Implementacija neuronske mreže u mikroupravljač s ARM Cortex-M4 jezgrom

Tomašić, Magdalena

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

IMPLEMENTACIJA NEURONSKE MREŽE U MIKROUPRAVLJAČ S ARM CORTEX-M4 JEZGROM

Završni rad

Magdalena Tomašić

Osijek, 2019.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 24.04.2019.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Magdalena Tomašić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3840, 26.09.2018.
OIB studenta:	53196472746
Mentor:	Doc.dr.sc. Ratko Grbić
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Implementacija neuronske mreže u mikroupravljač s ARM Cortex-M4 jezgrom
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	24.04.2019.
Datum potvrde ocjene Odbora:	15.05.2019.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 20.05.2019.

Ime i prezime studenta:	Magdalena Tomašić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3840, 26.09.2018.
Ephorus podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Implementacija neuronske mreže u mikroupravljač s ARM Cortex-M4 jezgrom**

izrađen pod vodstvom mentora Doc.dr.sc. Ratko Grbić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Algoritam je točno određen slijed radnji kojima se nakon konačnog broja koraka dolazi do konačnog rješenja nekog problema, a može se prikazati dijagramom toka. Za razliku od klasičnih algoritama, algoritmi strojnog učenja svoju učinkovitost poboljšavaju iskustvom. Ovakvi algoritmi temelje se na tome da na osnovu ulaznih podataka otkrivaju uzorke i uz pomoć tih uzoraka donose zaključke. To se sve odvija uz minimalno ljudsko uplitanje odnosno čovjek samo unosi podatke na kojima stroj uči. Strojno učenje rješava probleme koje je inače teško riješiti uobičajenim algoritmima, a podaci se koriste za stvaranje znanja za zaključivanje i predviđanje. Jedna od metoda strojnog učenja je neuronska mreža. Ova metoda koristi se za rješavanje problema prepoznavanja rukom pisanih brojeva. Čovjek rješava ovaj problem korištenjem naučenih pravila za prepoznavanje pojedinih znamenki, npr. zna da se znamenka osam sastoji od dvije kružnice postavljene tako da se jedna nalazi iznad druge. Pristupanje ovom problemu pomoću klasičnih algoritama nema smisla jer bi bilo previše pravila i iznimaka za prepoznavanje znamenaka koje bi napisani program trebao sadržavati, dok neuronske mreže treniraju na slikama znamenki i same pronalaze ta pravila te ih čovjek ne mora sam pisati. Na temelju tih pravila neuronska mreža prepoznaje koja se znamenka nalazi na slici.

1.1. Zadatak završnog rada

U okviru završnog rada strukturirati neuronsku mrežu te podesiti njezine parametre za klasifikaciju rukom pisanih brojeva. U sljedećem koraku potrebno je izgrađenu neuronsku mrežu implementirati na mikroupravljač STM32F407VG. Na kraju treba ispitati dobiveno rješenje te analizirati efikasnost implementacije mreže.

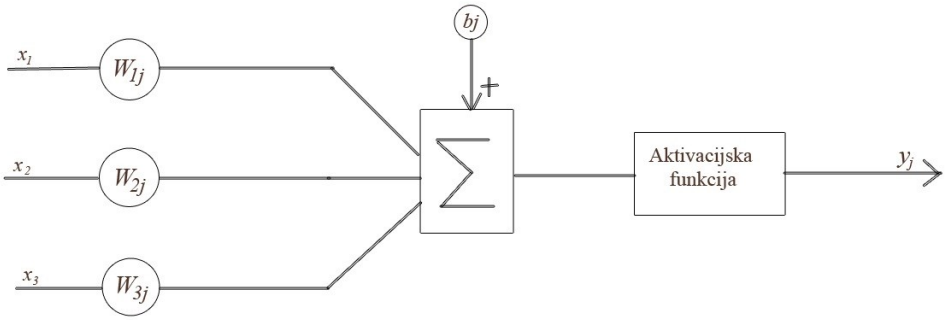
2. UMJETNE NEURONSKE MREŽE

Umjetna neuronska mreža inspirirana je ljudskom biološkom neuronskom mrežom. Napravljena je po uzoru na rad ljudskog mozga. Osnovna jedinica umjetnih neuronskih mreža je umjetni neuron[1]. U različitim područjima računarstva često se izostavlja pridjev umjetna i spominje se samo neuronska mreža stoga će i u narednim poglavljima to biti slučaj.

2.1. Osnovno o neuronima i neuronskim mrežama

Slično kao u ljudskom tijelu, umjetni neuroni primaju, obrađuju i prenose signale. Umjetni neuron ima više ulaza te samo jedan izlaz [1].

Na slici 2.1. prikazan je model neurona.



Sl. 2.1. Model neurona.

Obrada signala unutar neurona odvija se prema izrazu (2-1). Prvo se signal na svakom ulazu ($x_1, x_2, \ldots, x_n$) množi s težinama na tim ulazima ($w_{1j}, w_{2j}, \ldots, w_{nj}$ gdje je j redni broj neurona u sloju). Dobivene vrijednosti se zbrajaju i na konačnu sumu još se dodaje vrijednost nazvana *bias* (b_j). Težine i *bias* vrijednosti su parametri neuronske mreže koji se podešavaju prilikom učenja mreže. U posljednjem koraku rezultat prethodnih operacija se provlači kroz aktivacijsku funkciju f te se dobiva izlazni signal (y_j) koji se prosljeđuje sljedećem neuronu ili čini izlaz iz mreže [2]:

$$y_j = f\left(b_j + \sum_{i=1}^{n} (w_{ij} * x_i)\right). \qquad (2\text{-}1)$$

[1] Neuroni su živčane stanice živčanog sustava ljudskog organizma.

Prema [3, 4, 5] aktivacijska funkcija definira izlaz iz neurona te odlučuje hoće li se neuron aktivirati ili ne (slično kao što se aktiviraju neuroni u živom biću). Dijele se na linearne i nelinearne aktivacijske funkcije. Kod linearne aktivacijske funkcije se zbroj konačne sume i *biasa* (b_j) pomnoži s koeficijentom i dobije izlaz iz neurona (y_j). Nelinearne aktivacijske funkcije poprimaju različite oblike, a jedna od njih je ReLU (engl. *Rectified Linear Units*) funkcija koja prema izrazu (2-2) vraća nulu u slučaju ulaza manjeg od nule. U suprotnom, na izlazu iz funkcije se pojavljuje broj koji je na ulazu:

$$f(x) = \max(x, 0). \tag{2-2}$$

Postoje i druge nelinearne aktivacijske funkcije kao što su sigmoidna funkcija, ELU, LeakyReLU, Tanh itd.

Međusobno povezani neuroni čine neuronsku mrežu. Ti neuroni su podijeljeni u slojeve pa se tako razlikuju ulazni, skriveni i izlazni sloj. Ulazne vrijednosti ($x_1, x_2, ..., x_n$) prosljeđuju se preko neurona ulaznog sloja skrivenom sloju te se tamo vrše potrebne operacije. Izlazni sloj ne prosljeđuje dobivene vrijednosti dalje nego one predstavljaju izlaze iz neuronske mreže ($y_1, y_2, ..., y_n$) odnosno rezultat svih operacija. Jednoslojne mreže sadrže samo ulazni i izlazni sloj dok višeslojne imaju i jedan ili više skrivenih slojeva. Sve ulazne vrijednosti mogu se zajedno označiti s X, a sve izlazne s Y. X čini ulaz u neuronsku mrežu, a Y izlaz [2].

Posebna vrsta sloja neuronske mreže je *softmax* sloj. Taj sloj prema izrazu (2-3) skalira vrijednosti na izlazu između nule i jedinice tako da je zbroj svih izlaza jednak jedinici. To se postiže tako da se eksponencijalna vrijednost j-tog izlaza iz neurona jednog sloja (e^{z_j}) podijeli sa sumom eksponencijalnih vrijednosti izlaza iz svakog pojedinog neurona tog sloja (e^{z_k}). Dodavanjem ovog sloja na izlazima (y_j) iz neuronske mreže dobiju se decimalne vrijednosti između nule i jedinice. Zbroj tih vrijednosti jednak je jedinici. Takav izlaz (Y) povoljan je jer se pojedina vrijednost može gledati kao vjerojatnost:

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \ . \tag{2-3}$$

Mreže se mogu podijeliti i na cikličke i necikličke. Cikličke sadrže povratne veze dok necikličke ne sadrže nikakve povratne veze. S obzirom na povezanost razlikujemo djelomično povezane neurone i one koji su potpuno povezani odnosno svaki neuron jednog sloja povezan je sa svakim neuronom sljedećeg odnosno prethodnog sloja [2].

