SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

NEURONSKE MREŽE U OPENCV BIBLIOTECI

Završni rad

Bože Eugen Marković

Osijek, 2016.

SADRŽAJ

1. UVOD

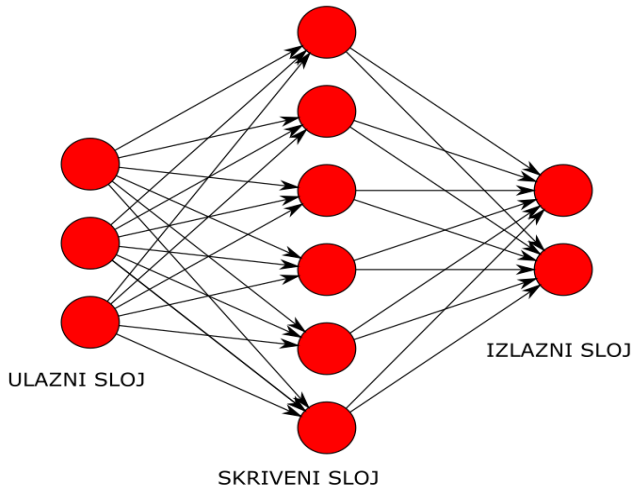
Ovim završnim radom prikazati će se teorijska obrada računalnih neuronskih mreža i neke od značajki i mogućnosti koje se vežu uz njih. Neuronske mreže već se uvelike koriste u praksi. Princip njihovog rada po mnogim je karakteristikama različit od standardnog sustava za digitalnu obradu podataka. Jedina sličnost je da imaju ulazne i izlazne podatke – sve ostalo je različito. To im daje mogućnost da budu primijenjene u mnogim područjima u kojima nije moguće primijeniti klasično računalo. Njihova osnovna osobina, da probleme rješavaju empirijski, a ne algoritamski, otvara mogućnost primjene u domenama koje zahtijevaju previše složene algoritme ili se čak i ne mogu riješiti algoritamski. Jedno takvo veliko područje nosi općeniti naziv klasifikacija podataka, što nam ne govori mnogo dok ne kažemo da je to prepoznavanje oblika (slova, ljudskih lica ili bilo kojih predmeta), generiranje i prepoznavanje ljudskog govora ili drugih zvukova, kao i predviđanja događaja u meteorologiji, sportu... Jedan od alata koji se koristi za računalnu interpretaciju neuronskih mreža jest OpenCV biblioteka u C++ programskom jeziku. U razradi ove teme prikazati će se implementacija neuronske mreže u OpenCV okruženju.

1.1. Zadatak završnog rada

Potrebno je teorijski proučiti karakteristike neuronskih mreža i mogućnosti kod OpenCV biblioteke. Neuronsku mrežu potrebno je implementirati u OpenCV biblioteci u programskom C++ okruženju te demonstrirati njene operacije.

2. NEURONSKE MREŽE

Prije svega, kada se govori o neuronskim mrežama, treba reći „umjetna neuronska mreža“ s obzirom da se radi o računalnim sustavima čija je ideja rada posuđena iz prirode. Osnovna ideja umjetnih neuronskih mreža jest oponašanje biološke neuronske mreže kako bi se riješio zadani problem. Ne postoji jedinstvena definicija neuronske mreže, no može ju se opisati kao zbir čvorova (procesora, neurona) gdje svaki od njih ima lokalnu memoriju u kojoj pamti podatke koje obrađuje. Neuroni su povezani komunikacijskim kanalima (vezama). Podaci koji se ovim kanalima razmjenjuju obično su numerički. Može se dodati da su neuronske mreže sustavi s naglašenim paralelizmom te čvorovima raspoređenim u više slojeva. [1]

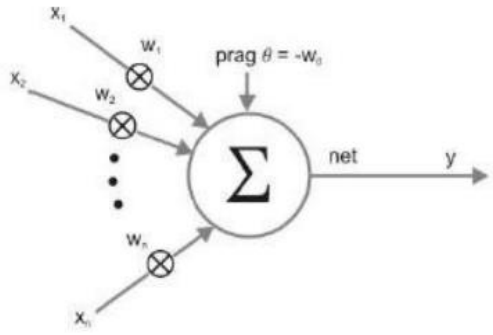


Sl. 2.1. Primjer modela neuronske mreže

Neuronsku mrežu tvore međusobno povezani čvorovi. Neuroni su raspoređeni u dva ili više slojeva. Prvi sloj uvijek je ulazni, a posljednji izlazni. Ako ih ima više, onda se unutarnji slojevi zovu skriveni slojevi. Način organizacije ovih slojeva i njihovog međusobnog povezivanja čini arhitekturu ili topologiju neuronske mreže.

Na slici 2.1. prikazan je shematski izgled neuronske mreže sa tri ulaza i dva izlaza. Sastoji se od tri sloja, a u skrivenom sloju ima šest neurona. Ovo je takozvana nepovratna (*feed forward*) mreža jer nema povratne veze – tok podataka ide samo u jednom smjeru, ali postoje i povratne (*feedback*) mreže. Isto tako, mreže mogu imati i veći broj slojeva, tako da neke slojeve čini veliki broj neurona, a neke samo jedan. Treba imati u vidu kako je ovo pojednostavljena hipotetska mreža i da je u nekim slučajevima ona daleko složenija. Na primjer, da ima onoliko ulaza koliko piksela ima neka slika i onoliko izlaza koliko različitih motiva treba prepoznati. [2]

U svakom čvoru obavlja se neka računska operacija, a svaka veza između čvorova prenosi signal. Jedan čvor može imati više ulaza i jedinstveni izlaz koji može slati signal na ulaze više drugih čvorova. Model jednog čvora prikazan je na slici 2.2.



