Sveučilište u Rijeci, Odjel za Informatiku

**Klasifikacija slika koristeći neuralne mreže i alat Tensorflow**

Seminarski rad

Studentica: Nola Čumlievski

Kolegij: Komunikacija čovjek stroj

Rijeka, 2021.

**Sadržaj**

[1. Uvod – neuralne mreže 1](#_Toc61368091)

[2. Skup podataka 2](#_Toc61368092)

[3. Priprema podataka 4](#_Toc61368093)

[3.1. Učitavanje skupa podataka 4](#_Toc61368094)

[3.2. Normalizacija podataka 6](#_Toc61368095)

[3.3. Distribucija klasa i izrada skupa za validaciju 7](#_Toc61368096)

[4. Izrada neuralne mreže 9](#_Toc61368097)

[5. Tesitranje mreže na skeniranim riječima 16](#_Toc61368098)

[5.1. Procesiranje i priprema slika za neuralnu mrežu 16](#_Toc61368099)

[6. Rezultati 21](#_Toc61368100)

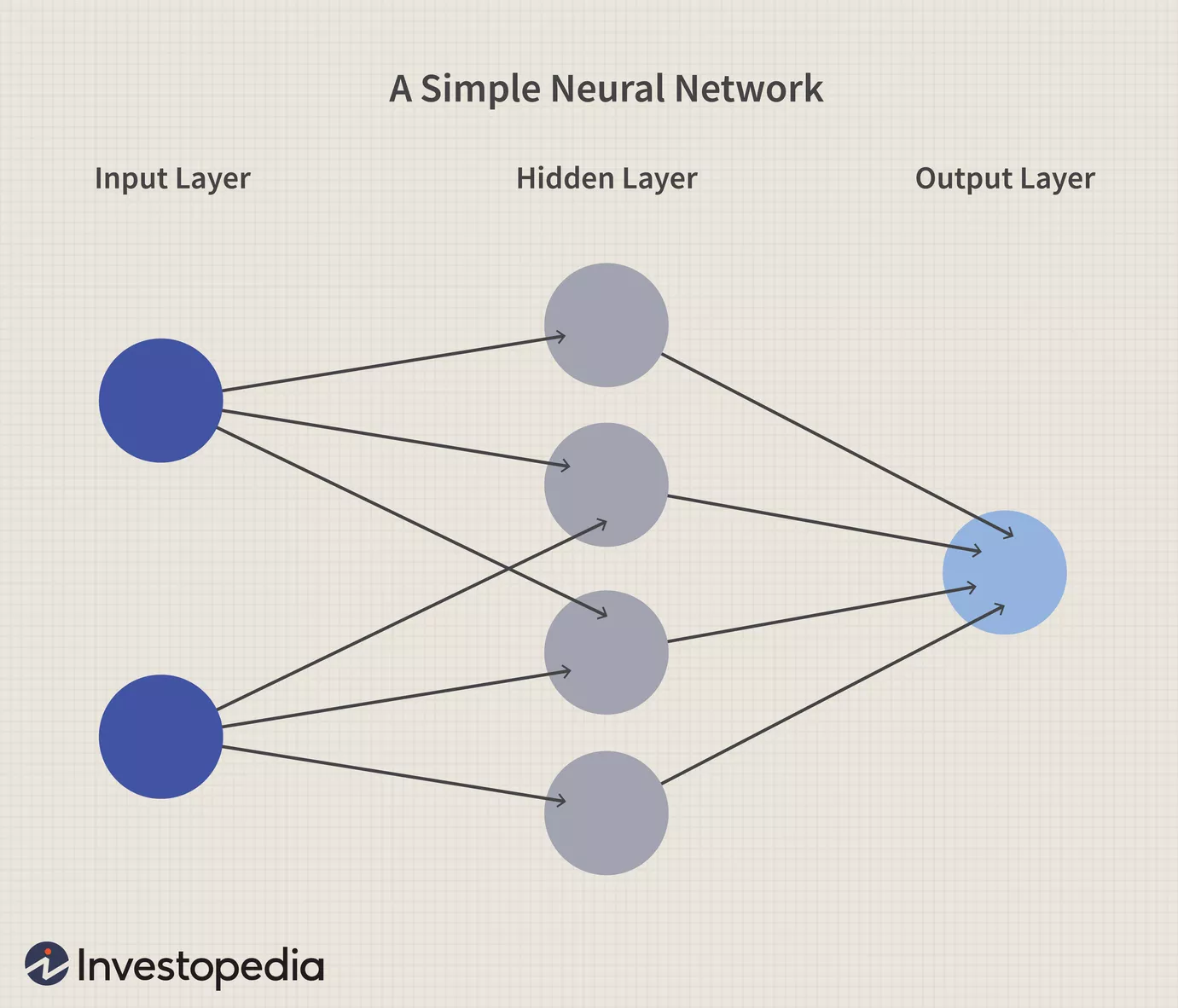
[6.1. Usporedba rezultata 24](#_Toc61368101)

[7. Zaključak 24](#_Toc61368102)

[8. Literatura 25](#_Toc61368103)

# **1. Uvod – neuralne mreže**

Neuralne mreže predstavljaju specifičnu vrstu algoritama koji služe za prepoznavanje veza između podataka unutar određenog skupa. Općenito, bazirane su na ideji ljudskog mozga, gdje se neuralne mreže referiraju na sustav neurona. Neuron u neuralnoj mreži predstalvja matematičku funkciju koja sakuplja i klasificira određene informacije, ovisno o arhitekturi mreže. Osim neurona, neuralna mreža sastoji se od određenog broja povezanih čvorova koji čine sloj neuralne mreže. Svaki čvor unutar sloja predstavlja perceptron koji šalje signal (produkt linearne regresije) u aktivacijsku funkciju koja može biti nelinearna. Osim unutarnjih slojeva (skrivenih slojeva), unutar arhitekture neuralne mreže postoji i ulazni sloj (prikuplja uzorke ulaznih podataka) te izlazni sloj, koji se sastoji od klasa koje ulazni sloj može poprimiti. Većina neuralnih mreža danas organizirane su na način da je svaki individualni čvor unutar sloja povezan sa određenim brojem čvorova u sloju ispod navedenog od kojeg navedeni prima podatke te sa određenim brojem čvorova u sloju iznad navedenog, kojem navedeni čvor šalje podatke.



