



Univerzitet u Sarajevu  
Elektrotehnički fakultet  
Računarstvo i informatika



Vještačka inteligencija

## **HTR za bosanski jezik Štampana slova**

Studenti:

Ivona Jozić (19357)  
Ismar Muslić (19304)

Profesorica:

Prof. dr Amila Akagić

Juni 2025. god.

# Sadržaj

Sadržaj.....	1
Sažetak.....	3
Abstract.....	3
1. Uvod.....	4
2. Pregled stanja u oblasti.....	5
2.1 Historijski pregled.....	5
2.2 Konvolucijske neuralne mreže.....	5
2.3 Rekurentne neuralne mreže.....	6
2.4 Hibridni modeli.....	6
2.5 LLM-ovi.....	7
2.6 Prostor za poboljšanje?.....	9
3. Dataset.....	10
3.1 Izrada dataset-a.....	10
3.2 Pregled dataset-a.....	11
3.3 Preprocesiranje dataset-a.....	11
3.3.1 HDF5 format.....	11
3.3.2 Proces prije HDF5.....	12
3.3.2.1 Izvlačenje pojedinačnih riječi.....	12
3.3.2.2 Izvlačenje linija teksta.....	16
3.3.2.3 Izvlačenje linija teksta - modifikacija.....	19
3.3.3 Podjela podataka na podatke za trening i testiranje.....	23
3.3.4 Kreiranje HDF5 formata.....	25
4. Model.....	27
4.1 Arhitektura neuralne mreže.....	27
4.2 Konfiguracija.....	29
4.2.1 Google Drive okolina.....	29
4.2.2 Python klase.....	30
4.3 Treniranje modela.....	33
4.3.1 Treniranje na pojedinačnim riječima.....	35
4.3.2 Treniranje na linijama teksta izrezanih izvornom metodom.....	36
4.3.3 Treniranje na linijama teksta izrezanih modificiranom metodom.....	37
4.4 Testiranje modela.....	38
4.5 Metrike.....	40
5. Analiza rezultata.....	41
5.1 Levenshtein-ova udaljenost.....	41
5.2 CER (Character Error Rate).....	41
5.3 WER (Word Error Rate).....	42

5.4 SER (Sequence Error Rate).....	42
5.5 Rezultati.....	43
5.6 Poređenje sa LLM-ovima.....	46
5.6.1 ChatGPT.....	47
5.6.2 ClaudeAI.....	50
5.6.3 Google Gemini.....	53
5.6.4 Osvrt na rezultate.....	54
6. Zaključak.....	56
7. Popis literature.....	58
8. Popis slika.....	59
9. Popis grafika.....	60
10. Popis tabela.....	60

## Sažetak

Ovaj rad prikazuje proces izrade sistema za prepoznavanje rukom napisanog teksta (HTR) na bosanskom jeziku, fokusiranje na štampana slova. Korišten je HTR-Flor model koji kombinuje konvolucijske i rekurentne neuralne mreže. *Dataset* je ručno kreiran, a tekstovi su segmentirani na riječi i linije korištenjem dvije različite metode segmentiranja. Podaci su konvertovani u HDF5 format i podijeljeni na skupove za treniranje, validaciju i testiranje. Model je evaluiran metrikama CER, WER i SER. Najniže greške su zabilježene pri treniranju na pojedinačnim riječima, dok je izvorna metoda segmentacije linija, uprkos slabijoj preciznosti, dala bolje rezultate od modificirane. Također je izvršeno poređenje sa performansama LLM-ova (GPT-4o, Claude, Gemini), koje su pokazale obećavajuće, ali još uvijek ograničene rezultate za bosanski jezik. Dobijeni rezultati ističu važnost kvalitetne segmentacije i potrebe za dodatnim istraživanjem u domenu HTR-a za manje zastupljene jezike.

## Abstract

This paper presents the process of developing a handwritten text recognition (HTR) system in Bosnian, focused on printed letters. The HTR-Flor model was used, which combines convolutional and recurrent neural networks. The dataset was manually created, and the texts were segmented into words and lines using two different segmentation methods. The data was converted to HDF5 format and divided into training, validation and testing sets. The model was evaluated using the CER, WER and SER metrics. The lowest errors were recorded when training on individual words, while the original line segmentation method, although less consistent, yielded better metric results than the modified one.. A comparison was also made with the performance of LLMs (GPT-4o, Claude, Gemini), which showed promising, but still limited results for the Bosnian language. The obtained results highlight the importance of high-quality segmentation and the need for additional research in the domain of HTR for less represented languages.

# 1. Uvod

Razvoj sistema za prepoznavanje rukom pisanog teksta (*Handwritten Text Recognition* – HTR) predstavlja sofisticiran izazov u polju veštačke inteligencije, računarske vizije i obrade prirodnog jezika. Glavni problem proizlazi iz ogromne varijabilnosti u načinu pisanja – različiti stilovi, debljina linije, naglašavanje i obrisi znakova značajno otežavaju precizno prepoznavanje.

Projekt ima za cilj da razvije model za prepoznavanje rukopisa koristeći HTR-Flor model. Koristit će se *dataset* koji je kreiran u okviru kursa i trenirati model da transkribuje rukopis. Kao metrike za evaluaciju korist ćemo *Character Error Rate* (CER), *Word Error Rate* (WER), kao i *Sequence Error Rate* (SER), koje su standard u polju.

Ovaj projekat obuhvata sljedeće korake:

1. Prikupljanje i anotiranje podataka (*dataset* rukopisa).
2. Preprocesiranje (normalizacija, segmentacija slika, augmentacija).
3. Trening HTR modela.
4. Evaluacija performansi koristeći CER, WER i SER.
5. Analiza rezultata i interpretacija grešaka.

## 2. Pregled stanja u oblasti

U ovom dijelu je izvršena analiza radova koji se bave istom temom kako bismo se upoznali sa dostignućima na ovom polju kao i problemima koji se najčešće javljaju.

### 2.1 Historijski pregled

HTR (*Handwritten Text Recognition*) kao metoda predstavlja značajan izazov u poljima prepoznavanja uzoraka i mašinskog učenja. Ovaj proces je u svom začecu bio zasnovan na heurističkim metodama, koje su ručno sastavljene kako bi pružile neki vid digitalizacije rukom pisanog teksta.

S napretkom mašinskog učenja, izmijenjeni su i pogledi na implementaciju ovog procesa. Došlo je do temeljnih izmjena paradigmi, koje su omogućile razvoj bolje prilagodljivih i robusnijih sistema za prepoznavanje. Najrecentnije, s mogućnostima koje pruža duboko učenje u vidu konvolucijskih i rekurentnih neuralnih mreža, došlo je do neprevaziđenih dostignuća u ovom polju, prvenstveno u vidu performansi. Ove dvije arhitekture omogućile su velike pomake u procesima prepoznavanja rukom pisanog teksta, prvenstveno zbog svojih mogućnosti da efektivno prepoznaju daleke veze i zavisnosti, kao i sekvence varijabilne dužine. Sa prepoznavanja individualnih slova, riječi i linija teksta, danas postoje modeli koji uspješno prepoznaju tekst cijelih stranica ili dokumenata sa velikom pouzdanošću i efikasnošću [1].

### 2.2 Konvolucijske neuralne mreže

Ova arhitektura postala je standard za zadatke prepoznavanja slike, ali se pokazala efektivnom i u zadacima HCR-a (*Handwritten Character Recognition*). Modeli kao što su *LeNet-5* (Yann LeCun, 1998.) i *AlexNet* (Alex Krizhevsky, 2012.), a koji su zapravo konvolucijske neuralne mreže, pokazali su se izrazito kvalitetnim i za HTR zadatke. [2]

*LeNet-5* je klasična konvolucijska neuralna mreža koja je osmišljena primarno za korištenje u zadacima prepoznavanja rukom pisanih cifara. Sastoji se od dva konvolucijska sloja, više *subsampling* slojeva, te potpuno povezanog i izlaznog sloja. Ova arhitektura je bila revolucionarna u svoje vrijeme, postavljajući temelje za daljnji razvoj konvolucijskih neuralnih mreža [2].

*AlexNet* je duboka konvolucijska neuralna mreža koja se proslavila pobjedom na *ImageNet Large Scale Visual Recognition* takmičenju. Ova, dublja i kompleksnija mreža sa više slojeva, omogućila je prepoznavanje i znatno istančanijih detalja na slikama. Iz tog razloga se ista, iako je primarno namijenjena za zadatke prepoznavanja slika, može efektivno koristiti i za prepoznavanje rukom pisanih cifara [2].

## 2.3 Rekurentne neuralne mreže

Rekurentne neuralne mreže široko su korištene za modeliranje sekvenci u prepoznavanju rukom pisanog teksta. Posebno su korisne za prepoznavanje pisanih slova, gdje je od najveće važnosti prepoznavanje konteksta. Standardne RNN arhitekture najčešće se modificiraju kako bi se postigla mogućnost prepoznavanja zavisnosti među sekvencama, čime se daleko poboljšavaju performanse u HTR zadacima. Ove izmjene najčešće uključuju povećanje dubine mreže, kao i precizno uštima vanje aktivacijskih funkcija, kako bi se bolje prepoznavali detalji pisanih slova [2].

Međutim, postoje ipak izvjesna ograničenja prilikom korištenja RNN za HTR zadatke. Primarno je u pitanju problem nestajućeg gradijenta [9], koji izrazito negativno utiče na sposobnosti mreže u prepoznavanju dalekih zavisnosti. Također, ova arhitektura je sklona *overfitting*-u, naročito kada su trening podaci oskudni, što vodi do zahtjeva za velikim računarskim resursima, smanjujući efikasnost procesiranja dužih sekvenci [2].

## 2.4 Hibridni modeli

Nedavna istraživanja uputila su na mogućnost kombiniranja različitih modela, kako bi se istovremeno dobili benefiti više arhitekture. Ovakvi hibridni modeli su dizajnirani s naglaskom na preciznost, obuhvaćajući kako prostorne, tako i sekvencijske zavisnosti rukom pisanog teksta [2].

Korištenje HMM (Hybrid Markov Model) pokazalo se kao izrazito kvalitetno za HTR zadatke međutim također s određenim poteškoćama. Konkretno, u pitanju je bezmemorijski princip rada HMM-a, kao i proces ručnog odabira karakteristika. Iz tog razloga se ovaj model kombinira sa CNN ili RNN, što značajno poboljšava performanse [3].

Kada je u pitanju kombiniranje CNN i RNN, takav model su analizirali istražitelji u [2]. Arhitektura modela je zasnovana na CNN za ekstrakciju

karakteristika, te bidirekcionalnoj rekurentnoj jedinici za procesiranje sekvenci, što omogućava prepoznavanje konteksta i prethodne i naredne sekvence, poboljšavajući sposobnosti interpretacije kompleksnih struktura i uzoraka rukom pisanog teksta, gdje je kontekst od ključnog značaja. Zatim slijedi mehanizam pažnje, implementiran pomoću *sequence-to-sequence* modela, koji dodatno poboljšava hibridnu arhitekturu fokusirajući se na ključne dijelove ulazne sekvence tokom dekodiranja. Ovaj mehanizam dinamički mjeri težine različitih karakteristika, omogućavajući modelu da se koncentriše na kritične uzorke u rukom pisanom tekstu, poboljšavajući sposobnosti razumijevanja konteksta.

Rezultati analize performansi koju su istražitelji sproveli u sklopu [2] ukazuju na nadmoć hibridnog modela u odnosu na pojedinačne CNN ili RNN. Evidentirana je preciznost u prepoznavanju slova od 98.14%, ali i poboljšanje performansi od 2% i 11% u odnosu na samu CNN i RNN respektivno. Performanse hibridnog modela se ogledaju u njegovim sposobnostima da efikasno obuhvati kako prostorne zavisnosti, tako i zavisnosti među sekvencama. Kombinacijom sposobnosti CNN i RNN u singulanu cjelinu, hibridni model uspijeva razumjeti i interpretirati i sitnije detalje u rukom pisanom tekstu. Ovo poboljšanje vodi ka boljoj klasifikaciji slova, čak i sa kompleksnijim i raznovrsnijim rukopisima.

Dodatno, model je pokazao i značajna poboljšanja u WER (Word Error Rate) i SER (Sequence Error Rate) metrikama. Ovime je demonstrirana robusnost i široka mogućnost primjene hibridnog CNN-RNN modela, te efikasnost istog u velikom broju različitih HTR zadataka.

## 2.5 LLM-ovi

Iako prethodno navedeni modeli daju impresivne rezultate, dosadašnji principi treniranja modela za HTR zahtijevaju značajnu količinu detaljno labeliranih podataka, kako bi model savladao i prepoznavanje specifičnih rukopisa. Ovakva zavisnost čini postojeće HTR modele slabo prilagodljivim, naročito kod historijskih tekstova, gdje je pristup takvoj količini podataka oskudan. Prirodno je zaključiti da je ovakav radni tok iscrpljujući i sklon greškama, te zahtijeva ogromne količine vremena [4].

*Large Language* modeli su postali masovno popularni u posljednjih nekoliko godina. Njihova široka primjena i brojne mogućnosti ih čine sjajnim



pomoćnim alatima u brojnim sferama nauke, tehnike, programiranja kao i svakodnevnog života. S time u vidu, mogućnost primjene LLM-ova u HTR zadacima je daleko od zanemarive ideje.

Istražitelji u [4] izvršili su komparativnu analizu nekih od najpopularnijih LLM-ova: *GPT-4o*, *GPT-4o-mini*, te *Claude Sonnet 3.5*, kao i brojnih manjih open-source modela u poređenju sa *Transkribus* modelima, namijenjenim specifično za historijske dokumente i prepoznavanje rukom pisanog teksta u istima. Korišteni su brojni *dataset*-ovi, kako oni s modernim tekstovima, tako i historijskim, neki od kojih su *RIMES*, *IAM*, *LAM* i slični. Također, dizajnirani su specijalizirani *prompt*-ovi kako bi se model što efikasnije iskoristio i što bolje prilagodio problemu. Korektivni *prompt*-ovi pisani su modelima nakon što su utvrđene greške u prenosu iz rukom pisanog u digitalizirani tekst, kako bi se utvrdile sposobnosti samopopravljanja grešaka.

Analiza rezultata *benchmark*-a ukazala je na to da modeli uspješno prepoznaju tekst moderne tematike, pisan rukom na engleskom jeziku, dok kvalitet prepoznavanja progresivno opada kod prepoznavanja teksta pisanog drugim jezicima.

Naročito velika razlika se primjećuje između modernih i historijskih tekstova, gdje određeni modeli daju značajno lošije rezultate. U modernim jezicima, LLM-ovi daju bolje rezultate od *Transkribus* modela s CER manjim od 5%. Na *IAM dataset*-u (engleski jezik, moderni tekst), *GPT-4o-mini* pokazuje CER od 1.71% te WER od 3.34%, dok na *RIMES dataset*-u (francuski jezik, moderni tekst) *GPT-4o* dostiže CER od 1.69% i WER od 3.66%, čime značajno nadmašuje *Transkribus*-ov supermodel. Međutim, prilikom transkripcije historijskog teksta, najbolje rezultate ostvaruje *Transkribus*-ov open-source model *The Text Titan*, sa CER od 7.07% i WER od 12.41%, dok ni LLM-ovi nisu daleko od ovih rezultata [4].

Ovakvi rezultati uzrokovani su inherentnim *bias*-om u podacima za treniranje LLM-ova, s obzirom na to da su isti većinski na engleskom jeziku. Također, razlika između performansi LLM-ova sa modernim u odnosu na historijski tekst dodatno doprinosi ovoj tvrdnji. Stoga su ovi modeli *biased* ne samo u jezičkom smislu, već i u fizičkim karakteristikama rukopisa koje pokušavaju prepoznati [4].

S ovim rezultatima u vidu možemo zaključiti da su LLM-ovi izrazito kvalitetan alat u HTR zadacima na engleskom jeziku, kako s modernim tako i historijskim tekstovima. Međutim, za druge jezike i raznovrsnije rukopise, kvalitet transkripcija koje pružaju ovi alati progresivno opada, s obzirom na to da isti nisu (ili su vrlo malo) trenirani tekstovima na jezicima osim engleskog.

## 2.6 Prostor za poboljšanje?

Kao što je već naglašeno, savremeni modeli trenirani za HTR zadatke i dalje pokazuju značajna ograničenja u pogledu prilagodljivosti, univerzalnosti i jednostavnosti implementacije. Iako *large language* modeli i generativni AI daju impresivne rezultate na engleskom jeziku, utoliko kvalitet tih rezultata opada pri prepoznavanju rukom pisanog teksta na drugim jezicima, naročito za raznovrsnije rukopise, tekst historijskog karaktera, kao i tekst kompleksnijeg sadržaja.

Zahtjev da setovi podataka za treniranje ovakvih modela budu što veći te detaljno labelirani, kao i nepraktičnost činjenice da su sami dizajn i treniranje izrazito dugotrajni procesi (dajući nekad zaista vrlo dobre rezultate ali samo za relativno malen skup rukom pisanih tekstova specifičnih karakteristika) ukazuju na to da postoji značajan prostor za poboljšanje u ovom polju. Međutim, prirodno se nameće pitanje da li je taj prostor zaista moguće ispuniti razvojem sve većih, kompleksnijih i “pametnijih” modela?

Transkripcija rukom pisanih tekstova ostaje izazovan problem, prvenstveno zbog njihove ogromne raznolikosti i kontekstualne složenosti. Imajući to u vidu, opravdano je postaviti tvrdnju da je dizajniranje univerzalnog, “*one-size-fits-all*” modela koji bi s visokim stepenom tačnosti rješavao sve HTR zadatke vjerovatno nedostižan cilj. Ipak, brojni naučnici se trude kombinirati mogućnosti različitih arhitektura i modela kako bi dizajnirali prilagodljivije modele koji bi pokrivali širok spektar HTR zadataka.

### 3. Dataset

Za rad na projektu je korišten dataset koji je kreiran u sklopu predmeta Vještačka inteligencija ranije u toku kursa.

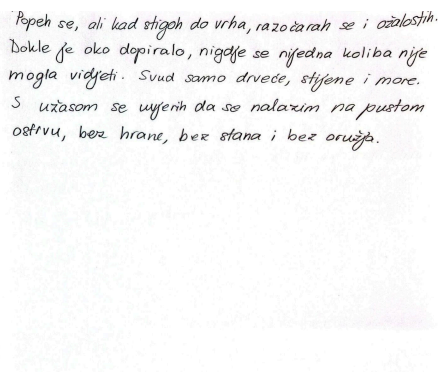
#### 3.1 Izrada *dataset*-a

Objasnit ćemo način na koji je *dataset* keiran.

