežádoucím efektům zpracovávaného obrazu. V základu můžeme tyto detektory rozdělit do dvou kategorií – metody využívající *první* nebo *druhou derivaci*. Při použití první derivace je výsledný hranový gradient porovnáván s prahovým koeficientem, který určuje, zda se jedná o hranu či nikoli. U metod druhé derivace je využíván k detekci hrany význam prostorové změny v polaritě druhé derivace.[25]



Obrázek 2.8: Ukázka tvarů hran, z leva: skoková hrana, zašuměná hrana, tenká linie, střechová hrana



Obrázek 2.9: Znázornění hrany a její derivace

Pokud je hrana definována jako relativně velká změna hodnoty jasové funkce, bude v místě hrany velká hodnota derivace této funkce(viz obr. 2.9).

Maximální hodnota této derivace (gradientu) bude v kolmém směru k hraně. K výpočtu tohoto gradientu můžeme přistupovat jako ke konvolučnímu filtrování obrazu. Konvoluce [24] je matematický operátor zpracovávající dvě funkce. Pracujeme-li s digitálním obrazem používá se tzv. *diskrétní konvoluce*, která je dána vztahem (2.6),

$$I(x,y) * h(x,y) \quad = \quad \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x-i, y-j) h(i,j) \tag{2.6}$$

kde $I(x,y)$ je zde diskrétní obraz a $h(x,y)$ je jádrem konvoluce.

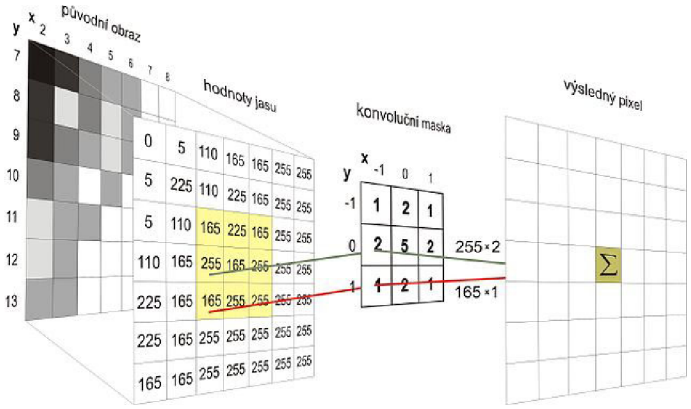
V podstatě si diskrétní konvoluci můžeme představit jako operaci s maticí pomocí jiné matice zvané jádro. Jádro položíme na příslušné místo obrazu a každý pixel překrytý tabulkou vynásobíme koeficientem v příslušné buňce jádra a všechny tyto hodnoty sečteme. Tím dostaneme jeden nový pixel. Tento výpočet ilustruje obrázek 2.10.

Jednotlivé hranové detektory se od sebe odlišují právě jádrem konvolučního filtru (matice používaná ke konvoluci). Tato matice udává, které body se při výpočtu účastní a jaké mají váhy.

Obecně platí pravidlo přímé úměry – čím větší je tato matice, tím je detektor odolnější proti šumu.

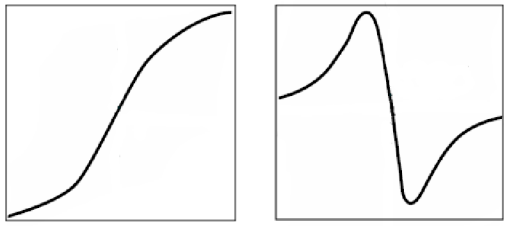
V tabulce 2.1 uvádím několik významných filtrů a jejich konvoluční jádra, využívající právě první derivace.

Metody založené na druhé derivaci jasové funkce se snaží nalézt průchody této derivace nulou (angl. zerro-crossing) (viz obr. 2.11) [6]. Využívá se skutečnosti, že je mnohem snažší nalézt průchod nulou, než-li extrémem funkce. [25]



Obrázek 2.10: Použití konvolučního jádra, zdroj[a]

[a]http://cs.wikipedia.org/wiki/Soubor:Konvoluce_2rozm_diskretni.jpg



Obrázek 2.11: Znázornění hrany a její druhá derivace

Bohužel druhá derivace je ještě citlivější na šum v obrazu, než předchozí. Proto je vhodné neli nutné kombinovat její výpočet s takovým vyhlazením, které odstraní maximální množství šumu a při tom nepoškodí hrany obrazu.

Příkladem je **Marrův-Hildrethové operátor** a **Cannyho hranový detektor**, o kterém bude dále řeč.

Cannyho hranový detektor

Patří k nejpoužívanějším algoritmům pro detekci hran a je obecně znám jako optimální hranový detektor. Detektor je navržen tak, aby splňoval tři základní požadavky [6]:

- *Detekční kritérium* požaduje, aby významné hrany nebyly přehlédnuty, a aby na jednu hranu nebyly vícenásobné odezvy.
- *Lokalizační kritérium* požaduje, aby rozdíl mezi skutečnou a nalezenou polohou hrany byl minimální.
- Požadavek *jedné odezvy zajišťuje*, aby detektor nereagoval na jednu hranu v obraze vícenásobně. Toto očekávání je zaměřeno zejména na zašumělé a nehladké hrany.

Abychom dosáhli požadovaných vlastností, musíme použít širší škálu postupů [20]. Pro výpočet mapy hran je potřeba znát nejen velikost gradientu, ale i směr. Prvním krokem

	G_x	G_y
Sobelův	$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Robertsův	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Prewittové	$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Frei-Chenův	$\begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$

Tabulka 2.1: Ukázka konvolučních jader některých významných filtrů

je eliminace šumu Gaussovým filtrem, který lze realizovat pomocí konvoluce, jejíž výpočet není výpočetně náročný. Poté následuje aplikace Sobelova operátoru pro nalezení velikostí gradientů a jejich směrů. Protože po tomto výpočtu vznikají tlusté hrany a my chceme aby hrana byla detekována pouze tam, kde se vyskytuje, je nutné provézt ztenčení. Ztenčení se zde provádí pomocí zvláštní poloprahovací techniky, které se říká *Nonmaxima suppression*. U této techniky se vychází z předpokladu, že hrana dává největší odezvu v místě, kde se skutečně nachází. Tímto krokem bohužel rozpojíme hrany v místech, kde se stýkají tři hrany v jednu a vytváří tak tvar **T**. K rozpojení dojde kvůli tomu, že jedna hrana je vždy větší než ta druhá a vliv jejího gradientu je v místě spojení dominantní. Posledním krokem je tzv. *hysterezní prahování*, které nám slouží k ohodnocení významu nalezených hran. Cannyho detektor má mezi vstupními parametry hodnoty dvou prahů T_1 a T_2 se kterými jsou pak porovnávány hodnoty nalezených gradientů následujícím způsobem:

- Pokud je hodnota gradientu větší než práh T_2, je označen jako hrana přímo.
- Pokud je hodnota gradientu menší než práh T_1, jedná se o nehranový gradient.
- Leží-li hodnota gradientu v rozmezí prahů T_1 a T_2, je chápán jako hrana jen v případě, že sousedí s bodem, který už jako hrana uznán byl.

Reprezentace hran

Pouhé zobrazení hran v obraze není pro většinu úloh (jako je i ta naše) dostačující. Obvykle je nutné pro další práci si hrany nějakým způsobem uchovat. Otázkou je, jak je reprezentovat z hlediska počítačových dat. Kritériem pro výběr správného přístupu je uvědomění si, jaký typ křivek bude převládat. Jedná-li se převážně o zaoblené křivky s minimem ostrých rohů, je vhodné tyto křivky reprezentovat přímo pomocí matematických funkcí. Naopak při zpracování obrazu, kde převládají kontury s množstvím ostrých změn směru (jako je např.

u rastrových dat ze skeneru či digitální fotografie) je výhodné uložit data do tzv. *Řetězcových kódů*. Řetězcovému kódu se budeme věnovat v další kapitole.

2.1.4 Houghova transformace

Houghova transformace [2, 11] je metoda, která se používá pro hledání jednoduchých útvarů, jako je úsečka, elipsa či kružnice. Princip této metody byl publikován v roce 1959 P. V. C. Houghem, který si ji o tři roky později nechal patentovat. Abychom mohli transformaci použít, je třeba znát analytický popis hledaného objektu. Přímku lze například vyjádřit v kartézském součinu rovnicí $y = a * x + b$.

Nevýhodou tohoto vyjádření je, že pro přímky rovnoběžné s osou Y se a stává nekonečným, proto se používají spíše polární tvary. Jde tedy vlastně o transformaci z kartézského souřadnicového systému do systému polárního.

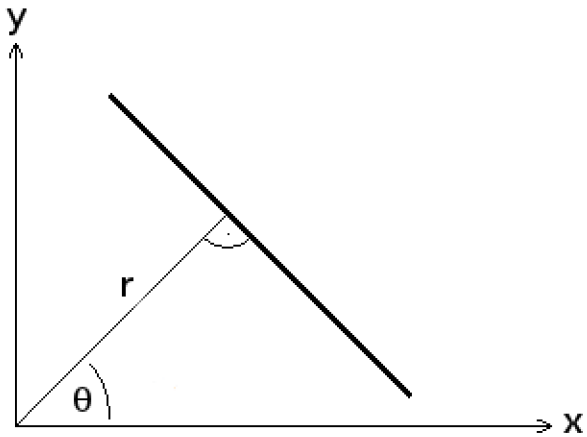
Významné použití této transformace je právě při analýze textu, kdy s její pomocí lze nalézt řádek textu v obraze. Úlohu pro Houghovu transformaci je možné formulovat jako hledání takové podmnožiny bodů v obraze, která co nejvíce odpovídá části přímky – úsečce.

Parametrické vyjádření přímky je v (rov. 2.7, 2.8),

$$r = x * \cos\theta + y * \sin\theta \tag{2.7}$$

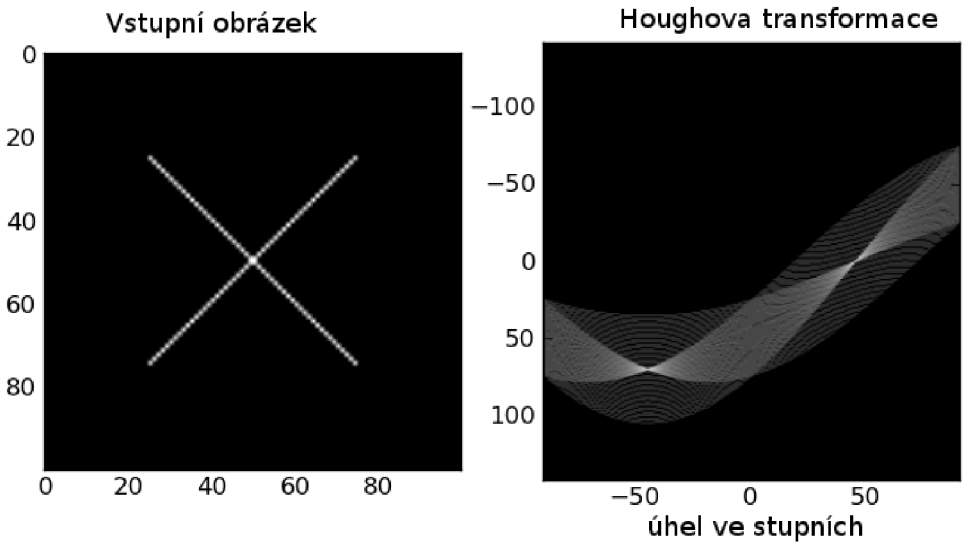
$$y = (-\frac{\cos\theta}{\sin\theta}) * x + (\frac{r}{\sin\theta}) \tag{2.8}$$

kde r je délka normály od přímky k počátku souřadnic a θ je úhel mezi normálou a osou x (viz. obr. 2.12).



Obrázek 2.12: Polární definice přímky

Při detekci se používá tzv. *akumulátor*. Pro detekci přímek je dvourozměrný o parametrech r a θ a lze si ho představit jako dvourozměrný graf (viz obr. 2.13). Využívá se také skutečnosti, že každý bod obrazu vyjádřený souřadnicemi x a y lze prezentovat formou křivek v parametrickém prostoru.



Obrázek 2.13: Příklad naplněného akumulátoru, zdroj[a]

[a]http://scikits-image.org/docs/dev/_images/plot_hough_transform_1.png. Maxima akumulátoru představují nejsvětlejší body.

Samotná transformace pak probíhá tak, že parametry x a y každého bodu obrazu jsou postupně dosazovány do rovnice 2.7. Za hodnotu θ je postupně dosazována hodnota z intervalu $< 0°, 360° >$ a parametr r je dopočítán. V akumulátoru je pak na pozici r a pro všechny hodnoty θ zvýšena intenzita.

Takto vznikají v parametrickém prostoru křivky – pro každý bod obrazu jedna. Pokud leží body na jedné přímce, tak se jejich křivky v parametrickém prostoru protínají v jednom místě – vznikají maxima v akumulátoru. Z těchto míst pak lze dosazením příslušných hodnot r a θ do rovnice 2.8 dostat klasický přepis pro přímku $y = ax + b$.

Nakonec bych chtěl ještě zmínit, že vstupem obvykle bývá obraz již zpracovaný detektorem hran a jinými morfologiemi. Je to výhodné, protože v drtivé většině případů zvyšuje tento postup pravděpodobnost nalezení hledaných struktur a významným způsobem snižuje i časovou náročnost celé detekce.

Kapitola 3

Extrakce příznaků

Extrakce příznaků patří k nejobtížnější a nejproblematičtější části při rozpoznávání vzorů. Cílem extrakce je získat základní charakteristiky každého znaku. Na kvalitě příznaků je závislá kvalita a rychlost výsledného klasifikátoru. Tyto charakteristiky musí být diskriminativní, abychom mohli jednotlivé znaky od sebe odlišit a tak určit, do které třídy patří. Většina metod je založena na popisu znaků přímo z rastru obrazu. Některé metody se ovšem zaměřují na určité rysy znaku a dalších pro ně nedůležitých atributů si nevšímají. Dále uvedu některé možnosti pro extrakci příznaků, které budou použity ve výsledné aplikaci.

3.1 Intenzita pixelů

Nejjednodušší metodou při extrakci příznaku je přímo použití jasu jednotlivých pixelů. Pokud má však obrázek znaku velké rozměry, je tato metoda pro některé klasifikátory nepoužitelná. Výkupní cenou za použití těchto příznaků jsou velké výpočetní a paměťové nároky. Použijeme-li toto řešení, je vhodné dále zvážit využití metody pro redukci dimenzionality – PCA.

3.1.1 Principal Component Analysis

Principal Component Analysis – PCA, v překladu znamená Analýza hlavních komponent. Je jednou z nejstarších a nejvíce používaných metod vícerozměrné analýzy. Poprvé byla zavedena Pearsonem již v roce 1901, jako popisná statistická metoda, sloužící především k redukci vícerozměrných dat. H. Hotelling zobecnil v roce 1933 postup aplikací komponentní analýzy na náhodné vektory a navrhl použití analýzy hlavních komponent pro rozbor kovarianční struktury proměnných [13]. Proto se v praxi můžete setkat i s názvy, jako Hotellingova transformace nebo Karhunen-Loeveho transformace. Cílem analýzy hlavních komponent je především zjednodušení popisu skupiny vzájemně lineárně závislých (korelovaných) znaků. Techniku lze popsat jako metodu lineární transformace původních znaků na nové, lineárně nezávislé proměnné, nazvané hlavní komponenty. Tyto komponenty shrnují informaci o původních za cenu minimální ztráty informace. Hlavní komponenty jsou setříděny sestupně podle míry variability, neboli rozptylu. Může být chápána jako transformace z původního do nového souřadnicového systému, jehož osy jsou tvořeny hlavními komponentami. Osy procházejí směry maximálního rozptylu, protože podmínka nezávislosti komponent vede ke kolmosti os [22].

PCA je technikou, která je široce používána v aplikacích, které potřebují např. redukovat dimenze, kompresi dat s co nejmeší ztrátovostí, extrakci příznaků apod. Je využívaná, protože je jednoduchou, neparametrickou metodou pro extrakci důležitých informací z neuspořádaných dat. Je citlivá na změnu měřítka, proto se musí nejdříve provést normalizace původních proměnných.

Postup pro redukci dimenzí[17]:

1. Máme I obrázků, každý o rozměrech $N \times N$ pixelů. Každý obrázek uložíme do jednoho vektoru o délce N^2, kde hodnoty vektoru jsou intenzity jednotlivých pixelů. Z těchto vektorů, poté vytvoříme matici O:

$$\mathbf{obrazMatice} = \begin{pmatrix} obrazVektor1 \\ obrazVektor2 \\ \vdots \\ obrazVektorI \end{pmatrix}$$

2. Nyní vypočteme průměrnou hodnotu pro každou dimenzi. Průměrnou hodnotu dimenze X budeme značit $\overline{X}$. Od každé hodnoty odečteme průměrnou hodnotu a uložíme do matice P.