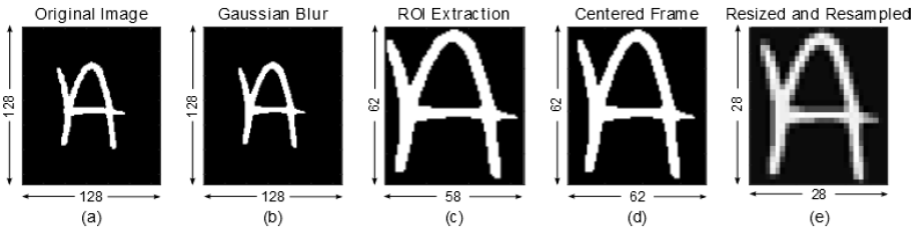
Slika 1. Primjer jednostavne neuralne mreže. Preuzeto sa: <https://www.investopedia.com/thmb/clD2Q1_Xwa-d2yE0066Cs8oI3J8=/1381x0/filters:no_upscale():max_bytes(150000):strip_icc():format(webp)/dotdash_Final_Neural_Network_Apr_2020-01-5f4088dfda4c49d99a4d927c9a3a5ba0.jpg>

Prilikom svakog i slijedećeg povezivanja, čvor određuje broj poznat kao „težina“. U trenutku kada je mreža aktivna, čvor preuzima određeni podatak – broj, koji kroz svako povezivanje množi sa asociranom težinom. Zatim, zbraja (sumira) rezultate kako bi rezultat bio jedan broj (suma težina). Ako je navedeni broj manji od određene granice, čvor ne šalje podatke slijedećem sloju. U suprotnom slučaju, čvor šalje broj (sumu težina) svim slijedećim poveznicama. Prilikom treniranja neuralne mreže, sve težine i granice (engl „*thresholds*) su inicirane nasumično. Podaci za treniranje primljeni su od strane ulaznoj sloja i šalju se svim slojevima unutar mreže, gdje se množe i zbrajaju na kompleksan način, dok se transformirani i procesirani podaci ne pošalju na izlazni sloj. Tijekom procesa treniranja, težine i granice se kontinuirano prilagođavaju sve dok podaci za treniranje sa istim oznakama ne rezultiraju sličnim rezultatima.

# **2. Skup podataka**

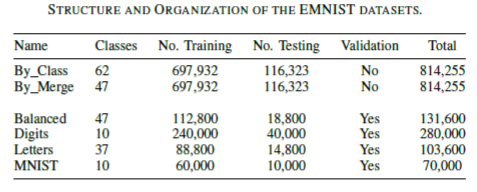
Za potrebe ovog projekta, korišten je EMNIST balanced skup podataka, koji se sastoji od 131,600 instanci koje su klasificirane u 47 klasa (0-46) gdje svaka pojedina klasa označava broj, malo ili veliko slovo. Određena slova u skupu podataka (C, I, J, K, L, M, O, P, S, U,

V, W, X, Y i Z ) spojena su u jednu klasu zbog sličnosti između malih i velikih slova navedenih klasa. Slike unutar skupa su preprocesirane na slijedeći način:



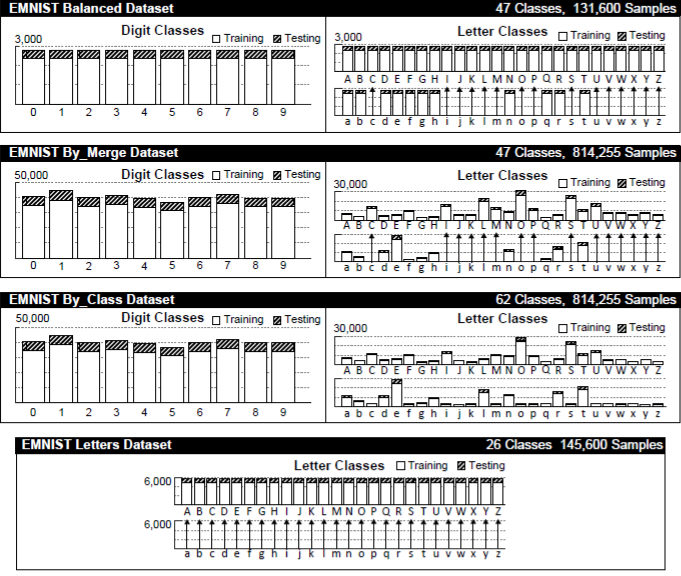
Slika 2. Postupak procesiranja slika. Preuzeto sa: Cohen, Afshar, Tapson, Schaik: EMNIST: an extension of MNIST to handwritten letters

Originalne slike su pohranjene kao 128 x 128 binarne slike (a). Gaussian filter sa σ = 1 primjenjen je na navedene slike kako bi se ublažili rubovi slova (b). Izrezan je višak pozadine oko slova s obzirom da slovo nije pokrivalo većinu slike (c). Oznaka je zatim centirirana (d) a da je pritom očuvan omjer širine i visine slike. Konačno, slika je smanjena na 28 x 28 piksela te je raspon vrijednosti piksela skaliran na [0, 255] što je rezultiralo crno bijelom slikom od 28 x 28 piksela (e).



Slika 3. Struktura EMNIST skupa podataka. Preuzeto sa: Cohen, Afshar, Tapson, Schaik: EMNIST: an extension of MNIST to handwritten letters

Na slici 3. prikazane su vrste i strukture pojedinih skupova podataka unutar EMNIST skupa podataka. Za potrebe ovog projekta, moglo se birati između 4 skupa podataka: Balanced, By\_Class, By\_Merge i Letters. Na slici 4. prikazani su sadržaji pojedinog skupa podataka.



