

8. Umjetne neuronske mreže.

8.1 Cilj Vježbe

Upoznati se s umjetnim neuronskim mrežama i njihovom izgradnjom u Keras aplikacijskom okviru za strojno učenje. Primijeniti znanje stečeno o neuronskim mrežama na problem klasifikacije rukom pisanih brojeva.

8.2 Teorijska pozadina

Područje umjetnih neuronskih mreža (engl. *Artificial Neural Networks* – ANN) intenzivno se razvija posljednjih desetak godina, osobito nakon uspješne primjene dubokih neuronskih mreža (engl. *deep neural network* – DNN) u području obrade slika i računalnog vida. U ovoj vježbi studenti se upoznaju s umjetnim neuronskim, njihovim gradbenim jedinicama i načinom učenja ovakvih modela. Razmatra se problem klasifikacije rukom pisanih brojeva te načinom njegovog rješavanja pomoću neuronskih mreža u Keras aplikacijskom okviru za strojno učenje.

8.2.1 Umjetni neuron

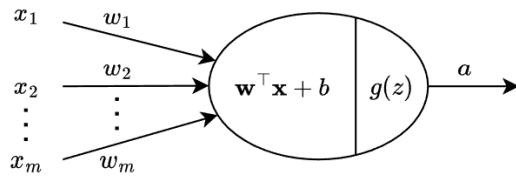
Umjetni neuroni su osnovni gradbeni dijelovi neuronskih mreža. Na slici 8.1 je prikazan umjetni neuron s n ulaznih veličina i izlaznom veličinom a koja predstavlja tzv. aktivaciju. Aktivacija se dobiva na način da se svaka ulazna pomnoži s odgovarajućom težinom w_i te se na sumu dodaje vrijednost pomaka b . Na ovu vrijednost primjenjuje se aktivacijska funkcija g :

$$a = g(z) = g(w_1x_1 + w_2x_2 + \dots + w_mx_m + b) \quad (8.1)$$

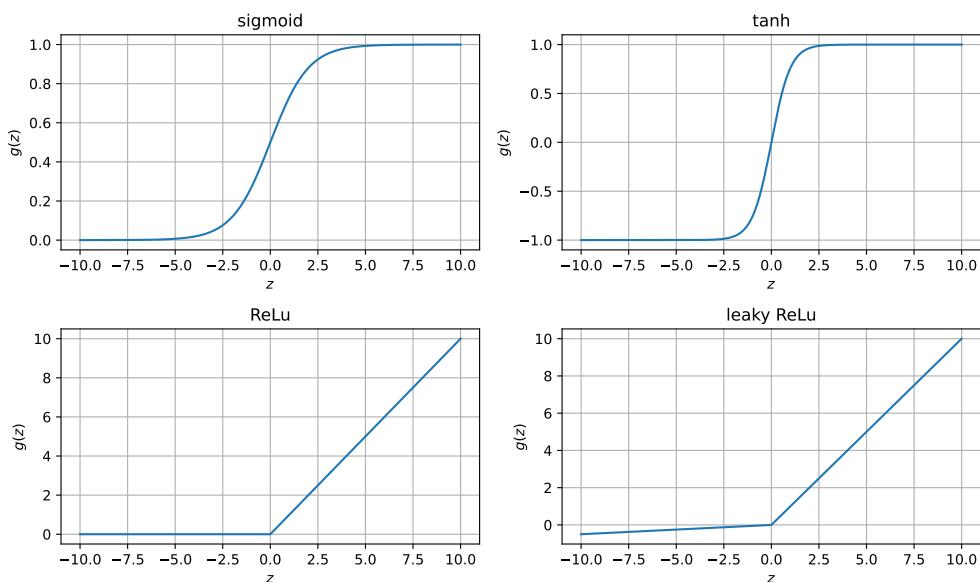
Postoje razne aktivacijske funkcije, a neke od najčešće korištenih su prikazane na slici 8.2.