3. Vypočteme kovarianční matici. Abychom ji mohli vypočítat, musíme nejprve zjistit, jak spočítat kovarianci:

$$cov(X,Y) \quad = \quad \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{n-1} \tag{3.1}$$

   Nyní se tedy podíváme na samotnou kovarianční matici C:

$$\mathbf{C} = \begin{pmatrix} cov(X,X) & cov(X,Y) & cov(X,Z) & \dots & cov(X,N) \\ cov(Y,X) & cov(Y,Y) & cov(Y,Z) & \dots & cov(Y,N) \\ cov(Z,X) & cov(Z,Y) & cov(Z,Z) & \dots & cov(Z,N) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ cov(N,X) & cov(N,Y) & cov(N,Z) & \dots & cov(N,N) \end{pmatrix}$$

   Kovarianční matice je čtvercová o rozměrech NxN a je symetrická podle hlavní diagonály, protože platí:

$$cov(X,Y) \quad = \quad cov(Y,X) \tag{3.2}$$

4. Z kovarianční matice vypočteme tzv. vlastní čísla (angl. eigenvalues) a vlastní vektory (angl. eigenvectros.), tyto hodnoty jsou navzájem propojené. Vlastní čísla uvádějí míru variability jednotlivých vlastních vektorů.

5. V této části se rozhodujeme, které dimenze jsou méně významné. Ty poznáme tak, že mají malou hodnotu vlastního čísla. Platí totiž pravidlo, že má-li nějaký původní znak malý či dokonce nulový rozptyl, není schopen přispívat k rozlišení mezi objekty.

   Vytvoříme si matici FV jehož hodnoty jsou hodnoty vlastních vektorů sestupně seřazené podle hodnot vlastních čísel. Délka vektoru je rovna počtu dimenzí, které chceme zachovat.

6. Získáme finální data:

$$FinalData \quad = \quad radekMaticeFV \times radekMaticeP \tag{3.3}$$

Z uvedeného postupu je patrné, že model PCA odpovídá aproximaci zdrojové matice dat O, který použijeme místo původní zdrojové matice. Tato aproximace má řadu výhod v interpretaci dat. PCA má za úkol transformovat data do nového systému os a nalézt a vypustit šum, čímž snížíme rozměrnost úlohy. Problémem zůstává, kolik hlavních komponent je nutné použít. Existuje horní mez počtu hlavních komponent, které mohou být odvozeny ze zdrojové matice dat O. Největší počet hlavních komponent se buď rovná číslu $m-1$, nebo n v závislosti na tom, které z těchto dvou čísel je menší. Je-li matice O složena například z $n = 800$ obrázků a $m = 400$ pixelů v obrázku, bude maximální počet hlavních komponent 399. Počet efektivních hlavních komponent se rovná hodnosti zdrojové matice O [13].

3.2 HU momenty

Další technikou, kterou navrhuji pro extrakci příznaků, je použití momentů. Tuto techniku použiji spíše ze zájmu, neočekávám nijak dobré výsledky. Použití těchto momentů s klasifikátorem k-nearest neighbour mělo bídnou úspěšnost. Bude zajímavé sledovat, jak si bude vést klasifikátor založený na neuronových sítích.

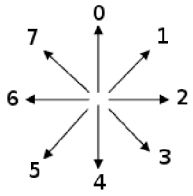
Za rys znaku je brán např. moment tmavých znaků k vybranému počátku souřadnic. Hu momenty jsou sadou absolutně ortogonálních momentových invariantů a to včetně rotace. Tyto momenty mohou být použity pro rozpoznávání nezávisle na rotaci, měřítku či pozici rozpoznávaného objektu [8]. Výpočet jednotlivých momentů naleznete v uvedeném článku nebo na wikipedii [21].

3.3 Řetězcový kód – Freemanův řetězcový kód

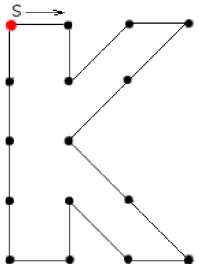
Jak jsem se zmínil v minulé kapitole, řetězcový kód slouží k reprezentaci jednotlivých hran. Jeho princip je založen na faktu, že cestu z jednoho bodu do sousedního bodu popisované křivky lze označit např. číslicí. Pokud přijmeme fakt, že se pohybujeme na úrovni jednotlivých pixelů v mřížce obrazu, lze takto označit čtyři respektive osm směrů (podle uvažovaného okolí pixelu).

Pro extrakci příznaků bude v naši úloze použit Freemanův řetězcový kód, který pracuje s osmi směry, označenými číslicemi od 0 do 7. Tyto směry ilustruje obrázek 3.1. Pro obrázek 3.2 z počátku S bude tedy Freemanův řetězcový kód následující: 2412553367460000. Z jeho výpočtu je vidět, že pro každé písmeno bude kód odlišný a s proměnnou délkou.

Protože pro rozpoznávání jednotlivých písmen bude použita neuronová síť, je zapotřebí stejného počtu příznaků pro každé písmeno. Toho dosáhneme tak, že z daného řetězce uděláme histogram, z kterého následně vytvoříme histogram normalizovaný. Ten vzniká tak, že každou hodnotu histogramu dělíme celkovým počtem prvků ve Freemanově kódu a výsledný podíl zapíšeme na příslušné místo. Výhodou normalizovaného histogramu by měla být relativní nezávislost na velikosti zpracovávaného znaku.



Obrázek 3.1: Označení směrů Freemanova kódu



Obrázek 3.2: Příklad reprezentace popisované kontury

3.4 Gradient, Structural and Concavity - GSC

Dle literatury [3] lze dosáhnout 98,87% úspěšnosti v rozpoznávání ručně psaných číslic, což zkusíme ověřit pomocí neuronových sítí nad databazí MNIST[1].

Jedná se o 512-bitový binární příznakový vektor jež nese informaci o gradientu, struktuře a konkávnosti znaků. Nejprve je nalezeno nejmenší ohraničení znaku aby se eliminovaly výpočetní nároky a odstranila přebytečná informace v podobě černých pixelů náležících pozadí. Následně je obraz rozdělen do 4×4 oblastí, pro něž jsou počítány jednotlivé příznaky.

3.4.1 Gradient

Gradientní příznak zachycuje směr gradientu každého bodu hrany znaku. Po aplikaci Sobelova hranového detektoru je vypočítána hodnota směru gradientu ze vztahu 3.4.

$$\alpha = \arctan \frac{Gy}{Gx} \tag{3.4}$$

Třistašedesát stupňů je rozděleno do dvanácti výsečí po 30° a hodnota směru gradientu spadá do jedné z nich. Tato hodnota je spočítána pro každý pixel oblasti a je vytvořen histogram. Potom je zvolen práh, který na histogram aplikujeme, a který určí, které příznaky jsou aktivní, a které ne. Na konci tedy pro znak dostaneme 192-bitový gradientní vektor $(4 \times 4 \times 12)$.

3.4.2 Struktura

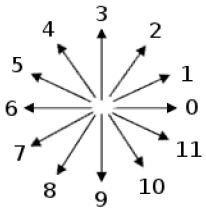
Strukturální příznaky zjišťují výskyt některých vzorů v obrazu. Ty jsou hledány na základě mapy gradientů a dvanácti pravidel (viz tab. 3.1). Tyto pravidla manipulují s osmi–okolím každého pixelu. Na základě sousedství a přípustného gradientu (obr. 3.3) jsou potom tyto vzory identifikovány. Jedná se o diagonály, vodorovné, svislé hrany a 4 typy rohů.

Volbou vhodného prahu opět určíme, které strukturní příznaky jsou významné, a které nikoliv. Pro celý znak dostaneme opět 192-bitový vektor.

3.4.3 Konkávnost

Příznaky konkávnosti jsou nejhrubší z jmenovaných a slouží k určení vztahu mezi tahy ve velkém měřítku v celém snímku. Rozlišujeme 8 příznakových typů: hustota bílých pixelů,

[1] Volně stažitelná databáze rukou psaných číslic, dostupná na adrese http://yann.lecun.com/exdb/mnist/



Obrázek 3.3: Směr gradientu a sousedství pixelu

Pravidlo	Popis	Soused 1	Soused 2
0	Horizontální hrana typ 1	$S_0(2,3,4)$	$S_4(2,3,4)$
1	Horizontální hrana typ 2	$S_0(8,9,10)$	$S_4(8,9,10)$
2	Vertikální hrana typ 1	$S_2(5,6,7)$	$S_6(5,6,7)$
3	Vertikální hrana typ 2	$S_2(1,0,11)$	$S_6(1,0,11)$
4	Vzestupná diagonála typ 1	$S_5(4,5,6)$	$S_1(4,5,6)$
5	Vzestupná diagonála typ 2	$S_5(0,11,10)$	$S_1(0,11,10)$
6	Sestupná diagonála typ 1	$S_3(3,2,1)$	$S_7(3,2,1)$
7	Sestupná diagonála typ 2	$S_3(7,8,9)$	$S_7(7,8,9)$
8	Pravoúhlý roh typ 1	$S_2(5,6,7)$	$S_0(8,9,10)$
9	Pravoúhlý roh typ 2	$S_6(5,6,7)$	$S_0(2,3,4)$
10	Pravoúhlý roh typ 3	$S_4(8,9,10)$	$S_2(1,0,11)$
11	Pravoúhlý roh typ 4	$S_4(4,3,2)$	$S_6(1,0,11)$

Tabulka 3.1: Pravidla pro extrakci strukturálních příznaků

vodorovné a svislé tahy, levá, pravá, stoupající a klesající konkávnost a díry. Celkově vektor tvoří 128 příznaků.

Pro stanovení **Hrubé hustoty pixelů** jsou nejprve spočítány pixely patřící objektu, které jsou poté prahovány a vzniká 1 bit pro oblast.

Pro celý snímek dostaneme tedy 16-bitový vektor ($4 \times 4 \times 1$).

Dlouhé tahy

Tyto příznaky se snaží zachytit dlouhé svislé a vodorovné tahy. Nejprve je zjištěna pro každý pixel objektu délka souvislých pixelů ve svislém a ve vodorovném směru.

Na základě jejich vztahu a zvoleném prahu potom určíme, zda se jedná o horizontální nebo vertikální tah, který je potom reprezentován 1 na příslušném místě.

Dostaneme tedy dva bity pro oblast a 32-bitový vektor pro celý snímek ($4 \times 4 \times 2$).

Typ konkávnosti

Z každého černého pixelu (pozadí) jsou vyslány paprsky v osmi směrech a je sledován způsob jejich nárazu. Paprsek může narazit buď na okraj snímku nebo na bílý pixel (objekt). Podle toho jakým způsobem končí paprsky, je potom rozhodnuta jeho třída. Pokud všech osm paprsků narazí na objekt – díra, pokud horních pět – stoupající, atd.

Po prahování dostaneme 80-bitový vektor pro celý snímek ($4 \times 4 \times 5$).

Kapitola 4

Klasifikace

V této kapitole si povíme obecné informace o klasifikátorech. Existuje mnoho metod klasifikace a žádná z nich není univerzální. Proto musíme při nasazení do praktických úloh zvažovat, kterou cestou se vydat a jakou metodu použít. Při výběru metody musíme brát zřetel na výpočetní prostředky, množství trénovacích dat, variabilitu předkládaných dat atd. My jsme pro rozpoznávání rukou psaných hůlkových písmen zvolili neuronové sítě, kterým je věnována celá následující kapitola.

4.1 Trénování klasifikátorů

Trénování klasifikátorů se obvykle provádí učením s učitelem (supervised training), kdy optimalizujeme objektivní funkci ze známých dat. O těchto datech je dopředu známo, do které třídy patří. Klasifikátor je potom podle dat, které během učení viděl, schopen předpovědět klasifikační třídu pro neznámá data. Až na některé specifické případy nemůžeme při učení klasifikátoru obsáhnout všechny případy předkládaných dat, proto bude klasifikátor vykazovat určitou chybu. Naším úkolem je zajistit, aby tato chyba byla co nejnižší. Se snižováním chyby však u některých klasifikátorů roste komplexnost klasifikátoru či snížená schopnost generalizace, a tak se často volí kompromis mezi rychlostí vybavování a chybovostí řešení.

Pro trénování se používají dva typy vzorků: *trénovací* a *testovací* data.

Na první jmenované sadě se optimalizuje klasifikační funkce a na druhé se ověřuje chyba klasifikace. Trénovací sada se v některých případech dělí na vlastní trénovací sadu a sadu *validační*, která se používá pro detekci přetrénování.

Obecně můžeme říci, že trénovací sada obsahuje obrazy jednotlivých objektů a jejich správné zařazení do klasifikačních tříd. Pro jednoduchost zatím uvažujme, že máme naučit klasifikátor na rozpoznání číslice 1, další číslice pro tento případ nehrají roli. Budeme mít tedy dvě klasifikační třídy - jedničky a cokoliv jiného. Třída jedničky bude obsahovat obrazce jedniček napsané různým pisatelem a v druhé bude vše, co se jako jednička nemá klasifikovat. Abychom tedy dosáhli co nejlepších výsledků, musíme pečlivě vybírat trénovací sadu vzorků. Platí totiž pravidlo, že čím lepší trénovací sadu máme k dispozici, tím lepších výsledků při rozpoznávání dosáhneme.

4.1.1 Průběh trénování

Před samotným trénováním jsou ze vstupních dat vyextrahovány příznaky a je vytvořen příznakový vektor. Samotný průběh trénování probíhá ve dvou krocích:

1. Klasifikace
   V tomto kroku se použije příznakový vektor ke klasifikaci a vypočte se chyba klasifikace.

2. Úprava klasifikátoru
   Zde dochází k úpravě klasifikační funkce podle zjištěné chyby v prvním kroku.

Tyto kroky se opakují dokud není splněna podmínka pro zastavení. Jednou z možností je např. porovnávání chyb mezi jednotlivými kroky klasifikace, kdy se pokračuje tak dlouho, dokud pokles chyby nepřesáhne určitou hodnotu. Další podmínkou může být počet iterací, které se mají vykonat. Používaným kritériem je i velikost chyby klasifikátoru. Klasifikátor bude trénován tak dlouho, dokud se nedostane pod hodnotu maximální chyby.

Jak tedy vyplývá, trénuje se, dokud klesá chyba na celé trénovací sadě. Poté je vybrána konfigurace, která dává nejlepší výsledky a je nasazena na rozpoznávání validační sady. Pokud i nadále klesá chyba klasifikace, můžeme pokračovat v trénování. V opačném případě hrozí přetrénování (při příliš velkém množství trénovacích dat klesá chyba na trénovacích, ale roste chyba na testovacích datech), proto proces ukončíme.

4.1.2 Výpočet chyby klasifikátorů

Chyba klasifikátoru se zjišťuje na trénovací sadě, kde je dopředu znám výsledek. Obecně můžeme říci, že výsledná chyba je počet chybně klasifikovaných dat ku datům celé datové sady. Pro výpočet chyby se zjišťují tyto hodnoty: True Positive (TP, správné přijetí), True Negative (TN, správné odmítnutí), False Positive (FP, špatné přijetí) a False Negative (FN, špatné odmítnutí)viz. [4]. Výsledná chyba se poté vypočítá například ze vztahu 5.1, kde P je počet všech pozitivně zařazených dat a N značí chybně zařazená data.

$$chyba = \frac{FP + FN}{P + N} \tag{4.1}$$

Kapitola 5

Neuronové sítě

Jedním z hlavních podnětů k rozvoji neuronových sítí byl zájem člověka zjistit funkci lidského mozku, snaha pochopit, jakým způsobem myslíme. Informace jsou čerpány zejména z neurofyziologie, která nám umožnila vytvářet zjednodušené matematické modely. Vlastnosti mozku se staly podkladem k vytváření teorií pro umělé neuronové sítě. Je to také důvod k neustálému studiu této problematiky. K popisu umělých neuronových sítí je tedy dobré si osvojit základy činnosti a biologické struktury mozku.