Prema [6, 7, 8] postoji nadzirano, nenadzirano i podržano učenje. Kod nadziranog učenja (engl. *supervised learning*) dostupne su vrijednosti ulaznih i izlaznih varijabli, dok kod učenja bez nadzora (engl. *unsupervised learning*) nisu poznate izlazne varijable. Podržano učenje (engl. *reinforcement learning*) podrazumijeva učenje interakcijom s okolinom. Nadzirano učenje dijeli se na regresijske i klasifikacijske probleme. Regresijski problemi imaju kontinuiranu odnosno numeričku izlaznu varijablu dok klasifikacijski imaju diskretnu odnosno kategoričku izlaznu varijablu što znači da regresijski problemi predviđaju kvantitet, a klasifikacijski svrstavaju ulazne podatke u kategorije. Učenje s nadzorom primjenjuje se kod predviđanja kretanja cijena dionica, klasifikacije teksta, prepoznavanja lica i govornika itd. Učenje bez nadzora koristi se kod analize ponašanja kupaca, segmentiranja tržišta, grupiranja teksta po sličnosti i fotografija po sadržaju, a podržano učenje za igranje igara, robotsko kretanje, autonomnu navigaciju, učenje kontrolnih strategija itd. Fokus ovog rada bit će na nadziranom učenju unaprijedne neuronske mreže odnosno mreže bez povratnih veza.

2.2. Izgradnja neuronske mreže

Izgradnja neuronske mreže može se podijeliti na tri faze. U prvoj fazi se određuje struktura mreže (broj slojeva i neurona te tip aktivacijske funkcije). Zatim se odvija faza učenja ili treniranja mreže u kojoj se određuju parametri mreže na temelju trening skupa podataka, a potom se vrši evaluacija izgrađene mreže.

Ovisno o problemu koji je potrebno riješiti, potrebno je odabrati prikladnu strukturu mreže. Svaka mreža sadrži jedan ulazni sloj za ulazne podatke i jedan izlazni sloj koji procjenjuje vrijednost izlazne veličine. Veličina ulaznog i izlaznog sloja ovisi o konkretnom problemu. Nadalje, potrebno je odrediti broj skrivenih slojeva te broj neurona u pojedinom skrivenom sloju. Svaki skriveni, kao i izlazni sloj, može imati različitu aktivacijsku funkciju.

Učenjem neuronske mreže započinje druga faza. Učenje (treniranje) neuronske mreže provodi se na skupu podataka za treniranje. U ovom radu koristi se nadzirano učenje što znači da se skup podataka za učenje sastoji od vrijednosti ulaznih veličina i odgovarajućih vrijednosti izlaznih veličina. Učenje mreže podrazumijeva podešavanje parametara mreže kako bi se minimizirala pogreška na skupu podataka za učenje. Kao mjera pogreške se kod neuronskih mreža najčešće koristi gubitak unakrsne entropije (engl. *cross entropy loss*) kod problema klasifikacije odnosno srednja kvadratna pogreška (engl. *mean squared error*) u slučaju regresijskih problema.

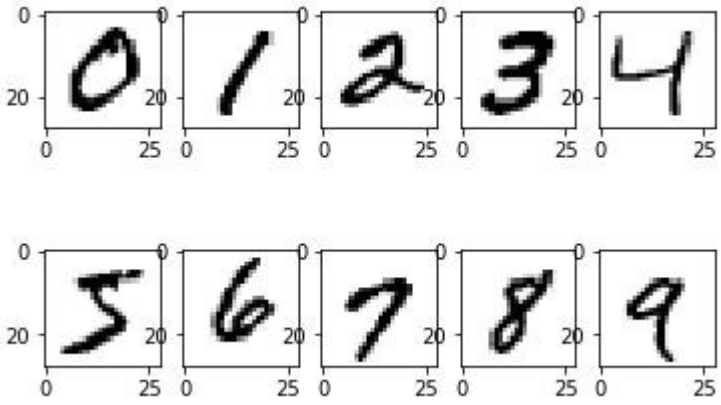
Podešavanje mreže odvija se u iteracijama pomoću odgovarajućih numeričkih postupaka poput gradijentne metode.

Po završetku treniranja mreže njeni parametri se fiksiraju te se provodi evaluacija neuronske mreže tako da joj se na ulaz dovode novi podaci odnosno podaci koji nisu korišteni u procesu treniranja. Pri tome se uspoređuje vrijednost izlaza koji daje neuronska mreže sa stvarnim izlazom za konkretni ulazni podatak. Uobičajeno se ovaj skup podataka naziva skup podataka za testiranje [1, 2].

3. IZRADA I IMPLEMENTACIJA NEURONSKE MREŽE

Kao što je već spomenuto u uvodnom poglavlju, za klasifikaciju rukom pisanih brojeva poželjno je koristiti algoritme strojnog učenja. U ovom radu koristi se višeslojna neuronska mreža koja na svom ulazu (X) prima slike znamenaka od nula do devet. To su slike veličine 28x28 piksela što znači da ukupno postoje 784 elementa na ulazu (x_1, x_2, ..., x_{784}) odnosno potreban je toliki broj neurona ulaznog sloja. Izlaz iz ove mreže (Y) predstavlja vjerojatnost da je određena znamenka na slici odnosno izlaz iz prvog neurona (y_1) predstavlja vjerojatnost da je na ulazu učitana slika znamenke nula, izlaz iz drugog (y_2) da je učitana slika znamenke jedan i tako redom za svih 10 znamenki (y_1, y_2, ..., y_{10}) što znači da je potrebno 10 neurona na izlazu [9].

Slike koje se koriste su slike znamenaka iz MNIST skupa (sl. 3.1.). Taj skup sadrži 70 000 slika veličine 28x28 piksela. Svaki piksel predstavljen je cjelobrojnom vrijednosti između 0 i 255 gdje broj 255 predstavlja crnu, a broj 0 bijelu boju dok sve vrijednosti između ta dva broja označavaju nijanse sive. Na slikama su znamenke od nula do devet, a ukupan broj slika podijeli se na dio za trening i dio za test. Dio za trening sadrži 60 000 slika znamenaka, dok dio za test sadrži njih 10 000. U oba su skupa znamenke poredane uzlazno od nula do devet. Prije same podjele potrebno je skalirati vrijednosti svakog piksela tako da se dobiju veličine između nula i jedan u zapisu s pomičnim zarezom jer neuronska mreža podrazumijeva korištenje realnih brojeva [9].



Sl. 3.1. Slike znamenaka iz MNIST skupa.

U skladu s prethodno odabranim brojem ulaznih i izlaznih neurona, a prije izrade neuronske mreže, odredi se broj neurona skrivenih slojeva i broj skrivenih slojeva u neuronskoj mreži te izabire pogodna vrsta aktivacijske funkcije.

Nakon što je odabrana struktura neuronske mreže, njena struktura i proces učenja implementiraju se u programskom jeziku *Python*. Potom se provodi evaluacija neuronske mreže te ako se postiže zadovoljavajuća preciznost klasifikacije smatra se da je neuronska mreža uspješno izgrađena.

Budući da se programiranje mikroupravljača radi u C programskom jeziku, potrebno je *Python* implementaciju naučene neuronske mreže prebaciti u odgovarajući C programski kod. Za ovaj korak koristi se *Visual Studio* okruženje zbog jednostavnosti programiranja i otklanjanja pogrešaka u C programu. Izgradi se identična neuronska mreža kao u *Python* programskom jeziku. Kako bi obje mreže imale iste parametre, parametri naučene mreže se u *Python-u* spreme u tekstualnu datoteku koja se zatim učita u C program. Vrijednosti tih parametara se pridruže odgovarajućim varijablama u C programu. Na kraju se testira ispravnost izgrađene mreže.