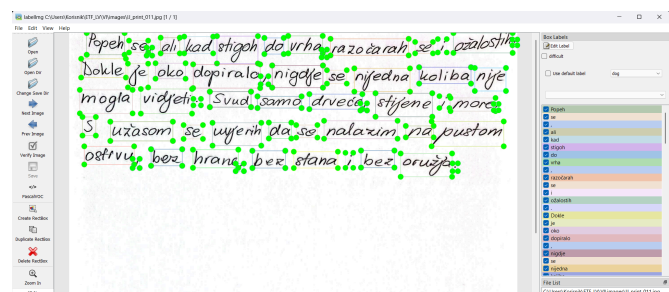
Prvo je bilo potrebno pronaći tekstove iz različitih izvora (internet, štampana literatura, generisati ih pomoću ChatGPT-a) kao i različitog sadržaja (stručna literatura, književna literatura, dijalozi, brojevi, administrativni tekstovi).

Nakon što su tekstovi pronađeni, bilo je potrebno svaki od njih napisati rukom na papiru u tri različita stila - štampanim, pisanim i mix slovima. Naročito je bilo važno voditi računa o tome da je svaki tekst tačno napisan i da odgovara njegovoj transkripciji.

Kada su tekstovi bili napisani, svi su skenirani (rezolucija do 300dpi) kao na slici 1, nakon čega je bilo potrebno označiti svaku pojedinačnu riječ koristeći alat *Labellmg* [5] i PascalVOC format labela (izgled označenog teksta je prikazan na slici 2). Sve slike i njihove oznake u .xml formatu su nakon toga učitane u odgovarajuće foldere.



Slika 1 - Rukom pisani tekst

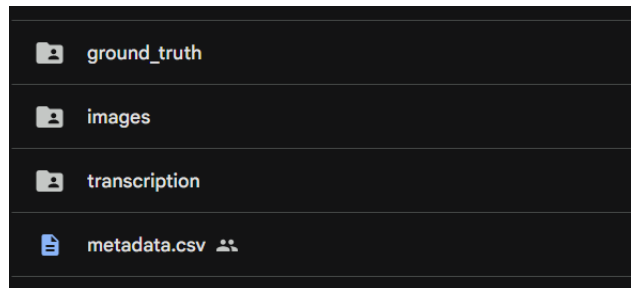


Slika 2 - Označeni tekst

Tokom izrade *dataset*-a su, dakle, kreirani sljedeći folderi (slika 3):

- 1) transcription - u njemu se nalaze .txt sa originalnim tekstovima, na osnovu kojih su tekstovi pisani rukom
- 2) images - u kojem se nalaze .jpg skenirani tekstovi koji su pisani rukom

- 3) ground\_truth - u kojem se nalaze .xml u kojima su oznake za pojedinačni tekst
- 4) metadata.csv - u kojem su nazivi slika, vrsta teksta koju sadrže, kao i inicijali osoba i dob osoba koje su pisale tekstove



Slika 3 - Kreirani folderi tokom izrade dataset-a

## 3.2 Pregled *dataset-a*

Iz kreiranog skupa podataka su izdvojeni tekstovi napisani štampanim slovima i njihove oznake, te je na kraju u *dataset-u* bilo 156 tekstova koje je kreiralo 8 autora. Sve fotografije su bile u .jpg formatu, a oznake za odgovarajuće tekstove u .xml. Nakon izdvajanja dijela podataka vezanog za štampana slova, urađeno je preprocesiranje.

## 3.3 Preprocesiranje *dataset-a*

Kako bi se mogao koristiti HTR-Flor model potrebno je da podaci budu u HDF5 formatu.

### 3.3.1 HDF5 format

HDF5 (*Hierarchical Data Format version 5*) je format *file-a* i set alata dizajniran za spremanje i organizaciju velikih količina podataka. Njegove osobine su:

- Hijerarhijska struktura: Slično kao datotečni sistem – podaci se organizuju u grupe i *dataset-ove*.
- Podrška za različite tipove podataka: Skalarne vrijednosti, nizovi, slike, video, tabele...
- Efikasnost: Optimiziran za rad sa velikim *dataset-ovima* – brzo čitanje i pisanje.

- Prenosivost: Može se čitati i pisati u više programskih jezika (Python, C, C++, Java, MATLAB).
- Kompresija: Podržava kompresiju podataka radi uštede memorije.

### 3.3.2 Proces prije HDF5

Prije nego se podaci iz *dataset*-a prebace u HDF5 format, potrebno je nekoliko modifikacija. S obzirom da će model biti treniran nad pojedinačnim riječima i linijama teksta, potrebno je iz označenih tekstova izvući pojedinačne riječi, a zatim i linije teksta.

#### 3.3.2.1 Izvlačenje pojedinačnih riječi

Potrebno je proći kroz sve .xml u kojima se nalaze oznake riječi sa njihovom tačnom pozicijom na slici kao i labelom, te ih spremiti u jedan .txt file na osnovu kojeg će se vršiti izdvajanje pojedinačnih riječi. Prvo se kreira *word\_labels.txt* u folderu *ground\_truth*, a u koji se spremaju pojedinačne riječi u formatu (1). Potrebno je proći kroz sve .xml *file*-ove u *ground\_truth* folderu i na osnovu njih spremiti riječi. Ovu radnju vrši funkcija na slici 4.

**filename|img\_width|img\_height|word\_idx|xmin|ymin|xmax|ymax|word** (1)

```
def generate_word_labels(path, labels):
    with open(f"{path}/word_labels.txt", "w", encoding="utf8") as file:
        for label in labels:
            xml_file = f"{path}/{label}"
            boxes = read_content(xml_file)
            label = label.removesuffix(".xml")
            for box in boxes:
                b = [str(i) for i in box]
                file.write(f'{"|".join(b)}\n')
```

Slika 4 - Funkcija za izvlačenje pojedinačnih riječi na osnovu .xml

Dodatno se koristi pomoćna funkcija prikazana na slici 5, a koja iterira kroz pojedinačni .xml i izvlači riječ po riječ iz hijerarhije unutar .xml *file*-a.

```
def read_content(xml_file: str):

    tree = ET.parse(xml_file)
    root = tree.getroot()

    list_with_all_boxes = []
    filename = root.find("filename").text

    size_node = root.find("size")
    h = int(size_node.find("height").text)
    w = int(size_node.find("width").text)

    for w_idx, boxes in enumerate(root.iter("object")):
        ymin, xmin, ymax, xmax = None, None, None, None

        ymin = int(boxes.find("bndbox/ymin").text)
        xmin = int(boxes.find("bndbox/xmin").text)
        ymax = int(boxes.find("bndbox/ymax").text)
        xmax = int(boxes.find("bndbox/xmax").text)
        name = str(boxes.find("name").text).upper()

        row = [filename, w_idx, w, h, xmin, ymin, xmax, ymax, name]
        list_with_all_boxes.append(row)

    return list_with_all_boxes
```

Slika 5 - Pomoćna funkcija koja iterira kroz pojedinačni .xml

Da bi se metode izvršile, potrebno je pokrenuti kod prikazan na slici 6, a koji specificira na kojoj lokaciji se kreira .txt, te poziva metode za izdvajanje pojedinačnih riječi, a kasnije i linija teksta.

```
path = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/P

labels = os.listdir(path)
labels = [l for l in labels if l.endswith(".xml")]

generate_word_labels(path, labels)
generate_line_labels(path, labels)
generate_line_labels_m(path, labels)
```

Slika 6 - Definisanje lokacije na koju se sprema word\_labels.txt i pozivanje metode za izdvajanje pojedinačnih riječi i linija teksta

Na kraju ovog procesa dobivamo word\_labels.txt, čiji je sadržaj prikazan na slici 7.

```

LB_print_005.jpg|0|3307|4676|81|86|147|160|"
LB_print_005.jpg|1|3307|4676|149|118|331|248|AKO
LB_print_005.jpg|2|3307|4676|385|128|513|230|TE
LB_print_005.jpg|3|3307|4676|569|122|905|246|PUSTIM
LB_print_005.jpg|4|3307|4676|927|180|973|260|,
LB_print_005.jpg|5|3307|4676|1001|88|1321|234|HOĆEŠ
LB_print_005.jpg|6|3307|4676|1363|106|1445|218|LI
LB_print_005.jpg|7|3307|4676|1487|92|1841|226|DRŽATI
LB_print_005.jpg|8|3307|4676|1891|106|2177|246|JEZIK
LB_print_005.jpg|9|3307|4676|2203|112|2331|214|ZA

```

Slika 7 - Prikaz sadržaja u datoteci *word\_labels.txt*

Nakon što je kreiran *.txt file*, moguće je izdvojiti slike pojedinačnih riječi koristeći slike iz foldera *images* i pohraniti ih u folder *labels\_w*. Za ovo se koristi metoda *generate\_word\_images* prikazana na slici 8. Ova metoda radi na sljedeći način:

- Čita podatke iz *word\_labels.txt* – svaki red opisuje jednu riječ: poziciju riječi unutar slike, ime slike, bounding box koordinate, i samu riječ.
- Učita odgovarajuću sliku iz *src* foldera - u ovom slučaju *images* folder
- Isječe sliku riječi koristeći koordinate *bounding box*-a (pomoću *extract\_rectangle* funkcije).
- Spasi isječenu riječ kao zasebnu *.jpg* sliku u folder *dest/word (labels\_w)*, pod imenom *[originalni\_naziv]\_[index\_rijeci].jpg*.
- Zapiše ažurirane podatke o riječima (s novim imenom slike) u *word.txt* unutar *dest* foldera *labels\_w*.

Pomoćna funkcija *extract\_rectangle* (prikazana na slici 9) radi sljedeće:

- Prima sliku i koordinate gornjeg lijevog (*top\_left*) i donjeg desnog (*bottom\_right*) ugla pravougaonika.
- Ako je zadat *wiggle\_room* (opciono), proširuje taj pravougaonik u svim smjerovima za taj broj piksela (ali pazi da ne izađe izvan slike).
- Vraća izrezani dio slike – pravougaonik između tih koordinata.

```

import cv2

word_labels = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/"

path = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/Projekta"

labels = os.listdir(path)
labels = [l for l in labels if l.endswith(".xml")]

def generate_word_images(word_labels, src, dest):
    with open(word_labels, "r", encoding="utf8") as read:
        with open(f"{dest}/word.txt", "w", encoding="utf8") as write:
            while line := read.readline():
                line = line.removesuffix("\n").split("|")
                filename, word_idx, w, h, xmin, ymin, xmax, ymax, word = line
                img = cv2.imread(f"{src}/{filename}")
                if img is None:
                    print(f"{src}/{filename}")
                word_img = extract_rectangle(img, [int(xmin), int(ymin)], [int(xmax), int(ymax)])
                filename = filename.removesuffix(".jpg")
                img_path = f"{dest}/word/{filename}_{word_idx}.jpg"
                try:
                    cv2.imwrite(img_path, word_img)
                    line[0] = f"{filename}_{word_idx}.jpg"
                    write.write(f"{'|'.join(line)}\n")
                except Exception as e:
                    print(e)

labels_w_path = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencij
if not os.path.isdir(labels_w_path):
    os.mkdir(labels_w_path)

labels_w_word_path = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka intelig
if not os.path.isdir(labels_w_word_path):
    os.mkdir(labels_w_word_path)

generate_word_images(word_labels, path, labels_w_path)

```

Slika 8 - Metoda koja vrši izdvajanje slika pojedinačnih riječi

```

def extract_rectangle(image, top_left, bottom_right, wiggle_room=None):
    if wiggle_room is not None:
        w, h, _ = image.shape
        x1p, y1p = top_left
        x2p, y2p = bottom_right

        top_left = [
            x1p - wiggle_room if 0 <= x1p - wiggle_room <= w else x1p,
            y1p - wiggle_room if 0 <= y1p - wiggle_room <= h else y1p,
        ]
        bottom_right = [
            x2p + wiggle_room if 0 <= x2p + wiggle_room <= w else x2p,
            y2p + wiggle_room if 0 <= y2p + wiggle_room <= h else y2p,
        ]
        x1, y1 = top_left
        x2, y2 = bottom_right
        extracted_region = image[y1:y2, x1:x2]
        return extracted_region

```

Slika 9 - Pomoćna metoda koja vrši izrezivanje slike na osnovu gornjeg lijevog i donjeg desnog ugla



Kao rezultat ovog procesa dobit ćemo pojedinačne riječi koje su sadržane u tekstovima iz *dataset*-a kao što je prikazano na slici 10.



Slika 10 - Primjer pojedinačne slike riječi koju dobijemo na kraju procesa izdvajanja

File *word.txt* koji je kreiran tokom ovog procesa sadrži sve riječi koje su isječene i ima **11 592** redova, što odgovara broju pojedinačnih riječi koje su prisutne u *dataset*-u.

### 3.3.2.2 Izvlačenje linija teksta

Sam proces izvlačenja pojedinačnih linija iz tekstova je veoma sličan procesu izvlačenja riječi uz sitne izmjene. Sada se kreira *line\_labels.txt* u folderu *ground\_truth*, a u koji se spremaju pojedinačne linije u formatu (2). Potrebno je proći kroz sve *.xml* *file*-ove u *ground\_truth* folderu i na osnovu njih spremati linije. Ovu radnju vrši funkcija na slici 11.

**filename|img\_width|img\_height|line\_idx|line\_x\_min|line\_y\_min|line\_x\_max|line\_y\_max|text** (2).

Funkcija *generate\_line\_labels* radi sljedeće:

- Kreira *file line\_labels.txt* u zadatom folderu
- Za svaki *.xml file* čita sve riječi (*bounding box*-ove i tekst) iz XML fajla pomoću *read\_content* metode prikazane na slici 5
- Grupiše riječi u linije pomoću DBSCAN klasterovanja
  - Računa srednje y-koordinate riječi.
  - Pravi matricu vertikalnih udaljenosti između svih riječi.
  - Klasteruje riječi koje su blizu po visini, što odgovara jednoj liniji teksta.
  - *eps=25* znači da riječi unutar 25 piksela visine pripadaju istoj liniji.

- Organizuje riječi po linijama (klasterima) - na osnovu klaster ID-a
- Sortira riječi unutar linije po x-koordinati i rekonstruiše tekst
- Računa *bounding box* koji obuhvata cijelu liniju
- Piše liniju u *line\_labels.txt*

```
from sklearn.cluster import DBSCAN

def generate_line_labels(path, labels):
    with open(f"{path}/line_labels.txt", "w", encoding="utf8") as file:
        for label in labels:
            xml_file = f"{path}/{label}"
            boxes = read_content(xml_file)
            label = label.removesuffix(".xml")
            w0, h0 = boxes[0][2], boxes[0][3]
            filename0 = boxes[0][0]
            per_line = h0 // 11 + 2
            regions = [(i + 1) * per_line, []] for i in range(11)]
            regions.append((h0, []))
            middle = [(box[5] + box[7]) // 2 for box in boxes]
            distances = [[abs(i - j) for j in middle] for i in middle]
            cluster = DBSCAN(eps=25, min_samples=1, metric="precomputed").fit(distances)
            clusters = [i for i in cluster.labels_]
            regions = {}
            for clust, box in list(zip(clusters, boxes)):
                if regions.get(clust, None) is None:
                    regions[clust] = [box]
                else:
                    regions[clust].append(box)

            for l_idx, (clust, boxes) in enumerate(regions.items()):
                boxes = list(sorted(boxes, key=lambda box: box[3]))
                text = " ".join([box[-1] for box in boxes])
                text = text.strip(" ")
                if len(text) == 0:
                    continue
                l_x_min = min([box[4] for box in boxes])
                l_y_min = min([box[5] for box in boxes])
                l_x_max = max([box[6] for box in boxes])
                l_y_max = max([box[7] for box in boxes])
                line = [
                    filename0,
                    l_idx,
                    w0,
                    h0,
                    l_x_min,
                    l_y_min,
                    l_x_max,
                    l_y_max,
                    text,
                ]
                line = "|".join([str(el) for el in line])
                file.write(f"{line}\n")
```

Slika 11 - Metoda za izvlačenje linija teksta

Potrebno je izvršiti kod sa slike 6 kako bi se pokrenulo izvršenje metode *generate\_line\_labels* i kreirao file *line\_labels.txt* u *ground\_truth* folderu, a čiji je sadržaj nakon završetka procesa prikazan na slici 12.

```
LB_print_005.jpg|0|3307|4676|81|84|2927|160|" "
LB_print_005.jpg|1|3307|4676|149|88|2875|248|AKO TE PUSTIM HOĆEŠ LI DRŽATI JEZIK ZA ZUBIMA ?
LB_print_005.jpg|2|3307|4676|927|180|973|260|,
LB_print_005.jpg|3|3307|4676|81|290|2279|440|NISAM SE POTRUDIO DA MU ODGOVORIM .
LB_print_005.jpg|4|3307|4676|79|510|131|582|"
LB_print_005.jpg|5|3307|4676|135|524|2873|698|HOLDENE AKO TE PUSTIM , HOĆEŠ LI DRŽATI JEZIK ZA
LB_print_005.jpg|6|3307|4676|521|632|555|694|,
LB_print_005.jpg|7|3307|4676|141|692|1705|838|ZUBIMA ? REKAO JE PONOVO .
LB_print_005.jpg|8|3307|4676|587|686|631|748|"
LB_print_005.jpg|9|3307|4676|655|786|699|854|,
LB_print_005.jpg|10|3307|4676|85|892|507|968|" "
LB_print_005.jpg|11|3307|4676|163|918|429|1038|HOĆU
LB_print_005.jpg|12|3307|4676|431|992|489|1038|.
KH_print_000.jpg|0|1179|1692|36|24|1088|130|PRILIKOM RJEŠAVANJA PROBLEMA LINEARNOG PROGRAMIRANJA ,
KH_print_000.jpg|1|1179|1692|40|119|997|210|ČESTO JE , RADI LAKŠEG RAČUNANJA , POŽELJNO VRŠITI
```

Slika 12 - Sadržaj *line\_labels.txt*

Može se primjetiti da izvlačenje linija i nije baš najuspješnije, s obzirom da se znakovi navoda i ostali interpunkcijski znakovi zbog velike razlike u visini prepoznati kao posebne linije. Upravo iz ovog razloga smo odlučili modifikovati metodu za izdvajanje linija teksta o čemu će više govora biti u dijelu 3.3.2.3.