5.1 Stručná historie

Jak sem již psal v úvodu, tak kořeny studií neuronových sítí sahají na začátek minulého století, kdy se vědci po mnoha desetiletí snažili přijít na to, jak funguje náš nervový systém. V roce 1890 William James vyslovil teorii, která se stala základem mnoha dalších prací. V ní tvrdil, že množství aktivity v daném místě na mozkové kůře je sumou všech tendencí, které do ni byly vpuštěny. Taková tendence záleží pak na tom, kolik vzrušení lze přivést do daného bodu a na intenzitě těchto vzruchů. [12]

Za počátek vzniku oboru neronových sítí je práce Warrena McCullocha a Waltera Pittse, která vznikla v roce 1943. Tito pánové vytvořili velmi jednoduchý model neuronu, což je základní jednotka nervové soustavy. Číselné hodnoty parametrů v tomto modelu byly převážně bipolární (tj. z množiny $\{-1, 0, 1\}$). Tímto modelem ukázali, že lze v principu počítat libovolnou aritmetickou nebo logickou funkci. Ačkoliv neočekávali nějaké praktické bezprostřední využití svého modelu, jejich článek ovlivnil další badatele. Jmenujme například zakladatele kybernetiky Norberta Wienera, který se jím nechal inspirovat při studiu podobnosti činnosti nervové soustavy a výpočetní techniky. [26][12]

V roce 1949 napsal Donald Hebb knihu *The Organization of Behavior* ve které navrhl učící pravidlo pro synapse neuronů a to na základě pozorování biologických neuronů. Vyslovil pravidlo, že synaptické spojení mezi dvěma ve stejnou chvíli aktivovanými neurony se posiluje.[12]

V roce 1957 vynalezla skupina kolem Franka Rosenbalta nový model neuronu tzv. *perceptron*, který je zobecněním McCullochova a Pittsova modelu neuronu pro reálný číselný obor parametrů. Pro jednovrstvou síť perceptronů dokonce dokázal stanovit učící algoritmus, o kterém matematicky dokázal, že pro daná treninková data nalezne po konečném počtu kroků odpovídající váhový vektor a to nezávisle na počátečním nastavení. [12] [26]

Chvíli po objevu perceptronu vynalezl Bernard Widrow se svými studenty další typ neuronového výpočetního prvku, který nazval ADALINE (ADAptive LINear Element). Tento

model byl vybaven novým výkonným učícím pravidlem, které se používá až dodnes. [26]

Bohužel pro teorii neuronových sítí, se v šedesátých létech potýkal tento obor s problémy, které vyústily téměř k dvacetileté pauze a studii neuronových sítí se věnovalo pouze pár nadšenců. Bylo to způsobeno několika faktory. Většina badatelů přistupovala k neuronovým sítím spíše z experimentálního hlediska, a když se doslýchaly z úst nadšených výzkumných pracovníků nepodložené informace, jako že do několika let bude vyvinut umělý mozek, mělo to dopad na smýšlení odborníků z jiných oblastí. Tyto skutečnosti odradily i ty vědce, kteří se dříve o neurovýpočty zajímali. Pomyslným hřebíkem do rakve byl důkaz Marwina Minského a Seymoura Paperta v roce 1969, ve kterém stálo, že jednovrstvá perceptronová síť nedokáže řešit XOR problém. Rosenbalt sice tušil, že vícevrstvá síť dokáže řešit jakýkoliv problém včetně problému XOR, ale nebyl schopný nalézt pro ně učící algoritmus. [26]

Začátkem let osmdesátých dochází opět k obrození tohoto vědního oboru.V roce 1986 byl nezávisle „znovu"objeven učící algoritmus pro vícevrstvé neuronové sítě pány Rumelhartem, Hintonem a Williamsem. Tento algoritmus je známý jako *back-propagation* (zpětné šíření chyby). Později se ukázalo, že pánové nebyli první a v době „temného"období byl již algoritmus několikrát publikován (např. Arthur Bryson a Yu-Chi Ho, 1969; Paul Werbos, 1971; David Parker, 1985). Přestože back-propagation ještě není zcela ideálním obecným algoritmem pro učení neuronových sítí v porovnání s lidským uvažováním, dokáže řešit mnoho problémů včetně XOR, které jednovrstvé perceptrony řešit nedokáží. [26]

Od roku 1987 zažívá tento vědní obor skvělé období. Byla založena mezinárodní společnost pro výzkum neuronových sítí INNS (Intenational Neural Networks Society). Jsou také pořádány větší konference se specilizací na neuronové sítě a v neposlední řadě mnoho renomovaných univerzit zakládá nové výzkumné ústavy. Tento trend trvá dodnes. [26]

5.2 Lidský mozek

Jak jsem již několikrát uvedl, idea učení neuronových sítí vychází z pozorování činnosti našeho mozku a nervové soustavy. Pojďme si o ní něco říct.

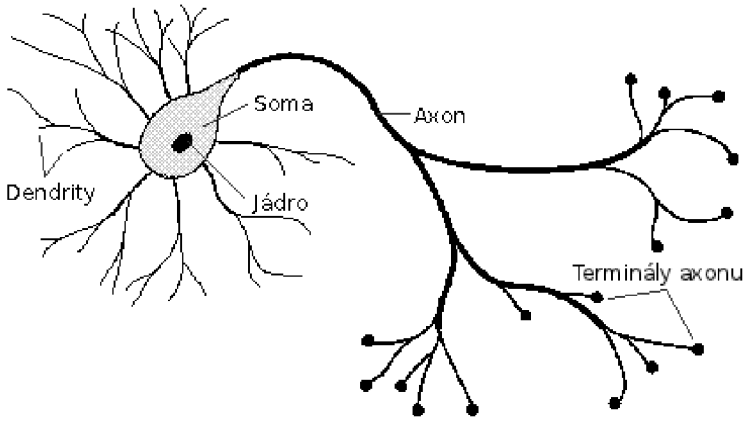
Mozková kůra je tvořena velkým množstvím specializovaných buněk, které jsou vzájemně propojeny. U těchto buněk se neustále mění jejich struktura a vzájemné vztahy. Nejdůležitější z nich je soubor asi 20 až 100 miliard neuronů. Neurony jsou živé buňky, specializované na přenos, zpracování a uchování informací. Proto jsou základem celé složité struktury mozku. Neuronů je několik druhů a jsou navzájem propojeny do velmi složitých neuronových sítí. Uvádí se, že na jeden neuron v průměru připadá 10 tisíc až 100 tisíc propojení s jinými neurony. Což znamená, že celková informační mohutnost neuronových sítí lidského mozku dosahuje řádu asi 0.1 až 10 trilionů [15].

Zajímavostí je, že pokud vypadnou některé z neuronů dodávajících informace, tak se výsledné chování neuronové sítě nezmění.

Průměrná hmotnost mozku muže je 1350g a ženy asi 1200g. Výkonnost a informační mohutnost mozku však hmotností posuzovat nemůžeme, ta pravděpodobně záleží na hustotě neuronů a na složitosti a funkční dokonalosti vzájemného propojení jeho neuronových sítí a na schopnosti jejich adaptivity [15].

Zajímavost: V lidském mozku je průměrně 2 - 3 miliony neuronů na 1 mm^3. Denně jich zahyne asi 10000. Přestože se odumřelé neurony neobnovují, udává se, že za 75 let života odumře asi jen 0,2 až 0,5 % celkového počtu neuronů.

Biologický neuron se skládá z několika částí. Z těla (soma), které je veliké několik mikrometrů a z něhož vybíhá několik tisíc výběžků - dendritů, které tvoří vstupy neuronu a dále jedno vlákno - axon, které tvoří výstup neuronu. Délka dendritů je maximálně 2 až 3 mm,



Obrázek 5.1: Biologický neuron

zato axon může být u zvířat dlouhý několik metrů. U člověka měří nejdelší axon asi 1 metr. Vede od páteře až po konečky prstů u nohou. Z axonu odbočuje řada větví (terminálů), která se převážně stýká s výběžky (trny) dendritů jiných neuronů. K přenosu informace pak slouží synapse, což je takové mezineuronové rozhraní [15].

Synapse jsou jedním z nejdůležitějších prvků ve stavbě mozku i ve struktuře jednotlivých neuronů. Z funkčního hlediska lze synapse rozdělit na excitační, které umožňují rozšíření vzruchu v nervové soustavě, a na inhibiční, které způsobují jeho útlum. Soma i axon jsou obaleny membránou, která je schopna za určitých okolností generovat elektrické impulzy. Tyto impulzy jsou pomocí synaptických bran přenášeny z axonu na dendrity ostatních neuronů. Synaptické brány mají určitou propustnost, která určuje intenzitu podráždění dalších neuronů. Pokud tyto podrážděné neurony dosáhnou určité hraniční meze, tzv. prahu, samy pak generují impulz a zajišťují tak šíření příslušné informace [19].

Každým průchodem signálu se mění synaptická propustnost, což je předpokladem paměťové schopnosti neuronů. Během života neustále vznikají nové synaptické spoje. Jejich počet je závislý na procesu učení organismu. V průběhu učení se nové paměťové stopy vytvářejí, naopak při zapomínání se synaptické stopy přerušují [19].

Funkce reálného biologického neuronu je velice složitá a dodnes není úplně prozkoumaná. Píší se o ní tlusté knihy, proto se můžete setkat s více formálními matematickými modely neuronu, z nichž žádný není dokonalý.

5.3 Umělý neuron

K reprezentaci biologického neuronu se používá formální neuron. Historicky první formální neuron vznikl v roce 1943, kdy Warren McCulloch a Walter Pitts navrhli jednoduchý matematický model neuronu (viz kap. 5.1). Dodnes je s mírnými modifikacemi nejpoužívanějším matematickým modelem [26].

Formální neuron je zjednodušený model biologického neuronu, od kterého očekáváme stejné nebo alespoň podobné chování. Tento neuron může mít n vstupů (modelují dendrity), ale pouze jeden výstup (modeluje axon). Jak si můžete všimnout na obrázku 5.2, tak umělý neuron získáme přeformulováním zjednodušené funkce neurofyziologického neuronu (obr. 5.1) do matematické řeči. Zde je schematicky zobrazen neuron se třemi vstupy

x_1, x_2 a x_3.

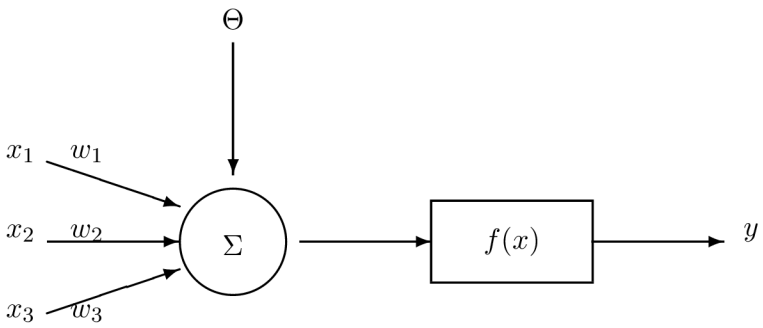
Vstupy jsou ohodnoceny odpovídajícími, obecně reálnými synaptickými váhami w_1, w_2 a w_3, které určují jejich propustnost. Ve shodě s neurologickou motivací mohou být synaptické váhy záporné, čímž se vyjadřuje jejich inhibiční charakter. Zvážená suma vstupních hodnot představuje vnitřní potenciál neuronu. Hodnota vnitřního potenciálu Σ po dosažení tzv. prahové hodnoty Θ indikuje výstup neuronu y [26].

Činnost neuronu je možné vyjádřit známým vztahem (5.1) [15],

$$y = f \sum_{i=1}^{N} x_i w_i - \Theta \tag{5.1}$$

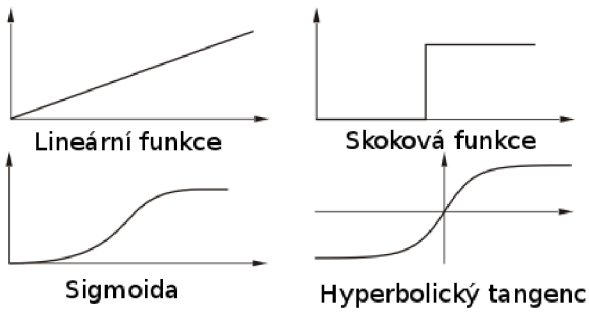
Kde:

y Výstup neuronu
f Aktivační/přenosová funkce. Argument je označován jako vnitřní potenciál neuronu
x_i Vstupy neuronu, počet vstupů je N
w_i Váhové vstupy neuronu (také označované jako synaptické váhy), počet je N
Θ Práh neuronu



Obrázek 5.2: Model umělého neuronu se třemi vstupy.

Výstup neuronu je určen funkční hodnotou aktivační funkce $f(x)$ vnitřního potenciálu neuronu. Aktivační funkce může být skoková, spojitá lineární nebo spojitá nelineární viz obr. 5.3. Neurony pak rozlišujeme na prahové (treshold units), lineární (linear units) nebo nelineární (non-linear units).

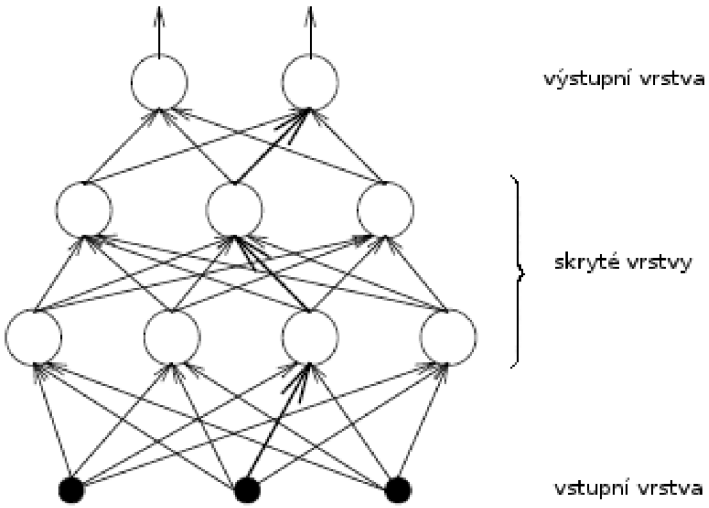


Obrázek 5.3: Aktivační funkce neuronů

5.4 Topologie sítě

Neuronová síť se skládá z formálních neuronů, které jsou navzájem propojeny tak, že výstupem neuronu je vstup několika dalších neuronů. Počet neuronů a jejich vzájemné propojení určuje tzv. topologie sítě. Z hlediska využití neuronů rozlišujeme v síti:

- vstupní neurony: jsou to tzv. pasivní prvky sítě, sloužící ke vstupu signálu a k následnému rozdělení další vrstvě neuronů.
- pracovní neurony: někdy nazývané jako skryté, mezilehlé neurony. Tyto neurony transformují vstupy a výsledek předávají další vrstvě.
- výstupní neurony: výstupem těchto neuronů je odezva na signály vstupní.



Obrázek 5.4: Propojení neuronů v síti

Šíření a zpracování informace na cestě v síti je umožněno změnou stavů neuronů ležících na této cestě. Stavy všech neuronů v síti určují tzv. *stav* neuronové sítě a synaptické váhy všech spojů neuronů tvoří tzv. *konfiguraci*. Neuronová síť se s časem vyvíjí, mění se stav a propojení neuronů a adaptují se váhy. Vzhledem k těmto změnám lze rozdělit celkovou dynamiku do tří dynamik a uvažovat tak tři režimy práce sítě – *organizační* (změna topologie), *aktivní* (změna stavu) a *adaptivní* (změna konfigurace) [26].

Organizační dynamika specifikuje architekturu sítě a její případnou změnu. Většinou předpokládá pevnou architekturu, která se již nemění. V zásadě rozlišujeme dva typy architektury – *cyklickou*, kde existuje skupina neuronů zapojených v kruhu, a *acyklickou* (obr. 5.4) [26].

Aktivní dynamika specifikuje počáteční stav sítě a způsob, jak se mění v čase při pevné topologii a konfiguraci. V aktivním režimu se na začátku nastaví vstupní neurony na tzv. vstup sítě a zbylé neurony jsou v počátečním stavu. Po této inicializaci probíhá vlastní výpočet. Většinou se předpokládá diskrétní čas. Na počátku se síť nachází v čase 0 a mění se v čase 1, 2 , 3... V každém takovém kroku je vybrán jeden neuron (sekvenční zpracování) nebo více neuronů (paralelní zpracování) a aktualizuje svůj stav na základě

svých vstupů, tedy stavů sousedních neuronů, jejichž výstupy jsou přivedeny na vstup právě aktualizovaných neuronů. Stav výstupních neuronů, který se mění v čase, je výstupem neuronové sítě. Avšak obvykle se uvažuje taková aktivní dynamika, že po určitém čase jsou stavy výstupních neuronů konstantní, a tak v aktivním režimu realizuje neuronová síť nějakou funkci na vstupním prostoru [26].

Adaptivní dynamika specifikuje počáteční konfiguraci sítě a to, jak se v čase mění jednotlivé váhy v síti. V adaptivním režimu se na začátku nastaví počáteční konfigurace, kdy se např. váhy nastaví náhodně. Po inicializaci konfigurace pak probíhá vlastní adaptace. Cílem adaptace je nalézt takovou konfiguraci, která by v aktivním režimu realizovala potřebnou funkci. Aktivní režim se používá k vlastnímu výpočtu funkce sítě pro daný vstup a adaptivní režim slouží k učení této funkce [26].