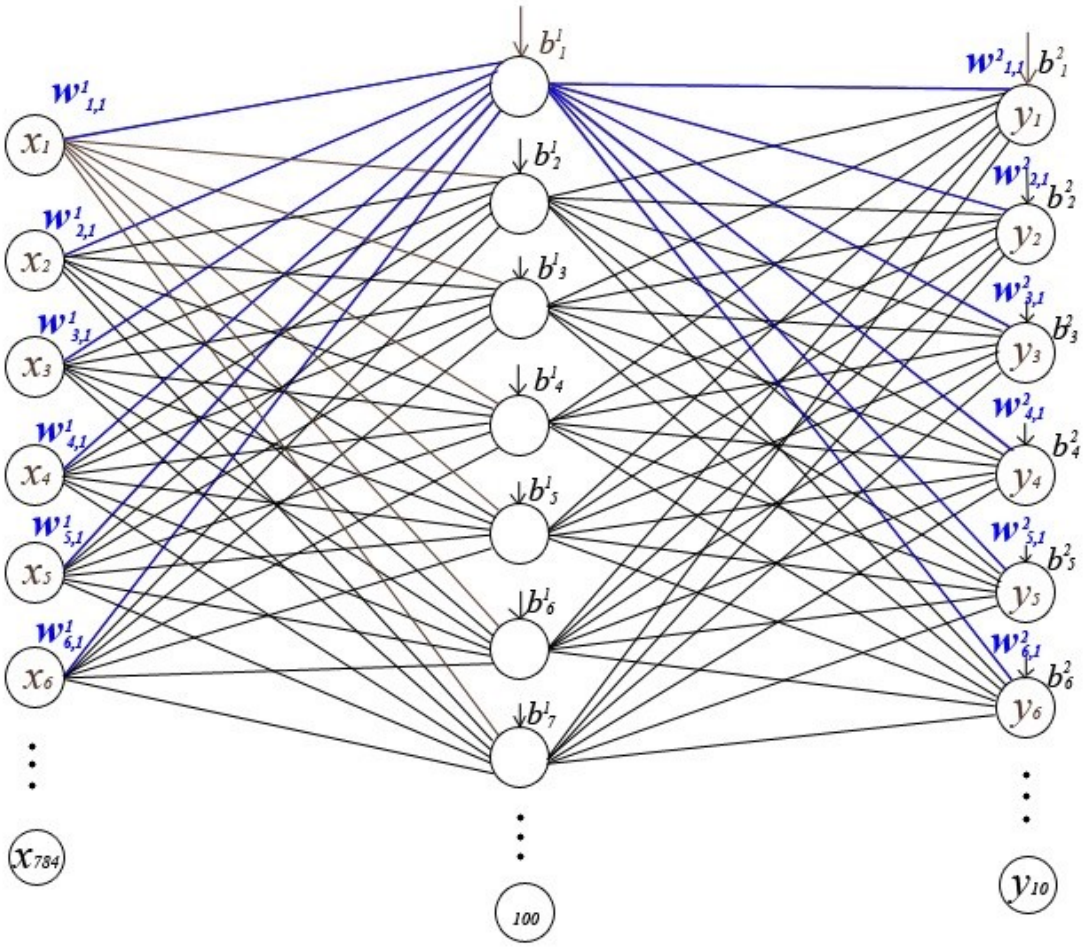
Sljedeći korak je implementacija izgrađene neuronske mreže u mikroupravljač uz pomoć C implementacije koja je napravljena u okviru *Visual Studia*. Budući da mikroupravljač ne posjeduje medij za pohranu slika, testne slike šalju se preko serijskog sučelja na mikroupravljač gdje se, po primitku zadnjeg elementa slike, izračunavaju izlazi neuronske mreže. Provjerava se točnost i preciznost mreže te prati vrijeme klasificiranja. Primjerice, ako je poslana slika znamenke tri, na izlazu bi najveću vrijednosti trebala poprimiti varijabla y_4. Ostale varijable trebale bi težiti nuli, dok varijabla y_4 jedinici.

3.1. Struktura mreže

Kao što je određeno u uvodu ovog poglavlja, neuronska mreža sadrži 784 ulazna i 10 izlaznih neurona. Sastoji se od 3 sloja: jedan ulazni, jedan izlazni i jedan skriveni sloj. Središnji sloj sadržava 100 neurona jer se taj broj neurona pokazao dovoljnim za postizanje zadovoljavajuće preciznosti klasifikacije.

Aktivacijska funkcija skrivenog sloja je nelinearna aktivacijska funkcija ReLU, a izlaznog sloja *softmax* funkcija. Djelovanja ovih funkcija objašnjena su u prethodnom poglavlju.

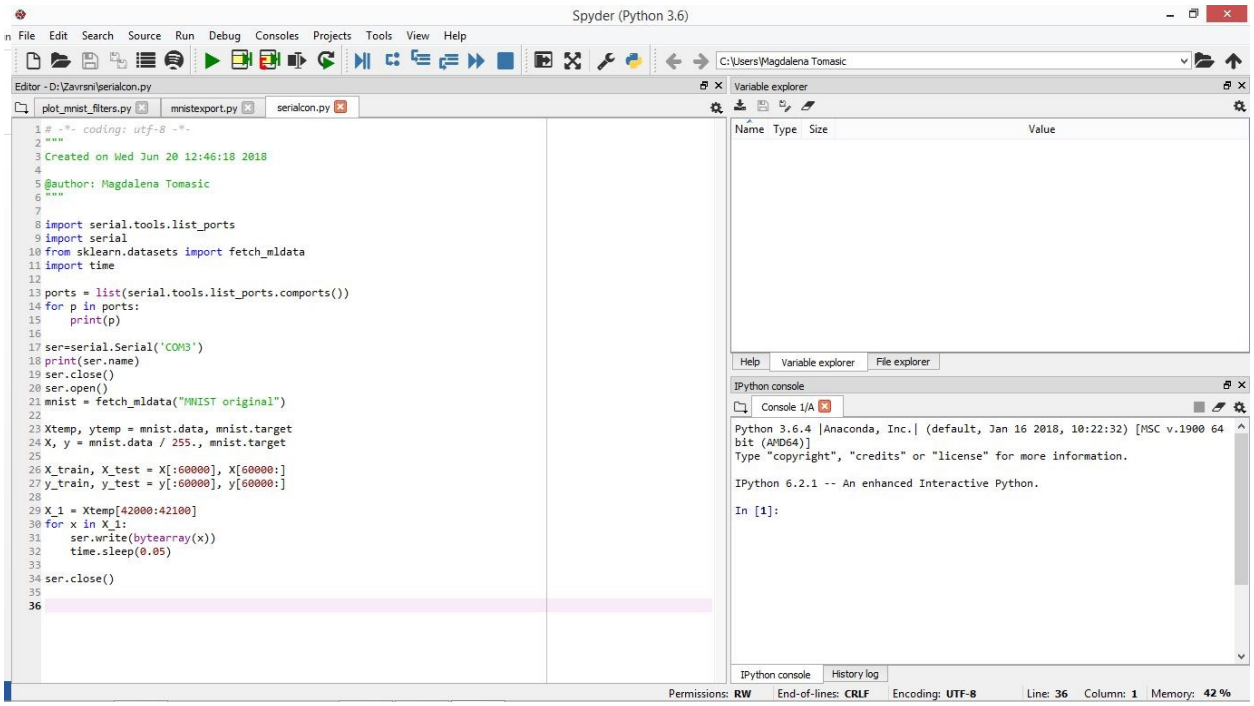
Na slici 3.2. prikazana je opisana struktura mreže. Ulazi x_i čine ulazni sloj od 784 neurona, a vrijednosti y_j čine izlazni sloj od 10 neurona. Vektor $\mathbf{b^1}$ sadrži *bias* svakog pojedinog neurona prvog sloja, a vektor $\mathbf{b^2}$ *bias* vrijednosti drugog sloja. Matrice $\mathbf{W^1}$ i $\mathbf{W^2}$ su težine prvog odnosno drugog sloja.



Sl. 3.2. Struktura mreže.

3.2. Izgradnja neuronske mreže u programskom jeziku *Python*

Za izgradnju neuronske mreže koristi se programski jezik *Python* (verzija 3.6.5), *Anaconda* distribucija i razvojno okruženje *Spyder* prikazano na slici 3.3.



Sl. 3.3. *Spyder* razvojno okruženje.

Cijeli izvorni kod *Python* programa za učenje i testiranje neuronske mreže dostupan je u prilogu 3.1.

Prvo se dohvaćaju potrebne biblioteke (*sklearn.datasets* i *sklearn.neural_network*) i potrebne slike se iz MNIST skupa podataka zapisuju u varijablu *mnist* (sl. 3.4.).

```
from sklearn.datasets import fetch_mldata
from sklearn.neural_network import MLPClassifier

mnist = fetch_mldata("MNIST original")
```

Sl. 3.4. Dohvaćanje biblioteka.

Zatim se skaliraju pikseli kao što je objašnjeno u uvodu ovog poglavlja. Vrijednosti koje se nalaze u *mnist.data* upisuju se u varijablu *X*, a one u *mnist.target* u varijablu *y*. Vrijednosti zapisane u *mnist.data* predstavljaju piksele slika, a u *mnist.target* su vrijednosti od nula do devet. Zatim se te vrijednosti podijele na dio za trening i dio za test tako da se prvih 60 000 koristi za učenje mreže, a preostalih 10 000 za testiranje mreže. Ovi skupovi podataka predstavljeni su varijablama *X_train*, *X_test*, *y_train* i *y_test* (sl. 3.5.).

```
X, y = mnist.data / 255., mnist.target
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
```

Sl. 3.5. Skaliranje i podjela podataka.