Sl. 2.2. Model čvora u neuronskoj mreži [3]

U prikazu modela umjetnog neurona signali su opisani numeričkim iznosom i na ulazu u neuron množe se težinskim faktorom. Tako dobiveni signali nakon toga se zbrajaju i ako je dobiveni iznos iznad definiranog praga neuron daje izlazni signal. [3] Ulazni signali označeni su s $x_1, x_2, ..., x_n$, težinski faktori s $w_1, w_2, ..., w_n$, a prag s w_0 prema formuli (2-1).

$$y = \sum_{i}^{n} w_i x_i - w_0 \tag{2-1}$$

3. OPENCV BIBLIOTEKA

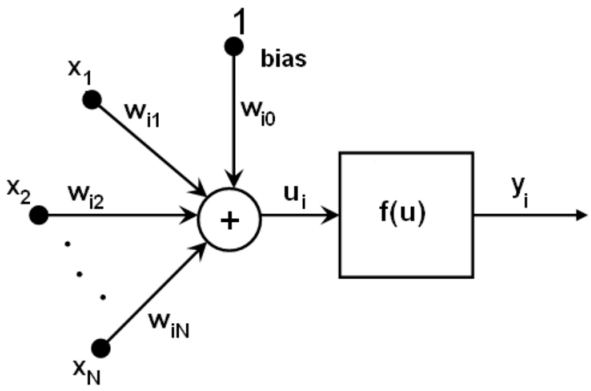
OpenCV (engl. *Open Source Computer Vision Library*) računalna je biblioteka otvorenog izvornog koda koja obuhvaća koncepte računalnog vida (*Computer Vision*) i strojnog učenja *(Machine Learning*). Biblioteka sadrži više od 2500 optimiziranih algoritama. [4]

Računalni vid je područje pod koje spadaju metode za stjecanje, obradu, analiziranje i razumijevanje slike i, općenito, višedimenzionalnih podataka iz realnog svijeta s ciljem dobivanja numeričkih ili simboličkih informacija. Ovo područje razvijeno je sa željom da se ostvari mogućnost percepcije ljudskog vida u elektroničkom obliku i razumijevanja slike u digitalnom smislu. Razumijevanje fotografija može biti viđeno kao odvajanje simboličke informacije iz slikovnih podataka koristeći modele napravljene uz pomoć, između ostalog, i neuronskih mreža. Obrada i analiza slike se prvenstveno usmjerava na proces proizvodnje slike. Primjerice, medicinske slike zahtijevaju značajnu analizu u svrhu dobivanja korisnih informacija. Prepoznavanje uzoraka koristi razne metode za izdvajanje informacija iz signala u cjelini, uglavnom na temelju statističkih pristupa. Značajan dio ovog područja posvećen je primjeni metoda na slikovne podatke. Neke od svrha računalnog vida su prepoznavanje objekata u prostoru, lica ili znakova. [5]

OpenCV je objavljen pod BSD licencom i stoga je besplatan i za akademsku i komercijalnu uporabu. Ima C ++, C, Matlab, Python i Java sučelja i podržava Windows, Linux, Mac OS, iOS i Android. OpenCV je dizajniran za računalnu efikasnost s jakim naglaskom na „*real-time*“ aplikacije. Pisana je u optimiziranom C/C ++ kodu, te biblioteka može iskoristiti višejezgrenu obradu. OpenCV je prihvaćen diljem svijeta, ima više od 47 tisuća korisnika i procijenjeni broj preuzimanja iznad 9 milijuna. Smanjenjem cijena i dostupnošću kamera visoke kvalitete na svakom mobilnom uređaju javlja se potražnja za novim i naprednijim mogućnostima te značajno raste broj aplikacija koje koriste OpenCV. [4]

4.IMPLEMENTACIJA NEURONSKE MREŽE U OPENCV

Višeslojni perceptroni (engl. MLP - *multi-layer perceptrons*) implementiraju se kao mreža bez povratne veze. To je najčešće korištena vrsta neuronske mreže. Svi neuroni su slični. Svaki ima nekoliko ulaznih veza kojima uzima izlazne vrijednosti od nekoliko neurona prethodnog sloja te također ima nekoliko izlaznih veza kojima prosljeđuje signal neuronima u sljedećem sloju. [7] Vrijednosti se (svaka individualno) na ulazu u neuron množe sa svojim težinskim faktorima i dodaje se „bias term“ u slučaju da su svi ulazni signali jednaki nuli, a očekivani izlaz je različit od nule. [6] Dobiveni zbroj ulaznih signala transformira se korištenjem aktivacijske funkcije.



Sl. 4.1. Model neurona

Ako su x_j izlazi sloja n, a y_i izlazi sloja $n+1$ dobiva se:

$$u_i = \sum_j (w_{i,j}^{n+1} * x_j) + w_{i,bias}^{n+1} \quad [7] \tag{4-1}$$

$$y_i = f(u_i) \tag{4-2}$$

U programskom C++ okruženju koriste se moduli, klase i funkcije iz OpenCV biblioteke. Za početak, kreira se model neuronske mreže. To se postiže instancom:

ANN_MLP::create().

Na ovaj su način zadane vrijednosti svih težinskih faktora mreže postavljene na nulu. Zatim je potrebno definirati slojeve neuronske mreže funkcijom:

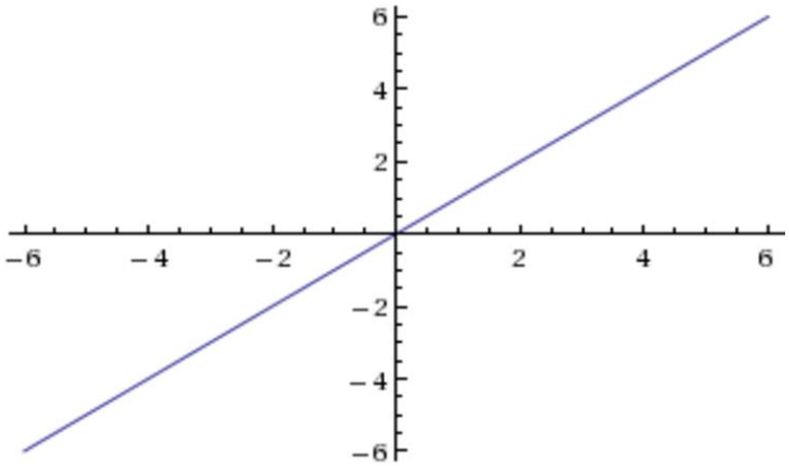
setLayerSizes().