5.5 Modely neuronových sítí

Existuje mnoho druhů modelů neuronových sítí. My si v našem výkladu podrobněji představíme Perceptronovou síť a vicevrstvou perceptronovou síť se zpětným šířením chyby – *back-propagation*. Pokud by vás zajímala teorie dalších modelů a jejich uplatnění odkáži vás na velice dobrou knihu v českém jazyce – *„Teoretické otázky neuronových sítí"[26]* nebo anglicky psanou *„Artificial Neural Networks"[12]* .

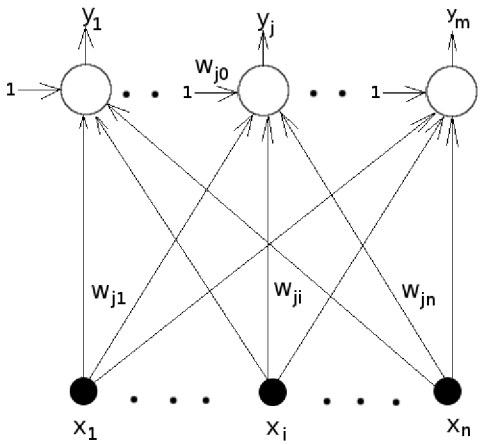
5.5.1 Perceptronová síť

Nejprve si řekneme obecné věci o perceptronových sítích. Prvky perceptronové sítě jsou perceptrony, což jsou neurony s binárními výstupy nebo s nelineárním výstupem (nelineární perceptron). Ve výstupní vrstvě se většinou používají právě binární perceptrony. Perceptronová síť je neuronová síť s dopřednými vazbami, což znamená, že vstupy každého neuronu jsou napojeny na výstupy všech neuronů z předchozí vrstvy. Spojení neuronů v jedné vrstvě nebo spojení se vzdálenější vrstvou neexistuje. Každý neuron má tak přesně tolik vstupů, kolik je neuronů v předchozí vrstvě. Vstupní vrstva sítě slouží pouze k distribuci vstupních hodnot.

Pro přenosovou funkci vrstvených perceptronových sítí není vhodná funkce skoková ani lineární. Lineární síť není nutné učit, příslušné váhy lze rovnou vypočítat. Avšak vrstva lineárních perceptronů provádí zase jen lineární zobrazení, což nemá žádné praktické využití. Nejpoužívanějším typem je tedy síť nelineárních perceptronů. Ty používají jako aktivační funkci tzv. *saturační funkci*, která součet impulsů transformuje do intervalu <0, 1>, a to tak, že u vstupních hodnot v blízkosti nuly prudce roste, zatímco u vysokých a nízkých hodnot se mění jen nepatrně [26]. Jako saturační funkce se obvykle používá sigmoida (5.2).

$$g(x) = \frac{1}{1 + e^{-\lambda x}} \tag{5.2}$$

Historicky prvním úspěšným modelem neuronové sítě byla **síť perceptronů**. Její **organizační dynamika** je taková, že na začátku má pevnou architekturu jednovrstvé sítě typu $n - m$. Síť se tedy skládá z n vstupních neuronů a z m neuronů výstupních (viz obr. 5.5)
Aktivní dynamika
Reálné stavy neuronů ve vstupní vrstvě se nastaví na vstup sítě a výstupní neurony počítají svůj binární stav, který určuje výstup sítě. To znamená, že každý perceptron nejprve vypočte svůj vnitřní potenciál jako příslušnou afinní kombinaci vstupů (rov. 5.3),



Obrázek 5.5: Perceptronová síť

$$\xi_j = \sum_{i=0}^{n} w_{ji} x_i \quad j = 1, ..m \tag{5.3}$$

kde n je počet vstupů, m počet výstupů a koeficienty $w = (w_{10} \ldots, w_{1,n} \ldots, w_{m0} \ldots, w_{mn})$ tvoří konfiguraci sítě.

Adaptivní dynamika

V adaptivním režimu je požadovaná funkce sítě perceptronů zadána tréninkovou množinou (5.4),

$$\tau = \{(x_k, d_k) \left| \begin{array}{l} x_k = (x_{k1}, \ldots, x_{kn}) \in R^n \\ d_k = (d_{k1}, \ldots, d_{km}) \in 0, 1 \end{array} \right. \quad k \ = 1, \ldots p\} \tag{5.4}$$

kde x_k je reálný vstup k-tého tréninkového vzoru a d_k je odpovídající požadovaný binární výstup. Cílem adaptace je, aby síť pro každý vstup x_k (k=1,...,p) z tréninkové množiny odpovídala v aktivním režimu požadovaným výstupem d_k, tj. aby platilo pravidlo:

$$y(w, x_k) = d_k \quad k \ = 1, \ldots p \tag{5.5}$$

Protože tréninková množina nemusí být vždy funkcí (tj. na jeden vstup jsou požadovány dva výstupy) a také ne každou funkci lze počítat jedním perceptronem, nelze vždy splnit tuto podmínku (rov. 5.5). V takovém případě se potom snažíme o naučení sítě co nejvíce vzorů.

Na začátku adaptace v čase 0 jsou váhy konfigurace w(0) nastaveny náhodně blízko nuly. V každém časovém kroku učení t = 1,2,3...je síti předložen jeden vzor z tréninkové množiny a síť se ho snaží naučit, tj. adaptuje podle něj své váhy.

5.5.2 Backpropagation

Nejznámější a nejpoužívanější model neuronové sítě. Je to vícevrstvá neuronová síť s učícím algoritmem zpětného šíření chyby (backpropagation). Tento model vyvolal obrovský

zájem o neuronové sítě po roce 1985. Je zobecněním sítě perceptronů pro architekturu se skrytými vrstvami. Organizační dynamika specifikuje na začátku většinou pevnou topologii. Standardně se používá dvouvrstvý či třívrstvý model (obr. 5.4).

Metoda backpropagation je založena na principu učení s učitelem (viz kap. 4.1), kdy síti předkládáme vzor, který by měla rozpoznat a necháme probíhat samotný algoritmus – měnit váhy dle zjištěné chyby. Na výstupu požadujeme rozpoznání právě tohoto vzoru.

Samotný algoritmus obsahuje tři etapy: dopředné (feedforward) šíření vstupního signálu tréninkového vzoru, zpětné šíření chyby a aktualizace váhových hodnot na spojeních. Princip si vysvětlíme na síti s jednou skrytou vrstvou. Během dopředného šíření obdrží každý neuron ve vstupní vrstvě vstupní signál a zprostředkuje jeho přenos ke všem neuronům vnitřní vrstvy. Každý neuron ve vnitřní vrstvě vypočítá svou aktivaci a pošle tento signál všem neuronům ve výstupní vrstvě. Každý neuron ve výstupní vrstvě vypočítá svou aktivaci, která odpovídá jeho skutečnému výstupu (m-tého neuronu) po předložení vstupního vzoru. Tímto způsobem v podstatě dostaneme odezvu neuronové sítě na vstupní podnět daný excitací neuronů vstupní vrstvy [26].

A jsme opět u inspirace biologií, takto se totiž šíří signál v biologické nervové soustavě.

Rozšířili jsme signál do ostatních neuronů a nyní vyvstává otázka, jakým způsobem jsou stanoveny synaptické váhy a jak je měnit, abychom se dostali ke korektní odezvě na vstupní signál. Proces stanovení synaptických vah je zde právě řešen metodou zpětného šíření. Na rozdíl od už popsaného dopředného chodu při šíření signálu neuronové sítě spočívá tato metoda adaptace v opačném šíření informace směrem od vrstev vyšších k vrstvám nižším. Během adaptace neuronové sítě metodou backpropagation jsou srovnávány vypočítané aktivace výstupních neuronů s požadovanými výstupními hodnotami pro každý neuron ve výstupní vrstvě a pro každý tréninkový vzor.

Na základě tohoto srovnání je definována chyba neuronové sítě, pro kterou je vypočítán faktor $\sigma_k (k = 1, \ldots, m)$. σ_1 odpovídá chybě prvního výstupního neuronu, σ_2 chybě druhého, atd. Faktor se šíří zpětně do neuronů předcházející vrstvy, které s ním mají definované spojení. Úprava váhových hodnot na spojeních mezi neurony vnitřní a výstupní vrstvy potom závisí na faktoru σ_k a aktivacích neuronů ve vnitřní vrstvě. Obdobným způsobem se šíří chyba z vnitří vrstvy do vrstvy vstupní (je vypočten faktor pro vnitřní vrstvu).

Z uvedeného postupu vychází, že váhy spojů se používají dvakrát – jednou při šíření signálu jedním směrem a šíření chyby směrem druhým.

Ačkoliv jsem v předchozím postupu uvedl, že se chyba počítá přes všechny tréninkové vzory, v praxi to většinou bývá tak, že se neuronové síti postupně předloží tréninkový vzor, po kterém se hned přepočítají váhové spoje a až poté je představen další vzorek.

Aktivační funkce pro neuronové sítě s adaptační metodou backpropagation musí mít následující vlastnosti: – musí být spojitá, diferencovatelná a monotónně neklesající. Nejčastěji používanou aktivační funkcí je proto standardní sigmoida (viz obr. 5.3).

Následuje algoritmus, ze kterého by mělo být zpětné šíření chyby snadno porozumitelné.

Algoritmus [23]

Formální vzorce pro výpočet chyby a změny jednotlivých vah naleznete v knize "Teoretické otázky neuronových sítí"[26] na str.57. My si ukážeme jednodušší a přehlednější postup, konkrétně pro případ s aktivační funkcí sigmoida.

Nejprve se celá síť inicializuje tak, že všem vahám je náhodně přiřazeno malé číslo (např. z intervalu <-1, 1>). Na vstup se potom přivede vzor a je vypočítán výstup celé sítě (dle rov. 5.1). Tento průchod je tzv. *feed-forward*. Jelikož byly hodnoty vah nastaveny

náhodně, výsledek prvního průchodu je zcela odlišný od očekávaného. Proto nyní vypočteme chybu (Error) podle vzorce (5.6),

$$Error = Output * (1 - Output) * (Target - Output) \tag{5.6}$$

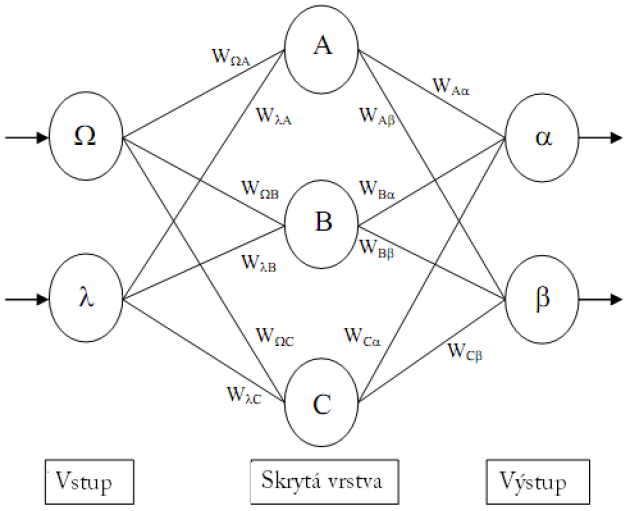
Kde:
Output Aktuálně získaná výstupní hodnota
Target Požadovaná výstupní hodnota, tzv. cíl – Target

Zpětně šíříme tuto chybu a pomocí ní přepočítáme synaptické váhy. W^+ představuje nově vypočítanou váhu a W váhu před výpočtem. $Input$ je hodnota vstupu neuronu.

$$W^+ = W * (Error * Input) \tag{5.7}$$

Dalším průchodem by se měla hodnota výstupu přiblížit požadované a tedy i chyba bude menší. Tento proces provádíme tak dlouho, dokud nedosáhneme kýženého výsledku rozpoznávání nebo dokud není dosaženo požadované velikosti chyby.

Příklad [23]



Obrázek 5.6: Propojení neuronů v síti – příklad

1. Výpočet chyby(Error) výstupních neuronů

$$\delta_\alpha = output_\alpha(1 - output_\alpha)(target_\alpha - output_\alpha) \tag{5.8}$$
$$\delta_\beta = output_\beta(1 - output_\beta)(target_\beta - output_\beta) \tag{5.9}$$

2. Změna vah výstupní vrstvy

$$W^+_{A\alpha} = W_{A\alpha} * \eta\delta_\alpha * output_A \tag{5.10}$$
$$W^+_{B\alpha} = W_{B\alpha} * \eta\delta_\alpha * output_B \tag{5.11}$$
$$W^+_{C\alpha} = W_{C\alpha} * \eta\delta_\alpha * output_C \tag{5.12}$$
$$W^+_{A\beta} = W_{A\beta} * \eta\delta_\beta * output_A \tag{5.13}$$
$$W^+_{B\beta} = W_{B\beta} * \eta\delta_\beta * output_B \tag{5.14}$$
$$W^+_{C\beta} = W_{C\beta} * \eta\delta_\beta * output_C \tag{5.15}$$

3. Výpočet (zpětné šíření) chyby výstupní vrstvy

$$\delta_A = output_A(1 - output_A)(\delta_\alpha W_{A\alpha} + \delta_\beta W_{A\beta}) \tag{5.16}$$
$$\delta_B = output_B(1 - output_B)(\delta_\alpha W_{B\alpha} + \delta_\beta W_{B\beta}) \tag{5.17}$$
$$\delta_C = output_C(1 - output_C)(\delta_\alpha W_{C\alpha} + \delta_\beta W_{C\beta}) \tag{5.18}$$

4. Změna vah výstupní vrstvy

$$W^+_{\alpha A} = W_{\alpha A} * \eta\delta_A * input_\alpha \tag{5.19}$$
$$W^+_{\alpha B} = W_{\alpha B} * \eta\delta_B * input_\alpha \tag{5.20}$$
$$W^+_{\alpha C} = W_{\alpha C} * \eta\delta_C * input_\alpha \tag{5.21}$$
$$W^+_{\Omega A} = W_{\Omega A} * \eta\delta_A * input_\Omega \tag{5.22}$$
$$W^+_{\Omega B} = W_{\Omega B} * \eta\delta_B * input_\Omega \tag{5.23}$$
$$W^+_{\Omega C} = W_{\Omega C} * \eta\delta_C * input_\Omega \tag{5.24}$$

Konstanta η (nazývána learning rate) se používá pro zrychlení nebo naopak zpomalení učícího algoritmu.

Průběh algoritmu [23]

V předchozím odstavci jsme si detailně seznámili s tím, jak se přepočítávají jednotlivé váhy a jak celý algoritmus funguje při předložení jednoho příkladu. Nyní se zaměříme na průběh algoritmu s velkou datovou sadou. Nyní pro jednoduchost předpokládejme, že požadujeme po neuronové síti rozpoznání prvních čtyř číslic, tedy 0, 1, 2, 3.

První správný způsob učení neuronové sítě, nazývaný incrementální, je takový, že nejprve předložíme síti 0 a pouze jednou přepočítáme všechny váhy v síti. Dále předložíme 1 a stále pouze jednou přepočteme všechny váhy, s 2 a 3 provedeme to stejné.

Jakmile jsme přeložili všechny čtyři číslice, vrátíme se opět k prvnímu a proces opakujeme tak dlouho, dokud nedosáhneme požadované chyby nebo se nedostaneme do tzv. stop kriteria, které bude diskutované dále v textu.

Další správnou variantou je tzv. batch algoritmus, kdy jsou váhy přepočítány až po předložení všech dat. Tento způsob však u pattern recognition není příliš využívaný.

Častou chybou začátečníků při učení neuronové sítě bývá právě fakt, že nejprve předloží síti první číslici a upravují váhy, dokud se nedostanou na požadovanou odchylku. Až posléze předloží další číslici a proces opakují.

Pokud to takhle provedeme, nejprve naučíme síť rozpoznávat první číslici, poté ji přimějeme zapomenout a naučit se druhou číslici atd. Na konci tohoto učení je pak neuronová síť schopna rozpoznat pouze poslední předkládanou číslici.