Neuronska mreža definira se kao objekt klase *MLPClassifier* kao što je prikazano na slici 3.6. Prilikom definiranja objekta moguće je postaviti vrijednosti određenih parametara vezanih za strukturu i učenje neuronske mreže kao što je broj neurona skrivenog sloja, broj iteracija učenja, numerički postupak, vrstu aktivacijske funkcije i sl. Nije nužno definirati svaki parametar jer klasa *MLPClassifier* ima unaprijed definirane standardne vrijednosti za pojedine parametre.

```
mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=10, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)
```

Sl. 3.6. Kreiranje *mlp* objekta.

Mreža se uči pomoću metode *fit* kojoj se predaje ulazni skup podataka (*X_train, y_train*). Nakon učenja na trening podacima (*X_train*, *y_train*) neuronska mreža može prepoznati znamenke na novim uzorcima (*X_test*). Koristeći metodu *predict_proba* dobiju se vrijednosti izlaznih veličina za uzorke u testnom skupu podataka (*X_test*) koje se zapisuju u varijablu *probability* i one čine izlaz iz neuronske mreže (*Y*). Metodi *predict* također se predaju uzorci u testnom skupu podataka (*X_test*). Rezultat te metode su predviđene znamenke koje se nalaze na uzorcima, a on se zapisuje u varijablu *prediction*. Može se primijetiti da se kod učenja mreži predaju i ulazni i izlazni podaci dok je kod testiranja mreže izlaz mreži nepoznat (sl. 3.7.).

```
mlp.fit(X_train, y_train)
probability=mlp.predict_proba(X_test)
prediction=mlp.predict(X_test)
```

Sl. 3.7. Treniranje i testiranje mreže.

Pomoću metode *score* može se izračunati preciznost neuronske mreže na temelju predanih parametara: *X_train* i *y_train* za izračunavanje preciznosti mreže prilikom klasifikacije uzoraka iz trening skupa podataka te *X_test* i *y_test* za izračunavanje preciznosti mreže prilikom klasifikacije uzoraka iz testnog skupa podataka (sl. 3.8.). Preciznost se računa na temelju broja točno klasificiranih uzoraka.

```
print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))
```

Sl. 3.8. Izračunavanje preciznosti mreže.

Dobivene vrijednosti su prikazane na slici 3.9. Može se vidjeti da je vrijednost veća kod klasifikacije uzoraka na trening skupu što znači da je neuronska mreža preciznija prilikom klasifikacije uzoraka iz trening skupa nego prilikom klasifikacije uzoraka iz testnog skupa.

```
Training set score: 0.994950
Test set score: 0.979200
```

Sl. 3.9. Dobivene vrijednosti za preciznost mreže.

Zbog implementacije naučene neuronske mreže u C programski jezik potrebno je binarno zapisati slike znamenaka i atribute objekta *mlp* (*coefs* i *intercepts*) u tekstualne datoteke. Atribut *coefs* je matrica težina, a atribut *intercepts* predstavlja *bias* vrijednosti. Svaka slika sastoji se od 784 elemenata tipa *float64* odnosno svaki piksel veličine je 8 bajtova stoga veličina ove datoteke na disku treba biti 6272 bajtova. Isto je i s veličinama datoteka u kojima se nalaze težine i *bias* vrijednosti (sl. 3.10.).

```
39 X_1 = X_test[1,:]
40 f_handle_image = open('image0.txt','wb')
41 X_1.tofile(f_handle_image, sep="")
42 f_handle_image.close()
43
44
45 # export weights
46 weights1=mlp.coefs_[0]
47 f_handle_image=open('weights1.txt','wb')
48 weights1.tofile(f_handle_image, sep="")
49 f_handle_image.close()
50
51 weights2=mlp.coefs_[1]
52 f_handle_image=open('weights2.txt','wb')
53 weights2.tofile(f_handle_image, sep="")
54 f_handle_image.close()
55
56
57 # export biases
58 biases1=mlp.intercepts_[0]
59 f_handle_image=open('biases1.txt','wb')
60 biases1.tofile(f_handle_image, sep="")
61 f_handle_image.close()
62
63 biases2=mlp.intercepts_[1]
64 f_handle_image=open('biases2.txt','wb')
65 biases2.tofile(f_handle_image, sep="")
66 f_handle_image.close()
```

Sl. 3.10. Zapisivanje parametara mreže i prve testne slike u tekstualne datoteke.

Nakon što su potrebni parametri zapisani u datoteke, naučena se mreža implementira u C programski jezik u okviru *Visual Studia*. Cjelokupni kod nalazi se u prilozima 3.4, 3.5 i 3.6.

Prvo se otvore datoteke u kojima se nalaze atributi klase (parametri mreže) i binarni zapis znamenke, a to su onih pet datoteka koje su kreirane u Python programskom jeziku. Vrijednosti učitanih datoteka učitaju se u dvodimenzionalna polja odnosno matrice realnih brojeva (*image0*, *biases1*, *biases2*, *weights1*, *weights2*). Postupak prikazan na slici 3.11. odnosi se samo na *bias* vrijednosti prvog sloja, ali postupak je potpuno isti za ostale parametre i za spremljenu sliku znamenke. Na slici 3.12. nalazi se definicija funkcije *load_from_file* za čitanje podataka spremljenih u datoteci.

```
if ((fptr = fopen("D:\\Zavrsni\\biases1.txt", "rb")) == NULL)
{
    printf("Error! opening file");
    exit(1);
}
biases1 = load_from_file(fptr);
fclose(fptr);
```

Sl. 3.11. Otvaranje datoteke.

```
double* load_from_file(FILE *pointer)
{
    unsigned long fileLen;

    fseek(pointer, 0, SEEK_END);
    fileLen = ftell(pointer);
    fseek(pointer, 0, SEEK_SET);

    double *buffer = (double*)malloc(fileLen + 1);
    if (!buffer)
    {
        fprintf(stderr, "Memory error");
        fclose(pointer);
        return 0;
    }

    fread(buffer, fileLen / 8, 8, pointer);

    return buffer;
}
```

Sl. 3.12. Čitanje podataka iz datoteke.

Iako je moguće računati s jednodimenzionalnim poljima u kojima se nalaze vrijednosti učitane iz datoteka, zbog lakšeg računanja i vizualizacije u sljedećem koraku se te vrijednosti

učitaju u dvodimenzionalna polja (*img*, *b1*, *b2*, *w1*, *w2*). Kao što je prikazano na slici 3.13., vrši se memorijska alokacija, a zatim se poziva funkcija *load_to_2Darray* (sl. 3.14.). Na kraju se oslobađa memorija zauzeta pri kreiranju jednodimenzionalnog polja koje nam više nije potrebno. Kao i na prethodnim slikama, prikazan je postupak za jedan parametar (*b1*), ali vrijedi i za sve ostale (*b2*, *w1*, *w2*) kao i za sliku (*img*).

```
ROWS = 1;
COLUMNS = 100;

double **b1 = (double**)malloc(ROWS * sizeof(double*));
for (I = 0; I < ROWS; I++)
    b1[I] = (double*)malloc(COLUMNS * sizeof(double));

load_to_2Darray(biases1, ROWS, COLUMNS, b1);
free(biases1);
```

Sl. 3.13. Kreiranje nove varijable, poziv funkcije *load_to_2Darray* i oslobađanje memorije.

```
void load_to_2Darray(double* array1D, int rows, int columns, double** array2D)
{
    for (i = 0; i < rows; i++)
        for (j = 0; j < columns; j++)
            array2D[i][j] = array1D[i * columns + j];
}
```

Sl. 3.14. Funkcija *load_to_2Darray.*

Nakon što su svi potrebni parametri i slika znamenke pohranjeni u varijable, poziva se funkcija *predict* koja prima te parametre i sliku. Rezultat te funkcije su vjerojatnosti spomenute u uvodu ovog poglavlja, a on se zapisuje u varijablu *result*. Definicija te funkcije prikazana je na slici 3.15., a pripadajuće funkcije koje se pozivaju unutar nje prikazane su na slikama 3.16., 3.17., 3.18., 3.19. i 3.20.

```
double** predict(double** weights1, double** weights2, double** biases1, double** biases2, double** image)
{
    rows = 1;
    columns = 100;

    double **res1 = (double**)malloc(rows * sizeof(double*));
    for (i = 0; i < rows; i++)
        res1[i] = (double*)malloc(columns * sizeof(double));

    initialize(res1, rows, columns);

    matrix_mul(res1, image, weights1, rows, columns, 784);

    matrix_sum(res1, biases1, rows, columns);

    ReLU(res1, rows, columns);

    rows = 1;
    columns = 10;

    double **res2 = (double**)malloc(rows * sizeof(double*));
    for (i = 0; i < rows; i++)
        res2[i] = (double*)malloc(columns * sizeof(double));

    initialize(res2, rows, columns);

    matrix_mul(res2, res1, weights2, rows, columns, 100);

    matrix_sum(res2, biases2, rows, columns);

    Softmax(res2, rows, columns);

    return res2;
}
```