Funkcija prima vektor cijelih brojeva veličine jednake broju slojeva u mreži. Vektor specificira broj neurona u svakom sloju na način da je na nultom indeksu vektora broj čvorova ulaznog sloja, na zadnjem indeksu broj čvorova izlaznog sloja i između je specifikacija elemenata skrivenih slojeva ako postoje. Nakon definiranja slojeva određuje se aktivacijska funkcija koja transformira zbroj signala koji ulaze u određeni čvor te određuje konačni izlaz promatranog čvora. To se postiže sljedećom funkcijom:

setActivationFunction().

Ovom funkcijom specificiraju se tip aktivacijske funkcije i parametri α i β koji ju određuju. U upotrebi su različite aktivacijske funkcije. Višeslojne mreže koriste tri standardne:

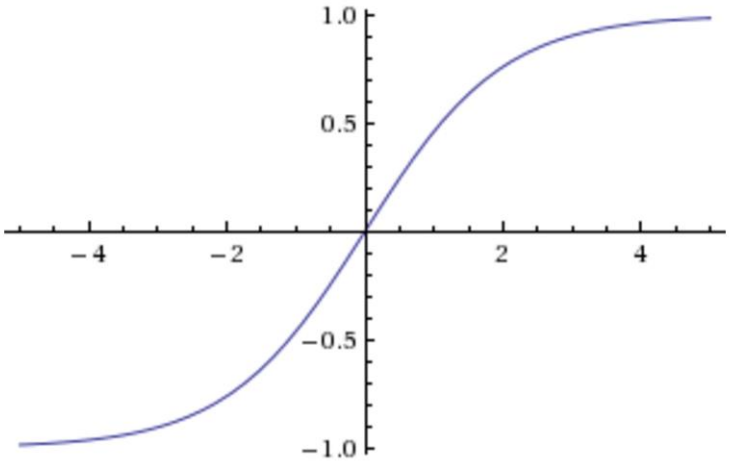
- funkcija identiteta (ANN_MLP::IDENTITY): $f(x) = x$



Sl. 4.2. Funkcija identiteta

- simetrična sigmoidalna funkcija

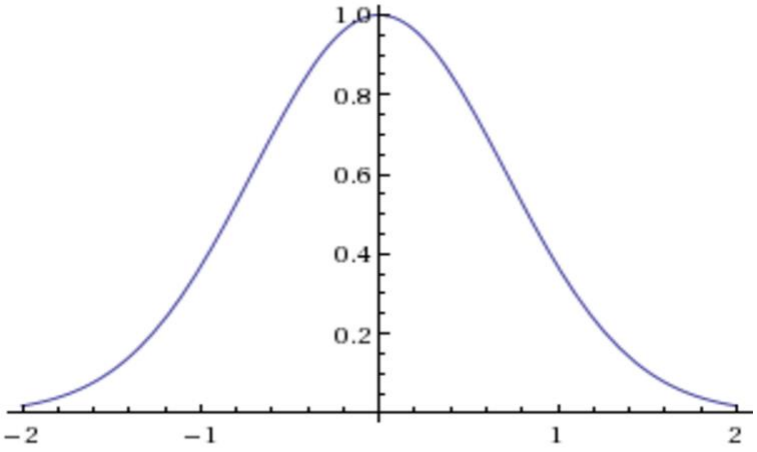
  (ANN_MLP::SIGMOID_SYM): $f(x) = \beta * (1 - e^{-\alpha x}) / (1 + e^{-\alpha x})$



Sl. 4.3. Simetrična sigmoidalna funkcija

Na slici 4.3. prikazana je simetrična sigmoidalna funkcija za $\beta = 1, \alpha = 1$.

- Gaussova funkcija (ANN_MLP::GAUSSIAN): $f(x) = \beta e^{-\alpha x * x}$



Sl. 4.4. Gaussova krivulja

Na slici 4.4. prikazana je Gaussova funkcija za $\beta = 1, \alpha = 1$.

Svi neuroni imaju iste aktivacijske funkcije sa istim parametrima (α, β). Parametre određuje korisnik i oni se ne mijenjaju u postupku treniranja (učenja). Kada su postavljeni slojevi i aktivacijska funkcija treba odrediti algoritam učenja mreže.

Postupak učenja funkcionira prema sljedećim koracima:

1. Uzima se početni ulazni vektor. Veličina vektora jednaka je veličini ulaznog sloja.
2. Ulazni sloj prosljeđuje vrijednosti prvom skrivenom sloju.
3. Računaju se izlazne vrijednosti skrivenog sloja koristeći težinske faktore i aktivacijske funkcije.
4. Vrijednosti se prosljeđuju kroz slojeve sve dok se ne dobije izlazna vrijednost na izlaznom sloju.

Pa prema tome, da bi se mreža mogla „trenirati“ potrebno je znati sve težinske faktore ($w_{i,j}^{n+1}$). Težinski faktori računaju se algoritmom za učenje. Algoritam uzima nekoliko ulaznih vektora i isti broj izlaznih vektora i iterativnim metodama prilagođava težinske faktore kako bi u pokušajima nakon procesa učenja neuronska mreža za odgovarajuće ulazne veličine dala zadovoljavajuće izlazne veličine. [8] Za postupak učenja mreže koriste se „*backpropagation*“ algoritam i RPROP93 algoritam. Algoritam učenja postavlja se funkcijom:

setTrainMethod().

Funkcija prima samo naziv metode za učenje koja se koristi. Međutim, nije dovoljno samo odrediti algoritam učenja već je potrebno odrediti i parametre algoritma. *Backpropagation* algoritam koji se najčešće koristi za treniranje mreža ima 2 parametra koji se definiraju funkcijama:

setBackpropMomentumScale(),

setBackpropWeightScale().

Prvi parametar pamti moment, odnosno razliku između iznosa težinskog faktora dvije prethodne iteracije, dok drugi parametar određuje gradijent težinskih faktora koji se izračunava kod svakog ponavljanja propagacije.

