Metódy rozpoznávania objektov v obraze

Bc. Adrián Španko

Diplomová práca
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Adrián Španko**
Osobní číslo:	**A15201**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Bezpečnostní technologie, systémy a management**
Forma studia:	**prezenční**
Téma práce:	**Metody rozpoznávání objektů v obraze**
Téma anglicky:	**Object Recognition in an Image Methods**

Zásady pro vypracování:

1. **Vypracujte literární rešerši zaměřenou na metody rozpoznávání objektů v obraze.**
2. **Popište aktuální trendy uplatňované při metodách rozpoznávání objektů v obraze.**
3. **Proveďte analýzu softwaru aplikovaného pro metody rozpoznávání objektů v obraze.**
4. **Navrhněte úlohu pro rozpoznání objektů v zadaném obraze.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **KOTEK, Z., Mařík, V. a kol.: Metody rozpoznávání a jejich aplikace. Academia, Praha, 1993.**
2. **KARBAN, Pavel. Výpočty a simulace v programech Matlab a Simulink. 1. vyd.. Brno: Computer Press, 2006. 220 s. ISBN 80-251-1301-9**
3. **ZAPLATÍLEK, Karel a Bohuslav DOŇAR. MATLAB: tvorba uživatelských aplikací. 1. vyd.. Praha: BEN, 2004. 215 s. ISBN 80-730-0133-0.**
4. **HUMUSOFT [online].Matlab: Jazyk pro technické výpočty. c 1991-2012 [cit. 2017-01-27]. Dostupné z: http://www.humusoft.cz/produkty/matlab/matlab/.**
5. **ICT kompetence Počítačová grafika: Barevný model, DPI. [online]. 2009 [cit. 2017-01-27]. Dostupné z: http://www.kteiv.upol.cz/frvs/ictkubricky/?page=pocitacova-gra_ka/barevny-model-dpi.**
6. **ŘÍHA, Kamil. Pokročilé techniky zpracování obrazu. Brno: FEKT VUT v Brně, 2007. 109 s.**
7. **SONKA, Milan, et al. Image processing, analysis, and machine vision. Toronto: Thomson, 2008.**

Vedoucí diplomové práce: **doc. Mgr. Milan Adámek, Ph.D.**
Ústav bezpečnostního inženýrství

Datum zadání diplomové práce: **3. února 2017**

Termín odevzdání diplomové práce: **24. května 2017**

Ve Zlíně dne 3. února 2017

L.S.

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

doc. Mgr. Milan Adámek, Ph.D.
děkan

doc. RNDr. Vojtěch Křesálek, CSc.
ředitel ústavu

Jméno, příjmení: Adrián Španko, Bc.

Název diplomové práce: Metódy rozpoznávania objektov v obraze

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomovépráci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledkůbudu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne ……

podpis diplomanta

ABSTRAKT

Detekce různých objektů či tvarů ve snímku je velmi důležitým prvkem veškerých operací zahrnujících práci s obrazem. Uplatňuje se nejen ve zpracování lékařských obrazových dat (hledání struktur v tomografech), ale i v bezpečnosti (např. kontrola zda vše je na svém místě), a v osobním sektoru u fotoaparátů a kamer (detekce obličeje, úsměvu…). Samozřejmě existuje ještě více využití.

Klíčová slova: Detekcia, objekt, bezpečnosť, kamera

ABSTRACT

Detection of different objects and shapes in an image is a very important element of any operations which involve image processing. It is applied not only in the processing of medical image data (searching for structures in tomographs), but also in security (e.g. checking if everything is in its place) and in the personal sector where it is connected with cameras and video cameras (detection of faces, smiles...). There are, of course, even more sorts of usage.

Keywords: Detection, objects, security, video camera

Toto cestou sa chcem poďakovať všetkým, ktorí mi pomohli pri písaní bakalárskej práca. Veľká vďaka patrí vedúcemu bakalárskej práca doc. Mgr. Milanovi Adámkovi, Ph.D. za cenné pripomienky, všestrannú a odbornú pomoc pri konzultáciách a samotnom písaní diplomovej práce.

OBSAH

ÚVOD

V súčasnej dobe sa čoraz viac získané informácie spracovávajú do digitálnej podoby. K prvotným a dôležitým vnemom u človeka patrí zachytávanie objektov okom a následné spracovanie mozgovým vnemom.

Podobne aj počítačové videnie je proces spracovania obrazu snímaného objektívom alebo iným obrazovým záznamom. Umožňuje porozumieť mu a interpretovať ho. Počítač nahradí prácu mozgu, uskutoční spracovanie a vyhodnotenie informácie. Dokáže rozpoznať, čo sa pred objektívom deje a vykoná ďalšie potrebné úkony.

Systémy počítačového videnia sa využívajú v mnohých oblastiach života. Možno ich využívať pri detekcii javov zachytenými prostredníctvom bezpečnostných kamier, k vzájomnej interakcii počítača s človekom, v oblasti obrazovej kontroly kvality výrobkov. Je možné ich využiť aj ako súčasť riadiacich systémov robotov alebo samostatne riadiacich automobilov. V bežnom živote sa najčastejšie stretávame s detekciou prvkov v obraze v kamere či fotoaparáte ako je detekcia tváre alebo úsmevu .

Cieľom tejto práce bude navrhnúť vhodnú metodiku na rozpoznanie objektov v obraze prostredníctvom programu Matlab. Najskôr je potrebné spracovať teóriu spracovania obrazu a neurónových sietí ako aj oboznámiť sa so samotným programom Matlab. Následne praktická časť už bude zameraná na samotnú tvorbu programu na rozpoznávanie objektov.

I. TEORETICKÁ ČASŤ

1 SPRACOVANIE OBRAZU

Spracovanie obrazu je možné všeobecne chápať ako prevod nasnímaného analógového obrazu do digitálnej podoby a jeho následné spracovanie. Proces nasnímania a rozpoznania obrazu je možné rozdeliť do niekoľkých krokov. Každá literatúra definuje celý postup inak a z toho dôvodu ho nemožno presne definovať. Je nutné aby boli všetky kroky uskutočnené správne a to v akom poradí, záleží na použitej aplikácii. [1]

Postupnosť základných krokov:

- snímanie a digitalizácia,
- predspracovanie,
- segmentácia,
- popis objektov,
- klasifikácia.

1.1 Snímanie a digitalizácia

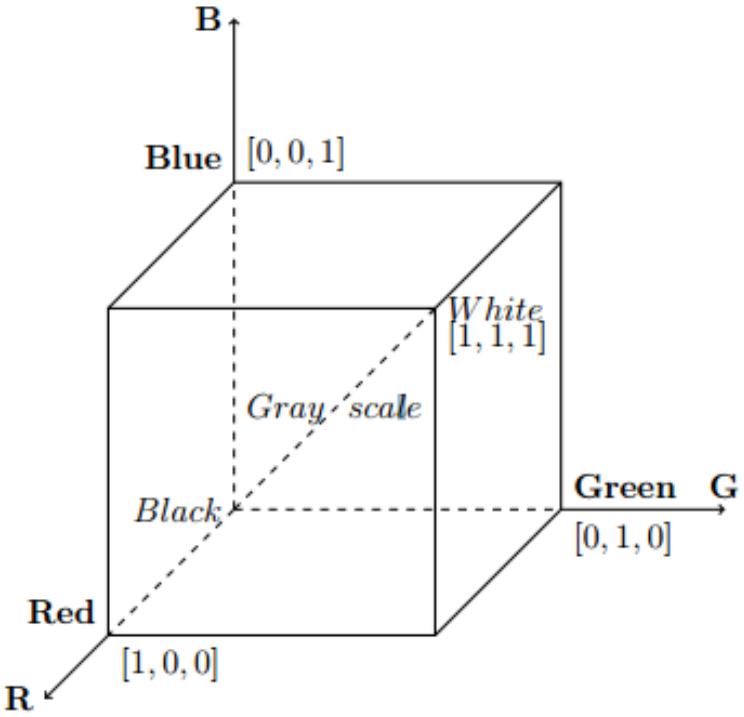
Získaný obraz prostredníctvom kamery, skeneru, lekárskych alebo iných zariadení sa ukladá v číselnej podobe do počítača. Vstupné informácie môžu byť jas alebo niekoľko spektrálnych zložiek ak sa jedná o farebné snímanie obrazu. Tieto zložky rozlišujeme podľa použitého farebného modelu ako napríklad RGB, HSV, CMYK a pod.

Digitalizácia je pojem, ktorý sa používa pri prevode analógového signálu do diskrétneho tvaru. Digitálny obraz je ekvivalent spojitej funkcie *f(x,y)*. V tejto funkcii sa používajú hodnoty *x* a *y* ako súradnice v nasnímanom obraze a funkčná hodnota zodpovedá napríklad jasu. Získava sa vzorkovaním obrazu do matice *MxN* bodov a kvantovaním do *K* úrovní. Prvky v takejto matici sú nazývané pixely. [1]

1.1.1 RGB

Farebný model RGB je momentálne najpoužívanejším modelom v digitálnych obrazoch. Využíva sa v obrazovkách, projektoroch, digitálnych fotoaparátoch apod. Ako už vyplýva z názvu, RGB model je zložený z troch základných farieb a to: červená (**R**ed), zelená (**G**reen) a modrá (**B**lue). Tieto farby sa ďalej kombinujú a tým sa vytvárajú odtiene rôznych farieb. Model RGB je založený na aditívnom miešaní farieb čo znamená, že čím sa zmieša viac farieb tým bude výsledná farba svetlejšia. [2]

Model RGB je možné zobraziť ako jednotkovú kocku na osiach, ktoré sú označené R, G a B. Počiatočný bod, teda [0,0,0], charakterizuje čiernu farbu a bod na opačnej strane súradnicovej sústavy, teda [1,1,1], charakterizuje bielu farbu. Jednotlivé farby sa nachádzajú na osiach v bodoch 1 a teda červená na osi R v bode [1,0,0], zelená na osi G v bode [0,1,0] a modrá na osi B v bode [0,0,1]. Ostatné vrcholy kocky tvoria tzv. doplnkové farby, ktoré sa označujú ako sekundárne. Popis tejto charakteristiky je možné vidieť na obrázku č.1. [2,17]



Obrázok č.1: RGB model [2]

1.1.2 CMYK

V tomto farebnom modeli je miešanie farieb riešené subtraktívne, teda čím zmiešame viacej farieb tým bude výsledná farba tmavšia. Model CMY je založený z troch základných farieb a to: modrozelená (**C**yan), fialová (**M**agenta) a žltá (**Y**ellow). Model CMYK bol navrhnutý a využíva sa výhradne v tlačiarenských procesoch. Tlačiarenská farba nesmie dokonale kryť a to z dôvodu, aby bolo možné dané farby kombinovať. Zmiešaním všetkých farieb sa nedostane čierna ale takzvaná špinavo hnedá a ak by sa táto farba využívala ako čierna viedlo by to k nadmernému spotrebovaniu tonerov. Pre tento prípad bola pridaná čierna farba a tak vznikol model CMYK, kde K reprezentuje čiernu farbu (blac**K**). [2]

Tento proces sa dá taktiež popísať jednotkovou kockou ako pri modeli RGB, len s rozdielom, že vrcholy čiernej a bielej sú prehodené. Prevod medzi modelmi RGB a CMY je vyobrazený na obrázku č.2. [2]

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

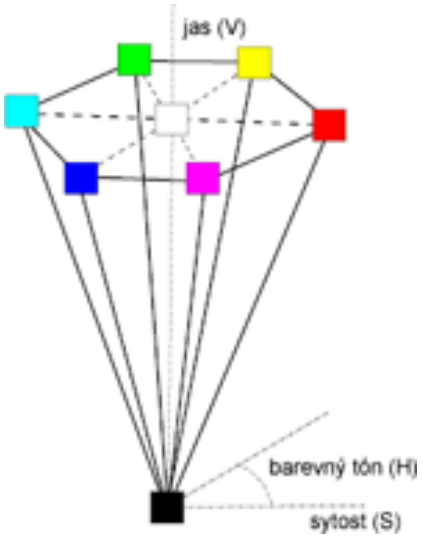
Obrázok č.2: Model CMY [2]

Prevod medzi týmito modelmi nie je dokonalý nakoľko sa občas nedá zobraziť konkrétna farba v oboch modeloch. Niektoré farby v modeli RGB sa nedajú napodobniť v modeli CMYK a tak sa nahrádzajú najbližšou možnou alebo najpodobnejšou farbou. To má za následok mierne zmeny farieb v tlačenom obraze oproti tomu, ktorý vidíme na monitoroch. [2]

1.1.3 HSV

Farebný model HSV, ktorý niekedy býva označený aj HSB, sa skladá z troch základných zložiek. Základný rozdiel medzi modelom HSV a modelmi RGB a CMYK je v tom, že informácia o farbe je v prvej zložke, nazývanej odtieň (**H**ue). Tu sa konkrétne odtiene farieb vyjadrujú uhlom v rozmedzí 0° až 360°. Následne konkrétny uhol reprezentuje prevládajúcu farbu. Druhou zložkou je sýtosť (**S**aturation) a tá sa mení v rozmedzí 0 až 1 čo udáva čistotu danej farby. Ako posledná zložka je hodnota jasu (**V**alue), ktorá je taktiež v rozmedzí 0 až 1 a udáva množstvo bieleho svetla. Najväčším rozdielom od predošlých modelov je možnosť oddelenia jasu a sýtosti čo umožňuje pracovať s nimi bez ohľadu na farebný odtieň. [2]

Pri práci s Matlabom sa využíva na prevod medzi modelmi RGB a HSV príkaz rgb2hsv alebo rgb2hsb. Pracuje sa zo samostatnou maticou jasu, čo nám následne neovplyvňuje zostávajúce matice farebného odtieňa a sýtosti farby. Jediným problémom pri práci s modelom HSV je prechod medzi čiernou a bielou farbou, ktorý nie je plynulý. Ten sa uskutočňuje na šesťuholníku a nie po kružnici. [2,17]



