UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Jelena Mrdak

SEGMENTACIJA I PREPOZNAVANJE TEKSTA U RUKOM PISANIM BELEŠKAMA

master rad

Beograd, 2021.

Mentor:

dr Ivan Čukić, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Saša Malkov, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Aleksandar Kartelj, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Naslov master rada: Segmentacija i prepoznavanje teksta u rukom pisanim beleškama

Rezime: Rad se bavi prepoznavanjem rukom pisanog teksta. Ulaz programa je slika na kojoj se nalazi tekst bez označenog regiona gde se karakteri mogu naći. Izlaz programa je niz karaktera koji treba što preciznije da odgovara tekstu sa slike.

Prvi deo rada je fokusiran na prilagođavanje i implementaciju ideja za segmentaciju linija u rukom pisanim tekstovima, razvijanih za persijske tekstove [1]. Drugi deo rada se bavi prepoznavanjem pojedinačnih slova i bigrama, dok je treći deo posvećen razvijanju programa koji prepoznaje tekst sa slike, tako što izdvaja linije, zatim komponente, određuje razmake, prepoznaje komponente i, na kraju, ispravlja greške.

Ključne reči: mašinsko učenje, neuronske mreže, segmentacija, slovo, bigram

Sadržaj

Glava 1

Uvod

U ovom radu bavićemo se prepoznavanjem teksta sa slike. Kako je u pitanju dosta kompleksan problem, uvešćemo izvesna ograničenja. Zahtevaćemo da tekst bude na engleskom jeziku, sačinjen isključivo od malih štampanih slova, bez ostalih karaktera kao što su tačka, zapeta, apostrof i slično. Takođe, zbog limitiranog skupa podataka, tekst ćemo pisati u programu Xournal++[1] i ograničićemo se na prepoznavanje određenog rukopisa, tj. teksta koji piše jedna osoba. Dodatno, koristićemo rečnik engleskih reči i biblioteku za ispravljanje slovnih grešaka u engleskom tekstu, kako bismo poboljšali rad programa.

Počećemo sa segmentacijom linija koristeći ideje opisane u radu [1] koji se bavi rešavanjem ovog problema u slučaju tekstova na persijskom jeziku. Opisaćemo algoritam predstavljen u pomenutom radu, kao i modifikacije koje smo napravili kako bismo ga prilagodili našim potrebama. Takođe, ukazaćemo na probleme i moguća unapređenja kroz nekoliko primera.

Nakon toga, naredno poglavlje posvetićemo prepoznavanju slova. Prikazaćemo naše skupove podataka za treniranje, validaciju i testiranje. Zatim ćemo predstaviti modele koji se mogu koristiti za prepoznavanje pojedinačnih slova i dva spojena slova. Bavićemo se analizom tih modela primenjujući tehnike za evaluaciju koje se koriste u problemima klasifikacije. Dodatno, prikazaćemo skup generisanih slova baziran na našem skupu rukom pisanih slova i testiraćemo model na njima. Pored toga, navešćemo i nedostatke u kojima se može naći razlog za uvođenje ograničenja za prepoznavanje samo određenog rukopisa.

U poslednjem poglavlju iskoristićemo rezultate iz prethodna dva dela i napra-

[1]Xournal++ je program za prikupljanje beleški sa podrškom za PDF anotiranje; `https://github.com/xournalpp/xournalpp`.

vićemo program za prepoznavanje teksta. Opisaćemo postupak za izdvajanje komponenata, određivanje razmaka u liniji i prepoznavanje izdvojenih komponenata uz dodatne optimizacije. Takođe ćemo prikazati i rezultate rada biblioteke za ispravljanje slovnih grešaka. Na kraju, demonstriraćemo rad programa kroz nekoliko primera uz osvrt na neke probleme i dalja unapređenja.

Glava 2

Segmentacija linija

Jedan od prvih koraka pri rešavanju problema prepoznavanja teksta sa slike jeste izdvajanje linija teksta. Pod linijom teksta podrazumevamo jedan red tog teksta, koji ne mora nužno biti napisan horizontalno, već može biti napisan i ukoso. U ovom delu prikazaćemo jedno uopšteno rešenje, bez fokusa na slike kojima ćemo se kasnije baviti, dok ćemo u poglavlju 4 prilagoditi algoritam za segmentaciju linija za naše potrebe.

Prvo ćemo prikazati algoritam za izdvajanje linija koji je opisan u radu [1]. Iako je rad baziran na persijskim tekstovima, mi ćemo algoritam predstaviti koristeći razne jezike, uključujući i programski. Naravno, sam smisao teksta je za ovaj problem nebitan. Važan je samo izgled karaktera i stil kojim je tekst napisan (pisana, štampana slova, i slično).

Opisaćemo i nekoliko izmena koje smo napravili u odnosu na originalnu verziju algoritma i zatim ćemo demonstrirati rad tako modifikovanog programa. Kroz primere videćemo određene probleme koji ostavljaju prostor za dalja unapređenja.

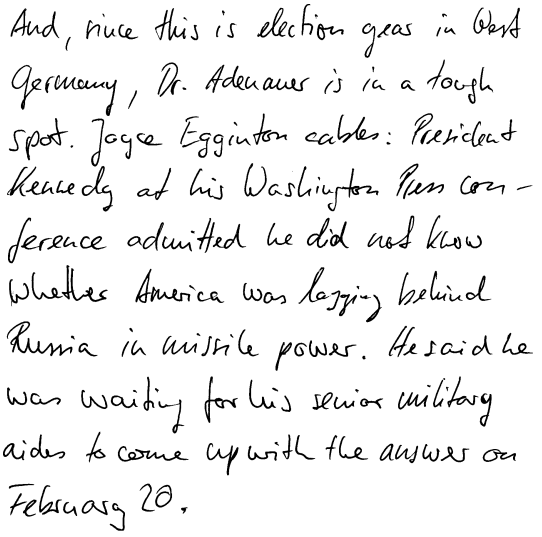
Implementacija algoritma koji će biti opisan u ovom delu se može naći na GitHub repozitorijumu [3].

2.1 Algoritam za segmentaciju linija

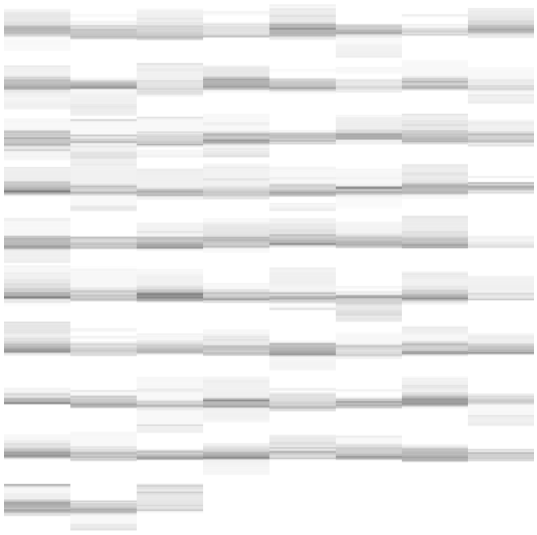
Bojenje segmenata i detektovanje praznina između linija

Na početku je potrebno podeliti sliku na određen broj kolona približno jednakih širina. Te kolone ćemo u daljem tekstu nazivati trakama (eng. *strips*). Zatim, svaka traka se boji pojedinačno, tako što se svaka vrsta u traci boji prosečnom vrednošću

piksela koji se nalaze u toj vrsti, kao na Slici 2.2.



Slika 2.1: Slika koju treba izdeliti na linije



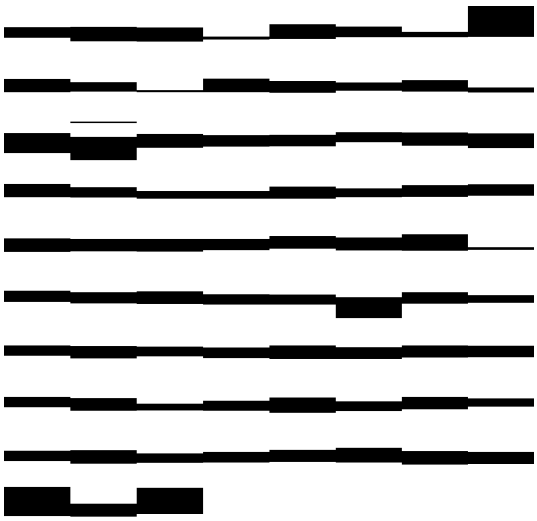
Slika 2.2: Osam traka. Svaka vrsta u traci je popunjena prosečnom vrednošću piksela u toj vrsti.

Nakon podele na trake i inicijalnog bojenja, na svaku traku posebno se primenjuje Otsu[1] tehnika (Slika 2.3) kako bi se dobili samo crni i beli pikseli. Ovako obojeni pikseli formiraju crne i bele blokove, tj. pravougaonike u trakama. Crni pravougaonici odgovaraju tekstu na ulaznoj slici, dok beli predstavljaju pozadinu. Da bismo izbegli potencijalne probleme u narednim koracima, bele pravougaonike čija je visina manja od određenog praga bojimo u crne (Slika 2.4) i obrnuto (Slika 2.5).

[1]Otsu tehnika se koristi pri obradi slika kako bi se pikseli podelili u dve klase. U našem slučaju, jednu klasu čine crni pikseli koji predstavljaju regione teksta, dok drugu čine beli pikseli koji predstavljaju pozadinu.

Slika 2.3: Otsu's thresholding. Na svaku traku posebno je primenjen metod Otsu.

Slika 2.4: Popunjavanje belih pravougaonika.

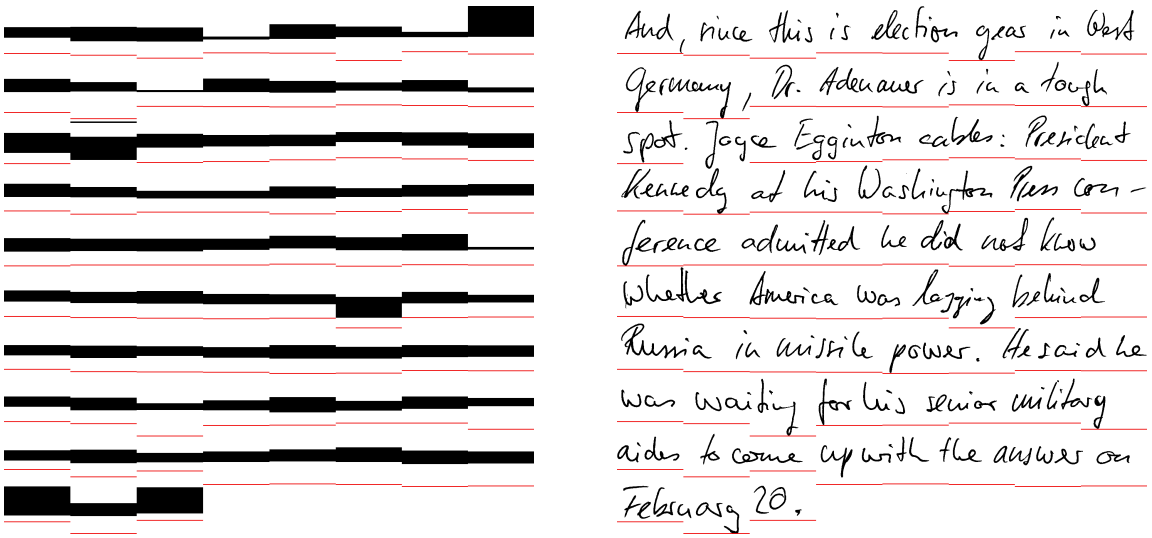


Slika 2.5: Brisanje crnih malih pravougaonika.

Crtanje linija vodilja na osnovu analize crno belih delova

U svakom belom pravougaoniku koji iznad sebe ima crni pravougaonik, pokušavamo nacrtati jednu liniju vodilju duž celog pravougaonika, odnosno trake. Uslov za crtanje linije jeste da ona prolazi samo kroz bele piksele na originalnoj slici. Jasno, to nije uvek moguće uraditi, tako da u nekim belim pravougaonicima nećemo imati liniju vodilju. Nekada je to poželjno, jer beli pravougaonik ne predstavlja uvek pozadinu, već se može nalaziti na mestu teksta (četvrti beli pravougaonik u drugoj

traci na Slici 2.6), pa i ne želimo da postoji linija na tom mestu, a nekada nam je potrebna linija na mestu tog pravougaonika, samo što ne možemo povući pravu liniju koja sadrži isključivo bele piksele (peti beli pravougaonik u drugoj traci na Slici 2.6). Linije vodilje treba povezati da bi se izvršila segmentacija.



Slika 2.6: Linije vodilje. Crvenom bojom označene su linije koje kasnije treba spojiti.

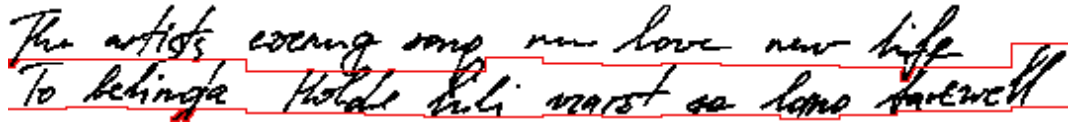
Rešavanje problema preklapajućih i dodirujućih komponenti

Kao što smo već napomenuli, linije vodilje ne moraju postojati u svakom belom pravougaoniku. Postoje dva razloga koja dovode do toga da linija vodilja ne postoji na mestu na kome razdvajajuća putanja[2] treba da prođe. Prvi je slučaj preklapajućih komponenti, što podrazumeva da se vertikalne projekcije dve komponente iz susednih linija seku ili dodiruju, a drugi je slučaj dodirujućih komponenti, koji nastaje kada se komponente iz različitih linija dodiruju (Slika 2.7). U prvom slučaju, komponente treba zaobići, dok u drugom ih je potrebno preseći, kao što je to prikazano na Slici 2.8. Ovde je glavni problem prepoznati koji je slučaj u pitanju, odnosno odlučiti da li zaobići komponentu ili je seći. Ako napravimo pogrešnu odluku i isečemo komponentu u slučaju kada bi je trebalo zaobići, deo jednog slova će se naći u pogrešnoj liniji. Sa druge strane, ako odlučimo da komponentu obiđemo u slučaju kada bi je trebalo seći, onda ćemo „preći" u pogrešnu liniju, tj. komponentu iz naredne linije pripojićemo trenutnoj. Ovde posebno dolazi do izražaja problem

[2]Niz povezanih piksela koji idu od levog do desnog kraja slike i koji ograničavaju jedan red teksta odozdo. Razdvajajuća putanja koja ograničava jedan red često nije jedinstvena.

raznolikosti slika koje program može da primi na ulazu, jer je teško optimizovati ovaj deo algoritma tako da pokrije različite situacije koje dolaze u obzir - gusto pisani tekst sa mnogo dodirujućih i preklapajućih komponenti, tekst pisan krupnim fontom sa većim razmacima između slova i dugačkim kukicama koje je potrebno zaobići, itd.

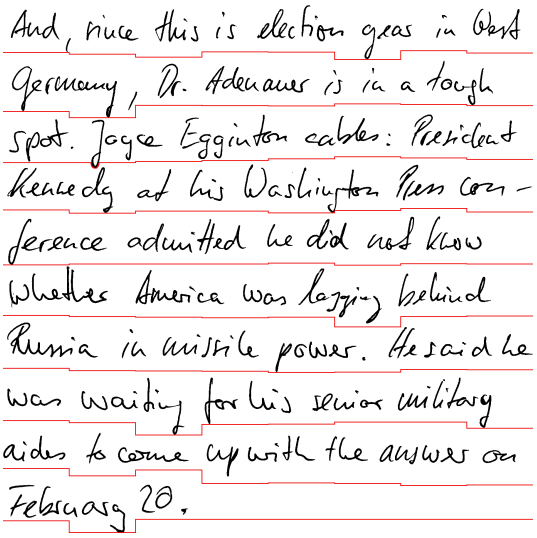
Slika 2.7: Crvenom bojom obojene su preklapajuće komponente, a zelenom dodirujuće.



Slika 2.8: Zaobilaženje i sečenje komponenti

Odabir glavne trake i pravljenje razdvajajućih putanja

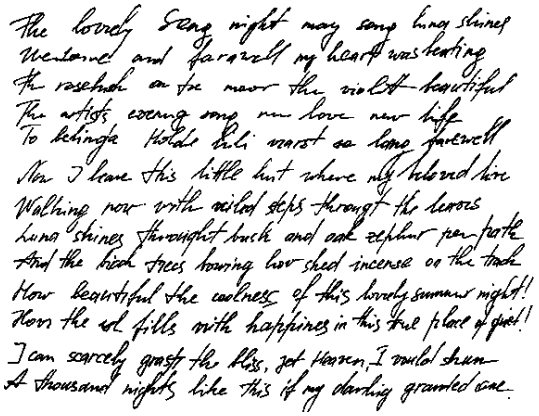
Glavna traka (eng. *Core Strip*) jeste traka sa najvećim brojem linija vodilja (u slučaju da ima više takvih traka, uzima se krajnja leva). Na Slici 2.6, glavna traka je prva. Svaku liniju vodilju iz ove trake potrebno je produžiti do leve i desne ivice slike, uzimajući u obzir preklapajuće i dodirujuće komponente, kako bi se dobile putanje koje razdvajaju linije teksta. Da bi se ovo postiglo, potrebno je što bolje povezati linije vodilje. Trenutna putanja se produžava tako što se pronalazi najbliža linija vodilja iz susedne trake i, ako je ona dovoljno blizu (posmatrajući vertikalnu udaljenost), one se spajaju vertikalnom linijom, pod uslovom da ta linija ne seče tekst. U suprotnom, pokušavamo da dodamo novu pravu liniju u narednoj traci (koja ne sme ići preko teksta i koja je blizu trenutne putanje) i onda da je spojimo vertikalnom linijom sa putanjom kao u prethodnom slučaju. Ako, pak, ni to nije moguće, onda je u pitanju neka prepreka i potrebno ju je zaobići ili iseći. Dakle, putanja se postepeno produžava, spajajući trenutnu putanju sa linijom vodiljom, ili sa novom linijom, ili produžavajući trenutnu putanju, zaobilazeći pritom prepreke.



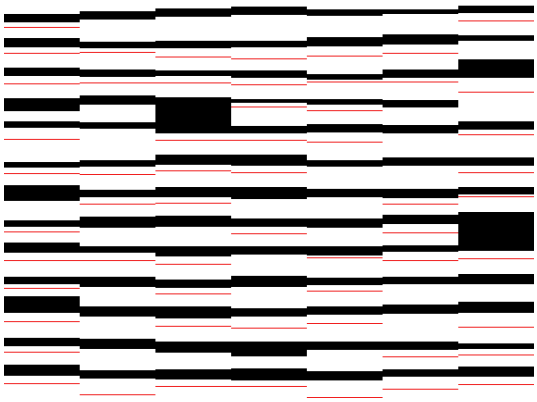
Slika 2.9: Slika izdeljena na linije.

Obrada nekorišćenih linija vodilja

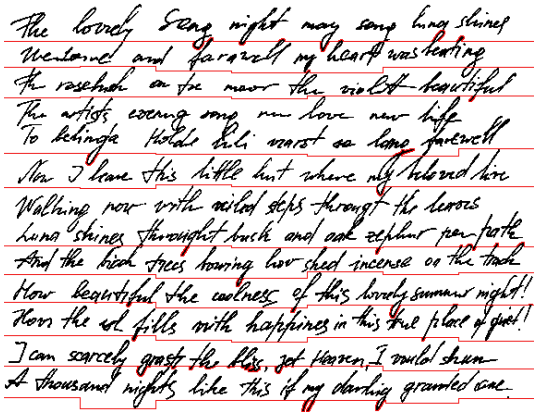
Može se desiti da neke linije vodilje ne budu iskorišćene i njih je potrebno razmotriti. Naime, glavna traka nam nije uvek dovoljna da bismo razdvojili sve linije teksta, jer se može desiti da neka linija nema liniju vodilju u glavnoj traci. Jedan razlog jeste nemogućnost da u traci povučemo liniju vodilju koja mora proći samo kroz bele piksele (Slika 2.11), a drugi je taj što tekst ne mora biti ravnomerno raspoređen, pa glavna traka uopšte ne mora sadržati sve linije (Slika 2.15). Shodno tome, potrebno je primeniti isti postupak pravljenja putanji počevši od ovakvih neiskorišćenih linija.