Kao i sa pojedinačnim riječima, došao je red na izrezivanje linija teksta. Korišten je isti princip izrezivanja i spremanja fotografija kao i u slučaju riječi, ali je metoda koja to radi prilagođena radu sa linijama i prikazana je na slici 13.

```
def generate_line_images(line_labels, src, dest):
    with open(line_labels, "r", encoding="utf8") as read:
        with open(f"{dest}/line.txt", "w", encoding="utf8") as write:
            while line := read.readline():
                line = line.removesuffix("\n").split("|")
                filename, l_idx, w, h, l_x_min, l_y_min, l_x_max, l_y_max, text = line
                img = cv2.imread(f"{src}/{filename}")
                line_img = extract_rectangle(img, [int(l_x_min), int(l_y_min)], [int(l_x_max), int(l_y_max)])
                filename = filename.removesuffix(".jpg")
                img_path = f"{dest}/line/{filename}_{l_idx}.jpg"
                try:
                    cv2.imwrite(img_path, line_img)
                    line[0] = f"{filename}_{l_idx}.jpg"
                    write.write(f"{'|'.join(line)}\n")
                except Exception as e:
                    print(e)
```

Slika 13 - Metoda za izrezivanje linija teksta

Metoda *generate\_line\_images* radi na sljedeći način:

- Čita liniju po liniju iz *line\_labels.txt* (ground\_truth folder).
- Priprema fajl *line.txt* (u *labels\_1* folderu koji je kreiran za spremanje pojedinačnih slika linija teksta) za pisanje novih podataka.
- Isijeca pravougaoni dio slike u kojem se nalazi linija teksta, koristeći prethodno definisanu funkciju *extract\_rectangle* (slika 9).
- Spašava izrezanu sliku u *labels\_1* i zapisuje podatke u *line.txt*

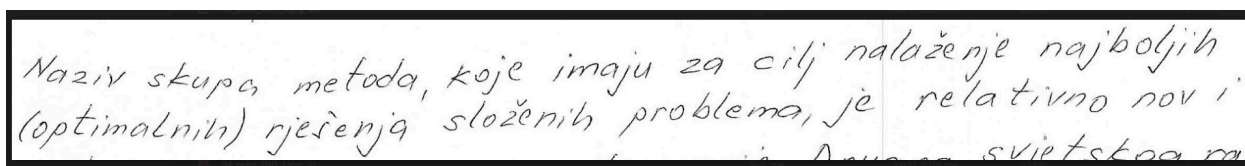
Kao rezultat ovog procesa dobiju se linije teksta prikazane na slikama 14.1, 14.2, 14.3.



Slika 14.1 - Primjer dobro izrezane linije teksta



Slika 14.2 - Primjer u kojem su znakovi navoda prepoznati kao posebna linija teksta



Slika 14.3 - Primjer u kojem je obuhvaćeno više linija teksta zbog preklapanja

Vidljivo je da rezultati variraju, ne samo zbog prosječne visine nego i zbog zbijenog rukopisa u kojem nije moguće razdvojiti dvije linije. Sve ovo bilo je povod za modifikaciju o kojoj će biti riječi u sljedećem dijelu.

Potrebno je istaknuti da je na kraju procesa u *line.txt* bilo upisano **1 497** redova što odgovara broju linija koje se nalaze u ovom *dataset*-u.

### 3.3.2.3 Izvlačenje linija teksta - modifikacija

Kao što je već nagoviješteno u prethodnom potpoglavlju, metoda za izvlačenje linija iz teksta preuzeta iz [7] ima prostora za poboljšanje, pa je projektni tim odlučio istu modificirati.

Osnovna razlika leži u načinu grupisanja riječi u linije. Izvorna funkcija koristi algoritam klasterizacije *DBSCAN* nad vertikalnim središtima riječi (srednja tačka između  $y_{min}$  i  $y_{max}$ ) kako bi grupisala riječi koje se nalaze blizu po visini. Tako formirane grupe se smatraju linijama. Funkcija potom sortira riječi unutar svake grupe po horizontalnoj poziciji i kreira pripadajući tekst i *bounding box*.

Korigovana funkcija, razvijena od strane projektnog tima, ne koristi klasterizaciju, već detektuje prelazak u novi red na osnovu horizontalnog "premotavanja". Ako  $x_{min}$  naredne riječi bude manji od prethodne, pretpostavlja se da je započeta nova linija. Riječi se grupišu u linije prema ovom pravilu, a tekst i *bounding box* se generišu na način da se od *bounding box*-ova koji pripadaju toj liniji pronađu min i max vrijednosti  $x$  i  $y$  koordinata. Unutar tih koordinata se nalazi sav tekst koji pripada toj liniji.

Obje funkcije računaju pozicije linija i pripremaju ih za kasniju ekstrakciju slika, ali njihova pouzdanost zavisi od kvaliteta rasporeda riječi unutar rukom pisanog dokumenta.

Modifikovana funkcija je prikazana na slici 15. Ona koristi pomoćnu funkciju koju je moguće vidjeti na slici 16 - služi za dodavanje razmaka između riječi, a sav ostali postupak je identičan kao i u slučaju sa nemodifikovanom metodom za izvlačenje linija iz rukom pisanog teksta.

```

def generate_line_labels_m(path, labels):
    with open(f"{path}/line_labels_m.txt", "w", encoding="utf8") as file:
        for label in labels:
            xml_file = f"{path}/{label}"
            boxes = read_content(xml_file)

            if not boxes:
                continue

            label = label.removesuffix(".xml")
            w0, h0 = boxes[0][2], boxes[0][3]
            filename0 = boxes[0][0]

            # Group boxes into lines based on wraparound in justified text
            lines = []
            current_line = []

            for i in range(0, len(boxes)):
                prev_x = boxes[i - 1][4] if i > 0 else None
                curr_x = boxes[i][4]

                if prev_x is not None and curr_x < prev_x:
                    lines.append(current_line)
                    current_line = [boxes[i]]
                else:
                    current_line.append(boxes[i])

            if current_line:
                lines.append(current_line)

            # Process each line
            for l_idx, line_boxes in enumerate(lines):
                # Sort boxes within line by xmin
                line_boxes = sorted(line_boxes, key=lambda box: box[4])

                text = assemble_text_with_spaces(line_boxes)

                if not text:
                    continue

                l_x_min = min(box[4] for box in line_boxes)
                l_y_min = min(box[5] for box in line_boxes)
                l_x_max = max(box[6] for box in line_boxes)
                l_y_max = max(box[7] for box in line_boxes)

                line = [
                    filename0,
                    l_idx,
                    w0,
                    h0,
                    l_x_min,
                    l_y_min,
                    l_x_max,
                    l_y_max,
                    text,
                ]
                line_str = "|".join(str(el) for el in line)
                file.write(f"{line_str}\n")

```

Slika 15 - Modificirana funkcija za izvlačenje linija iz teksta

```

import re

def assemble_text_with_spaces(boxes):
    text_parts = []
    punctuation = {'.', ',', '!', '?', ':', ';', }

    for i, box in enumerate(boxes):
        curr_text = box[-1]
        text_parts.append(curr_text)

        next_text = boxes[i + 1][-1] if i + 1 < len(boxes) else None

        if curr_text in punctuation:
            # Dodavanje razmaka nakon znaka interpunkcije
            text_parts.append(" ")
        else:
            # Ako je u pitanju riječ, razmak se dodaje samo ako ne slijedi interpunkcijski znak
            if next_text and next_text not in punctuation:
                text_parts.append(" ")

    text = "".join(text_parts)
    text = re.sub(r'\s+', ' ', text).strip()

    return text

```

Slika 16 - Pomoćna funkcija za dodavanje razmaka u linijama teksta

Nakon što je postupak završen, dobijen je *line\_labels\_m.txt* unutar *ground\_truth* foldera čiji je sadržaj prikazan na slici 17 - vidimo da interpunkcijski znakovi više nisu pojedinačne linije (usporedba sa slikom 12). Na osnovu sadržaja *line\_labels\_m.txt* napravljene su slike linija teksta i pohranjene u folder *labels\_L\_m*. Nekoliko primjera izvučenih slika je dato na slikama 18.1 i 18.2.

```

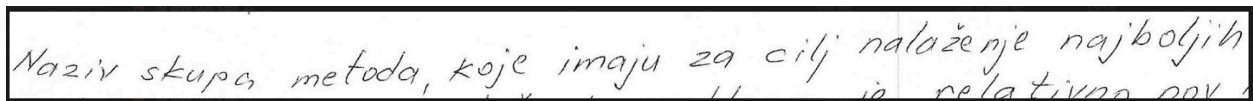
LB_print_005.jpg|0|3307|4676|81|84|2927|260|" AKO TE PUSTIM, HOĆEŠ LI DRŽATI JEZIK ZA ZUBIMA? "
LB_print_005.jpg|1|3307|4676|81|290|2279|440|NISAM SE POTRUDIO DA MU ODGOVORIM.
LB_print_005.jpg|2|3307|4676|79|510|2873|698|" HOLDENE, AKO TE PUSTIM, HOĆEŠ LI DRŽATI JEZIK ZA
LB_print_005.jpg|3|3307|4676|141|686|1705|854|ZUBIMA? ", REKAO JE PONOVO.
LB_print_005.jpg|4|3307|4676|85|892|507|1038|" HOĆU. "
KH_print_000.jpg|0|1179|1692|36|24|1088|130|PRILIKOM RJEŠAVANJA PROBLEMA LINEARNOG PROGRAMIRANJA,
KH_print_000.jpg|1|1179|1692|40|119|997|210|ČESTO JE, RADI LAKŠEG RAČUNANJA, POŽELJNO VRŠITI
KH_print_000.jpg|2|1179|1692|43|196|1091|294|SKALIRANJA OGRANIČENJA. MEĐUTIM, KAD GOD TO RADIMO,
KH_print_000.jpg|3|1179|1692|44|284|1029|381|POTREBNO JE VODITI RAČUNA O MJERNIM JEDINICAMA.
KH_print_000.jpg|4|1179|1692|44|359|1107|464|NAIME, BEZ OBZIRA ŠTO ĆE RJEŠENJE UVIJEK BITI TAČNO

```

Slika 17 - Sadržaj *line\_labels\_m.txt*

" Ako te pustim, hoćeš li držati jezik za zubima? "

Slika 18.1 - Navodnici su sada dio iste linije kao i tekst kojem pripadaju (usporedba sa slikom 14.2)



Slika 18.2 - Linije teksta su jasnije odvojene (usporedba sa slikom 14.3)

Potrebno je istaknuti da je na kraju procesa u *line.txt* u koji su se upisivale linije prilikom izrezivanja iz teksta, bilo upisano **1 359** redova što odgovara broju linija koje se nalaze u ovom *dataset*-u. Jasno je da je broj linija u odnosu na nemodificiranu funkciju smanjen, što upućuje na manje grešaka prilikom izdvajanja interpunkcijskih znakova u posebne linije.

### 3.3.3 Podjela podataka na podatke za trening i testiranje

Nakon što su iz napisanih tekstova izvučene pojedinačne riječi, linije teksta kao i modificirane linije teksta, bilo je potrebno podijeliti dobivene *dataset*-ove na skup podataka za treniranje, validaciju i testiranje. Ovaj postupak je obavljen pomoću funkcije prikazane na slici 19, a koja radi sljedeće:

- Koristi *train\_test\_split* iz *sklearn.model\_selection*
- Na osnovu kreiranih *word.txt* i *line.txt* u toku izrezivanja riječi i linija vrši podjelu i spremanje odgovarajućih labels u tri različita .txt - *trainset*, *testset* i *validset*
- Za treniranje se koristi 70% podataka iz *dataset*-a
- Za validaciju se koristi 20% podataka
- Za testiranje se koristi 10% podataka

Nakon izvršene podjele unutar *dataset*ova se nalazi sljedeći broj labela po grupama:

- *labels\_w* - trening (**8113**), validacija (**2319**), test (**1160**)
- *labels\_l* - trening (**1047**), validacija (**300**), test (**150**)
- *labels\_l\_m* - trening (**951**), validacija (**272**), test (**136**)



```

from sklearn.model_selection import train_test_split

def split_train_set(src_txt: str, out: str, train_size=0.7, val_size=0.2, test_size=0.1, random_state=None):
    assert abs(train_size+val_size+test_size-1.0) < 1e-5, "Train, validation, and test sizes must sum to 1.0"
    elements = []
    with open(src_txt, "r", encoding="utf8") as read:
        while line := read.readline():
            elements.append(line)

    if not elements:
        return 0,0,0

    train_val_data, test_data = train_test_split(elements, test_size=test_size, random_state=random_state)
    val_relative_size = val_size / (train_size + val_size)
    train_data, val_data = train_test_split(train_val_data, test_size=val_relative_size, random_state=random_state)
    with open(f"{out}/trainset.txt", "w", encoding="utf8") as train:
        for el in train_data:
            train.write(el)

    with open(f"{out}/testset.txt", "w", encoding="utf8") as test:
        for el in test_data:
            test.write(el)

    with open(f"{out}/validset.txt", "w", encoding="utf8") as val:
        for el in val_data:
            val.write(el)

    return len(train_data), len(val_data), len(test_data)

_out_w = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/Projekat_VI/HTR_bos1:
_in_w = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/Projekat_VI/HTR_bos1:

_out_l = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/Projekat_VI/HTR_bos1:
_in_l = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/Projekat_VI/HTR_bos1:

_out_l_m = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/Projekat_VI/HTR_bos1:
_in_l_m = "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/Projekat_VI/HTR_bos1:

if not os.path.isdir(_out_w):
    os.mkdir(_out_w)

if not os.path.isdir(_out_l):
    os.mkdir(_out_l)

if not os.path.isdir(_out_l_m):
    os.mkdir(_out_l_m)

split_train_set(_in_w, _out_w)
split_train_set(_in_l, _out_l)
split_train_set(_in_l_m, _out_l_m)

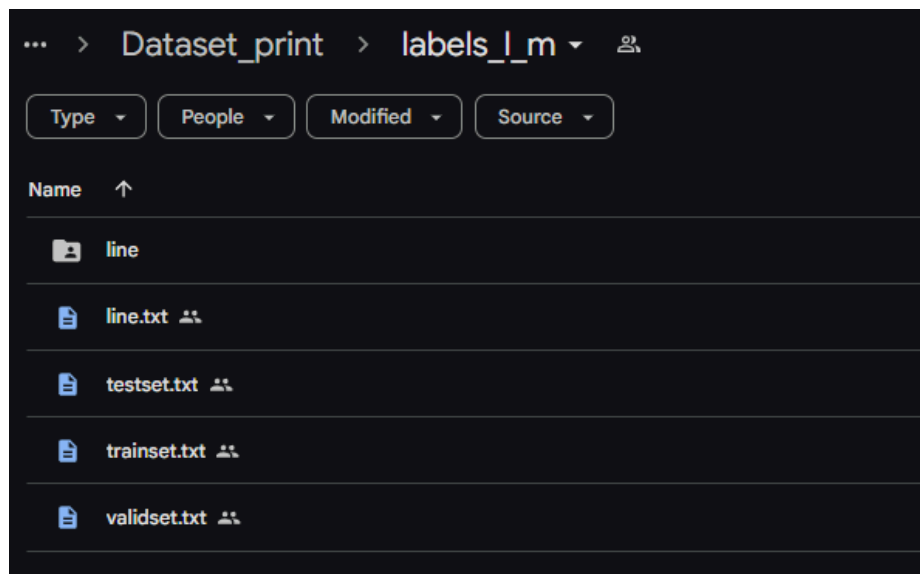
```

Slika 19 - Metoda koja vrši podjelu dataseta-a na trening i testni skup podataka

Na slikama 20 i 21 je prikazan izgled Google Drive foldera u kojem se nalazi *dataset* sa svim kreiranim pojedinačnim *dataset*-ovima za riječi i linije, kao i izgled jednog od foldera sa *dataset*-om, tj. *labels\_L\_m* u kojem se nalaze linije teksta (folder *line*), podaci o kreiranim linijama (*line.txt*), kao i labela podijeljene na trening, validaciju i testne podatke.



Slika 20 - Izgled foldera sa dataset-om nakon preprocesiranja podataka



Slika 21 - Izgled foldera jednog od dataset-ova (labels\_l\_m)

Nakon što su izvršena preprocesiranja i *dataset*-ovi podijeljeni na instance za trening, validaciju i testiranje, potrebno je kreirati HDF5 format koji je neophodan za HTR-Flor model.

### 3.3.4 Kreiranje HDF5 formata

Proces kreiranja HDF5 formata potrebno je napraviti lokalno (ne u sklopu *Google Colab*-a), a sastoji se iz sljedećih koraka:

1. Podesiti verziju Python-a na neku stariju (mi smo koristili 3.11.9)
2. Klonirati handwritten-text-recognition repozitorij [10]
3. U root-u kreirati *raw* folder (ukoliko ne postoji) i u njega dodati *dataset* (čitav *labels\_w* odnosno *labels\_l* i *labels\_l\_m* folder)

4. U terminalu: `python -m venv .venv`
5. U terminalu: `source .venv/Scripts/activate`
6. U terminalu: `pip install -r requirements.txt`
7. U terminalu: `cd src`
8. U terminalu: `python main.py --source=DATASET_NAME --transform`  
(umjesto DATASET\_NAME će biti ili labels\_w ili labels\_l ili labels\_l\_m)
9. Nakon ovoga bi se u *root*-u trebao kreirati data folder, u kojem se nalazi odgovarajući hdf5 file
10. Nakon toga je potrebno *upload*-ati čitave foldere *data* i *src* u *Google Drive* folder za *dataset*.

Kada su *upload*-ana sva tri *dataset*-a .hdf5 formatu, kao i *src* folder u kojem se nalaze potrebne skripte za model i treniranje dataset-a, moguće je preći na sljedeći korak - treniranje i testiranje modela.

## 4. Model

### 4.1 Arhitektura neuralne mreže

FLORe (Full-Gated Convolutional Recurrent) mreža predstavlja specijalizovanu arhitekturu za prepoznavanje rukom pisanog teksta (HTR). Sastoji se od duboke kombinacije konvolucionih i rekurentnih slojeva, uz korištenje CTC (Connectionist Temporal Classification) gubitka koji omogućava treniranje bez eksplicitnog poravnanja karaktera i labela.

Ulaz u mrežu je slika (npr. linija teksta), koja prolazi kroz šest konvolucionih blokova. Svaki blok koristi standardne Conv2D slojeve sa PReLU aktivacijom, a zatim i FullGatedConv2D sloj koji primjenjuje mehanizam ulazne kontrole (*gating*). Ova metoda omogućava selektivno propuštanje informacija kroz paralelnu linearnu i sigmoidnu granu, što se pokazalo korisnim u prepoznavanju varijabilnih obrazaca pisanja. Nakon konvolucija i prostornih transformacija (*stride* i *pooling*), izlazna mapa karakteristika se oblikuje u sekvencu vektora pomoću Reshape sloja.