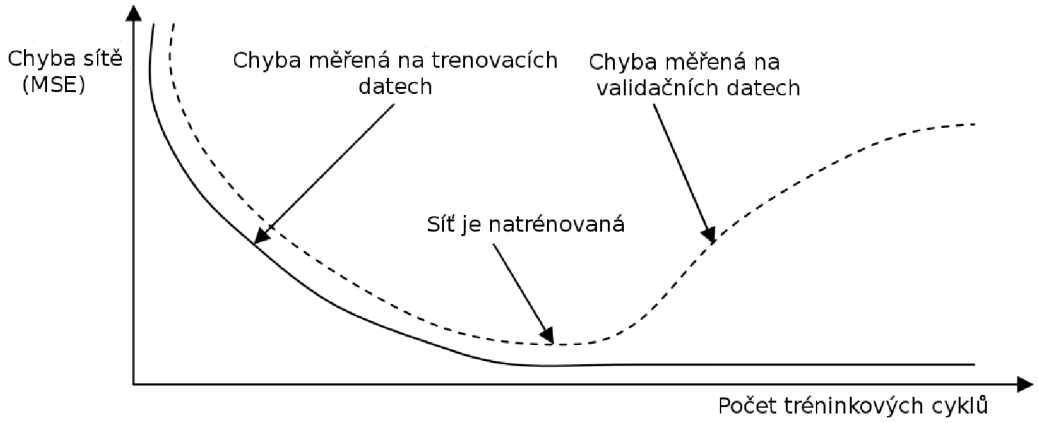
Zastavení učení - stop kriterium [23]

Kdy učení zastavit? Nejlepší by bylo, kdyby síť úspěšně a bezchybně rozpoznala všechny číslice. Jedním z kriterií může být mean square error (MSE).

Jeho výpočet je následující: – pokud p je požadovaný výsledek a a je aktuální hodnota výstupu neuronu, tak square error je $(p - a)$. Pokud máme alespoň dva výstupní neurony, tak mean square error těchto neuronů je průměr těchto dvou square errorů.

A jak tedy zastavit? Nastavením hodnoty přesnosti α, pokud se mean square error, spočtený přes celou trénovací sadu, dostane pod tuto hodnotu, učení zastavíme. Bohužel tato metoda skýtá jednu poměrně velkou nevýhodu. Naše neuronová síť se tímto může dostat do stavu přetrénování (angl. overfitting).

Přetrénování je stav, kdy se síť adaptuje přesně na předkládaná trénovací data a ztratí tak svou schopnost generalizovat. Abychom tomu předešli, používá se v praxi tzv. *validační* sada. Po každé trénovací epoše, se vypočte mean square error na této sadě, aniž by se přepočítávali váhy. Neuronovou síť pak trénujeme tak dlouho, dokud má MSE klesající tendenci.



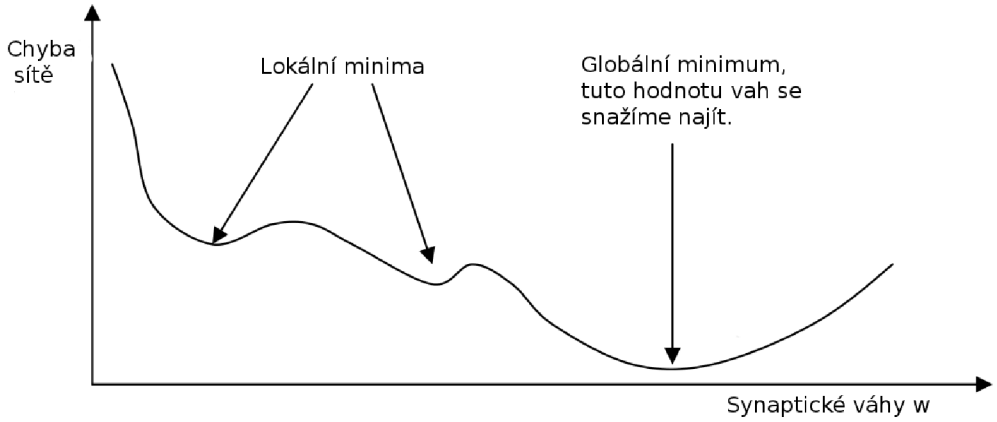
Obrázek 5.7: Chybové křivky při trénování

Backpropagation problem

Velice známý problém backpropagation algoritmu je tzv. *lokální minimum*. Tyto minima se vyskytují, protože algoritmus vždy přepočítává a mění váhy,k aby snížil chybu. Avšak chyba nám může náhle vzrůst, aby pak v globále ještě klesala. Tento problém názorně ukazuje obrázek (5.8).

Problém uváznutí v lokálním minimu můžeme řešit [23, 26]:

- vhodným nastavením počátečních synaptických vah.
  Konkrétní hodnoty parametrů či univerzální strategie výběru neexistují. Tyto parametry jsou silně závislé na trénovací množině, proto je zjištění optimálních hodnot otázkou experimentu. Pokud se dostaneme do lokálního minima je jednoduchým



Obrázek 5.8: Problém lokálního minima

řešením vyresetovat hodnoty synaptických vah na novou náhodnou hodnotu a začít trénovat znovu.

- vhodnou strategií výběru trénovacích vzorů.
  Nejúčinější strategie pro výběr vzorů pro překonání lokálních minim se jeví náhodný výběr. Naopak nejméně odolnou strategií je sekvenční výběr.

- přidat tzv. *momentum* α. Změna synaptických vah nebude záviset pouze na aktuální chybě, ale také na předešlé změně.

Dalším neméně velkým problémem je volba vhodné topologie pro řešení konkrétního problému. Málokdy známe podrobně vztahy mezi vstupy a výstupy, abychom toho mohli využít při návrhu speciální architektury. Většinou se volí topologie s jednou nebo dvěma vnitřními vrstvami neuronů a čeká se, že algoritmus backpropagation zobecní vztahy mezi vstupními a výstupními vzory a promítne je tak do svých synaptických vah. Přesto se musí volit počet neuronů ve skryté vrstvě. Jejich počet by měl odpovídat složitosti řešeného problému – počtu tréninkových vzorů, jejich vstupů a výstupů a také struktuře vztahů, které popisují. Příliš malý počet zastaví většinou algoritmus někde v lokálním minimu a je potřeba doplnit síť o další neurony a dát ji tak větší volnost při učení. Naopak s příliš velkým počtem neuronů je velké riziko přeučení, protože nalezená konfigurace zase příliš zohledňuje trénovací vzory. Nehledě na to, že s vyšším počtem neuronů roste počet synaptických vah a tím roste výpočetní náročnost adaptace. [26]

Sice existují teoretické odhady na horní počet skrytých neuronů, které by měly být postačující pro realizaci libovolné funkce z určité třídy, avšak nejsou pro praktické využití příliš použitelné. Stále je nutnost experimentace. Po adaptaci se v případě velké chyby sítě neuron přidá nebo při špatné generalizaci se naopak několik neuronů ubere a adaptivní režim se zopakuje.

Kapitola 6

Návrh řešení

Z předchozích kapitol je zřejmé, že rozpoznávání znaků z obrazu není jednoduchou úlohou. K tomu abychom dostali požadovaný výsledek vede dlouhá cesta skládající se z několika úkonů, jejichž jednotlivé výsledky přímo ovlivňují úspěšnost rozpoznání. Jaké úkony a v jakém pořadí si řekneme dále v textu.

Výslednou aplikaci můžeme rozdělit do dvou pomyslných částí – *experimenty s neuronovými sítěmi* a *převod textu v obrázku na text elektronický.*

Experimenty s neuronovými sítěmi

Aplikace bude umožňovat tvorbu trénovacích, validačních a testovacích dat ze zadané databáze obrázků podle zvolených příznaků. Dále umožní zvolit topologii sítě (počet neuronů ve skrytých vrstvách[1]), počet trénovacích epoch (cyklů) atd. Po zadání potřebných parametrů lze danou síť natrénovat a také vyzkoušet její úspěšnost nad testovacími daty.

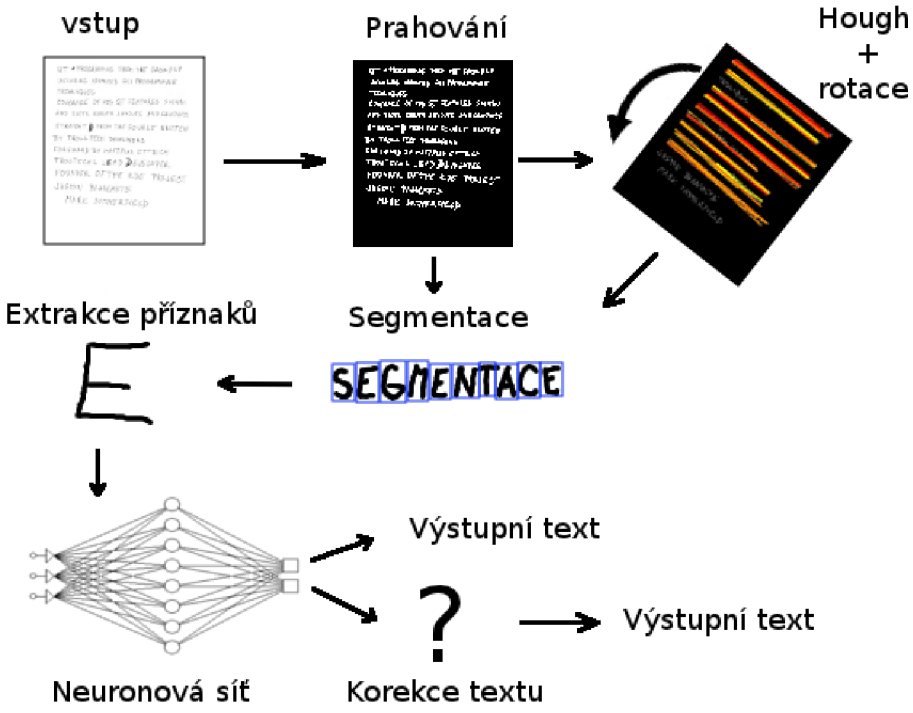
Převod textu z obrazu

Vstupem aplikace bude obrázek s textem v dostatečném rozlišení a velikostí rozpoznávaných znaků. Výstupem bude zobrazení textu v ASCII.
Princip aplikace lze shrnout do několika kroků(viz. obr.6.1) :

1. Načtení samotného obrázku pro rozpoznání
2. Prahování a odstranění šumu
3. Zjištění orientace textu a případné otočení do vodorovné polohy
4. Segmentace jednotlivých znaků
5. Extrakce příznaků
6. Klasifikace neuronovou sítí
7. Výpis výstupního textu
8. Jazyková korekce textu
9. Výstupní text po korekci

Jednotlivé kroky budou detailně probrány v následující kapitole.

[1] Počet vstupních a výstupních neuronů je dán trénovacími daty



Obrázek 6.1: Blokové schéma průběhu programu

Kapitola 7

Implementace

V této kapitole je podrobně popsána vlastní implementace aplikace. Jsou zde detailně vysvětleny jednotlivé kroky programu.

Dále se čtením této kapitoly dozvíte, na jaká úskalí jsem při řešení narazil a způsob jejich řešení.

V minulé kapitole jsem nastínil, že aplikace se dá rozdělit do dvou pomyslných částí. Tyto části jsou vzájemně provázány a jednotlivé principy lze aplikovat na obě části. Proto se budeme věnovat hlavní části – *převodu textu z obrázku do elektronické podoby.*

Ukázkový program je implementován v jazyce C++ a byl navržen tak, aby byl jejich zdrojový kód přenositelný. Vyvíjen a odzkoušen byl na operačním systému Ubuntu 11.10 Oneiric Ocelot.

Pro tvorbu grafického uživatelského prostředí byla použita knihovna *Qt*. Jde o multiplatformní nástroj, který založila v r.1999 firma Trolltech. Od roku 2008 je majetkem firmy Nokia. Aplikace napsané s pomocí této knihovny je možno distribuovat pod licencí *GPL* či *LGPL*. Aplikace byla napsána s verzí Qt4.8.1.

Pro manipulaci s obrazovou informací byla použita externí knihovna *OpenCV* (Open Source Computer Vision), která byla vytvořena firmou Intel. Jde o volně dostupnou knihovnu, která je šířena pod *BSD* licencí. Byla použita verze 2.1.

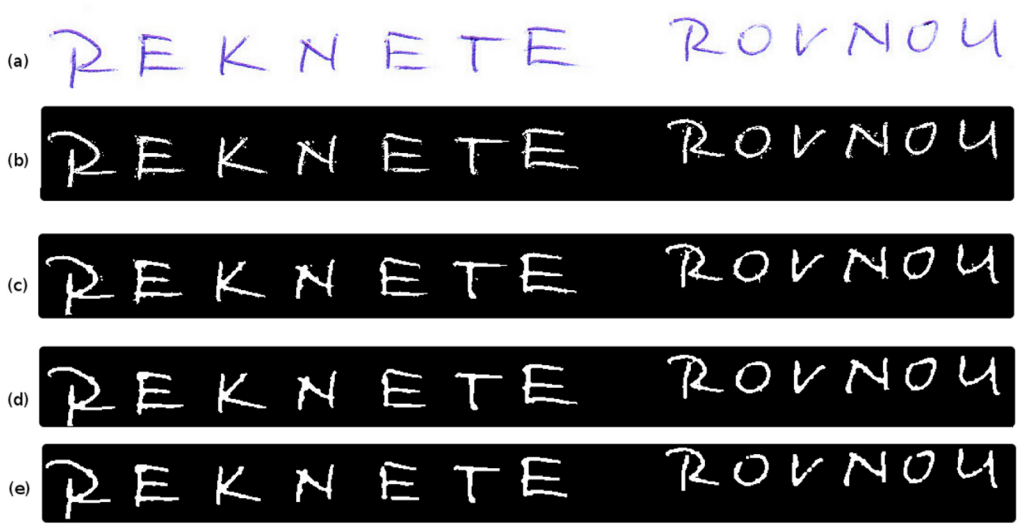
Experimenty s neuronovými sítěmi byly možné díky knihovně *FANN* (Fast Artificial Neural Net). Jde o volně dostupnou *open source* knihovnu implementující vícevrstvé neuronové sítě jak plně, tak částečně propojené. V projektu byla použita verze 2.1.

7.1 Zpracování obrazu

7.1.1 Prahování

Po načtení obrazu je nutné se zbavit barevné informace, která je pro klasifikaci nepotřebná. Toho dosáhneme prahováním jehož teorie je popsána v kapitole Zpracování obrazu (2.1.1). Protože hodnota prahu je pro každý případ specifická, byla experimentálně zjištěna – vz240. Podotýkám, že bylo použito inverzní prahování.

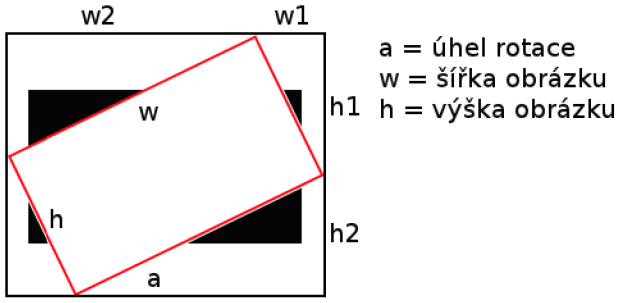
Následně byla použita morfologická operace uzavření (2.1.2). Abychom se potlačili díry ve znacích, či opravili přerušení. Jak je vidět na obrázku (obr. 7.1(b)), vysokou hodnotou prahu jsme zavedli do obrázku šum, který je pro naše účely kontraproduktivní. Zbavení šumu je možné operací otevření, ovšem to bychom opět získali porušení objektů, proto je na místo této operace použit mediánový filtr.



Obrázek 7.1: Předzpracování obrazu: (a) – vstupní obraz, (b) – po prahování, (c) – po uzavření, (d) – po filtraci mediánovým filtrem, (e) – otevření

7.1.2 Orientace textu

Metoda segmentace (dále v textu) předpokládá vodorovný text. Může se ovšem stát, že bude text pootočený třeba vinou skenování či pisatel bude psát tzv. »do kopce« a metoda segmentace zde selže. Proto je nutné sklon řádků detekovat a případnou rotací korigovat. Detekce sklonu je detekována pomocí Houghovy transformace, jejíž teorie je popsána v kapitole (kap. 2.1.4). Ta nalezne úsečky odpovídající řádkům. Výsledný úhel otočení textu potom odpovídá váženému průměru úhlu jednotlivých úseček. Pokud je zjištěno natočení větší než 1° je obraz rotován do vodorovné polohy. Obrázek (7.3) ilustruje detekci a pootočení obrazu do vodorovné polohy. V tomto případě byl detekován úhel $-10,5412°$.