Obrázok č.3: Model HSV ako šesťboký ihlan [2]

1.2 Predspracovanie obrazu

Predspracovanie obrazu je potrebné na vylepšenie obrazu aby bolo možné jeho ďalšie spracovanie. V podstate ide o potlačenie šumu, ktorý vzniká pri digitalizácii obrazu alebo o zvýraznenie iných parametrov obrazu. [3,4]

Základné rozdelenie predspracovania obrazu:

- geometrická transformácia,
- transformácia jasu,
- filtrácia a zaostrenie.[1]

1.2.1 Geometrická transformácia

Geometrická transformácia objektov, či už sa jedná o dvojrozmernú alebo trojrozmernú, je najdôležitejšou operáciou, ktorá sa využíva v digitálnom zobrazení obrazu. Medzi geometrické transformácie sa zaraďuje zmena mierky, škálovanie, rotáciu, skosenie a rôzne kombinácie týchto metód, ktoré sa nazývajú zložené transformácie. Táto transformácia sa uskutočňuje v priestore homogénnych súradníc. V nasledujúcom vyjadrení je možné vidieť prevod z kartézskych do homogénnych súradníc. [1]

$$P' = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \mathbf{A}_{3x3}P = \mathbf{A}\begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Obrázok č.4: Prevod medzi Kartézskymi a homogénnymi súradnicami [5]

1.2.1.1 Zmena mierky

Zmenou mierky sa ovplyvňuje veľkosť a poloha transformovaného objektu v závislosti k počiatku súradníc. Spomínaná operácia zapríčiňuje zmenšenie alebo zväčšenie objektu, podľa toho aké hodnoty získavajú koeficienty mierky. Ak sa koeficienty nachádzajú v intervale (0,1), dochádza k zmenšeniu objektu a zároveň posunu k počiatku súradnicového systému. Ak sa koeficienty nachádzajú v intervale (1,∞) dochádza k zväčšeniu objektu a zároveň posunu od počiatku súradnicového systému. Ak niektorá z hodnôt v intervale bude záporná, dôjde k rovnakým procesom, ale v opačnom smere. Nasledujúca matica zobrazuje transformáciu 2D objektu, kde S_x je koeficient zmeny mierky na osi X a S_y je koeficient zmeny mierky na osi Y. [5]

$$\mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Obrázok č.5:Transformačná matica 2D objektu [5]

1.2.1.2 Rotácia

Rotáciou objektu sa myslí posunutie objektu o určitý uhol, ktorý zviera s počiatkom súradnicovej sústavy. Objekt môže byť natočený ľubovoľným spôsobom, no vychádza sa z predpokladu, že zviera s počiatkom súradnicovej sústavy nulový uhol. Pre rotáciu objektu sa zvolí uhol α a podľa znamienka určíme smer rotácie, buď kladný (v smere hodinových ručičiek) alebo záporný (proti smeru hodinových ručičiek). Následne sa zvolené hodnoty dosadia do transformačnej matice. [5]

$$\mathbf{R}(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Obrázok č.6: Transormačná matica rotácie objektu [5]

1.2.1.3 Skosenie

Skosenie sa uskutočňuje v závislosti na osi x alebo y. V závislosti orientácie podľa osi x alebo y sa dosadzujú hodnoty za koeficient do niektorej z matíc. Inverziou matíc sa získava skosenie v zápornom smere. [5]

$$\mathbf{Sh}_x(sh_x) = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{Sh}_y(sh_y) = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Obrázok č.7: Transformačné matice skosenia [5]

1.2.1.4 Zložené transformácie

Zložením viacerých transformácií sa získava matica, ktorá popisuje všetky využité operácie. Keďže ide o násobenie matíc, záleží na poradí jednotlivých operácií. Teda je rozdiel ak najskôr uskutočníme posunutie a následne rotáciu okolo počiatočného bodu súradnicového systému alebo daný proces zameníme. Ak by sa vypočítali nové koeficienty v závislosti na prvej operácii, je možné sa dostať k rovnakému výsledku. Nasledujúca matica sa skladá zo skosenia a zmeny mierky, kde s_x a s_y sú koeficienty zmeny mierky a sh_x a sh_y sú koeficienty skosenia v smere danej osi. [5]

$$\mathbf{V}(sh_x, sh_y, s_x, s_y) = \mathbf{Sh}_x(sh_x)\ \mathbf{Sh}_y(sh_y)\ \mathbf{S}(s_x, s_y) =$$
$$\begin{bmatrix} s_x + s_x sh_y sh_x & s_y sh_x & 0 \\ s_x sh_y & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

Obrázok č.8: Matica zloženej transformácie [5]

1.2.2 Transformácia jasu

Transformácia jasu je úpravou stupnice jasu nasnímaného obrazu, ktorá má za cieľ zvýrazniť určité rysy obrazu. Niektoré transformácie jasu majú za následok zvýraznenie nežiaduceho šumu. Základné transformácie sú bodové, lokálne a globálne, záleží čím sa riadi daná operácia. Bodové transformácie jasu sa riadia jasom konkrétneho pixelu, ktorý podľa stanoveného vzorca následne zvýrazní vybraný pixel. Lokálne transformácie využívajú malé okolie vybraného bodu a podľa ich hodnôt sa mení jas v danom pixely. Globálne transformácie využívajú hodnoty jasu celého obrazu a následne ich aplikujú na vybraný pixel v obraze. [6]

1.2.2.1 Negatív

Prevod obrazu na negatív je považovaný za najjednoduchšiu transformáciu jasu keďže zdrojový obraz nie je potrebné analyzovať. Pri prevode sú hodnoty intenzity jasu *f* nahradené inou hodnotou intenzity podľa už vopred stanovenej inverznej operácie pre všetky x,y v obraze. [6]

$$g(x,y) = 1 - f(x,y)$$

Obrázok č.9: Rovnica inverzného výpočtu negatívu [6]

Šedotónový obraz, ktorý sa využíva pri predspracovaní obrazu je definovaný intervalom (0, L-1) diskrétnych úrovní jasu a následne sa zmení aj transformačná rovnica. Táto rovnica sa využíva len v prípade, ak je nutné invertovať binárny teda prahový obraz za účelom detekcie regiónov pixlov.[6]

$$g(x,y) = L - 1 - f(x,y)$$

Obrázok č.10: Rovnica šedotónového obrazu [6]

1.2.2.2 Logaritmická transformácia a gama korekcia

Jednou z ďalších transformácií, ktoré nezahŕňajú okolie pixlov sú logaritmické transformácie a gama korekcie. Ich cieľom je kompresia hodnôt jasu a tá je daná vzťahom:

$$g = c\log(1+f)$$

V tomto prípade musí byť f ≥ 0, čo reprezentuje hodnotu jasu s nevyhovujúcim dynamickým rozsahom a konštanta *c* sa využíva k normovaniu výslednej hodnoty do inter-

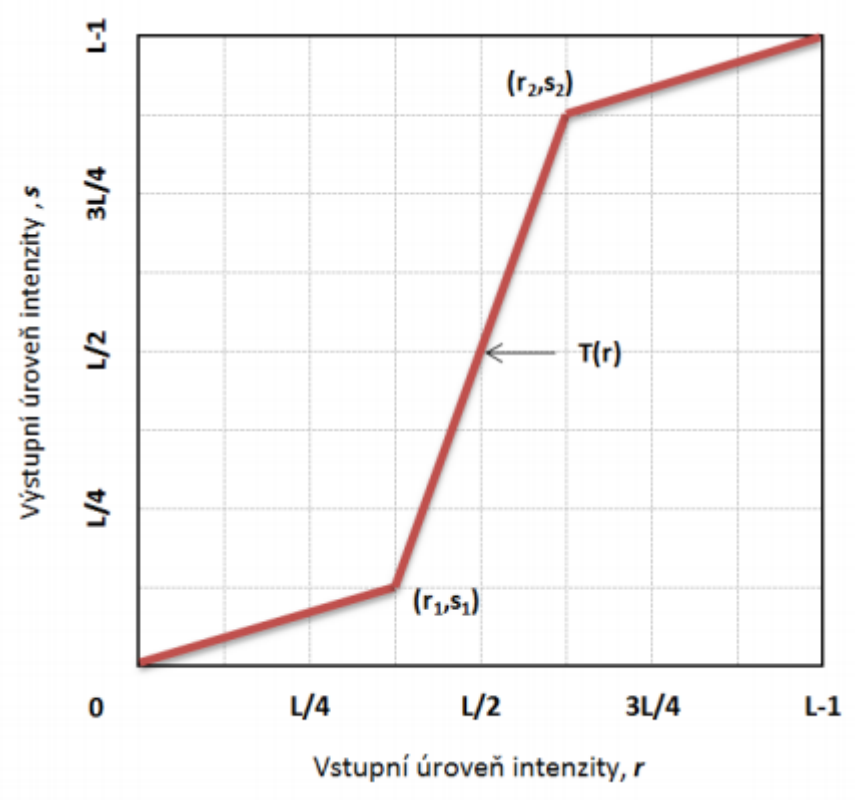
valu g ϵ (0,1). Z dôvodu nelineárnej závislosti starých CRT obrazoviek vznikla gama korekcia, ktorá je definovaná vzťahom:

$$g = \mathrm{c}\, f^{\Upsilon}$$

Predpokladá sa, že vstupná hodnota f ϵ (0,1) a hodnota ϒ je daná typom použitej obrazovky. [6]

1.2.2.3 Roztiahnutie kontrastu

Roztiahnutie kontrastu sa uskutočňuje po častiach a je to jedna z najjednoduchších lineárnych transformácií. Ak je rozdiel v úrovniach jasu veľmi malý a obraz je nekonštantný, vtedy sa využíva tento typ transformácie.Spomínané nedostatky v obraze môžu byť spôsobené slabým osvetlením, nízkym dynamickým rozsahom snímacieho senzoru alebo nesprávnym nastavením záznamového zariadenia. Roztiahnutie kontrastu slúži na zvýšenie dynamického rozsahu úrovní jasu v spracovanom obraze. Na nasledujúcom grafe je možné vidieť typickú transformáciu používanú pri rozťahovaní kontrastu. Body (r_1,s_1) a (r_2,s_2) určujú tvar transformačnej funkcie. [6]



Obrázok č.11: Rozťahovanie kontrastu [6]

Ak nastane prípad, keď $r_1 = s_1$ a $r_2 = s_2$, vtedy je transformácia lineárna a v takomto prípade nedochádza k žiadnej zmene vo výslednom obraze. Ak $r_1 = r_2$, $s_1 = 0$ a $s_2 = L-1$ z transformácie sa stáva prahová funkcia a to má za následok vytvorenie binárneho, teda čiernobieleho obrazu. Za predpokladu, že r_1 je minimálna hodnota jasu a r_2 je maximálna hodnota jasu v obraze, hodnoty z intervalu (r_1, r_2) sa normalizujú nasledovne: $r \epsilon (0, 1)$. Za týchto okolností sa transformácia uskutočňuje tak, že od hodnoty jasu r sa odráta minimálna hodnota jasu a vynásobí sa koeficientom roztiahnutia $1/(r_2 - r_1)$. Následne vzniká vzťah: [7]

$$s = \frac{r - r1}{r2 - r1}$$

1.2.3 Filtrácia a zaostrenie

Táto metóda má za úlohu odstránenie šumu z digitalizovaného obrazu a zviditeľnenie horšie rozpoznateľných častí. V danej metóde sa rozpoznávajú dve základné filtrácie. Prvou je filtrácia v priestorovej oblasti, ktorá má za úlohu spracovať obraz ako lineárnu kombináciu vstupného obrazu s koeficientmi filtrov. Základným matematickým nástrojom tohto druhu filtrácie je konvolúcia. Druhým tipom je filtrácia vo frekvenčnej oblasti. Tá najskôr uskutoční prevod obrazu do frekvenčnej reprezentácie, ktorá sa filtruje a následne sa prevedie späť na obraz. Prevod sa uskutočňuje niektorou z metód ako Fourierová, vlnová alebo diskrétna kosínusová transformácia. Filtráciu zaisťujú filtre typu dolná, horná a pásmová priepusť. [8]

1.3 Segmentácia obrazu

Segmentácia obrazu je súbor procesov, v ktorom dochádza k analýze obsahu spracovaných dát. Za najdôležitejší faktor pre správnu segmentáciu sa považuje charakter vstupného obrazu. [9]

Úlohou segmentácie je rozdelenie obrazu na časti, ktoré aspoň určitou mierou súvisia s realitou. Pre všetky oblasti v danej množine je potrebná ich vzájomná nezlučiteľnosť. Medzi nájdenými segmentmi a skutočnými objektmi sa hľadá miera zhody. Podľa miery zhody medzi nimi sa rozdeľujú na kompletnú a čiastočnú segmentáciu. Čiastočná segmentácia má menšiu mieru zhody ako kompletná segmentácia. [4]

Ak sa využíva kompletná segmentácia, je potrebné použiť vyššiu úroveň spracovania a aj dostatočné znalosti riešenia daného problému. Taktiež je možné naraziť na problémy, ktoré si nevyžadujú spomínanú vyššiu úroveň spracovania. Následne je možné použiť nižšiu úroveň spracovania. Tento prípad nastáva vtedy, ak sú skúmané objekty s pozadím, ktoré má konštantný jas. [10]

Pri čiastočnej segmentácii sa získava obraz, ktorý je rozdelený do niekoľkých samostatných častí. Najväčšou výhodou tohto typu segmentácie je schopnosť spracovať aj zložité scény a zníženie objemu spracovaných dát. Výsledkom je konečný súbor homogénnych oblastí, ktoré majú určité vlastnosti farby, jasu apod. Takto nadobudnuté oblasti majú možnosť vzájomného prekrytia. Jedinou nevýhodou čiastočnej segmentácie je potreba využitia ďalších postupov, aby bolo možné získať relevantné výsledky. [1]

Segmentáciu obrazu je možné rozdeliť podľa použitých vlastností na tri základné metódy. Prvá metóda je založená na globálnej znalosti obrazu alebo aspoň jeho častí. Táto metóda je reprezentovaná pomocou histogramu. Druhá metóda sú algoritmy, ktoré sú založené na určovaní hraníc medzi oblasťami. Tretia metóda spočíva v algoritmoch, ktoré spomínané oblasti vytvárajú. [9,16]

1.4 Popis objektov

Popis objektov je možný dvomi spôsobmi. Prvý je takzvaný kvantitatívny prístup a to je možné za pomoci súboru číselných charakteristík. Pod pojmom súbor číselných charakteristík sa dá predstaviť veľkosť objektu, farebný rozptyl alebo komparatívnosť. Ďalšou možnosťou popisu objektov je kvalitatívny prístup. Tento spôsob popisu hovorí o súvislosti medzi vybranými objektmi a popisuje ich tvarové vlastnosti. Spôsob akým bude popis uskutočnený závisí podľa ďalšieho využitia. Pri rozpoznávaní objektov sú tieto popisy považované za vstupné informácie. Následný výsledok popisov závisí od postupu, ktorý bol využitý na klasifikáciu objektov. [1,15]

1.5 Klasifikácia

Klasifikácia má za úlohu zaradiť objekt, ktorý sa našiel v obraze do niektorej zo skupín známych tried. Metódy klasifikácie sa rozdeľujú do dvoch skupín a to príznakovú a štrukturálnu klasifikáciu.