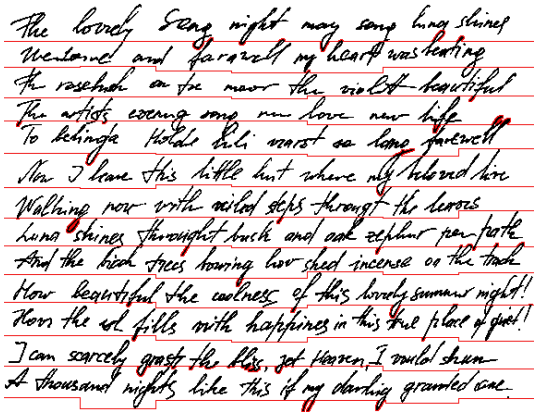
Slika 2.10: Ulazna slika



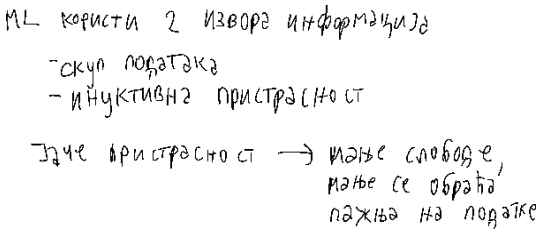
Slika 2.11: Linije vodilje. Glavna traka je prva jer ima najviše linija vodilja. Četvrta linija nije detektovana jer nemamo liniju vodilju za nju u glavnoj traci.



Slika 2.12: Rezultat koji koristi samo puteve iz glavne trake. Dve linije nisu detektovane jer glavna traka nema linije vodilje u njima.

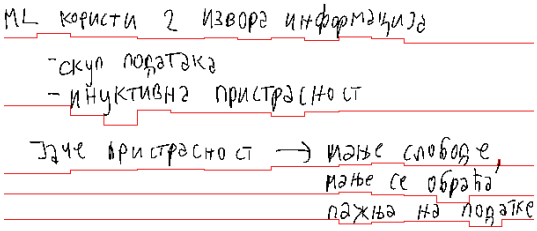


Slika 2.13: Rezultat koji koristi linije vodilje i van glavne trake.

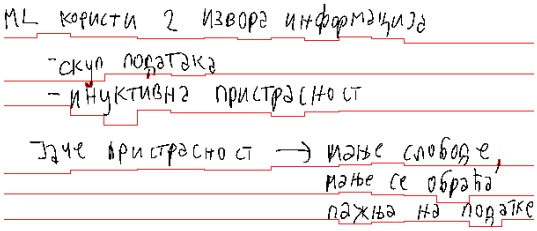


Slika 2.14: Ulazna slika

Slika 2.15: Linije vodilje. Glavna traka je deseta jer ima najviše linija vodilja. Ona ne sadrži tekst iz druge linije, te drugu liniju ne možemo detektovati samo pomoću nje.



Slika 2.16: Rezultat koji koristi samo puteve iz glavne trake. Dve linije nisu detektovane jer glavna traka nema linije vodilje u njima.



Slika 2.17: Rezultat koji koristi linije vodilje i van glavne trake.

Izmene

Napravili smo nekoliko izmena u odnosu na rad [1].

- Promenili smo pragove (eng. *thresholds*). Koristili smo srednju vrednost visina belih pravougaonika, umesto mode visina koja je korišćena u radu, kao jedan od uslova pri donošenju odluka vezanih za bojenje pravougaonika, zaobilaženje i sečenje komponenti i slično.

- Modifikovali smo definiciju iskorišćenih linija vodilja, tako da pravimo razliku između onih linija koje nisu iskorišćene, ali su razmatrane i onih koje nisu ni razmatrane. Prvi slučaj nastaje, recimo, kada je linija bila „daleko" od trenutne putanje pa je zato neiskorišćena, ali je uzeta u obzir kao najbliža toj putanji. Ovakve linije treba samo zanemariti tako da se one više ne mogu koristiti ni za produžavanje neke putanje ni za započinjanje nove. U slučaju neiskorišćenih linija koje nisu razmatrane, postupak pravljenja putanji iz njih nastavljamo, kao što je to i opisano u radu, s tim da te nove putanje dodajemo u rešenje samo ako ne seku postojeće putanje.

- Dodali smo još jedan potreban uslov da bismo trenutnu putanju proširili linijom vodiljom iz naredne trake. Pored uslova koji se odnosi na vertikalnu udaljenost, razmatramo i boje pravougaonika.

- U radu [1] se pogodna širina trake dobija u drugom prolazu. Najpre je početna širina vrednost koja zavisi od prosečne širine komponenti. Zatim se određuje razmak između linija posmatrajući visinu belih pravougaonika i na taj način se dobije pogodna širina. Umesto toga, opredelili smo se da koristimo širinu trake koja ne zavisi od razmaka linija. Širinu trake određujemo tako da postoje linije vodilje u bar 50% belih pravougaonika pre nego što započnemo postupak pravljenja putanji. Polazimo od veće širine[3] koju smanjujemo ako taj uslov nije zadovoljen. Ovaj deo algoritma je jedan od ključnih. Problem je što ne možemo lako reći šta je dobra širina, jer slike mogu da budu dosta različite. Ako je širina previše velika, nećemo imati dovoljno linija od kojih započinjemo razdvajajuće putanje i lako se može desiti da pogrešimo. Ako je širina previše

[3]U ovom poglavlju, početna širina trake je četiri puta prosečna širina komponenti, jer se ona pokazala kao „najbolja", uzimajući u obzir različite slike, dok ćemo kasnije promeniti početnu širinu tako da bude duplo veća, kako bismo je prilagodili slikama koje ćemo mi koristiti (slike visoke rezolucije na kojoj se nalazi tekst sa štampanim slovima).

mala, možemo imati „lažne" linije (npr. linija vodilja se nalazi na sredini slova „e") i onda ćemo napraviti pogrešnu putanju koja će ~~preseći tekst ovako~~.

- Postavili smo uslov da sve trake moraju biti iste širine do na jedan piksel. U radu [1] se samo veličina poslednje trake može razlikovati i to za veći broj piksela. Kada je poslednja traka značajno manja od ostalih, može se desiti da dobijemo „lažne" linije vodilje, jer radimo threshold (Otsu) po traci, koje kasnije često dovode do pogrešne segmentacije.

Kôd

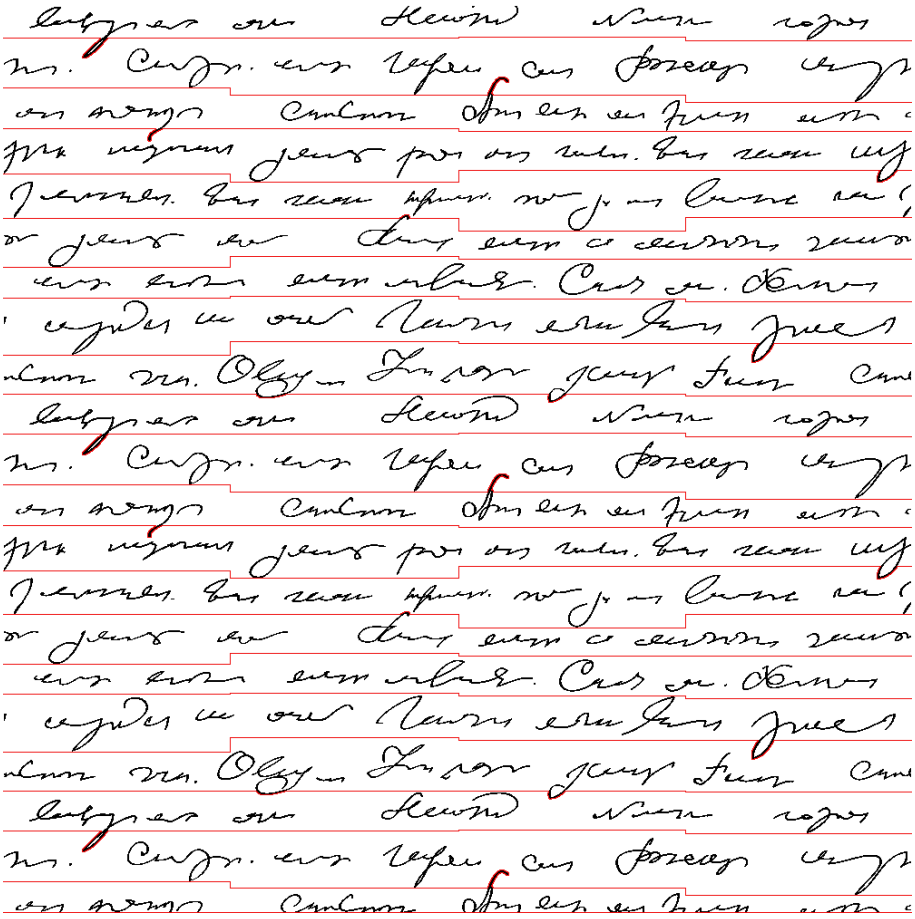
U ovom delu dajemo skicu kôda koji se odnosi na pravljenje razdvajajućih putanji. Napominjemo da je u pitanju pseudokod koji služi za rekapitulaciju, a ne za implementaciju.

```
POVEZI_LINIJE():
    // niz putanji koje predstavljaju granice za linije na slici
    putanje ← []

    // svaku liniju vodilju u glavnoj traci produzi do levog i
       desnog kraja slike
    za svaku liniju u glavnoj traci:
        // oznaci liniju kao iskoriscenu da je ne bismo opet
           kasnije koristili za neku putanju
        linija.iskoriscena ← True

        kandidat_putanja ← [linija]
        // produzi liniju do desnog kraja slike
        produzi(linija.desni_kraj, desno)
        // produzi liniju do levog kraja slike
        produzi(linija.levi_kraj, levo)

        // dodaj kandidat putanju u sve putanje
        putanje ← putanje + kandidat_putanja

    // probaj da produzis svaku neiskoriscenu liniju vodilju
    za svaku traku:
        za svaku neiskoriscenu liniju u traci:
            linija.iskoriscena ← True

            kandidat_putanja ← [linija]
            produzi(linija.desni_kraj, desno)
            produzi(linija.levi_kraj, levo)

            ako je presek(putanje, kandidat_putanja) = ∅:
                putanje ← putanje + kandidat_putanja

    vrati putanje
```

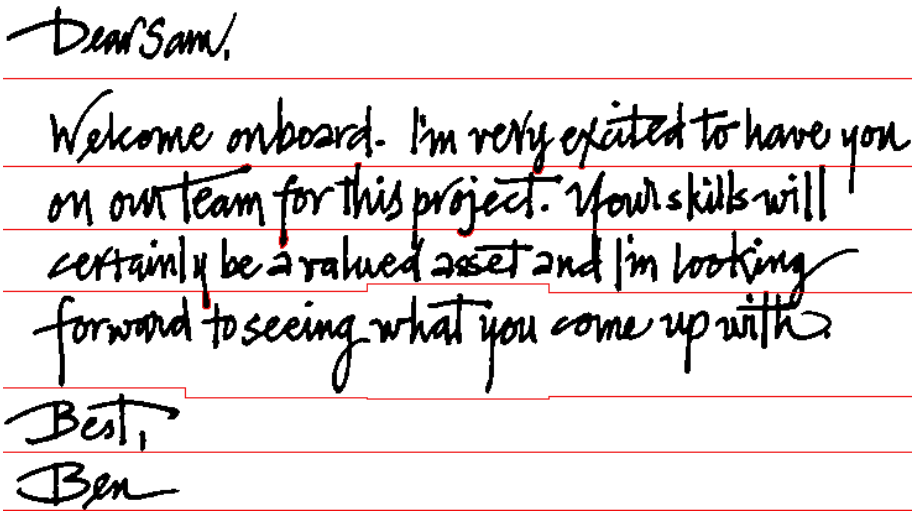
```
PRODUZI(piksel, smer):
    // piksel: poslednji piksel koji se nalazi u kandidat
       putanji koju treba da produzimo u datom smeru
    // smer: levo ili desno

    // pokusaj da produzis liniju koristeci liniju vodilju
    novi_piksel ← produzi_sa_linijom_vodiljom(piksel, smer)

    ako je novi_piksel = piksel:
        // nismo uspeli da produzimo put preko linije vodilje
        // pokusaj da napravis novu liniju
        novi_piksel ← produzi_sa_pravom_linijom(piksel, smer)

    ako je novi_piksel = piksel:
        // nismo uspeli da produzimo put preko nove prave linije
        // pokusaj da produzis liniju tako sto ces obici
           prepreku ili preci preko nje
        novi_piksel ← idi_okolo_ili_iseci(piksel, smer)

    ako nije novi_piksel = piksel:
        // napredovali smo, nastavi od novog poslednjeg piksela
           u kandidat putanji
        PRODUZI(novi_piksel, smer)
```

2.2 Demonstracija rada algoritma

U ovoj sekciji demonstriraćemo rad algoritma kroz nekoliko primera. Na Slici 2.18 prikazana je segmentacija linija u slučaju pisanih slova, gde su linije ravnomerno raspoređene, dok na Slici 2.19 možemo videti izlaz algoritma u slučaju neravnomernih linija.

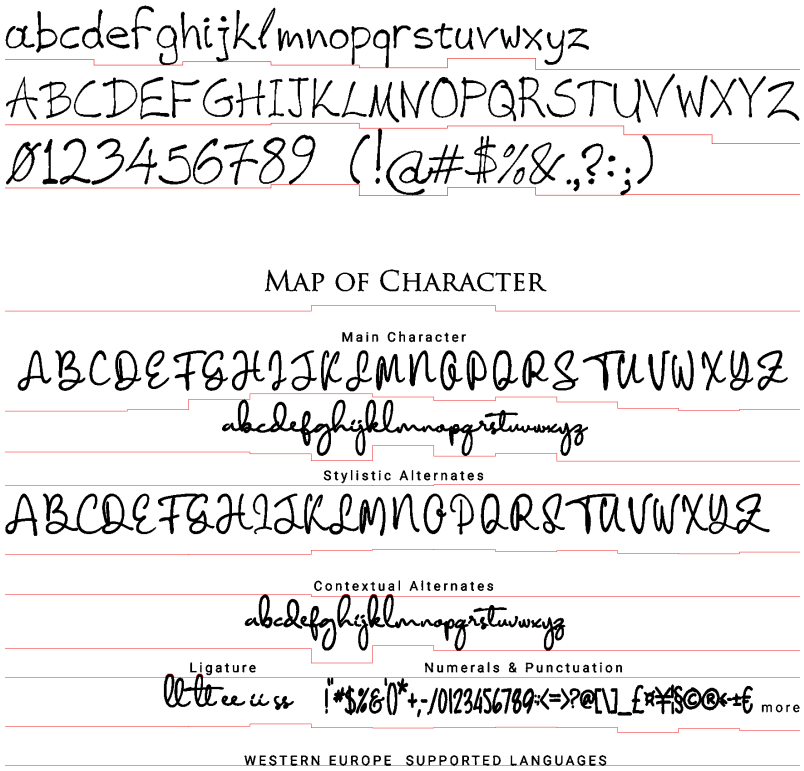


Slika 2.18: Segmentacija linija u slučaju teksta sa pisanim slovima i ravnomerno raspoređenim linijama



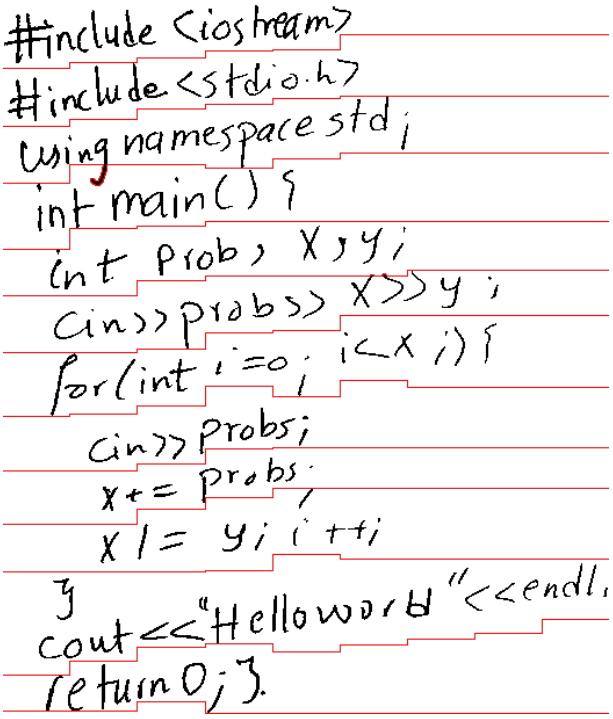
Slika 2.19: Segmentacija linija u slučaju teksta sa pisanim slovima i neravnomerno raspoređenim linijama

Takođe, zanimljivo je videti kako se algoritam ponaša i u nekim nestandarnim slučajevima (Slika 2.20).



Slika 2.20: Segmentacija linija u nestandarnim slučajevima

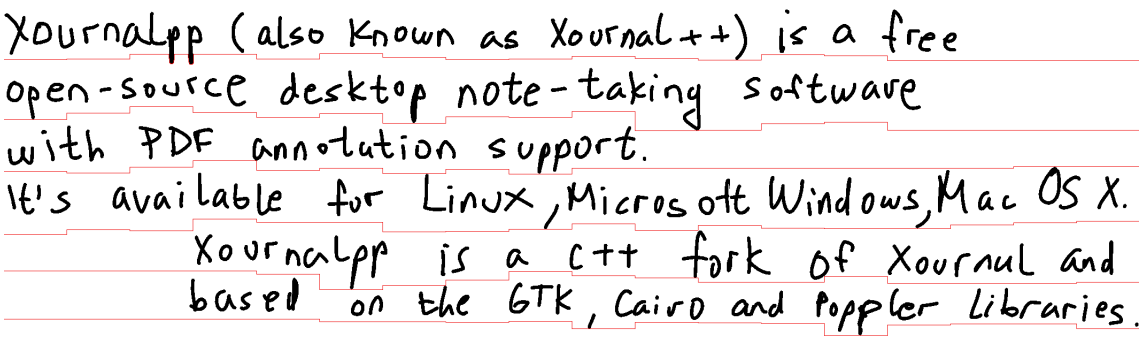
Iako algoritam nije dizajniran za segmentaciju linija kada je u pitanju programski kôd, primećujemo da se nije loše „snašao" u tom slučaju (Slika 2.21).



Slika 2.21: Segmentacija linija u slučaju programskog kôda

Na kraju, dajemo izlaz programa u slučaju slika koje ćemo mi koristiti. To su slike visoke rezolucije na kojim je prikazan tekst pisan štampanim slovima na engleskom

jeziku u programu Xournal++.

