Neuronske mreže - Prepoznavanje znamenki

Autor - Dominik Horvat

Uvod

U ovom projektu biti će predstavljena implementacija umjetne neuronske mreže s visokom točnošću prepoznavanja znamenaka brojeva između 0 i 9. Neuronske mreže (umjetne) vrlo su zanimljiv koncept u računarstvu i strojnom učenju jer se ljudski mozak smatra kompleksnim "strojem". Postavlja se pitanje kako takav način razmišljanja kao i pogled na svijet preslikati na stroj. Općenito, ljudski mozak nema problema s prepoznavanjem neke brojke, ali pitanje je kako postići istu efikasnost prepoznavanja kod računalnog stroja. Impresivno je kako za isti broj napisan na razne načine možemo točno utvrditi da je riječ o tom broju. Je li tako i kod računala? Ovdje glavnu ulogu imaju pikseli i njihova istaknutost koja doprinosi uvelike raspoznavanju određenog broja. Priloženi programski kod bit će detaljno opisan. Nakon implementacije umjetne neuronske mreže bit će prezentirana njezina efikasnost.

Potrebni podaci i materijali

Prije početka izrade nekog projekta potrebna je neka baza, materijali ili podaci na kojima se sve zasniva. Za ovaj projekt korištena je baza podataka pod imenom MNIST - "*Modified National Institute of Standards and Technology database*". Ova baza podataka obuhvaća 60 000 slika znamenaka od 0 do 9. Svaka je slika dimenzije 28x28 piksela i znamenke su bijele boje na crnoj podlozi. Zasigurno se pitate kako je to postignuto? Istraživači koji stoje iza ove baze podataka prikupili su 60 000 slika ovakvog tipa i platili nasumičnim osobama da ih označe. Na primjer, za znamenku koja predstavlja broj nula, pridodali su oznaku (eng. *label*) 0. Objavom baze podataka MNIST postaje globalno popularna i danas je kao "Hello world" za strojno učenje.

Biblioteke za rad

Kao što podnaslov govori potrebne su određene biblioteke kako bi se umjetna neuronska mreža uspješno implementirala i testirala. Potrebne biblioteke su:

- **Matplotlib Pyplot** – ovu bibiloteku koristimo za vizualizaciju / prikaz brojeva, a *.pyplot* čini da Matplotlib radi slično kao MATLAB
- **Numpy** – ne treba pretjerano objašnjavati. Ukratko, omogućuje mnoge matematičke funkcije i rad s višedimenzionalnim nizovima i matricama
- **cv2** – „Computer Vision“, omogućuje rad sa slikama u Pythonu. Služi za učitavanje, prikaz, manipulaciju i filtriranje slika
- **Tensorlow** – olakšava ljudima raznih stupnja poznavanja programskih jezika stvaranje modela strojnog učenja za računalne uređaje. Ova biblioteka ističe se za razvoj aplikacija čija je pozadina neuronska mreža. U ovom projektu ta biblioteka bit će najvažnija jer će omogućiti mnoge druge funkcije i podatke. Upravo ova

biblioteka sadrži i bazu projekta, a to je već spomenuti MNIST. Važno je napomenuti kako se aplikacije koje se temelje na TensorFlowu mogu pokrenuti na lokalnim računalima, oblaku, Andoridu, i tako dalje. Ukratko, u ovom projektu ovo omogućava dio sa strojnim učenjem

- **os** – Pythonov model koji omogućuje stvaranje datoteka i direktorija, te obavljanje raznih operacija nad njima
- **PIL** - slično kao i cv2, služi za rad sa slikama

Uvodimo ih na sljedeći način:

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import tensorflow as tf
import os
from PIL import Image

#Takodjer uvodimo i funkciju Dropout iz keras.layers
from keras.layers import Dropout
```

Napomena: U slučaju nedostatka nekih paketa / biblioteka poželjno ih je preuzeti na sljedeći način:

- ako je riječ o pythonu u odgovarajući terminal potrebno je napisati "pip install tensorflow"
- ako je riječ o Jupyter notebook "!pip install tensorflow"

Dohvaćanje podataka

Za potrebe ovog projekta iskoristit ćemo bazu podataka pod nazivom MNIST koja obuhvaća 60 000 crteža znamenaka od 0 do 9 veličine 28x28 piksela. Kako bi se dohvatili podaci iz te baze definiramo sljedeću varijablu:

```
mnist=tf.keras.datasets.mnist
```

U varijabli `mnist` pohranjeni su željeni podaci. Pomoću `.load_data()` razdvajamo `mnist` na dva para. Preciznije, `mnist.load_data()` vraća podatke za obuku uz oznake i podatke za testiranje uz odgovarajuće oznake. Podaci za obuku koriste se pri obučavanju modela, u ovom slučaju umjetne neuronske mreže. S druge strane, podaci za testiranje koriste se za procjenu modela kako bi se dobio uvid kolika je uspješnost izvođenja na podacima koje model prije nije vidio. Ukratko, ovo je ideja obuke i testiranja.

```
(x_train,y_train), (x_test,y_test) = mnist.load_data()
```

Linijom programskog koda iznad dobivamo dva para koja se sastoje od tuplea (generička struktura koja predstavlja uređenu n-torku). Ispisom svake strukture dobiva se uvid u njezin sadržaj.

```
print(x_train.shape)
```

```
(60000, 28, 28)
```

Numpy niz nijansi sive (*grayscale*) slika za obuku gdje je 28 veličina u pikselima. Dobiveni broj 60000 ukazuje na broj slika.

```
print(y_train.shape)
```

```
(60000,)
```

Riječ je o Numpy nizu oznaka znamenki - cijeli brojevi u rasponu od 0 do 9 oblika ispisanog iznad za podatke o obuci.