Algoritam širi signal kroz mrežu te ažurira težinske faktore. Signal se prvo širi od ulaza prema izlazu kako bi se dobile izlazne vrijednosti. Zatim se impuls širi nazad od izlaza te se računa razlika između dobivenih i očekivanih vrijednosti.

Opći algoritam za učenje [9] prikazan na mreži s tri sloja:

```
Inicijalizacija težinskih faktora mreže (često male nasumične vrijednosti)
  radi
     zaSvaki  uzorak za treniranje  imena  uzorak
        očekivani = očekivani izlaz(mreža, uzorak)
        dobiveni = dobiveni_izlaz(uzorak)
        izračunaj pogrešku (očekivani - dobiveni) izlaznih jedinica

        izračunaj      sve težinske faktore od skrivenog sloja do izlaznog
                       sloja

        izračunaj      sve težinske faktore od ulaznog sloja do skrivenog
                       sloja
        ažuriraj  težinske  faktore  // ulazni  sloj  se  ne  ažurira  dobivenom
pogreškom
  sveDok  svi  uzorci  pravilno  klasificirani  ili  je  neki  drugi  kriterij
zaustavljanja zadovoljen
  vrati mreža
```

Prije samog postupka treniranja mreže potrebno je još samo navesti koliko se dugo mreža treba trenirati, odnosno s koliko iteracija. Za to se koristi funkcija:

setTermCriteria().

Na ovaj način postavlja se maksimalni broj iteracija algoritma za treniranje i uz to se određuje kriterij prekida postupka učenja i u slučaju da je promjena u vrijednosti težinskih faktora jako mala što bi označavalo da je mreža istrenirana. Kada su svi ovi parametri postavljeni može se pokrenuti proces treniranja mreže. Treniranje odrađuje funkcija:

train()

koja prima matricu uzoraka pomoću kojih se mreža trenira, matricu očekivanih vrijednosti koje bi se trebale dobiti na izlazu iz mreže i tip uzoraka, odnosno način na koji su oni spremljeni. Tip uzoraka može biti ROW_SAMPLE i COL_SAMPLE. ROW_SAMPLE označava da svaki red trening matrice predstavlja jedan uzorak, a COL_SAMPLE da svaki uzorak zauzima jedan stupac matrice.

Uspješnost procesa treniranja može se provjeriti funkcijom:

isTrained()

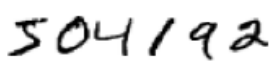
koja je tipa bool i vraća vrijednost 1 ako je model uspješno treniran, u suprotnom vraća vrijednost 0. Kada je mreža istrenirana mogu joj se predavati testni uzorci za provjeru radi li mreža i na kojoj razini točnosti. Predviđanje izlaza testnog uzorka radi funkcija:

predict().

Funkcija prima jedan testni uzorak i prazan vektor. Vektor je veličine jednake broju izlaznih neurona mreže pa ova funkcija u taj vektor zapisuje vjerojatnost za svaki mogući izlaz. U praksi se kao predviđani izlaz uzima onaj sa najvećom vjerojatnošću. Ako se testira više uzoraka funkcija predict() stavlja se u petlju te se svi dobiveni rezultati testiranja uspoređuju sa očekivanima i na temelju toga se izračunava postotak točnosti. [10]

5.PRIMJENA NEURONSKIH MREŽA NA PROBLEM PREPOZNAVANJA RUKOPISA

Primjena neuronskih mreža je široka no najviše se upotrebljava za kategorizaciju podataka u raznim područjima. Na primjer, primjena u prepoznavanju ljudskog rukopisa.



Sl. 5.1. Ljudski rukopis

Većina ljudi bi znamenke na slici pročitala sasvim nesvjesno načina na koji ih je prepoznala, međutim ljudski mozak sastoji se od milijuna neurona i znamenke prepoznaje prema iskustvima koje je imao. Na sličan način funkcionira i neuronska mreža nakon procesa treniranja. Problem se javlja kada je potrebno napisati računalni program koji bi prepoznao simbole napisane ljudskim rukopisom. Ako se analizira tekst sa više simbola računalu je problematično razlučiti gdje je granica između dvaju znakova (problem segmentacije), tog problema nema ako se za analizu uzima samo jedan znak.

Ulazni neuroni sadrže informacije o pikselima slike prema skali sive boje (engl. *greyscale)*. To znači da su im vrijednosti određene prema intenzitetu boje na pikselu kojeg predstavljaju. Tako će neuron koji predstavlja potpuno crni piksel imati vrijednost 1, a potpuno bijeli vrijednost 0. Sivi pikseli ovisno o intenzitetu imat će vrijednost u intervalu [0,1].

Za klasifikaciju pojedinih znamenki može se koristiti troslojna mreža kojoj bi broj ulaza bio jednak broju piksela na slici na kojoj je rukopis. Broj izlaza, odnosno neurona na izlaznom sloju bio bi jednak broju svih simbola koji se mogu nalaziti na slici. Broj neurona u skrivenom sloju za početak je nasumičan pa se kasnije eksperimentalno određuje onaj koji je najoptimalniji. Isto tako broj slojeva može se povećati u slučaju da će na taj način algoritam postati efikasniji. [11]

Za praktični dio ovog završnog rada korištena je biblioteka OpenCV verzije 3.0.0. implementirana u programu Microsoft Visual Studio 2012. Za implementaciju OpenCV biblioteke prvo ju je potrebno besplatno preuzeti sa njihove službene stranice te kreirati *Environment Variable* imena „OPENCV_DIR“ u naprednim postavkama sustava na osobnom računalu. Također, u svojstvima svakog projekta u kojem se žele koristiti OpenCV naredbe potrebno je navesti dodatne „*Include*“ direktorije i dodatne „*library* “ direktorije.