Príznaková klasifikácia využíva skupiny charakteristických čísel objektu, ktoré popisujú jeho vlastnosti ako sú poloha alebo veľkosť a pracuje s kvantitatívnym popisom objektov. Ako príznaková klasifikácia sa môže brať aj takzvaná zhluková analýza. Tato analýza triedi objekty do skupín tak, aby boli objekty s určitými vlastnosťami zaradené v jednej skupine.

Štrukturálna klasifikácia využíva priradenie istých vlastností, ktoré objekt charakterizujú a pracuje s kvalitatívnym popisom objektov. Spomínané vlastnosti sú následne podrobené algoritmom rozboru slova, ktoré popisujú objekt a následne sa kontroluje syntax, aby mohol byť definovaný jazyk, gramatika a abeceda.[7,8]

2 NEURÓNOVÉ SIETIE

Neurónové siete sa využívajú na rozpoznávanie a klasifikáciu objektov. Sú inšpirované biologickými neurónovými sieťami. Táto vlastnosť ich predurčuje, aby sa mohli správať do určitej miery rovnako alebo aspoň podobne ako biologické neurónové siete. Z toho vyplýva, že vytvorenie umelého ľudského mozgu so všetkými funkciami je takmer nemožné, či už z pohľadu počtu neurónov, ich spôsobu prepojenia alebo správania jednotlivých typov neurónov. No je tu príležitosť nasimulovať niektoré funkcie ľudského myslenia a následne ich implementovať. [11]

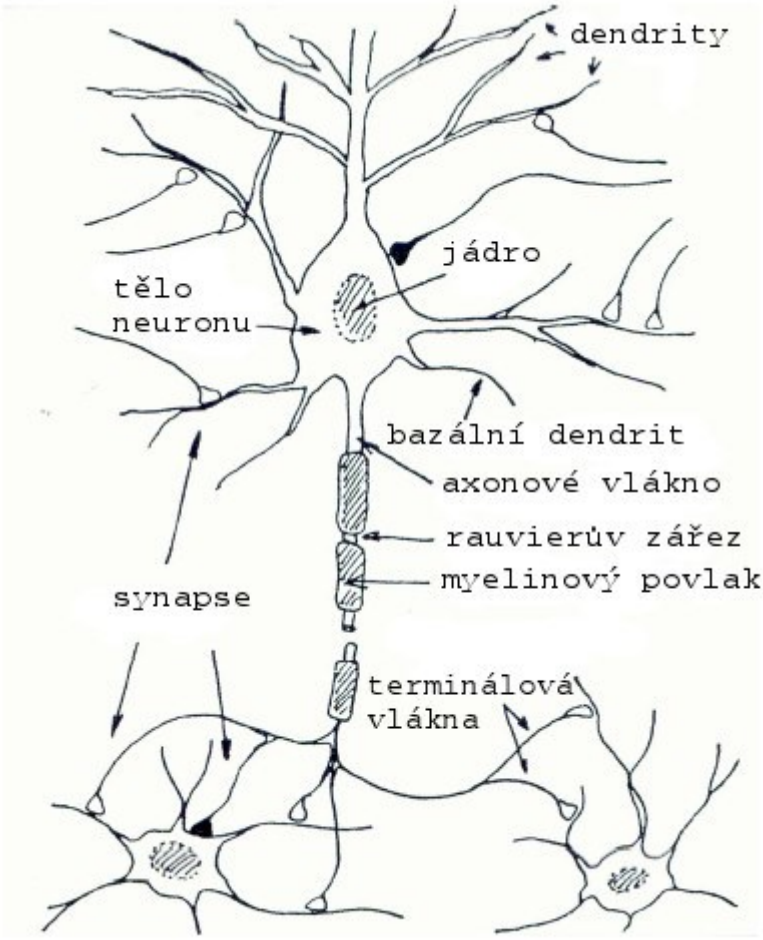
Pri uskutočňovaní výpočtov sa pracuje s distribuovaným, paralelným spracovaním informácií. Ukladanie, spracovanie a následné predávanie informácií neprebieha len za pomoci určitých pamäťových častí, ale prostredníctvom celej neurónovej siete. Spracovanie informácie a pamäť je v podstate globálna a nie lokálna. [11]

Prostredníctvom väzieb medzi jednotlivými neurónmi sa ukladajú nadobudnuté znalosti. Väzby, ktoré vedú k získaniu správnych odpovedí, sú naďalej posilňované a naopak väzby smerujúce k nesprávnym odpovediam sa oslabujú pomocou opakovaných expozícií príkladov, ktoré popisujú problémový priestor. [11]

Najpodstatnejšia a najzákladnejšia vlastnosť neurónových sietí je učenie. Tento fakt poukazuje na rozdiel medzi bežným využitím počítačov a využitím prostriedkov, ktoré pracujú na báze neurónových sietí. Doteraz sa tvorili algoritmy užívateľských programov, ktoré transformovali množinu vstupných dát do množiny výstupných dát, no pri použití neurónových sietí nebude potrebné vykonávať spomínaný proces. To akým spôsobom sa budú transformovať dáta závisí na procese učenia, ktorý je založený na expozícii **vzorcov** popisujúcich riešenú problematiku, čo je takzvaná trénovacia množina. V tomto prípade už nie je potrebná algoritmizácia úloh, pretože je nahradená spomínanými trénovacími množinami neurónovej siete a jej následným učením. [11]

2.1 Model neurónu

Sieť viacerých navzájom prepojených jednoduchých procesorov sa nazýva neurónová sieť. Podľa biologickej predlohy sa popisujú umelo vytvorené neuróny, ktoré tvoria základnú jednotku zložitého komplexu a ten sa nazýva neurónová sieť. Na nasledovnom vyobrazení sa nachádza model neurónovej siete. [11,13]

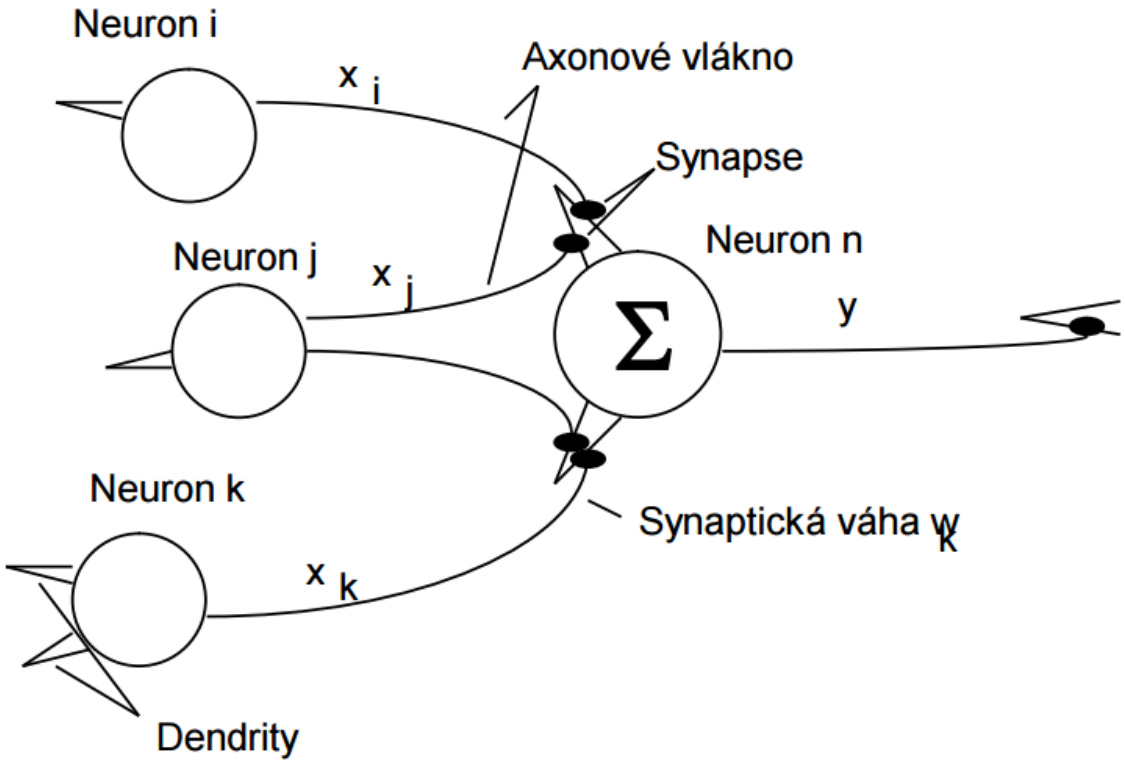


Obrázok č.12: Schéma biologického neurónu [12]

Na obrázku č.12 sú vyobrazené jednotlivé časti neurónu. Teda je možné vidieť že neurón sa skladá z:

- *"Dendrity reprezentujú miesto vstupu signálov do tela neurónu.*
- *Telo bunky sčíta signály dané okolitými neurónmi. Takto stanovený vnútorný potenciál vedie k excitácii (vybudeniu) neurónu.*
- *Axonové vlákno prenáša signál, daný stupňom excitácie k synapsiám.*
- *Synapsie tvoria výstupné zariadenie neurónov, ktoré signál zosilňuje či zoslabuje a odovzdáva ho ďalším neurónom.*
- *Myelinový povlak je to vodivá membrána, ktorá pokrýva celé telo axonového vlákna"* [11].

Vyššie uvedený model neurónu je možné vyjadriť nasledovne:



Obrázok č.13: Schematický model neurónu [11]

Kde $x_{i,j,k}$ reprezentujú vstupné signály neurónov (i,j,k), $w_{i,j,k}$ sú synaptické váhy upravujúce výstupný signál neurónov (i,j,k) a y je výstupný excitačný signál neurónu n. [11]

2.1.1 Formálny neurón

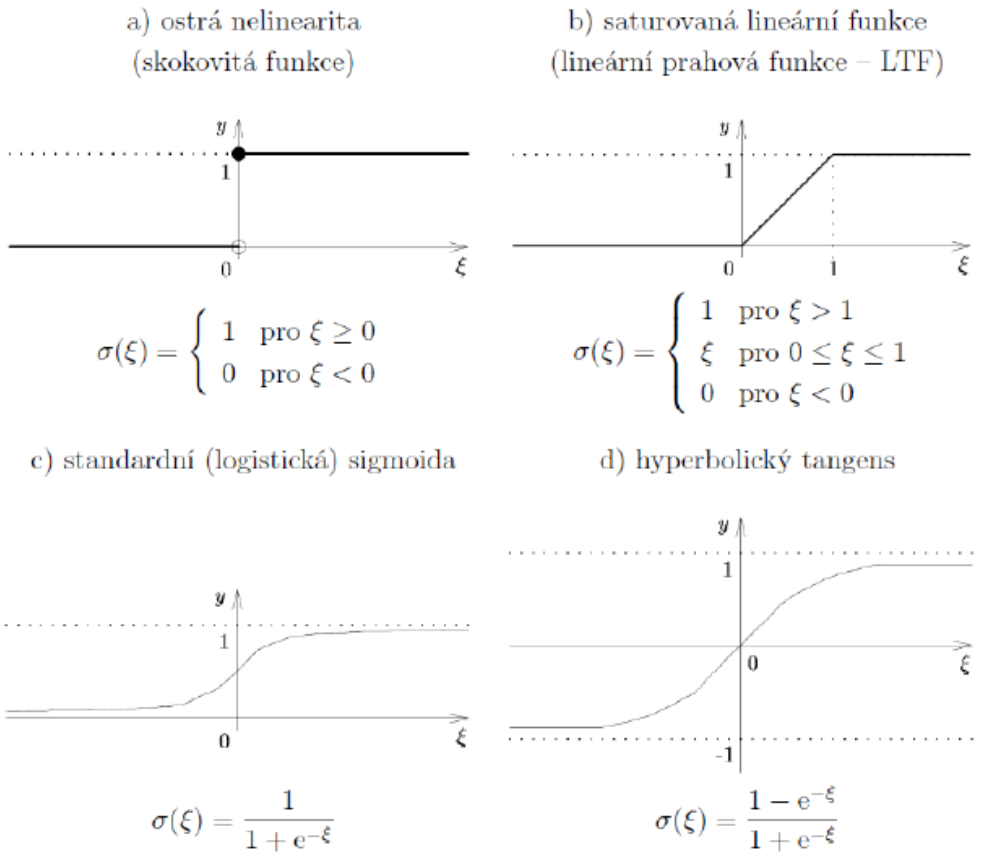
Formálny neurón je považovaný za základnú jednotku matematického vyjadrenia neurónových sietí, ktorí sa získava zo zjednodušeného biologického neurónu. Schematické znázornenie formálneho neurónu je vyobrazená na obrázku č.13. Spomínaný neurón ma *n* reálnych vstupov ($x_{1.....n}$) a to sú v skutočnosti dendrity. Dané vstupy môžu reprezentovať aj výstupy z iných neurónov alebo aj podnety z vonkajšieho okolia. Vstupy sa ohodnocujú reálnymi synaptickými váhami ($w_{1.....n}$) a tie určujú mieru priepustnosti daných vstupov. Teda na úpravu vstupného signálu a určenie dôležitosti daného spoja vedúceho k neurónu sa využívajú práve už spomínané synaptické váhy. [13,14]