Sl. 3.15. Funkcija *predict*.

```
void initialize(double** array2D, int rows, int columns)
{
    for (i = 0; i<rows; ++i)
        for (j = 0; j<columns; ++j)
            array2D[i][j] = 0;
}
```

Sl. 3.16. Funkcija za inicijalizaciju polja.

```
void matrix_mul(double** result, double** array2D, double** array2d, int rows, int columns, int rc)
{
    for (i = 0; i<rows; ++i)
        for (j = 0; j<columns; ++j)
            for (n = 0; n<rc; ++n)
                result[i][j] += array2D[i][n] * array2d[n][j];
}
```

Sl. 3.17. Funkcija za množenje matrica.

```
void matrix_sum(double** array2D, double** array2d, int rows, int columns)
{
    for (i = 0; i < rows; i++)
        for (j = 0; j < columns; j++)
            array2D[i][j] = array2D[i][j] + array2d[i][j];
}
```

Sl. 3.18. Funkcija za zbrajanje matrica.

```
void ReLU(double** array2D, int rows, int columns)
{
    for (i = 0; i < 1; i++)
        for (j = 0; j < 100; j++)
            if (array2D[i][j] < 0) array2D[i][j] = 0;
}
```

Sl. 3.19. ReLU funkcija.

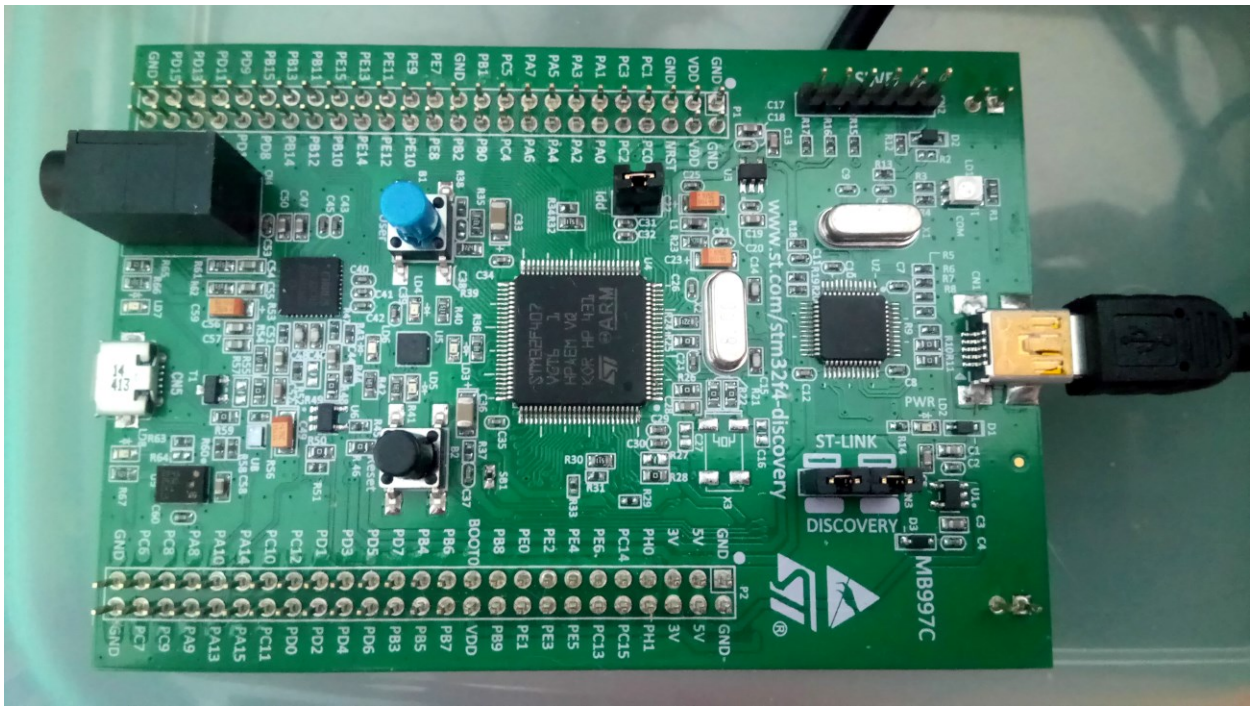
```
void Softmax(double** array2D, int rows, int columns)
{
    double expsum = 0;

    for (i = 0; i < 1; i++)
        for (j = 0; j < 10; j++)
            expsum += exp(array2D[i][j]);

    for (i = 0; i < 1; i++)
        for (j = 0; j < 10; j++)
            array2D[i][j] = exp(array2D[i][j]) / expsum;
}
```

Sl. 3.20. *Softmax* funkcija.

Sam postupak izgradnje neuronske mreže i dobivanja izlaza iz nje je drugačiji nego u *Python* programskom jeziku jer ne postoje gotove funkcije za definiranje strukture mreže kao što je to klasa *MLPClassifier* u *Python* programskom jeziku već se koriste standardne C funkcije i parametri mreže izračunati u programskom jeziku *Python.* Međutim, ovakav C kod znatno je lakše implementirati u mikroupravljač.

3.3. Implementacija izgrađene neuronske mreže u mikroupravljač

Izgrađena neuronska mreža implementira se u mikroupravljač s ARM Cortex-M4 jezgrom prikazan na slici 3.21.



Sl. 3.21. Mikroupravljač s ARM Cortex-M4 jezgrom.

Napajanje uređaja odvija se preko USB kabela (sl. 3.22.) koji spaja mikroupravljač s osobnim računalom, a USART (engl. *Universal synchronous asynchronous receiver transmitter*) periferija ovog mikroupravljača omogućava komunikaciju mikroupravljača s osobnim računalom. Sastoji se od predajnika i prijemnika s odgovarajućim pinovima na mikroupravljaču označenim s TX (pin za slanje podataka) i RX (pin za prijem podataka). Preko ovih pinova odvija se serijska komunikacija mikroupravljača i osobnog računala. Kada se želi poslati neki podatak, vrši se zapisivanje vrijednosti u podatkovni registar predajnika, a kod primanja podataka čita se podatkovni registar prijemnika. USART se konfigurira tako da se podešavaju vrijednosti u konfiguracijskim registrima USART-a.

Sl. 3.22. USB kabel za napajanje mikroupravljača [10].

U slučaju da se želi ostvariti komunikacija između mikroupravljača i osobnog računala koje nema serijski port, potrebno je koristiti pretvornik PL2303TA prikazan na slici 3.23. Crnu žicu potrebno je spojiti na GND pin mikroupravljača, zelena služi za primanje podataka i spaja se na RX pin, a bijela za slanje podataka i spaja se na TX pin.

Sl. 3.23. PL2303TA pretvornik [11].

Mikroupravljač STM32F407VG sadrži tri USART sučelja. Za potrebe slanja slika na mikropuravljač koristi se USART1 sučelje. Nakon što se fizički poveže mikroupravljač s osobnim računalom, potrebno je uključiti takt za USART1 i *port* B i konfigurirati pinove PB6 i PB7. Pomoću funkcije *GPIO_PinAConfig* povezuju se ti pinovi s njihovom ulogom na USART1 (TX ili RX). Nakon toga se podešava USART1 tako da se odredi brzina prijenosa podataka, paritet, broj stop bitova i veličina podatka koji se istovremeno prenosi. Rad USART1 periferije omogućen je funkcijom *USART_Cmd* (sl. 3.24.).

```
//UART
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);// Enable clock for GPIOB
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);// Enable clock for USART1

GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_USART1);// Connect PB6 to USART1_Tx
GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_USART1);// Connect PB7 to USART1_Rx

// Initialization of GPIOB
GPIO_InitTypeDef GPIO_InitStructUART;
GPIO_InitStructUART.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
GPIO_InitStructUART.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructUART.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructUART.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructUART.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOB, &GPIO_InitStructUART);

// Initialization of USART1
USART_InitTypeDef USART_InitStruct;
USART_InitStruct.USART_BaudRate = 9600;
USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStruct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_InitStruct.USART_Parity = USART_Parity_No;
USART_InitStruct.USART_StopBits = USART_StopBits_1;
USART_InitStruct.USART_WordLength = USART_WordLength_8b;
USART_Init(USART1, &USART_InitStruct);

// Enable USART1
USART_Cmd(USART1, ENABLE);
```

Sl. 3.24. USART1 konfiguracija.