```
print(x_test.shape)
```

```
(10000, 28, 28)
```

Numpy niz nijansi sive (*grayscale*) slika za testiranje gdje je 28 veličina u pikselima. Dobiveni broj 10000 predstavlja broj slika.

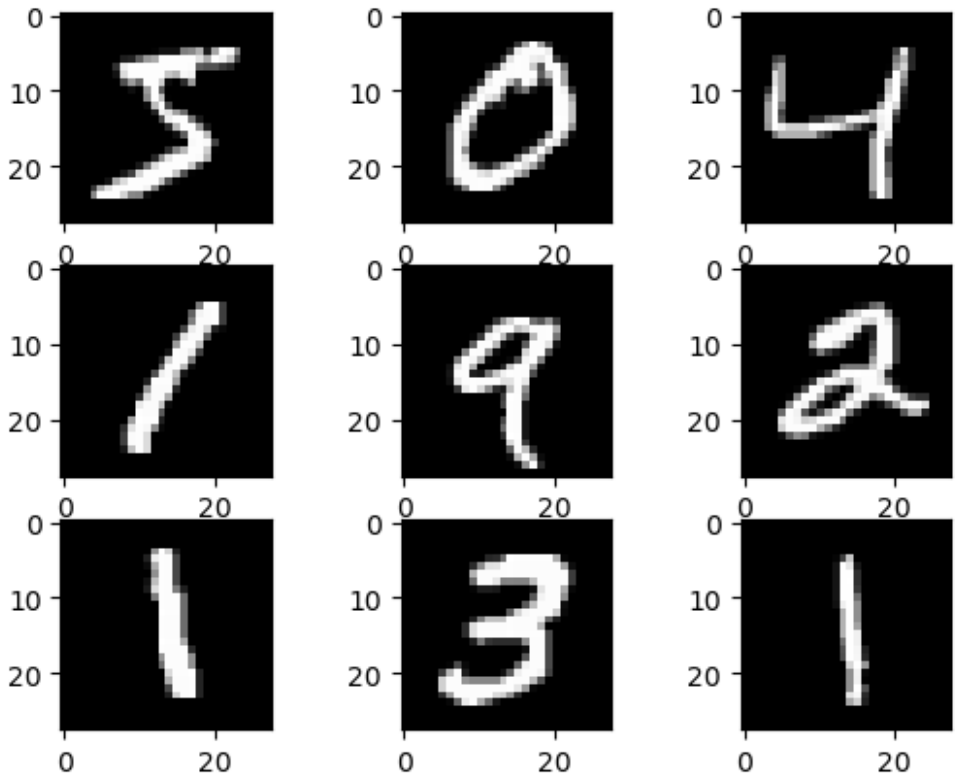
```
print(y_test.shape)
```

```
(10000,)
```

Riječ je o Numpy nizu oznaka znamenki - cijeli brojevi u rasponu od 0 do 9 oblika ispisanog iznad za podatke o testiranju.

Napomenimo da se vrijednosti piksela kreću od 0 do 255 gdje vrijednost 0 predstavlja crnu boju, a vrijednost 255 bijelu boju. Brojevi između 1 i 254 su različite nijanse sive boje od tamnije prema svijetlijoj. Iako se bijela i crna boja ne smatraju sivom, u globalu govorimo o ukupno 256 nijansi sive boje. Kako bi se dobio uvid u ove slike i izgled istih u ovoj bazi podataka implementiran je idući kod koji će ispisati nekoliko njih. Primjerice, prvih 9 slika ove baze podataka.

```
for i in range(9):
    plt.subplot(int('33' + str(i+1)))
    plt.imshow(x_train[i],cmap=plt.get_cmap('gray'))
plt.show()
```



Iz slike je jasno kako se prvih 9 slika odnosi na brojeve 5, 0, 4, 1, 9, 2, 1, 3, 1. Ako se implementira kod koji ispisuje prvih 9 članova niza `y_train` trebao bi se dobiti ispis upravo ovih 9 brojeva.

```
for i in range(9):
    if i == 8:
        print(y_train[i])
    else:
        print(y_train[i],end="")
        print(",",end="")
```

```
5,0,4,1,9,2,1,3,1
```

Dakle, kao što je prethodno navedeno `y_train` označava Numpy niz oznaka slika čija je vrijednost jednaka broju znamenke koju slika prikazuje.

Oblikovanje, promjena tipa i normalizacija podataka

Zbog lakšeg načina implementiranja i obuke umjetne neuronske mreže važno je sljedećih nekoliko koraka kao dio pripreme. Trenutno je `x_train` oblika (60000,28,28), a `x_test` oblika (10000,28,28). Preciznije riječ je o nizu od 60000 elemenata čije su vrijednosti 28x28. Potrebno je preoblikovati trenutni format na način da novi oblik bude oblika (60000,784). Jasno je da 784 dobivamo na način da pomnožimo 28 s 28 ($28 \cdot 28$). Ovo je iznimno važno jer svaki piksel predstavlja jedan neuron u ulaznom sloju neuronske mreže. Dakle, riječ je o 784 neurona. Ovo se postiže na sljedeći način:

```
pozeljno=784
x_train = x_train.reshape(60000,pozeljno)
x_test = x_train.reshape(60000,pozeljno)
```

Uz prethodne linije koda potrebno je podatke pretvoriti u **float32** zbog korištenja 32-bitne preciznosti u uvježbavanju neuronske mreže. Za promjenu tipa podataka primjenjujemo `.astype()` gdje se unutar zagrada u jednostrukim navodnicima navodi željeni tip podatka.

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

Također, podaci se moraju normalizirati tako da ulaz, to jest vrijednost piksela koji označava neuron bude u intervalu $[0,1)$. Trenutne vrijednosti svakog piksela kreću se od 0 do 255. Kako bi se vrijednosti dovele u željeni intervali dovoljno je `x_train` i `x_test` podijeliti brojem 255.

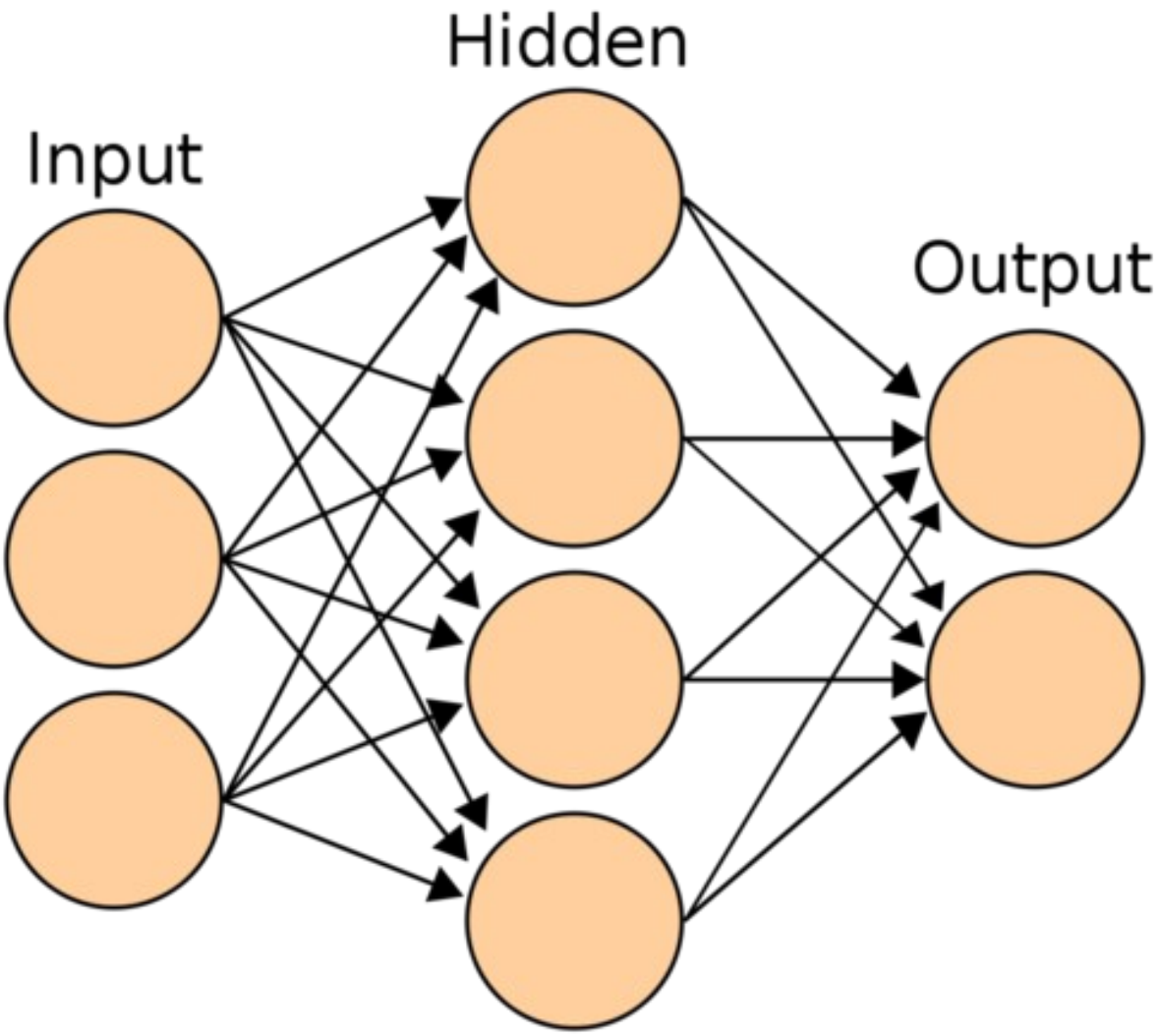
```
x_train /= 255
x_test /= 255
```

Sada su vrijednosti svakog piksela (neurona) između 0 i 1 što omogućava jednostavnije izvođenje. Preostala stavka koju je potrebno učiniti prije stvaranja neuronske mreže je *One-Hot* kodiranje. Ovaj način omogućava da oznake brojeva prikažemo pomoću vektora nula i jedinica. Na primjer, za sliku koju prepoznajemo kao broj 2, uz nju je sadržana pripadajuća oznaka vrijednosti 2. Primjenjujući ovu tehniku na oznaku 2, nova *One-Hot* reprezentacija za tu oznaku je $[0,0,1,0,0,0,0,0,0,0]$. Postoji i *One-Cold* kodiranje gdje je obratan slučaj. Jedna nula, a sve ostale vrijednosti su jedinice. Kao što se moglo naslutiti *One-Hot* kodiranjem djelovat će se na `y_train` i `y_test`. Koristi se funkcija `.to_categorical` koja se nalazi u *TensorFlow* biblioteci. `

```
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

Umjetna neuronska mreža

Kada je riječ o neuronskoj mreži zamišljamo hrpu neurona koji su međusobno povezani. Na taj način se omogućuje protok informacija između njih. Isto tako je i s umjetnom neuronskom mrežom. Neuroni su grupirani u slojeve. Također, kako je riječ o mreži, neurone možemo nazvati čvorovima. Postoje tri vrste sloja: ulazni, sakriveni i izlazni (redom eng. *input, hidden, output layer*). Sakrivenih slojeva može biti, ali i ne mora. Dakle, broj sakrivenih slojeva varira između nula i beskonačno. Slika ispod prikazuje osnovnu neuronsku mrežu koja se sastoji od ulaznog, jednog sakrivenog i izlaznog sloja.



Također, može se uočiti kako je prethodni sloj povezan s idućim na način da jedan neuron iz prethodnog sloja ima povezanost sa svakim iz idućeg sloja. Svaki sloj je povezan s idućim, to jest potpuno su povezani. Postoje još neki načini povezivanja, ali ovakvo povezivanje je najviše korišteno u primjeni. Formalniji naziv jedne grane je težina koja ima veliku vrijednost u aktivaciji. Ulazni sloj označava ulazne vrijednosti koje se prosljeđuju neuronskoj mreži. Sakriveni sloj prima vrijednosti iz prethodnog sloja, upotrebljava svoju "magiju" i prenosi neuronske vrijednosti na sljedeći sloj. Izlazni sloj je sličan kao i sakriveni, prima vrijednosti, ali neuronske vrijednosti su sada korištene kao izlaz i ne prosljeđuju se dalje. U ovom projektu umjetna neuronska mreža obuhvaćat će nekoliko sakrivenih slojeva radi veće efikasnosti modela.