Celkový podnet neurónu udáva vážený súčet:

$$\xi = \sum_{i=1}^{n} w * x - Ø$$

Spomínaný podnet sa niekedy označuje aj ako potenciál neurónu a naň neurón reaguje výstupnou odozvou $Z = S(\xi)$, kde S tvorí prenosovú funkciu. Ak prenosová funkcia naberá tvar skokovej funkcie, vtedy sa jedná o potenciál, ktorý pracuje ako binárny klasifikátor. Ďalšou veličinou, čo určuje neurón, je takzvaný prah Ø. Tento prah ma trvalý výstup -1 a považuje sa za špeciálny prípad váhy spoja, ktorý vedie do fiktívneho neurónu. Následne je možné označiť $Ø_0 = w_0$ a $x_0 = 1$ kedy dostaneme vzťah $\xi = \sum_{i=1}^{n} w * x - Ø = w * x$. Ak podnet prekročí hodnotu 0, teda w * x > 0, tak bod x sa nachádza v kladnom nadpriestore, čo vymedzuje separujúca nadrovina w * x = 0, inak leží v doplnkovom nadpriestore. [11,13]

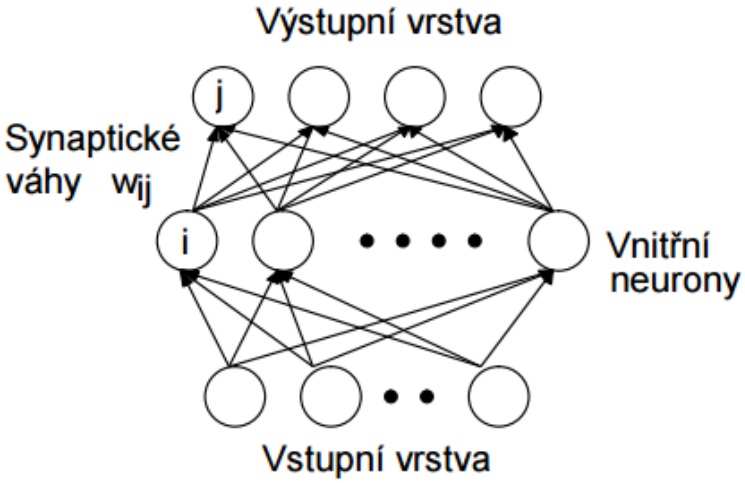
Aktivačná funkcia neurónu udáva hodnotu expozície y a tá sa niekedy označuje aj ako prenosová funkcia. Takže prenosová funkcia S uskutočňuje vnútorný prevod potenciálu neurónu na definovaný obor výstupných hodnôt y. Najčastejšie sa obor výstupných hodnôt obmedzuje intervalom (0,1) alebo (-1,1). Na nasledujúcom obrázku je možné vidieť tvar a matematické vyjadrenie najpoužívanejších prenosových funkcií. [14]



Obrázok č.14: Tvar a vyjadrenie najčetejšie využívaných funkcií [14]

2.2 Viacvrstvové siete a model backpropagation

Medzi najrozšírenejší spôsob prepojenia spojitých perceptrónov patria viacvrstvové siete. Topológiu siete je možné vidieť na nasledujúcom obrázku. [11]



Obrázok č.15: Viacvrstvová neurónová sieť [11]

Na uvedenom znázornení je zreteľne vidieť, že neurónová sieť sa skladá minimálne z troch vrstiev neurónov a to sú vstupná, výstupná a minimálne jedna vnútorná vrstva. Medzi dvoma susediacimi vrstvami je úplné neurónové prepojenie, čo znamená, že každý neurón z nižšej vrstvy je prepojený s každým neurónom z vyššej vrstvy. V tomto type neurónovej siete sa informácie spracúvajú dopredným (feedforward) šírením signálu. Postup spracovania je nasledovný:

- *"Najprv sú excitované na zodpovedajúcu úroveň (v rozmedzí 0 až 1) neuróny vstupnej vrstvy.*
- *Tieto excitácie sú pomocou väzieb privedené k nasledujúcej vrstve a upravené (zosilnené či zoslabené) pomocou synaptických váh.*
- *Každý neurón tejto vyššej vrstvy vykoná sumár upravených signálov od neurónov nižšej vrstvy a je excitovaný na úroveň danú svojou aktivačnou funkciou.*
- *Tento proces prebieha cez všetky vnútorné vrstvy až k vrstve výstupnej, kde potom získame excitačné stavy všetkých jej neurónov"* [11].

Takýmto spôsobom sa získava odozva neurónovej siete na vstupné podnety, ktoré sú dané excitáciou neurónov výstupnej vrstvy. Aj v biologickom systéme prebieha podobný spôsob šírenia signálov, kde napríklad vstupnú vrstvu tvoria zrakové bunky a vo výstupnej vrstve, mozgu, sú rozpoznané jednotlivé pozorované objekty. Korektná odozva na

vstupnom signály je stanovená synaptickými váhami. Samotný proces stanovenia synaptických váh je opäť spojený s pojmom učenia neurónovej siete. [11,13]

Pre učenie neurónovej siete je primárne potrebná trénovacia množina, ktorá obsahuje prvky popisujúce riešenú problematiku a ďalej taká metóda, aby bola schopná tieto prvky zafixovať v neurónovej vrstve vo forme synaptických váh. Trénovacia množina určuje, akým spôsobom sú excitovaná neuróny vstupnej a výstupnej vrstvy. Definícia trénovacej množiny T hovorí o množine prvkov alebo vzoriek, ktoré sú určené, ako usporiadané dvojice, nasledujúcim spôsobom: [11]

$$T = \{\{I_1,O_1\}\{I_2,O_2\}.....\{I_p,O_p\}\},$$
$$I_i = (i_1,i_2,.....i_k),\ i_j \in (0,1),$$
$$O_i = (o_1,o_2,....o_m),\ o_j \in (0,1).$$

Kde *p* predstavuje počet vzoriek v trénovacej množine, I_i je vektor excitácie vstupnej vrstvy tvorenej *k* neurónmi, O_i je vektor excitácie výstupnej vrstvy tvorenej *m* neurónmi a i_j, o_j sú excitácie j-tého neurónu vstupnej alebo výstupnej vrstvy. [11]

Pojem backpropagation predstavuje metódu, ktorá umožní adaptáciu neurónovej siete nad danú trénovaciu množinu. Na rozdiel od spomínaného dopredného šírenia signálu táto metóda pracuje v opačnom smere. Teda informácie šíri od vyšších vrstiev k nižším. Postup tejto metódy je nasledovný: [11]

- *"Vezmeme najprv vektor I_i i-teho prvku trénovacej množiny, ktorým excitujeme neuróny vstupnej vrstvy na zodpovedajúcu úroveň.*
- *Známym spôsobom vykonáme dopredné šírenie tohto signálu až k výstupnej vrstve neurónov.*
- *Porovnáme požadovaný stav daný vektorom O_i i-teho prvku trénovacej množiny so skutočnou odozvou neurónovej siete.*
- *Rozdiel medzi skutočnou a požadovanou odozvou definuje chybu neurónovej siete. Túto chybu potom v určitom pomere - learning rate - "vraciame späť" do neurónovej siete formou úpravy synaptických váh medzi jednotlivými vrstvami smerom od horných vrstiev k vrstvám nižším tak, aby chyba pri nasledujúcej odozve bola menšia.*
- *Po vyčerpaní celej trénovacej množiny sa vyhodnotí celková chyba cez všetky vzory trénovacej množiny a ak je táto vyššia než požadovaná, celý proces sa opakuje znova"* [11].

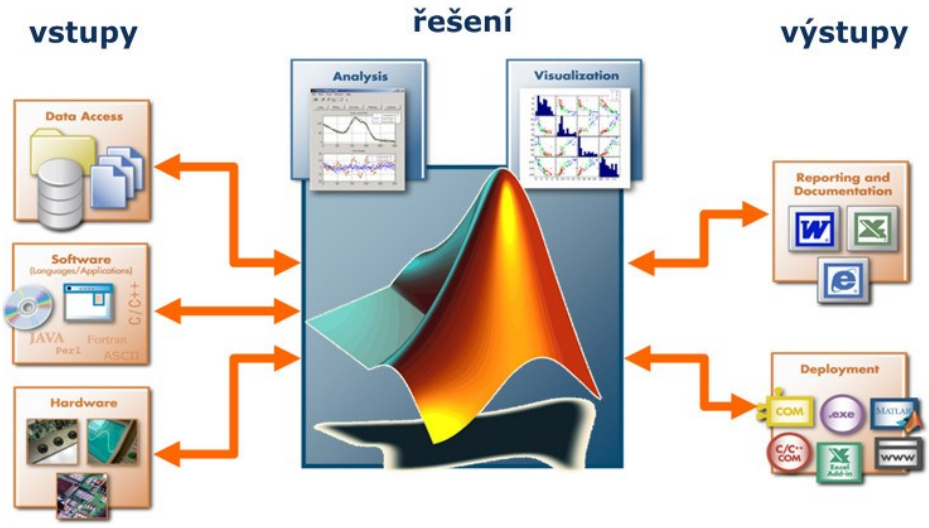
3 MATLAB

Matlab je skratka z anglického Matrix Laboratory, čo znamená maticové laboratórium. Je to interaktívne prostredie, ktoré sa využíva na vedecké výpočty. V tomto prostredí sa spájajú technické výpočty, vizualizácia dát a programovací jazyk. Matlab obsahuje veľké množstvo prídavných modulov a tým vytvára ideálne prostredie pre vedcov, inžinierov alebo učiteľov pri riešení problémov z mnohých oblastí. Spomínanú aplikáciu vyvíja spoločnosť MathWorks a distribuuje ju s veľkým množstvom rozšírení, ktoré sa nazývajú toolboxy. [18]

Matlab neposkytuje užívateľom len grafické a výpočtové nástroje, ale aj špecializované knižnice a výkonný programovací jazyk. Knižnice sú tak rozsiahle, že ich je možné využiť v každej sfére ľudských činností. Matlab je nástroj, ktorý umožňuje pohodlnú interaktívnu prácu, vývoj širokého spektra aplikácií, meranie a spracovanie signálu, návrhy algoritmov, simuláciu, analýzu a prezentáciu dát apod. Vďaka svojej architektúre je aplikácia určená najmä pre ľudí, ktorí potrebujú riešiť zložité výpočtové úlohy. Najväčšou výhodou Matlabu je jednoduchosť jeho jazyka, ktorý je oveľa jednoduchší ako napríklad programovací jazyk C. Taktiež obsahuje rýchle výpočtové jadro s optimálnymi algoritmami, ktoré boli preverené dlhoročnými prevádzkami po celom svete. Matlab je podporovaný a pracuje na všetkých významných typoch operačných systémoch ako je Windows, Linux alebo MAC. [19]

Kľúčové vlastnosti Matlabu:

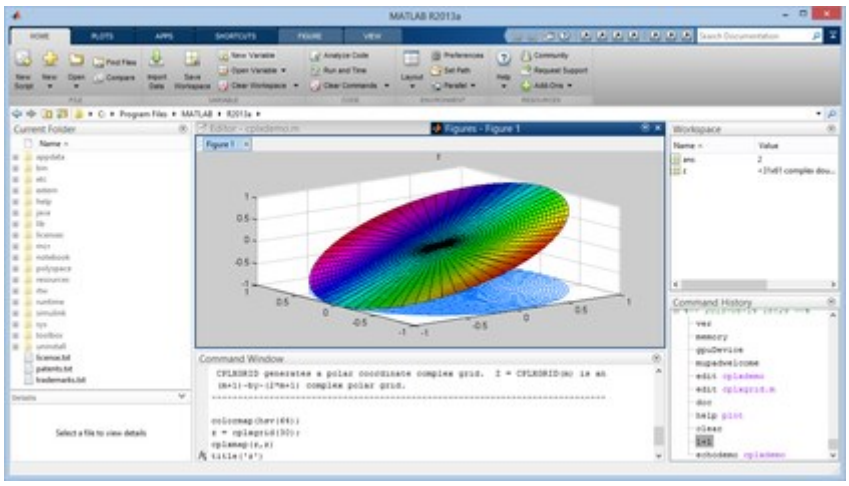
- *"vysokoúrovňový jazyk pre technické výpočty,*
- *otvorený a rozšíriteľný systém,*
- *veľké množstvo aplikačných knižníc,*
- *podpora viacrozmerných polí a dátových štruktúr,*
- *interaktívne nástroje pre tvorbu grafického používateľského rozhrania,*
- *import a export dát do mnohých formátov,*
- *komunikácie s externými meracími a monitorovacími prístrojmi v reálnom čase,*
- *rozšíriteľnosť modulov jazyky C, C ++, Fortran, Java"* [18].