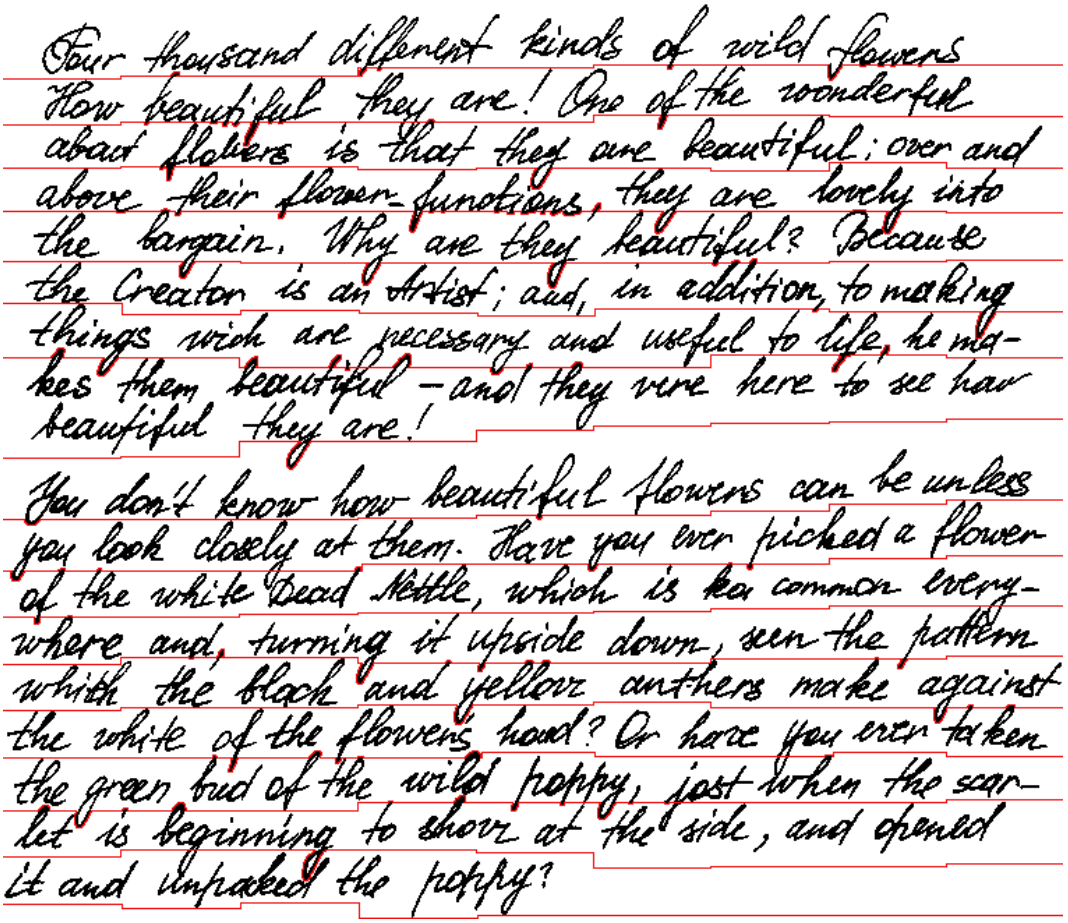


2.3 Problemi i dalja unapređenja

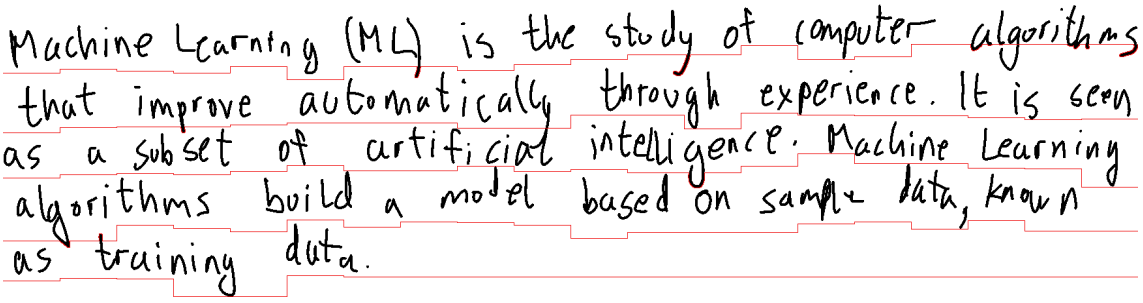
Ključ za uspeh algoritma jeste raspored pravougaonika, tj. inicijalna podela slike na regione teksta i pozadine. Odabir širine trake u velikoj meri utiče na pomenutu podelu. S obzirom na to da je teško odrediti idealnu širinu za svaku sliku, dešava se da beli pravougaonici ne predstavljaju uvek pozadinu i da crni pravougaonici ne predstavljaju uvek tekst. Zbog toga, potrebno je doneti niz odluka kako bi se ti problemi rešili i na kraju dobila ispravna segmentacija. To, nažalost, nije uvek lako. Glavni izazovi tiču se definisanja uslova koji utiču na te odluke. Tako, na primer, važno je što bolje odrediti uslove pod kojima je potrebno produžiti trenutnu putanju nekom linijom vodiljom. Ako je uslov previše slab, dešavaće se da pređemo u liniju ispod, a ako je uslov prejak, može se desiti da ne iskoristimo neku liniju vodilju koju je trebalo i da onda produžavamo putanju „bez navođenja", rizikujući da napravimo grešku. Takođe, nije uvek lako ni odrediti da li su u pitanju preklapajuće ili dodirujuće komponente i sa koje strane obići komponentu, odnosno gde je iseći.

U prethodnoj sekciji smo opisali izmene koje smo napravili sa namerom da poboljšamo algoritam kako bismo dobili bolje rezultate. Međutim, treba imati na umu da probleme ipak nismo u potpunosti otklonili. Na primer, i dalje se može desiti da neku liniju ne detektujemo, kao što je to prikazano na Slici 2.22, ili da isečemo neko slovo (Slika 2.23). Slično, greške se mogu desiti i u slučaju pogrešnog oblilaženja zareza, kao što se vidi na Slici 2.17, kao i u slučaju slova i i j, gde linija može otići ispod tačke.

Nažalost, opisani algoritam nije otporan na veće nagibe teksta, pre svega zbog načina na koji spajamo linije vodilje, ali ima potencijala za poboljšanje u ovom smeru.



Slika 2.22: Dve linije su spojene u jednu, jer su linije vodilje iz tih linija bile dovoljno blizu i zadovoljavale su kriterijume za povezivanje.



Slika 2.23: Slovo „g" u prvom pojavljivanju reči „algorithms" je isečeno, umesto da je linija išla oko njega kao što je to slučaj u drugom pojavljivanju iste reči. Sa druge strane, slovo „y" u reči „automatically" je opravdano isečeno, jer nije bilo drugog izbora.

Glava 3

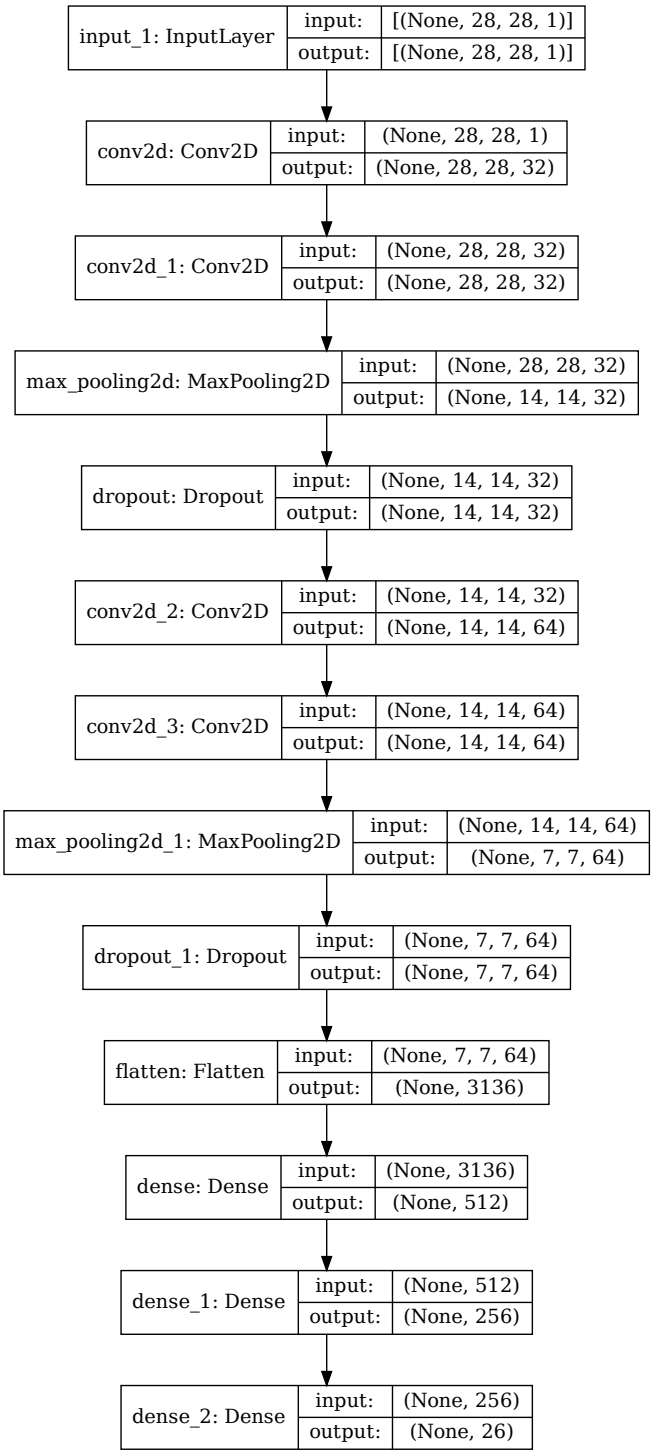
Prepoznavanje slova

U ovom odeljku bavićemo se klasifikacijom slova, konkretno prepoznavanjem pojedinačnih slova i bigrama. Predstavićemo naše skupove podataka koje smo koristili za rešavanje ovog problema. Deo podataka je ručno napravljen, dok je ostatak generisan.

Najpre ćemo opisati model koji služi za prepoznavanje jednog slova, zatim modele koji se mogu koristiti za prepoznavanje dva spojena slova i, na kraju, model koji je u stanju da odredi da li je na slici jedno slovo ili su dva spojena. Pomenuti modeli spadaju u grupu *konvolutivnih neuronskih mreža* i njihova struktura je prikazana na Slici 3.1. Prednost ovih mreža je u tome što su u stanju da same konstruišu atribute iz sirovih podataka i dosta su fleksibilne. Međutim, fleksibilnost koju nude nosi sa sobom dodatne izazove pri optimizaciji. Možemo reći da je jedan od glavnih problema pri obučavanju ovih mreža svakako preprilagođavanje (*eng. overfitting*). Da bismo videli u kojoj meri su naši modeli uspeli da prevaziđu ovaj problem i da generalizuju na osnovu trening podataka, primenili smo tehnike evaluacije koje ćemo analizirati u nastavku. Dodatno, videćemo uspešnost modela u slučaju generisanih slova.

Na kraju poglavlja, ukazaćemo na probleme i nedostatke koji su uglavnom karakteristični za konvolutivne mreže.

U ovom poglavlju nećemo ulaziti u previše implementacionih detalja, kao što su vrednosti određenih parametara, već ćemo akcenat staviti na rezultate. Sve dodatne informacije vezane za implementaciju se mogu pronaći na GitHub repozitorijumu [2].



Slika 3.1: Arhitektura modela. Svi modeli imaju istu strukturu, s tim što se dimenzija izlaza u poslednjem sloju razlikuje i zavisi od broja klasa koje mreža prepoznaje. U našem slučaju, veličina izlaza može biti 26, kao što je i prikazano na slici, ili 2, ako je u pitanju model zadužen za binarnu klasifikaciju (slovo ili bigram).

3.1 Prepoznavanje jednog slova

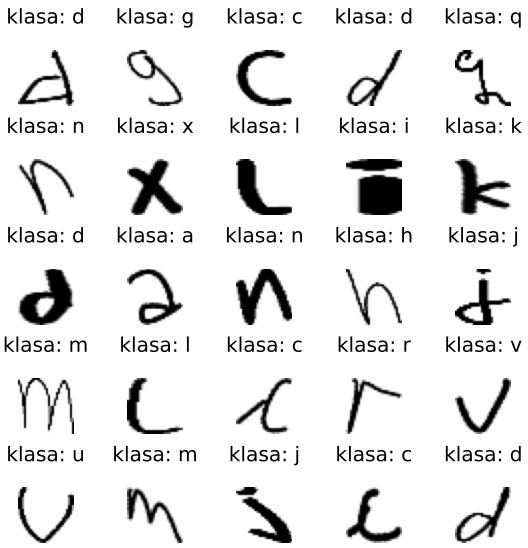
Skupovi podataka koji su korišćeni sastoje se od malih štampanih slova engleskog alfabeta, rukom pisanih u programu Xournal++. Od jedne slike napravljeno je devet slika tako što su upotrebljene tri različite debljine olovke (normal, medium i bold) kako bismo dobili tri slike od polazne, a zatim je svaka od njih rotirana za 0, 15 i -15 stepeni. U nastavku ćemo uračunati augmentaciju kada budemo govorili o veličini skupova. Svaka slika ima jedan kanal i vrednosti piksela su celi brojevi u intervalu $[0, 255]$. U fazi pretprocesiranja, slike sečemo do ivica, menjamo im veličinu na 28×28 piksela i vrednost svakog piksela skaliramo na interval $[0, 1]$.

Važno je istaći da je skup podataka ručno podeljen na trening, validacioni i test skup, kako bismo izbegli optimističnu ocenu modela. Naime, ako bismo svaku sliku nasumično raspoređivali u skupove, dešavalo bi se da neke od pomenutih devet varijanti jedne slike završe u trening skupu, a ostale u validacionom/test skupu, što nije dobro, jer time olakšavamo modelu da napravi tačnu predikciju. Ručnom podelom podataka ćemo izbeći taj problem. Na Slici 3.2 prikazano je po deset nasumično odabranih slova svake klase iz trening skupa. Validacioni i test skupovi su slične strukture kao trening skup i za njih , takođe, važe gore navedena svojstva. Nekoliko nasumično odabranih slika iz test skupa je dato na Slici 3.3.

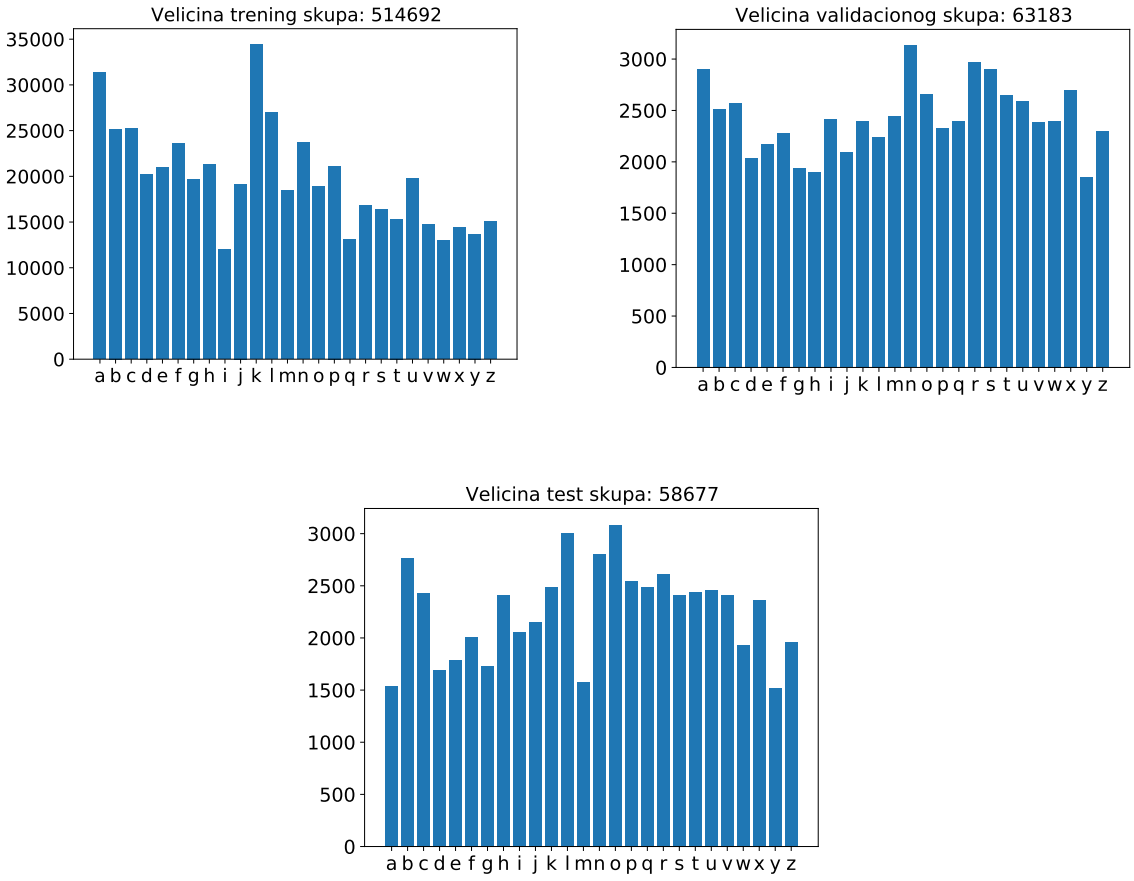
Sva tri skupa su nebalansirana. Razlog zašto je trening skup nebalansiran je taj što je mreži bilo teže da nauči neka slova, pa su u trening skupu ta slova češća od ostalih, dok su validacioni i test skupovi takvi zbog ručne podele podataka, iako nam je svako slovo podjednako bitno[1]. Na Slici 3.4 su prikazani histogrami koji predstavljaju njihovu raspodelu.

[1]Naravno, prilikom korišćenja modela, bitnije je da se greške ređe dešavaju na frekventnijim slovima, međutim, posmatrajući samo model čiji je cilj da prepozna slova, nismo hteli nijedno slovo da favorizujemo.

Slika 3.2: Trening skup. U svakoj vrsti je prikazano po deset nasumično odabranih instanci jedne klase.

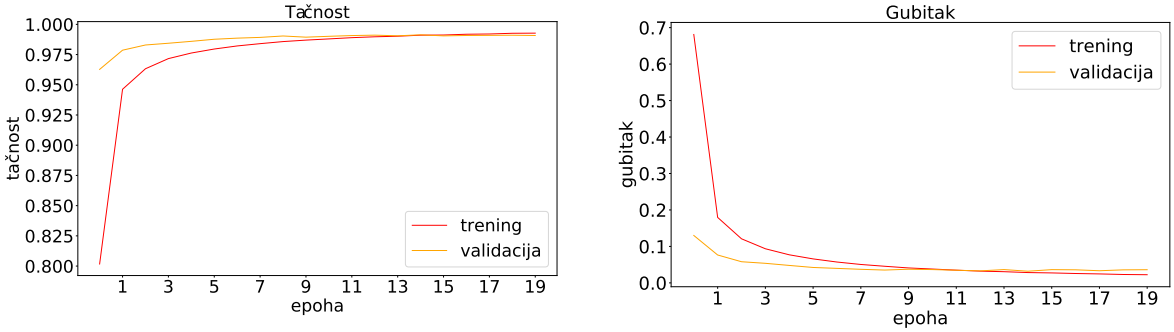


Slika 3.3: Test skup



Slika 3.4: Raspodela podataka

Validacioni skup koristimo kako za ručno podešavanje parametara mreže, kao što su dropout, broj slojeva, filtera i slično, tako i za određivanje broja epoha. Naime, primenićemo metod ranog zaustavljanja (*eng. early stopping*) - sačuvaćemo model koji se najbolje pokazao na validacionom skupu tokom svih epoha. Na taj način, pokušaćemo da izbegnemo preprilagođavanje modela trening podacima. Grafici koji prikazuju tačnost (*eng. accuracy*) i gubitak (*eng. loss*) na trening i validacionom skupu prikazani su na Slici 3.5. Maksimalan broj epoha je 30, međutim, zbog platoa[2] na koji smo naišli, odlučili smo da zaustavimo dalje obučavanje nakon 20 epoha, dok smo model sačuvali nešto ranije.



Slika 3.5: Tačnost i gubitak

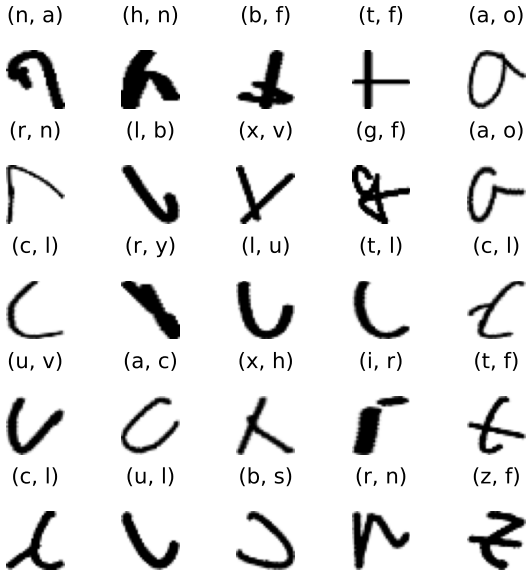
U Tabeli 3.1 može se videti tačnost na trening, validacionom i test skupu. Primećujemo da je tačnost na trening skupu veća nego na validacionom, što nije slučaj na grafiku (relevantna nam je epoha kada smo poslednji put sačuvali model). Razlog je taj što se dropout ne primenjuje kada se računa tačnost i gubitak na trening skupu tokom obučavanja, pa na grafiku dobijamo nešto niže vrednosti za tačnost na trening skupu, odnosno više vrednosti za gubitak.