Sekvenca se zatim prosljeđuje kroz dva dvosmjerna GRU (Gated Recurrent Unit) sloja sa dropout regularizacijom. Ovi slojevi omogućavaju modeliranje vremenskog konteksta (lijevo i desno od trenutnog karaktera), što je ključno za dekodiranje slova iz kompleksnih uzoraka rukopisa.

Na kraju, Dense sloj sa softmax aktivacijom generiše distribuciju vjerovatnoća za svaki karakter u vokabularu po svakom vremenskom koraku. Tokom treniranja koristi se CTC *loss* koji omogućava učenje bez eksplicitnog mapiranja ulaznih i izlaznih sekvenci.

U tabeli 1 je dat detaljan pregled svih slojeva arhitekture neuralne mreže.

Layer (type)	Output Shape	# of Parameters
input (InputLayer)	(None, 1024, 128, 1)	0
conv2d (Conv2D)	(None, 512, 64, 16)	160
p_re_lu (PReLU)	(None, 512, 64, 16)	16
batch_normalization (BatchNormalization)	(None, 512, 64, 16)	64
full_gated_conv2d (FullGatedConv2D)	(None, 512, 64, 16)	4,640
conv2d_1 (Conv2D)	(None, 512, 64, 32)	4,640
p_re_lu_1 (PReLU)	(None, 512, 64, 32)	32
batch_normalization_1 (BatchNormalization)	(None, 512, 64, 32)	128
full_gated_conv2d_1 (FullGatedConv2D)	(None, 512, 64, 32)	18,496
conv2d_2 (Conv2D)	(None, 256, 16, 40)	10,280
p_re_lu_2 (PReLU)	(None, 256, 16, 40)	40
batch_normalization_2 (BatchNormalization)	(None, 256, 16, 40)	160
full_gated_conv2d_2 (FullGatedConv2D)	(None, 256, 16, 40)	28,880
dropout (Dropout)	(None, 256, 16, 40)	0
conv2d_3 (Conv2D)	(None, 256, 16, 48)	17,328
p_re_lu_3 (PReLU)	(None, 256, 16, 48)	48
batch_normalization_3 (BatchNormalization)	(None, 256, 16, 48)	192
full_gated_conv2d_3 (FullGatedConv2D)	(None, 256, 16, 48)	41,568
dropout_1 (Dropout)	(None, 256, 16, 48)	0
conv2d_4 (Conv2D)	(None, 256, 4, 56)	21,560
p_re_lu_4 (PReLU)	(None, 256, 4, 56)	56

batch_normalization_4 (BatchNormalization)	(None, 256, 4, 56)	224
full_gated_conv2d_4 (FullGatedConv2D)	(None, 256, 4, 56)	56,560
dropout_2 (Dropout)	(None, 256, 4, 56)	0
conv2d_5 (Conv2D)	(None, 256, 4, 64)	32,320
p_re_lu_5 (PReLU)	(None, 256, 4, 64)	64
batch_normalization_5 (BatchNormalization)	(None, 256, 4, 64)	256
max_pooling2d (MaxPooling2D)	(None, 256, 2, 64)	0
reshape (Reshape)	(None, 256, 128)	0
bidirectional (Bidirectional)	(None, 256, 256)	198,144
dense (Dense)	(None, 256, 256)	65,792
bidirectional_1 (Bidirectional)	(None, 256, 256)	296,448
dense_1 (Dense)	(None, 256, 108)	27,756

Tabela 1 - Slojevi neuralne mreže

## 4.2 Konfiguracija

Prilikom rada vezanog uz model praćen je postupak i korišten kod iz tutorijal sa repozitorija *handwritten-text-recognition* [10].

Kompletna priprema za treniranje će biti prikazan na jednom od tri *dataset-a* - *labels\_w* (slike pojedinačnih riječi) dok je postupak izveden na preostala dva skoro pa identičan. Sve razlike će biti naglašene.

### 4.2.1 Google Drive okolina

Prvo je bilo potrebno podesiti Google Drive okolinu i *TensorFlow 2.x*, što je prikazano na slici 22. Nakon toga je bilo potrebno mount-ati Google Drive particiju što je vidljivo unutar druge ćelije na slici 22.

```
2.1) TensorFlow 2.x

[ ] # pokretati na GPU

%tensorflow_version 2.x
import tensorflow as tf

device_name = tf.test.gpu_device_name()

if device_name != "/device:GPU:0":
    raise SystemError("GPU device not found")

print("Found GPU at: {}".format(device_name))

Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
Found GPU at: /device:GPU:0

2.2) Google Drive

[ ] %cd "/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/Projekat_VI/HR_bos1: HTR za bosanski jezik (stampana slova)/Dataset_print/src"
ls -l

/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/Projekat_VI/HR_bos1: HTR za bosanski jezik (stampana slova)/Dataset_print/src
total 36
drwx----- 2 root root 4096 Jun 19 11:00 data
-rw----- 1 root root 0 Jun 16 21:01 __init__.py
-rw----- 1 root root 11532 Jun 16 21:35 main.py
drwx----- 2 root root 4096 Jun 19 11:00 network
-rw----- 1 root root 16178 Jun 12 13:07 tutorial.ipynb
```

Slika 22 - Podešavanje Google Drive okoline i TensorFlow-a; mount particije

## 4.2.2 Python klase

Prvenstveno je potrebno podesiti *environment* varijable:

- source - izvor podataka, tj. odgovarajući *dataset*
- arch - mreža koju koristimo - flor
- epochs - broj epoha u treniranju (ostavljeno na 1000 kao i u [10])
- batch\_size - broj uzoraka koji se istovremeno obrađuju u jednoj iteraciji prilikom treniranja modela (za pojedinačne riječi je 64, dok je za linije smanjen na 16)

Nakon toga smo definirali iz kojeg *dataset*-a se uzimaju podaci (na slici 23 je to *labels\_w* ali je rađeno i sa preostala dva), te u koji folder se spremaju rezultati treniranja i testiranja.

Dodatno su definisani *input size* (oblik ulazne slike (dimenzije i kanali)), maksimalna dužina linije, te set karaktera koji se uzima u obzir - pored osnovnog smo dodali i afrikate karakteristične za bosanski jezik.

```
▼ 3.1) Environment

[ ] # Priprema za slike pojedinačnih riječi
import datetime
import string

# define parameters
source = "labels_w"
arch = "flor"
epochs = 1000
batch_size = 64

# define paths
source_path = os.path.join("../", "data", f"{source}.hdf5")
output_path = os.path.join("../", "output", source, arch)
target_path = os.path.join(output_path, "checkpoint_weights.weights.h5")
os.makedirs(output_path, exist_ok=True)

# define input size, number max of chars per line and list of valid chars
input_size = (1024, 128, 1)
max_text_length = 128
charset_base = string.printable[:95]
charset_base = charset_base + "čćđšžž"

print("source:", source_path)
print("output:", output_path)
print("target:", target_path)
print("charset:", charset_base)

source: ../data/labels_w.hdf5
output ../output/labels_w/flor
target ../output/labels_w/flor/checkpoint_weights.weights.h5
charset: 0123456789abcdefghijklmnopqrstuvwxyz!#$%&'()*+,-./:;<=>?@[\]^_`{|}~ čćđšžž
```

Slika 23 - Podešavanje Python klase

Kada su definisane varijable, izvorišne i odredišne lokacije, pozvana je klasa *DataGenerator* čija je svrha da omogući učitavanje podataka (trening, validacijski i testni dio podataka), te da generiše podate u *batch*-evima. Ova klasa koristi i *Tokenizer* klasu koja pretvara tekst u nizove *integer*-a i obrnuto – brojčane nizove nazad u tekst. Implementacije ovih klasa je moguće naći u folderu `Dataset_print/src/data/generator.py`, ali kako smo samo pozivali ovu klasu ovdje nije prikazana njena implementacija. Na slici 24 je prikazan način korištenja klase.

```
▼ 3.2) DataGenerator Class

[ ] import sys
sys.path.append('/content/drive/MyDrive/ETF/Prvi ciklus studija/Treća godina/6. semestar/Vještačka inteligencija/Projekat_VI/

[ ] from data.generator import DataGenerator

dtgen = DataGenerator(source=source_path,
                      batch_size=batch_size,
                      charset=charset_base,
                      max_text_length=max_text_length)

print(f"Train images: {dtgen.size['train']}")
print(f"Validation images: {dtgen.size['valid']}")
print(f"Test images: {dtgen.size['test']}")

Train images: 8113
Validation images: 2319
Test images: 1160
```

Slika 24 - Korištenje *DataGenerator* klase za učitavanje podataka



Vidimo da se broj učitanih podataka poklapa sa brojem podataka iz dijela 3.3.3 u kojem su podaci podijeljeni na trening, validacijski i testni skup.

Sada je potrebno učitati HTR model koji se nalazi u `Dataset_print/src/network/model.py`. To radimo pomoću koda prikazanog na slici 25. Konkretni model je prethodno detaljnije opisan u dijelu 4.1.

```
▼ 3.3) HTRModel Class

from network.model import HTRModel

# Kreiranje i kompajliranje modela
model = HTRModel(architecture=arch,
                  input_size=input_size,
                  vocab_size=dtgen.tokenizer.vocab_size,
                  beam_width=10,
                  stop_tolerance=20,
                  reduce_tolerance=15,
                  reduce_factor=0.1)

model.compile(learning_rate=0.001)
model.summary(output_path, "summary.txt")

# Učitavanje postojećeg modela (ako postoji)
model.load_checkpoint(target=target_path)

callbacks = model.get_callbacks(logdir=output_path, checkpoint=target_path, verbose=1)
```

Slika 25 - Učitavanje HTR modela

HTRModel klasa razvijena je kako bi bila apstrahirala kompliciranu logiku HTR sistema. Odgovorna je za sljedeće:

- Kreiranje modela za prepoznavanje rukom pisanog teksta, u kojem se računa funkcija gubitka pomoću CTC (Connectionist Temporal Classification) i dekodiranje izlaza radi izračunavanja metrika HTR-a (CER, WER i SER).
- Čuvanje i učitavanje modela.
- Učitavanje težina u modele (za treniranje/inferenciju).
- Izvršavanje procesa treniranja/predviđanja pomoću generatora.
- Da bi HTRModel bio dinamičan, njegovi parametri su arhitektura, veličina ulaza (`input_size`) i veličina vokabulara (`vocab_size`).

## 4.3 Treniranje modela

U dijelu vezanom za treniranje osvrnut ćemo se na treniranje svakog od pojedinačnih *dataset*-ova.

Kod pomoću kojeg je vršeno treniranje prikazan je na slici 26. Ovaj kod radi sljedeće:

- Započinje mjerenje vremena prije treniranja (*start\_time*).
- Trenira model pomoću generatora podataka (*dtgen*) i funkcije *model.fit()*, uz validaciju na validacionom skupu, praćenje gubitka (*loss*) i korištenje *callback*-ova.
- Izračunava ukupno vrijeme treniranja i druge statistike:
  - Najniži validacioni gubitak i u kojoj se epohi desio.
  - Prosječno vrijeme po epohi i po uzorku.
- Sastavlja izvještaj o treniranju u obliku teksta (*t\_corpus*), koji uključuje broj slika za treniranje / validaciju, *batch* veličinu, vrijeme, broj epoha, najbolju epohu i gubitke.
- Upisuje izvještaj u fajl *train.txt* u zadati *output\_path* (*output* folder unutar *Dataset\_print* za odgovarajući *dataset*) i ispisuje ga u konzolu.

Ono što je važno napomenuti je mogućnost ranijeg zaustavljanja koju posjeduje model, a koja zaustavlja treniranje ukoliko se vrijednost validacijskog gubitka ne smanji u 20 epoha. Ova mogućnost je bila aktivirana prilikom treniranja sva tri *dataset*-a te nijedan nije treniran na predviđenom maksimalnom broju epoha (1000).

Nakon završenog treniranja, kreirani su i grafici na kojima je moguće vidjeti odnos između gubitka treniranja (mjeri koliko dobro model odgovara trening podacima) i gubitka validacije (provjerava mogućnost modela da generalizira neviđene podatke). Kod pomoću kojeg su grafici kreirani je prikazan na slici 27, a konkretni grafici će biti prikazani i analizirani prilikom analiziranja trening procesa pojedinačnih *dataset*-ova.

Prije nego se napravi osvrt na cjelokupan proces treniranja modela na pojedinačnim *dataset*-ovima, važno je istaknuti to da je treniranje vršeno u *Google Colab*-u uz korištenje ograničenih resursa. Treniranje linija teksta nije

bilo vremenski zahtjevno, te se uklapalo u dnevnu količinu dodijeljenih resursa. Kada je riječ o treniranju *dataset*-a sa pojedinačnim riječima, treniranje je trajalo duže nego što su dozvoljavali dostupni resursi te je bilo neophodno kombinirati više korisničkih računa kako bi se maksimizirala količina dostupnih resursa i kako bi se treniranje uspješno privelo kraju.

```
[ ] start_time = datetime.datetime.now()

# Treniranje modela
h = model.fit(x=dtgen.next_train_batch(),
              epochs=epochs,
              steps_per_epoch=dtgen.steps['train'],
              validation_data=dtgen.next_valid_batch(),
              validation_steps=dtgen.steps['valid'],
              callbacks=callbacks,
              shuffle=True,
              verbose=1)

total_time = datetime.datetime.now() - start_time

loss = h.history['loss']
val_loss = h.history['val_loss']

min_val_loss = min(val_loss)
min_val_loss_i = val_loss.index(min_val_loss)

time_epoch = (total_time / len(loss))
total_item = (dtgen.size['train'] + dtgen.size['valid'])

t_corpus = "\n".join([
    f"Total train images:      {dtgen.size['train']}",
    f"Total validation images: {dtgen.size['valid']}",
    f"Batch:                    {dtgen.batch_size}\n",
    f"Total time:                {total_time}",
    f"Time per epoch:            {time_epoch}",
    f"Time per item:             {time_epoch / total_item}\n",
    f"Total epochs:              {len(loss)}",
    f"Best epoch                 {min_val_loss_i + 1}\n",
    f"Training loss:              {loss[min_val_loss_i]:.8f}",
    f"Validation loss:           {min_val_loss:.8f}"
])

with open(os.path.join(output_path, "train.txt"), "w") as lg:
    lg.write(t_corpus)
    print(t_corpus)
```

Slika 26 - Treniranje HTR modela

```

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

# --- Loss ---
plt.subplot(1, 2, 1)
plt.plot(h.history['loss'], label='Gubitak treniranja')
plt.plot(h.history['val_loss'], label='Gubitak validacije')
plt.title('Gubitak po epohama')
plt.xlabel('Epoha')
plt.ylabel('Gubitak')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

```

Slika 27 - Kod koji vrši iscrtavanje grafika za trening i validacijski gubitak

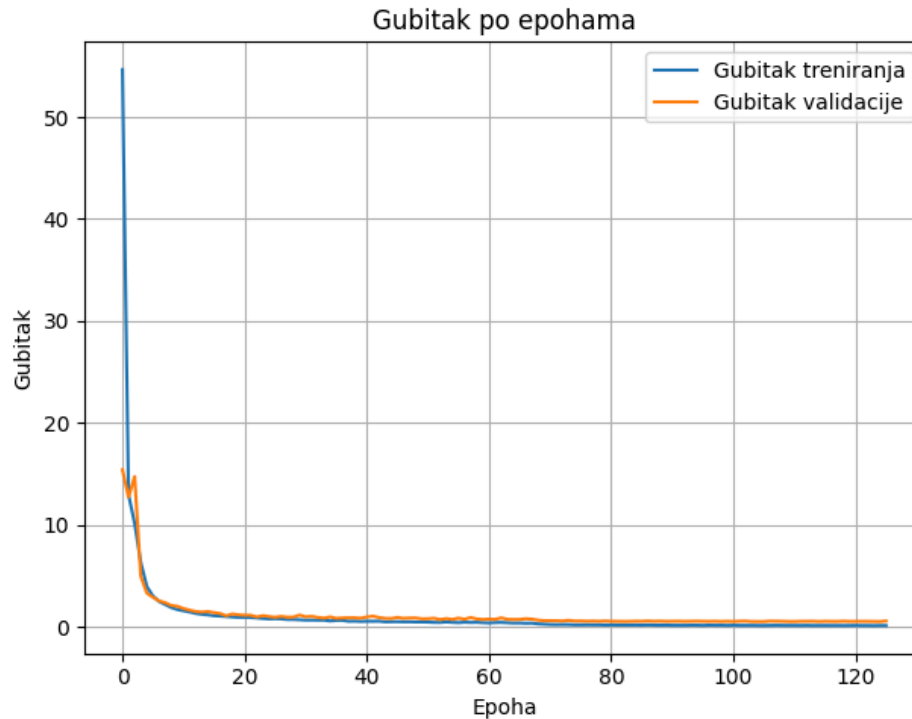
### 4.3.1 Treniranje na pojedinačnim riječima

Kada je u pitanju proces treniranja na pojedinačnim riječima, treniranje je prošlo kroz 126 epoha i trajalo je nešto manje od 4h, koristeći T4 GPU Runtime na *Google Colab*-u. Najbolji rezultati su postignuti u 106. epohi, a treniranje je zbog *Early stopping*-a završeno nakon 126 epoha. Detalje o samom treniranju je moguće vidjeti na slici 28.

Total time:	3:54:22.086345
Time per epoch:	0:01:51.603860
Time per item:	0:00:00.010698
Total epochs:	126
Best epoch	106
Training loss:	0.11550388
Validation loss:	0.50083512

Slika 28 - Treniranje modela na pojedinačnim riječima

Na grafiku 1 je moguće vidjeti na koji način su se mijenjali gubitak treniranja i validacije kroz epohe, dok nisu dostigli najbolje vrijednosti od 0.12 i 0.50 respektivno. Kao što je vidljivo sa grafika nije došlo do *overfitting*-a jer validacijski gubitak prati gubitak treniranja.