Slika 4. Sadržaj pojedinog skupa podataka. Preuzeto sa: Cohen, Afshar, Tapson, Schaik: EMNIST: an extension of MNIST to handwritten letters

Za potrebe ovog projekta odabran je EMIST balanced skup podataka iz slijedećih razloga: *By\_Class* skup podataka ima sve potrebne klase ali nisu balansirane, *By\_Merge* skup podataka, što se tiče slova, također sadrži spojena gore navedena slova zbog sličnosti u izgledu ali klase su također nebalansirane. Skup podataka *Letters* sadrži samo 26 klasa, što znači da je svako slovo (malo i veliko) spojeno u jednu klasu (u ovom slučaju imamo samo velika slova) te s obzirom da neke inačice slova (npr. malo i veliko „a“) uopće nisu slične po izgledu, odlučila sam izabrati EMNIST *Balanced* skup podataka koji sadrži mala i velika slova, gdje su spojene klase slova samo onih koji su slični izgledom te su klase balansirane po broju uzoraka.

# **3. Priprema podataka**

Unutar ovog poglavlja opisani su načini učitavanja i procesiranja podataka te je prikazan i ukratko objašnjen kod koji je korišten prilikom navedenih postupaka.

## **3.1. Učitavanje skupa podataka**

Za potrebe ovog projekta, instalirane su i učitane slijedeće biblioteke: *numpy, tensorflow, pandas, emnist, matplotlib, keras, sklearn, seaborn, cv2, imutils* i *PIL.* Korišten je Spyder IDE, te ukoliko ne posjedujete neki od određenih paketa, mogu se instalirati upisivanjem slijedeće naredbe unutar konzole:

python -m pip install “SomePackage” - gdje umjesto “SomePackage” unesemo samo ime paketa kojeg želimo instalirati.

Nadalje, skup podataka moguće je učitati na više načina, možemo skinuti isti sa Kaggle stranice (<https://www.kaggle.com/crawford/emnist>) ili sa službene NIST stranice (<https://www.nist.gov/itl/products-and-services/emnist-dataset>) te učitati dobivene csv datoteke ili koristiti Tensorflow ili Python za direktno skidanje i učitavanje podataka. U ovom projektu, EMNIST skup podataka skinut je direktno putem konzole instalacijom paketa *emnist* te učitavanjem instanci u određene varijable.

list\_datasets()

# izrada skupova za treniranje i testiranje

treniranje, treniranje\_oznake = extract\_training\_samples('balanced')

testiranje, testiranje\_oznake = extract\_test\_samples('balanced')

Koristimo funkciju *list\_datasets* kako bi nam Python prikazao skupove podataka koje je moguće učitati, zatim pomoću funkcije *extract\_samples* učitavamo željeni skup podataka – u ovom slučaju je to *balanced* skup podataka.

Čisto da se uvjerimo da je sve učitano dobro, izvodimo kod za očitavanje jedne od slika u skupu podataka:

plt.imshow(treniranje[1].reshape([28, 28]), cmap='Greys\_r')

plt.show()

gdje želimo grafički prikazati sliku s indeksom „1“ te dobijemo slijedeći rezultat:

A picture containing qr code

Description automatically generated

Slika 5. Prikaz druge slike skupa podataka

Na slici 5. vidljivo je da je riječ o malom slovu „a“ te vidimo da nije potrebna rotacija slika te da je skup podataka dobro učitan (prilikom istraživanja načina kako da učitam skupove podataka, neke csv datoteke učitaju rotirane slike te ih je potrebno prije korištenja rotirati).

**Raspored klasa**

Prema službenoj dokumentaciji EMNIST skupa podataka: <https://arxiv.org/abs/1702.05373> klase su kodirane na slijedeći način: Brojevi od 0-9 odgovarajućim idenksima [0-9], A – 10, B – 11, C – 12, D – 13, E – 14, F – 15, G – 16, H – 17, I – 18, J – 19, K – 20, L – 21, M – 22, N – 23, O – 24, P – 25, Q – 26, R – 27, S – 28, T – 29, U – 30, V – 31, W - 32, X – 33, Y – 34, Z – 35, a – 36, b – 37, d – 38, e – 39, f – 40, g – 41, h – 42, n – 43, q – 44, r – 45, g - 46

## **3.2. Normalizacija podataka**

Prilikom procesiranja, inače se podaci skaliraju na način da rezultat bude više numerički stabilan (u ovom slučaju, želimo rezultat u rasponu vrijednosti piksela od 0 do 1 umjesto od 0 do 255).



Slika 6. Originalni izgled instance

# skaliranje vrijednosti

treniranje\_podaci = np.copy(treniranje)

treniranje\_podaci = treniranje\_podaci.astype('float32')

treniranje\_podaci /= 255.

testiranje\_podaci = np.copy(testiranje)

testiranje\_podaci = testiranje\_podaci.astype('float32')

testiranje\_podaci /= 255.

Kao što je prethodno rečeno, slike unutar EMNIST skupa podataka sastoje se od piksela u vrijednosti od 0 do 255, gdje svaka vrijednost predstavlja jednu nijansu od ukupno 256 nijansi sive. Dijeljenjem svake pojedine vrijednosti sa 255 (vrijednosti moraju biti *float* vrste podataka) dobijemo vrijednosti od 0 do 1, gdje 0 predstavlja maksimalno crnu boju, a 1 maksimalno bijelu boju.



Slika 7. Novi izgled instance

## **3.3. Distribucija klasa i izrada skupa za validaciju**

Iako je prethodno navedeno i prikazano da je *balanced* skup podataka izrazito balansiran što se tiče broja uzoraka unutar pojedine klase, uvijek je dobro imati grafički prikaz istog koji će potvrditi navedeno. Jedna od metoda za provjeru distribucije klasa je izrada histograma:

plt.figure(figsize=(8,5))

plt.hist(treniranje\_oznake, bins=47)

plt.gca().set(title='Frequency Histogram', ylabel='Frequency')

gdje pod parametar *bins* navodimo broj klasa skupa podataka (u ovom slučaju 47). Pokretanjem ovog dijela koda dobijemo slijedeći rezultat:

Chart, histogram

Description automatically generated

Slika 8. Histogram frekvencije

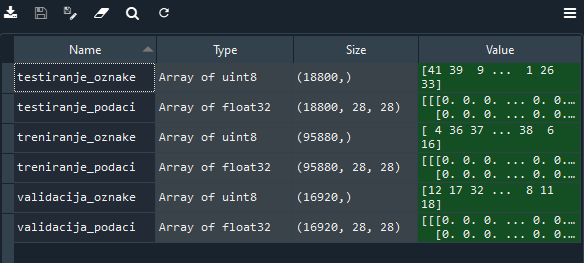
Na slici 8. možemo vidjeti da su klase balansiranje te da postoji otprilike jednak ili sličan broj opservacija unutar svih navedenih klasa.