Trening skup	99.68%
Validacioni skup	99.13%
Test skup	98.23%

Tabela 3.1: Tačnost na skupovima

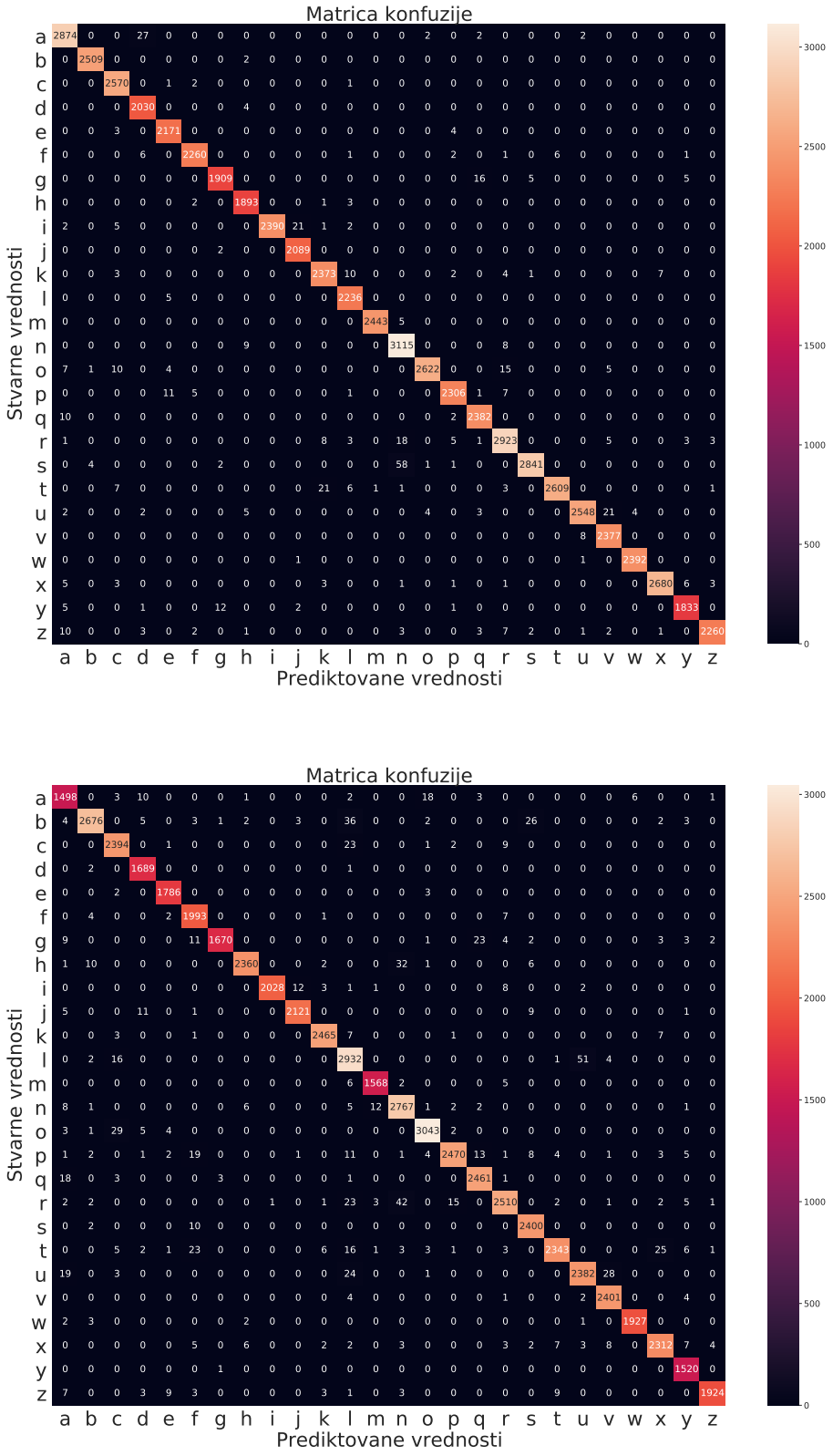
Na Slici 3.6 je izdvojeno nekoliko slova koja su pogrešno klasifikovana. Možemo primetiti da bi neke od tih grešaka i čovek napravio, kao i da su neke pogrešne klasifikacije posledica grešaka u podacima.

[2]U našem slučaju, plato je deo prostora grešaka u kojem nije bilo smanjenja validacionog gubitka tokom pet uzastopnih epoha.



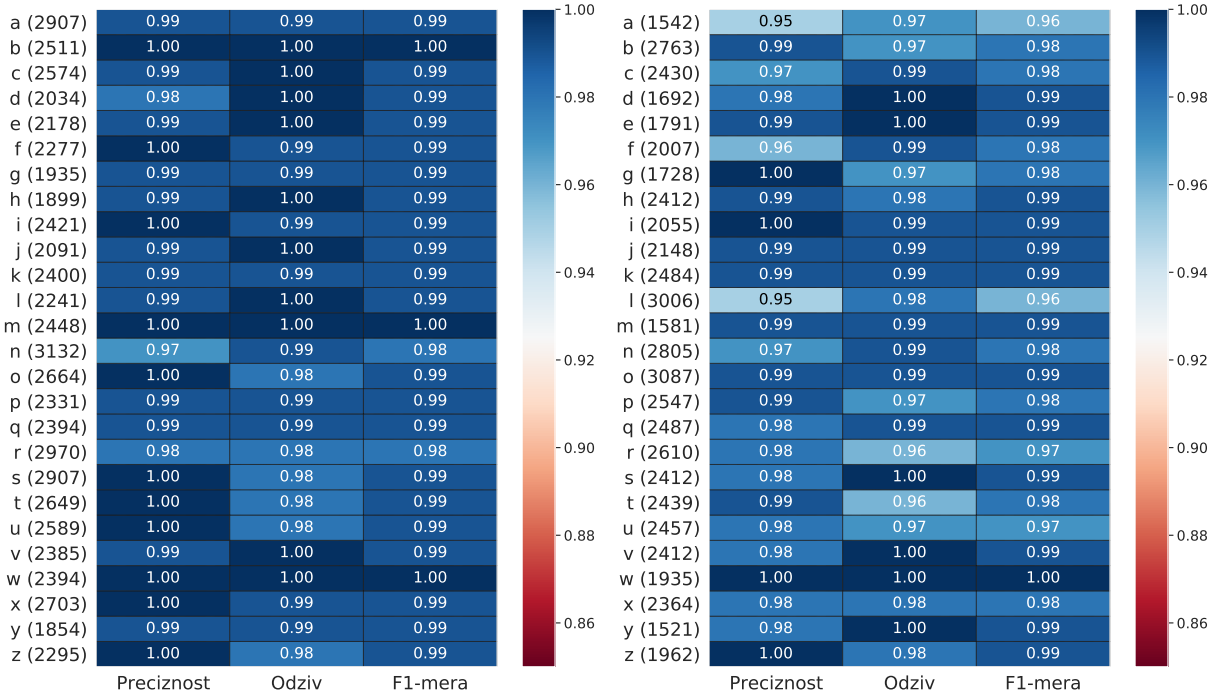
Slika 3.6: Greške na test skupu. Prvo slovo u zagradi označava stvarnu klasu, a drugo prediktovanu.

Matrice konfuzije na validacionom i test skupu su prikazane na Slici 3.7. Zapažamo da su matrice dijagonalno dominantne, što je svakako svojstvo koje želimo, jer je tačnost modela pozitivno korelisana sa brojem elemenata na dijagonali. Posmatrajući elemente van glavne dijagonale, lako se može uočiti koja slova model često meša. Tako, na primer, vidimo da je u test skupu slovo l najčešće zamenjeno slovom u. Ove informacije iz matrice konfuzije moguće je iskoristiti da se isprave greške pri prepoznavanju reči - ako je prepoznata reč *sfart*, verovatnije je u pitanju bila reč *start* nego *smart*, jer je slovo t češće pomešano sa f (23) nego što je slovo m pomešano sa f (0), što možemo zaključiti posmatrajući kolonu f u matrici konfuzije na test skupu.



Slika 3.7: Matrica konfuzije. Gornja slika je matrica za validacioni skup, a donja za test skup.

Klasifikacioni izveštaji na validacionom i test skupu su prikazani na Slici 3.8. Preciznost je mera koja nam govori u koliko posto slučajeva smo bili u pravu kada smo rekli da je nešto određena klasa. Na primer, ako je preciznost za klasu a jednaka 0.99, to znači da je 99% instanci koje smo klasifikovali kao a, stvarno a, dok u preostalih 1% nismo bili u pravu. Odziv, sa druge strane, govori nam koliko smo posto instanci određene klase prepoznali da pripada toj klasi. Tako, na primer, ako je odziv za klasu a jedanak 0.99, za 99% instanci klase a smo rekli da zaista i pripada toj klasi, dok smo pogrešili za ostalih 1% instanci. Kako je potrebno analizirati obe mere prilikom evaluacije modela, često je lakše izvesti jednu meru iz njih i nju pratiti. F1-mera (*F1-score*) je harmonijska sredina preciznosti i odziva. Intuitivno, možemo je posmatrati kao broj između preciznosti i odziva koji je bliži lošijoj vrednosti.



Slika 3.8: Klasifikacioni izveštaj. Na slici levo je prikazan klasifikacioni izveštaj na validacionom skupu, a na slici desno na test skupu.

Iz prikazanog izveštaja možemo zaključiti da model najčešće greši kada proglasi neko slovo za a ili l (najmanja preciznost, 0.95), dok najteže prepoznaje slova r i t (najmanji odziv, 0.96). Dodatno, iz matrice konfuzije vidimo da je u slučaju pogrešno klasifikovanog slova kao a, najčešće u pitanju slovo u (19) i slično važi za slova l i b (36). Takođe, slovo r se najčešće pogrešno klasifikuje kao n (42), dok se t najčešće pogrešno klasifikuje kao x (25). Namopinjemo da ovi zaključci ne oslikavaju

nužno karakteristike modela kako smo ih mi ovde opisali, naročito iz razloga što test skup nije izbalansiran, tako da i rezultati koje smo dobili za pojedinačna slova nisu uporedivi. Isto tako i greške u podacima mogu dovesti do pogrešnih zaključaka, te naš model uopšte ne mora mešati slovo b sa l u praksi.

3.2 Prepoznavanje bigrama

Do sada smo videli kako možemo da prepoznamo jedno slovo. Međutim, zanima nas šta se dešava u slučaju da su dva slova (slučajno) spojena. Ako, recimo, sliku ad propustimo kroz mrežu koju smo dobili u prethodnoj sekciji, ne možemo očekivati da će prepoznati slovo a ili slovo d. S obzirom na to da se ovakva spajanja dešavaju, a da ih nije lako razdvojiti u procesu segmentacije, napravićemo model koji će biti u stanju da prepozna slova koja se pojavljuju u bigramu. Kako bigrama ima 26^2, bio bi nam potreban model koji prepoznaje toliko različitih klasa. Čak i da posmatramo redukovani skup bigrama (one koji se najčešće pojavljuju u engleskom jeziku), opet bi broj mogućih klasa bio značajno veći od 26. Dakle, već naslućujemo dosta manju tačnost nego u slučaju modela koji je prepoznavao samo jedno slovo. Takođe, postavlja se i pitanje kako bi model koji klasifikuje bigrame grešio - da li bi češće pogrešio oba slova, ili samo jedno. Zbog ovih problema, pokušaćemo da se zadržimo na istom broju klasa kao i u slučaju jednog slova. Umesto jednog modela, napravićemo dva.

Prvi model ćemo obučiti tako da prepoznaje prvo slovo, a drugi model drugo slovo u bigramu. Na primer, rezultat prvog modela za sliku ad treba da bude a, a rezultat drugog modela d. Ovi modeli će imati istu strukturu, ali će biti trenirani na različitim podacima, pa će naučiti različite vrednosti za parametre. Takođe, biće nezavisni, tj. grešiće nezavisno.

Ostaje još da napravimo bigrame. To ćemo uraditi tako što ćemo za svaki bigram[3] izabrati iz skupova slova nasumično prvo i drugo slovo tog bigrama i spojiti ih, ukoliko zadovoljavaju određene uslove koji se odnose na veličinu slika. Važno je napomenuti da ćemo trening skup dobiti spajanjem pojedinačnih slova iz trening skupa za jedno slovo, validacioni skup spajanjem slova iz validacionog skupa i test skup spajanjem slova iz test skupa. U suprotnom, mogli bismo da dobijemo optimističnu ocenu modela. Primera radi, ako jedno konkretno slovo a spojimo sa sa nekim

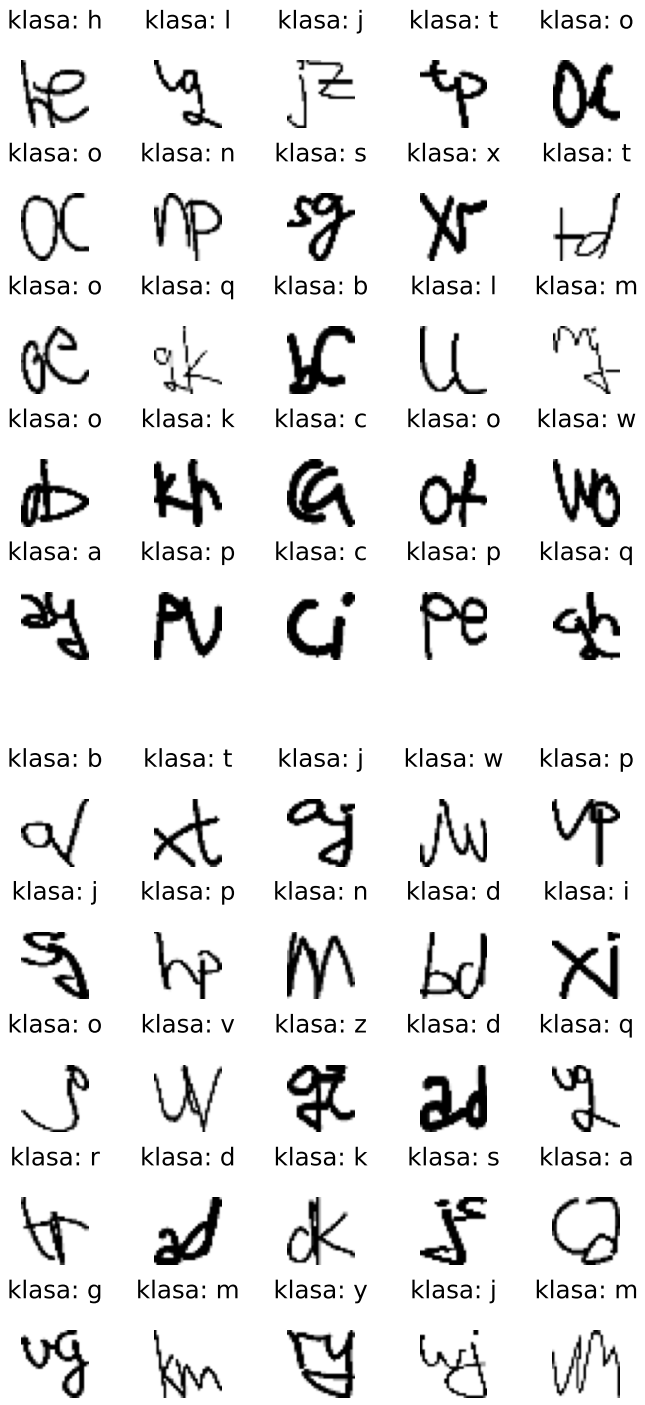
[3]I u ovom slučaju smo pokušali da redukujemo broj bigrama koje posmatramo na one koji se najčešće pojavljuju u engleskom jeziku, međutim, to nam nije dalo bolji rezultat, što možda možemo da opravdamo činjenicom da se broj klasa nije smanjio nakon eliminacije retkih bigrama.

slovom b da bismo dobili instancu ab u trening skupu, a potom to isto a spojimo sa nekim drugim slovom b da bismo dobili instancu ab u test skupu, onda će te dve instance imati nešto zajedničko, što može olakšati prvom modelu da tačno klasifikuje prvo slovo kao a. To svakako ne želimo, jer nam je cilj da testom iznenadimo model kako bismo dobili što realističniju ocenu.

Na Slici 3.9 prikazano je nekoliko instanci iz svake klase u trening skupu za prvo slovo bigrama, a na Slici 3.10 za drugo. Takođe, možemo videti i nekoliko nasumično odabranih instanci u test skupu za prvo slovo i drugo slovo (Slika 3.11).

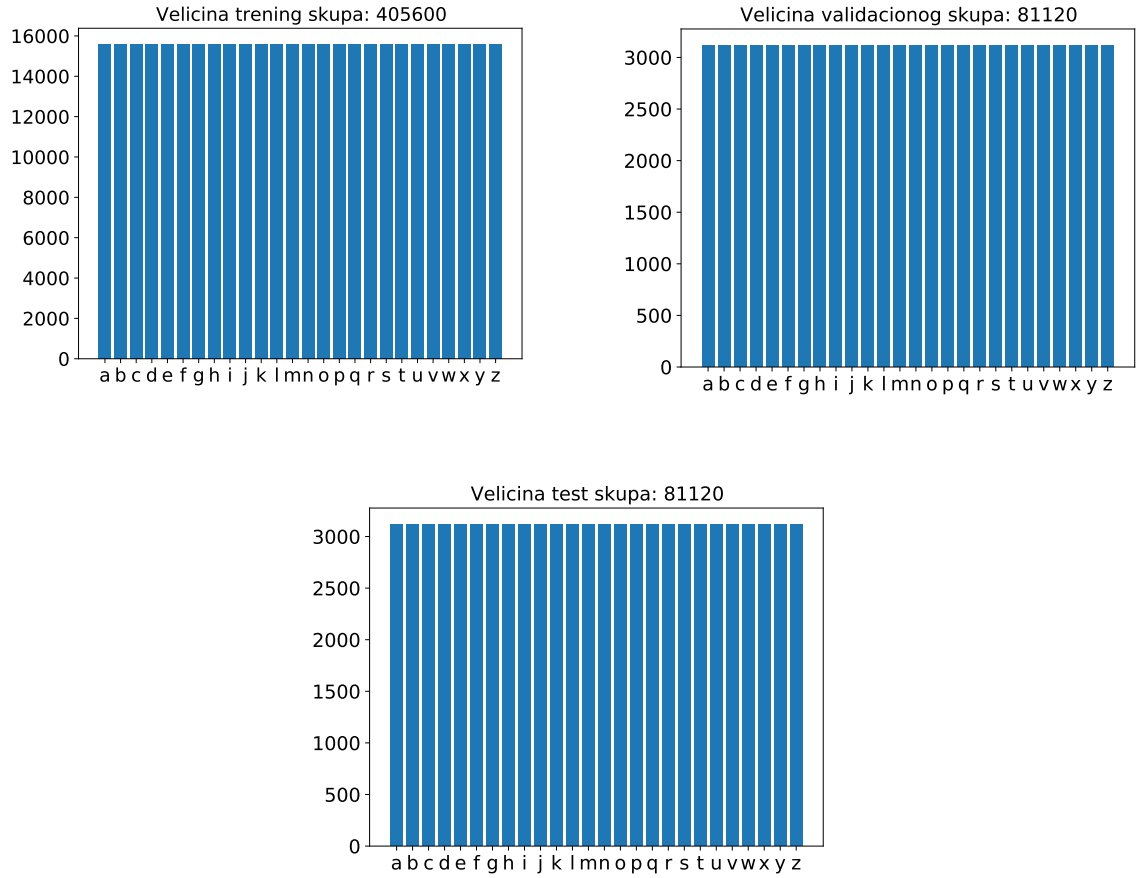
Slika 3.9: Prvo slovo bigrama: trening skup. U svakoj vrsti je prikazano po deset nasumično odabranih instanci jedne klase.

Slika 3.10: Drugo slovo bigrama: trening skup. U svakoj vrsti je prikazano po deset nasumično odabranih instanci jedne klase.



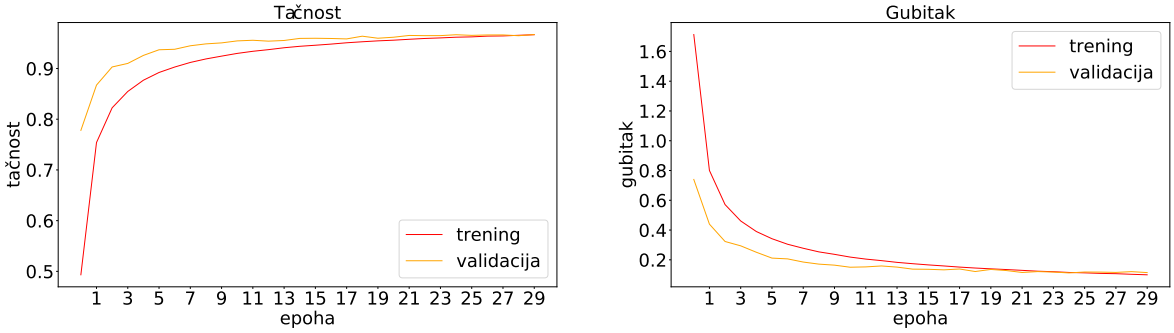
Slika 3.11: Test skup. Na gornjoj slici nalaze se test podaci za prvo slovo, a na donjoj slici za drugo slovo bigrama.

Za razliku od prethodnog slučaja, sada ćemo imati izbalansirane skupove. Na Slici 3.12 možemo videti raspodelu slova u trening, validacionom i test skupu. Ista je raspodela u skupu koji koristimo za prepoznavanje prvog slova, kao i u skupu za prepoznavanje drugog slova.

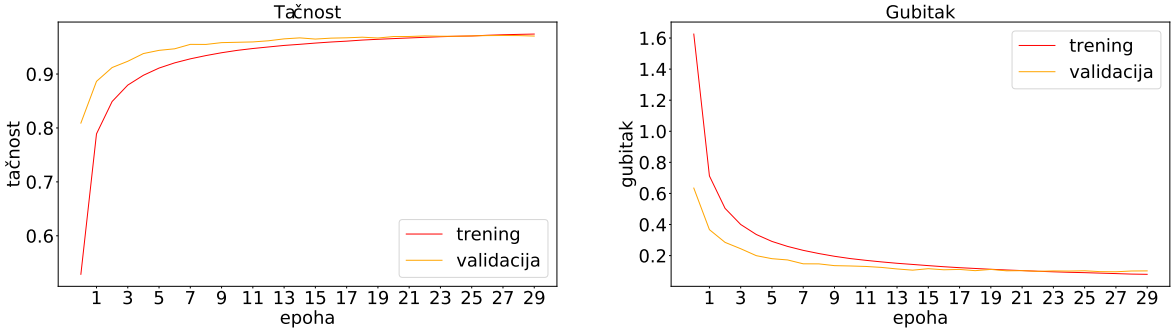


Slika 3.12: Raspodela podataka u skupu za prepoznavanje prvog (drugog) slova bigrama. Svaki par slova u trening skupu sadrži po 200 instanci za svaku od tri debljine linije, što ukupno čini $26^2 \cdot 200 \cdot 3 = 405\ 600$ *instanci. Slično, svaki par slova u validacionom i test skupu sadrži po 40 instanci za svaku od tri debljine linije,* $26^2 \cdot 40 \cdot 3 = 81\ 120$.