8.2.2 Višeslojna potpuno povezana neuronska mreža

Više neurona povezuju se u neuronsku mrežu. Jedna od osnovnih umjetnih neuronskih mreža je unaprijedna višeslojna neuronska mreža (engl. *feedforward multilayer neural network*) gdje se



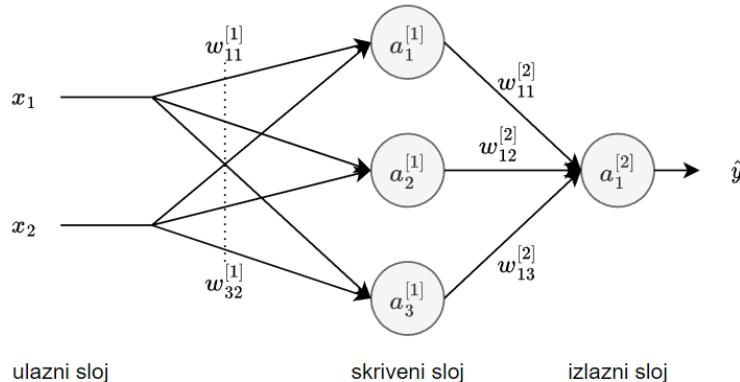
Slika 8.1: Umjetni neuron.



Slika 8.2: Najčešće korištene aktivacijske funkcije.

propagacija signala u mreži odvija u jednom smjeru, od ulaza prema izlazu, bez povratnih veza. Slojevi se nižu jedan iza drugoga pri čemu je svaki neuron sloja povezan sa svim neuronima prethodnog sloja. Ovakvi slojevi nazivaju se potpuno povezani slojevi, a mreža potpuno povezana neuronska mreža (engl. *fully connected neural network*). Na slici 8.3 je prikazana jedna takva potpuno povezana mreža s dvije ulazne veličine, jednim skrivenim slojevima i jednim izlaznim slojem pri čemu su neuroni predstavljeni kružićima. Ova mreža u prvom skrivenom sloju ima tri neurona i jedan neuron u izlaznom sloju. Ako mreža ima više od jednog skrivenog sloja smatra se dubokom neuronskom mrežom. Ulazni primjer se najprije unaprijed propušta (eng. *forward propagation*) kroz prvi skriveni sloj te se dobivaju aktivacije prvog sloja koje se dalje prosljeđuju do izlaznog neurona.

Neuronske mreže već i s manjim brojem neurona mogu opisati različite nelinearne odnose između ulaznih veličina i izlazne veličine. Međutim, neuronske mreže imaju i znatno više parametara (težina) koje je potrebno procijeniti nego što je slučaj kod jednostavnih modela poput linearog regresijskog modela ili logističke regresije. Na primjer, mreža prikazana na slici 8.3 ima ukupno 6 težina i 3 *bias-a* u prvom sloju i 3 težine i 1 *bias* u izlaznom sloju što znači da mreža ima ukupno 13 parametara. Povećavanjem broja neurona u slojevima značajno i raste broj parametara mreže.



Slika 8.3: Primjer unaprijedne višeslojne neuronske mreža.

8.2.3 Izgradnja potpuno povezane neuronske mreže za klasifikaciju

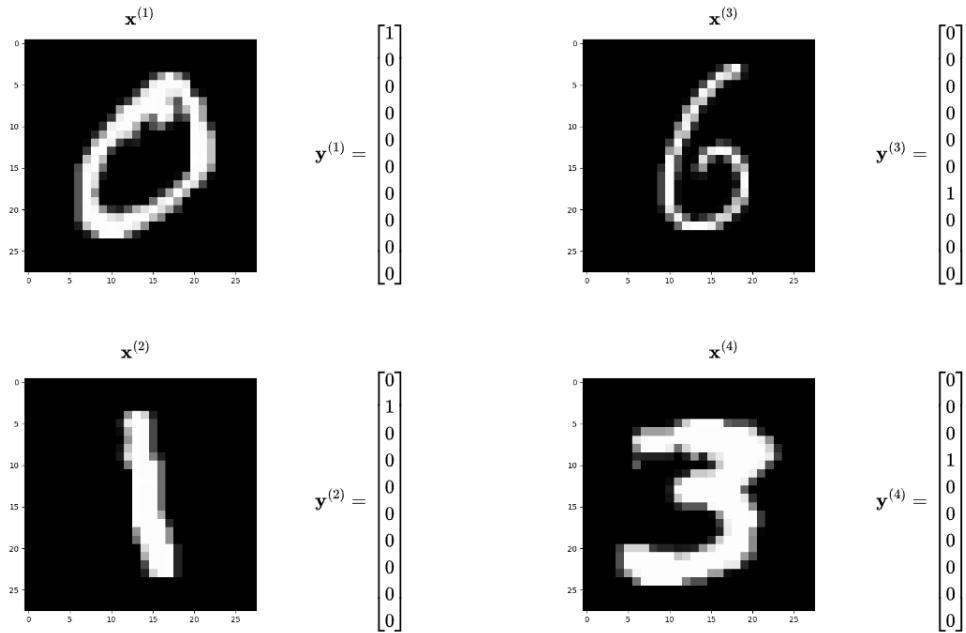
Pod pojmom učenje potpuno povezane neuronske mreže podrazumijeva se strukturiranje neuronske mreže (odabir broja slojeva, odabir broja neurona u pojedinom sloju, odabir tipa aktivacijske funkcije neurona), a zatim i procjena nepoznatih parametara neuronske mreže, tj. težina mreže. Prilikom izgradnje mreže za potrebe klasifikacije, izlazni sloj se konstruira tako da je broj izlaznih neurona jednak broju klasa K , a podatkovni primjeri se kodiraju na način da se izlazna vrijednost (oznaka) kodira pomoću 1-od- K kodiranja. U izlaznom sloju mreže se tada koristi softmax aktivacijska funkcija koja skalira izlaze mreže tako da je njihova vrijednost u intervalu $[0,1]$ i njihova suma jednaka 1.