Grafik 1 - Gubitak po epohama za treniranje dataset-a u kojem su pojedinačne riječi

#### 4.3.2 Treniranje na linijama teksta izrezanih izvornom metodom

U slučaju ovog dataset-a je treniranje završeno nakon 101 epohe i 32 minute. Najbolji rezultati su postignuti u 81. epohi, a treniranje je zbog *Early stopping*-a završeno nakon 101 epohe. Detalje o samom treniranju je moguće vidjeti na slici 29.

```

Total time:           0:32:03.886557
Time per epoch:      0:00:19.048382
Time per item:       0:00:00.014162

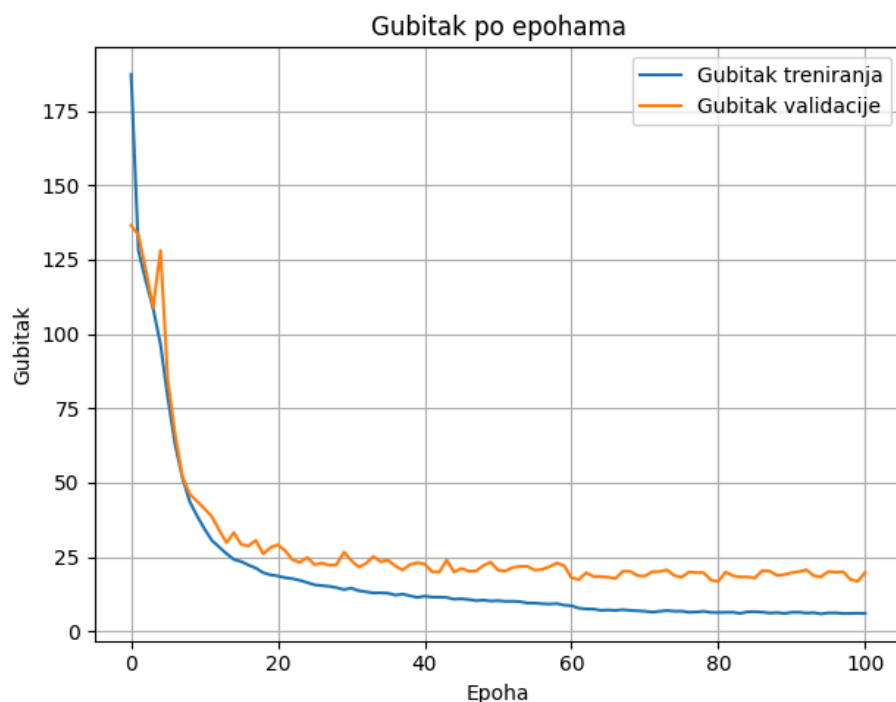
Total epochs:        101
Best epoch            81

Training loss:        6.34184980
Validation loss:      16.79165840
  
```

Slika 29 - Detalji treniranja modela na linijama teksta

Na grafiku 2 je moguće vidjeti na koji način su se mijenjali gubitak treniranja i validacije kroz epohe, dok nisu dostigli najbolje vrijednosti od 6.34 i 16.79

respektivno. Kao što je vidljivo sa grafika došlo je do *overfitting*-a jer validacijski gubitak veći od gubitka treniranja. Model je dobro naučio podatke, ali je loš kod generalizacije.



Grafik 2 - Gubitak po epohama za treniranje dataset-a u kojem su linije teksta

#### 4.3.3 Treniranje na linijama teksta izrezanih modificiranom metodom

Treniranje je završeno nakon 135 epoha, a najbolji rezultati su postignuti u 115. Treniranje je trajalo oko 42 minute ponovno koristeći T4 GPU Runtime, a završeno je zbog *Early stopping*-a. Detalji su prikazani na slici 30.

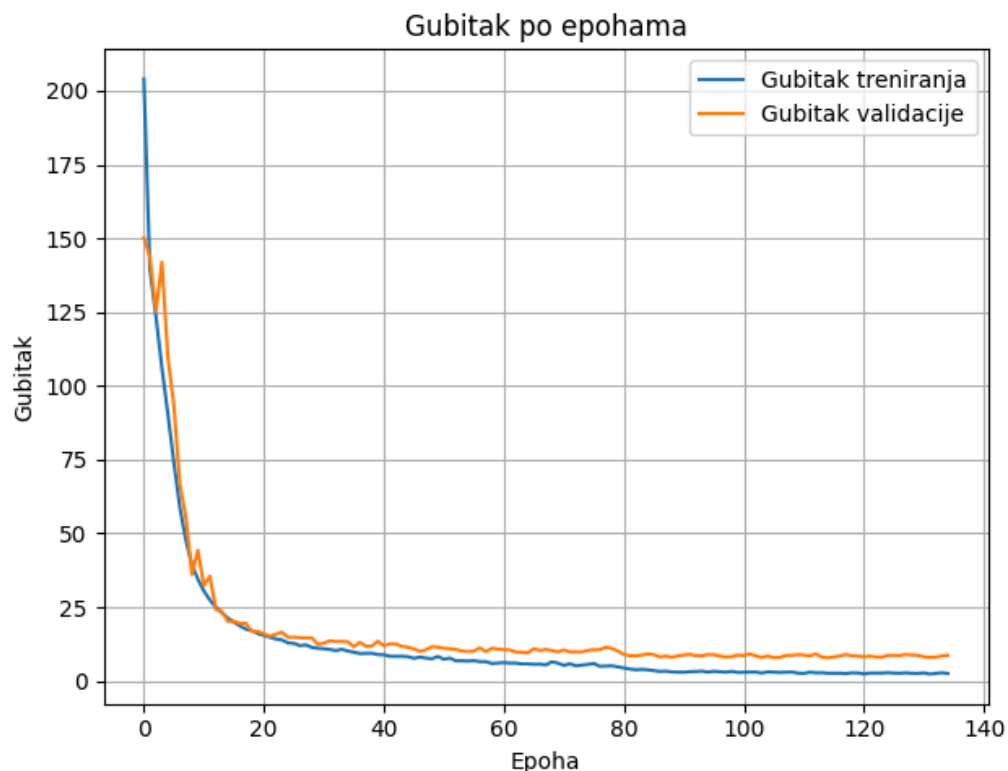
```
Total time:          0:42:03.257401
Time per epoch:      0:00:18.690796
Time per item:       0:00:00.015283

Total epochs:        135
Best epoch            115

Training loss:        2.61487579
Validation loss:       7.84266376
```

Slika 30 - Detalji treniranja modela na modificiranim linijama teksta

Na grafiku 3 je moguće vidjeti na koji način su se mijenjali gubitak treniranja i validacije kroz epohe, dok nisu dostigli najbolje vrijednosti od 2.61 i 7.84 respektivno. Kao što je vidljivo sa grafika došlo je do *overfitting*-a jer validacijski gubitak veći od gubitka treniranja. Za razliku od prethodnog *dataset*-a, ovaj model ima nešto bolje vrijednosti gubitaka.



Grafik 3 - Gubitak po epohama za treniranje *dataset*-a u kojem su modificirane linije teksta

## 4.4 Testiranje modela

Kada je riječ o testiranju modela, ono je rađeno pomoću metode *preprocess* koja se nalazi u `Dataset_print/src/data/preproc.py`. Poziv funkcije je prikazan na slici 31. Prikazani kod mjeri vrijeme koje je potrebno za testiranje, pretvara predikciju koja je niz slova u string i zapisuje podatke u odgovarajući file unutar *output* foldera.

```
[ ] from data import preproc as pp
    from google.colab.patches import cv2_imshow

    start_time = datetime.datetime.now()

    # predict() funkcija vraća vjerovatnoće predikcije za uzorke test dataseta
    predicts, _ = model.predict(x=dtgen.next_test_batch(),
                                steps=dtgen.steps['test'],
                                ctc_decode=True,
                                verbose=1)

    # Dekodiranje predikcija iz numeričkih vektora u stringove
    predicts = [dtgen.tokenizer.decode(x[0]) for x in predicts]

    # Učitavanje stvarnih (ground truth) stringova za test skup
    ground_truth = [x.decode() for x in dtgen.dataset['test']['gt']]

    total_time = datetime.datetime.now() - start_time

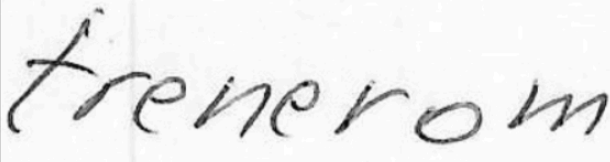
    with open(os.path.join(output_path, "predict.txt"), "w") as lg:
        for pd, gt in zip(predicts, ground_truth):
            lg.write(f"TE_L {gt}\nTE_P {pd}\n")
```

Model Predict  
19/19 5s 210ms/step  
CTC Decode  
19/19 63s 3s/step

Slika 31 - Kod koji pokreće testiranje modela

Nakon testiranja modela, prikazano je nekoliko prvih rezultata. Kada je riječ o konkretnim rezultatima oni su prikazani u sljedećem poglavlju, a ovdje ćemo prikazati kod (slika 32) koji prikazuje prvih 20 testnih slika uz tačnu labelu i tekst koji je model prepoznao.

```
# Prikaz prvih 20 testnih slika zajedno s labelama i predikcijama
for i, item in enumerate(dtgen.dataset['test']['dt'][:20]):
    print("=" * 1024, "\n")
    cv2_imshow(pp.adjust_to_size(item))
    print("Ispravna labela: ", ground_truth[i])
    print("Predikcija modela: ", predicts[i], "\n")
```



Ispravna labela: TRENEROM  
Predikcija modela: TRENEROM

Slika 32 - Prikaz jedne od 20 testnih slika sa rezultatima na dataset-u sa pojedinačnim riječima



## 4.5 Metrike

Metrike koje su korištene su WER (Word Error Rate), CER (Character Error Rate) i SER (Sequence Error Rate) o kojima se više govori u sljedećem poglavlju, a ovdje će biti prikazan kod za njihovo računanje. Metrike se računaju na osnovu funkcije `ocr_metrics` koja se nalazi u `Dataset_print/src/data/evaluation.py` i upoređuje stvarne labela sa onima koje je pročitao model. Prikazani kod na slici 33 poziva spomenutu funkciju, a zatim ispisuje dobivene rezultate.

```
from data import evaluation

evaluate = evaluation.ocr_metrics(predicts, ground_truth)

e_corpus = "\n".join([
    f"Total test images:      {dtgen.size['test']}",
    f"Total time:             {total_time}",
    f"Time per item:          {total_time / dtgen.size['test']}\n",
    f"Metrics:",
    f"Character Error Rate: {evaluate[0]:.8f}",
    f"Word Error Rate:      {evaluate[1]:.8f}",
    f"Sequence Error Rate:  {evaluate[2]:.8f}"
])

with open(os.path.join(output_path, "evaluate.txt"), "w") as lg:
    lg.write(e_corpus)
    print(e_corpus)
```

```
⇒ Total test images:      1160
Total time:              0:01:08.301057
Time per item:           0:00:00.058880

Metrics:
Character Error Rate: 0.03117975
Word Error Rate:     0.09310345
Sequence Error Rate: 0.09310345
```

Slika 33 - Kod za računanje metrika i njihov ispis

## 5. Analiza rezultata

U HTR zadacima kvalitet sistema procjenjuje se pomoću tri ključne metrike: *CER* (Character Error Rate), *WER* (Word Error Rate) i *SER* (Sentence Error Rate). Sve tri mjere temelje se na *Levenshtein-ovoj udaljenosti*, koja kvantificira razliku između prepoznatog i stvarnog teksta na osnovu minimalnog broja operacija zamjene, brisanja ili umetanja.

### 5.1 Levenshtein-ova udaljenost

Levenshtein-ova udaljenost (poznata i kao *edit distance*) između dvije sekvence predstavlja najmanji broj elementarnih operacija:

- Umetanje (insertion),
- Brisanje (deletion),
- Zamjena (substitution),

potrebnih da se jedna sekvenca transformiše u drugu. Koristi se kao osnova za izračun grešaka na nivou znakova, riječi i rečenica [8].

#### **Primjer:**

Uporedimo riječi *Stol* (referentni tekst) i *Stop* (prepoznati tekst)

Koraci transformacije:

Zamijeniti slovo *l* sa *p* → "Stol" → "Stop"

(Ukupno: 1 zamjena)

Dakle, Levenshtein-ova udaljenost između "Stol" i "Stop" iznosi 1.

### 5.2 CER (*Character Error Rate*)

*Character Error Rate* je metrika zasnovana na stopi grešaka u pojedinačnim znakovima. Predstavlja procenat pogrešno prepoznatih znakova u odnosu na ukupni broj znakova u referentnom tekstu.

CER se računa na sljedeći način:

$$CER = \frac{S + D + I}{N} \times 100\%$$

gdje S predstavlja broj zamjena znakova (*substitutions*), D broj brisanja znakova (*deletions*), I broj umetanja znakova (*insertions*), dok N predstavlja ukupan broj znakova u referentnom tekstu.

### 5.3 WER (*Word Error Rate*)

*Word Error Rate* je metrika koja se bazira na greškama u riječima. Predstavlja procenat pogrešno prepoznatih riječi u odnosu na ukupan broj riječi u referentnom tekstu.

WER se računa na sljedeći način:

$$WER = \frac{S + D + I}{N} \times 100\%$$

gdje oznake S, D i I predstavljaju iste vrijednosti kao kod CER, samo na nivou riječi a ne pojedinačnih znakova, dok oznaka N predstavlja ukupan broj riječi u referentnom tekstu.

### 5.4 SER (*Sequence Error Rate*)

*Sequence Error Rate* označava postotak sekvenci u kojima se javlja makar jedna greška. U zavisnosti od sistema i *dataset*-a, jedna "sekvenca" može biti rečenica, linija rukopisa, cijeli unos korisnika ili neki drugi segment. SER je korisna metrika u situacijama gdje je bitna tačnost cijelih logičkih jedinica, a ne samo pojedinačnih riječi i znakova, te se ista smatra dobrom mjerom razumijevanja konteksta od strane modela. Dodatno, ova metrika je najstrožija od svih navedenih, s obzirom na to da sa samo jednom greškom na nivou riječi ili znaka, cijela sekvenca smatra netačnom.

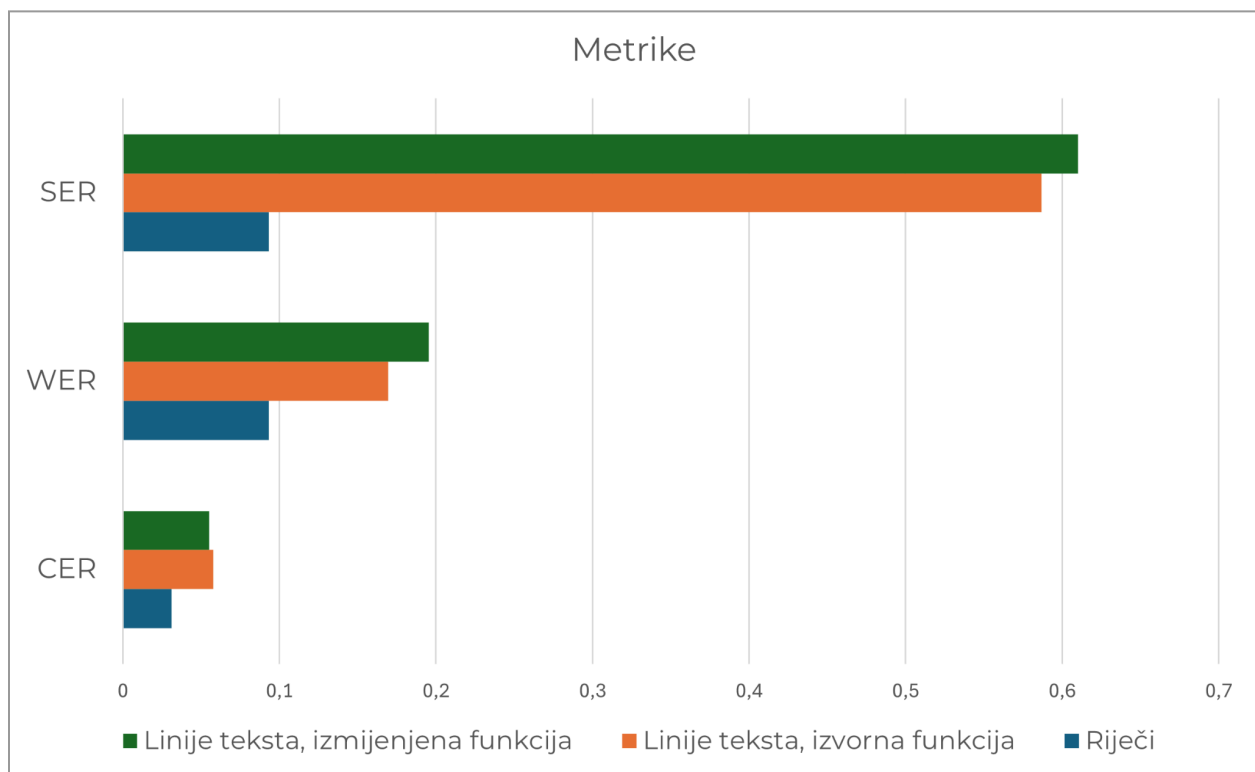
SER se računa na sljedeći način:

$$SER = \frac{N_{err}}{N_{total}} \times 100\%$$

gdje  $N_{err}$  predstavlja broj pogrešnih sekvenci, a  $N_{total}$  ukupan broj sekvenci.

## 5.5 Rezultati

Analiza objašnjenih metrika nakon treniranja modela sa sva tri *dataset*-a (pojedinačne riječi, te linije teksta izvlačene dvjema različitim funkcijama) dala je sljedeće rezultate:



Grafik 4 - Prikaz metrika dobijenih nakon testiranja modela

Kod *dataset*-a sa pojedinačnim riječima dobijene su konzistentno jako dobre vrijednosti svih metrika, CER, WER i SER. Bitno je naglasiti da je u ovom slučaju vrijednost SER i WER jednaka, s obzirom na to da je jedna sekvenca kod seta podataka koji sadrži samo riječi zapravo - jedna riječ. Stoga greška u riječi automatski znači grešku u sekvenci.

Kod setova podataka sa linijama teksta, bolje vrijednosti metrika postignute su korištenjem izvorne funkcije za izvlačenje linija teksta, iako ta funkcija često nije radila ispravno. U nekim slučajevima, izdvajala bi cijele pasuse kao linije, a češće samo pojedine riječi ili interpunkcijske znakove. S druge strane, modificirana funkcija demonstrirala je konzistentno ponašanje u segmentaciji: dobijene slike su skoro uvijek sadržavale tačno jednu liniju teksta, osim u slučajevima kada se rukopis fizički preklapao, što nijedna funkcija nije mogla u potpunosti izbjeći. Korigovana funkcija je pritom, dizajnirana tako da se uvijek generiše tačno onoliko slika koliko ima linija u pisanom tekstu.