Moguće je podesiti USART periferiju mikroupravljača tako da generira zahtjev za prekidom kada se ispune određeni uvjeti. U ovom radu izvor prekida bit će primanje novog podatka. Nakon inicijalizacije USART-a definira se prioritet prekida i omogućuju USART1 zahtjevi za prekidom prilikom primanja novog podatka (sl. 3.25.).

```
USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
NVIC_SetPriorityGrouping(NVIC_PriorityGroup_4);
NVIC_SetPriority(USART1_IRQn, NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0,0));
NVIC_EnableIRQ(USART1_IRQn);
```

Sl. 3.25. Omogućavanje USART1 prekida uslijed primanja podataka.

Za mjerenje vremena koristi se 24-bitni sistemski brojač vremena *SysTick* koji broji prema „dolje“ od određene vrijednosti s odgovarajućim taktom. Ta se vrijednost naziva *RELOAD* vrijednost i učitava se u brojač kada on dosegne vrijednost 0. Generiranje prekida u željenim vremenskim intervalima moguće je tako da se postavi odgovarajuća *RELOAD* vrijednost. Podešavanje ovog brojača vremena omogućeno je pomoću funkcije *SysTick_Config* (sl. 3.26.) koja

prima broj otkucaja između dva prekida (umnožak željenog perioda i sistemske frekvencije). Primjerice, u ovom radu ta funkcija prima cjelobrojnu vrijednost 168 000 jer sistemska frekvencija iznosi 168 MHz, a željeni period 1 ms (1 ms * 168 MHz = 168 000) stoga će se svake milisekunde za 1 povećati vrijednost varijable *noMs* koja će služiti za mjerenje proteklog vremena (sl. 3.27.).

```
SysTick_Config(168000);
```

Sl. 3.26. Podešavanje brojača vremena mikroupravljača.

```
void SysTick_Handler(void)
{
    noMs++;
}
```

Sl. 3.27. Kod prekidne rutine.

Nakon implementacije, preko serijskog porta pošalju se vrijednosti proizvoljne slike iz MNIST skupa u mikroupravljač. Slika se šalje tako da se prenosi osam bitova istovremeno. Osam po osam bitova dok se ne prenese cijela slika znamenke veličine 28x28 piksela odnosno 784 piksela. Da bi se utvrdilo da je poslana slika ispravno primljena, tijekom razvoja rješenja uspoređivane su vrijednosti elemenata slike u *Python* programskom jeziku s primljenim vrijednostima koje se nalaze u varijabli *IMG* koja predstavlja polje u RAM memoriji mikroupravljača u koju se sprema testna slika. Parametri mreže, odnosno težine i *biasi* su definirani kao konstantna polja te su tako spremljeni u *flash* memoriju mikroupravljača.

Na slici 3.28. prikazan je dijagram toka programa koji je implementiran u mikroupravljač. U glavnom dijelu programa izvodi se kod za prepoznavanje znamenke na slici, a konačno rješenje upisuje se u varijablu *r2*. Prekid se događa kada mikroupravljač primi bajt na serijskom sučelju. Prilikom prekida izvođenja glavnog dijela programa izvodi se prekidna rutina u kojoj se elementi poslane slike primaju i učitavaju u varijablu *IMG*. Prilikom primitka svih elemenata slike, glavni dio programa nastavlja s izvođenjem. U slučaju da slika nije poslana, program čeka da se ona pošalje.

Sl. 3.28. Dijagram toka programa.

Na slici 3.29. prikazan je dio koda koji se izvodi prilikom prekida izvođenja glavnog dijela programa. Pomoću funkcije *USART_ReceiveData* elementi slike učitavaju se u polje cijelih brojeva *IMG*. Potrebno je skalirati dobivene vrijednosti pa se one dijele s brojem 255 tako da se dobiju vrijednosti između nula i jedan, a one se upisuju u polje realnih brojeva *imag*. To polje predstavlja ulaz neuronske mreže *X*. Kada se učitaju svi pikseli poslane slike, glavni dio programa nastavlja s izvođenjem.

```
void USART1_IRQHandler(void)
{
    IMG[n] = USART_ReceiveData(USART1);
    imag[n] = IMG[n]/255.0;

    if(n==783)
    {
        n=0;
        flag = 1;
    }
    else n++;

    USART_ClearITPendingBit(USART1, USART_IT_RXNE);
}
```

Sl. 3.29. Prekidna rutina koja se poziva prilikom primanja podatka na serijsko sučelje.

Pri primitku svih vrijednosti slike, množe se te vrijednosti s vrijednostima dvodimenzionalne matrice koja predstavlja težine prvog sloja (dvodimenzionalno polje realnih brojeva *w1*), dodaju se *bias* vrijednosti zapisane u matrici *b1*, a zatim se dobiveni brojevi provlače kroz aktivacijsku funkciju ReLU. Ovaj međurezultat upisuje se u polje realnih brojeva *r1* (sl. 3.30.).

```
for(i=0;i<100;i++){
    for(j=0;j<784;j++){
        r1[i]+=imag[j]*w1[j][i];
    }
}

for(i=0;i<100;i++){
    r1[i]+=b1[0][i];
    if (r1[i] < 0.0) r1[i] = 0.0;
}
```

Sl. 3.30. Izračunavanje vrijednosti prvog sloja.

U sljedećem koraku se međurezultat množi s elementima matrice težina *w2*, zbraja s *bias* vrijednostima iz matrice *b2* i na kraju se koristi *softmax* funkcija. Krajnji rezultat je vektor *r2* s deset vrijednosti koje predstavljaju ranije spomenute vjerojatnosti da se na ulazu pojavila određena znamenka (sl. 3.31.).

```
for(i=0;i<10;i++){
    for(j=0;j<100;j++){
        r2[i]+=r1[j]*w2[j][i];
    }
}

for(i=0;i<100;i++){
    r2[i]+=b2[0][i];
    if (r2[i] < 0.0) r2[i] = 0.0;
}

for (i = 0; i < 10; i++)
    expsum += exp(r2[i]);

for (i = 0; i < 10; i++)
    r2[i] = exp(r2[i]) / expsum;
```

Sl. 3.31. Izračunavanje konačnih vrijednosti.

Za potrebe mjerenja vremena izvođenja glavnog dijela programa odnosno mjerenja vremena klasifikacije izvodi se kod na slikama 3.32. i 3.33. Kod na slici 3.32. izvodi se prije klasifikacije. Varijabla *imageCounter* povećava svoju vrijednost te tako broji prenesene slike, a u varijablu *startTime* se zapisuje vrijednost iz varijable *noMs* koja mjeri proteklo vrijeme.

```
imageCounter++;
startTime = noMs;
```

Sl. 3.32. Brojač slika i mjerač vremena.

Nakon što se dobije konačni rezultat u glavnom dijelu programa (vektor *r2*), izvodi se kod na slici 3.33. U varijablu *stopTime* upisuje se trenutna vrijednost zapisana u varijabli *noMs*. Razlika vrijednosti trenutno zapisanih u varijablama *stopTime* i *startTime* predstavlja vrijeme potrebno za klasifikaciju rukom pisane znamenke. Uz pomoć brojača slika lako se može pratiti vrijeme za svaku prenesenu sliku.

```
stopTime = noMs;
diff[imageCounter-1] = stopTime - startTime;
```

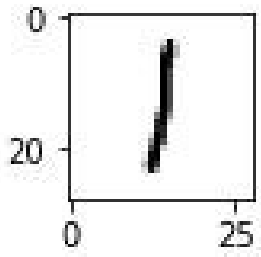
Sl. 3.33. Vrijeme potrebno za klasifikaciju rukom pisane znamenke.

4. REZULTATI

Ispravnost implementacije neuronske mreže u mikroupravljač provjerena je usporedbom s vrijednostima koje za iste ulazne slike daje mreža implementirana u programskom jeziku *Python* tako da je za svaku sliku testnog skupa uspoređen izlaz koji daje metoda *predict* (vrijednosti od 0 do 9) objekta klase *MLPClassifier* s indeksom maksimalne vrijednosti vektora *r2* koja se dobije za pojedinu sliku.

Za svaku testnu sliku mreža implementirana u mikroupravljaču dala je jednaku izlaznu vrijednost kao i mreža u programskom jeziku *Python* što potvrđuje ispravnost implementacije.