Grafici koji prikazuju tačnost (*eng. accuracy*) i gubitak (*eng. loss*) na trening i validacionom skupu za prvo slovo bigrama prikazani su na Slici 3.13, a za drugo slovo bigrama na Slici 3.14. Maksimalan broj epoha je 30 i korišćena je metoda ranog zaustavljanja. U ovom slučaju nismo naišli na plato, kao što je to bio slučaj u prethodnoj sekciji, tj. završili smo svih 30 epoha treniranja.



Slika 3.13: Prvo slovo bigrama: tačnost i gubitak



Slika 3.14: Drugo slovo bigrama: tačnost i gubitak

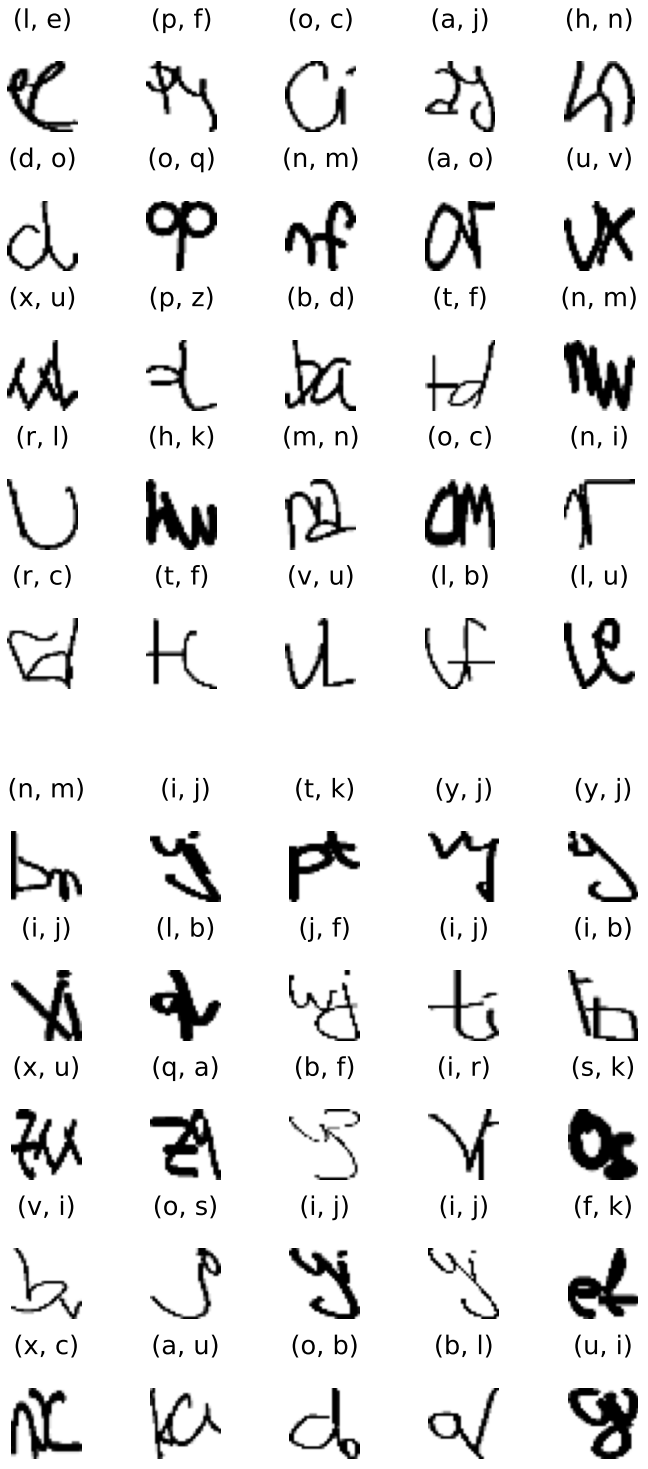
U Tabeli 3.2 može se videti tačnost na trening, validacionom i test skupu za prvo slovo, odnosno u Tabeli 3.3 za drugo slovo bigrama. Ponovo napominjemo da se dropout ne koristi tokom računanja tačnosti i gubitka na trening skupu tokom obučavanja, te da zato dolazi do razlike u ovim merama koje uočavamo u tabelama i na graficima. Iz dobijenih rezultata može se zaključiti da je ovaj problem nešto teži nego u slučaju klasifikacije samo jednog slova, što je i očekivano, s obzirom na to da spajanjem neka dva slova možemo dobiti neko drugo slovo, kao što je to slučaj sa r i n, čijim spajanjem možemo dobiti slovo m. Dalji uvid u neka pogrešno klasifikovana slova (Slika 3.15) i matrice konfuzije (Slika 3.16 i 3.17) otkriva još takvih primera.

Trening skup	99.00%
Validacioni skup	96.70%
Test skup	96.13%

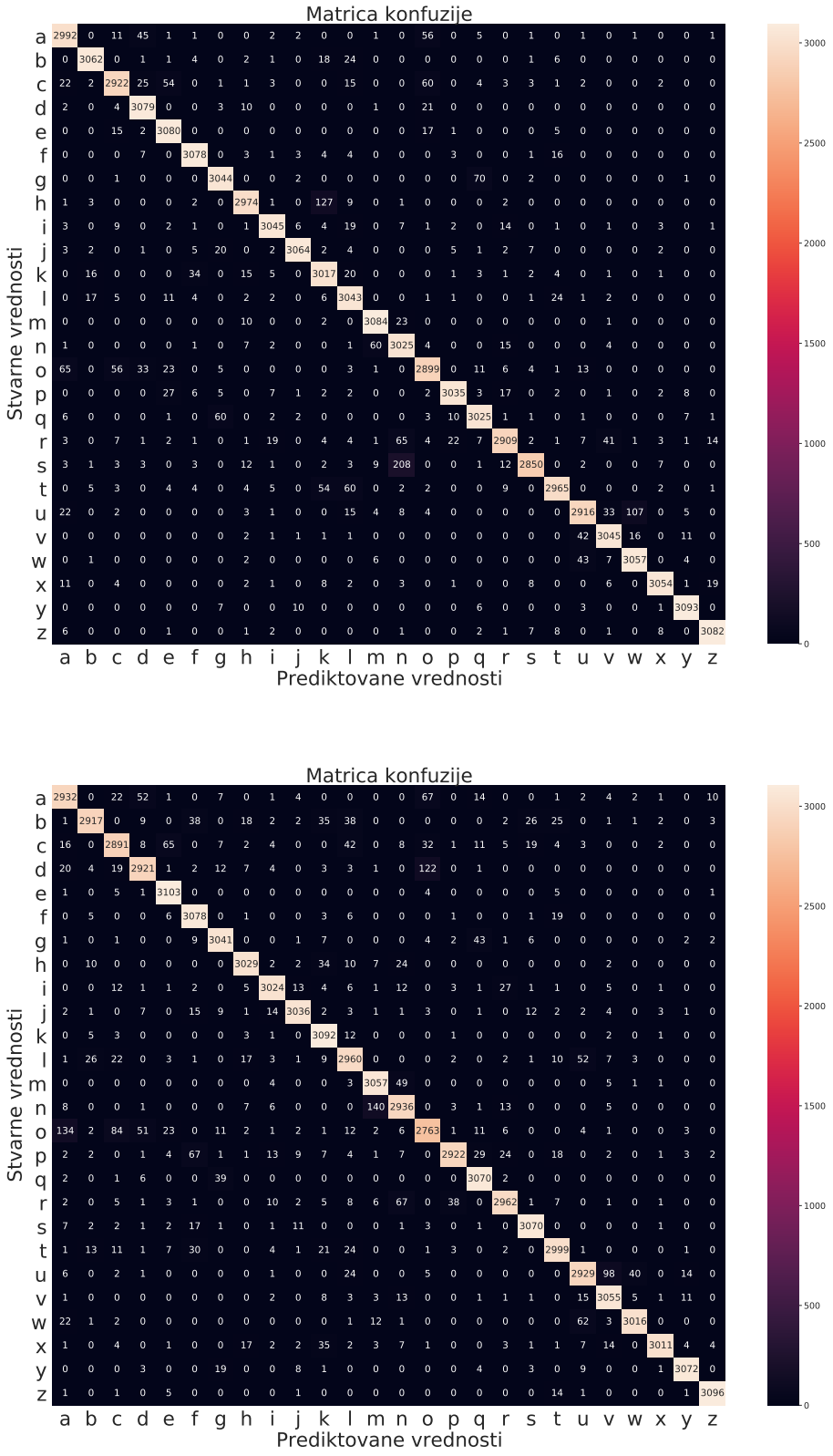
Tabela 3.2: Prvo slovo bigrama: tačnost

Trening skup	99.24%
Validacioni skup	97.15%
Test skup	96.44%

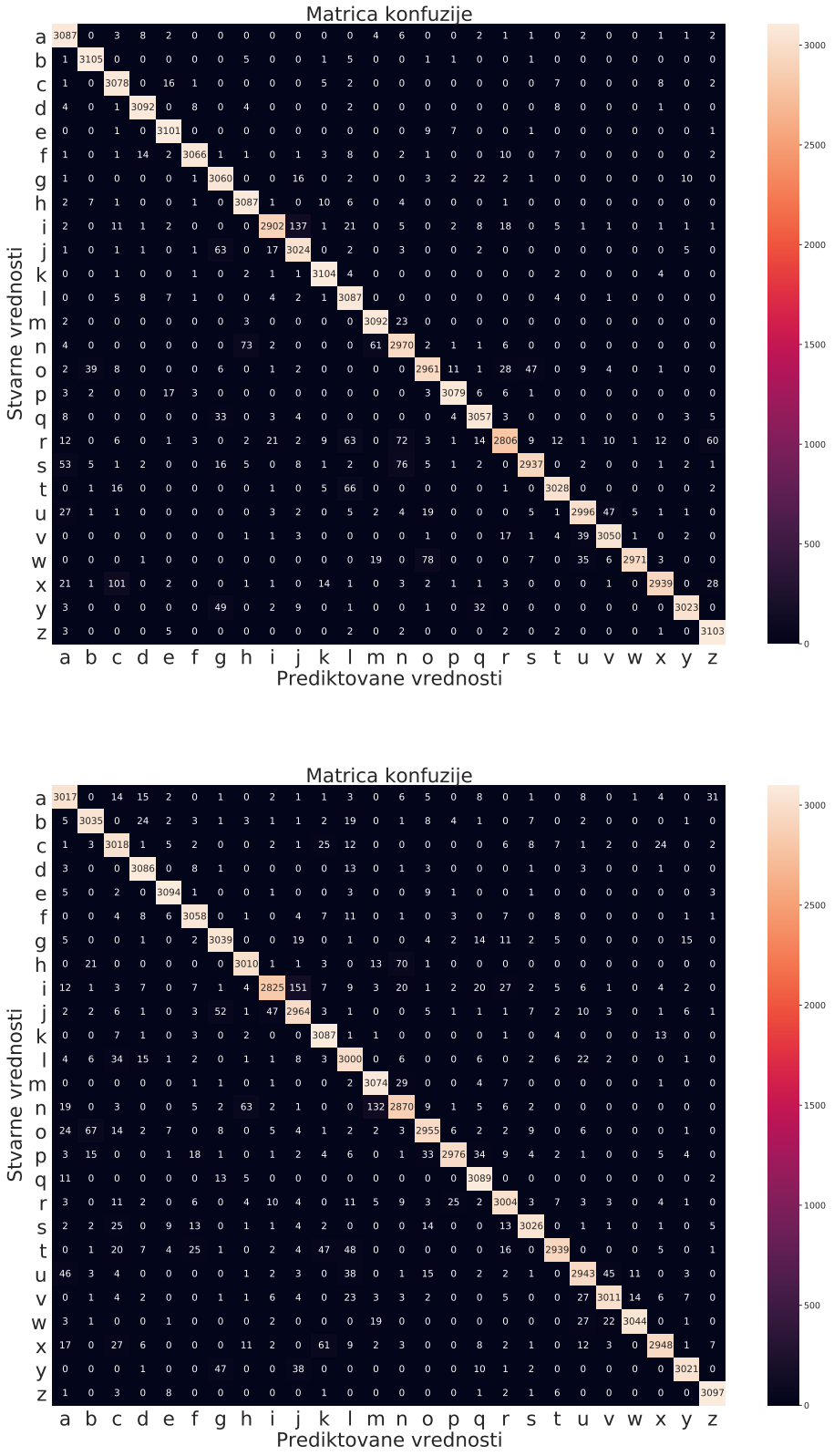
Tabela 3.3: Drugo slovo bigrama: tačnost



Slika 3.15: Greške na test skupu. Na gornjoj slici prikazane su greške za prvo slovo, a na donjoj slici za drugo. Prvo slovo u zagradi označava stvarnu klasu, a drugo prediktovanu.

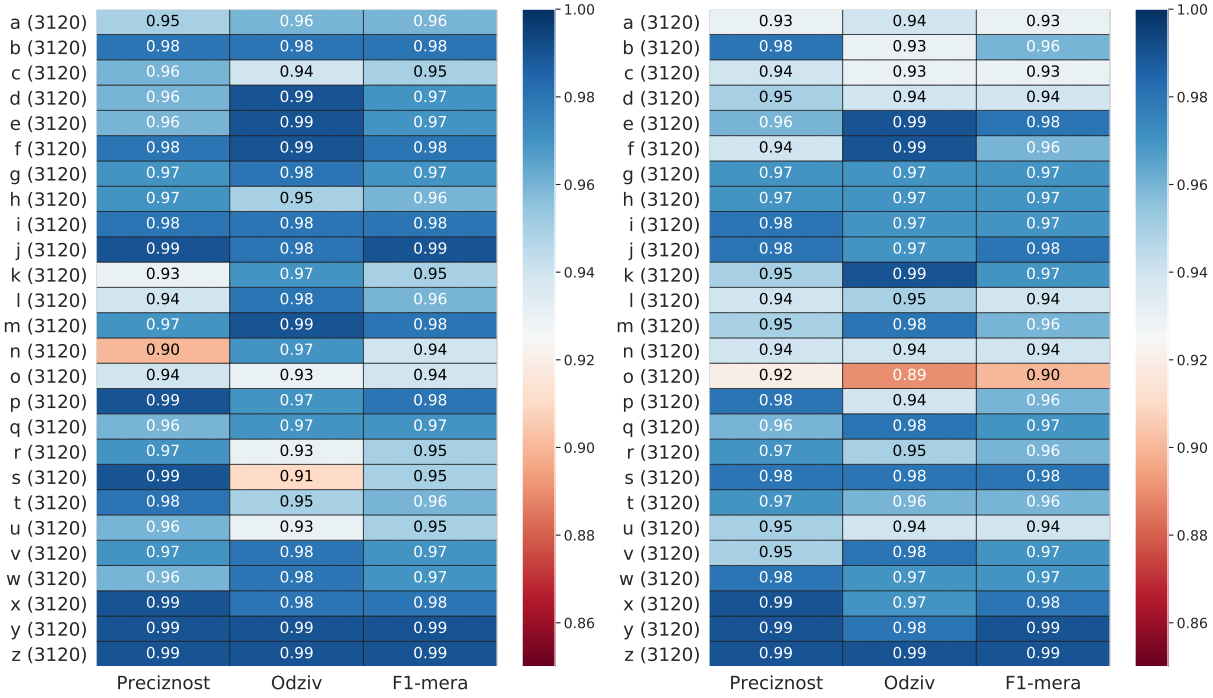


Slika 3.16: Prvo slovo bigrama: matrica konfuzije. Gornja slika je matrica za validacioni skup, a donja za test skup.

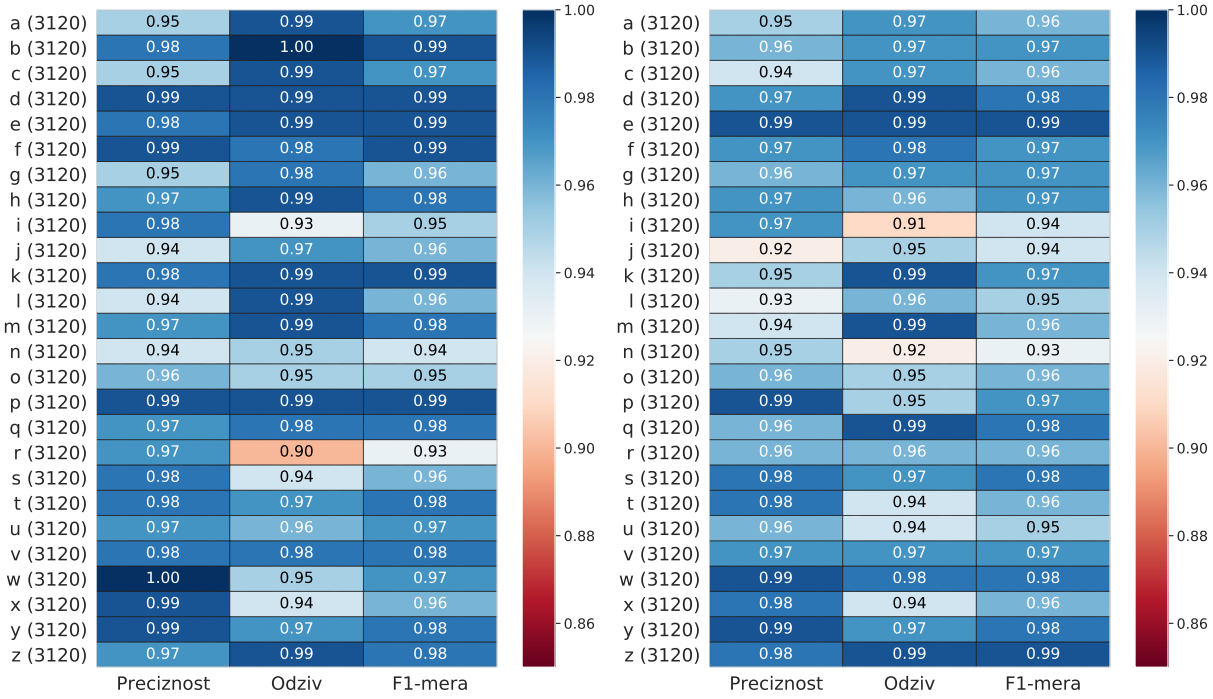


Slika 3.17: Drugo slovo bigrama: matrica konfuzije. Gornja slika je matrica za validacioni skup, a donja za test skup.

Klasifikacioni izveštaji za ova dva modela dati su na Slikama 3.18 i 3.19.



Slika 3.18: Prvo slovo bigrama: klasifikacioni izveštaj. Na slici levo prikazan je klasifikacioni izveštaj na validacionom skupu, a na slici desno na test skupu.

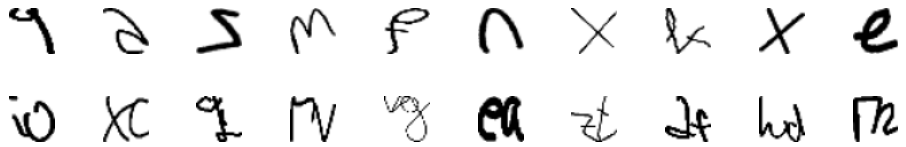


Slika 3.19: Drugo slovo bigrama: klasifikacioni izveštaj. Na slici levo prikazan je klasifikacioni izveštaj na validacionom skupu, a na slici desno na test skupu.

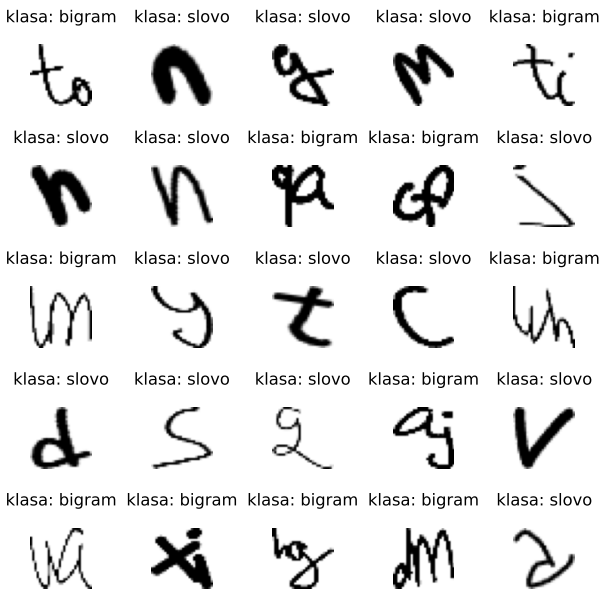
3.3 Klasifikacija jednog slova i bigrama

Pošto smo sad u stanju da prepoznamo pojedinačno slovo i bigram, bilo bi dobro da znamo kad koji model da koristimo, odnosno, potrebno je da odredimo da li se na slici nalazi jedno slovo, ili su dva spojena.