Svrha programa je raspoznavanje ljudskog rukopisa te je u programu obrađeno prepoznavanje pojedinačnih znamenaka (0-9). Ideja programa je da uzima dvije CSV(engl. *comma separated values*) datoteke određenih veličina, jednu sa uzorcima za treniranje mreže i drugu sa uzorcima za testiranje mreže te da sa danim podacima trenira i testira mrežu. CSV datoteke su preuzete sa MNIST [12] baze podataka u kojoj se nalaze datoteke sa različitim količinama uzoraka. Jedan uzorak predstavlja jednu znamenku napisanu ljudskim rukopisom. Svaki uzorak je niz od 784 cjelobrojne vrijednosti u intervalu [0,255], gdje svaka vrijednost predstavlja nijansu prema skali crno-bijele boje. Svaka od 784 vrijednosti predstavlja piksel slike veličine 28x28. Prema tome, ulazni sloj neuronske mreže imat će 784 čvora tako da svaki čvor prima jedan od atributa uzorka, odnosno jedan piksel, a izlazni sloj imat će 10 čvorova jer je na izlazu moguće dobiti 10 različitih znamenaka.

Sve OpenCV klase i funkcije smještene su u CV imeniku pa ga je nužno navesti na početku programa kako bi se pristupilo njegovim funkcionalnostima:

```
using namespace cv;
```

Prije glavne i ostalih funkcija definirane su globalne varijable koje opisuju vrijednosti koje se uzimaju iz CSV datoteka, a to su: broj uzoraka u CSV trening datoteci, broj atributa u svakom uzorku u obje CSV datoteke, broj uzoraka u testnoj CSV datoteci i broj mogućih izlaza:

```
#define BROJ_UZORAKA_ZA_UCENJE 1000
#define BR_VRIJEDNOSTI_U_UZORKU 784
#define BROJ_TEST_PRIMJERA 100
#define BR_KLASA 10
```

Definicija matrica u koje će se spremati uzorci za treniranje i očekivane vrijednosti:

```
Mat_<float> trening_uzorci(BROJ_UZORAKA_ZA_UCENJE, BR_VRIJEDNOSTI_U_UZORKU);
Mat trening_klase=Mat::zeros(BROJ_UZORAKA_ZA_UCENJE,BR_KLASA, trening_uzorci.type());
```

Isto je odrađeno i za testne uzorke.

Pozivaju se funkcije u kojima se odrađuje učitavanje vrijednosti iz CSV datoteka u navedene matrice. Dvije su funkcije, za učitavanje testnih i trening uzoraka. Prva vrijednost u svakom uzorku učitavanom iz datoteke očekivana je vrijednost, odnosno znamenka koja je zapisana rukopisom pa se ona sprema u posebnu matricu:

```
if (stupac ==0)
{
	fscanf(f, "%f,", &labela);
	klase.at<float>(red, labela) = 1.0
}
```

Ostali atributi uzorka jednostavno se učitavaju iz datoteke.

Slijedi definiranje vektora slojeva. Vektor ima tri vrijednosti, jedna za broj čvorova u svakom sloju:

```
int slojevi_d[] = { BR_VRIJEDNOSTI_U_UZORKU, 10,  BR_KLASA};
```

Kreiranje modela neuronske mreže imena „mreza“:

```
Ptr< ANN_MLP >  mreza = ANN_MLP::create();
```

Dodavanje definiranih slojeva mreži:

```
mreza->setLayerSizes(slojevi);
```

Određivanje simetrične sigmoidalne aktivacijske funkcije sa parametrima $\alpha = 0.6, \beta = 1$:

```
mreza->setActivationFunction(ml::ANN_MLP::SIGMOID_SYM, 0.6, 1);
```

Prilikom pisanja koda bitno je prvo definirati slojeve mreže pa tek onda aktivacijsku funkciju jer u suprotnom neuronska mreža se neće pravilno kreirati. Nadalje, potrebno je definirati algoritam koji se koristi za učenje neuronske mreže, odabran je *Backpropagation* algoritam te su specificirana dva potrebna parametra za izračun težinskih faktora. Preporučeno je da se za obje početne vrijednosti postavi 0.1.

```
mreza->setTrainMethod(cv::ml::ANN_MLP::BACKPROP);
```

```
mreza->setBackpropMomentumScale(0.1);
mreza->setBackpropWeightScale(0.1);
```

Nakon definiranja metode treniranja, navodi se kriterij za prekid procesa treniranja. Učenje neuronske mreže prestaje na 2000 iteracija ili prije ako se dogodi drugi uvjet da je promjena u vrijednosti težinskih faktora manja od 0.000001:

```
mreza->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER+TermCriteria::EPS,2000,
0.000001));
```