Uprkos tome, metričke vrijednosti ukazuju na to da korigovana funkcija daje slabije rezultate od izvorne. Ovakvo intrigantno ponašanje može se pripisati načinu na koji svaka funkcija obrađuje strukturu i šum u rukom pisanom tekstu.

*Izvorna funkcija* koristi DBSCAN klasteriranje na osnovu vertikalnog položaja, grupišući riječi u linije prema visinskoj blizini. Iako ova metoda nerijetko izoluje interpunkciju ili spaja susjedne linije, često proizvodi fragmentirane ili pojednostavljene ulaze. Ove kratke, često trivijalne segmente linija HTR modelu je lakše ispravno prepoznati, posebno pod metrikama kao što je SER, gdje čak i jedna netačna riječ rezultuje greškom u cijeloj sekvenci. Posljedično, model postiže vještački niže stope grešaka na takvim ulazima.

Nasuprot tome, *korigovana funkcija* identificira prekide linija na osnovu horizontalnog pomaka u pisanju, pretpostavljajući da novi red počinje kada se riječ pojavi dalje lijevo od prethodne. Ova logika daje konzistentnije ekstrakcije u cijelom redu, koje bolje prate stvarnu strukturu teksta. Međutim, osjetljiva je na nagnut, zbijen ili preklapajući rukopis što dovodi do spojenih ili pogrešno poravnatih linija. Ovi složeni unosi povećavaju greške u sekvencama i na nivou riječi, jer jedan pogrešno prepoznat token u dugom redu utiče na cijeli metrički rezultat.

Kako je ranije spomenuto, SER tretira svaku grešku u znaku ili riječi kao potpunu grešku sekvence. S time u vidu, sekundarna funkcija za izvlačenje linija, iako je ispravnija u smislu pravilnog generisanja pojedinih linija teksta, kontraintuitivno daje lošije rezultate upravo iz razloga što su linije koje generiše konzistentno sadrže više riječi odnosno znakova, a ukupan broj linija je manji. Samim time raste i vjerovatnoća da će u jednoj takvoj liniji doći do greške.

S druge strane, “linije” koje generiše izvorna, tj. neispravna funkcija su često bili izolirani interpunkcijski znakovi, same riječi ili generalno kraći dijelovi teksta, pa je statistički gledano manja vjerovatnoća da se napravi greška u sekvenci. Također, broj sekvenci koje ova funkcija generiše je, upravo zbog grešaka u procesu segmentacije veći, pa je uticaj eventualne greške u jednoj sekvenci manji za ukupnu metriku SER-a.

Iako korigovana funkcija preciznije odražava strukture linija iz stvarnog svijeta, njena osjetljivost na nepravilnosti rasporeda i duže, gušće ulaze vjerovatno je dovela do nešto lošijih metričkih performansi. U međuvremenu, fragmentirani izlazi neispravne metode išli su u korist modela pojednostavljujući zadatak prepoznavanja, uprkos lošijem semantičkom poravnanju.

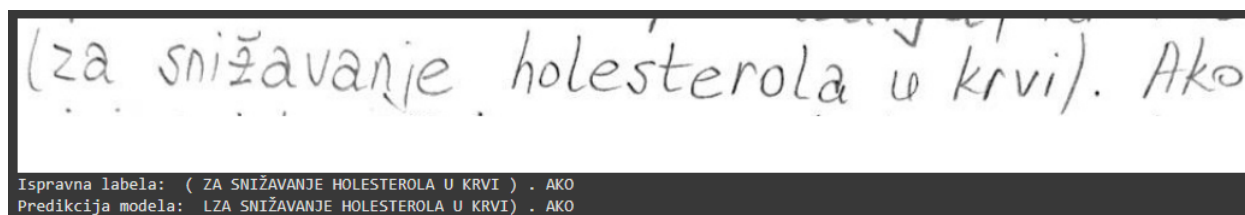
Kada su u pitanju metrike CER i WER, ubjedljivo najbolje rezultate daje model treniran na *dataset*-u sa pojedinim riječima, u odnosu na linije teksta. Ovo je očekivan rezultat, s obzirom na to da ovaj dataset po sekvenci (riječi) sadrži manje znakova, pa je i vjerovatnoća greška manja. Uz to, analizirajući slike samih riječi, modelu je lakše prepoznati i sitnije detalje u pojedinačnim slovima / znakovima.

Model treniran na *dataset*-u kreiranom korigovanom funkcijom za generisanje linija teksta daje marginalno bolje rezultate od *dataset*-a generisanog izvornom funkcijom, kada je u pitanju CER metrika. Ovakav rezultat je uzrokovan činjenicom da modificirana funkcija generiše konzistentnije slike sa ujednačenom količinom teksta, za razliku od izvorne, koja nerijetko uključi više linija ili cijeli tekst u jednu sliku. Modelu je stoga lakše prepoznati individualne znakove ukoliko je treniran na setu podataka gdje su sami znakovi veće rezolucije, te ih ima manje, a kakve slike upravo i generiše modificirana funkcija.

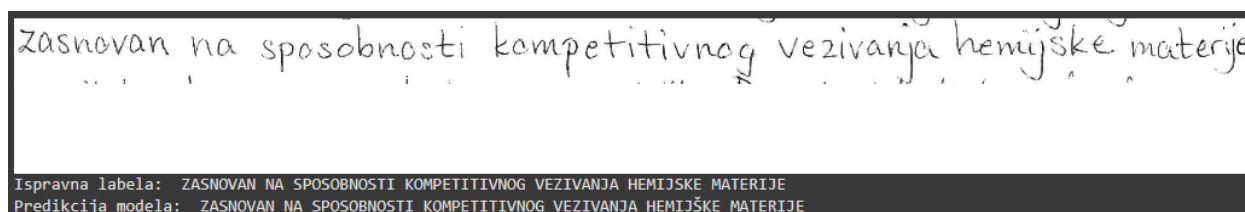
Kada je u pitanju WER metrika, *dataset* generisan izvornom funkcijom daje bolje rezultate, ponovno iz istog razloga kao što je prethodno objašnjeno u vezi SER metrike. Nekonzistentnost u količini teksta pri generisanju slika linija istog rezultovala je većom količinom slika koje sadrže samo jednu riječ ili jedan znak, što je ponovno išlo u prilog modelu prilikom prepoznavanja pojedinih riječi.

## 5.6 Poređenje sa LLM-ovima

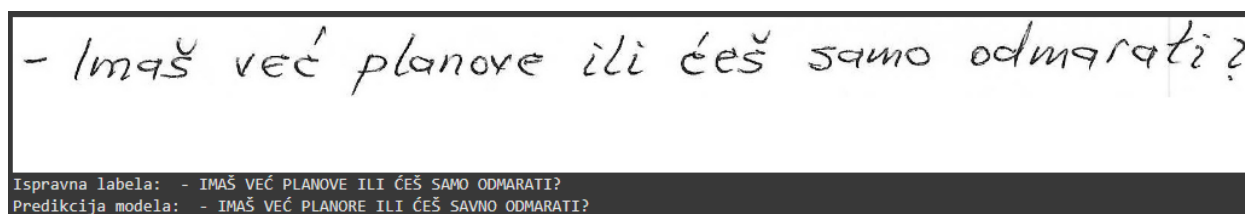
Analizom testnih primjera na linijama teksta, projektni tim je uočio određene nepravilnosti u funkcionisanju modela. Naime, model bi nekada mijenjao, odnosno pogrešno prepoznavao rukopisom slična slova, kao što su recimo c i e. Nekoliko primjera ovog ponašanja dato je na fotografijama ispod, koje sadrže sliku linije teksta, ispravnu labelu i predikciju modela.



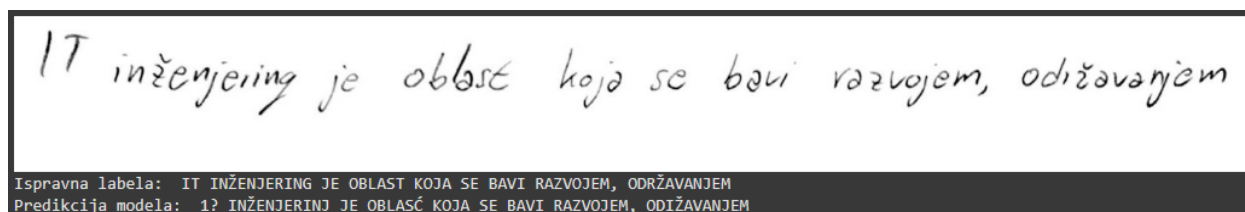
Slika 34 - Pogrešno prepoznata otvorena mala zagrada kao slovo l



Slika 35 - Preneseno slovo j iz prethodnog reda pogrešno prepoznato kao kvačica na š



Slika 36 - Slovo v pogrešno prepoznato kao r; m kao spojeno v i n



Slika 37 - Slovo l prepoznato kao broj 1; malo slovo t kao ć; slovo r kao i

Kao što je vidljivo na primjerima, ovo ponašanje bi se moglo svrstati pod "honest mistakes" modela u prepoznavanju, gdje bi zbog nepreciznog rukopisa predikcija modela zaista mogla biti ispravna. Neki bolno očigledni primjeri su vidljivi na slikama 35 i 37. U primjeru na slici 35 model je nesretno

prepoznao prenesen donji dio slova j iznad slova s kao kvačicu na slovu š, iako je u pitanju riječ “hemijski”. Zatim, u primjeru na slici 37 model griješi na nekoliko mjesta: veliko slovo l je prepoznato kao broj 1, zatim malo slovo t, napisano vrlo iskrivljenim rukopisom je prepoznato kao slovo ć (na što zaista i liči), te je slovo r prepoznao kao slovo i, s obzirom da je u rukopisu zaista slovo r napisano tako da nalikuje na i, bez tačke.

Model sa boljim sposobnostima prepoznavanja konteksta mogao bi zaključiti da slovo m nisu spojena slova v i n, ili da u riječi “hemijske” nema slova š. Međutim, model koji je projektni tim trenirao, osim što nije pokazao izvanredne sposobnosti prepoznavanja konteksta, to nije niti bilo u ciljevima projektnog zadatka. Stoga je projektni tim odlučio uporediti rezultate s onima koji bi dali popularni, javno dostupni LLM-ovi.

Slike linija teksta koje su odabrane za primjere iznad poslane su jedna za drugom LLM-ovima, sa sljedećim promptom:

*“I am uploading an image that contains handwritten text. Please scan the image, recognize the handwritten text, and return the result as clean, digitalized plain text.*

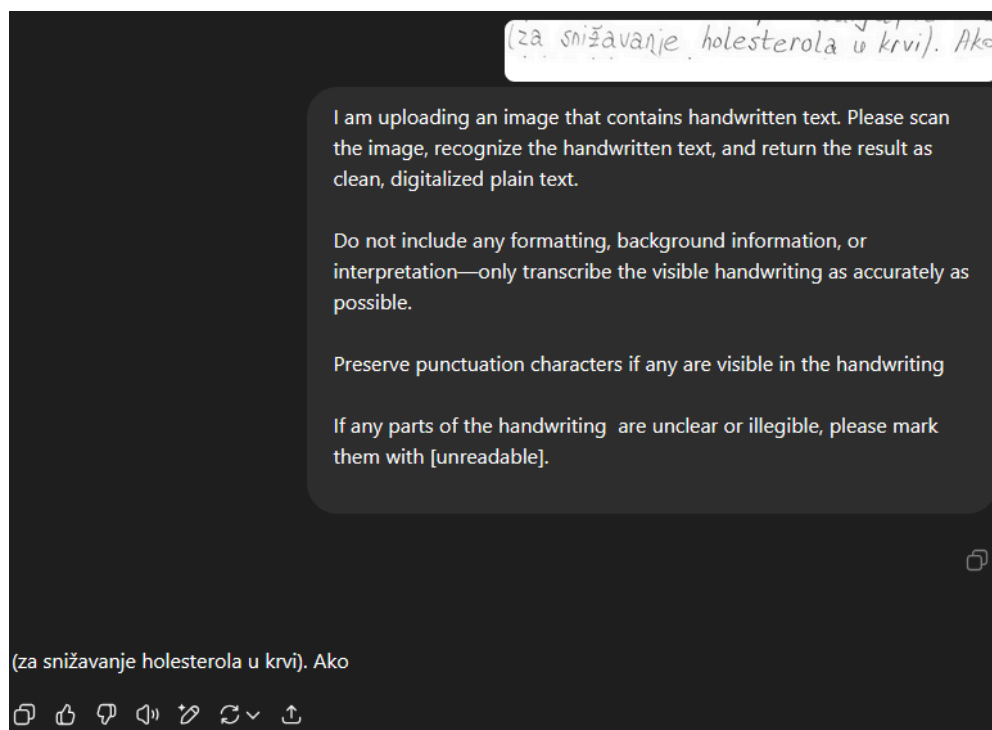
- Do not include any formatting, background information, or interpretation—only transcribe the visible handwriting as accurately as possible.*
- Preserve punctuation characters if any are visible in the handwriting*
- If any parts of the handwriting are unclear or illegible, please mark them with [unreadable].”*

Odabrani LLM-ovi za testiranje i poređenje su ChatGPT, ClaudeAI i Google Gemini.

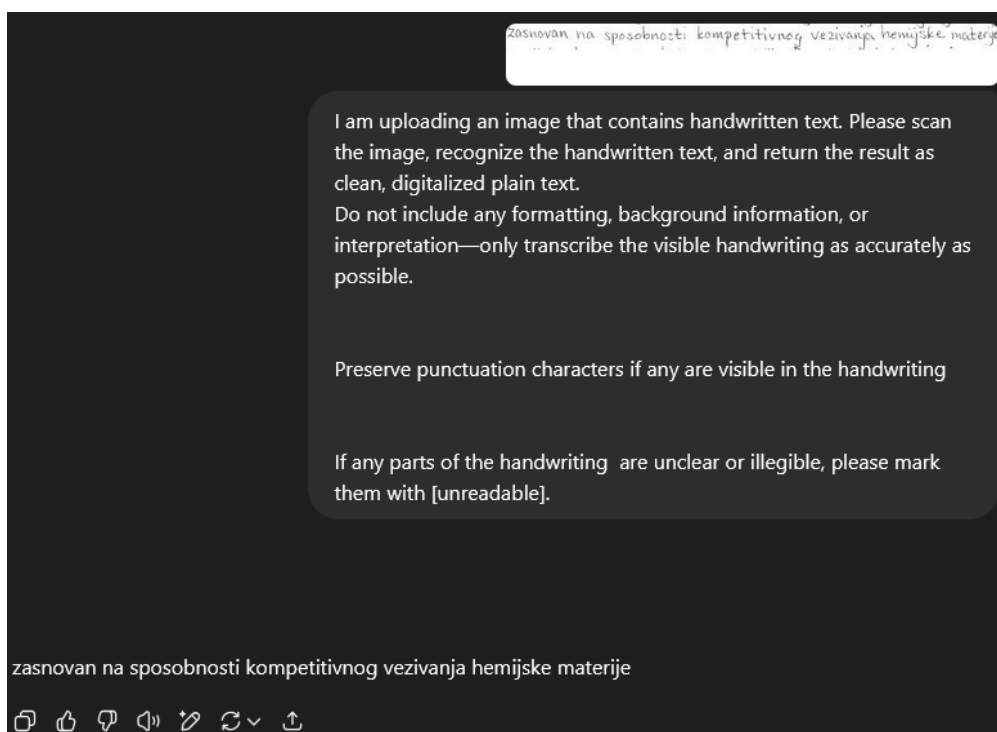
### 5.6.1 ChatGPT

Korištena verzija ChatGPT modela je GPT-4o. Dobiveni rezultati su vidljivi na slikama 38 - 41.

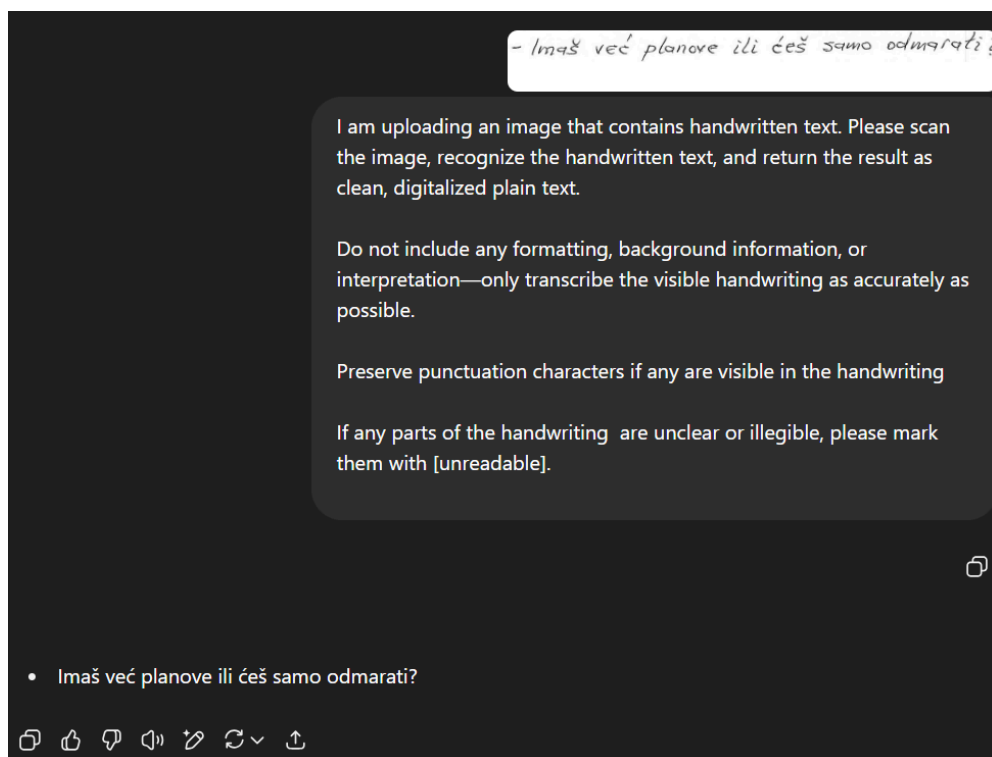




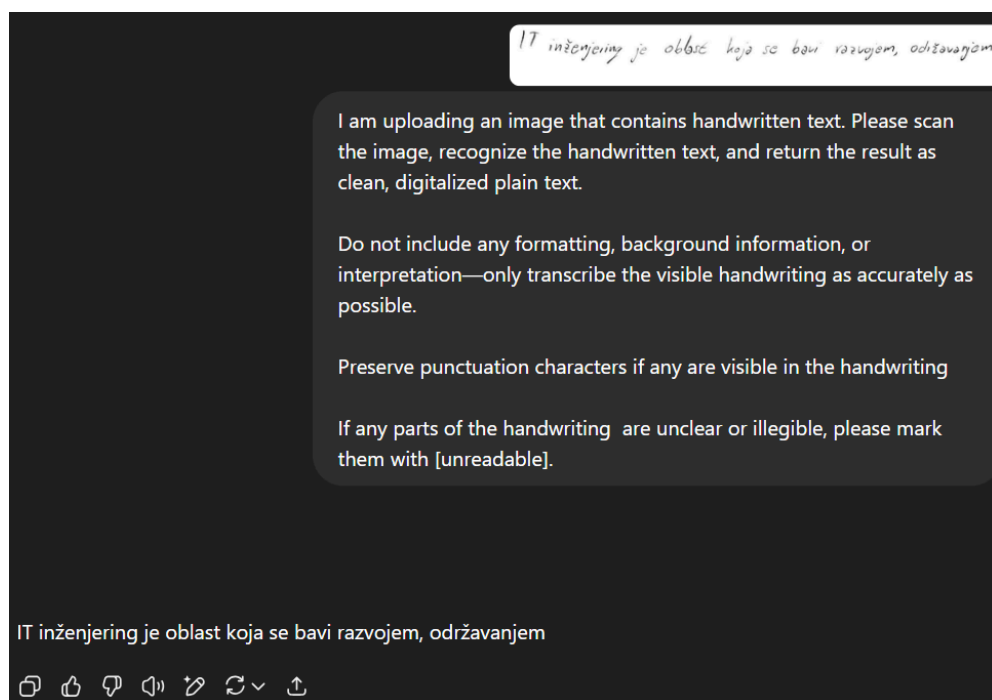
Slika 38 - Rezultat ChatGPT-ja prilikom skeniranja prve slike