Spojićemo trening skupove iz prethodne dve sekcije da bismo dobili trening skup za ovaj model (Slika 3.20). Slično, spajamo validacione skupove i test skupove.

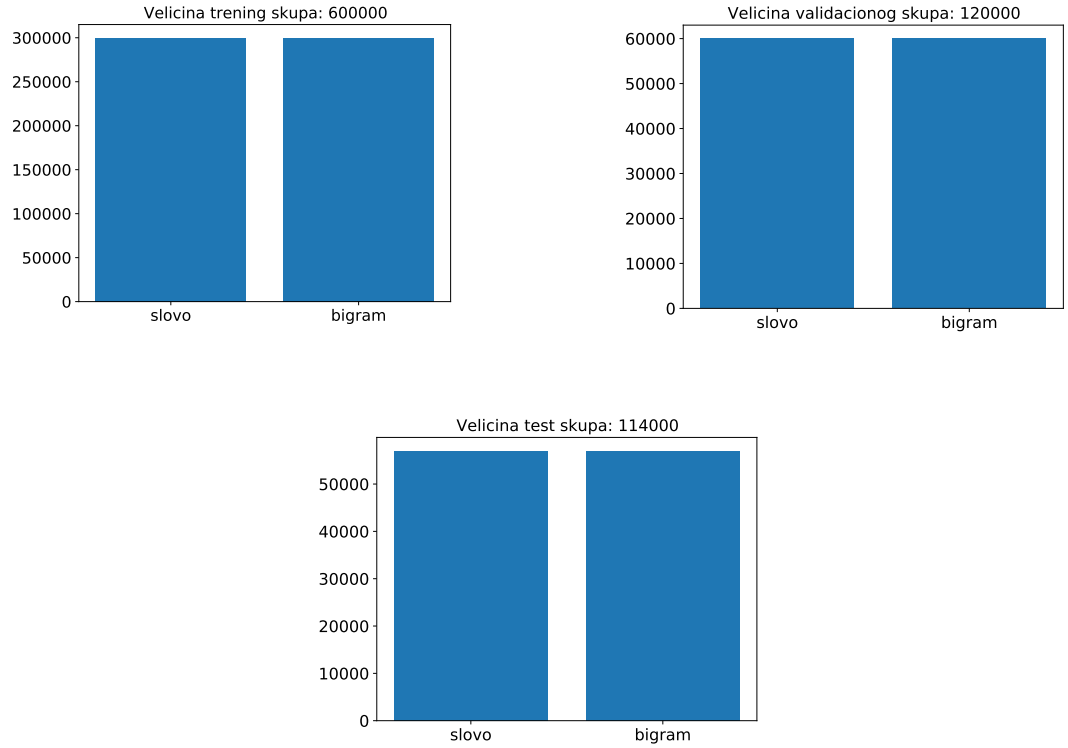


Slika 3.20: Trening skup. U prvoj vrsti se nalaze pojedinačna slova, a u drugoj bigrami.



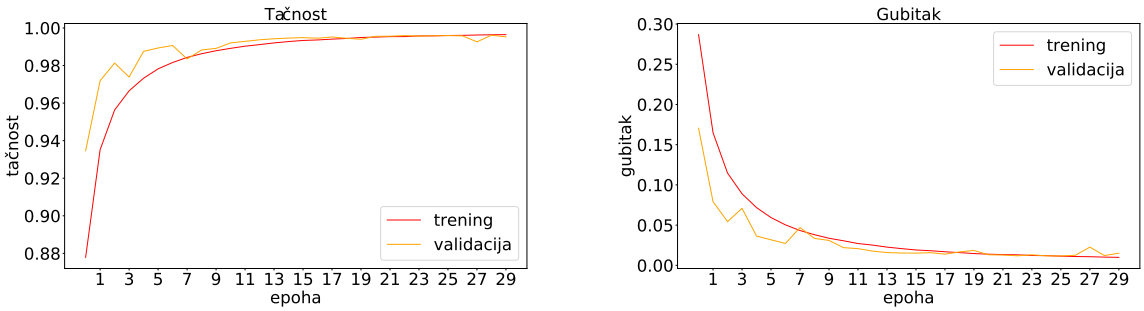
Slika 3.21: Test skup

Skupovi su izbalansirani. Njihova raspodela je prikazana na Slici 3.22. Primećujemo da smo uzeli samo podskup skupa za jedno i dva slova.



Slika 3.22: Raspodela podataka

Grafici koji prikazuju tačnost (*eng. accuracy*) i gubitak (*eng. loss*) na trening i validacionom skupu prikazani su na Slici 3.23. Kao što je to bio slučaj za prethodne modele, i ovde primenjujemo metod ranog zaustavljanja da bismo izbegli preprilagođavanje. Maksimalan broj epoha za treniranje je 30, dok na grafiku možemo videti da smo se zaustavili nešto ranije. Takođe, nije bilo platoa, pa smo izvršili svih 30 epoha obučavanja.



Slika 3.23: Tačnost i gubitak

U Tabeli 3.4 mogu se videti tačnost na trening, validacionom i test skupu. Iako se i ovde može desiti da bigram, zapravo, predstavlja istovremeno i jedno slovo, rezultati koje smo dobili su najbolji do sad, što možemo objasniti dosta manjim brojem klasa (2 naspram 26 koje smo imali do sad).

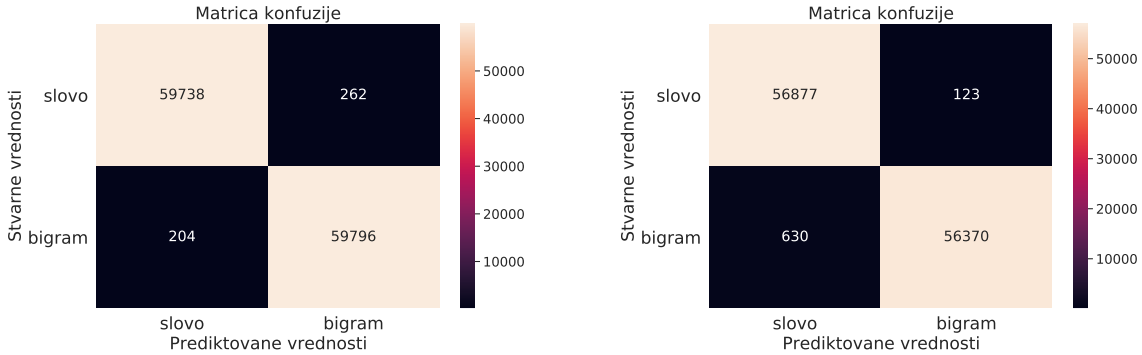
Trening skup	99.92%
Validacioni skup	99.61%
Test skup	99.34%

Tabela 3.4: Tačnost na skupovima

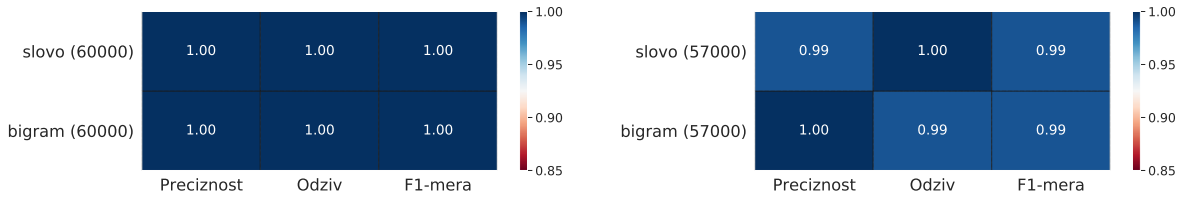
Na Slici 3.24 prikazano je nekoliko pogrešno klasifikovanih instanci.

Slika 3.24: Greške na test skupu. Prva klasa u zagradi je stvarna, a druga prediktovana.

Matrice konfuzije na validacionom i test skupu su prikazane na Slici 3.25, a klasifikacioni izveštaj na Slici 3.26.



Slika 3.25: Matrica konfuzije. Levo je prikazana matrica na validacionom skupu, a desno na test skupu.



Slika 3.26: Klasifikacioni izveštaj. Na slici levo je prikazan klasifikacioni izveštaj na validacionom skupu, a na slici desno na test skupu.

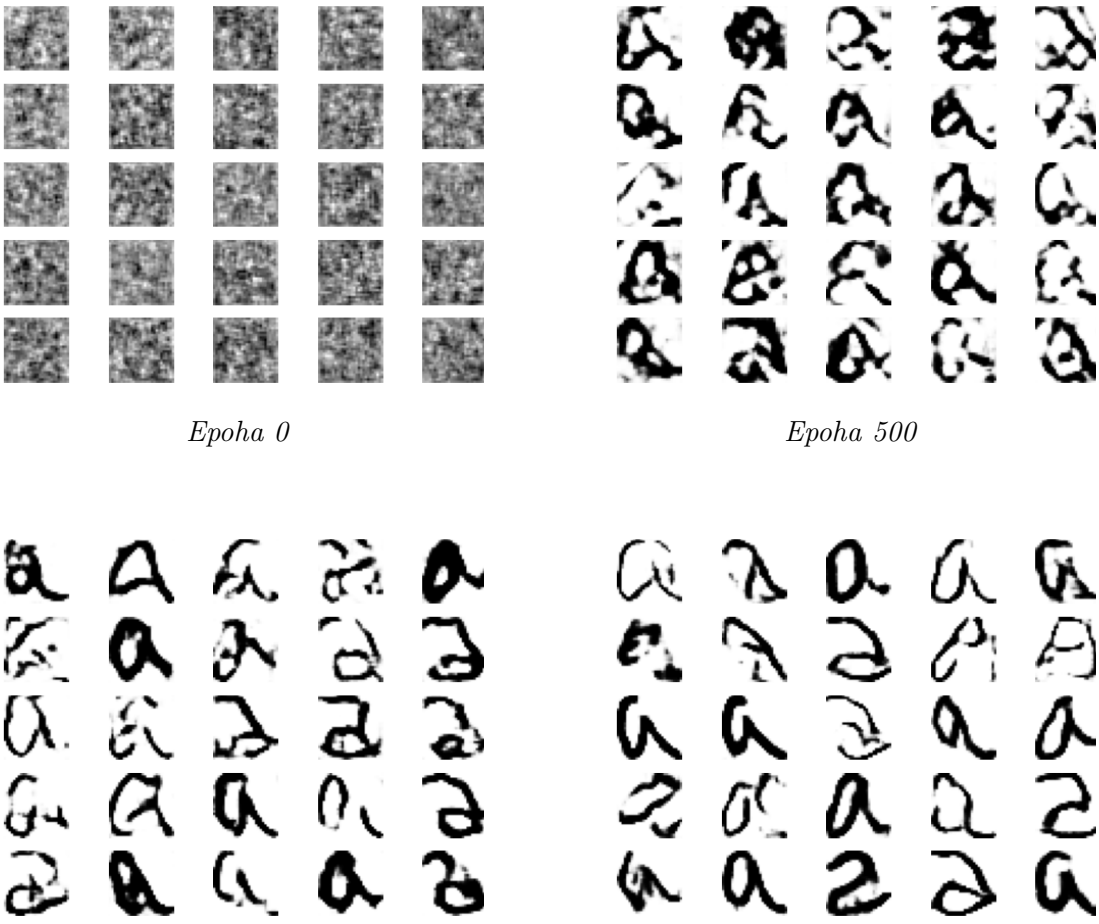
3.4 Prepoznavanje generisanih slova

U ovoj sekciji ćemo se baviti generisanjem novih slova na osnovu podataka koje smo do sad videli i analizom modela primenjenih na njih.

Generativni modeli modeluju zajedničku raspodelu promenljivih $p(x)$. Modelovanje zajedničke raspodele često zahteva veliku količinu podataka i prate je razni problemi poput prokletstva dimenzionalnosti, ali zato nudi mogućnost generisanja novih podataka. Jedan pristup za modelovanje visokodimenzionalnih podataka koristi generativne suparničke mreže, skraćeno GAN-ove (eng. *Generative Adversarial Networks*). GAN-ovi se zasnivaju na dva modela - *generatoru* i *diskriminatoru*. Cilj diskriminatora je da nauči da razlikuje prave slike od generisanih, dok je cilj generatora da generiše slike koje diskriminator neće moći da razlikuje od pravih. Ova dva modela se naizmenično treniraju. Da bi se trenirao diskriminator, potrebno mu je dati generisane i „prave" slike čiju raspodelu želimo da modelujemo. Postoje mnogi problemi pri treniranju GAN-ova i generalno važe za modele koji se teško obučavaju, ali zato, kada obučavanje uspe, daju odlične rezultate.

Koristićemo DCGAN (eng. *Deep Convolutional Generative Adversarial Networks*) implementaciju[4] da bismo generisali slova. Treniraćemo model na neaugmentovanim podacima iz trening skupa za prepoznavanje jednog slova koji smo koristili u Sekciji 3.1. Iako bi bilo bolje koristiti podatke iz test skupa, ipak ćemo se odlučiti za trening skup zbog veće količine podataka. Na Slici 3.27 možemo videti kako su se menjale slike koje je generator generisao za slovo *a* tokom epoha.

[4] `https://github.com/eriklindernoren/Keras-GAN`

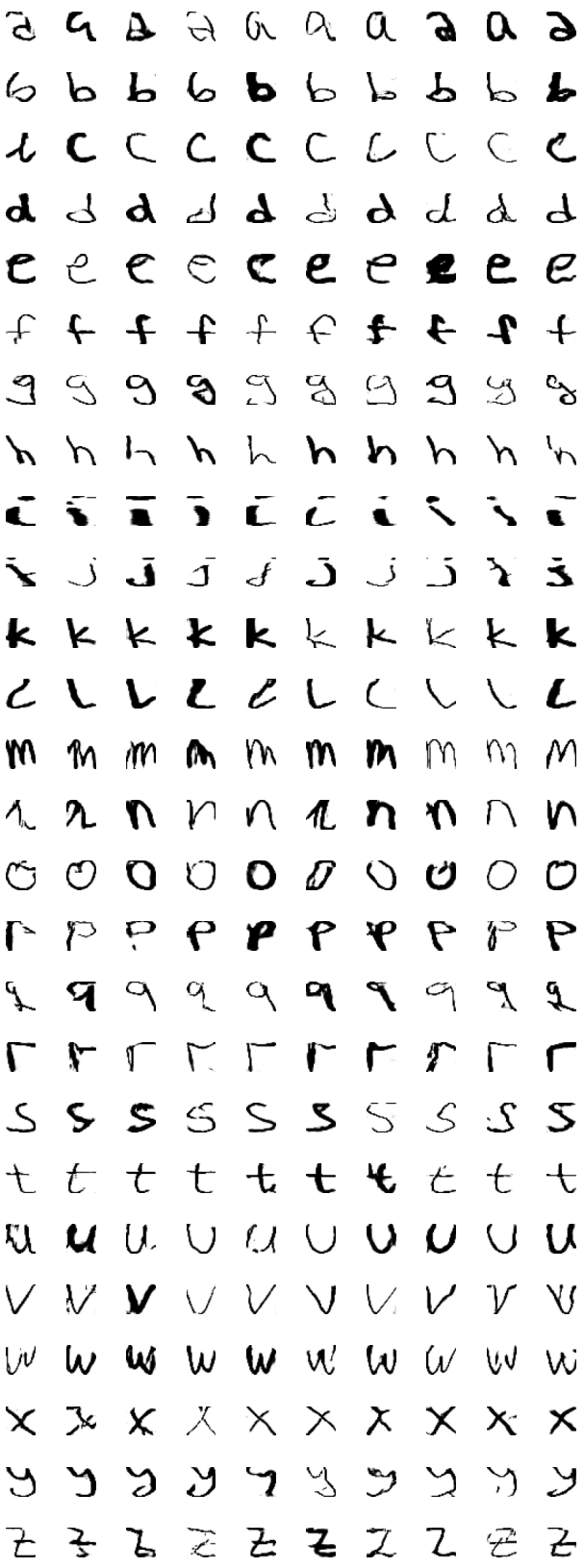


Epoha 0 *Epoha 500*

Epoha 1500 *Epoha 3950*

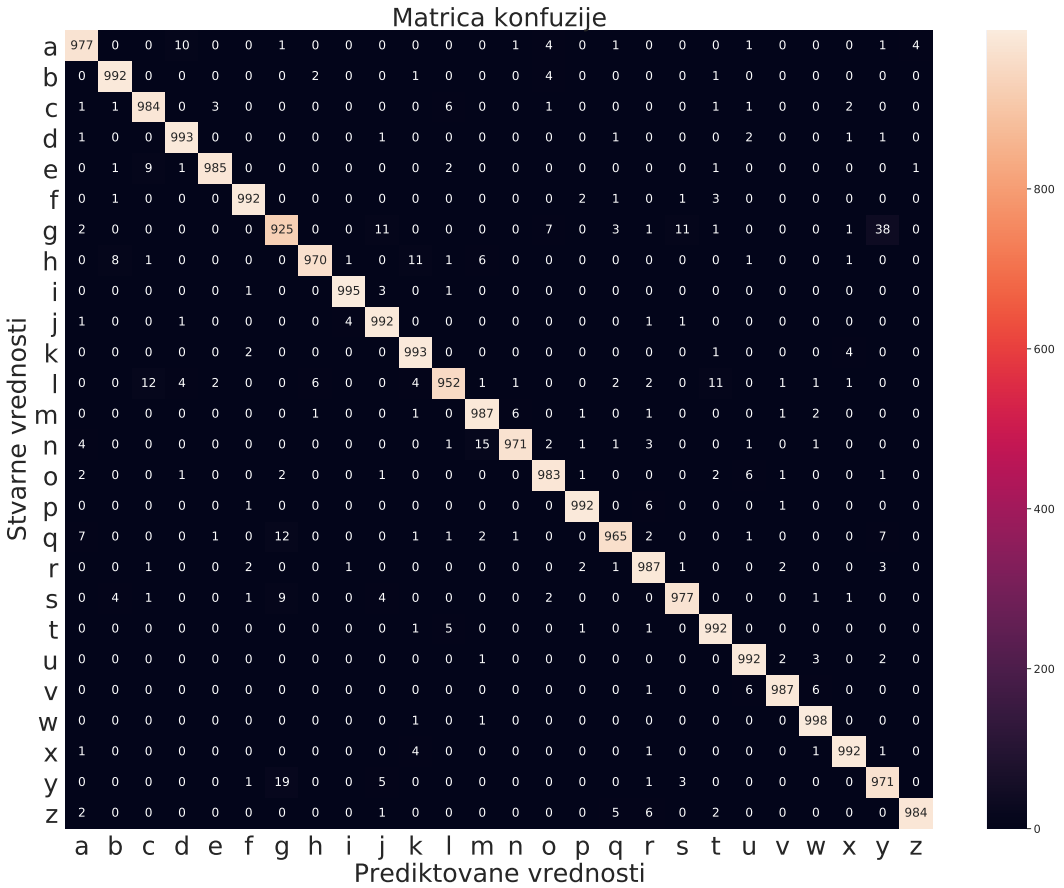
Slika 3.27: Generator za slovo a

Napravićemo novi skup podataka koji se sastoji iz generisanih slova. U skupu će biti 1 000 slika za svako slovo, što je ukupno 26 000 slika. Slike su dobijene pomoću generatora i izabrane su one za koje je diskriminator rekao da su validne sa verovatnoćom bar 0.5. Dakle, nećemo uzimati slike za koje diskriminator misli da su generisane. Prikazaćemo po deset instanci svake klase ovako dobijenog skupa (Slika 3.28).