Primjerice, ako se na ulazu u implementiranu neuronsku mrežu nalazi slika znamenke jedan odnosno ako je poslana slika broj 2000 iz MNIST testnog skupa (sl. 4.1.), na izlazu se dobiju vrijednosti kao na slici 4.2. Te vrijednosti su zapisane u varijabli *r2*. Takve vrijednosti dobiju se i prilikom testiranja neuronske mreže u razvojnom okruženju *Spyder* (sl. 4.3.), a one su zapisane u varijabli *probability*.



Sl. 4.1. 2000. slika iz MNIST skupa.

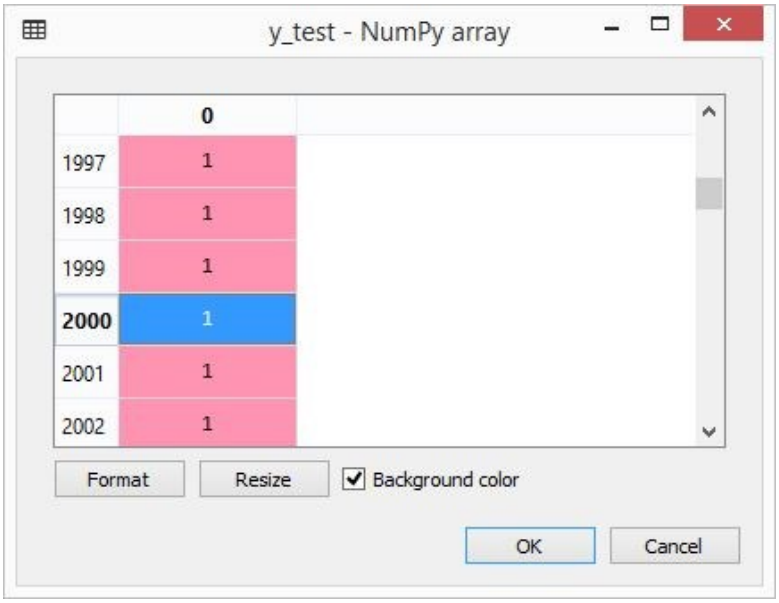
(x)= r2[0]	volatile float	5.5513492e-006
(x)= r2[1]	volatile float	0.999616146
(x)= r2[2]	volatile float	5.5513492e-006
(x)= r2[3]	volatile float	5.5513492e-006
(x)= r2[4]	volatile float	3.38833415e-005
(x)= r2[5]	volatile float	5.5513492e-006
(x)= r2[6]	volatile float	5.5513492e-006
(x)= r2[7]	volatile float	7.19572345e-005
(x)= r2[8]	volatile float	0.000244622584
(x)= r2[9]	volatile float	5.5513492e-006

Sl. 4.2. Varijabla *r2* za sliku broj 2000.

	0	1	2	3	4	5	6	7	8	9
1995	1.17776e-09	0.997699	6.50033e-05	5.14487e-06	0.00140857	2.48775e-09	8.85744e-10	0.00081686	4.93793e-06	1.7042e-08
1996	3.54957e-10	0.999975	8.75472e-06	3.53119e-07	5.23855e-07	5.5206e-11	8.42282e-11	1.52541e-05	1.01851e-07	2.10701e-10
1997	2.52586e-10	0.999976	1.74145e-05	1.32452e-07	3.18477e-07	3.31561e-11	1.14462e-10	5.9006e-06	1.7555e-07	4.77459e-11
1998	4.8481e-10	0.999438	2.70039e-07	1.92054e-06	7.67966e-06	2.50639e-11	2.07527e-11	0.00055117	5.69959e-07	3.0761e-09
1999	4.50367e-11	0.999673	8.28042e-07	1.23465e-05	4.57186e-05	1.75566e-11	1.72782e-11	0.000268233	2.17233e-07	3.97892e-09
2000	1.91809e-08	0.999647	1.28945e-06	1.69055e-07	3.38842e-05	7.52205e-07	3.93433e-07	7.196e-05	0.000244631	1.69917e-09
2001	1.73198e-09	0.998663	9.81659e-07	5.95101e-07	0.000172967	5.50322e-11	2.20771e-10	0.00116238	2.25798e-07	2.83487e-08
2002	1.95785e-10	0.999991	2.90724e-07	1.43086e-08	4.60334e-07	3.0183e-11	1.80196e-10	7.79324e-06	2.94051e-07	5.34975e-10
2003	6.66709e-11	0.999899	5.46551e-06	3.66583e-06	1.86837e-06	5.52891e-11	1.41061e-11	8.95421e-05	1.64158e-07	1.54326e-09
2004	3.26184e-07	0.997638	3.51467e-06	1.79313e-07	0.00225156	2.10889e-11	1.59009e-06	9.10897e-05	1.36846e-05	1.47424e-07

Sl. 4.3. Varijabla *probability* za sliku broj 2000.

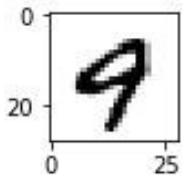
Kao što se može vidjeti, najveća vjerojatnost je ona da se na ulazu pojavila znamenka jedan. U varijabli *y_test* nalaze se znamenke koje bi se trebale pojaviti na izlazu iz neuronske mreže. Na ulaz je poslana 2000. znamenka po redu koja se nalazi u MNIST testnom skupu slika te bi neuronska mreža trebala klasificirati ovu znamenku kao jedinicu što je i slučaj i to se može potvrditi sljedećom slikom (sl. 4.4.).



	0
1997	1
1998	1
1999	1
2000	1
2001	1
2002	1

Sl. 4.4. Varijabla *y_test* za sliku broj 2000.

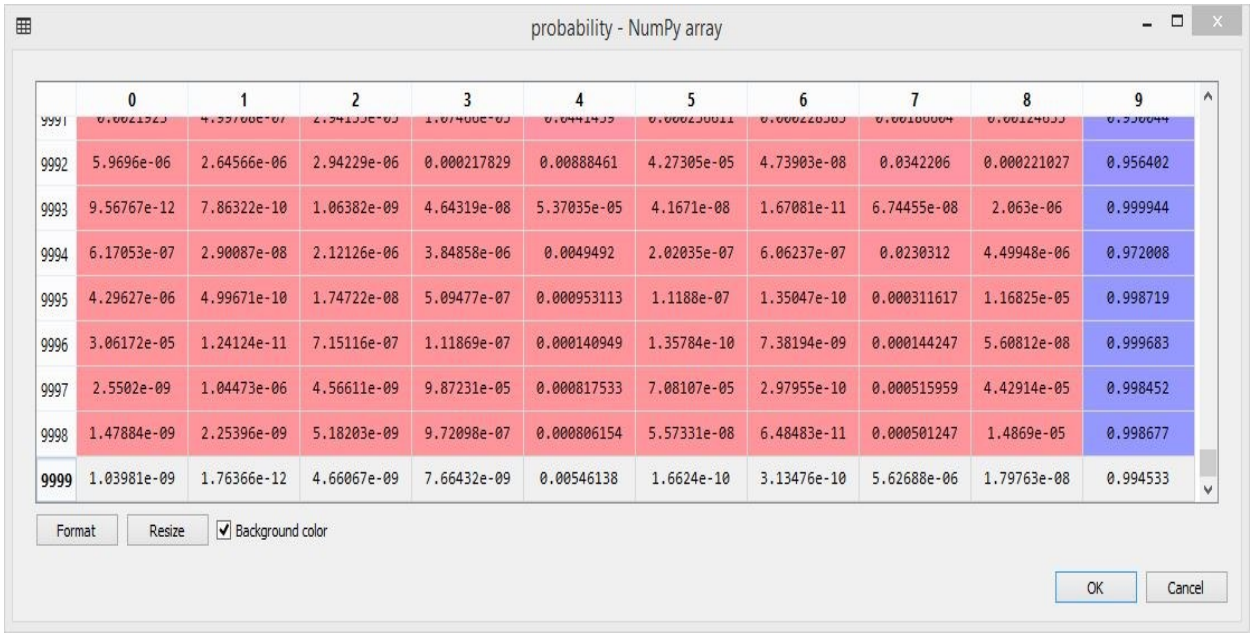
Ako je na ulazu zadnja slika iz testnog skupa odnosno znamenka devet (sl. 4.5.), varijable *r2* i *probability* poprimaju vrijednosti kao na slikama 4.6. i 4.7.



Sl. 4.5. Zadnja slika iz MNIST skupa.