Nakon provjere distribucije klasa i procesiranja podataka, možemo izraditi skup podataka za validaciju. Izrada ovog skupa podataka nije obavezna, no dobra praksa nalaže da se model trenira na skupu za treniranje i za validaciju, te da se prilikom testiranja modela koristi potpuno novi skup podataka – za testiranje. Skup za validaciju izrađen je kao dio skupa za treniranje, s obzirom da se dosta veći broj opservacija nalazi u navedenom skupu.

# izrada skupa za validaciju

treniranje\_podaci, validacija\_podaci, treniranje\_oznake, validacija\_oznake = train\_test\_split(treniranje\_podaci, treniranje\_oznake, test\_size=0.15, random\_state=1)

Pod parametar *test\_size* navodimo koliki dio skupa za treniranje želimo odvojiti u skup za validaciju, te je u ovom primjeru odabrano 15% nasumičnih podataka skupa za treniranje kao skup za validaciju.



Slika 9. Konačan popis skupa podataka

Na slici 9. možemo vidjeti izgled i strukturu konačnih skupova podataka. Imamo skup za testiranje od 18800 instanci i njima pripadajućim klasama unutar druge varijable (*testiranje\_oznake*)*,* skup za treniranje od 95880 instanci te skup za validaciju od ukupno 16920 instanci.

# **4. Izrada neuralne mreže**

U ovom poglavlju opisani su i prikazani postupci prilikom izrade neuralne mreže.

# parametri

input\_veličina = 784 # podaci su formata 28 X 28

output\_veličina = 47 # broj klasa

hidden\_layer\_veličina = 700

Input veličina predstavlja veličinu ulaznih podataka, koja je u ovom slučaju 784 (podaci su oblika 28 x 28). Output veličina predstavlja veličinu izlaznog sloja neuralne mreže koji predstavlja broj klasa čiju vrijednost podatak može poprimiti (u ovom slučaju to je 47). Veličina skrivenog sloja mreže je opcionalna, broj 700 dobiven je kao „suboptimalni“ broj prilikom više pokretanja treniranja gdje su se koristile različite veličine skrivenih slojeva. Alternativno, može se koristiti različita veličina sloja za svaki skriveni sloj.

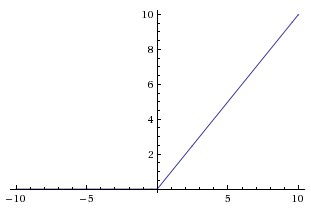
model = tf.keras.Sequential([tf.keras.layers.Flatten(input\_shape = (28, 28, 1)), tf.keras.layers.Dense(hidden\_layer\_veličina, activation = "relu"),

tf.keras.layers.Dense(hidden\_layer\_veličina, activation = "relu"),tf.keras.layers.Dense(output\_veličina, activation = "softmax")])

Za izradu modela koristi se funkcija *Sequential* koja predstavlja dio *keras* biblioteke. Podaci su originalno oblika (28, 28, 1) te je ulazne podatke potrebno preoblikovati u vektor. Za to se koristi metoda *Flatten* unutar koje se navodi originalni oblik podataka. Navedena metoda transformira („izravnava“) podatke u vektor. Metoda *Dense* koristi se za izradu slojeva mreže. Unutar ovog modela korištene su aktivacijske funkcije „*relu“*(za skrivene slojeve mreže)i *„softmax“* (za izlazni sloj mreže) te je definicija istih navedena u nastavku.

**Rectified Linear Unit – relu**

Relu aktivacijska funkcija predstavlja najviše korištenu aktivacijsku funkciju unutar modela dubokog učenja (neuralnih mreža). Funkcija vraća 0 ako primi negativan ulaz, ali za svaku pozitivnu vrijednost *x* vraća originalnu vrijednost. Može se napisati kao f(x)=max(0,x) te grafički izgleda ovako:



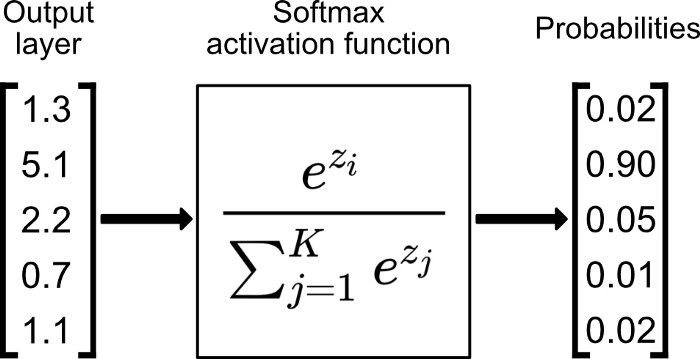
Slika 10. Relu aktivacijska funkcija. Preuzeto sa: <https://i.imgur.com/gKA4kA9.jpg>

Aktivacijske funkcije koriste se u dvije svrhe: 1) pomažu modelu prilikom detekcije interakcijskog utjecaja koji predstavlja situaciju kada određena varijabla *x* utječe na predikciju modela drugačije nego kad je ista pod utjecajem vrijednosti druge varijable – *y.* 2) pomažu modelu prilikom detekcije nelinearnih utjecaja, što znači ako grafički prikažemo varijablu na horizontalnoj osi i predikcije na vertikalnoj osi, linija predikcije nije ravna. Drugim riječima, utjecaj povećanja određenog prediktora je različit kada se radi o drugačijim vrijednostima navedenog prediktora.