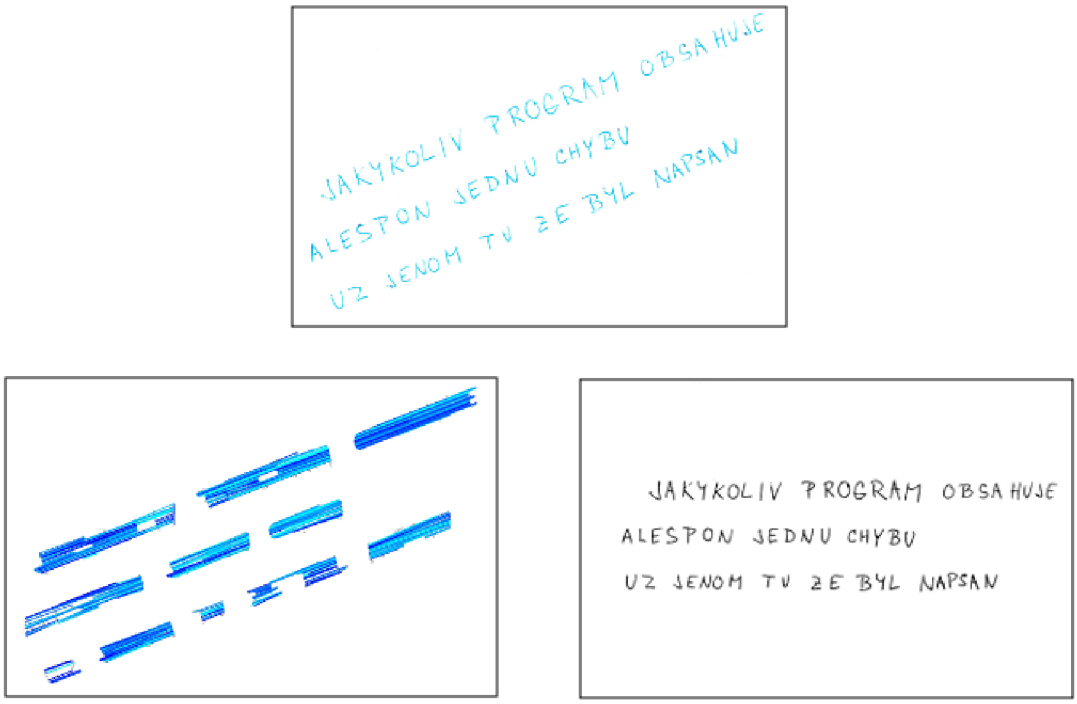


Obrázek 7.2: Ukázka změny rozměrů obrázku při rotaci

Pokud chceme otáčet obraz kolem svého centrálního bodu, musíme si uvědomit, že výsledný obraz mění své rozměry a podle toho se také zařídit, abychom se nepřipravili o cenné informace(obr. 7.2). Musíme tedy zjistit hranice rotovaného obrazu. Naším cílem je tedy

nalézt šířku w' a výšku h' nového obrazu.

$$w' = w_1 + w_2 \tag{7.1}$$
$$h' = h_1 + h_2 \tag{7.2}$$
$$h1 = h * cos\alpha \tag{7.3}$$
$$w1 = w * sin\alpha \tag{7.4}$$
$$h2 = h * sin\alpha \tag{7.5}$$
$$w2 = w * cos\alpha \tag{7.6}$$



Obrázek 7.3: Detekce úhlu a následná rotace obrazu: vstupní obraz, Houghova transformace, otočení do vodorovné polohy

7.1.3 Segmentace

Obrázek má správnou polohu, můžeme nyní pokročit k segmentaci jednotlivých znaků. Program nejprve detekuje pozici jednotlivých řádků vytvořením horizontálního histogramu, kdy ukládá počet bílých pixelů na řádku obrazu. Ze zjištěných hodnot histogramu je pak možné zjistit počet, pozici a výšku jednotlivých řádků.

Obdobně pak probíhá segmentace jednotlivých znaků. Pro každý takto detekovaný řádek je vytvořen vertikální histogram, z něhož lze zjistit nejen polohu a rozměry celých slov, ale také jednotlivých znaků.

Jelikož na vstupu předpokládáme obrázek s rukou psanými znaky, je pro takto detekovaný znak ještě vypočítán horizontální histogram. Děláme to proto, abychom zjistili skutečnou výšku znaku, neboť pozice a rozměry písmene povětšinou nejsou stejné a pro správnou klasifikaci je potřeba nalézt znak s minimálním ohraničením.

Při výzkumu a sběru vzorků dat, jsem si všiml, že většina lidí nedotahuje písmena až do konce a vznikají tak někdy přerušené znaky. Proto program spočítá průměrnou mezeru mezi znaky a pokud je mezera krajně menší než průměrná, je ignorována.

Algoritmus segmentace funguje správně, pokud jsou řádky textu napsané rovně a mají dostatečné svislé rozestupy. Jinak nalezené řádky splývají a pro korekci je nutné upravit parametry algoritmu. Pro bezproblémové rozpoznání znaků je tedy nutná i menší dávka spolupráce pisatele.

7.1.4 Extrakce příznaků

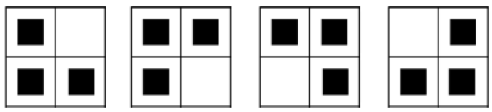
Pro adaptivní i aktivní režim neuronové sítě byly vyzkoušeny příznaky jejichž extrakce je pochopitelná již z teorie, která je popsána v kapitole 3. Na tomto místě si proto detailněji rozebereme jen některé z nich.

Freemanův kód

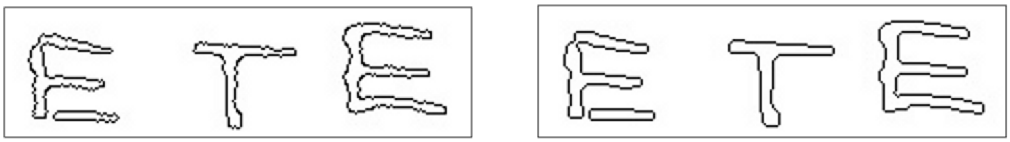
Jak je z teorie patrné, pro získání Freemanova kódu potřebujeme získat nepřerušenou konturu znaku. Pro získání těchto kontur jsem předpokládal vhodný výběr z hranových detektorů uvedených v sekci 2.1.3. Nejlepších výsledků bylo dosaženo použitím Cannyho hranového detektoru. Avšak v některých případech docházelo k porušení některých kontur, pravděpodobně vlivem komprese, nebo byly hrany »roztřepené«. Proto jsem se rozhodl pro morfologickou detekci vnějšího obrysu. Při této detekci provedeme dilataci (kap. 2.1.2) obrazu a vyhodnotíme nonekvivalenci (XOR) tohoto upraveného obrazu s obrazem původním a získáme tak přesně jednopixelové souvislé hranové obrysy objektů.

Abychom mohli přistoupit k výpočtu Freemanova kódu, je zapotřebí se zbavit tzv. *rohů* (obr. 7.4), neboť algoritmus počítá pouze s posunem po křivce pouze v jednom směru.

Redukce nepotřebných pixelů je řešena skeletonizací (viz kap. 2.1.2).



Obrázek 7.4: Čtyři typy rohů



Obrázek 7.5: Ukázka detekce obrysu: vlevo–Cannyho hranový detektor, vpravo – morfologická operace. Obrázek byl invertován.

Nyní již nám nic nebrání v převodu obrysu na výsledný Freemanův kód a vytvořit z něj normalizovaný histogram (viz kap. 3.3). Dostaneme tedy 8-bitový příznakový vektor typu *float*, který předložíme neuronové síti. Jak se mi potvrdilo, tento příznakový vektor je natolik malý, že úspěšnost klasifikace není velká. Nicméně jsem byl překvapen, že z takto málo informací se úspěšnost klasifikace číslic pohybovala kolem 45 %.

Pro zvýšení úspěšnosti jsem se nechal inspirovat článkem Husseina El Saadiho [16]. Jsou zde stále použity principy Freemanova kódování, ale výsledný příznakový vektor je daleko větší. Princip výpočtu je následující:

1. Vypočteme tzv.*centroid*. Jedná se o bod, jež leží v centru objektu. X-ová/y-ová souřadnice X_c/Y_c je součtem všech x-ových/y-nových souřadnic objektu podělená počtem pixelů v objektu:
$$X_c = \frac{\sum x}{\sum f(x,y)}$$
$$Y_c = \frac{\sum y}{\sum f(x,y)}$$
2. Zjistíme nejdelší vzdálenost od centroidu. Vzdálenost mezi pixely vypočítáme podle Pythagorovy věty:
$$vzdalenost = \sqrt{(X_c - x)^2 + (Y_c - y)^2}$$
3. Vhodně si rozdělíme tuto vzdálenost čímž získáme kruhové sektory. Tyto sektory si dále rozdělíme do výsečí (360°/ počet výsečí).
4. Nyní procházíme obrysem znaku a převádíme na Freemanův kód. Číslo směru ovšem uložíme do příslušného sektoru na základě vzdálenosti od centroidu a do příslušné výseče na základě vypočteného úhlu:
$$\alpha = \arctan(\frac{(y-Y_c)}{(x-X_c)}$$
5. Pro tuto práci jsem zvolil tři sektory se čtyřmi výseči. Získal jsem tedy 8 směrů × 3 sektory × 4 výseče = vektor s 96 příznaky taktéž typu *float*.

Shrnutí příznaků

Příznaky jsou extrahovány ze vzorků 20 × 20 pixelů.

1. Intenzita pixelů – binární vektor se 400 příznaky.
2. PCA – proměnná délka vektoru(200 - 400) s hodnotami typu float.
3. Hu momenty – 7 hodnot typu float
4. GSC – 512-bitový binární vektor.

7.1.5 Klasifikace neuronovou sítí

Po předchozích krocích se program dostal do fáze, kdy jsou vypočítány charakteristiky znaků a jako vektor jsou předány na vstup již adaptované neuronové sítě. Příznaky jsou předávány postupně po znaku funkci z knihovny FANN *fann_run*, jejíž výstupem je pole hodnot typu float. V ideálním případě se v poli, pokud je znak správně rozpoznán, objeví pouze jedna hodnota 1.000 nebo jen o něco málo nižší právě na místě odpovídající pořadí znaku v abecedě.

Ovšem s ideálními případy se v životě setkáváme málo. Je možné, že se ve výsledku objeví více těchto hodnot, většinou na místech znaků, které jsou si podobné (nebo se tak jeví neuronové síti), nebo jsou hodnoty příliš nízké. Výsledným znakem je potom ten, na

jehož pozici je nejvyšší hodnota. Je-li výsledek kladný je vypočítáno procentuelní zastoupení výsledku: *procento = hodnota na pozici / suma všech kladných hodnot*. Tato hodnota pak nese informaci, jak kvalitně byl tento znak rozpoznán. Je-li na pozici záporné číslo blížící se -1, lze tento znak považovat za nerozpoznaný.

Výstupem neuronové sítě je znak a její hodnota udávající jistotu rozpoznaného znaku, který je potom vypsán a dále předán další funkci, která provádí korekce nad celými slovy.

7.1.6 Korekce textu

Pro naši práci je dostačující nějaký jednoduchý algoritmus na korekci textu. Inspiraci jsem nalezl v tutoriálu [14]. Program je zde uveden v Pythonu a je dostupný i v dalších jazykových modifikacích. Abych lépe pochopil činnost korekce, rozhodl jsem se nevyužít již hotové řešení, ale pokusit se jej přepsat sám – *SpellCorrect.hpp*.

Jak tato korekce funguje? Vstupem funkce je slovo a my se snažíme vrátit nejpravděpodobnější pravopisnou korekci tohoto slova. Bohužel neexistuje způsob, jak s jistotou vybrat správnou korekci (např. pro slovo Manželk, má se opravit na Manžel nebo Manželka?). Snažíme se tedy najít takovou korekci k, ze všech možných oprav, která maximalizuje pravděpodobnost k vzhledem k původnímu slovu s:

$$argmaxc\ P(k|s)$$

Podle Bayesova teorému je to ekvivalentní s:

$$argmaxc\ P(s|k)P(k)/P(s)$$

Vzhledem k tomu, že P(s) je pro každou korekci stejné, můžeme jej ignorovat a dostaneme:

$$argmaxc\ P(s|k)P(k) \tag{7.7}$$

- P(k) je pravděpodobnost, že navrhovaná korekce platí sama o sobě. Jedná se o jazykový model, udává jaká je pravděpodobnost výskytu v textu. P(„a“) je mnohem pravděpodobnější než P(„xzx“), která se blíží k nule.
- $P(s|k)$ je pravděpodobnost, že do textu bylo napsáno s, když autor chtěl k.
- argmaxc je kontrolní mechanismus, který vybere ze všech možných korekcí tu, která dává nejlepší kombinaci pravděpodobnostního skóre.

A jak tedy samotný program funguje?

1. $P(k)$. Načteme soubor, který obsahuje všechna česká slova bez diakritiky. A nějaký dlouhý text. Vše převedeme na velká písmena a odstraníme nonalfanumerické znaky. Poté extrahujeme jednotlivá slova a uložíme si je do hash tabulky (Dictionary), která udržuje informaci i o počtu jednotlivých slov. Kombinace korekcí jsou potom porovnávány s těmito známými slovy.
2. Nyní se podíváme na problém výčtu všech možných oprav k daného slova s. Úpravy mohou být následující: vypustíme nebo naopak přidáme písmeno, prohodíme sousední písmena nebo změníme písmeno za jiné. Tím získáme celkem velkou sadu kandidátů. Pro slovo délky n, dostaneme n kandidátů po mazání, 26(n+1) po vložení, n-1 po výměně a 26n po záměně písmene. Dohromady tedy 54n + 25 kombinací, z nichž některé jsou ovšem duplicity. Tyto kombinace zajišťuje funkce *makeAlternative*.

Většina překlepů se děje ve vzdálenosti 1. My ovšem použijeme program pro korekci textu z obrázku. Může se stát že budou segmentována 2 písmena jako jedno(v případě, že se dotýkají), proto budeme hledat i korekce ve vzdálenosti 2. Pokud není nalezeno slovo ve slovníku při korekcích ve vzdálenosti 1, je na vstup funkce makeAlternative přiveden výstup té stejné funkce. Tím nalezneme kombinace korekce ve vzdálenosti 2.

3. Pro pravidlo $P(s|k)$ je aplikován triviální rozhodovací model, jež říká, že všechny korekce ve vzdálenosti jedna od slova jsou pravděpodobnější než korekce ve vzdálenosti 2, ale mnohem méně pravděpodobné než korekce ve vzdálenosti 0.

Kapitola 8

Trénování neuronové sítě

Neuronová síť je základním prvkem navrhovaného klasifikátoru rukou psaných písmen. Doteď jsme se zabývali pouze přípravou koeficientů, které budou vstupovat do neuronové sítě. Samotný proces rozpoznávání zabezpečuje již natrénovaná (adaptovaná) síť. To že je síť adaptovaná znamená, že její váhové koeficienty jsou již optimálně nastavené pro prostředí, ve kterém se bude používat. V našem případě tedy budeme hledat optimální váhové koeficienty nad databází rukou psaných písmen.

Na klasifikaci je použit vícevrstvý perceptron s trénovacím algoritmem zpětného šíření chyby (backpropagation)(kap. 5.5.2). Jak jsem již v teorii zmínil, jedná se o neuronovou síť, jejíž úlohou je klasifikovat vzory do předem známých tříd. V našem případě bude neuronová síť klasifikovat vzory do deseti tříd při rozpoznávání rukou psaných číslic a do 26 tříd při rozpoznávání rukou psaných písmen.

8.1 Návrh neuronové sítě

Návrh vhodné topologie neuronové sítě je závislý od vstupních dat. Počet vstupních neuronů je roven počtu příznaků získaných extrakcí znaku z obrázku, stejně tak je počet výstupních neuronů udán počtem klasifikačních tříd. Počet skrytých vrstev a počet jejich neuronů je otázkou experimentu. Podle Kolmogorova theorému (lit. [10]) může neuronová síť s dvěmi skrytými vrstvami oddělit jakýkoliv problém. Zvýšením počtu neuronů v první skryté vrstvě, zvýšíme počet oddělovačů v systému. Zvýšením počtu neuronů ve druhé skryté vrstvě, zvýšíme počet oddělovaných regionů. Testy tedy můžeme omezit na sítě s jednou a s dvěmi skrytými vrstvami.

Nalezení optimálních váhových koeficientů ovlivňují i další faktory:

- *Samotná charakteristika dat.*
- *Volba přenosové funkce.* Pro trénování neuronových sítí byly testovány dvě přenosové funkce. V prvním případě byla použita sigmoidní přenosová funkce. Na výstupu neuronové sítě jsem požadoval 1 na pozici odpovídající znaku a 0 na ostatních pozicích. Ve druhém případě jsem použil přenosovou funkci hyperbolické tangenty a to z toho důvodu, že na výstupu sítě jsem požadoval 1 na pozici odpovídající znaku a na ostatních pozicích -1 (značí jak moc si je síť jistá, že to není tenhle znak). Průběhy těchto funkcí jsou na obrázku (obr. 5.3).
- *Tzv. learning rate.* Learning rate je parametr neuronové sítě, který udává rychlost trénování. Čím větší tento parametr bude, tím rychleji se bude neuronová síť učit.