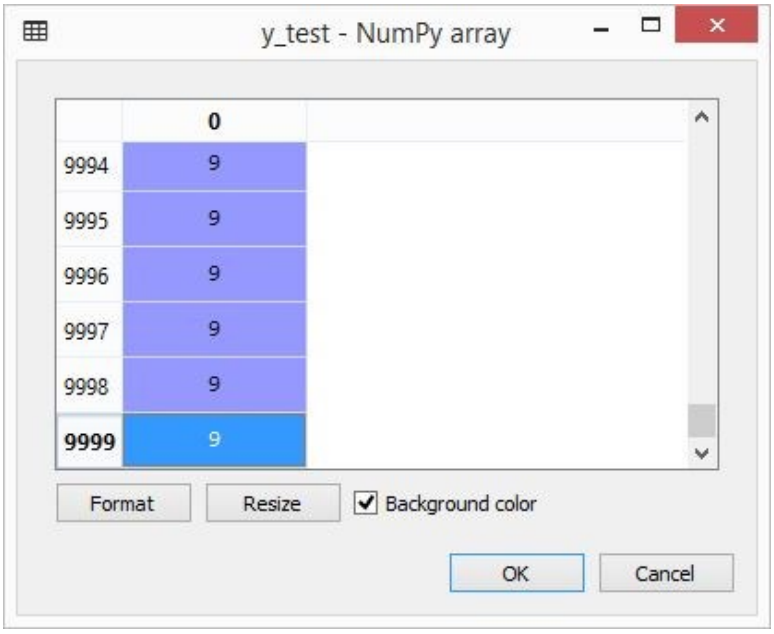
(x)= r2[0]	volatile float	4.85768297e-008
(x)= r2[1]	volatile float	4.85768297e-008
(x)= r2[2]	volatile float	4.85768297e-008
(x)= r2[3]	volatile float	4.85768297e-008
(x)= r2[4]	volatile float	0.0054615289
(x)= r2[5]	volatile float	4.85768297e-008
(x)= r2[6]	volatile float	4.85768297e-008
(x)= r2[7]	volatile float	5.62689684e-006
(x)= r2[8]	volatile float	4.85768297e-008
(x)= r2[9]	volatile float	0.994532526

Sl. 4.6. Varijabla *r2* za zadnju sliku.



Sl. 4.7. Varijabla *probability* za zadnju sliku.

Kao i kod prethodne znamenke, vrijednosti su gotovo iste te je znamenka ispravno klasificirana kao znamenka devet što se može vidjeti na slici 4.8.



Sl. 4.8. Varijabla *y_test* za zadnju sliku.

U oba primjera postoje male razlike u izlazu mreže u mikroupravljaču i one implementirane u *Python* programskom jeziku. Razlog je različita preciznost brojeva s pomičnim zarezom jer Python ima 64 bitni zapis brojeva s pomičnim zarezom dok je u mikroupravljaču 32 bitni zapis. U okviru *Visual Studia* ne postoji to odstupanje u vrijednostima jer se koristi isti zapis brojeva s pomičnim zarezom.

Vrijeme potrebno neuronskoj mreži da klasificira primljenu sliku izmjereno je pomoću brojača vremena mikroupravljača i upisano za svaku sliku u polje prirodnih brojeva. Na slici 4.9. može se vidjeti vrijeme izraženo u milisekundama za nekoliko slika.

Expression	Type	Value
razlika[0]	volatile unsigned int	33
razlika[1]	volatile unsigned int	33
razlika[2]	volatile unsigned int	33
razlika[3]	volatile unsigned int	34
razlika[4]	volatile unsigned int	33
razlika[5]	volatile unsigned int	33
razlika[6]	volatile unsigned int	33
razlika[7]	volatile unsigned int	34
razlika[8]	volatile unsigned int	33
razlika[9]	volatile unsigned int	33
razlika[10]	volatile unsigned int	33
razlika[11]	volatile unsigned int	34
razlika[12]	volatile unsigned int	33

Sl. 4.9. Vrijeme klasifikacije.

Vrijeme je približno isto jer se jednak broj operacija izvršava za svaku sliku.

5. ZAKLJUČAK

U ovom završnom radu je izgrađena, implementirana i testirana neuronska mreža za klasifikaciju rukom pisanih znamenaka. Korištena je neuronska mreža s tri sloja: ulazni, skriveni i izlazni sloj. Ulazni sloj čine 784 neurona koji predstavljaju po jedan piksel slike iz MNIST skupa slika rukom pisanih znamenaka, a izlazni 10 neurona čije vrijednosti predstavljaju vjerojatnosti da se u ulaznom sloju nalazi slika pojedine znamenke. Dio MNIST skupa podataka za trening korišten je za treniranje neuronske mreže. Ta mreža je zatim implementirana u C programski jezik na osobnom računalu u okviru *Visual Studia*, a potom i u mikroupravljač te testirana na preostalih 10 000 slika znamenaka koje su poslane na ulaz neuronske mreže preko serijskog *porta*. Brzina prepoznavanja znamenke neovisna je o tome koja se slika pošalje na ulaz neuronske mreže i iznosi približno 33 milisekunde. Usporedbom rezultata u prethodnom poglavlju može se vidjeti da nema značajne razlike između izlaza neuronske mreže implementirane u programskom jeziku *Python* i mreže implementirane u C programskom jeziku i pokrenute na mikroupravljaču te da klasifikacija rukom pisanih znamenki uz pomoć izgrađene neuronske mreže postiže zadovoljavajuće rezultate.

LITERATURA

[1] https://appliedgo.net/perceptron/ [19.03.2019.]

[2] http://neuralnetworksanddeeplearning.com/chap1.html [19.03.2019.]

[3] https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html [19.03.2019.]

[4] https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 [19.03.2019.]

[5] https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions [19.03.2019.]

[6] http://degiorgi.math.hr/~singer/ui/ui_1415/ch_18a.pdf [19.03.2019.]

[7] https://www.geeksforgeeks.org/regression-classification-supervised-machine-learning/ [19.03.2019.]

[8] https://medium.com/quick-code/regression-versus-classification-machine-learning-whats-the-difference-345c56dd15f7 [19.03.2019.]

[9] https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d [20.03.2019.]

[10] https://www.chipoteka.hr/images/proizvodi/usb-kabel-delock-usb-a-20-m-na-mini-usb-b-20-m-15m-crni-1166-1063.jpg [08.04.2019.]

[11] https://ae01.alicdn.com/kf/HTB1CGzLQFXXXXc0XVXXq6xXFXXXj/PL2303TA-USB-TTL-to-RS232-Converter-Serial-Cable-Module-PL2303TA-Download-Cable-for-Win-XP-VISTA.jpg_640x640.jpg [08.04.2019.]

SAŽETAK

Problem klasifikacije rukom pisanih brojeva riješen je pomoću višeslojne neuronske mreže. Mreža je trenirana u *Python* programskom jeziku na MNIST skupu podataka u kojem se nalaze slike rukom pisanih znamenaka. Nakon faze učenja, neuronska mreža je testirana na testnom skupu podataka. Uz pomoć spremljenih težina i *bias* vrijednosti, ista takva mreža kreirana je u C programskom jeziku. Na samome kraju navedena mreža je implementirana u mikroupravljač s ARM Cortex-M4 jezgrom te su testirane brzina i točnost mreže.

Ključne riječi: klasifikacija, neuronska mreža, MNIST, mikroupravljač

ABSTRACT

Implementation of neural network into the microcontroller with ARM Cortex-M4 core

Problem of classification of handwritten digits is solved with multilayer neural network. Network was trained in Python programming language on MNIST dataset which contains pictures of handwritten digits. After learning phase, neural network was tested on test dataset. With saved weights and biases, the same network is created in C programming language. At the very end, aforementioned network was implemented into the microcontroller with ARM Cortex-M4 core and speed and precision of network were tested.

Keywords: classification, neural network, MNIST, microcontroller

ŽIVOTOPIS

Magdalena Tomašić rođena je u Vinkovcima 16. lipnja 1996. godine, gdje je završila osnovnu i srednju školu. Gimnaziju Matije Antuna Reljkovića završila je 2015. godine, smjer Jezična gimnazija, te iste godine upisuje Elektrotehnički fakultet u Osijeku, danas Fakultet elektrotehnike, računarstva i informacijskih tehnologija, gdje trenutno pohađa treću godinu Preddiplomskog sveučilišnog studija računarstva.

PRILOZI

P. 3.1. Učenje neuronske mreže i zapisivanje njenih parametara u datoteke (*Python* programski jezik)

P. 3.2. Slanje slika preko serijskog *porta* (*Python* programski jezik)

P. 3.3. Implementacija neuronske mreže u mikroupravljač i primanje slika preko serijskog *porta* (programski jezik C)

P. 3.4. Neuronska mreža (programski jezik C) – *NeuralNet.h*

P. 3.5. Neuronska mreža (programski jezik C) – *NeuralNet.c*

P. 3.6. Neuronska mreža (programski jezik C) – *main.c*

Prilozi se nalaze na CD-u.