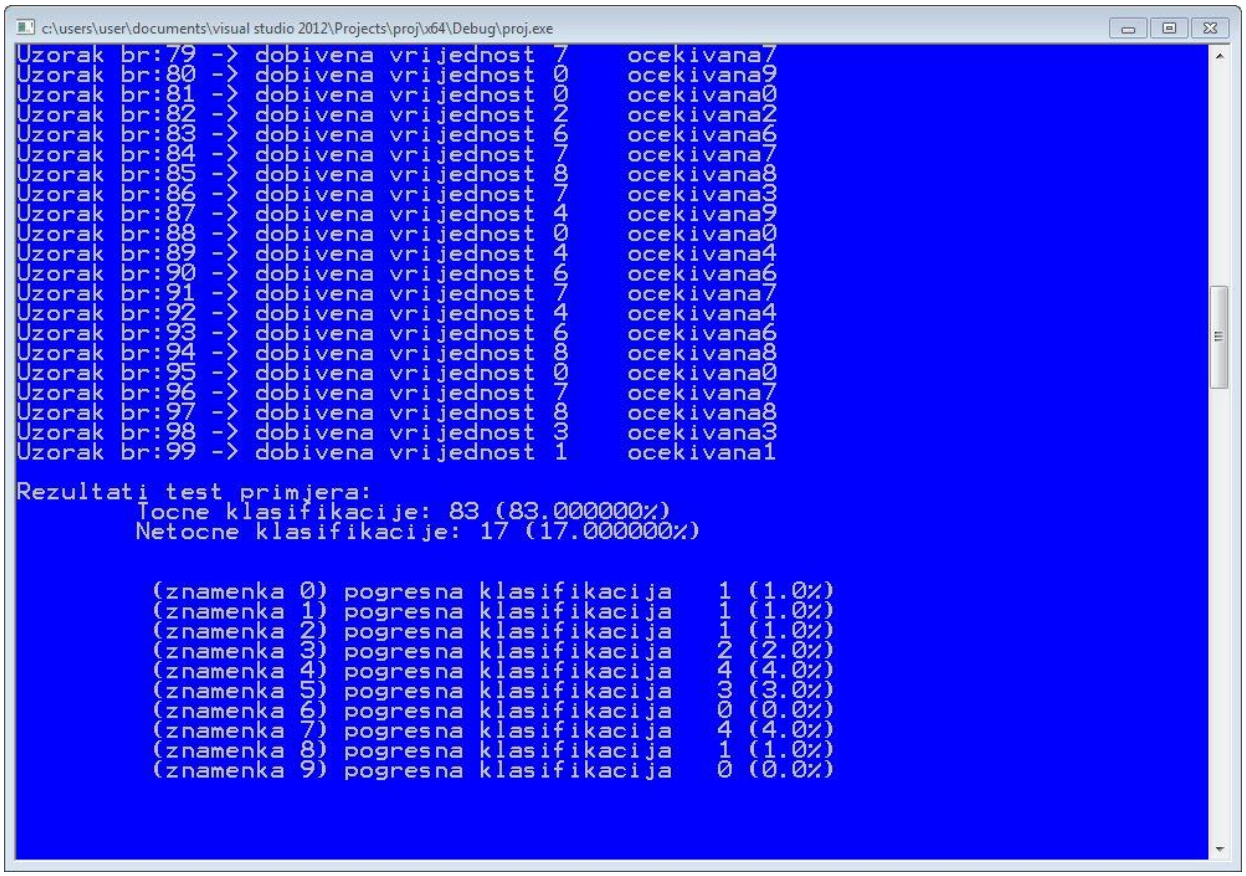
Pokreće se treniranje mreže funkcijom `train()` kojoj se predaje matrica sa uzorcima u kojoj je svaki redak jedan uzorak i matrica očekivanih vrijednosti na izlazu za svaki uzorak. Nakon toga funkcijom `isTrained()` provjerava se jeli proces učenja uspješno završen. Ako je ispisuje se poruka. Treniranje mreže može potrajati, ovisno o broju uzoraka. Za simulaciju u ovom programu korištena je CSV datoteka s 1000 trening uzoraka.

```
mreza->train(trening_uzorci, ROW_SAMPLE, trening_klase);
if (mreza->isTrained())
	cout << "Trening uspjesno obavljen\n\n";
```

Sada kad je obavljeno učenje neuronske mreže mogu joj se predati test primjeri i provjeriti točnost. Uzima se uzorak po uzorak, odnosno redak po redak iz matrice testnih primjera, provlači se kroz neuronsku mrežu i na izlazom sloju dobivaju se vrijednosti na 10 izlaznih čvorova, od kojih svaki predstavlja jednu od mogućih znamenaka. Dobivene vrijednosti zapisuju se u vektor vjerojatnosti i najveća vrijednost iz vektora uzima se kao „pobjednik“ te znamenka koju predstavlja ta vjerojatnost uzima se kao konačni izlaz za razmatrani uzorak:

```
mreza->predict(redak, test_rezultat);
minMaxLoc(test_rezultat, 0, 0, 0, &max_loc);
```

U simulaciji je neuronska mreža testirana sa 100 uzoraka. Za svaki test primjer ispisuje se dobivena i očekivana vrijednost. Prebrojavaju se sve točne i netočne klasifikacije i uz to netočne klasifikacije za pojedinu znamenku te se rezultat na kraju ispisuje na ekranu prema slici 5.2.



Sl. 5.2. Rezultati testiranja mreže

6. ZAKLJUČAK

Ljudsko oko, odnosno ljudski mozak refleksno raspoznaje znakove, bile one tiskane ili napisane nečijim rukopisom, ali računalu to nije jednostavno. U ovom radu pokazano je kako je to moguće uspješno izvesti. Na primjeru učenja neuronske mreže u C++ programskom okruženju s implementiranom OpenCV bibliotekom dobivena je točnost od 83% u prepoznavanju znamenaka napisanih ljudskim rukopisom. Prepoznavanje se izvršavalo uz pomoć neuronske mreže koja se istrenirala sa 1000 uzoraka i nakon toga je testirana sa 100 test primjera. Rezultati bi bili točniji da se učenje neuronske mreže provodilo sa više od 1000 uzoraka, koliko je korišteno u simulaciji. Napisani program može se proširiti u nekoliko smjerova. Jedan od načina jest da program uzima sliku, a ne CSV datoteku te da sam razdvaja piksele i procjenjuje o kojoj se znamenki radi. Na isti način, program bi mogao uzimati niz simbola, razlučiti granicu između njih i prepoznavati jedan po jedan. Uz to može se proširiti opseg simbola koji se promatraju pa osim znamenaka učiti sustav da prepoznaje i slova.