Hodnota tohoto parametru by měla být v intervalu (0, 1). Matematicky tento parametr znamená, jak velké změny váhových koeficientů se provedou. Pokud budou změny příliš velké nemusíme nalézt minimální chybu, neboť může dojít k přeskočení této hodnoty na hodnotu vyšší.

Nejprve si nastavíme tento parametr na vyšší hodnotu, řekněme na parametr 0,9 a až se budeme blížit k optimální chybě na validačních datech, budeme tuto hodnotu snižovat. Tím dosáhneme větší přesnosti adaptace.

- *Stop kritérium.* Jeden z velmi důležitých parametrů sítě. Podrobně je rozebrán již v teorii neuronových sítí (viz kap. 5.5.2). Nejvyšší prioritu pro zastavení má průběh MSE na validační sadě, jakmile začne mít funkce vzrůstající hodnotu zastavíme. Druhou v pořadí je maximální chyba MSE na trénovacích datech a v poslední řadě sledujeme maximální počet iteračních cyklů.

8.2 Experimenty

Cílem experimentů je nalezení optimálních váhových koeficientů neuronové sítě kombinací vhodných příznaků a jejich parametrů pro rozpoznávání rukou psaných písmen. Bude také sledováno jak moc ovlivní rozhodovací proces změna některých parametrů neuronové sítě.

Abychom vůbec mohli přistoupit k trénování neuronové sítě, je zapotřebí obstarat si vhodnou databázi izolovaných znaků. Já jsem zde použil dataset z mé předchozí práce ([7]). Protože měl tento dataset různorodý počet jednotlivých znaků, rozhodl jsem se testování rozdělit do více částí. První sada bude obsahovat stejný počet znaků a druhá naopak všechny dostupné. Jelikož naměřená data nad tímto datasetem nelze s nikým porovnat a jinou volnou databázi rukou psaných hůlkových písmen jsem nenalezl, zařadil jsem i několik testů nad databází MNIST[1].

Dataset 1

Tento set obsahuje 300 vzorků od každého znaku abecedy v trénovací množině. Sto vzorků od každého ve validační a 124 vzorků v množině testovací.

Dataset 2

Tento set obsahuje také písmena a jednotlivé zastoupení znaků zobrazuje tabulka (tab. 8.1).

Dataset 3

Databáze rukou psaných číslic MNIST. Obsahuje 4000 trénovacích vzorků od každého znaku, kolem 2000 validačních a 1000 testovacích vzorků.

Všechny tyto datasety obsahují obrázky o velikosti 20 × 20 pixelů. Proto abychom mohli síť trénovat, potřebujeme textový soubor s příznaky trénovacích dat. Klikneme v programu na *"Vytvořit trénovací data"* a vybereme složku, ve které se nachází roztříděné obrázky v jednotlivých složkách pojmenované podle příslušného znaku. Vybereme příznaky, které chceme uložit. Program projde jednotlivé složky a uloží příznaky do textového souboru ve správném formátu:

[1] Volně stažitelná databáze rukou psaných číslic, dostupná na adrese http://yann.lecun.com/exdb/mnist/

počet_vzorků délka_příznakového_vektoru počet_klasifikačních_tříd
příznakový_vektor_hodnota1 příznakový_vektor_hodnota2 ...
výsledný_vektor_hodnota1 ...
...

Stejným způsobem vytvoříme validační soubor.

Testovací soubor vytvoříme obdobným způsobem. Klikneme na *"Vytvořit testovací data"* a vybereme buď složku s obrázky nebo samostatný obrázek. Program extrahuje požadované příznaky a uloží je do souboru jehož formát je následující:
počet_vzorků délka_příznakového_vektoru
příznakový_vektor1_hodnota1 příznakový_vektor_hodnota2 ...
příznakový_vektor2_hodnota1 příznakový_vektor2_hodnota2 ...
...

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
Trénovací	1800	650	600	900	1100	454	482	680	1400
Validační	699	150	130	200	281	100	100	116	242
Testovací	804	149	136	233	300	100	100	120	300
	J	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**
Trénovací	400	1300	1000	900	1295	1499	731	300	1500
Validační	129	265	150	150	250	324	150	100	240
Testovací	140	300	173	174	300	400	150	124	272
	S	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	
Trénovací	1100	1099	848	900	399	300	500	700	
Validační	200	200	143	132	100	100	100	130	
Testovací	216	228	150	150	109	132	149	150	

Tabulka 8.1: Tabulka ukazuje počet jednotlivých znaků

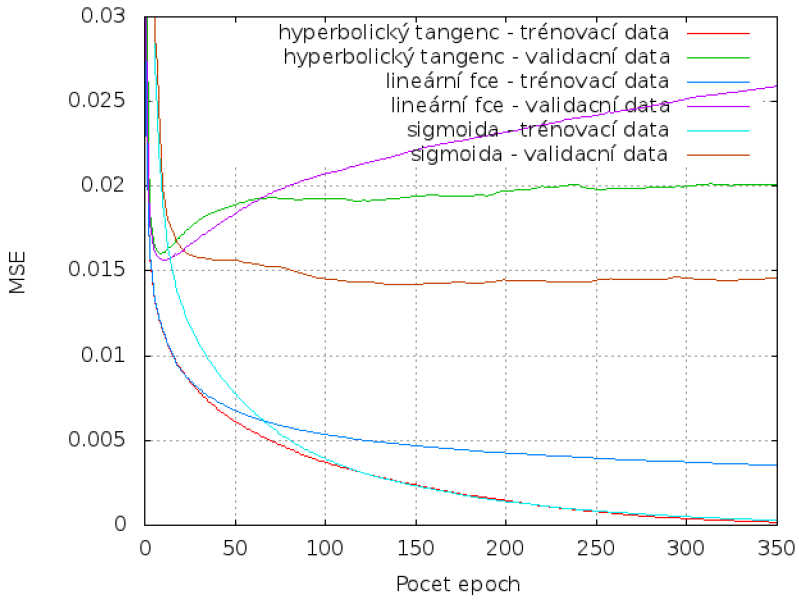
Neuronovou síť natrénujeme kliknutím na *"Trénování sítě"*. Zadáme potřebné parametry a necháme síť adaptovat své váhy. Poté ji je možné otestovat volbou *"Testování sítě"*. Program vyhodnotí data a uloží výsledky do dvou textových souborů. V prvním jsou uloženy hodnoty výstupních neuronů a ve druhém je identifikace znaku s hodnotou jakou byl klasifikován.

Nyní se již přesuneme k samotným experimentům.

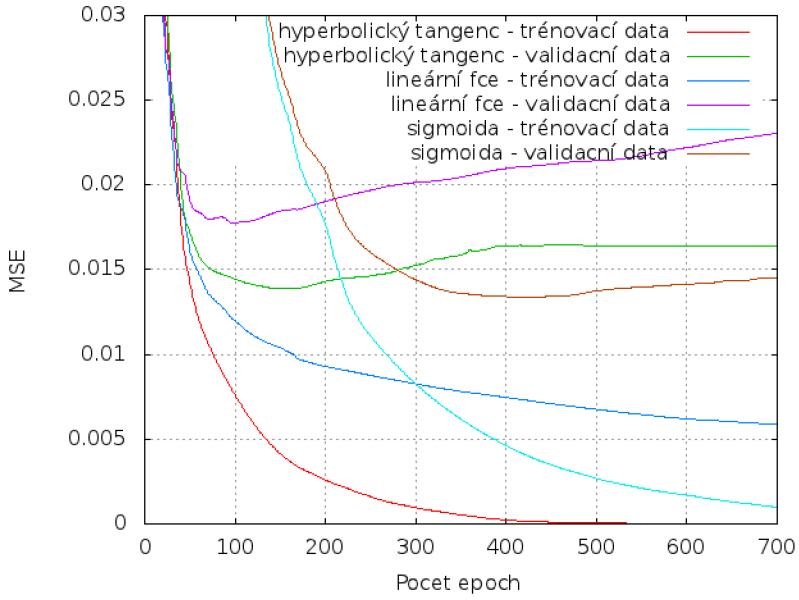
8.2.1 Výběr přenosové funkce

Prvními pokusy bylo potřeba zjistit vhodnou aktivační funkci neuronů ve skrytých vrstvách. Jak je vidět na přiložených grafech (viz obr. 8.1, obr. 8.2) u neuronových sítí s jednou skrytou vrstvou dosahuje nejlepších vlastností sigmoidní přenosová funkce. Chybová křivka na validačních datech pozvolna klesá a její minimum je hodně pod hranicí minim ostatních dvou funkcí.

U neuronových sítí s dvěmi skrytými vrstvami už to tak jednoznačné není. Neuronová síť s aktivační funkcí hyperbolický tangenc se stačí adaptovat již po několika epochách, její chybovost je však stále větší než se sigmoidní funkcí.



Obrázek 8.1: Chybové křivky při trénování dvouvrstvé neuronové sítě



Obrázek 8.2: Chybové křivky při trénování třívrstvé neuronové sítě

Pro naše účely je tedy vhodné zvolit sigmoidní přenosovou funkci. Nalezení optimálního řešení vyžaduje více iterací, ale dosáhneme větší přesnosti při nalezení stop-kriteria a tím lepší adaptace na data.

Zkratky

Zkratky, které budou použity v tabulkách:

- [400 30 26] – takto budeme značit topologii sítě, 400 vstupních neuronů, 30 neuronů ve skryté vrstvě a 26 výstupních neuronů.
- $U[\%]$ – úspěšnost klasifikace znaku
- $U_c[\%]$ – celková úspěšnost klasifikátoru
- Hit – počet správně klasifikovaných znaků

Úspěšnosti $U[\%]$ a $U_c[\%]$ jsou vypočítány ze vztahu *počet správně klasifikovaných/celkový počet.*

8.2.2 První testy

První testy byly provedeny nad datasetem 1 a jako příznakový vektor byly brány hodnoty jasu jednotlivých pixelů.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
$U[\%]$	62.09	76.61	87.90	71.77	77.41	83.87	74.19	75.81	90.32
	J	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**
$U[\%]$	85.48	**45.97**	90.32	81.45	70.97	69.35	84.68	72.58	66.13
	S	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	
$U[\%]$	77.42	91.13	79.03	85.48	90.32	85.48	72.58	75.81	

Tabulka 8.2: Úspěšnost rozpoznávání znaku NS[400 30 26], $U_c = 77{,}35\,\%$

Celková úspěšnost U_c je sice 77,35%, ale z tabulky je patrné, že rozložení úspěšnosti není rovnoměrné. NS se nedokázala adaptovat na písmeno K. Postupným přidáváváním neuronů do skryté vrstvy se úspěšnosti začínají srovnávat. Nejlepších výsledků s jednou skrytou vrstvou bylo dosaženo s NS[400 180 26] s $U_c = 78\,\%$. Od hranice 230 neuronů ve skryté vrstvě pomalu ztrácí NS schopnost generalizace.

Lepší volbou se ukázala volba NS s dvěmi skrytými vrstvami (tab. 8.3).

Klasifikátor je tak úspěšný jako jeho nejslabší článek – D (67,74 %), což pro naši práci není dostačující. Pokusíme se ji vylepšit.

Zajímavou volbou se mi jevilo použití PCA (viz kap. 3.1.1). Za cenu minimální ztráty informace zredukujeme příznakový vektor, jehož důsledkem se sníží i výpočetní náročnost neuronové sítě. Předpokládal jsem, že redukce šumu také pozitivně ovlivní rozhodování neuronové sítě. Ovšem výsledky mi můj předpoklad vyvrátily. Úspěšnost klasifikace se nezlepšila, ba naopak byla o něco horší a to i při zachování všech dimenzí. Pozitivním zjištěním bylo alespoň to, že při zachování relativně stejné úspěšnosti bylo možné zredukovat počet dimenzí na 250 a urychlit tak adaptivní činnost NS.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
$U[\%]$	76,61	72.58	81.45	**67.74**	85.48	87.10	70.97	75.81	90.32
	J	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**
$U[\%]$	77.42	70.16	90.32	83.06	73.39	71.77	83.06	68.55	68.55
	S	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	
$U[\%]$	80.65	88.71	82.64	85.48	83.06	76.61	75.81	73.39	

Tabulka 8.3: Úspěšnost rozpoznávání znaku NS[400 80 26 26], Dataset 1, $U_c = 78{,}42\,\%$, nejlepší konfigurace pro jas pixelů.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
$U[\%]$	76,31	77.58	80.45	**66.84**	84.48	84.10	72.97	74.81	89.32
	J	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**
$U[\%]$	77.42	68.16	85.32	82.06	76.39	77.77	81.06	69.54	68.55
	S	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	
$U[\%]$	80.51	81.71	80.64	83.48	85.06	78.61	73.81	73.39	

Tabulka 8.4: Úspěšnost rozpoznávání znaku NS[250 70 26 26], Dataset 1, $U_c = 77{,}95\,\%$, Nejlepší konfigurace pro PCA.

8.2.3 Freemanův kód

Jak jsem již psal výše, tak klasifikace pouze s jedním normalizovaným histogramem nebyla vůbec úspěšná. Sedm příznaků je pro 26 klasifikačních tříd málo. Avšak použitím algoritmu popsaným dříve (viz kap. 7.1.4) jsme se dostali k mnohem lepším výsledkům. Již od prvních pokusů bylo zřejmé, že jsme se vydali správným směrem.

Nepsané pravidlo pro odhad počtu skrytých neuronů je rovno součtu vstupních a výstupních neuronů poděleno dvěmi. V našem případě tedy 66 neuronů. Výsledky ukazuje tabulka (tab. 8.5). Celková úspěšnost již přesáhla 90 %, nejhůře rozpoznatelné je písmeno R.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
Hit	115	112	113	110	121	118	100	113	118
$U[\%]$	92.74	90.32	91.13	88.71	97.58	95.16	80.65	91.13	95.16
	J	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**
Hit	119	106	122	109	106	103	114	105	**95**
$U[\%]$	95.97	85.48	98.39	87.90	85.48	83.06	91.94	84.68	**76.61**
	S	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	
Hit	114	120	105	111	113	117	115	114	
$U[\%]$	91.94	96.77	84.68	89.52	91.13	94.35	92.74	91.94	

Tabulka 8.5: Úspěšnost rozpoznávání znaku NS[96 66 26], Dataset 1, $U_c = 90.20\,\%$

Zvyšováním počtu neuronů NS ztrácí schopnost generalizace, dochází k přetrénování. Naopak snížení počtu vykazuje lepší výsledky (tab. 8.6).

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
Hit	107	100	115	111	119	116	113	121	117
$U[\%]$	86.29	80.64	92.74	89.52	95.97	93.55	91.13	97.58	94.35
	J	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**
Hit	117	106	117	103	106	109	109	110	**99**
$U[\%]$	94.35	85.48	94.35	83.06	85.48	87.90	87.90	88.71	**79.84**
	S	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	
Hit	110	116	101	112	116	118	112	115	
$U[\%]$	88.71	93.55	81.45	90.32	93.55	95.16	90.32	92.74	

Tabulka 8.6: Úspěšnost rozpoznávání znaku NS[96 15 26], Dataset 1, $U_c = 89{,}80\,\%$, Nejlepší konfigurace, počet epoch 13745. Nejhůře klasifikovatelné je písmeno R, častokrát zaměňováno za písmeno K a A. Zajímavé je, že je ve skryté vrstvě méně neuronů než je klasifikačních tříd. Neuronová síť si vybírá pouze důležité charakteristické vlastnosti.
Na první pohled se může zdát, že tato konfigurace je horší než ta předchozí. Celková úspěšnost je nepatrně horší, ale celkové rozložení je na lepší úrovni.

Dataset č. 2 nám umožní získat relevantnější výsledky a zároveň nám poskytne informace, zda záleží na stejném počtu znaků v trénovací sadě. Zda se neuronová síť adaptuje na jeden typ znaku na úkor druhého, nebo bude vykazovat lepší výsledky díky větší databázi.

	A	**B**	**C**	**D**	**E**	**F**	**G**
Hit	774/804	120/149	117/136	203/220	285/300	96/100	90/100
$U[\%]$	96.27	80.54	86.03	92.27	95.0	96.0	90.0
	H	**I**	**J**	**K**	**L**	**M**	**N**
Hit	99/120	293/300	128/138	278/300	167/173	154/173	253/273
$U[\%]$	82.5	97.67	92.75	92.67	96.53	89.02	92.67
	O	**P**	**Q**	**R**	**S**	**T**	**U**
Hit	371/399	137/147	**93/120**	253/271	212/216	221/228	144/150
$U[\%]$	92.98	93.20	**77.5**	92.67	98.15	96.93	96.0
	V	**W**	**X**	**Y**	**Z**		
Hit	142/150	102/109	126/132	132/149	143/150		
$U[\%]$	94.6	93.58	95.45	88.59	95.3		

Tabulka 8.7: Úspěšnost rozpoznávání znaku NS[96 30 26], Dataset 2, $U_c = 91{,}10\,\%$, Nejlepší konfigurace, počet epoch 16768. Nejhůře klasifikovatelné je opět písmeno R, často klasifikované jako K. Obavy o preferenci jednoho znaku se nepotvrdily, zvětšení databáze má pozitivní efekt, i když nejsou znaky zastoupeny rovnoměrně.