Postoje i druge nelinearne aktivacijske funkcije, kao što su Sigmoid i Hyperbolic tangent. Unutar ove mreže relu funkcija bolje odgovara opisu, s obzirom da je npr. Sigmoid funkcija više prilagođena binarnim klasifikacijama (kada imamo samo 2 klase koje predviđamo). Generalni problem kod Sigmoid i Tanh funkcije jest saturacija, što znači da se vrijednosti, koje su znatno veće od 1 i znatno manje od 0 preusmjeravaju na -1 i 1 (tanh funkcija) i 0 i 1 (sigmoid funkcija) što algoritmu predstavlja problem kod prilagođavanja težina kako bi poboljšao performanse modela. Kod relu funkcije, sve negativne vrijednosti preusmjeravaju se na 0, dok se pozitivna vrijednost vraća u originalnom obliku.

**Softmax aktivacijska funkcija**

Softmax aktivacijska funkcija koristi se za klasifikacijske probleme (u izlaznom sloju neuralne mreže), posebno kod klasifikacijskih problema gdje postoji više od dvije klase predviđanja. Navedena funkcija vraća „confidence score“ za svaku klasu. S obzirom da se radi o rezultatima vjerojatnosti, svi rezultati softmax funkcije zbrojeni daju 1. Predviđena klasa (izbačena kao rezultat modela) je objekt unutar liste gdje je confidence score najveći.



Slika 11. Softmax funkcija. Preuzeto sa: [https://miro.medium.com/max/700/1\*ReYpdIZ3ZSAPb2W8cJpkBg.jpeg](https://miro.medium.com/max/700/1*ReYpdIZ3ZSAPb2W8cJpkBg.jpeg)

Izračun vjerojatnosti temelji se na dva koraka. Prvi korak je potenciranje svakog izlaznog elementa i sumiranje rezultata (u ovom slučaju rezultat je ~ 181.73). Drugi korak je potenciranje svakog individualnog elementa izlaznog sloja i podjela istog sa sumom dobivenom u prvom koraku (exp(1.3) / 181.37 = 3.67 / 181.37 = 0.02). Ovaj postupak se izvodi za svaki pojedini element i bira se ona vjerojatnost (klasa) gdje je dobiveni rezultat najveći.

Nakon definicije modela, potrebno je odrediti optimizacijski algoritam istog.

# specifikacija optimizatora

optimizer = keras.optimizers.Adam(lr=0.005)

model.compile(optimizer = optimizer, loss = "sparse\_categorical\_crossentropy", metrics = "accuracy")

Unutar ovog projekta koristi se optimizator *Adam. Adam* je optimizacijski algoritam koji se može koristiti umjesto klasičnog algoritma – *stochastic gradient descent* za ažuriranje težina mreže nad skupom za treniranje. Stopa učenja je također proizvoljna – dobivena kao “suboptimalna” prilikom višestrukog testiranja različith vrijednosti. Kao metrika se navodi točnost s obzirom da je riječ o klasifikacijskom problem te kao *loss* funkciju navodimo *“sparse\_categorical\_crossentropy”*. Postoje 3 vrste loss funkcije, a to su: *binary crossentropy* (referira se na slučajeve kad koristimo binarno kodiranje), *categorical crossentropy* i *sparse categorical crossentropy* su indentične, no prva podrazumjeva da smo prilikom procesiranja primjenili postupak one-hot kodiranja gdje je svaka kategorija (klasa) prikazana vektorom od dužine ukupnog broja klasa i sadrži jedinicu samo na mjestu njemu pripadajuće klase (ako je klasa podatka 24, vektor sadrži 47 instance gdje su sve nule osim 24. instance gdje je 1). S obzirom da ovaj postupak nije primjenjen prilikom procesiranja podataka, koristi se *sparse categorical crossentropy.*

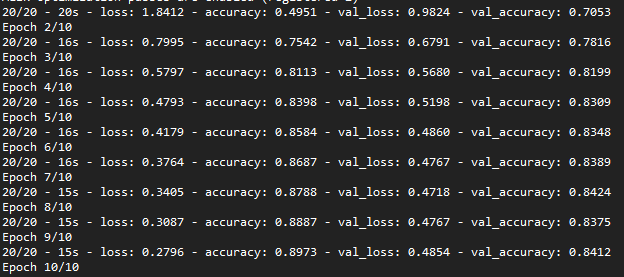
# treniranje modela

broj\_epoha =10

np.random.seed(30)

fit = model.fit(treniranje\_podaci, treniranje\_oznake, batch\_size = 5000, epochs = broj\_epoha, verbose=2, validation\_data=(validacija\_podaci, validacija\_oznake))

Nakon definicije modela i optimizacijskog algoritma, potrebno je istrenirati model. Koristimo funkciju *fit* te navodimo podatke za treniranje, veličinu batcha koja predstavlja količinu podataka koja se procesira u jednom trenutku (koristi se kad se radi s većim skupovima podataka), broj epoha te podatke za validaciju. Pokretanjem navedenog koda dobijemo slijedeći rezultat:



Slika 12. Rezultat treniranja

Na slici 12. vidimo rezultat treniranja modela, gdje je za svaku epohu (iteraciju) prikazan *loss* i točnost modela nad skupom za treniranje i validaciju.

# testiranje modela na skupu za testiranje

rezultati = model.evaluate(testiranje\_podaci, testiranje\_oznake)

Zatim model testiramo nad skupom za testiranje i dobijemo slijedeći rezultat:

588/588 [==============================] - 3s 6ms/step - loss: 0.4940 - accuracy: 0.8425

Iz navedenoga možemo primjetiti da model nad skupom za testiranje ima točnost od ~ 84%.