Slika 39 - Rezultat ChatGPT-ja prilikom skeniranja druge slike



Slika 40 - Rezultat ChatGPT-ja prilikom skeniranja treće slike

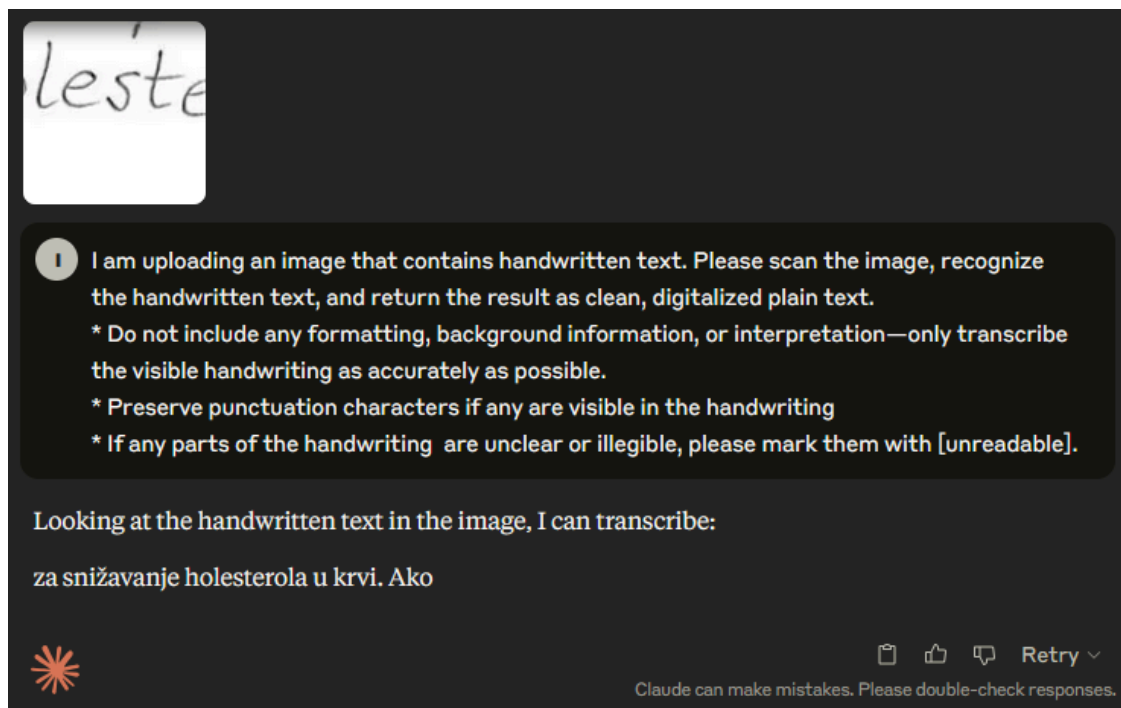


Slika 41 - Rezultat ChatGPT-ja prilikom skeniranja četvrte slike

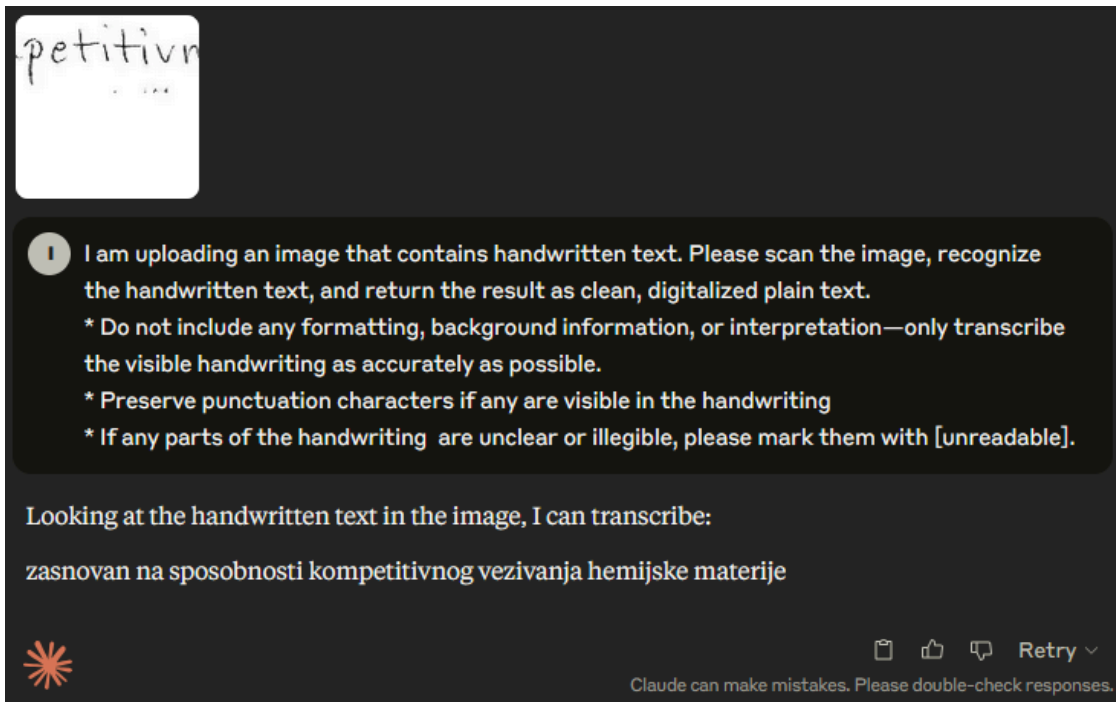
Kao što je vidljivo na slikama 38 - 41, ChatGPT potpuno ispravno prepoznaje sav tekst na slikama, neovisno o tome kakvim su rukopisom napisane.

## 5.6.2 ClaudeAI

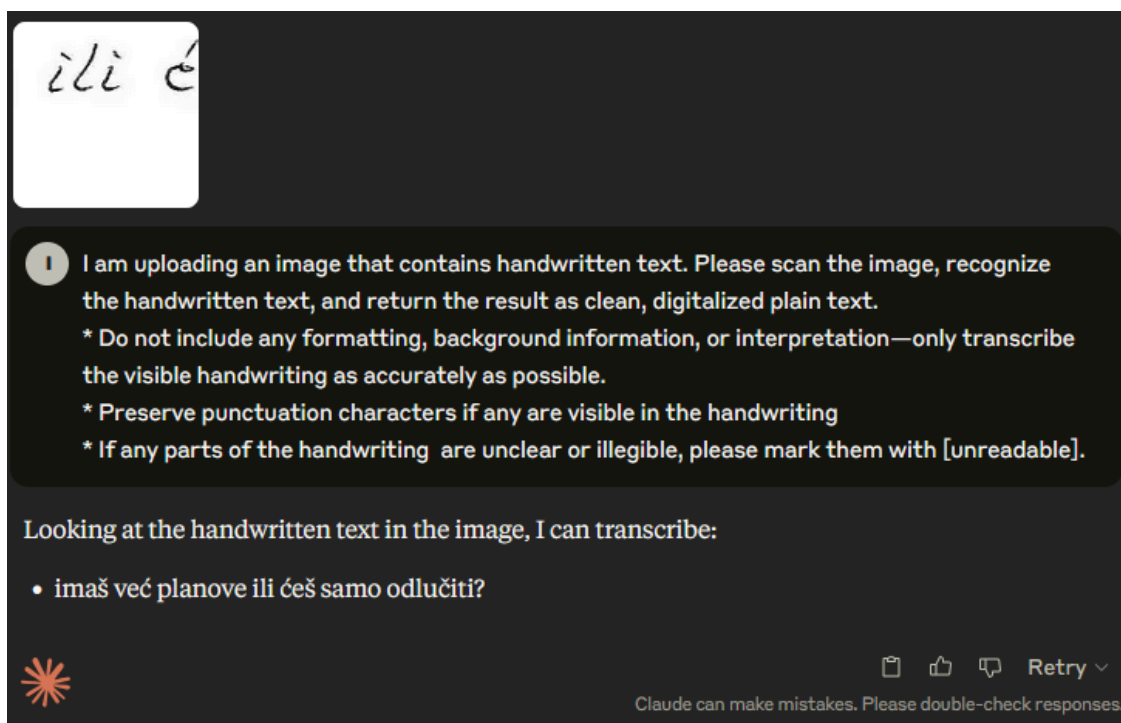
Korištena verzija ClaudeAI modela je Claude Sonnet 4, njihova najmoćnija verzija modela. Rezultati slijede na slikama 42 - 45.



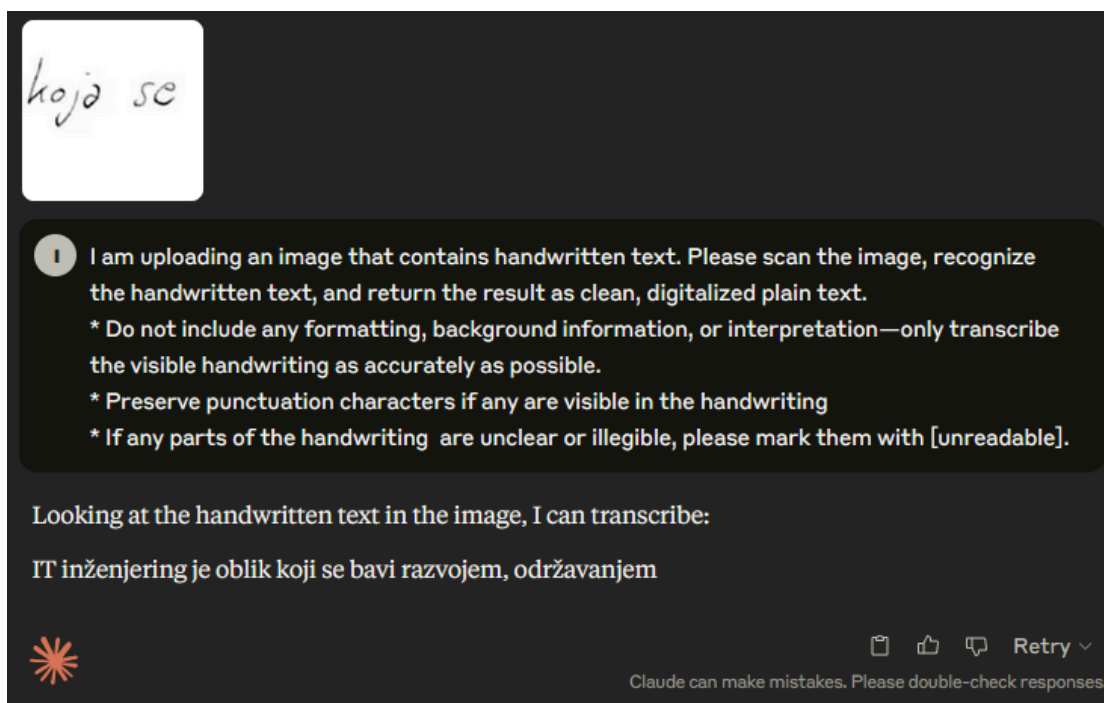
Slika 42 - Rezultat ClaudeAI prilikom skeniranja prve slike



Slika 43 - Rezultat ClaudeAI prilikom skeniranja druge slike

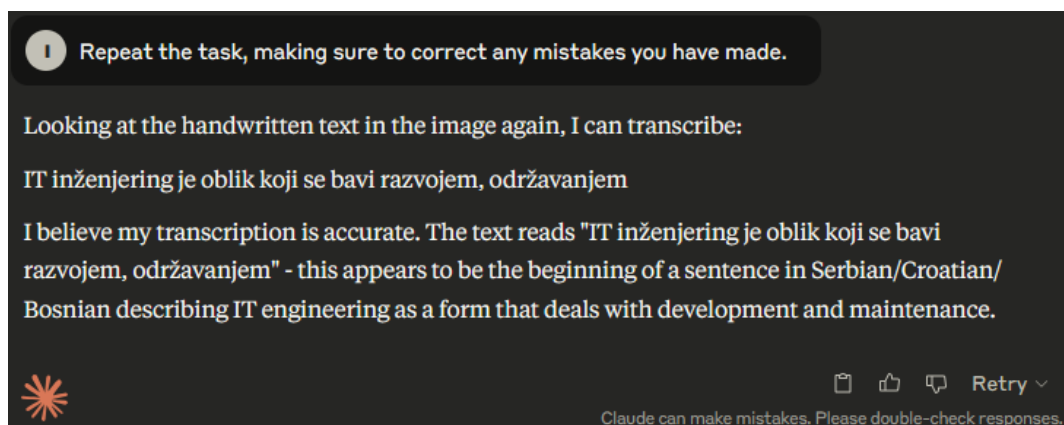


Slika 44 - Rezultat ClaudeAI prilikom skeniranja treće slike



Slika 45 - Rezultat ClaudeAI prilikom skeniranja četvrte slika

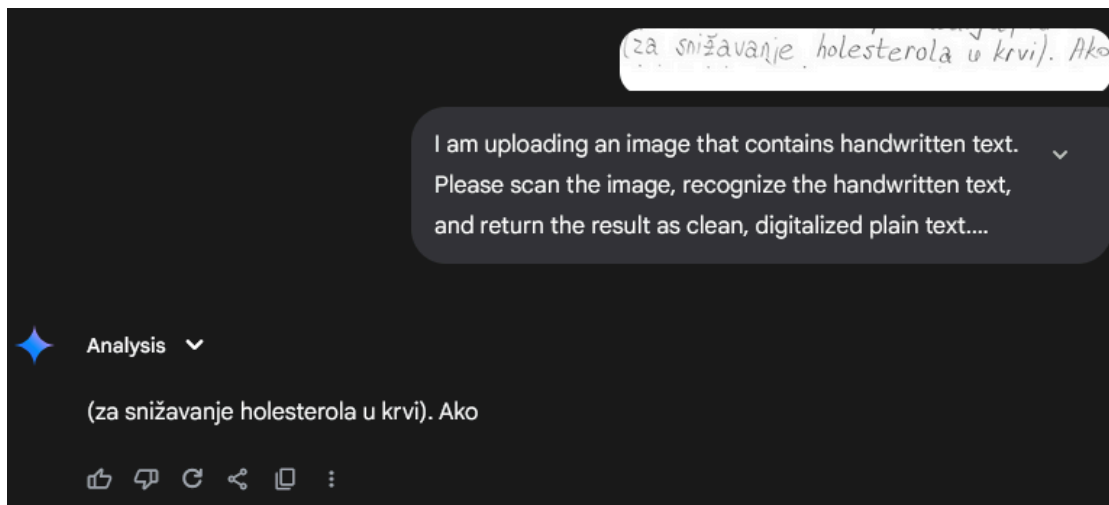
Rezultati skeniranja modelom Claude Sonnet 4 daju određene nepravilnosti u prepoznavanju teksta. Model izostavlja male zagrade u tekstu primjera 1, veliko slovo I mijenja malim, te potpuno pogrešno prepoznaje riječ “oblik” umjesto ispravnog “oblast” u primjeru 4, međutim, interesantno je primijetiti da je došlo do izmjene roda atributa “koji” kako bi isti ostao u gramatičkoj kongruenciji s pogrešno prepoznatom imenicom “oblik”. Ova izmjena pokazuje izvjesno poznavanje gramatike (tačnije sintakse i morfologije) B/H/S jezika, ali ne i generalno dobre sposobnosti kontekstualizacije teksta, odnosno razumijevanja semantike. Ponavljanje istog upita, kao i slanje novog sa zahtjevom za korekcijom grešaka, nije dao drugačije rezultate.