Stvaranje umjetne neuronske mreže

Za početak treba inicijalizirati prazan model. Inicijalizacija praznog modela postiže se funkcijom `.Sequential()` iz biblioteke *TensorFlow*.

```
model = tf.keras.Sequential()
print("Uspjesno!")
```

```
Uspjesno!
```

Idući korak je dodavanje prethodno navedenih slojeva. Koristi se funkcija `.add()` s odgovarajućim parametrima. U ovom slučaju kreiranja neuronske mreže, unutar zagrada funkcije `.add()` navodi se `tf.keras.layer.Dense()`. Ovime se dobiva potpuno povezan sloj. Ujedno ovakav sloj se kao što je i prije navedeno koristi u višeslojnim modelima. Zbog prethodnih prilagodbi podataka za rad u podnaslovu "Oblikovanje, promjena tipa i normalizacija podataka" prvom funkcijom stvaramo zajedno ulazni i prvi sakriveni sloj. Parametri unutar prvog `.Dense()` bit će definirani redom: koliko prvi sakriveni sloj ima neurona, ulazni sloj s odgovarajućim oblikom i aktivacijska funkcija.

```
model.add(tf.keras.layers.Dense(512, input_shape=(784,), activation =
'relu'))
```

Jasno je da za ulazni sloj odabiremo 784 neurona zbog prethodnih tvrdnji kako je slika u MNIST bazi podataka dimenzija 28x28 piksela. Također, radi odgovarajućih prilagodbi podataka navodimo `input_shape=(784,)`. Za prvi sakriveni sloj odabrano je 512 neurona, a aktivacijska funkcija je *ReLU*. Postoji mnogo aktivacijskih funkcija među kojima je i *Sigmoid*.

Pokazalo se da je funkcija *ReLU - Rectified Linear Unit* puno efikasinija za treniranje modela.Nakon prvog sakrivenog sloja kao uvodi se jedan "trik". Dodan je `Dropout()` s vrijednošću 0.4, to jest `model.add(Dropout(0.4))`. Dropout pospješuje model i pomaže u izbjegavanju *overfittinga* - najveći neprijatelj strojnog učenja. To je sitacija u kojoj algoritam ne generalizira dobro nove podatke. Sve to dovodi do slabe procjene i netočnosti u predviđanju podataka. Poželjno je vrijednost unutar zagrada postaviti između 0.2 do 0.5. Za potrebe projekta i kako bi model bio što uspješniji odabrana je vrijednost 0.4.

```
model.add(Dropout(0.4))
```

Zatim su dodana još tri sakrivena sloja s dropoutom nakon svakoga, od kojih sljedeći sakriveni sadrži 512 neurona, a preostala dva po 256 neurona.

```
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(Dropout(0.4))
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(Dropout(0.4))
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(Dropout(0.4))
```

Umjetna neuronska mreža završava izlaznim slojem. Intuitivno je kako se izlazni sloj sastoji od 10 neurona (čvorova) koji predstavljaju po jednu znamenku. Na primjer, prvi neuron nulu, drugi jedinicu, i tako dalje. Jedina veća razlika je u aktivacijskoj funkciji koja sada više nije *relu*, već *softmax*. Izlazi svih 10 neurona se zbrajaju u sumu koja je jednaka 1. Ova funkcija ukratko daje vjerojatnosti svakog broja da bude rješenje. Najveća vjerojatnost, to jest vrijednost predstavljat

će rješenje koje označava određen neuron. Taj neuron predstavlja pretpostavku za broj koji se nastoji pogoditi s priložene slike.

Napomena: sve vrijednosti neurona (čvorova) u izlaznom sloju su nenegativne

```
model.add(tf.keras.layers.Dense(10, activation = 'softmax'))
```

Sastavljanje, obuka i evaulacija modela

Potrebno je sastaviti model. Ovo se izvodi pomoću funkcije `.compile()` koja prima tri parametra:

- `optimizer='...'`
- `loss='...'`
- `metrics='...'`

Važno je odabrati dobar optimizacijski algoritam koji čini ključnu komponentu koja služi kao pomoć neuronskoj mreži za učinkovito učenje. Izabiremo optimizacijski algoritam pod nazivom "Adam" - primarni cilj odnosi se na stabilizaciju procesa obuke i pomaže neuronskoj mreži pri konvergenciji do optimanlog rješenja. Ujedno, ovo je najčešće korišten optimizacijski algoritam za ovakve tipove zadataka. Funkcija gubitka (*loss*) kao namjenu ima izračunati količinu koju program treba minimizirati za vrijeme trajanja obuke. Funkcija gubitka pod nazivom *categorical_crossentropy* primjenjuje se za modele klasifikacije s više klasa gdje postoje dvije ili više izlaznih oznaka. Također, ako je izlazna oznaka u obliku cijelog broja onda se pretvara u kategoričko kodiranje koristeći *keras.utils to_categorical* metodu. *Metrics=['accuracy']* izračunava koliko često je pretpostavka jednaka oznaci.

```
model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

Nakon prethodnog koraka slijedi obuka modela. Model će biti obučen na podacima za obuku. Ovdje definiramo epohe (`epochs`) i veličinu serije (`batch_size`). Epohe označavaju koliko će puta model proći kroz skup podataka za obuku. Veličina serije (*batch size*) označuje broj instanci obuke koje će se prikazati modelu prije ažuriranja težine. Težina se odnosi na povezanost, to jest na grane s kojima su neuroni povezani. Implementacija programskog koda provodi se pomoću slijedećeg retka:

```
training = model.fit(x_train, y_train, batch_size=256, epochs=32)

Epoch 1/32
235/235 [==============================] - 6s 21ms/step - loss: 0.5072
- accuracy: 0.8387
Epoch 2/32
235/235 [==============================] - 5s 22ms/step - loss: 0.1882
- accuracy: 0.9466
Epoch 3/32
235/235 [==============================] - 5s 22ms/step - loss: 0.1424
- accuracy: 0.9593
Epoch 4/32
```