Obrázok č.16: Tok informácií a dát v programe Matlab [19]

3.1 Popis prostredia Matlabu

Hneď ako sa spustí Matlab objaví sa pracovné prostredie, v ktorom je niekoľko častí. Najdôležitejšou a taktiež najpoužívanejšou časťou je Command Window, ktorá slúži na komunikáciu s výpočtovým jadrom programu. Do spomínanej časti sa vpisujú príkazy, ale taktiež je v nej možné sledovať výpisy chybových hlásení, varovania alebo obsah premenných. Ďalšou časťou je Workspace, v ktorej je možné sledovať použité premenné. Command History slúži na priebežné ukladanie všetkých príkazov, ktoré sa napísali do Command Window. Poslednou časťou je Current Directory, to je aktuálny pracovný adresár a ten je možné ľubovoľne meniť. Na nasledujúcom obrázku je možné vidieť pracovné prostredie programu Matlab. Výzor pracovného prostredia sa mení v závislosti od použitej verzie programu. [18]



Obrázok č.17: Pracovné prostredie [20]

3.2 M-súbory

Pri práci s Matlabom sa všetky príkazy zapisujú do už spomínaného Command Window. Po potvrdení príkazu prostredníctvom klávesy Enter je možné ihneď vidieť odozvu alebo výsledok práce. Tento spôsob práce je vhodný pri jednoduchých úlohách. No v prípade ukončenia Matlabu alebo neočakávaného vypnutia počítača by boli príkazy už nedostupné a museli by sa písať nanovo. Z toho dôvodu sa zavádzajú takzvané m-súbory. [21]

M-súbory obsahujú všetky príkazy aké boli použité a v adresári, ktorý bolo vybraný na uloženie práce, je ich možné nájsť pod príponou *.m. M-súbor je možné vytvoriť v ľubovoľnom textovom editore. Jedinou podmienkou je, aby daný editor nepridával žiadne informácie ohľadom formátovania a hlavne musí podporovať zápis vlastnej prípony *.m. Najjednoduchším spôsob vytvorenia m-súboru je využiť vstavaný editor v samotnom Matlabe. Matlab vytváraním m-súborov pripomína programovanie, ktoré je známe z iných vývojárskych prostredí. [21]

M-súbory sa rozdeľujú na dve základné skupiny:

- skripty,
- funkcie.

Pojmom skript sú označené jednoduché m-súbory a v nich sú obsiahnuté príkazy Matlabu. V momente, kedy je spustený skript v Matlabe, spúšťa príkazy, ktoré našiel v danom súbore. Použité príkazy skriptového súboru pracujú globálne s dátami. Skripty sa využívajú na uskutočnenie analýz, riešenie problémov alebo vytvorenie dlhých príkazov, ktoré by inak trvali pomerne dlhú dobu. [21]

Funkcie taktiež patria medzi m-súbory, no od skriptov sa odlišujú tým, že dokážu pracovať s jedným alebo viacerými vstupnými parametrami a na výstup predávajú jeden alebo viac parametrov. To sa využíva v prípade, ak jedným m-súborom voláte iný a to sa prejaví predaním jeho výsledkov. Funkcia sa dá v podstate nazvať vylepšeným skriptom, ktorý má väčšie využitie a umožňuje vytvárať funkcie k už existujúcim funkciám. [21]

3.3 Toolbox

Toolbox je v podstate knižnica funkcií. Tieto knižnice umožňujú rozšíriť program do patričných vedných a technických odborov. Toolboxy ponúkajú špecializované funkcie, ktoré je možné rozširovať modifikovať alebo len čerpať z nich informácie. Aj toolbox je možné klasifikovať ako m-súbor, ktorý bol vytvorený na podporu riešenia úloh s rôznym pracovným zameraním.[19]

3.3.1 Image Processing Toolbox

Image Processing Toolbox je výkonný a pomerne ľahko ovládateľný nástroj na spracovanie a analýzu obrazu. Matlab má vybudované nadstavby na návrhy, rekonštrukciu a analýzu obrazov, manipuláciu s farbami, geometriou a štruktúrou obrazov. Matlab spolu s Image Processing Toolboxom je pre jeho výpočtovú mohutnosť a štruktúru aplikačných knižníc vynikajúcim nástrojom pre odbor digitálneho spracovania obrazu. [22]

3.3.2 Image Acquisition Toolbox

Tento toolbox umožňuje získať obrazové dáta a video zo záznamových zariadení priamo do Matlabu. Hardware sa deteguje automaticky a umožňuje upravovať jeho nastavenia, zobraziť náhľad alebo uložiť obraz priamo na disk. Podporuje širokú škálu záznamových zariadení, tým je myslené, že nezáleží na tom, či je pripojené lacnejšie zariadenie alebo špičková vedecká kamera. [22]

3.3.3 Neutral Network Toolbox

Je nástroj pre návrh, tvorbu, simuláciu, vizualizáciu alebo trénovanie neurónových sietí. Tento druh sietí sa uplatňuje tam, kde je zložité alebo nemožné použiť formálne analytické nástroje ako napríklad rozpoznávanie a riadenie nelineárnych sústav. Tento druh toolboxu ponúka grafické rozhranie a to jednoznačne uľahčuje prácu s neurónovými sieťami. Taktiež obsahuje možnosť tvorby funkcií a sietí prostredníctvom modulárnej, rozšírenej a otvorenej stavby toolboxu. Súčasné verzie umožňujú prácu so základnými modelmi neurónových sietí, ale taktiež aj s novými variantmi učenia, ktorých súčasťou sú rýchle a optimalizačné algoritmy. [23]

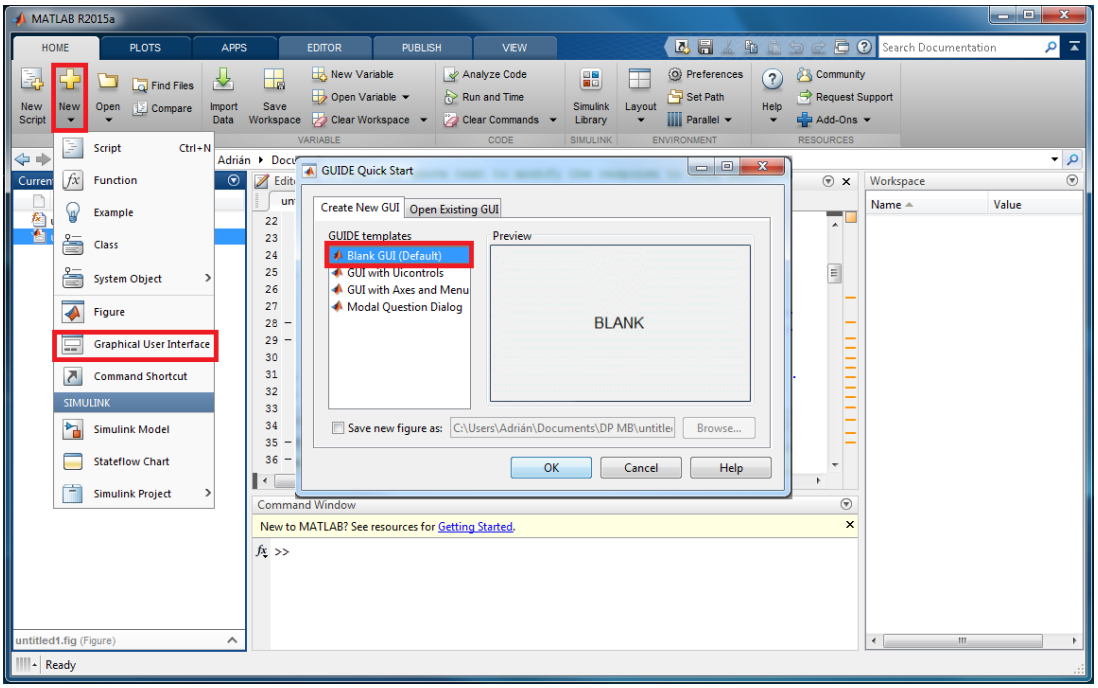
II. PRAKTICKÁ ČASŤ

4 METÓDA POROVNÁVANIA OBJEKTOV PROSTREDNÍCTVOM PROGRAMU MATLAB

V praktickej časti sa budem zaoberať tvorbou vlastného programu na rozpoznávanie objektov v obraze prostredníctvom programu Matlab. Ako prvé bolo nutné vymyslieť celú štruktúru programu a hlavne pre uľahčenie práce s programom vytvoriť grafické užívateľské rozhranie (v Matlabe pomenované a ďalej používané len GUI).

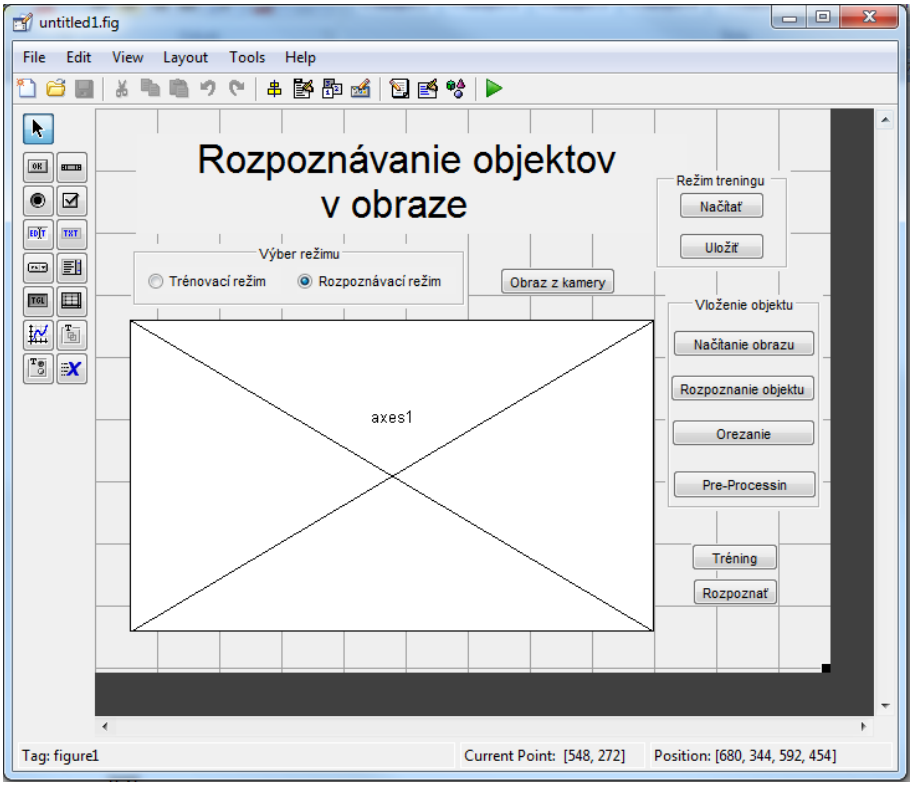
4.1 Návrh a vytvorenie GUI

GUI vytvoríme cez panel NEW, následne vyberieme možnosť Graphical User Interface. Potom sa spustí panel GUIDE Quick Start, kde si môžeme vybrať typ aký chcem využiť. Ja som použil typ Blank GUI (Defoult), aby som si mohol navrhnúť vlastné prostredie.



Obrázok č.18: Prostredie matlab

Následne sa mi vytvoril súbor s príponou *.fig a otvoril sa grafický editor pre návrh rozhrania. V tomto editore si vložíme všetky tlačidlá a stredový panel kde budeme neskôr vidieť prácu celej aplikácie.



Obrázok č.19: Vytvorené užívatelské grafické rozhranie

4.2 Funkcie aplikácie a ich popis

V tejto časti práce sa budem zaoberať popisom všetkých tlačidiel a ich funkcií, ktoré som vložil cez grafické rozhranie. Ako prvé som vykonal inicializáciu celého programu. V podstate to bolo uskutočnené automaticky, keďže Matlab samostatne vygeneruje funkciu s názvom „názov súboru´´_OpeningFcn. Ďalej som už písal príkazy, ktoré slúžili na blokovanie niektorých tlačidiel. Keďže nemá zmysel používať ich kým nie je načítaný obraz. Boli to tlačidlá na výber, orezanie, úpravu a samotné rozpoznanie. Na to aby boli tlačidlá znepristupnené do správnej doby, som použil príkaz *set* a nastavil parameter *enable* na hodnotu *off*. Ďalej sa nastaví aj pole kde sa budú ukladať názvy objektov, premenná pre celkový počet načítaných objektov a premenná pre ukladanie počtu typov objektov. Príkazy vyzerali nasledovne:

```
handles . cnt=1;
handles . pocobj=1;
handles . menaObj='' ;

set ( handles . rozpButton , ' enable ' , ' off ' ) ;
set ( handles . vyberButton , ' enable ' , ' off ' ) ;
set ( handles . orezButton , ' enable ' , ' off ' ) ;
set ( handles . pripravaButton , ' enable ' , ' off ' ) ;
```

4.2.1 Načítanie obrazu

Ako prvé po spustení aplikácie by sme mali načítať obrázok, či už to bude obrázok, ktorý je uložený v PC alebo prostredníctvom pripojeného záznamového zariadenia. Ako prvé ozrejmím načítanie obrázku z PC.