Jsem si vědom toho, že dosud předkládané výsledky nejsou moc relevantní. Proto jsem provedl pokusy na již zmiňované databázi MNIST. Většina publikací právě vztahuje výsledky k rozpoznávání rukou psaných číslic, proto nám tato informace dodá představu o kvalitě uvedených postupů.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$U[\%]$	98.26	99.21	97.18	96.83	95.93	96.08	97.70	96.60	**93.22**	93.76

Tabulka 8.8: Úspěšnost rozpoznávání znaku NS[96 40 10], MNIST, $U_c = 96{,}52\,\%$, Nejlepší konfigurace, počet epoch 15081. Největší problémy měla NS s rozpoznáváním číslice 8. Je vidět, že s větší datovou sadou roste i úspěšnost klasifikace. Na druhou stranu je nutné také podotknout, že počet klasifikačních tříd nám klesl skoro na třetinu. Nicméně chybovost menší jak 3,5 % lze považovat za dobrý výsledek.

8.2.4 GSC

Zde již uvedu pouze nejlepší výsledky. Postup hledání optimální konfigurace je stejný. Postupně přidáváme neurony a sledujeme průběh trénování a porovnáváme výsledky nad testovacími daty. S příliš velkým počtem neuronů uvízneme v lokálním minimu a naopak příliš malý počet neuronů zase nedokáže korektně oddělit jednotlivé vzory.

GSC patří k pokročilým technikám popisu objektu, proto se daly očekávat nejlepší výsledky. Avšak na této databázi se tato technika nedokázala dostatečně projevit.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
Hit	118	**99**	119	117	119	117	105	112	120
$U[\%]$	95.16	**79.83**	95.97	94.35	95.97	94.35	84.68	90.32	96.77
	J	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**
Hit	121	112	115	102	111	116	110	112	110
$U[\%]$	97.58	90.32	92.74	82.26	89.52	93.55	88.71	90.32	88.71
	S	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	
Hit	114	123	102	109	121	116	105	116	
$U[\%]$	91.94	99.19	82.26	87.90	97.58	93.55	84.68	93.55	

Tabulka 8.9: Úspěšnost rozpoznávání znaku NS[512 40 26], Dataset 1, $U_c = 91{,}22\,\%$, Nejlepší konfigurace GSC, počet epoch 348. Nejhůře klasifikovatelné je písmeno B, docházelo k záměnám za písmeno D a R.

Při klasifikaci písmen jsem dosahoval zajímavých výsledků s třívrstvými sítěmi, kde ve druhé skryté vrstvě byla polovina počtu výstupních neuronů.

Jak jsem již psal výše, druhá vrstva se dá chápat jako počet oddělovacích regionů. Zajímalo mě, jakých výsledků dosáhneme nad velkou databází, nad databází MNIST.

K mému udivení s menším počtem tříd bylo třeba volit větší počet neuronů. Do 25 neuronů NS uvízla vždy po 21 epochách v lokálním minimu. Volbou parametru momentum byla sice posléze překonána, nicméně se vždy dala najít lepší konfigurace bez použití tohoto parametru.

	A	**B**	**C**	**D**	**E**	**F**	textbfG
Hit	779/804	**124/149**	123/136	194/220	281/300	91/100	89/100
$U[\%]$	96.89	**83.22**	90.44	88.18	93.67	91.0	89.0
	H	**I**	**J**	**K**	**L**	**M**	**N**
Hit	102/120	295/300	130/138	273/300	167/173	152/173	255/273
$U[\%]$	85.0	98.33	94.20	91.0	96.53	87.86	93.41
	O	**P**	**Q**	**R**	**S**	**T**	**U**
Hit	381/399	135/147	110/120	251/271	209/216	222/228	139/150
$U[\%]$	97.94	91.84	91.67	92.62	96.76	97.37	92.67
	V	**W**	**X**	**Y**	**Z**		
Hit	145/150	104/109	124/132	133/149	144/150		
$U[\%]$	96.67	95.41	93.94	89.26	96.0		

Tabulka 8.10: Úspěšnost rozpoznávání znaku NS[512 50 26], Dataset 2, U_c = 94,07 %, Nejlepší konfigurace GSC, počet epoch 453. Opět nejhorší výsledky vykazovalo písmeno B. Nicméně s touto konfigurací jsem dosáhl nejlepších výsledků nad testovacími daty, největší chybovost je pod 17 %. Myslím, že nad takto malou datovou sadou jde o dobrý výsledek.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$U[\%]$	98.88	99.36	98.40	96.34	96.93	97.08	97.70	97.61	**94.22**	96.76

Tabulka 8.11: Úspěšnost rozpoznávání znaku NS[512 221 10], MNIST, U_c = 97,32 %, Nejlepší konfigurace GSC, počet epoch 228.

8.2.5 Shrnutí

- Nejlepších výsledků bylo dosaženo kombinací sigmoidní přenosové funkce ve skrytých vrstvách s funkcí hyperbolické tangenty ve výstupní vrstvě.
- Vhodná topologie sítě a jejich parametrů nezávisí pouze na příznakových vektorech, ale také na velikosti trénovací a validační množiny.
- Pro redukci lokálních minim je dobré nejprve data randomizovat, nepředkládat je neuronové síti v blocích.
- Pro zastavení učení NS v této úloze je krajně vhodné vyhradit si crossvalidační sadu a podle průběhu její chybové křivky zastavit. Většina testů končila právě touto podmínkou.
- Jas jednotlivých pixelů, jakožto i použití PCA, slouží dobře jako odrazový můstek k seznámení s neuronovými sítěmi, ale pro praktické použití jsou tyto obrazové příznaky zcela nevhodné.
- Rozhodnutí, která z metod extrakce příznaků (Freemanův kód – GSC) je lepší, je obtížné. Obě vykazují podobné výsledky. Na straně Freemanova kódu je malý počet neuronů a velmi rychlé vybavení příslušného znaku, avšak v adaptivním režimu potřebuje větší počet iterací.

Při použití GSC je neuronová síť relativně rychle natrénovaná, ale při vybavování je pomalejší. U této metody je vhodné zařadit do testů i sítě s dvěmi skrytými vrstvami. U Freemanova kódu to nemá valného významu neboť narazíme na lokální minima.

- Lepších výsledků bylo dosaženo s NS s jednou skrytou vrstvou.
- Nejhůře rozpoznatelným znakem je písmeno **B**. Časté záměny jsou z trojice znaků $\{O, B, D\}$, $\{U, V, Y\}$ a $\{H, M, N\}$. Je to způsobeno samotnou databází znaků, neboť u většiny pisatelů je těžké najít rozdíl např. mezi písmeny U a V. Proto při sestavování trénovací a validační množiny je nutné rozhodnout, co se ještě bude brát jako znak U a co jako písmeno V, abychom se vyhnuli náhodnému výběru.

Kapitola 9

Závěr

Rozpoznávání rukou psaných znaků závisí na mnoha faktorech – od kvality pořízeného snímku, vhodné segmentace až po výběr vhodných charakteristik znaků, volbě samotného klasifikátoru a v neposlední řadě na velikosti a různorodosti trénovací a validační množiny. Každá tato část se podílí velkou měrou na samotném výsledku rozpoznávání.

V této práci byly popsány jednotlivé kroky, vedoucí ke správné klasifikaci. Stěžejní částí projektu jsou neuronové sítě, jež tvoří základní stavební kámen projektu a od kterých se odvíjela volba obrazových charakteristik znaku. Těmto charakteristikám byla věnována 3. kapitola, kde byly detailně popsány způsoby jejich extrakce.

Neuronové sítě byly diskutovány v kapitole 5. Bylo zde popsáno pozadí algoritmu zpětného šíření chyby – backpropagation. Tado metoda je svým způsobem učení přímo předurčena pro klasifikaci vzorů. Avšak při adaptivním režimu sebou nese také různé překážky, jejichž způsob řešení byl také diskutován.

Součástí práce byla vytvořena aplikace pro převod textu v obrázku pomocí neuronových sítí. Návrh a pozadí funkčnosti bylo popsáno v kapitole 6 a 7.

Poslední kapitola popisuje samotnou adaptaci neuronových sítí nad databází písmen a některé realizované experimenty, které dávají přehled o úspěšnosti klasifikace.

Nejlepších výsledků bylo dosaženo s dvouvrstvou neuronovou sítí s 50 neurony ve skryté vrstvě a s gradientními, strukturálními a konkávními příznaky. Celková úspěšnost klasifikátoru je 94 %, přičemž nejhůře klasifikovaný znak má chybovost 17 %.

Rozpoznávání rukou psaných písmen je mnohdy obtížná věc i pro člověka, neboť se některá písmena od některých pisatelů velmi málo liší. Jmenovitě např. písmeno O a písmeno D, či písmeno U s písmenem V. Mnoho lidí snad mezi nimi nedělá rozdíl a bez kontextu celého slova či věty je správné rozhodnutí otázkou náhody. Proto by při dalším pokračování bylo vhodné zařadit jazykový model, který by udržoval tento kontext, na jehož základě by se ulehčilo rozhodování neuronové sítě.

Literatura

[1] Bräunl, T.: *Parallel image processing.* Springer, 2001, ISBN 3-540-67400-4.

[2] Duda, R. O.; Hart, P. E.: Use of the hough transformation to detect lines and curves in pictures. 1971.

[3] Favata, J. T.; Srikantan, G.: A multiple feature/resolution approach to handprinted digit and character recognition. *International Journal of Imaging Systems and Technology*, ročník 7, č. 4, 1996 [cit. 30.4.2012].

[4] Fawcett, T.: An introduction to ROC analysis. `http://www.csee.usf.edu/ candamo/site/papers/ROCintro.pdf`, [online] [cit. 29.12.2010].

[5] Hejmans, H. J.; Roerdink, J. B.: *Mathematical morphology and its applications to image and signal Processing.* Kluwer Academic Publishers, 1998, ISBN 0-7923-5133-9.

[6] Hlaváč, V.; Sedláček, M.: Zpracování signálu a obrazu. 1999, 77 s.

[7] Horký, V.: *Rozpoznávání ručně psaných číslic metodou K-nearest neighbor.* Diplomová práce, FIT,Vysoké učení technické v Brně, 2009.

[8] Hu, M. K.: Visual Pattern Recognition by Moment Invariants. *Information Theory,*, ročník 8, 1962: s. 179–187, ISSN 0018-9448.

[9] Kratochvíl, P.; Geiger, J.: Papírové dokumenty v počítači [online]. `http://earchiv.chip.cz/cs/earchiv/vydani/r-2009/chip-09-2009/papirove-dokumenty-pci.html`, 9.2009 [cit. 28.12.2010].

[10] Kurková, V.: Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, ročník 5, č. 3, 1992: s. 501–506.

[11] Leikep, B.: Houghova transformace pro detekci čar. 2009, 21 s.

[12] Mehrotra, K.; Mohan, C. K.; Ranka, S.: *Artificial Neural Networks.* Messachusetts Institute of Technology, 1997, ISBN 0-262-13328-8.

[13] Meloun, M.; Militný, J.; Hill, M.: *Počítačová analýza vícerozměrných dat v příkladech.* Academia, 2005, ISBN 80-200-1335-0.

[14] Norvig, P.: How to Write a Spelling Corrector. `http://norvig.com/spell-correct.html`, [online] [cit. 3.12.2010].

[15] Novák, M.: *Neuronové sítě a informační systémy živých organismů.* Grada, 1993, ISBN 80-85424-95-9.

[16] Saadi, H. E.: A C# Project in Optical Character Recognition (OCR) Using Chain Code. http://www.codeproject.com/Articles/160868/A-C-Project-in-Optical-Character-Recognition-OCR-U, [online] [cit. 30.4.2012].

[17] Smith, L. I.: A tutorial on Principal Components Analysis. http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf, [online] 2002 [cit. 3.12.2010].

[18] Soille, P.: *Mathematical image analysis*. Springer, 1999, ISBN 3540656715.

[19] Volná, E.: Od biologického neuronu k Hebbovu učení. *Automatizace*, ročník 51, č. 5, 2008 [cit. 30.12.2010].

[20] Wikipedie: Canny edge detector. http://en.wikipedia.org/wiki/Canny_edge_detector, [online], [rev. 15.4.2012], [cit. 16.4.2012].

[21] Wikipedie: Image moment. http://en.wikipedia.org/wiki/Image_moment, [online], [rev. 28.12.2010], [cit. 30.12.2010].

[22] WWW stránky: Analýza hlavních komponent v programu SAS. http://info.lu2.name/soubory/Principal_components_analysis_481.pdf, [online] [cit. 3.12.2010].

[23] WWW stránky: The Back Propagation Algorithm. www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf, [online] [cit. 3.1.2011].

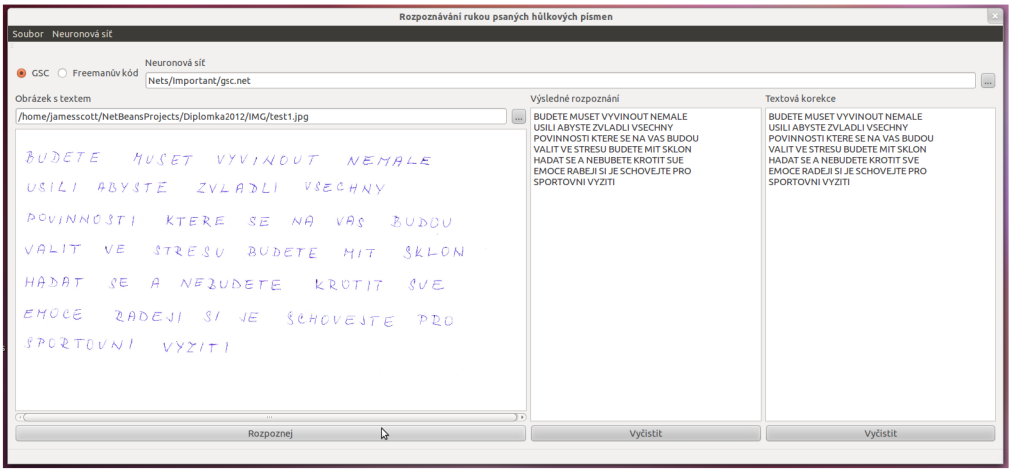
[24] Šonka, M.; Hlaváč, V.: *Počítačové vidění*. Grada, 1992, ISBN 8085424673.

[25] Španěl, M.; Beran, V.: Obrazové segmentační techniky. http://www.fit.vutbr.cz/ spanel/segmentace/#_Toc125769332, [online] [cit. 29.12.2011].

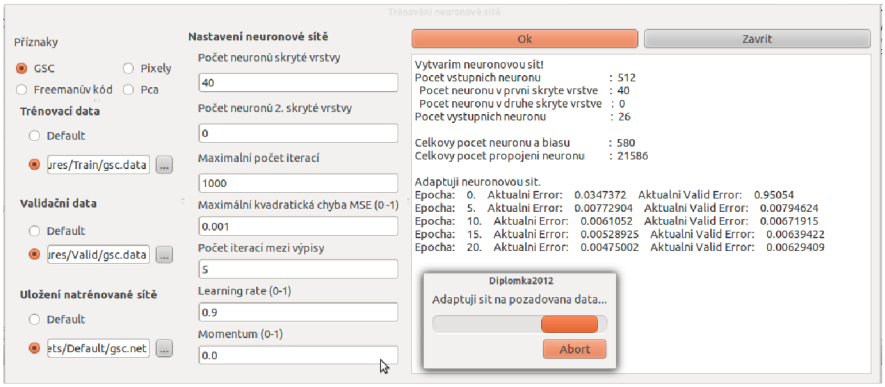
[26] Šíma, J.; Neruda, R.: *Teoretické otázky neuronových sítí*. Matfyzpress, 1996, ISBN 80-85863-18-9.

Příloha A

Ukázka aplikace



Obrázek A.1: Hlavní okno



Obrázek A.2: Trénování neuronové sítě

Příloha B

Obsah CD

Na přiloženém CD je uložena elektronická podoba tohoto dokumentu. Kromě zdrojových souborů aplikace je zde také přiložena databáze rukou psaných znaků o velikosti 20×20 pixelů roztříděné do odpovídajících složek. Součástí je i instalační a ovládací manuál.