```
235/235 [==============================] - 5s 21ms/step - loss: 0.1158
- accuracy: 0.9674
Epoch 5/32
235/235 [==============================] - 5s 22ms/step - loss: 0.1016
- accuracy: 0.9709
Epoch 6/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0882
- accuracy: 0.9741
Epoch 7/32
235/235 [==============================] - 5s 22ms/step - loss: 0.0835
- accuracy: 0.9757
Epoch 8/32
235/235 [==============================] - 5s 22ms/step - loss: 0.0778
- accuracy: 0.9772
Epoch 9/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0689
- accuracy: 0.9794
Epoch 10/32
235/235 [==============================] - 5s 22ms/step - loss: 0.0620
- accuracy: 0.9819
Epoch 11/32
235/235 [==============================] - 5s 22ms/step - loss: 0.0592
- accuracy: 0.9826
Epoch 12/32
235/235 [==============================] - 5s 22ms/step - loss: 0.0573
- accuracy: 0.9833
Epoch 13/32
235/235 [==============================] - 5s 22ms/step - loss: 0.0549
- accuracy: 0.9845
Epoch 14/32
235/235 [==============================] - 5s 21ms/step - loss: 0.0519
- accuracy: 0.9845
Epoch 15/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0490
- accuracy: 0.9853
Epoch 16/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0467
- accuracy: 0.9863
Epoch 17/32
235/235 [==============================] - 5s 22ms/step - loss: 0.0442
- accuracy: 0.9866
Epoch 18/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0413
- accuracy: 0.9878
Epoch 19/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0409
- accuracy: 0.9876
Epoch 20/32
235/235 [==============================] - 5s 22ms/step - loss: 0.0378
```

```
- accuracy: 0.9887
Epoch 21/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0388
- accuracy: 0.9880
Epoch 22/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0370
- accuracy: 0.9891
Epoch 23/32
235/235 [==============================] - 5s 22ms/step - loss: 0.0367
- accuracy: 0.9894
Epoch 24/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0304
- accuracy: 0.9909
Epoch 25/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0343
- accuracy: 0.9897
Epoch 26/32
235/235 [==============================] - 5s 21ms/step - loss: 0.0344
- accuracy: 0.9903
Epoch 27/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0322
- accuracy: 0.9903
Epoch 28/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0315
- accuracy: 0.9908
Epoch 29/32
235/235 [==============================] - 5s 21ms/step - loss: 0.0313
- accuracy: 0.9904
Epoch 30/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0322
- accuracy: 0.9905
Epoch 31/32
235/235 [==============================] - 5s 23ms/step - loss: 0.0302
- accuracy: 0.9908
Epoch 32/32
235/235 [==============================] - 5s 22ms/step - loss: 0.0279
- accuracy: 0.9919
```

I za kraj evaluacija modela. Nažalost u Jupyter notebooku ovo neće proći, ali programski kod za python je sljedeći:

```
test_loss,test_acc = model.evaluate(x_test, y_test)
#loss & accuracy:
print(test_loss)
print(test_acc)
```

Za *accuracy* u ispisu dobivamo više od 0.98, to jest više od 98% što je vrlo poželjno. Općenito, na internetu se mogu naći mnogobrojne implementacije gdje je točnost između 91% - 97%. Također, postoje i mnogobrojna poboljšanja kodova koja rezultiraju većim postocima.

Preostaje spremiti model kako ne bi trebalo stalno vršiti obuku pri pokretanju programa i kako bi se mogao koristiti u raznim primjenama (na primjer ako želimo stvoriti aplikaciju koja prepoznaje znamenke brojeve od 0 do 9).

```
model.save('mreza_jupyter.keras')
print("Model je uspjesno spremljen!")
```

```
Model je uspjesno spremljen!
```

Ovo je sve za kraj ovog podnaslova u kojem imamo naš model koji može prepoznati znamenke brojeva od 0 do 9 na slikama. Iako trenutno nije nužno jer je sve u jednoj *.ipynb* datoteci, kreirani model se pozvati sljedećom linijom koda:

```
model=tf.keras.models.load_model('mreza_jupyter.keras')
print("Model je uspjesno ucitan!")
```

```
Model je uspjesno ucitan!
```

Prilagodba rukopisnih brojeva

Za potrebe testiranja umjetne neuronske mreže (modela koji je prethodno kreiran) pomoću grafičkog tableta u aplikaciji "Microsoft Whiteboard" napisano je 50 znamenaka brojeva od 0 do 9 od kojih je svaka znamenka napisana 5 puta na razne načine. Znamenke su pisane crnom

bojom na bijeloj podlozi. Na primjer:

Iako je navedeno da MNIST baza podataka sadrži 60000 znamenaka brojeva napisanih bijelom bojom na crnoj podlozi, ovdje to nije bio slučaj kreiranja znamenaka. Općenito, u svakodnevici ne pišemo bijelom bojom po crnoj podlozi, već obratno. Prirodno je pisati olovkom po bijelom papiru. Ne treba birnuti o tome jer sve slike rukopisnih znamenaka koje su napravljene uz grafički tablet biti će prilagođene znamenkama iz MNIST-a. Brojke koje su napisane na grafičkom tabletu i izvezene kao slike puno su većih dimenzija, nego slike u MNIST bazi podataka. Također, ovo ne stvara pretjerani problem jer posebnim programskim kodom možemo svaku sliku preoblikovati tako da njezine dimenzije budu 28x28 pikesela. Implementacija ovog programskog koda je sljedeća:

```
broj_slike=1

while os.path.isfile(f"znamenke/0_do_4/znamenka{broj_slike}.png"):
    original_slika =
Image.open(f"znamenke/0_do_4/znamenka{broj_slike}.png")
    # Promeni dimenzije na 28x28
    resized_slika = original_slika.resize((28, 28))