Razmotrimo problem klasifikacije rukom pisanih znamenki. Za izgradnju modela za klasifikaciju rukom pisanih znamenki na raspolaganju je skup podataka pod nazivom MNIST. Ovaj skup sadrži slike rukom pisanih znamenki koje su pisali zaposlenici u *United States Census Bureau* i američki studenti. Slike su zapisane u sivim tonovima odnosno svaki element slike ima vrijednost u rasponu od 0 do 255. Slike su rezolucije 28x28 piksela. Budući da se na slici nalazi jedna od znamenki od 0 do 9, ovo je problem višeklasne klasifikacije. Četiri različita primjera slika iz ovog podatkovnog skupa i 1-od- K kodiranje izlazne veličine (oznake) za dane slike prikazan je na slici 8.4.

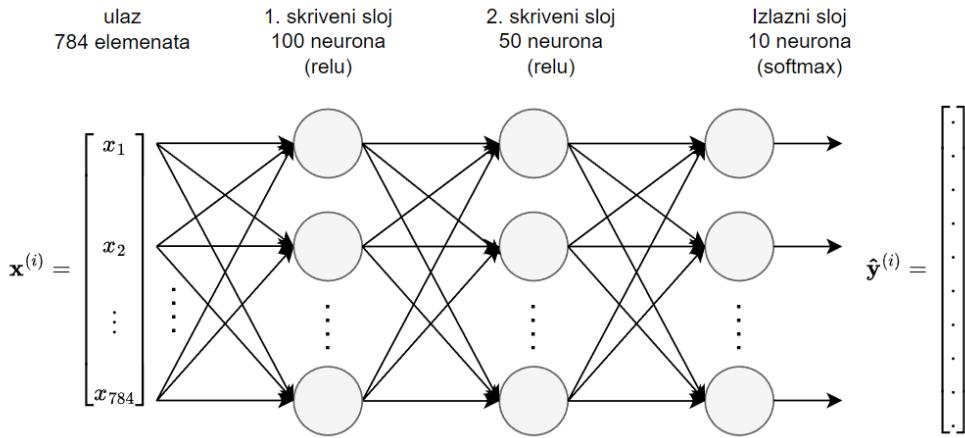
Primjer potpuno povezane mreže koja može poslužiti za rješavanje problema klasifikacije rukom pisanih brojeva dana je na slici 8.4. Ulaz je slika rezolucije 28x28 piksela u sivim tonovima koja se predstavlja obliku 1D vektora dimenzija 784 koji sadrži vrijednosti svjetline svakog elementa slike. Prvi skriveni sloj sadrži 100 neurona, drugi skriveni sloj sadrži 50 neurona, a izlazni sloj sadrži 10 neurona jer ukupno postoji 10 klasa (znamenki). U skrivenim slojevima koristi se primjerice ReLU aktivacijska funkcija dok se u izlaznom sloju koristi softmax aktivacijska funkcija.

Uz definiranu strukturu neuronske mreže, procjena parametra mreže (težina i bias-a) potrebno je odabrati odgovarajuću funkciju gubitka koja će se koristiti prilikom podešavanja parametara mreže. Za navedeni slučaj višeklasne klasifikacije kada su oznake kodirane 1-od- K pristupom koristi se gubitak kategoričke unakrsne entropije (engl. *categorical cross entropy loss*) koji pokazuje iznos pogreške načinjene na jednom primjeru:

$$L(y, \hat{y}) = - \sum_{k=1}^K y_k \log \hat{y}_k \quad (8.2)$$



Slika 8.4: 1-od-K kodiranje oznaka.