7.PRILOZI

Programski kod:

```
#include<opencv2\opencv.hpp>
#include<opencv\cv.hpp>
#include<opencv\ml.h>
#include <opencv2/ml/ml.hpp>
#include <iostream>
#include <stdio.h>
#include <fstream>
#include<opencv2/core/traits.hpp>

using namespace cv;
using namespace ml;
using namespace std;

#define BROJ_UZORAKA_ZA_UCENJE 1000
#define BR_VRIJEDNOSTI_U_UZORKU 784
#define BROJ_TEST_PRIMJERA 100
#define BR_KLASA 10


                    // funkcija za citanje iz datoteke sa uzorcima za treniranje mreze
int ucitaj_datoteku( const char* train1000 , Mat uzorci, Mat klase, int br_uzoraka)
{
    float labela;
    float t;

       FILE* f = fopen( "train1000.csv" , "r" ); // dohvaćanje datoteke
    if( !f )
    {
              cout<<"nemoguce pronaci datoteku"<<endl; // u slučaju da je nemoguće
              return 0;            // pronaći datoteku - ispis poruke i prekid funkcije
    }

    for(int red = 0; red < br_uzoraka; red++)
    {

        for(int stupac = 0; stupac < (BR_VRIJEDNOSTI_U_UZORKU + 1); stupac++)
        {
            if (stupac ==0)   // prva vrijednost u uzorku je labela(očekivana vrijednost)
            {
                    fscanf(f, "%f,", &labela); //učitavanje vrijednosti iz datoteke u u
                                                    //varijablu "labela"
                    klase.at<float>(red, labela) = 1.0; // učitavanje u matricu "klase"
                                                      //na odgovarajuci indeks
            }
            else if (stupac <= BR_VRIJEDNOSTI_U_UZORKU ) // ostale vrijednosti u JEDNOM
                                                  //uzorku predstavljaju piksele JEDNE slike
            {
                    fscanf(f, "%f,", &t); // učitavanje vrijednosti iz datoteke u
                                                 //"temporary" varijablu
                    uzorci.at<float>(red, stupac-1) = t;  //učitavanje u matricu
                                                // "uzorci", svaki zorak u jedan redak matrice
                    }
        }
    }

    fclose(f); // zatvaranje otvorene datoteke
```

```
    return 1; // funkcija vraća vrijednost 1
}


            // funkcija za citanje iz datoteke sa uzorcima za testiranje mreze
int ucitaj_datoteku1( const char* test100 , Mat uzorci, Mat klase, int br_uzoraka, Mat
ocekivani)
{
    float labela;
    float t;

       FILE* f = fopen( "test100.csv" , "r" );  // dohvaćanje datoteke
    if( !f )
    {
            cout<<"nemoguce pronaci datoteku"<<endl;   // u slučaju da je nemoguće ,
            return 0;           //pronaći datoteku - ispis poruke i prekid funkcije

    }

       for(int red = 0; red < br_uzoraka; red++)
    {

        for(int stupac = 0; stupac < (BR_VRIJEDNOSTI_U_UZORKU + 1); stupac++)
        {
            if (stupac ==0) // prva vrijednost u uzorku je labela(očekivana vrijednost)
            {
                fscanf(f, "%f,", &labela);     //učitavanje vrijednosti iz datoteke u
                                               //u varijablu "labela"
                klase.at<float>(red, labela) = 1.0;   // učitavanje u matricu "klase"
                                                      //na odgovarajuci indeks
                ocekivani.at<float>(0,red) = labela;     // učitavanje svih labela u
                                                         //vektor "ocekivani"
            }
            else if (stupac <= BR_VRIJEDNOSTI_U_UZORKU)  // ostale vrijednosti u JEDNOM
                                                 //uzorku predstavljaju piksele JEDNE slike
            {
                fscanf(f, "%f,", &t); // učitavanje vrijednosti iz datoteke u
                                      // "temporary" varijablu
                uzorci.at<float>(red, stupac-1) = t; //učitavanje u matricu "uzorci",
                                                 //svaki uzorak u jedan redak matrice
                }
        }
    }
    fclose(f);  // zatvaranje otvorene datoteke

    return 1;  // funkcija vraća vrijednost 1
}
//*****************************************************************************/

int main( int argc, char** argv )

{
Mat_<float> trening_uzorci(BROJ_UZORAKA_ZA_UCENJE, BR_VRIJEDNOSTI_U_UZORKU);
                          // kreiranje matrice u koju se učitavaju trening uzorci
Mat trening_klase = Mat::zeros(BROJ_UZORAKA_ZA_UCENJE, BR_KLASA, trening_uzorci.type());
                                //matrica klasa(potrebno za treniranje mreže)

Mat ocekivani = Mat::zeros(1, BROJ_TEST_PRIMJERA, CV_32FC1); //vektor očekivanih izlaza

Mat test_primjeri = Mat(BROJ_TEST_PRIMJERA, BR_VRIJEDNOSTI_U_UZORKU,
trening_uzorci.type());     //kreiranje matrice u koju se učitavaju test uzorci
```

```
Mat test_klase = Mat::zeros(BROJ_TEST_PRIMJERA, BR_KLASA, trening_uzorci.type());
                                                                    //matrica test klasa

      ucitaj_datoteku( argv[1], trening_uzorci, trening_klase, BROJ_UZORAKA_ZA_UCENJE) ;
                                    //poziv funkcije za učitavanje trening uzoraka
      ucitaj_datoteku1(argv[2], test_primjeri, test_klase,
BROJ_TEST_PRIMJERA,ocekivani);           //poziv funkcije za učitavanje test uzoraka


       int slojevi_d[] = { BR_VRIJEDNOSTI_U_UZORKU, 10,  BR_KLASA};// deiniranje slojeva

       Mat slojevi = Mat(1,3,CV_32SC1);
       slojevi.at<int>(0,0) = slojevi_d[0];//broj ulaznih čvorova jednak broju
                                          //vrijednosti u uzorku
       slojevi.at<int>(0,1) = slojevi_d[1];          // -10 čvorova u skrivenom sloju
       slojevi.at<int>(0,2) = slojevi_d[2]; // -10 izlaznih čvorova prema 10
                                         //mogućih izlaznih vrijednosti
      Ptr< ANN_MLP >  mreza = ANN_MLP::create();      // kreiranje neuronske mreže

      mreza->setLayerSizes(slojevi);  //postavljanje slojeva
      mreza->setActivationFunction(ml::ANN_MLP::SIGMOID_SYM, 0.6, 1);
            //postavljanje sigmoidne aktivacijske funkcije i parametara alfa i beta


      mreza->setTrainMethod(cv::ml::ANN_MLP::BACKPROP); //postavljanje "backpropagation"
                                                        //algoritma za trening
      mreza->setBackpropMomentumScale(0.1);      // i parametara (preporučeno) 0.1 i 0.1
      mreza->setBackpropWeightScale(0.1);

      mreza->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER+TermCriteria::EPS,
2000, 0.000001));     //kriterij za prestanak treninga(određeni broj iteracija ili jako
                      //mala promjena težinskih faktora


      mreza->train(trening_uzorci, ROW_SAMPLE, trening_klase); //pokretanje treninga

            if (mreza->isTrained())
                  cout << "Trening uspjesno obavljen\n\n"; // ispis poruke ako je mreža
                                                          //istrenirana

       Mat redak;
       int tocan_rez = 0;
       int pogresan_rez = 0;
       int vektor_rez [BR_KLASA] = {0,0,0,0,0,0,0,0,0,0};

            Mat test_rezultat = Mat(1, BR_KLASA, trening_uzorci.type());
            Point max_loc = Point(0,0);

                  cout<<"\Testiranje mreze uzorcima iz datoteke: \n\n";

       for (int i = 0; i < BROJ_TEST_PRIMJERA; i++)
       {
           redak = test_primjeri.row(i);  // uzmanje i-tog retka iz trening matrice

           mreza->predict(redak, test_rezultat);  //provlačenje test uzoraka kroz
                                                  //istreniranu mrežu

// mreža daje vektor vjerojatnosti za svaku klasu, uzima se ona s najvećom vjerojatnošću
           minMaxLoc(test_rezultat, 0, 0, 0, &max_loc);
                  cout<<"Uzorak br:"<<i<<" -> dobivena vrijednost "<<max_loc.x<<"
ocekivana"<<ocekivani.at<float>(0,i)<<endl;
```

```
                if (!(test_klase.at<float>(i, max_loc.x))) // usporedba dobivene i
                                                           //očekivane znamenke
        {
             pogresan_rez++;                       //brojač pogrešnih klasifikacija
            vektor_rez[(int) max_loc.x]++; // vektor pogrešnih klasifikacija za svaku
                                           //znamenku
        }
        else
        {
                        tocan_rez++;  // brojač točnih klasifikacija
        }
    }
    printf("\nRezultati test primjera: \n"
           "\tTočne klasifikacije: %d (%lf%%)\n"
           "\tNetočne klasifikacije: %d (%lf%%)\n\n\n",
           tocan_rez, (double) tocan_rez*100/BROJ_TEST_PRIMJERA,
           pogresan_rez, (double) pogresan_rez*100/BROJ_TEST_PRIMJERA);
                                    //ispis poruke nakon odrađenog testiranja



    for (int j = 0; j < BR_KLASA; j++)
    {
           printf( "\t (znamenka %d) pogresna klasifikacija   %d (%.2f%%)\n", j,
                   vektor_rez[j],
                   (double) vektor_rez[j]*100/BROJ_TEST_PRIMJERA);
                                    //ispis poruke nakon odrađenog testiranja
    }

return 0;
}
```