    # Sačuvaj promenjenu sliku
    resized_slika.save(f"testiranje/0_do_4/podesena{broj_slike}.png")
    broj_slike +=1

print("Uspjesno prilagodjene slike!")
```

```
Uspjesno prilagodjene slike!
```

Prepoznavanje rukopisnih brojeva

Kao kruna ovog projekta pred neuronsku mrežu bit će iznesene prethodno navedene slike znamenaka od 0 do 9. Cilj je ispisati odgovarajuće znamenke s odgovarajućim pretpostavkama od umjetne neuronske mreže. Prije samog ispisa trebamo za svaku sliku znamenke u predviđenom direktoriju pomoću neuronske mreže pogoditi o kojem se broju, to jest o kojoj je znamenki riječ. Kako bi olakšali kasniju implementaciju vezanu za prikaza brojeva i odgovarajućih pretpostavki na jednoj slici, ideja implementacije predviđanja znamenki je sljedeća. Prvo stvaramo prazno polje:

```
za_testiranje = []
```

Zatim početno stavljamo varijablu `broj_slika` na 1 (pretpostavit ćemo da postoji barem jedna slika u našem direktoriju).

```
broj_slika=1
```

Sve dok postoji neka slika u direktoriju učitamo tu sliku u sivoj skali boja te spremamo kao `np.array` u polje sivih skala boja. Dodatno naglašavamo kako je svaki piksel intenzitet svijetlosti sive boje da osiguramo učitavanje slike u sivoj nijansi boje.

```
while os.path.isfile(f"testiranje/0_do_4/podesena{broj_slika}.png"):
    img = cv2.imread(f"testiranje/0_do_4/podesena{broj_slika}.png",
cv2.IMREAD_GRAYSCALE)
    za_testiranje.append(np.array(img))
#na kraju povećamo varijablu broj_slika za 1
    broj_slika +=1
```

Dolazimo do glavnog djela programskog koda u kojem će umjetna neuronska mreža pretpostaviti o kojoj znamenki sa slike je riječ. Bitne stavke su:

- prilagoditi boju MNIST bazi podataka

- prilagoditi oblik načinu prepoznavanja znamenki
- vrijednosti piksela dovesti u interval $[0, 1)$
- pomoću funkcije `.predict()` pogoditi o kojoj je znamenci riječ
- funkcijom `np.argmax()` pronaći najveći indeks, to jest broj koji odgovara znamenki

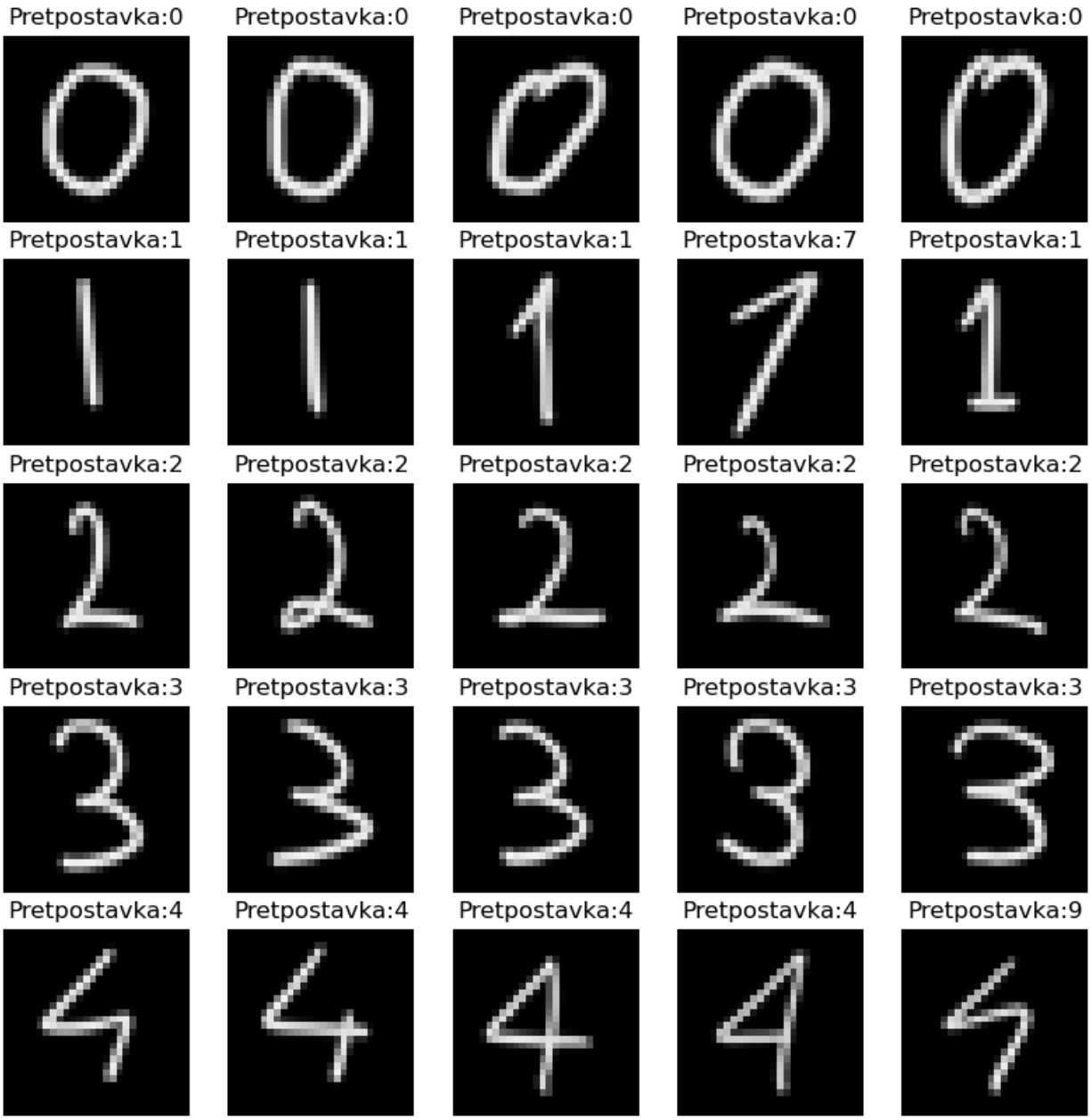
Implementacija programskog koda je sljedeća:

```
temp_array = np.invert(np.array(za_testiranje))
array_flat = temp_array.reshape(len(temp_array), 28*28)
array_flat = array_flat / 255
za_pretpostaviti = model.predict(array_flat)
pretpostavka = [np.argmax(i) for i in za_pretpostaviti]
```

```
1/1 [==============================] - 0s 65ms/step
```

Preostaje kreirati ispis učitanih slika s njihovim pretpostavkama. Implementacija je sljedeća:

```
fig, axs = plt.subplots(5, 5, figsize=(10, 10))
# Iteriraj kroz slike
i=0
for j in range(1, broj_slika):
    # Kreiraj naziv slike

    putanja_slike = os.path.join("testiranje/0_do_4",
f"podesena{j}.png")

    # Učitaj sliku
    img = Image.open(putanja_slike)

    # Invertiraj boje
    inverted_img = Image.eval(img, lambda x: 255 - x)

    # Odredi indekse redaka i stupaca za prikaz
    redak = (j - 1) // 5
    stupac = (j - 1) % 5
    # Prikazi sliku na odgovarajućem mjestu u subplotu
    axs[redak, stupac].imshow(inverted_img, cmap='gray')
    axs[redak, stupac].set_title("Pretpostavka:"
+str(pretpostavka[i]))
    axs[redak, stupac].axis('off')  # Isključi oznake osi
    i=i+1
```



Nakon što smo gotovi s procesom prepoznavanja znamenki, slijedi brisanje testnih slika kako bi se omogućila daljnja upotreba direktorija u koji želimo spremati prilagođene slike određene dimenzije 28x28 piksela za daljnja testiranja.

```
za_obrisati=1

#uklanjamo sve podesene slike velicina 28x28 piksela
while os.path.isfile(f"testiranje/podesena{za_obrisati}.png"):
    datoteka = (f"testiranje/podesena{za_obrisati}.png")
    os.remove(datoteka)
    za_obrisati +=1
```

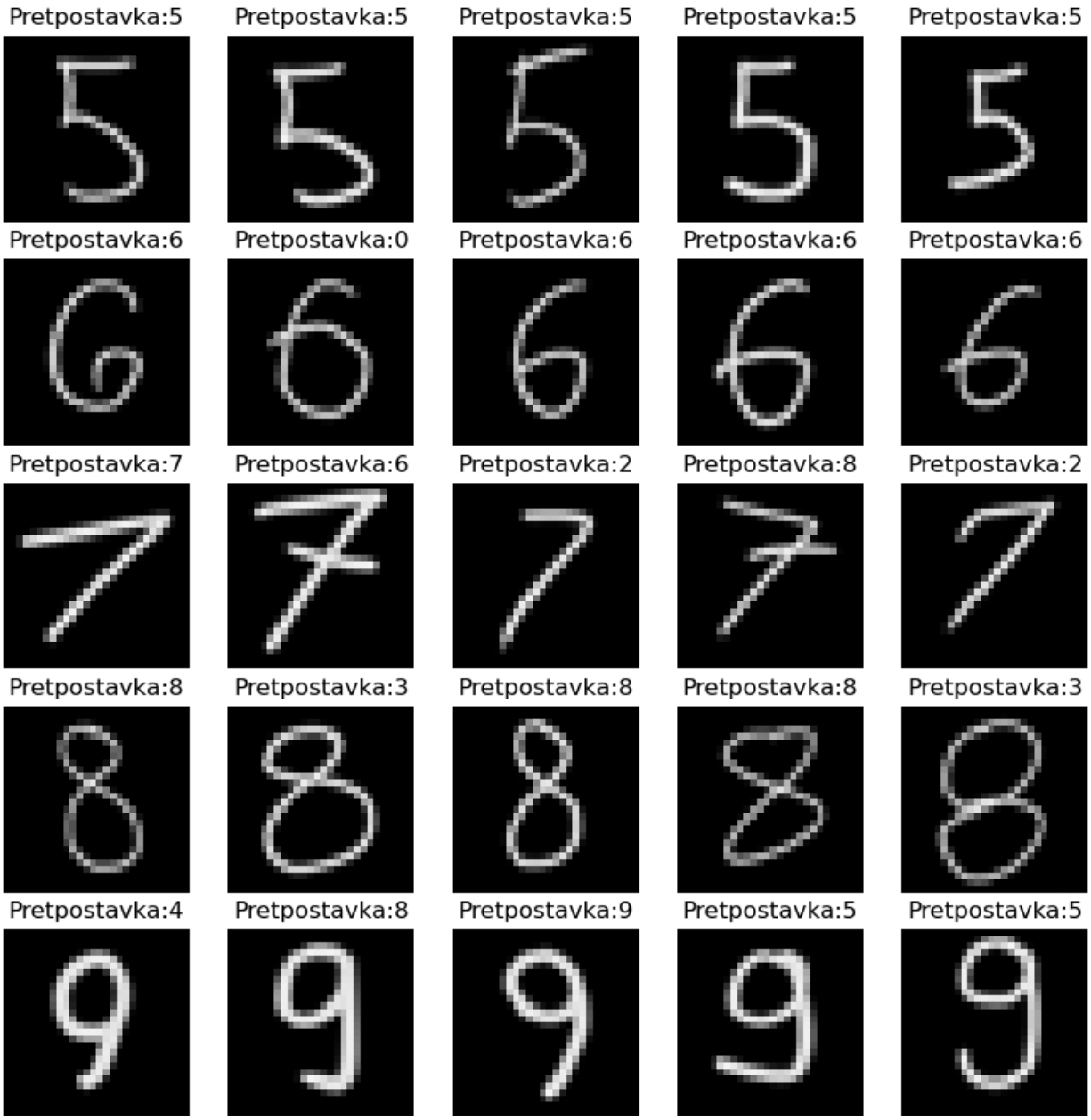
```
print("Uspjesno obrisane slike!")
```

```
Uspjesno obrisane slike!
```

Također, sada će sličan programski kod biti ponovljen za znamenke brojeva od 5 do 9, a razlika je samo u nazivu direktorija u koji su spremljene.