Načítanie obrázku sa uskutoční po stlačení príslušného tlačidla v grafickom rozhraní. Po jeho stlačení sa zavolá funkcia loadBotton_Callback. Načítanie obrázku sa uskutoční cez príkaz *imread.* Tento príkaz vyžaduje dva parametre. Prvý je cesta k súboru a druhý je názov samotného súboru. Prvý parameter sa zadávať nemusí, ale pre uľahčenie práce s aplikáciou je zadaná cesta k pracovnému súboru. Na začiatku som spomínal že aplikácia má grafické rozhranie, takže by nebolo správne, aby užívateľ zadával cestu k súboru cez príkazový riadok. Aby sa vyvolalo štandardné okno na výber súboru použil som príkaz *uigetfile.* V parametroch tohto príkazu sa zadávajú typy súborov, ktoré sa budú dať otvoriť. Teraz sa obrázok bude dať otvoriť no treba ešte náhľad v hlavnom okne aplikácie. Pre zobrazenie načítaného obrázku som použil príkaz *imshow,* do ktorého som zapísal ako parameter premennú nášho obrázku. Ešte je potrebné definovať, kde sa má obrázok otvoriť, inak by sa zobrazil v novom okne. Na to slúži príkaz *axes*, keďže v GUI je náhľadové okno nazvané mainAxes. Vo funkcii som povolil aj jedno tlačidlo, presnejšie Výber, ktorý bude neskôr potrebný. Taktiež sa zobrazí aj dialógové okno s informáciu o úspešnosti načítania. Toto okno sa vyvoláva cez príkaz *msgbox*, ktorý má ako parametre text pre užívateľa a titul okna. Príkaz *msgbox* som zaobalil príkazom *uiwait*, čo slúži na to, aby si užívateľ mohol prečítať správu a musel následne použiť dialógové okno. Všetko čo bolo spomenuté vyššie je v nasledovnej funkcii.

```
function loadButton_Callback ( hObject , eventdata , handles )

[ filename , pathname ] = uigetfile({'*.bmp' ; '*.jpg' ; '*.gif ' ; ' *.*'},
'Vyber obrázok ');
S~= imread ( [ pathname , filename ] ) ;
uiwait (msgbox ( ' Trénovacie dáta načítané ! ' , 'Dáta načítané ! ' ) ) ;

axes ( handles . mainAxes ) ;
imshow(S ) ;
handles.S~= S ;

set ( handles.vyberButton , 'enable' , ' on ' ) ;

guidata ( hObject , handles ) ;
```

Druhou možnosťou ako získať obraz je prostredníctvom kamery. Pre tento účel som použil klasickú webkameru. Načítanie obrázku z kamery uskutočníme prostredníctvom tlačidla v GUI, Obraz z kamery. Tým sa spustí funkcia kamButton_Callback. Z tohto dôvodu som musel vytvoriť objekt, ktorí reprezentuje vstup z kamery. Tento objekt je v podstate spojenie medzi Matlabom a konkrétnym zariadením, ktoré používame na snímanie obrazu. Na vytvorenie takéhoto objektu som použil príkaz *videoinput.* To vyžaduje ako povinný parameter názov pripojeného zariadenia, čo bola v tomto prípade webkamera čiže som mohol použiť všeobecný názov, nakoľko som pracoval na nootebooku. No pre prípad pripojenia inej kamery, je príkaz imaqhwinfo, ktorý zobrazí všetky pripojené zariadenia s ich názvami. Druhým parametrom bolo ID zariadenia. Ak nebolo zadané, automaticky vyberie prvé možné zariadenie. Ako tretí a posledný parameter som zadával rozlíšenie obrazu. Ja som si vybral 640x480. Funkcia zodpovedajúca popisu:

```
function kamButton_Callback ( hObject , eventdata , handles )

vid = videoinput ( ' winvideo ' , 1 , 'RGB24 640x480 ' ) ;
axes ( handles . mainAxes ) ;

vidRes = get ( vid, ' VideoResolution ' ) ;
nBands = get ( vid, 'NumberOfBands ' ) ;
hImage = image( zeros ( vidRes ( 2 ) , vidRes ( 1 ) , nBands ) ) ;
```

Snímanie z kamery už bolo pripravené, teraz som potreboval funkciu na zobrazenie náhľadu. To sa robí prostredníctvom príkazu *preview.* Keďže po zapnutí kamery chvíľu trvá, kým sa obraz vyjasní a ustáli nie je možné ihneď snímať daný obraz. Preto som použil príkaz *pause,* aby som pozastavil program a dobu čakania som stanovil na 4 sekundy. Až po uplynutí stanoveného času bude možné získať obraz a to sa robí prostredníctvom príkazu *getsnapshot*. Získaný obraz sa nám zobrazí v náhľadovom okne. Potrebná funkcia pre spomínaný postup:

```
preview ( vid , hImage ) ;
pause ( 4 ) ;
snap=get snapshot ( vid ) ;

imshow( snap ) ;
handles . img crop=snap ;
set ( handles . pripravaBut , ' enable ' , ' on ' ) ;

guidata ( hObject , handles ) ;
```

4.2.2 Výber a orezanie obrazu

Funkcia Výber objektu spúšťa proces vyberania objektu v obraze, aby sme vedeli aký objekt budeme hľadať. Túto funkciu využijeme v prípade, ak bude na obraze viacero objektov. Po stlačení tlačidla sa spustí funkcia vyberButton_Callback. Funkcia zobrazí výberový obdĺžnik, ktorý slúži na označenie objektu. Výberový obdĺžnik je možne meniť podľa veľkosti požadovaného objektu. Pre výber použijeme nasledovnú funkciu:

```
function vyberButton_Callback ( hObject , eventdata , handles )
S~= handles.S ;
axes ( handles.mainAxes ) ;

if       isfield ( handles , 'api ' )
         handles . api . delete ( ) ;
         rmf ield ( handles , ' api ' ) ;
         rmf ield ( handles , ' hRect ' ) ;
         axes ( handles.mainAxes ) ;
         imshow(S ) ;
end

axes ( handles . mainAxes ) ;
s z = size (S ) ;

handles.hRect = imrect (gca , [ round( sz ( 2 ) / 2 ) round( sz ( 1 ) / 2 ) 50 50 ] ) ;
handles.api = iptgetapi ( handles . hRect ) ;

set ( handles.orezButton , ' enable ' , ' on ' ) ;
guidata ( hObject , handles ) ;
```

Po stlačení tlačidla orezanie, sa spustí funkcia orezButton_Callback. Funkcia najskôr zisťuje polohu výberového obdĺžnika a následne sa uskutoční orezanie obrázku cez funkciu *imcrop*. Táto funkcia má dva parametre. V prvom sa uvádza obrázok, ktorý bude orezaný a druhý ako bude orezaný. Po orezaní sa výsledný obrázok zobratí na náhľadovom poli. Funkcia na orezanie obrázku:

```
function orezButton_Callback ( hObject , eventdata , handles )

handles.loc = handles.api.get Position ( ) ;
assignin ( ' base ' , ' api ' , handles.api ) ;
axes ( handles . mainAxes ) ;
S~= handles.S ;

handles.img_crop = imcrop(S , handles.loc ) ;
axes ( handles.mainAxes ) ;
imshow( handles.imgcrop ) ;
```

```
set ( handles.pripravaButton , ' enable ' , ' on ' ) ;
guidata ( hObject , handles ) ;
```

4.2.3 Pre-processing

Ďalšou funkciu je Pre-processing, alebo predspracovanie obrazu. Po spustení sa vyvolá príkaz pripravaButton_Callback. Ako prvé načíta orezaný obrázok a následne ho prevedie z farebného modelu do odtieňu šedej. Teda odstráni sa farebný tón, sýtosť, ale zachová sa jas obrázku. Tento krok sa uskutoční prostredníctvom rgb2gray. Ako ďalší krok nasleduje prahovanie. Na to som využil dve funkcie. Prvá bola *graythresh,* ktorá vypočíta globálne prahové hodnoty. Prahom bude normalizovaná hodnota intenzity v rozmedzí 0-1. Následne príkazom *im2bw* spustíme vlastné prahovanie. Z toho získame binárny obrázok, ktorý obsahuje len čiernu a bielu farbu. Tento postup má nasledujúcu funkciu:

```
function pripravaButton_Callback ( hObject , eventdata , handles )

img crop = handles.img crop ;
imgGray = rgb2gray ( img_crop ) ;
prah=graythresh ( imgGray ) ;
bw = im2bw( img_crop,prah ) ;
```

Nasleduje funkcia automatického orezania, ktorá oreže objekt až k jeho hranám, tak aby na ňom neboli žiadne biele plochy. Pre zjednodušenie som vytvoril skript s názvom auto_orez a vyzerá nasledovne:

```
bw2 = auto_orez (bw) ;
axes ( handles.mainAxes ) ;
imshow(bw2 ) ;
handles.bw2 = bw2 ;
```

Vstupným parametrom skriptu je orezaný obrázok. Najskôr sa zistia a uložia rozmery objektu a uskutoční sa inicializácia potrebných premenných.

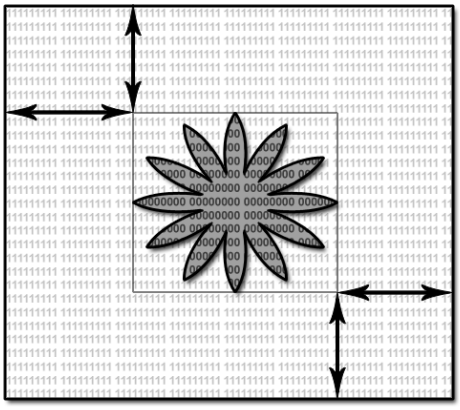
```
function bw2 = auto_orez(bw)
[ y2pom x2pom] = size (bw) ;
x1=1;
y1=1;
x2=x2pom;
y2=y2pom;
```

Teraz nasledujú cykly na zistenie bielych medzier v obrázku. Fungujú na princípe sumy hodnôt prvkov v stĺpci matice. Pokiaľ budú v stĺpci hodnoty 1 (biela farba) budú sa priratúvať. V momente, keď sa objaví hodnota 0 (čierna farba = obraz), cyklus skočí ďalej. Takto sme získali vzdialenosť, ktorá sa má odstrániť. Najskôr sa odstránia biele medze-

ry zľava, následne nad obrázkom, sprava a na koniec pod obrázkom. Cykly na vyhľadávanie bielych medzier:

```
pocB=1;
while (sum(bw( : , pocB))==y2pom)
        x1=x1+1;
        pocB=pocB+1;
end

pocB=1;
while (sum(bw(pocB, : ) )==x2pom)
        y1=y1+1;
        pocB=pocB+1;
end

pocB=x2pom;
while (sum(bw( : , pocB))==y2pom)
        x2=x2-1;
        pocB=pocB-1;
end

pocB=y2pom;
while (sum(bw(pocB, : ) )==x2pom)
        y2=y2-1;
        pocB=pocB-1;
end
```

Pri hľadaní bieleho priestoru nad obrázkom sa prechádzali riadky obrázka, vpravo sa prechádzali stĺpce z prava do ľava apod. Takto sme získali štyri body, ktoré vytvorili nový orezávací sektor vďaka čomu bude obraz lepšie orezaný. To je možné vidieť na obrázku č.20.



Obrázok č.20: Hľadanie bieleho priestoru a vytvorenie automatickéhoorezávacieho sektoru [22]

Následne sa pomocou príkazu *imcrop* uskutoční finálne orezanie a obrázok je výstupom daného skriptu. Funkcia pre orezanie:

```
bw2=imcrop (bw, [ x1 , y1 , ( x2-x1 ) , ( y2-y1 ) ] ) ;
```

Teraz je potrebné vytvoriť premennú NNdata, podľa ktorej sa vytvorí matica obrázkov, aby sa ich mohla neurónová sieť učiť. Keďže všetky obrázky musia mať rovnakú veľkosť, boli upravené pred uložením do spomínanej databázy. V momente, ako sa vloží obrázok do databázy sa navršuje premenná ukazujúca celkový počet obrázkov v databáze. Pri vkladaní obrázku sa s ním rotuje o 360° a rotácia prebieha po 45°. Každá rotácia sa ukladá ako nový obrázok a je to z dôvodu zlepšenia učenia neurónovej siete. Toto pomôže pri rozpoznávaní, keďže každý objekt bude v databáze uložený v rôznych uhloch. Rotáciu som uskutočnil cez príkaz *imrotate*, ktorého parametre sú obrázok, uhol rotácie a interpolačná metóda. Pred rotáciou som musel obrázok invertovať, pre správnu prácu a následne po ukončení rotácie sa invertuje späť a je orezaný pomocou funkcie auto_orez. Celá funkcia vyzerá nasledovne:

```
i f get ( handles.radiotren , ' value ' ) == 1
        for i ~= 1 : 8
                imshow(bw2)
                pause ( 1 ) ;
                handles.NNdata ( : , : , handles.pocobj )=imresize (bw2 , [ 80 , 80 ] ) ;

                bw2 = (bw2 == 0 ) ;
                bw2=imrotate (bw2 , 4 5 , ' bilinear ' ) ;
                bw2 = (bw2 == 0 ) ;
                bw2 = auto_orez (bw2 ) ;
                handles.pocobj=handles.pocobj+1;
        end

else
        handles.NNdata ( : , : , handles.cnt )=imresize (bw2 , [ 80 , 80 ] ) ;
end

handles.cnt=handles.cnt+1;
```

Funkcia rozpoznávania objektov je prístupná v oboch režimoch, trénovacom aj rozpoznávacom. Takže funkcia musí mať aj podmienku, aby vedela rozlíšiť funkciu predspracovania, či už pre trénovací alebo rozpoznávací režim. V trénovacom režime sa najskôr zobrazí okno, kde je potrebné vypísať názov obrázku, aby ho bolo možné identifikovať. Ďalej sa vypíše oznam o úspešnom uložení do databázy. Teraz ešte budú sprístupnené všetky tlačidlá, aby bolo možné vložiť viac objektov, ktoré sa naučí neurónová sieť. V roz-

poznávacom režime pracuje funkcia inak. Sprístupní len tlačidlo ma rozpoznávanie a informuje nás o úspešnom načítaní obrázku. Vyššie popísaná funkcia vyzerá nasledovne:

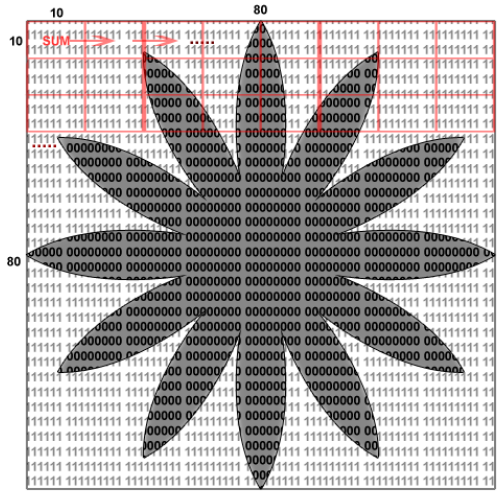
```
i f get ( handles.radiotrening , ' value ' ) == 1

prompt={'Názov objektu ' }
de fans={ ' ' }
f ields = { 'name ' }
info = inputdlg ( prompt , ' Vložte prosím názov objektu ' , 1 , defans )

i f      ~isempty( info )
         info = cell2struct ( info,fields )
         myname = info.name
         uiwa i t (msgbox ([ 'Tento objekt bol pomenovaný ako: ' myname ] ,' Potvrdenie ' ));
end

handles.menaObj=[ handles.menaObj ; cellstr (myname ) ];

uiwait (msgbox ( 'Objekt bol vložený do databáze !
Vložte prosím ďalší objekt alebo uskutočnite trénovanie ' , 'Objekt vložený ! ' ) ) ;

el se
uiwait (msgbox ( 'Objekt načítaný, teraz je možné použiť rozpoznávanie . ' ,
'Objekt vložený ! ' ) ) ;
set ( handles.rozpButton, ' enable ' , ' on ' ) ;
end

set ( handles.pripravaButton , ' enabl e ' , ' off ' ) ;
set ( handles.orezButton , ' enabl e ' , ' off ' ) ;
set ( handles.vyberButton , ' enabl e ' , ' off ' ) ;
guidata ( hObject , handles ) ;
```

4.2.4 Tréning neurónovej siete

Keďže sme spracovali a vložili objekty do databáze, je potrebné na nich natrénovať neurónovú sieť. To sa uskutočňuje cez tlačidlo Tréning a tým sa vyvolá funkcia treningButton_Callback. Najskôr je nutné overiť, či databáza obsahuje nejaké objekty a následne sa vyvolá skript traindata. Ten slúži na vytvorenie premenných, aby sa mohla vytvoriť a natrénovať neurónová sieť. Funkcia kontrola databáze a vyvolanie skriptu traindata:

```
function treningButton_Callback ( hObject , eventdata , handles )

hasField = isfield ( handles , 'NNdata ' )

i f hasField
NNdata = handles.NNdata ;
[ NNinput NNtarget ]=traindata (NNdata , handles.cnt , handles.pocobj ) ;
```

Skript traindata obsahuje na vstupe dva parametre, prvým je premenná NNdata (obsahuje matice vložených obrázkov) a druhým je celkový počet objektov v databáze. Skript sa skladá z troch cyklov. Prvý rozlišuje jednotlivé matice obrázkov, druhý určuje spracovanie po riadkoch a posledný spracovanie stĺpcov v danom riadku. V podstate to funguje tak, že sa postupne prejde celá matica po blokoch (veľkosť bloku je 10x10) a v každom bloku sa spočítajú všetky hodnoty. Obrázky som stanovil na pevnú veľkosť 80x80 a bloky sú veľkosti 10x10 čiže na jednom obrázku bude 64 blokov. Každý blok nám udáva jednu hodnotu (celkom 64 hodnôt) a tá bude pre úpravu vydelená hodnotou 100. Celý postup je znázornený na obrázku č.21. Takto sa spracujú všetky objekty a ich hodnoty sa zapíšu do premennej NNimput a sú poslané na výstup. Na výstup sa pošle aj premenná NNtarget, ktorá obsahuje cieľové hodnoty. NNtarget vytvoríme cez príkaz *zeros(M,N)* a ten nám vytvorí nulovú maticu o rozmere MxN. Rozmer matice nám určí celkový počet objektov na učenie (počet stĺpcov) a počet druhov objektov (počet riadkov). Následne sa prejde matica po riadkoch prostredníctvom cyklov *for* a doplňuje sa hodnota 1 na príslušné miesto. Podľa toho v akom riadku a stĺpci sa hodnota 1 nachádza sa určí o aký objekt ide. Hodnota 1 sa vyplňuje po ôsmich a je to z dôvodu, že každý objekt má 8 rotácií. Prechádzanie matice po blokoch a vyššie popísaná funkcia budú vyzerať nasledovne:



Obrázok č.21:Prechádzanie matice po blokoch [22]

```
function [ NNinput , NNtarget ] = traindata (x , poctypu , pocobj )

obr = x ;
poctypu=poctypu -1;
pocobj=pocobj -1;
```

```
for cntout = 1 : pocobj
        for cnt=1:8
                for cnt2=1:8
                        obrpom=sum( obr ((cnt *10-9:cnt*10 ) , ( cnt2*10-9:cnt2*10),
                        cntout ));
                        obrpom2 ((cnt-1)*8+cnt2)=sum( obrpom ) ;
                end
        end
        pic NNinput ( cntout , : )=obrpom2 /100;
end

NNinput=pic NNinput ' ;
NNtarget=zeros ( poctypu , pocobj ) ;

pociatok=1;
c i l =8;
for i =1: poctypu
        for j =1: pocobj
                i f j>=pociatok && j<=ciel
                        NNtarget ( i , j )=1;
                end
        end
        pociatok=pociatok+8;
        ciel=ciel+8
end
```

Teraz musíme vyvolať ďalší skript s názvom createnn. Ten vytvorí neurónovú sieť a trénuje ju na vstupných dátach.

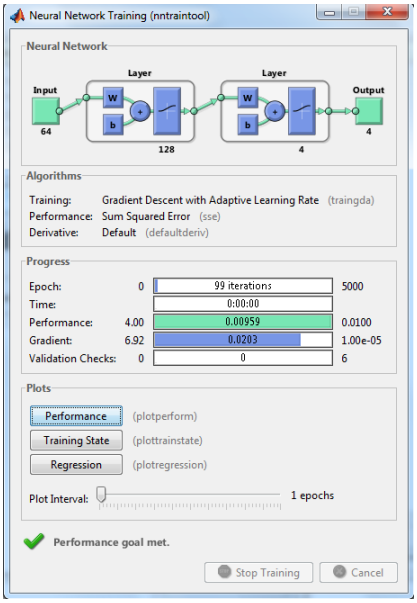
```
[ net , tr ] = creatnn (NNinput , NNtarget , handles.cnt )
handles . net=net ;
```

Creatnn potrebuje na vstup dáta z výstupu traindata, teda NNinput, NNtarget a taktiež informácie o počte objektov. Je niekoľko rôzny spôsobov ako vytvoriť neurónovú sieť. Ja využívam doprednú (feedforward) neurónovú sieť so spätným šírením (back-propagation) a z toho dôvodu som použil príkaz *newff*. Tento príkaz vyžaduje niekoľko parametrov. Prvým je rozsah vstupných hodnôt a ten získame cez príkaz *min-max(NNinput)*. Ďalší udáva počet neurónov v jednotlivých vrstvách. Vrstiev môže byť ľubovoľný počet, ja som použil dve (S1 = počet neurónov v prvej vrstve, S2 = počet neurónov v druhej vrstve). Hodnoty S1 a S2 si volí každý sám a je to skôr experiment. Lebo ak ich bude málo neurónová sieť sa slabo učí, ale ak ich bude príliš, predĺži sa doba učenia a môže dôjsť aj ku kolapsu siete. Jedna z metód hovorí, že by ich malo byť dvakrát viac ako je hodnôt na vstupe (64 blokov). Takže ako hodnotu S1 som zvoli 128 a za S2 celkový počet objektov. Nasledujúci parameter upresňuje prenosovú funkciu neurónov. Pre obe

vrstvy je zvolená *logsig*, čo je nelineárna funkcia, ktorú tvorení sigmoida. Posledný parameter je traingda. Ten udáva názov trénovacej funkcie. Vytvorená sieť sa ukladá do premennej pod názvom net. Ďalej som nastavil maximálny počet epoch na hodnotu 5000 a to z dôvodu, aby stačili aj pri väčšom množstve vstupných dát. Ešte som nastavil parametre *goal* (kedy sa má funkcia zastaviť pri chybe), *show* (ako často sa bude vykresľovať chybová funkcia, každých 20 epoch), a posledný parameter je hodnota chybovej funkcie *see* (Sum-Squared Error performance function). Celý skript createnn vyzerá nasledovne:

```
function [ net , tr ] = creatnn ( arg1 , arg2 , pocet )

NNinput = arg1 ;
NNtarget = arg2 ;
S1 = 128;
S2 = pocet -1;

net = newff (minmax(NNinput ) , [ S1 S2 ] , { ' logsig ' ' logsig ' } , ' traingda ' ) ;
net.LWf2 ,1g = net.LW { 2,1 } * 0.01 ;
net.b{2} = net.b { 2 } * 0.01 ;

net . performFcn = ' sse ' ;
net . trainParam . goal = 0.01 ;
net . trainParam . show = 20 ;
net . trainParam . epochs = 5000;

[ net , tr ] = train ( net , NNinput , NNtarget ) ;
```

Nastavil som všetko potrebné a už len spustiť proces tréningu. To sa vykoná prostredníctvom príkazu *train,* ako parametre som použil NNinput (vstupné dáta) a NNtarget (cieľové dáta). Po spustení príkazu *train* sa spustí aj nástroj Neural Network Training Tool. V tomto nástroji je možné vidieť informácie o sieti, topológiu neurónovej siete a počet neurónov v jednotlivých vrstvách, vykreslenie niekoľkých grafov popisujúcich priebeh učenia apod.



Obrázok č.22: Neural Network Training Tool

4.2.5 Rozpoznávanie objektov

Ako prvé musíme aktivovať rozpoznávací režim. To spravíme prostredníctvom GUI a výberom rozpoznávacieho režimu. Následne sa spustí funkcia radiotest_Callback. Funkcia overí, či existujú premenné NNdata a net. Teda či existuje nejaká databáza objektov a neurónová sieť. Ak neexistujú, nie je možné prejsť k rozpoznávaniu a aplikácia sa vráti do trénovacieho režimu. Funkcia na prechod do rozpoznávacieho režimu:

```
function radiotest_Callback ( hObject , eventdata , handles )
set ( handles.radiotest , ' value ' , 1 ) ;
set ( handles.radiotren , ' value ' , 0 ) ;
set ( handles.radiotren , ' enable ' , ' off ' ) ;

ha sField = isfield ( handles , 'NNdata ' )
hasField2 = isfield ( handles , ' net ' )

i f hasField && hasField2
        handles = rmf ield ( handles , 'NNdata ' )
        handles.cnt=1;

        set ( handles.treningButton , ' enable ' , ' off ' ) ;

        set ( handles.ulozbutton , ' enable ' , ' off ' ) ;
        set ( handles.nacitajbutton , ' enable ' , ' off ' ) ;

else
        warningMessage = sprintf ( ' Varovanie : Rozpoznávací režim je neprístupný,
        najskôr uskutočni tréning siete. ' ) ;
        uiwait ( warndlg ( warningMessage ) ) ;
        set ( handles.radiotren , ' enable ' , ' on ' ) ;
```

```
        set ( handles.radiotren , ' value ' , 1 ) ;
        set ( handles.radiotest , ' value ' , 0 ) ;
end

guidata ( hObject , handles ) ;
```

Ak existujú, je možné vložiť nové obrázky s objektmi na rozpoznanie. Vloženie, označenie, orezanie a príprava neznámeho obrázku sa uskutoční rovnako ako v trénovacom režime. Po vložení obrázka je možné spustiť rozpoznanie a to prostredníctvom tlačidla Rozpoznať. Aktivovaním toho tlačidla sa spustí funkcia rozpButton_Callback. Najskôr sa načítajú premenné NNdata a potom net. Ďalej sa vyvolá skript simdata, prostredníctvom neho získame vstupný vektor z načítaného obrázku. Simdata má dva vstupné parametre a to NNdata a počet objektov. Výstup bude vektor popisujúci objekt, ktorý bude vstupom do simulácie v neurónovej sieti. Teraz vykonáme simuláciu v neurónovej sieti prostredníctvom príkazu *sim*. Ten má ako parametre meno objektu a spomínaný vstupný vektor. Takto získame výstupný vektor simulácie, ktorý určí o aký objekt sa jedná, ak bol rozpoznaný správne. Výstupný vektor sa spracuje cez príkaz *compet.* To je prenosová funkcia, ktorá prevedie daný vektor do binárnej podoby, kedy najväčšia hodnota vo vektore bude nahradená 1 a všetky nižšie 0. Typ hľadaného objektu určuje index vektoru, čiže miesto s najväčšou hodnotou. Keďže máme vektor v binárnej podobe a najväčšia hodnota je 1 a ostatné sú na hodnote 0, bude sa ľahko hľadať tá, ktorú potrebujeme. To vykonáme cez príkaz *find.* Nakoniec sa v GUI vypíše príslušný názov o aký objekt ide. Celá funkcia vyzerá nasledovne:

```
function rozpButton_Callback ( hObject , eventdata , handles )

net=handles.net ;
NNdata = handles.NNdata ;
NNinput=simdata (NNdata , handles.cnt ) ;

NNoutput=sim( net , NNinput ) ;
NNoutput=compet (NNoutput ) ;
answer=find ( compet (NNoutput )==1);
txtAnswer=handles.menaObj ( int 8 ( answer ) ) ;
set ( handles.vyslPole , ' string ' , txtAnswer ) ;

handles.txtAnswer=txtAnswer ;
guidata ( hObject , handles ) ;
```

4.2.6 Ukladanie a načítavanie trénovacích dát

Poslednou funkciou je načítanie a ukladanie trénovacívh dát. Využijeme tieto funkcie najmä pri doplňovaní objektov do trénovacích súprav. Ak by sme nemali uložené dané súpravy a chceli by sme doplniť nejaký objekt, museli by sme všetky objekty pridávať odznova. Teraz len jednoducho načítame zvolenú sadu objektov a môžeme si ju upravovať. Taktiež máme možnosť si pripraviť viacero rôznych súprav na rozpoznávanie objektov.