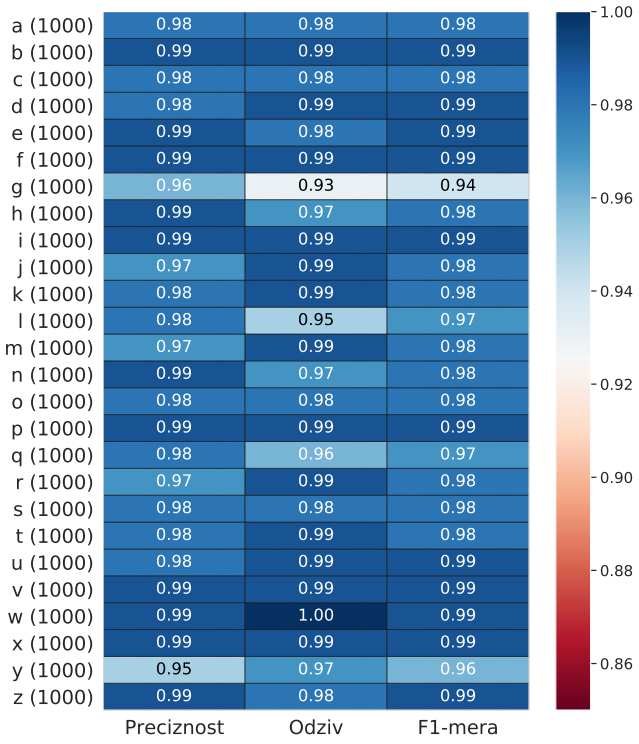


Slika 3.28: Generisani skup

Generisani skup ćemo dati mreži za prepoznavanje jednog slova koju smo opisali u Sekciji 3.1. Postignuta je tačnost od 98.18%. Matrica konfuzije je prikazana na Slici 3.29, a klasifikacioni izveštaj na Slici 3.30.



Slika 3.29: Matrica konfuzije



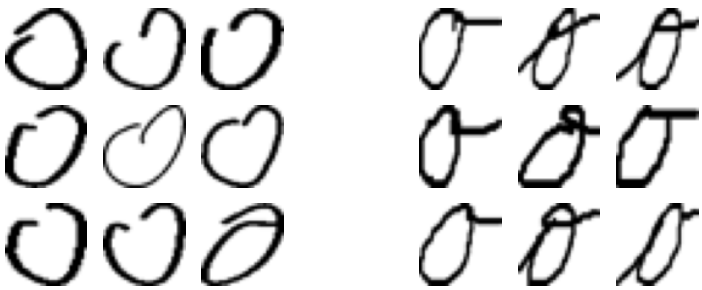
Slika 3.30: Klasifikacioni izveštaj

3.5 Nedostaci

Iako su se konvolutivne mreže pokazale odlično kada je reč o obradi slika, postoje problemi koje ni mi nismo zaobišli u ovom radu, a koji svakako ostavljaju prostor za dalja unapređenja. Naime, mreža je osetljiva na rotacije i skaliranja. Dodavanjem zarotiranih slika u trening skup pokušali smo da ublažimo problem, ali ne i u potpunosti da ga uklonimo. Takođe, možemo očekivati slabije performanse u slučaju slika značajno drugačije rezolucije od onih koje smo koristili pre pretprocesiranja. Napominjemo da se mreži mora dati ulaz fiksne dimenzije, u našem slučaju 28×28, zbog potpuno povezanog sloja koji se nalazi na kraju. Postoje određene tehnike kojima možemo da ublažimo problem fiksne dimenzije ulaza, ali se njima nećemo baviti u radu.

Nažalost, nisu svi nedostaci lako rešivi. Kada govorimo o performansama modela, pretpostavljamo da test podaci, odnosno stvarni podaci na kojima treba da primenjujemo svoj model, dolaze iz iste raspodele kao podaci koje smo koristili za obučavanje modela. Ako to nije slučaj, može doći do značajnih odstupanja od očekivanog ponašanja. Dakle, od modela možemo da očekujemo da dobro radi za instance koje prate skup za treniranje, dok za sve što je na neki nacin „novo”, ne možemo

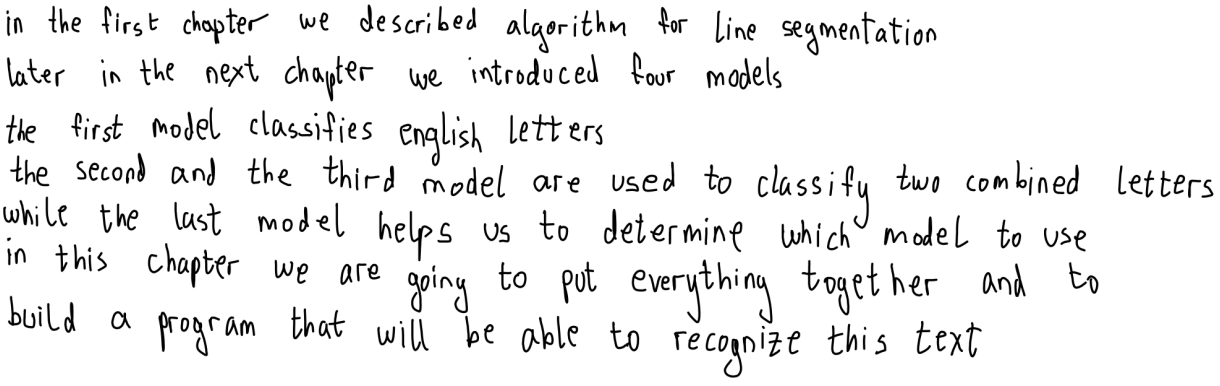
predvideti kako će se mreža ponašati. Slika 3.31 ilustruje primer različitih podataka u trening i test skupovima.



Slika 3.31: Različiti podaci u trening skupu (levo) i test skupu (desno)

Glava 4

Prepoznavanje teksta sa slike

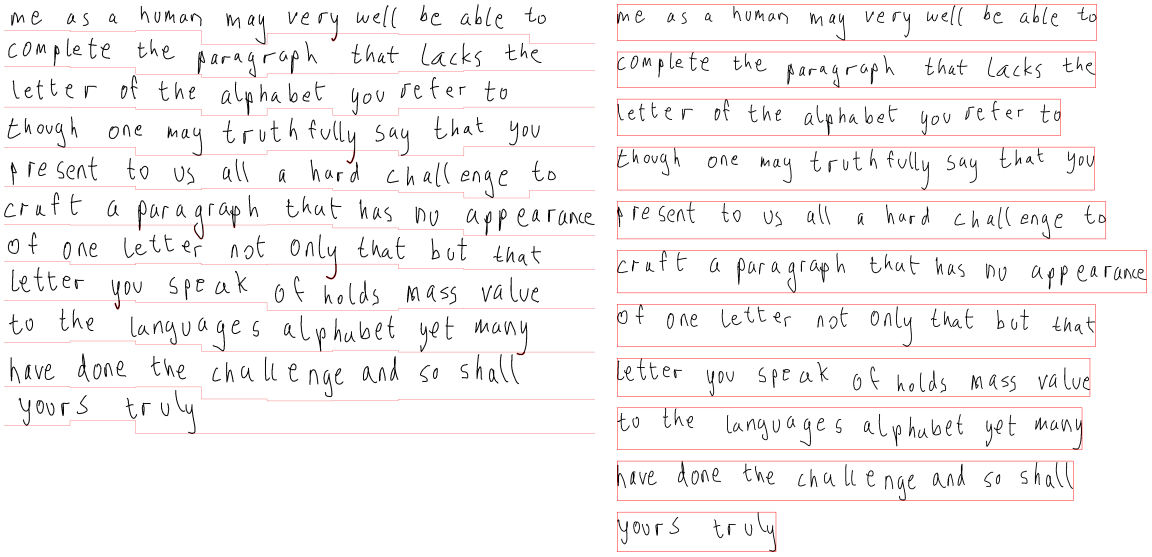


Do sada smo videli kako možemo odraditi segmentaciju linija i prepoznavanje slova, odnosno bigrama. Međutim, da bismo dobili program koji može da prepozna tekst sa slike, potrebno je još dosta toga uraditi. Neophodno je izdvojiti komponente, sa posebnom pažnjom na slova i i j, jer se ona sastoje iz dva dela, odrediti raspored komponenti, razmake u liniji, odlučiti da li koristiti model za prepoznavanje jednog slova ili bigrama i, na kraju, ispraviti što je više moguće grešaka. Iako neki od ovih problema izgledaju trivijalno na prvi pogled, kao na primer odrediti da li je komponenta tačka ili ne, ispostaviće se da nije tako. Za navedene probleme koristili smo heuristike kako bismo ih rešili, tako da ta rešenja ne rade u svim slučajevima, ali smo se trudili da ih optimizujemo tako da pokrijemo što više očekivanih situacija.

U nastavku ćemo prikazati samo najbitnije delove programa, dok se implementacija celog programa može pronaći na GitHub repozitorijumu [4].

4.1 Segmentacija linija

Da bismo dobili linije, koristićemo program opisan u poglavlju 2, uz dve sitne izmene - nećemo koristiti sečenje, već ćemo uvek pokušati da zaobiđemo komponentu i počećemo sa nešto većom širinom trake. Ove izmene pravimo kako bismo prilagodili algoritam za slike koje ćemo mi imati. Slike su visoke rezolucije i pretpostavićemo da se linije neće međusobno dodirivati. Na slikama se nalazi tekst pisan štampanim slovima engleskog alfabeta. Kada dobijemo granice za linije, primenićemo *Flood fill*[1] algoritam da bismo dobili slike koje sadrže linije.



Slika 4.1: Segmentacija linija. Na slici levo crvenom bojom su prikazane granice za linije. Na slici desno prikazane su izdvojene linije dobijene Flood fill algoritmom.

4.2 Izdvajanje komponenti

Da bismo odredili samo povezane komponente, koristićemo BFS. Većina komponenti koje smo dobili su slova (jedno ili dva spojena), ali se može desiti da među njima postoji tačka (ili više njih), deo slova *i*, deo slova *j*, ili bigram koji sadrži neka od ovih slova bez tačke. Najveći izazov predstavlja prepoznavanje tih slučajeva i njihovo grupisanje u *i* i *j*.

[1]Flood fill algoritam se koristi za određivanje svih čvorova za koje postoji put do ulaznog čvora. Put čini niz povezanih čvorova. Čvorovi i način na koji su oni povezani se definiše u samom algoritmu. U našem slučaju, čvorovi su pikseli i dva piksela su povezana ako su iste boje i ako imaju zajedničku stranicu.

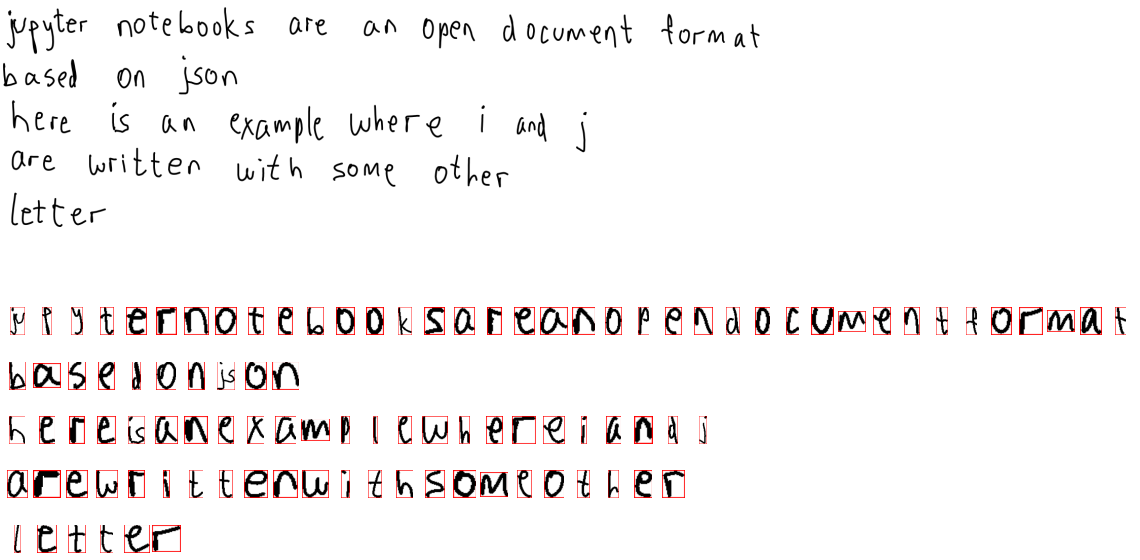
U daljem radu, kada budemo govorili o komponenti (njenoj veličini, površini, donjoj granici, itd.), podrazumevaćemo pravougaonik najmanje veličine koja sadrži tu komponentu.

Sada se postavlja pitanje kako da znamo da li neka komponenta predstavlja tačku ili ne. Možemo probati sledeće:

1. veličina komponente je manja od neke unapred zadate vrednosti, npr. 20×20,
2. veličina komponente je manja od neke vrednosti koja zavisi od prosečne veličine svih komponenti,
3. površina komponente je manja od neke unapred zadate vrednosti,
4. površina komponente je manja od neke vrednosti koja zavisi od prosečne veličine svih komponenti,
5. donja granica komponente je iznad gornje granice neke druge komponente,
6. ispod komponente (i neke njene okoline) postoje neki crni pikseli,
7. središnji pikseli su crni,
8. procenat crnih piksela komponente je veći od neke granice.

Pokazalo se da nam nijedan od gore navedenih uslova ni neka njihova kombinacija nije bila dovoljna da opišemo tačku. Zato, umesto da pokušamo da opišemo tačku, reći ćemo da je slika kandidat za tačku ako je zadovoljen uslov 1. Nakon toga, odredićemo komponentu koja joj je najbliža, koristeći udaljenost samo po x-osi u odnosu na središta komponenti i zatim ćemo spojiti potencijalnu tačku sa tom komponentom. Propustićemo tako dobijenu sliku kroz mrežu koja prepoznaje jedno slovo i kroz mrežu koja prepoznaje bigram. Ako neka od njih „kaže" da je na slici i ili j (u slučaju bigrama je dozvoljeno da bude još neko slovo pre ili posle), smatraćemo da se zaista radilo o tački. Ovaj metod se pokazao kao najbolji jer mreže opisane u poglavlju 3 retko daju lažno pozitivne rezultate u ovom slučaju.

Sada imamo niz komponenti koje idu nekim redosledom, koji često ne prati tekst sa slike, jer smo BFS radili po vrstama odozgo nadole. Potrebno je da ih sortiramo tako da redosled odgovara tekstu sa slike. To ćemo uraditi tako što ćemo ih sortirati u odnosu na središte komponente duž x-ose. Iako se može desiti da ne dobijemo željeni raspored (npr. kukica od j se završila mnogo ispred slova koje prethodi tom j), u praksi se to ipak retko dešava.



Slika 4.2: Komponente. Crveni pravougaonici označavaju komponente.

4.3 Određivanje kraja reči

Sada imamo slova i sledeći korak je da odredimo reči. Napravićemo niz razmaka, gde je razmak udaljenost između desnog kraja leve komponente i levog kraja desne komponente, pri čemu ćemo razmak ograničiti odozdo sa 0 i odozgo sa tri puta prosečnom širinom komponenti. Posmatrajući trenutnu komponentu, želimo da odredimo da li je ona kraj reči. Ako je ona poslednja komponenta u nizu, onda jeste kraj reči. U suprotnom, posmatraćemo trenutni razmak kao razmak između nje i naredne komponente. Ako su ispunjeni sledeći uslovi, reći ćemo da je kraj reči:

1. trenutni razmak je veći ili jednak od prethodnog razmaka uvećanog za 50%, ili narednog razmaka uvećanog za 50%; ako prethodni (naredni) razmak ne postoji, što je slučaj kada je trenutna komponenta prva (pretposlednja) u nizu, smatraćemo da je prethodni (naredni) razmak belina, odnosno, postavićemo ga na veličinu prosečanog razmaka uvećanog za 50%,

2. trenutni razmak je veći ili jednak prosečnom razmaku uvećanom za 50% i

3. trenutni razmak je veći od prosečne širine svih komponenti uvećane za 10%.

Algoritam je testiran nad ukupno 724 krajeva reči (ne računajući trivijalan slučaj kada je u pitanju poslednja reč u liniji) od kojih je pronašao njih 722 i, pritom, nije

označio lažne krajeve. Jedna greška je bila spajanje člana *a* sa rečju *story*, a druga spajanje *i* sa *am*.

Slika 4.3: Rezultat algoritma. Crvenom bojom obojena su slova koja predstavljaju kraj reči.

4.4 Prepoznavanje komponenata

Za kraj nam još ostaje da prepoznamo izdvojene komponente. Ovde ćemo koristiti modele dobijene u poglavlju 3, ali bitnu ulogu će imati i rečnik engleskih reči. Da bismo mogli da iskoristimo rečnik, bilo je neophodno da odredimo krajeve reči, što smo i uradili u prethodnoj sekciji.

Prvo što nas zanima jeste da li je na slici jedno, ili su dva spojena slova. Ako bismo se oslonili samo na model koji klasifikuje instance u ove dve grupe (sekcija 3.3), dešavalo bi se da nekada pogrešno klasifikujemo jedno slovo kao dva. Ovo je vrlo nepoželjno, jer smo pravili modele za prepoznavanje bigrama kako bismo poboljšali tačnost, a ako sada zbog toga pravimo greške na jednom slovu, može se desiti da smo pogoršali celokupnu tačnost. Da bismo ovo izbegli, uradićemo sledeće: za svaku reč napravićemo niz reči kandidata i na kraju ćemo proći kroz niz i pokušati da pronađemo kandidat reč u rečniku. Ako pronađemo reč u rečniku, zaustavljamo se, tj. ne obilazimo niz do kraja. Kandidat reči u nizu su uređene tako da ako se jedna reč nalazi pre neke druge reči u nizu, onda je ona verovatnije reč sa slike. U slučaju da nijedna reč nije pronađena, uzećemo *podrazumevanu* reč. Podrazumevana reč je ona reč čije je svako slovo dobijeno modelom koji prepoznaje jedno slovo.

Pri pravljenju kandidat reči, napravićemo nekoliko optimizacija u cilju poboljšanja krajnjeg rezultata. Prva optimizacija koju ćemo koristiti podrazumeva spajanje trenutne komponente sa prethodnom (sledećom) komponentom u reči, ako takva postoji, i prepoznavanje trenutne komponente koristeći model koji prepoznaje prvo (drugo) slovo bigrama. Nekada bismo slovo, posmatrajući ga kao pojedinačnu komponentu, klasifikovali na jedan način (kao „p” na primer), dok bismo ga drugačije klasifikovali ako poredimo njegovu veličinu i poziciju u odnosu na susedna slova. Ideja je da pokušamo da iskoristimo modele za prepoznavanje bigrama kako bismo otklonili neke greške koje je model za prepoznavanje jednog slova napravio, što je ilustrovano na Slici 4.4.

*Slika 4.4: Bez optimizacije: adve**o**tising, co**v**rts, everythin**y**, se**p**vice. Sa optimizacijom: advertising, courts, everything, service. Model koji prepoznaje pojedinačna slova je slovo „r" u reči „advertising" pogrešno prepoznao kao „o". Nakon spajanja „r" sa „t", model koji prepoznaje prvo slovo bigrama je tačno prepoznao slovo „r". Slično i za slovo „u" u reči „courts". Model koji prepoznaje drugo slovo bigrama je popravio greške u slučaju reči „everything" i „service".*

Za narednu optimizaciju, posmatraćemo kako model radi u praksi, odnosno koja slova često meša i to ćemo probati da ispravimo. U kandidat reči dodaćemo reči koje su dobijene od originalne reči (reč dobijena predikcijom modela bez optimizacija) gde su neka slova koja model često meša zamenjena. Tako, na primer, ako je prediktovano slovo l, a visina slike je manja od prosečne visine komponenti, u niz ćemo dodati reč gde je to l zamenjeno sa e (u treningu smo imali slova l sa petljom, što može ličiti na e) i reč gde je l zamenjeno sa c. Slično radimo i za slova j i i i slova p i e. Važno je napomenuti da se novodobijene reči nalaze u nizu iza reči koja je dobijena isključivo predikcijama modela, te da će one biti razmatrane posle te originalne reči, kao manje verovatne.

Ovim dodavanjem reči, ne samo da pokušavamo da sprečimo ove pojedinačne greške, već rešavamo još jedan problem. Naime, zamislimo da imamo bigram u reči i da smo ga tačno prepoznali, ali da smo pogrešno klasifikovali neko pojedinačno slovo e kao p. U tom slučaju, verovatno ne bismo našli reč sa prepoznatim bigramom u rečniku i koristili bismo podrazumevanu reč. Tako bismo, po svoj prilici, imali tri greške u toj reči - bigram koji smo prepoznali kao jedno slovo (verovatno dve greške) i e za koje smo rekli da je p. Međutim, sa druge strane, ako dodamo kandidat reč u kojoj smo p zamenili sa e i tačno prepoznamo bigram, velika je verovatnoća da nećemo pronaći kandidat reči koje su pre nje u rečniku i da ćemo se, na kraju, odlučiti za ovu ispravnu reč.

Radi boljeg razumevanja gore opisanog algoritma, dajemo njegov pseudokod na Slici 4.5. Rezultat njegovog rada prikazaćemo u narednoj sekciji.