Chart, line chart

Description automatically generated

Slika 13. Treniranje modela

Na slici 13. grafički je prikazan proces treniranja modela gdje možemo vidjeti postepeno poboljšavanje modela. Prilikom određivanja parametara (stope učenja, veličine batcha i slično) pazilo se da se model ne pretrenira, jer u nekim slučajevima (kada bi se postavila malo veća stopa učenja, manja veličina skrivenih slojeva ili manja veličina batcha) model bi u početku imao izrazito veliku točnost, no daljnim tokom epoha bi se točnost smanjivala i *loss* povećavao.

rezultati1 = model.predict\_classes(testiranje\_podaci, verbose=1)

data = {'Prave klase': testiranje\_oznake,'Predviđene klase': rezultati1}

df = pd.DataFrame(data, columns=['Prave klase','Predviđene klase'])

confusion\_matrix = pd.crosstab(df['Prave klase'], df['Predviđene klase'], rownames=['Prave klase'], colnames=['Predviđene klase'], margins = True)

Radi pogodnosti, izrađena je matrica konfuzije rezultata kako bi mogli grafički prikazati koje klase model više ili manje pogađa. Predviđene klase spremamo u određenu varijablu te pomoću *pandas* biblioteke izrađujemo *DataFrame* koji u presjeku svakog stupca i reda sadrži broj predviđenih instanci.

Chart

Description automatically generated

Slika 14. Matrica konfuzije

Na slici možemo vidjeti matricu konfuzije predikcija modela. Na dijagonali se nalaze točno predviđene vrijednosti dok su ostale lažno pozitivne ili lažno negativne. Možemo prokomentirati vrijednosti u čijim je slučajevima bilo 100 ili više pogrešno predviđenih klasa.

1) 120 klasa slova „O“ netočno su klasificirane kao 0. 2) 155 klasa slova „I“ netočno su klasificirane kao 1. 3) 164 klase slova „L“ netočno su klasificirane kao 1. 4) 99 klasa slova „q“ netočno su klasificirane kao 9. 5) 103 klase broja „5“ netočno su klasificirane kao slovo „S“. 6) 101 klase slova „n“ netočno su klasificirane kao „M“. Kao što vidimo prema rezultatima, većina pogrešnih klasifikacija mogla se riješiti izbacivanjem brojeva iz skupa podataka, no pošto su podaci u početku odvojeni od oznaka klasa te izmješani (nisu poredani redom sve opervacije sa klasom 0, pa 1...itd. gdje bi se samo mogao izbacili dio DataFrame-a) nije pronađen način kako bi se izbacili brojevi iz skupa podataka, no to je nešto na čemu bi se možda moglo dalje istraživati. Također, primjećujemo sličnosti između izgleda točne i pogrešno klasificirane klase.

# **5. Tesitranje mreže na skeniranim riječima**

Za potrebu ovog testiranja, izvučene su 3 slike riječi iz skupa podataka *Words MNIST* koji se sastoji od ~ 10000 slika. Skup podataka nalazi se na stranici <https://www.kaggle.com/backalla/words-mnist>, te je skinut folder sa slikama, raspakiran i odabrane su 3 nasumične riječi nad kojima će se vršiti testiranje. Te riječi su redom „marks“, „questions“ i „company“.



Slika 15. Primjer slika iz Words MNIST skupa podataka. Preuzeto sa: <https://storage.googleapis.com/kaggle-datasets-images/30321/38645/5ef5d0949d22218890dc8b35fb28ba28/dataset-cover.png?t=2018-06-06-09-47-47>

## **5.1. Procesiranje i priprema slika za neuralnu mrežu**

U ovom poglavlju navedena su i opisana dva algoritma potrebna za izdvajanje slova iz riječi i stvaranja pojedinih slika slova te algoritam za preoblikovanje dobivenih slika slova kako bi bile istih dimenzija kao ulazni podaci nad kojima se mreža trenirala.

Najprije, slika riječi je učitana u radni direktorij u kojem se također i nalazi skripta u kojoj je pisan kod radi pogodnosti.

image = cv2.imread("marks.png")

cv2.namedWindow("Output", cv2.WINDOW\_AUTOSIZE)

cv2.imshow("Output", image)

cv2.waitKey(0)

invert = cv2.bitwise\_not(image)

cv2.namedWindow("Output", cv2.WINDOW\_AUTOSIZE)

cv2.imshow("Output", invert)

cv2.waitKey(0)

Zatim sliku učitavamo pomoću funkcije *imread* i koristimo funkciju *bitwise\_not* kako bi preobratili boje slika tako da pozadina bude crna a slova bijela (pokušalo se testirati nad slikama nepromjenjenih broja – bijela pozadina i crna slova, no točnost modela bila je znatno smanjena). Primjenom gore navedenoga koda na sliku dobijemo slijedeći rezultat:



## 



# podjela slike na pojedina slova

gray = cv2.cvtColor(invert, cv2.COLOR\_BGR2GRAY)

thresh = cv2.threshold(gray,0,255,cv2.THRESH\_OTSU + cv2.THRESH\_BINARY)[1]

cnts = cv2.findContours(thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

cnts = cnts[0] if len(cnts) == 2 else cnts[1]

cnts, \_ = contours.sort\_contours(cnts, method="left-to-right")

ROI\_number = 0

for c in cnts:

    area = cv2.contourArea(c)

    if area > 10:

        x,y,w,h = cv2.boundingRect(c)

        ROI = 255 - image[y:y+h, x:x+w]

        cv2.imwrite('ROI\_{}.png'.format(ROI\_number), ROI)

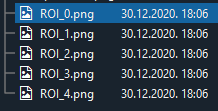
        cv2.rectangle(invert, (x, y), (x + w, y + h), (36,255,12), 1)

        ROI\_number += 1

cv2.imshow('thresh', invert)

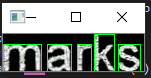
cv2.waitKey()

Najprije se obavlja pretvorba slike u „gray scale“, zatim primjenjujemo Otsu threshold (metoda koja se koristi za odvajanje piksela na piksele pozadine i piksela objekta), odvajamo konture prema vrijednostima piksela i sortiramo od lijevo prema desno (kako su i raspoređena slova unutar riječi). Stvara se pojedina slika prema konturama unutar for petlje i sprema se u pojedinu ROI sliku. Nakon prosljeđivanja slike algoritmu dobijemo slijedeći rezultat:



Slika 16. ROI slike riječi

gdje imamo ROI slika onoliko koliko ima slova u pojedinoj riječi. U ovom slučaju korištena je riječ „marks“ te vidimo da imamo 5 ROI slika, numeriranih od 0 - 4.