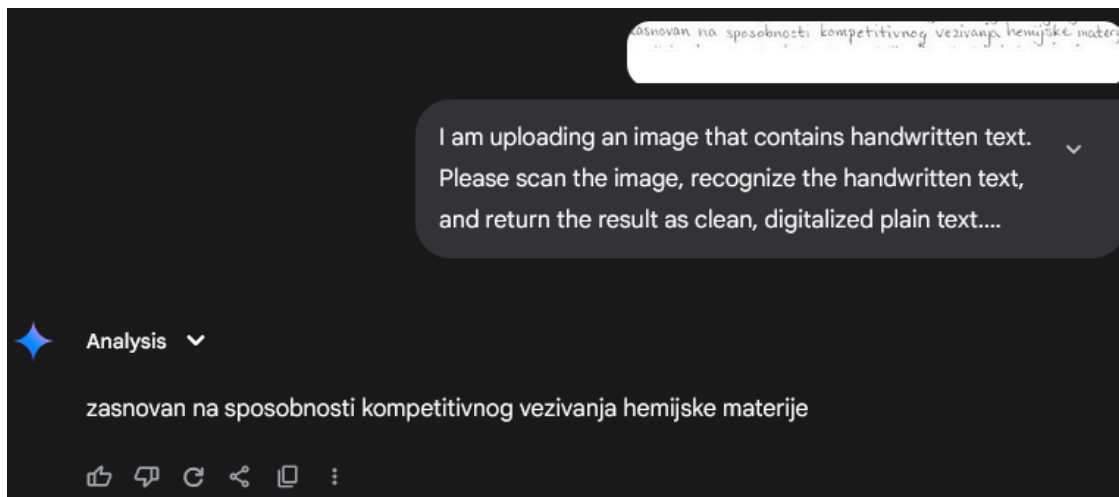
Slika 46 - Upit za korekciju grešaka daje isti rezultat

### 5.6.3 Google Gemini

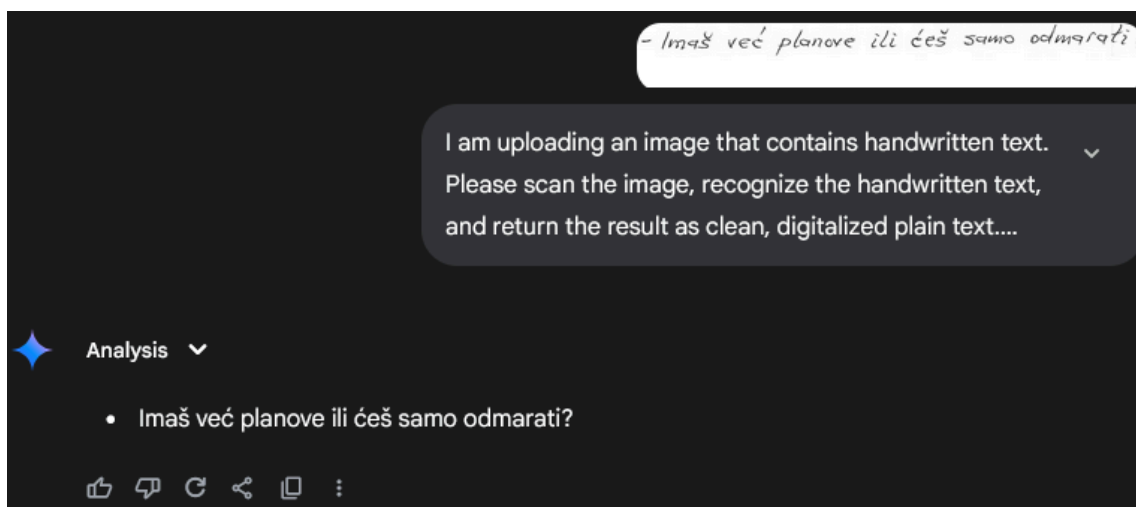
Korištena verzija modela Google Gemini je 2.5 Flash. Rezultati skeniranja su dati u slikama 47 - 50.



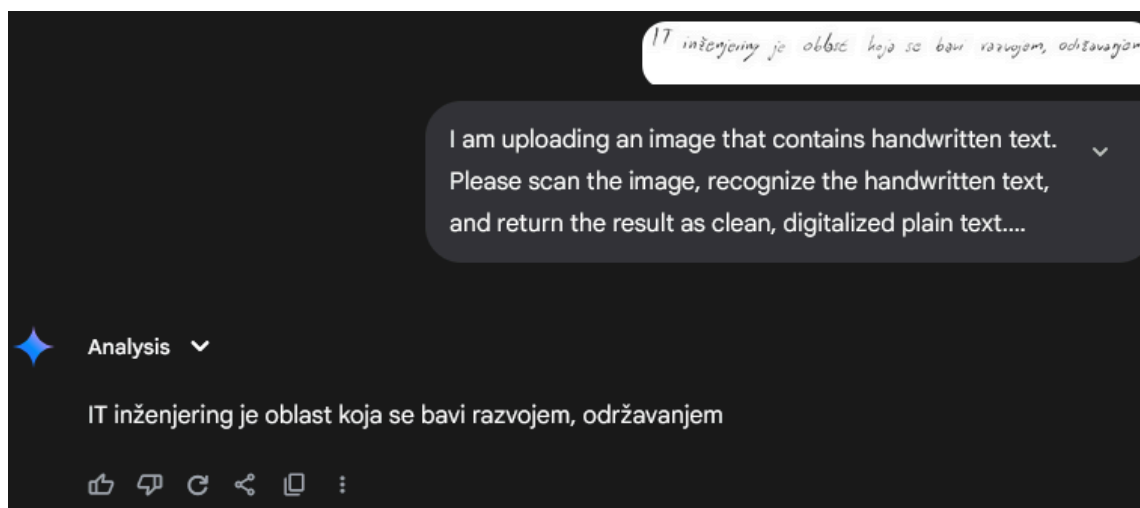
Slika 47 - Rezultati Google Gemini prilikom skeniranja prve slike



Slika 48 - Rezultati Google Gemini prilikom skeniranja druge slike



Slika 49 - Rezultati Google Gemini prilikom skeniranja treće slike



Slika 50 - Rezultati Google Gemini prilikom skeniranja četvrte slike

Rezultati skeniranja Google Gemini-jem su također u potpunosti ispravni. Specifičan rukopis kod nekih slova nije uticao na sposobnost ispravnog prepoznavanja riječi.

#### 5.6.4 Osvrt na rezultate

U zaključku, ustanovljeno je da su ChatGPT i Google Gemini 100% uspješni u prepoznavanju rukom pisanog teksta i na primjerima s nepreciznim rukopisom, gdje određena slova nalikuju jedna na druga. ClaudeAI, s druge strane, ima određenih generalnih poteškoća pri HTR zadacima, uključujući slabo razumijevanje konteksta i/ili ignorisanje cijelih znakova. Ovo se može

pripisati činjenici da je vjerovatno minorna količina podataka na B/H/S jezicima na kojima je treniran ovaj, kao i većina drugih LLM-ova. Smanjeno izlaganje ovim jezicima može uticati na sposobnosti HTR-a, kao i generalnog razumijevanja jezika u poređenju sa dominantnijim svjetskim jezicima kao što su engleski, španski, francuski, njemački ili kineski.



## 6. Zaključak

Cilj ovog rada bio je trenirati neuralnu mrežu - model koji će uspješno izvršavati HTR zadatke rukopisom štampanim slovima, na bosanskom jeziku.

Najprije se pristupilo prikupljanju i preprocesiranju podataka. Uspješno je kreiran *dataset* na kojem će se model trenirati, uključujući tri vrste podataka: pojedinačne riječi, kao i dva seta linija teksta, kreirana od strane dvije različite funkcije za ekstrakciju. Zatim su podaci pripremljeni za treniranje, učitani u okruženje i pristupljeno je treningu modela. Nakon što je treniranje završeno, model je bio spreman za korištenje, te je na testnim podacima odrađeno metričko ispitivanje modela.

Rezultati koje je model pokazao su zadovoljavajući, naročito s obzirom na veličinu *dataseta* kao i ograničenost računarskih resursa prilikom treniranja. Metričko testiranje dalo je respektivno vrijednosti CER, WER i SER metrika od 3.12%, 9.3% i 9.3% za set podataka koji sadrži samo riječi, 5.77%, 16.95% i 58.67% za set podataka koji sadrži linije teksta generisane izvornom funkcijom, kao i 5.5%, 19.53% i 61.03% za set podataka koji sadrži linije teksta generisane modificiranom funkcijom. Modificirana funkcija dala je nešto lošije rezultate WER i SER metrika, s obzirom na to da su greške koje je pravila izvorna funkcija išle u prilog modelu kada je u pitanju prepoznavanje, zbog prosječno kraćih generisanih sekvenci.

Za poboljšanje rezultata, moglo se izmijeniti nekoliko značajki pri izvedbi projekta. Naime, HTR modelima u generalnom slučaju preciznosti i efikasnosti u prepoznavanju doprinosi veličina i kvalitet seta podataka na kojem se isti treniraju. Tako bi i u ovom slučaju značajno doprinijela veća količina podataka za treniranje, ali noseći naravno sa sobom i veće zahtjeve za računarskim resursima. Upravo to je drugi aspekt koji se mogao poboljšati. Lokalno treniranje na dovoljno snažnom hardveru, umjesto korištenja besplatnih online servisa, moglo je uveliko ubrzati proces treniranja modela, što bi osiguralo više vremena za modifikacije modela, kao i fino podešavanje određenih značajki istog, a što bi doprinijelo preciznosti i efikasnosti.

Kada su u pitanju sposobnosti postojećih LLM-ova, mnogi su demonstrirali izniman uspjeh u jednostavnim ali i složenijim HTR zadacima na bosanskom jeziku. Međutim, ovim modelima za bolju efikasnost i kvalitetnije HTR

rezultate nedostaje veća količina trening podataka na bosanskom jeziku. Kako je većina trening podataka za moderne LLM-ove na engleskom jeziku, HTR zadaci na engleskom jeziku najbolje i funkcioniraju. Stoga postoji realističan prostor za poboljšanje ovih modela, s obzirom na to da oni koriste dosta dublje i kompleksnije arhitekture za sve zadatke koje izvršavaju, pa tako i za HTR.

Kreiranje efikasnog i široko primjenjivog HTR modela ostaje izazovan zadatak, prvenstveno s obzirom na ogromne raznovrsnosti ljudskog rukopisa. Najčešće setovi podataka za treniranje ovakvih modela trebaju stoga biti iznimno veliki, te time i zahtijevaju velike računarske resurse, kao i dugo vrijeme treniranja. Najefikasnijim neuralnim mrežama za HTR zadatke su se pokazale one hibridnih arhitektura, kombinujući najčešće sposobnosti CNN i RNN, sa dodatnim modifikacijama, kako bi se postigla što bolja preciznost i efikasnost modela. Naučnici i dalje razvijaju veliki broj hibridnih modela, tražeći najbolji balans između kompleksnosti, zahtjevnosti treniranja i kvalitetnih rezultata.

## 7. Popis literature

- [1] C. Garrido-Munoz, A. Rios-Vila, and J. Calvo-Zaragoza, "Handwritten Text Recognition: A Survey," *arXiv.org*, 2025. <https://arxiv.org/abs/2502.08417> (pristupljeno Jun. 25, 2025)
- [2] R. Li, "A review of neural networks in handwritten character recognition," *Applied and Computational Engineering*, vol. 92, no. 1, pp. 169–174, Oct. 2024, doi: <https://doi.org/10.54254/2755-2721/92/20241736> (pristupljeno Jun. 25, 2025)
- [3] W. AlKendi, F. Gechter, L. Heyberger, and C. Guyeux, "Advancements and Challenges in Handwritten Text Recognition: A Comprehensive Survey," *Journal of Imaging*, vol. 10, no. 1, p. 18, Jan. 2024, doi: <https://doi.org/10.3390/jimaging10010018> (pristupljeno Jun. 25, 2025)
- [4] G. Crosilla, L. Klic, and G. Colavizza, "Benchmarking Large Language Models for Handwritten Text Recognition," *arXiv.org*, 2025. <https://arxiv.org/abs/2503.15195>
- [5] HumanSignal, "GitHub - HumanSignal/labellmg: Labellmg is now part of the Label Studio community. The popular image annotation tool created by Tzutalin is no longer actively being developed, but you can check out Label Studio, the open source data labeling tool for images, text, hypertext, audio, video and time-series data.," GitHub, Dec. 03, 2018. <https://github.com/HumanSignal/labellmg?tab=readme-ov-file#steps-pascalvoc> (pristupljeno Maj, 2025).
- [6] arthurflor23, "GitHub - arthurflor23/handwritten-text-recognition: Handwritten Text Recognition using TensorFlow," GitHub, 2019. <https://github.com/arthurflor23/handwritten-text-recognition/blob/master/src/tutorial.ipynb> (pristupljeno Jun. 07, 2025).
- [7] abegovac2, "GitHub - abegovac2/masters-theses-HTR," GitHub, 2024. <https://github.com/abegovac2/masters-theses-HTR> (pristupljeno Jun. 07, 2025).
- [8] A. Begovac, "Ekstrakcija teksta metodama dubokog učenja iz zapisnika minsko-eksplozivnih prepreka," Elektrotehnički fakultet Univerziteta u Sarajevu, 2025.

[9] Amanatullah, "Vanishing Gradient Problem in Deep Learning: Understanding, Intuition, and Solutions," *Medium*, Jun. 12, 2023. <https://medium.com/@amanatulla1606/vanishing-gradient-problem-in-deep-learning-understanding-intuition-and-solutions-da90ef4ecb54>

[10] ijozic1, "GitHub - ijozic1/Handwritten-text-recognition: Project related to the Artificial Intelligence course at the Faculty of Electrical Engineering, University of Sarajevo," GitHub, 2025. <https://github.com/ijozic1/Handwritten-text-recognition> (pristupljeno Jun. 27, 2025).

## 8. Popis slika

Slika 1 - Rukom pisani tekst [stranica 9]

Slika 2 - Označeni tekst [stranica 9]

Slika 3 - Kreirani folderi tokom izrade dataset-a [stranica 10]

Slika 4 - Funkcija za izvlačenje pojedinačnih riječi na osnovu .xml [stranica 11]

Slika 5 - Pomoćna funkcija koja iterira kroz pojedinačni .xml [stranica 12]

Slika 6 - Definisanje lokacije na koju se sprema word\_labels.txt i pozivanje metode za izdvajanje pojedinačnih riječi i linija teksta [stranica 12]

Slika 7 - Prikaz sadržaja u datoteci word\_labels.txt [stranica 13]

Slika 8 - Metoda koja vrši izdvajanje slika pojedinačnih riječi [stranica 14]

Slika 9 - Pomoćna metoda koja vrši izrezivanje slike na osnovu gornjeg lijevog i donjeg desnog ugla [stranica 14]

Slika 10 - Primjer pojedinačne slike riječi koju dobijemo na kraju procesa izdvajanja [stranica 15]

Slika 11 - Metoda za izvlačenje linija teksta [stranica 16]

Slika 12 - Sadržaj line\_labels.txt [stranica 17]

Slika 13 - Metoda za izrezivanje linija teksta [stranica 17]

Slika 14.1 - Primjer dobro izrezane linije teksta [stranica 18]

Slika 14.2 - Primjer u kojem su znakovi navoda prepoznati kao posebna linija teksta [stranica 18]

Slika 14.3 - Primjer u kojem je obuhvaćeno više linija teksta zbog preklapanja [stranica 18]

Slika 15 - Modificirana funkcija za izvlačenje linija iz teksta [stranica 20]

Slika 16 - Pomoćna funkcija za dodavanje razmaka u linijama teksta [stranica 21]

Slika 17 - Sadržaj line\_labels\_m.txt [stranica 21]

Slika 18.1 - Navodnici su sada dio iste linije kao i tekst kojem pripadaju [stranica 21]

Slika 18.2 - Linije teksta su jasnije odvojene [stranica 22]

Slika 19 - Metoda koja vrši podjelu *dataset*-a na trening i testni skup podataka [stranica 23]

Slika 20 - Izgled foldera sa *dataset*-om nakon preprocesiranja podataka [stranica 24]

Slika 21 - Izgled foldera jednog od *dataset*-ova (labels\_l\_m) [stranica 24]

Slika 22 - Podešavanje Google Drive okoline i TensorFlow-a; mount particije [stranica 29]

Slika 23 - Podešavanje Python klase [stranica 30]

Slika 24 - Korištenje DataGenerator klase za učitavanje podataka [stranica 30]

Slika 25 - Učitavanje HTR modela [stranica 31]  
 Slika 26 - Treniranje HTR modela [stranica 33]  
 Slika 27 - Kod koji vrši iscrtavanje grafika za trening i validacijski gubitak [stranica 34]  
 Slika 28 - Treniranje modela na pojedinačnim riječima [stranica 34]  
 Slika 29 - Detalji treniranja modela na linijama teksta [stranica 35]  
 Slika 30 - Detalji treniranja modela na modificiranim linijama teksta [stranica 36]  
 Slika 31 - Kod koji pokreće testiranje modela [stranica 38]  
 Slika 32 - Prikaz jedne od 20 testnih slika sa rezultatima na *dataset*-u sa pojedinačnim riječima [stranica 38]  
 Slika 33 - Kod za računanje metrika i njihov ispis [stranica 39]  
 Slika 34 - Pogrešno prepoznata otvorena mala zagrada kao slovo l [stranica 45]  
 Slika 35 - Preneseno slovo j iz prethodnog reda pogrešno prepoznato kao kvačica na š [stranica 45]  
 Slika 36 - Slovo v pogrešno prepoznato kao r; m kao spojeno v i n [stranica 45]  
 Slika 37 - Slovo l prepoznato kao broj 1; malo slovo t kao ć; slovo r kao i [stranica 45]  
 Slika 38 - Rezultat ChatGPT-ja prilikom skeniranja prve slike [stranica 47]  
 Slika 39 - Rezultat ChatGPT-ja prilikom skeniranja druge slike [stranica 47]  
 Slika 40 - Rezultat ChatGPT-ja prilikom skeniranja treće slike [stranica 48]  
 Slika 41 - Rezultat ChatGPT-ja prilikom skeniranja četvrte slike [stranica 48]  
 Slika 42 - Rezultat ClaudeAI prilikom skeniranja prve slike [stranica 49]  
 Slika 43 - Rezultat ClaudeAI prilikom skeniranja druge slike [stranica 50]  
 Slika 44 - Rezultat ClaudeAI prilikom skeniranja treće slike [stranica 50]  
 Slika 45 - Rezultat ClaudeAI prilikom skeniranja četvrte slike [stranica 51]  
 Slika 46 - Upit za korekciju grešaka daje isti rezultat [stranica 51]  
 Slika 47 - Rezultati Google Gemini prilikom skeniranja prve slike [stranica 52]  
 Slika 48 - Rezultati Google Gemini prilikom skeniranja druge slike [stranica 52]  
 Slika 49 - Rezultati Google Gemini prilikom skeniranja treće slike [stranica 53]  
 Slika 50 - Rezultati Google Gemini prilikom skeniranja četvrte slike [stranica 53]

## 9. Popis grafika

Grafik 1 - Gubitak po epohama za treniranje dataset-a u kojem su pojedinačne riječi [stranica 35]  
 Grafik 2 - Gubitak po epohama za treniranje dataset-a u kojem su linije teksta [stranica 36]  
 Grafik 3 - Gubitak po epohama za treniranje dataset-a u kojem su modificirane linije teksta [stranica 37]  
 Grafik 4 - Prikaz metrika dobijenih nakon testiranja modela [stranica 42]

## 10. Popis tabela

Tabela 1 - Slojevi neuralne mreže [stranica 28]