Slika 8.5: Potpuno povezana mreža za klasifikaciju rukom pisanih znamenki.

pri čemu je K broj izlaznih neurona u mreži. Stoga, optimalni se parametri mreže pohranjeni u \mathbf{W} i \mathbf{b} dobivaju minimizacijom prosječnog gubitka na danom skupu podataka za učenje:

$$J(\mathbf{W}, \mathbf{b}) = \frac{-1}{n} \sum_{i=1}^n \sum_{k=1}^K y_k^{(i)} \log \hat{y}_k^{(i)} \quad (8.3)$$

pri čemu je n broj podatkovnih primjera u skupu za učenje.

Optimizacija kriterijske funkcije (8.3) provodi se iterativnim numeričkim postupcima koji se zasnivaju na gradijentu. Najjednostavniji način je korištenje gradijentne metode koja osvježava

parametre na način:

$$W := W - \alpha dWb := b - \alpha db \quad (8.4)$$

pri čemu je $\alpha > 0$ i predstavlja stopu učenja (engl. *learning rate*) dok su dW i dB vektori koji sadrže gradijente kriterijske funkcije (8.4) po svakom parametru mreže. Za efikasno računanje gradijenta kriterijske funkcije po svakom parametru mreže koristi se algoritam s povratnim rasprostiranjem pogreške (engl. *backpropagation algorithm* - BP). Ovaka pristup podešavanju parametara mreže se često naziva i *batch* metoda jer koristi sve podatkovne primjere u svakoj iteraciji. Ovo često nije praktično jer algoritam sporo konvergira ili nije moguće koristiti *batch* metodu zbog memorijskih ograničenja. Stoga se koristi *minibatch* stohastička gradijentna metoda koja u svakoj iteraciji za podešavanje parametara samo jedan manji dio podataka za učenje (*minibatch*). Jedna epoha predstavlja iskorištenje svih podatkovnih primjera iz skupa učenja. Stoga, koliko će biti iteracija podešavanja parametara u jednoj epohi ovisi o ukupnom broju podatkovnih primjera u skupu za učenje i koliko ima primjera u jednom *minibatch*-u (engl. *batch size*).

Većina aplikacijskih okvira za strojno učenje poput Keras-a već imaju ugrađene funkcije za inicijalizaciju mreže, kriterijske funkcije, optimizacijske algoritme i sl. pa se korisnik može fokusirati na brzu izgradnju mreže i eksperimentiranje. Osim pripreme podataka, korisnik treba strukturirati mrežu i tipično odabrati:

- broj epoha (engl. *number of epochs*)
 - ovisi o problemu, najčešće se mreža treniran nekoliko desetaka epoha
- veličina minibatch-a (engl. *batch size*)
 - uobičajene vrijednosti su 1, 32, 64 i sl.
- stopa učenja (engl. *learning rate*)
 - uobičajene vrijednosti su 0.01, 0.001 i sl.
- optimizacijski postupak (engl. *optimizer*)
 - osim spomenute stohastičke gradijentne metode postoje i druge poput Adam, Adagrad i sl.

Sa stajališta izgradnje prediktivskog modela važno je da mreža generira „točne“ odgovore i za vrijednosti ulaznih veličina koje se nisu koristile prilikom učenja mreže. Ova karakteristika neuronske mreže naziva se poopćavanje (engl. *generalization*). Loše poopćavanje može biti posljedica pretjeranog učenja. Pretjerano učenje se očituje u tome što mreža s prevelikom fleksibilnošću može opisati različite primjese (šum, stršeće vrijednosti i sl.) iz podataka za učenje, a koje nisu karakteristični za cijelu populaciju podataka. Ovakva mreža pokazuje izvrsne rezultate na podacima na temelju kojih su određeni njeni parametri, međutim, na drugim podacima iz iste populacije ova mreža može znatno grijesiti. Jedan od načina kako sprječiti pretjerano učenje je uvođenje validacijskog skupa podataka na kojem se prate performanse mreže tijekom učenja, ali i razne druge tehnike poput nasumičnog izbacivanja neurona (engl. *dropout*) tijekom učenja, korištenje funkcije gubitka koja uključuje regularizacijski dio i sl.