Ako prvá je funkcia Uložiť. Spustením funkcie ulozButton_Callback, ktorá sa spustí tlačidlom Uložiť. Najskôr sa načítajú všetky premenné, ktoré sa budú mať uložiť. Sú to NNdata (objekty na učenie), cnt a pocobj (celkový počet objektov a objekty rovnakého typu) a menaObj (názvy objektov). Tieto premenné sa ukladajú do štruktúry *state* cez príkaz *getfield.* Ďalej sa zobrazí štandardné okno, kde si vyberieme miesto uloženia a napíšeme názov. Po uložení sa vytvorí súbor s koncovkou *.mat. Príkazom *uiputfile* sa vyvolá správa o úspešnom uložení. Funkcia uloženia vyzerá nasledovne:

```
function ulozButton_Callback ( hObject , eventdata , handles )

state .NNdata= getfield ( handles , 'NNdata ' )
state. cnt= getfield ( handles , ' cnt ' )
state. pocobj= getfield ( handles , ' pocobj ' )
state. menaObj= getfield ( handles , ' menaObj ' )

[ filename , pathname ] = uiputfile ({' *.mat ' }, 'Vyber objekty ' ) ;
save ( [ pathname , filename ] , ' state ' )
uiwait (msgbox ( ' Trénovacie dáta sú uložené ! ' , 'Dáta uložené ! ' ) ) ;

guidata ( hObject , handles ) ;
```

Druhou funkciou aplikácie je Načítať. Po stlačení tlačidla sa spúšťa funkcia nacitajBotton_Callback. Táto funkcia pracuje podobne ako predošlá, ale v opačnom smere. Čiže najskôr sa zobrazí okno, kde si vyberieme sadu, ktorú chceme načítať, potom sa pošle názov a cesta k súboru na vstup funkcie a tým sa načítajú uložené sady objektov. Opäť sa zobrazí aj hlásenie o úspešnosti načítania. Celá funkcia vyzerá nasledovne:

```
function nacitajButton_Callback ( hObject , eventdata , handles )

[ filename , pathname ] = uigetfile ({ ' *.mat ' } , 'Vyber objekty ' ) ;
load ( [ pathname , filename ] , ' state ' )
uiwait (msgbox ( ' Trénovacie dáta sú načítané ! ' , 'Dáta načítané ! ' ) ) ;

handles .NNdata=state .NNdata ;
```

```
handles . cnt=state . cnt ;
handles . pocobj=state.pocobj ;
handles . menaObj=state.menaObj ;

guidata ( hObject , handles ) ;
```

4.3 Ošetrenie chybových udalostí

Pri tvorbe programu som myslel aj na možné chybové udalosti. Tie môžu byť spôsobené nesprávnou manipuláciou s programom. Niektoré sú ošetrené tým, že nie sú sprístupnené tlačidlá v GUI, ak je zbytočné s nimi pracovať. Napríklad v GUI sú zakázané niektoré tlačidlá pokiaľ ich nie je potreba, čiže keď sú potrebné postupne sa odblokovávajú a je možné s nimi pracovať. No nie všetko je možné riešiť cez blokáciu, tak sú pripravené aj chybové hlásenia ako napríklad keď si spustíme rozpoznávací režim bez toho, aby sme mali natrénovanú neurónovú sieť.

4.4 Vytvorenie samostatne spustiteľnej aplikácie

V Matlabe je možnosť vytvoriť samostatne spustiteľnú aplikáciu, ktorá pobeží aj na PC kde nie je Matlab. To nám dáva možnosť vytvorenia ľahko ovládateľnej aplikácie, ktorá bude mať koncovku *.exe. O vytvorenie aplikácie sa stará Matlab Compiler. Pomocou príkazu *mcc* vytvoríme aplikáciu, no môžeme ju vytvoriť dvomi spôsobmi.

Prvý spôsob:

mcc -m main

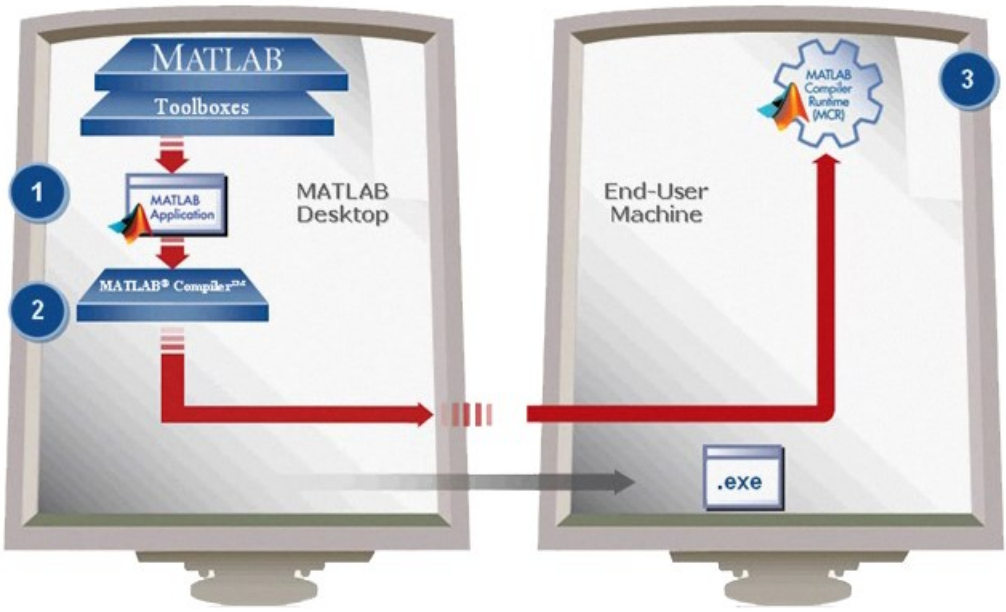
Druhý sposob:

mcc -m -e main

Parameter -m s príkazom mcc vytvorí samostatne spustiteľnú aplikáciu zo súboru main.m. Vytvorí ho aj s oknom, ktoré je podobné Command Window. Dané okno môže byť užitočné na ladenie aplikácie alebo pozorovanie chybových hlásení. Ak použijeme aj parameter -e, tak sa bude spúšťať len sama aplikácia bez spomínaného okna.

Vytvorenie tejto spustiteľnej aplikácie má aj nevýhody, ako napríklad nemožnosť vytvorenia a trénovania neurónovej siete. To znamená, že musíme mať už vytvorenú neu-

rónovú sieť a musí byť natrénovaná na určité objekty. Takže aplikácia sa bude dať používať len v režime rozpoznávania.



Obrázok č.23: Schéma tvorby samostatne spustiteľnej aplikácie [22]

ZÁVER

Cieľom tejto práce bolo navrhnúť vhodnú metodiku na rozpoznanie objektov v obraze prostredníctvom programu Matlab.

Teoretická časť bola rozdelené do troch kapitol. Prvá kapitola popisuje spracovanie obrazu, kde som ozrejmil pojmy ako snímanie, ditatalizácia, predspracovanie a segmentácia obrazu a niekoľko ďalších. Druha kapitola bola venovaná Neurónovým sieťam. V poslednej kapitole som sa zaoberal Programom Matlab. Tu som popísal jeho prostredie, základné súbory a najmä Toolboxy, ktoré som neskôr využíval v praktickej časti.

Praktickú časť som venoval samotnému návrhu aplikácie. Rozdelil som ju na štyri kapitoly. V prvej som opísal postup vytvorenia GUI, čo je grafické užívateľské rozhranie. Nasledujúca sa zaoberala funkciami aplikácie. Každá funkcia mala vlastnú podkapitolu, v ktorej som popísal ako daná funkcia pracuje. Taktiež som vypísal aj jednotlivé príkazy, ktorá som použil. Tretia kapitola sa zaoberala ošetrením chybových udalostí. Tu som sa snažil ozrejmiť prečo som blokoval niektoré funkcie aplikácie a aké iné ošetrenia pri chybách som použil. Posledná kapitola bola venovaná vytvoreniu samostatne spustiteľnej aplikácie. Kde som ozrejmil akými spôsobmi to je možné, ale zaoberal som sa aj nevýhodami ak vytvoríme takúto aplikáciu.

ZOZNAM POUŽITEJ LITERATURY

[1] HLAVÁČ, Václav a Miloš SEDLÁČEK. Zpracování signálů a obrazů. Vyd. 2.. Praha: ČVUT, 2005, 255 s. ISBN 80-010-3110-1.

[2] ŽÁRA, Jiří. a kol. Moderní počítačová grafika.. Brno: Computer Press, 1998, 448 s. ISBN 80-722-6049-9.

[3] JELÍNEK, I., SOCHOR, J.: Algoritmy počítačové grafiky. SVTS, Praha, 1989.

[4] SOCHOR, J., ŽÁRA, J., BENEŠ, B.: Algoritmy počítačové grafiky. ČVUT, Praha, 1998

[5] ŽÁRA, J. Moderní počítačová grafika. 2., přeprac. a rozš. vyd. Praha: Computer Press, 2004, s. 542-546. ISBN 80-251-0454-0.

[6] DOBEŠ, Michal. Zpracování obrazu a algoritmy v C#. Praha : BEN - technická literatura, 2008. 144 s. ISBN 978-80-7300-233-6.

[7] GONZALES, Rafael C., WOODS Richard E. Digital Image Proccesing. 2nd edition. Upper Saddle River, New Jersey 07458 : Prentice-Hall, 2002. 793 s. ISBN 0-20-118075-8

[8] ŠONKA, Milan, Václav HLAVÁČ a Roger BOYLE. Image processing, analysis, and machine vision. 3rd ed.. Toronto: Thomson, 2008, 829 s. ISBN 978-0-495-08252-1.

[9] HLAVÁČ, Václav a ŠONKA M. Počítačové vidění. Grada, Praha, 1993, ISBN 80–85424.

[10] Parker J. R. Algorithms for Image Processing and Computer Vision. Wiley Pub lishing, Indianapolis, 2011, ISBN 978–0–470–64385–3.

[11] VONDRÁK, CSC., prof. Ing. Ivo. *Neuronové sítě* [online]. Ostrava, 2009 [cit. 2017-04-19]. Dostupné z: http://vondrak.cs.vsb.cz/download/Neuronove_site.pdf

[12] CHUDÝ, Lukáš. *Umělá inteligence, díl 2. - neuronové sítě* [online]. 2005 [cit. 2017-05-12]. Dostupné z: http://programujte.com/clanek/2005061201-umela-inteligence-dil-2-neuronove-site/

[13] MAŘÍK, Vladimír. Umělá inteligence I.díl.. Praha: Academia, 1993, 264 s. ISBN 80-200-0496-3.

[14] ŠÍMA, Jiří a Neruda R. Teoretické otázky neuronových sítí. Vyd. 1.. Praha: MATFYZ press, 1996, 390 s. ISBN 80-858-6318-9.

[15] Kotek, Z., Mařík, V. a kol.: Metody rozpoznávání a jejich aplikace. Academia, Praha, 1993.

[16] ŘÍHA, Kamil. Pokročilé techniky zpracování obrazu. Brno : FEKT VUT v Brně, 2007. 109 s.

[17] ICT kompetence Počítačová grafika: Barevný model, DPI. [online]. 2009 [cit. 2017-04-17]. Dostupné z: http://www.kteiv.upol.cz/frvs/ictkubricky/?page=pocitacova-gra_ka/barevny-model-dpi.

[18] KARBAN, Pavel. Výpočty a simulace v programech Matlab a Simulink. 1. vyd.. Brno: Computer Press, 2006. 220 s. ISBN 80-251-1301-9.

[19] HUMUSOFT [online]. Matlab: Jazyk pro technické výpočty. c 1991-2012 [cit. 2017-04-21]. Dostupné z: http://www.humusoft.cz/produkty/matlab/matlab/.

[20] Matlab. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wiki media Foundation, 2001- [cit. 2017-05-21]. Dostupné z: htt ps://en.wikipedia.org/wiki/MATLAB#/media/File:MATLAB_R2013a_Win8_scree nshot.png

[21] ZAPLATÍLEK, Karel a Bohuslav DOŇAR. MATLAB: tvorba uživatelských aplikací. 1. vyd.. Praha: BEN, 2004. 215 s. ISBN 80-730-0133-0.

[22] *MathWorks* [online]. Colesville, 2017 [cit. 2017-04-21]. Dostupné z: http://www.mathworks.com/

[23] TUČKOVÁ, Jana, Marek BÁRTŮ a Petr ZETOCHA. Aplikace umělých neuronových sítí při zpracování signálů. 1. vyd.. Praha: ÈVUT, 2009. 140 s. ISBN 978-80-01-04400-1.

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

apod	a podobne
GUI	Graphical User Interface
PC	Personal Computer
α	Alfa
ϵ	leží
Υ	Gama
$\geq$	Väčšie alebo rovné

ZOZNAM OBRÁZKOV