Slika 17. Threshold

Na slici 17. prikazana je vizualizacija kontura slike. Vidimo da algoritam izolira pojedino slovo unutar slike riječi.

Nakon što smo izolirali pojedina slova, iste je potrebno procesirati tako da budu istog oblika kao i pojedine slike slova unutar EMNIST skupa podataka nad kojim se mreža trenirala.

def imageprepare(argv): # file path kao input

    im = Image.open(argv).convert('L')

    width = float(im.size[0])

    height = float(im.size[1])

    newImage = Image.new('L', (28, 28), (0))

    if width > height:

        nheight = int(round((20.0 / width \* height), 0))

        if (nheight == 0):

            nheight = 1

        img = im.resize((20, nheight), Image.ANTIALIAS).filter(ImageFilter.SHARPEN)

        wtop = int(round(((28 - nheight) / 2), 0))

        newImage.paste(img, (4, wtop))

    else:

        nwidth = int(round((20.0 / height \* width), 0))

        if (nwidth == 0):

            nwidth = 1

        img = im.resize((nwidth, 20), Image.ANTIALIAS).filter(Image

Filter.SHARPEN)

        wleft = int(round(((28 - nwidth) / 2), 0))

        newImage.paste(img, (wleft, 4))

    tv = list(newImage.getdata())  # vrijednosti piksela

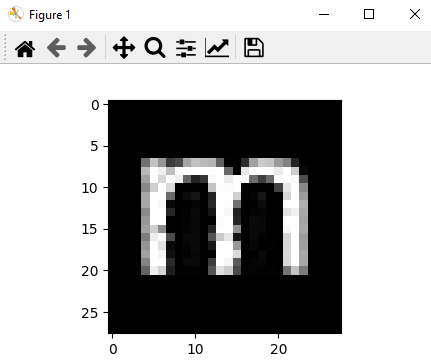
    # normalizacija piksela

    tva = [(x / 255.) for x in tv]

    print(tva)

    return tva

Funkcija kao input prima file path od slike. Otvara se crno bijela slika (parameter “L”), bilježi se visina i dužina slike te se stvara canvas **crne** pozadine (crna jer je pozadina slika slova crna). Provjerava se koja je dimenzija slike veća (dužina ili visina), te se obavlja preoblikovanje veličine slike i primjenjuje se filter za povećanje oštrine slike kako slova ne bi bila mutna. Nakon procesiranja, slika se postavlja na canvas. Konačno, vrijednosti piksela slika se normaliziraju i vraća se procesirana slika.



Slika 18. Procesirano slovo

Na slici 18. prikazano je grafički kako izgleda prva od 5 ROI slika. Originalne ROI slike, dobivene prošlom funkcijom su bile oštrije s manje crne pozadine iza slova (jer su originalne slike manje od poželjnog 28 x 28 oblika).

Konačno, predikcija svakog slova obavlja se na slijedeći način:

x=imageprepare('ROI\_0.png')

# treba biti dužina 784

x = np.array(x)

plt.imshow(x.reshape([28, 28]), cmap='Greys\_r')

plt.show()

# predviđanje

rezultat = model.predict\_classes(x.reshape((1, 28, 28)), verbose=1)

print(rezultat)

Učitavamo u varijablu *x* sliku prosljeđenu u prethodno ispisanu funkciju te je zatim pretvaramo u numpy polje. Dobiveni rezultat prosljeđujemo u funkciju *predict\_classes* gdje je polje od 784 elementa potrebno preoblikovati u polje oblika (1, 28, 28) kako bi bilo istog oblika kao i skup nad kojim se mreža trenirala. Ispišemo rezultat i dobijemo odgovor tipa:

[22] koji predstavlja klasu “M” što je točno s obzirom da smo ubacili ROI s slikom slova “m”.

Navedeni postupak, gdje pridodajemo sliku riječi u algoritam za razdvajanje slova, zatim pojedina slova u funkciju *imageprepare* a zatim i u funkciju *predict\_classes* ponavlja se za svako pojedino slovo svake riječi koje želimo testirati.

# **6. Rezultati**

U ovom poglavlju prikazani su rezultati dobiveni opisanim metodama.

**Riječ “marks”**

Tablica 1. Marks

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Slovo | M | a | r | k | s |
| Klasa | 22 | 36 | 45 | 20 | 28 |
| Točnost | **Badge Tick1 with solid fill** |  |  |  |  |

**Riječ „questions“**

Tablica 2. Questions

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Slovo | q | u | e | s | t | i | o | n | s |
| Klasa | 44 | 30 | 39 | 28 | 18 | 18 | 24 | 43 | 28 |
| Točnost |  |  | **Badge Tick1 with solid fill** | **Badge Tick1 with solid fill** | Close with solid fill | **Badge Tick1 with solid fill** | **Badge Tick1 with solid fill** | **Badge Tick1 with solid fill** | **Badge Tick1 with solid fill** |

Prema tablici 2. vidimo da je jedino pogrešno klasificirano slovo “t”. Klasificirano je kao „I“ što je razumljiva greška.

**Riječ „company“**

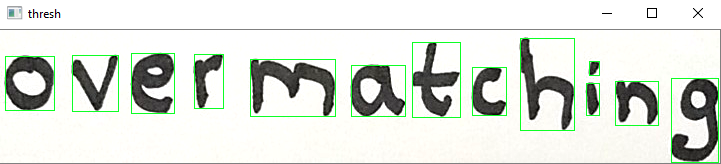
Tablica 3. Company

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Slovo | c | o | m | p | a | n | y |
| Klasa | 24 | 24 | 22 | 25 | 36 | 43 | 34 |
| Točnost | Close with solid fill |  |  |  |  |  |  |

Prema tablici 3. vidimo da je jedino netočno klasificirano slovo „c“ kao „o“. Kada sumiramo rezultate dobivene nad skeniranim slovima, dobijemo točnost od ~ 90% za testirane riječi iz skupa podataka. Međutim, ove riječi nisu ručno napisane, te je odlučeno testirati model i na vlastitim slikama riječi napisanim rukom.