8.2.4 Keras aplikacijski okvir za strojno učenje

Keras je aplikacijsko programsko sučelje visoke razine (engl. *high-level API*) za izradu neuronskih mreža. Keras API omogućuje vrlo jednostavnu izgradnju i učenje neuronskih mreža. Dva su glavna dijela potrebna za strukturiranje mreže:

- Models API
 - za izradu Keras modela (neuronske mreže)

- Layers API

za kreiranje slojeva koji se dodaju u model

Najjednostavniji način kreiranje Keras modela je korištenjem klase `Sequential` iz *Models API*. Ova klasa sadrži metodu `.add` pomoću koje se dodaju slojevi u mrežu (slijedno kako se pojavljuju). Primjer 8.1 prikazuje stukturiranje mreže sa slike 8.3 pomoću *Keras API*-a. Za definiranje veličine ulaznog sloja i potpuno povezanih slojeva koriste se odgovarajuće klase iz *Layers API*:

- `dense` klasa - kreira potpuno povezani sloj, najvažniji argumenti:
 - `units` – broj neurona u sloju, pozitivna cijelobrojna vrijednost
 - `activation` – tip aktivacijske funkcije
- `input` klasa - kreira ulazni sloj, najvažniji argument:
 - `shape` – tuple tip podatka koji definira dimenziju ulaza (ne uključuje veličinu serije)

Metoda `.summary` daje ispis koji sadrži informacije o kreiranom modelu. Točnije, ispisuje nazive slojeva i njihov tip, izlaznu dimenziju podataka iz svakog sloja, te broj parametra u svakom sloju. Na kraju se ispisuje ukupan broj parametara kreiranog modela. Ispis je prikazan u primjeru 8.2. Primijetite kako se kod ispisa dimenzije izlaznih podataka sloja koristi `None`. Naime, prva dimenzija definira veličinu *minibatch*-a, a model bi u teoriji trebao raditi s bilo kojom veličinom *batch*-a.

■ **Primjer 8.1**

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential()
model.add(layers.Input(shape=(2, )))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(1, activation="sigmoid"))

model.summary()
```

■ **Primjer 8.2**

Layer (type)	Output Shape	Param #
<code>dense</code> (Dense)	(<code>None</code> , 3)	9
<code>dense_1</code> (Dense)	(<code>None</code> , 1)	4
<hr/>		
Total params:	13	
Trainable params:	13	
Non-trainable params:	0	

Model training API je dio *Model API*-a i sadrži metode za konfiguriranje procesa učenja, pokretanje učenja modela, testiranje modela i sl. Najvažnije metode su:

- `.compile`
 - konfigurira model za proces učenja (odabir optimizacijskog postupka, funkcije gubitka i liste metrika na temelju kojih će se evaluirati model tijekom učenja i testiranja)
- `.fit`

- pokreće proces treniranja na skupu za učenje za dani broj epoha i veličinu minibatch-a
- `.predict`
izračunava predikcije mreže za ulazne podatke
- `.evaluate`
vraća vrijednosti prosječne vrijednosti funkcije gubitka i odabranih metrika za dane testne podatke

Opis argumenata svake metode moguće je pronaći u Keras dokumentaciji.

Primjer 8.3 sadrži dio programskog koda koji konfigurira model za proces učenja pomoću metode `.compile`, te se zatim pokreće učenje modela na dostupnim primjerima u numpy poljima `X_train` i `y_train`. Postupak učenja odvija se za zadani broj epoha s definiranom veličinom `batch_size`. Od skupa podataka za učenje se odvaja 10% primjera kao validacijski skup pomoću argumenta `validation_split` metode `.fit`. Nakon svake epohe se na ovom skupu izračunava funkcija gubitka i korištene metrike. Nakon što se odrede parametri modela postupkom učenja, model je moguće testirati na testnom skupu.

■ Primjer 8.3

```
model.compile(loss="categorical_crossentropy",
               optimizer="adam",
               metrics=["accuracy"])

batch_size = 32
epochs = 20
history = model.fit(X_train,
                     y_train,
                     batch_size=batch_size,
                     epochs=epochs,
                     validation_split=0.1)

predictions = model.predict(X_test)

score = model.evaluate(X_test, y_test, verbose=0)
```

Tipičan ispis prilikom učenja mreže prikazan je u primjeru 8.4. Prikazuje se trenutni broj epoha i trenutne serije, trenutna vrijednost prosječnog gubitka i odabrane metrike na skupu podataka za učenje.

■ Primjer 8.4