8.LITERATURA

[1] P. V. Balakrishnan; M. C. Cooper; V. S. Jacob; P. A. Lewis, A study of the classification capabilities of neural networks using unsupervised learning: A comparison with K-means clustering, Psychometrika, 59(4), 509-525,1994.

[2] V., Antonić, Pogled na neuronske mreže, PC Press, br. 175, ožujak,2011.

[3] K. Mehrotra; C. Mohan; S. Ranka, Elements of Artificial Neural Networks, The MIT Press, 1996.

[4] „OpenCV“ http://opencv.org/ lipanj 2016.

[5] H. G. Granlund; Hans Knutsson, Signal Processing for Computer Vision, Kluwer Academic Publisher, Linköping, 1995.

[6] S. Geman; E. Bienenstock; R. Doursat, Neural Computation, MIT Press, 4(1), 1-58, siječanj, 1992.

[7] „Neuronske mreže“ http://docs.opencv.org/2.4/modules/ml/doc/neural_networks.html# lipanj 2016.

[8] D.E., Rumelhart; G.E.Hinton; R.J. Williams, Learning representations by back-propagating errors, Nature, str. 533-536, br. 323., listopad 1986.

[9] „Backpropagation“ https://en.wikipedia.org/wiki/Backpropagation lipanj 2016. lipanj 2016.

[10]“OpenCV online dokumentacija“ http://docs.opencv.org/3.0.0/d0/dce/classcv_1_1ml_1_1ANN__MLP.html rujan 2016.

[11] M.A., Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[12] „MNIST baza podataka“ http://yann.lecun.com/exdb/mnist/ rujan 2016.

9.SAŽETAK

Naslov: Neuronske mreže u OpenCV biblioteci

Cilj ovog završnog rada bio je prikazati teorijsku podlogu umjetnih neuronskih mreža i njihovu primjenu u C++ programskom jeziku sa OpenCV bibliotekom. Obrađeni problem je prepoznavanje ljudskog rukopisa, točnije znamenaka. Potrebno je stvoriti neuronsku mrežu i definirati njene parametre što se postiže s klasama i funkcijama iz implementirane OpenCV biblioteke. Pokreće se proces učenja neuronske mreže sa što većim brojem uzoraka. Kad je proces treniranja gotov sustavu se predaju testni uzorci kako bi se utvrdila uspješnost procesa učenja. U simulaciji je mreža trenirana sa 1000 uzoraka, a testnih primjera predano joj je 100. Od 100 uzoraka prepoznato ih je 83 što bi značilo da je uspješnost na relativno visokoj razini.

Ključne riječi: neuronska mreža, OpenCV biblioteka, C++ jezik, prepoznavanje rukopisa

.

10.ABSTRACT

Title: Neural networks in OpenCV library

The aim of this bachelor thesis was to show the theoretical basis of artificial neural networks and their use in C ++ programming language with OpenCV library. Problem that was processed is human handwriting recognition, digits. It is necessary to create a neural network and define its parameters using classes and functions from the OpenCV library. The process of learning neural network is initiated with training samples. When the training process is finished, test samples are used in order to determine the success of the learning process. In the simulation, the network is trained with a 1000 samples and given a 100 test examples. Of the 100 samples, 83 were identified, which would mean that the success rate is relatively high.

Key words: neural network, OpenCV library, C++ language, handwriting recognition

11.ŽIVOTOPIS

Bože Eugen Marković rođen je 20. lipnja 1994. u Osijeku. U Osijeku je završio osnovnu školu te je nakon toga upisao Prirodoslovno-matematičku gimnaziju. Završetkom gimnazije upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija, smjer Računarstvo. Od stranih jezika služi se engleskim i njemačkim.