Ručno napisane riječi koje su odlučene testirati su redom „overmatching“, „TRANsfix“ te JudDgGmental“. Velika slova su ubačena jer se u prethodnim riječima model testirao samo na malim slovima. Riječi su izabrane na način da se ista slova unutar navedenih ne ponavljaju kako bi imali više raznoliko testiranje.

**Riječ „overmatcing“**



Slika 19. Threshold

Na slici 19. vidimo da su konture slova jednako dobro prepoznate kao i kod skeniranih riječi.

Tablica 4. Overmatching

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Slovo | o | v | e | r | m | a | t | c | h | i | n | g |
| Klasa | 24 | 31 | 39 | 45 | 22 | 36 | 46 | 12 | 42 | 18 | 43 | 28 |
| Točnost |  |  |  |  |  |  |  |  |  |  |  | Close with solid fill |

Na tablici 4. vidimo da je model pogrešno klasificrao samo jedno slovo – „g“ kao „s“.

**Riječ „TRANsfix“**

Tablica 5. Transfix

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Slovo | T | R | A | N | s | f | i | x |
| Klasa | 29 | 27 | 10 | 23 | 28 | 15 | 18 | 33 |
| Točnost |  |  |  |  |  | Close with solid fill |  |  |

Na tablici 5. vidimo da je jedino pogrešno klasificirano slovo „f“ kao „F“.

**Riječ „JudDgGmental“**

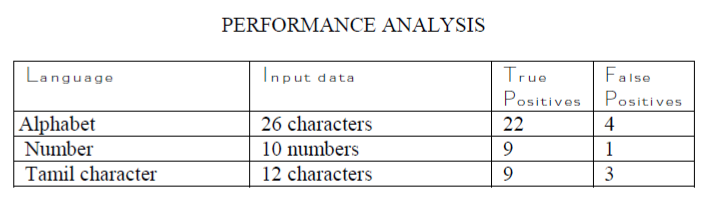
Tablica 6. Judgemental

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Slovo | J | u | d | D | g | G | m | e | N | t | a | L |
| Klasa | 19 | 30 | 38 | 13 | 8 | 9 | 22 | 39 | 43 | 46 | 36 | 21 |
| Točnost |  |  |  |  | Close with solid fill | Close with solid fill |  |  |  |  |  |  |

Na tablici 6. vidimo da je model pogrešno klasificirao malo i veliko „g“ kao brojeve 8 i 9 redom. Uočljive su sličnosti slova/brojeva unutar pogrešnih klasifikacija. Kada sumiramo rezultate dobivene nad ručno napisanim riječima, dobijemo točnost od ~87.5% što također predstavlja dobar rezultat.

## **6.1. Usporedba rezultata**

Prilikom pretraživanja sličnih istraživanja, naišla sam na slijedeće: Anandh K.S., Clintom D. J. *HANDWRITTEN CHARACTER RECOGNITION USING CNN* (dostupno na: <https://ijrar.org/papers/IJRAR1903931.pdf>), gdje je također korištena neuralna mreža za klasifikaciju slova engleske abecede, brojeva te i tamil znakova.



Slika 20. Rezultati autora. Preuzeto sa: <https://ijrar.org/papers/IJRAR1903931.pdf>

Autori su testirali 26 slova, od kojih je 4 bilo pogrešno klasificirano, što znači da model ima točnost od 84,6 % prilikom klasifikacije slova što je veoma slična stopa točnosti s modelom opisanim u ovom radu.

# **7. Zaključak**

Korištenjem izrađene neuralne mreže postigli su se iznenađujuće zadovoljavajući rezultati, kako nad skeniranim riječima tako i nad ručno napisanim slovima. Točnost predviđanja za testirana slova veća je od točnosti nad skupom za treniranje, no pretpostavka je ta što se ovdje koristilo dosta manji uzorak te pošto su riječi ručno napisane vremenski bi bilo veoma zahtjevno testirati isti broj uzoraka koliko ih ima skup za testiranje. Vidimo da postoje sličnosti između pogrešno predviđenih klasa (npr. slovo „t“ koje je klasificirano kao „I“, slovo „f“ kao „F“...) te da točnost klasifikacije uvelike ovisi o rukopisu pojedinca. Za daljne istraživanje moguće je, kao što je već i predloženo, možda sortirati podatke i izbaciti slike znamenki (koje nisu fokus ovoga projekta) te time i smanjiti broj pogrešno klasificiranih slova koji su izgledom slični pojedinim znamenkama ( 0 i „O“, „g“ i 9 i slične) te testirati neuralnu mrežu nad većim skupom podataka koji pojedinac može izraditi ručno.

# **8. Literatura**

1. Anandh K.S., Clintom D. J.: *HANDWRITTEN CHARACTER RECOGNITION USING CNN* [online]. Dostupno na: <https://ijrar.org/papers/IJRAR1903931.pdf> [12.01.2021.]

1. *Break the image of word into different images of character* [online]. Dostupno na: <https://stackoverflow.com/questions/58382937/break-the-image-of-word-into-different-images-of-character> [12.01.2021.]

2. *Convert own image to MNIST's image* [online]. Dostupno na: <https://stackoverflow.com/questions/35842274/convert-own-image-to-mnists-image> [12.01.2021.]

3. Chen J.: *Neural Network* [online]. Dostupno na: <https://www.investopedia.com/terms/n/neuralnetwork.asp> [12.01.2021.]

4. Cohen G., Afshar S., Tapson J., Schaik A: *EMNIST: an extension of MNIST to handwritten letters* [online]. Dostupno na: <https://arxiv.org/abs/1702.05373> [12.01.2021.]

5. *emnist 0.0* [online]. Dostupno na: <https://pypi.org/project/emnist/> [12.01.2021.]

6. Hardesty L.: *Explained: Neural networks* [online]. Dostupno na: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414> [12.01.2021.]

7. Radečić D.: *Softmax Activation Function Explained* [online]. Dostupno na: <https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60> [12.01.2021.]

8. *Rectified Linear Units (ReLU) in Deep Learning* [online]. Dostupno na: <https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning> [12.01.2021.]

9. *Words MNIST* [online]. Dostupno na: <https://www.kaggle.com/backalla/words-mnist> [12.01.2021.]