```
Epoch 1/15
844/844 [=====] - 1s 1ms/step - loss: 0.3088 -
accuracy: 0.9109 - val_loss: 0.1301 - val_accuracy: 0.9628
Epoch 2/15
844/844 [=====] - 1s 1ms/step - loss: 0.1278 -
accuracy: 0.9616 - val_loss: 0.1074 - val_accuracy: 0.9677
Epoch 3/15
844/844 [=====] - 1s 1ms/step - loss: 0.0875 -
accuracy: 0.9735 - val_loss: 0.0928 - val_accuracy: 0.9710
Epoch 4/15
844/844 [=====] - 1s 1ms/step - loss: 0.0672 -
accuracy: 0.9796 - val_loss: 0.0767 - val_accuracy: 0.9782
Epoch 5/15
844/844 [=====] - 1s 1ms/step - loss: 0.0511 -
accuracy: 0.9849 - val_loss: 0.0747 - val_accuracy: 0.9793
```

```

Epoch 6/15
844/844 [=====] - 1s 1ms/step - loss: 0.0411 -
accuracy: 0.9876 - val_loss: 0.0867 - val_accuracy: 0.9782
Epoch 7/15
844/844 [=====] - 1s 1ms/step - loss: 0.0328 -
accuracy: 0.9899 - val_loss: 0.0779 - val_accuracy: 0.9797
Epoch 8/15
844/844 [=====] - 1s 1ms/step - loss: 0.0273 -
accuracy: 0.9911 - val_loss: 0.0828 - val_accuracy: 0.9772
Epoch 9/15
600/844 [=====>.....] - ETA: 0s - loss: 0.0214 -
accuracy: 0.9933

```

Izgrađeni model može se iz radne memorije pohraniti na trajnu memoriju računala pomoću `.save` metode modela. Pomoću funkcije `load_model` moguće je model ponovo učitati u radnu memoriju računala kako je prikazano u primjeru 8.5.

■ Primjer 8.5

```

from keras.models import load_model

model.save("FCN/")
del model

model = load_model('FCN/')
model.summary()

```

8.3 Priprema za vježbu

1. Proučite poglavlje 8.2.
2. Po potrebi dodatno proučite dijelove Keras dokumentacije

8.4 Rad na vježbi

Riješite dane zadatke.

Zadatak 8.4.1 MNIST podatkovni skup za izgradnju klasifikatora rukom pisanih znamenki dostupan je u okviru Keras-a. Skripta `zadatak_1.py` učitava MNIST podatkovni skup te podatke priprema za učenje potpuno povezane mreže.

1. Upoznajte se s učitanim podacima. Koliko primjera sadrži skup za učenje, a koliko skup za testiranje? Kako su skalirani ulazni podaci tj. slike? Kako je kodirana izlazne veličina?
2. Pomoću `matplotlib` biblioteke prikažite jednu sliku iz skupa podataka za učenje te ispišite njezinu označku u terminal.
3. Pomoću klase `Sequential` izgradite mrežu prikazanu na slici 8.5. Pomoću metode `.summary` ispišite informacije o mreži u terminal.
4. Pomoću metode `.compile` podesite proces treniranja mreže.

5. Pokrenite učenje mreže (samostalno definirajte broj epoha i veličinu serije). Pratite tijek učenja u terminalu.
6. Izvršite evaluaciju mreže na testnom skupu podataka pomoću metode `.evaluate`.
7. Izračunajte predikciju mreže za skup podataka za testiranje. Pomoću scikit-learn biblioteke prikažite matricu zabune za skup podataka za testiranje.
8. Pohranite model na tvrdi disk.

Zadatak 8.4.2 Napišite skriptu koja će učitati izgrađenu mrežu iz zadatka 1 i MNIST skup podataka. Pomoću matplotlib biblioteke potrebno je prikazati nekoliko loše klasificiranih slika iz skupa podataka za testiranje. Pri tome u naslov slike napišite stvarnu oznaku i oznaku predviđenu mrežom.

Zadatak 8.4.3 Napišite skriptu koja će učitati izgrađenu mrežu iz zadatka 1. Nadalje, skripta treba učitati sliku `test.png` sa diska. Dodajte u skriptu kod koji će prilagoditi sliku za mrežu, klasificirati sliku pomoću izgrađene mreže te ispisati rezultat u terminal. Promijenite sliku pomoću nekog grafičkog alata (npr. pomoću Windows Paint-a nacrtajte broj 2) i ponovo pokrenite skriptu. Komentirajte dobivene rezultate za različite napisane znamenke.

8.5 Izvještaj s vježbe

Kao izvještaj s vježbe prihvaća se web link na repozitorij pod nazivom `OSU_LV`.