```
broj_slike=1

while os.path.isfile(f"znamenke/5_do_9/znamenka{broj_slike}.png"):
    original_slika =
Image.open(f"znamenke/5_do_9/znamenka{broj_slike}.png")
    # Promeni dimenzije na 28x28
    resized_slika = original_slika.resize((28, 28))

    # Sačuvaj promenjenu sliku
    resized_slika.save(f"testiranje/5_do_9/podesena{broj_slike}.png")
    broj_slike +=1

print("Uspjesno prilagodjene slike!")

#stvaranje polja za podatke, prepoznavanje, ispis
za_testiranje = []
broj_slika=1

while os.path.isfile(f"testiranje/5_do_9/podesena{broj_slika}.png"):
    img = cv2.imread(f"testiranje/5_do_9/podesena{broj_slika}.png",
cv2.IMREAD_GRAYSCALE)
    za_testiranje.append(np.array(img))
#na kraju povećamo varijablu broj_slika za 1
    broj_slika +=1

temp_array = np.invert(np.array(za_testiranje))
array_flat = temp_array.reshape(len(temp_array), 28*28)
array_flat = array_flat / 255
za_pretpostaviti = model.predict(array_flat)
pretpostavka = [np.argmax(i) for i in za_pretpostaviti]


# Iteriraj kroz slike
fig, axs = plt.subplots(5, 5, figsize=(10, 10))
# Iteriraj kroz slike
i=0
for j in range(1, broj_slika):
    # Kreiraj naziv slike

    putanja_slike = os.path.join("testiranje/5_do_9",
f"podesena{j}.png")
```

```
    # Učitaj sliku
    img = Image.open(putanja_slike)

    # Invertiraj boje
    inverted_img = Image.eval(img, lambda x: 255 - x)

    # Odredi indekse redaka i stupaca za prikaz
    redak = (j - 1) // 5
    stupac = (j - 1) % 5
    # Prikazi sliku na odgovarajućem mjestu u subplotu
    axs[redak, stupac].imshow(inverted_img, cmap='gray')
    axs[redak, stupac].set_title("Pretpostavka:"
+str(pretpostavka[i]))
    axs[redak, stupac].axis('off')  # Isključi oznake osi
    i=i+1

Uspjesno prilagodjene slike!
1/1 [==============================] - 0s 16ms/step
```



Uočimo kako umjetna neuronska mreža nije prepoznala efikasno brojeve 7 i 9. Na to uvelike utječe oblikovanje slika i istaknutost piksela. Također, način na koji su znamenke napisane u MNIST bazi. Preostaje obrisati testne slike:

```
za_obrisati=1

#uklanjamo sve podesene slike velicina 28x28 piksela
while os.path.isfile(f"testiranje/5_do_9/podesena{za_obrisati}.png"):
    datoteka = (f"testiranje/5_do_9/podesena{za_obrisati}.png")
    os.remove(datoteka)
    za_obrisati +=1
```

```
print("Uspjesno obrisane slike!")
```

```
Uspjesno obrisane slike!
```

Ovim brisanjem projekt je ujedno i gotov te preostaje rezimirati dosadašnji dio.

Zaključak

Implementacija umjetne neuronske mreže izgleda zahtjevno, ali koristeći pogodnosti koje Python pruža uz odgovarajuće biblioteke pretvara se u vrlo jednostavnu implementaciju. Prvo, istražili smo skup podataka i odgovarajuće biblioteke koje obiluju funkcijama kako bi olakšali daljnji programski kod. Promotrili smo arhitekturu, slojeve, aktivacijske funkcije i neke od metoda učenja. Kreirali smo vlastitu neuronsku mrežu na način da dobijemo što veću efikasnost na testiranju vlastitih rukopisnih znamenaka brojeva od 0 do 9. Ono što prethodno nije bilo navedeno, a dobro bi bilo sada navesti da je broj čvorova u sakrivenom sloju upravo izabran na takav način da se postigne što veća efikasnost na testu koja prema slikama iznad iznosi **37/50 =0.74** što je **74%**. Treba napomenuti da ponovnim pokretanjem ove bilježnice možda neće biti isti rezultat, to jest svako ponovno pokretanje obuke neuronske mreže ne mora rezultirati istim rezultatom. Umjetna neurosnka mreža kreirana je tako da su se vršila testiranja iznova i iznova sve dok se nije postigla visoka efikasnost.Tako da smo u ovom projektu postigli veoma visoku preciznost i efikasnot prepoznavanja znamenki. Općenito, na internetu se navodi kako je efikasnost u večini slučajeva za ovakav način primjene prema računu jednaka **70%**. Poboljšanje koje se predlaže u izgradnji bolje umjetne neuronske mreže s većom efikasnosti prepoznavanja znamenki je konvolucijska neuronska mreža, ali to je tema za neku drugu raspravu / projekt.

Literatura

[1] TensorFlow https://www.tensorflow.org/

[2] Harsha, A. Handwritten Digit Recognition with 98% Accuracy https://www.shiksha.com/online-courses/articles/handwritten-digit-recognition-with-98-accuracy/

[3] Vishwakarma, N. What is Adam Optimizer?, Analytics Vidhya https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/

[4] The Sequential model, TensorFlow https://www.tensorflow.org/guide/keras/sequential_model

[5] But what is a neural network? | Chapter 1, Deep learning https://www.youtube.com/watch?v=aircAruvnKk

[6] Gradient descent, how neural networks learn | Chapter 2, Deep learning https://www.youtube.com/watch?v=IHZwWFHWa-w&t=27s

[7] What is backpropagation really doing? | Chapter 3, Deep learning https://www.youtube.com/watch?v=Ilg3gGewQ5U&t=19s

[8] MNIST digits classification dataset https://keras.io/api/datasets/mnist/

[9] Dense layer https://keras.io/api/layers/core_layers/dense/

[10] Tanwar, S. Building our first neural network in keras
https://towardsdatascience.com/building-our-first-neural-network-in-keras-bdc8abbc17f5