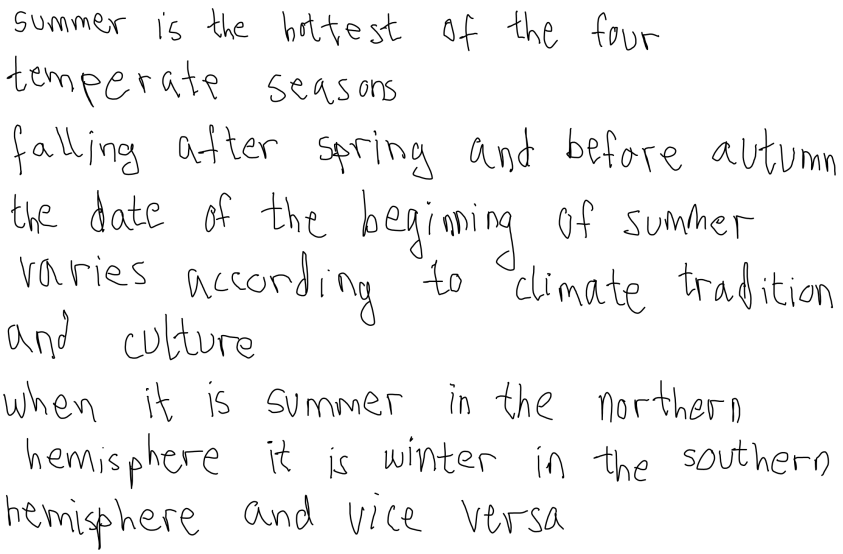
```
kandidat_reci ← [""]
podrazumevana_rec ← ""

za svaku komponentu u liniji:
    podrazumevana_rec ← podrazumevana_rec + slovo(komponenta)
    nove_reci ← []
    za svaku rec u kandidat_reci:
        ako je komponenta bigram
            nova_rec ← rec + bigram(komponenta)
            nove_reci ← nove_reci + (rec + slovo(komponenta))
        inace
            nova_rec ← rec + slovo(komponenta)

        // probaj da spojis tekucu komponentu sa sledecom
        ako nije poslednja komponenta i nije komponenta kraj
           reci:
            tekuca_sledeca_komponenta ← spoji(komponenta,
               sledeca_komponenta)
            nove_reci ← nove_reci + (rec + bigram(
               tekuca_sledeca_komponenta).prvo_slovo)

        // probaj da spojis prethodnu komponentu sa tekucom
        ako nije prva komponenta i nije prethodna komponenta
           kraj reci:
            prethodna_tekuca_komponenta ← spoji(
               prethodna_komponenta, komponenta)
            nove_reci ← nove_reci + (rec + bigram(
               prethodna_tekuca_komponenta).drugo_slovo)

        ako je slovo(komponenta) = 'l' i nije visina
           komponente iznad prosecne visine komponenti:
            nove_reci ← nove_reci + (rec + 'c')
            nove_reci ← nove_reci + (rec + 'e')
        ako je slovo(komponenta) = 'j' i nije visina
           komponente iznad prosecne visine komponenti:
            nove_reci ← nove_reci + (rec + 'i')
        ako je slovo(komponenta) = 'p' i nije visina
           komponente iznad prosecne visine komponenti:
            nove_reci ← nove_reci + (rec + 'e')

        // menjamo rec u nizu, kao je je u pitanju referenca
        rec ← nova_rec

    kandidat_reci ← kandidat_reci + nove_reci

    ako je komponenta kraj reci:
        // prodji kroz kandidat reci i pronadji rec koja se
           nalazi u recniku, a ako takva rec ne postoji
           koristi podrazumevanu rec
        kandidat_reci ← [""]
        podrazumevana_rec ← ""
```

Slika 4.5: Algoritam za prepoznavanje reči

4.5 Demonstracija rada programa

U ovom delu prikazaćemo rad programa kroz nekoliko primera. Koristićemo biblioteku za ispravljanje slovnih grešaka[2] kako bismo dobili preciznije rezultate. Biblioteka nudi model koji je treniran na rečenicama i njihovim delovima. Prosledićemo modelu svaku liniju koju prepoznamo i koristićemo njegov izlaz kao konačan rezultat našeg programa.

Na slici ispod možemo videti primer rukom pisanog teksta koji predstavlja ulaz programa, kao i izlaz koji program daje za taj tekst.

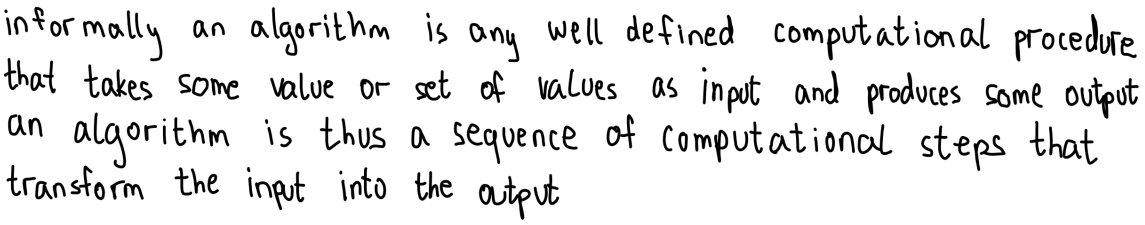



```
summer is the hottest of the four
temperate seasons
falling after spring and before autumn
the date of the beginning of summer
varies according to climate tradition
and culture
when it is summer in the northern
hemisphere it is winter in the southern
hemisphere and vice versa
```

Slika 4.6: Primer rada programa

[2] https://github.com/bakwc/JamSpell

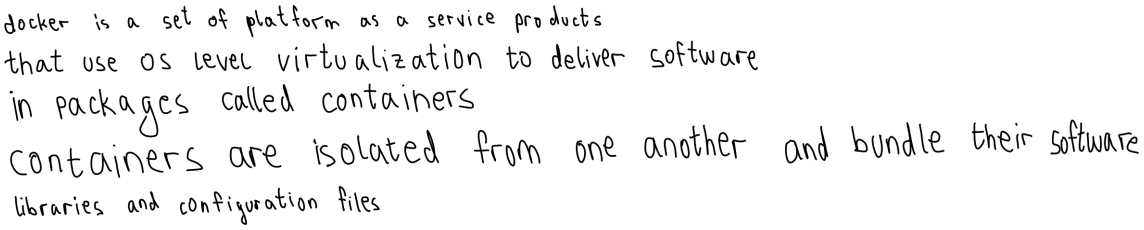
Iako to nije bio slučaj u prethodnom primeru, greške se dešavaju. Na Slici 4.7 dat je primer ulazne slike na kojoj se nalazi tekst koji je potrebno prepoznati. Crvenom bojom obeležene su pogrešno prepoznate reči, dok su labelom iznad strelice označene korekcije dobijene pomoću biblioteke za proveru pravopisa.




```
informally an algorithm is any well defined computational procedure
                                          input
that takes some value or set of values as inpt and produces some output
an algorithm is thus a sequence of computational steps that
transform the input into the output
```

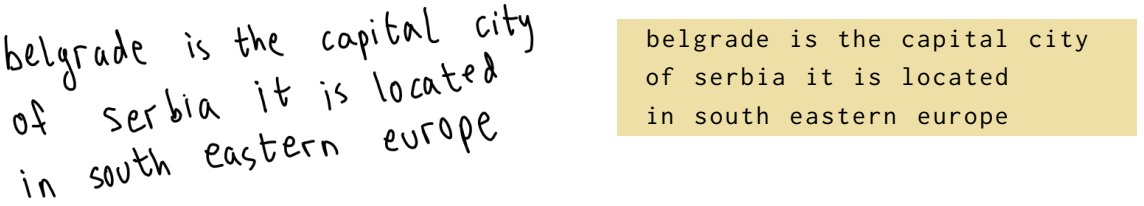
Slika 4.7: Reč „input" je pogrešno prepoznata kao „inpt" jer je bigram „pu" klasifikovan kao „pd". Kako reč „inpdt" nije pronađena u rečniku, koristi se podrazumevana reč, gde je „pu" prepoznato kao jedno slovo „p". Biblioteka za ispravljanje slovnih grešaka je uspešno prepravila „inpt" u „input".

Naravno, može se desiti da model za ispravljanje slovnih grešaka pogreši i ponudi pogrešnu korekciju, ili čak prepravi tačno prepoznatu reč, što ćemo i videti u nastavku. Na Slici 4.8 imamo primer teksta gde se veličina fonta razlikuje od linije do linije. Primećujemo da je pri ispravljanju grešaka napravljena nova greška, tj. da je zamenjena tačno prepoznata reč. To je, svakako, moguće da se desi, ali uglavnom u nekim graničnim slučajevima, kao što su skraćenice.



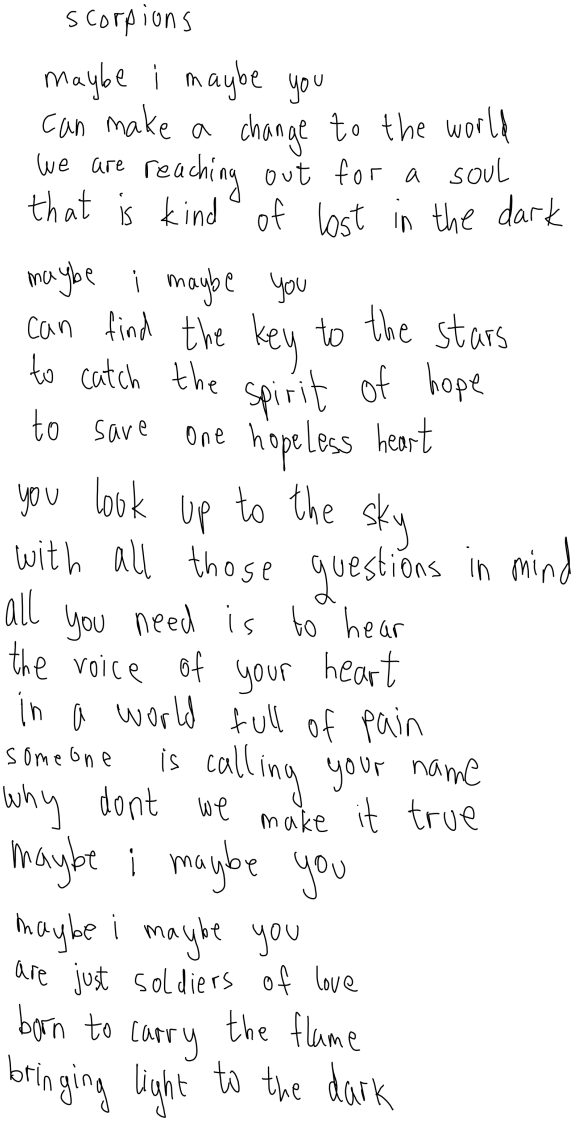
Slika 4.8: Različita veličina slova. Jedna reč je s razlogom ispravljena, dok je reč „os" pogrešno detektovana kao neispravna. Reč „products" je prepoznata kao „produjs", što se objašnjava načinom na koji izdvajamo komponente. Komponenta „c" je dovoljno mala da je razmatramo kao tačku i spojena je sa komponentom „t" kao najbližom. Model za prepoznavanje slova je tako dobijenu novu komponentu prepoznao kao „j" i zato smo odlučili da je „c" tačka.

Takođe, možemo videti i kako se program ponaša u slučaju blago nakrivljenog teksta (Slika 4.9). Zbog načina na koji vršimo segmentaciju linija, program nije otporan na veće nagibe.



Slika 4.9: Tekst ukoso

Prikazujemo još i rad programa u slučaju pesme (Slika 4.10) i nešto dužeg teksta (Slika 4.11).

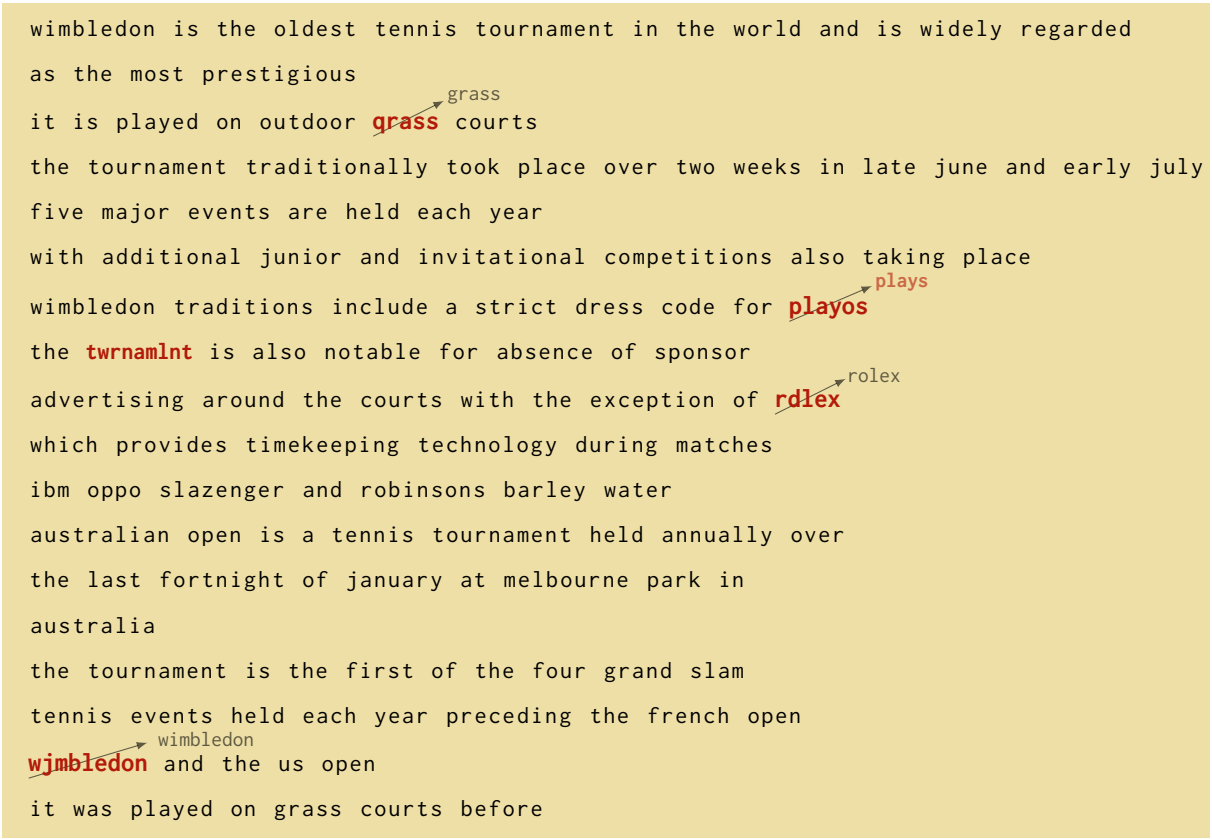



```
scorpions
maybe i maybe you
can make a change to the world
we are reading out for a soul
that is kind of lost in the dark
maybe i maybe you
can find the key to the stars
to catch the spirit of hope
to save one hopeless heart
you look up to the sky
with all those questions in mind
all you need is to hear
the voice of your heart
in a world full of pain
someone is calling your name
why dont we make it true
maybe i maybe you
maybe i maybe you
are just soldiers of love
born to carry the flame
bringing light to the dark
```

Slika 4.10: Scorpions, Maybe I Maybe You

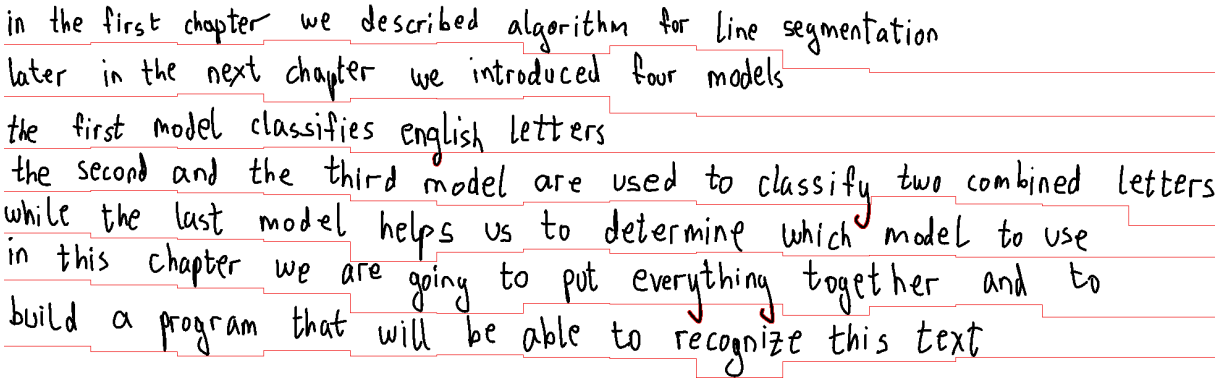
wimbledon is the oldest tennis tournament in the world and is widely regarded as the most prestigious
it is played on outdoor grass courts
the tournament traditionally took place over two weeks in late june and early july
five major events are held each year
with additional junior and invitational competitions also taking place
wimbledon traditions include a strict dress code for players
the tournament is also notable for absense of sponsor advertising around the courts with the exception of rolex which provides timekeeping technology during matches
ibm oppo slazenger and robinsons barley water

australian open is a tennis tournament held annually over the last fortnight of january at melburne park in australia
the tournament is the first of the four grand slam tennis events held each year preceding the french open wimbledon and the us open
it was played on grass courts before

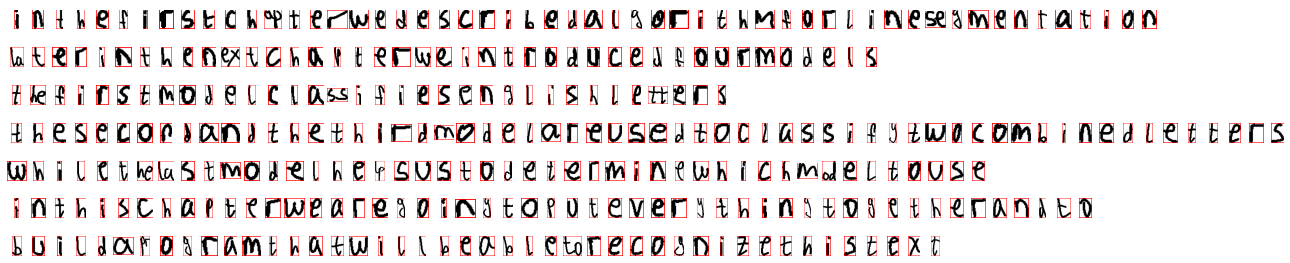


Slika 4.11: Wimbledon & AO

Kao što smo obećali da ćemo napraviti program koji je u stanju da prepozna tekst sa početka poglavlja, prikazujemo rezultat njegovog rada na Slici 4.14.



Slika 4.12: Segmentacija linija



Slika 4.13: Izdvajanje komponenti

```
in the first chapter we described algorithm for line segmentation
later in the next chapter we introduced four models
the first model classifies english letters
the second and the third model are used to classify two combined letters
while the last model helps us to determine which model to use
in this chapter we are goiny to put everything together and to
build a program that will be able to recognize this text
```

going

Slika 4.14: Izlaz programa

4.6 Problemi i moguća unapređenja

U prethodnoj sekciji smo prikazali mogućnosti programa, a sada ćemo skrenutni pažnju na njegove nedostatke. Naime, kao što smo naslutili u sekciji 3.5 kada smo govorili o nedostacima modela za prepoznavanje slova, program koji koristi te modele verovatno neće moći da radi dobro za različite rukopise. Ovo se, nažalost, ispostavilo kao tačno. U slučaju da je rukopis koji treba da prepoznamo značajno različit od onog na kom je model obučavan, program neće biti u stanju da ga uspešno prepozna. Takođe, ovde se još više ističe problem sa bigramima o kome smo već diskutovali - ako je neko slovo u reči pogrešno prepoznato (i nismo uspeli da ga ručno ispravimo), nećemo prepoznati ni bigram u reči. U tom slučaju, već imamo tri greške i biblioteka za proveru pravopisa će teško ispraviti tu reč. Program je osetljiv, ne samo na različit rukopis u smislu stila pisanja pojedinačnih slova, već i na razlike druge vrste, kao što je „podrhtavajući" rukopis. Isto tako, ako je tekst napisan u nekom drugom programu, koji nije Xournal++, ili postoje šumovi, ne možemo očekivati dobre rezultate. Dakle, program radi prihvatljivo u situacijama za koje je obučen, ali ne više od toga.

Važno je još napomenuti da tekst može sadržati i više od dva spojena slova, što nismo u stanju da prepoznamo. Ovaj problem nije lako otkloniti, jer rešenje koje smo mi koristili u ovom radu se ne skalira dobro, ali smo se opredelili za njega kako bismo dobili bolju tačnost u slučaju dva spojena slova.

Pored gore navedenih nedostataka, koji su teže rešivi, postoje i drugi koji su posledica odluke autora da ne implementira podršku za prepoznavanje složenijih tekstova. Naime, modeli nisu u stanju da prepoznaju velika slova, tačke, zareze i ostale specijalne karaktere neophodne za korišćenje programa u praksi. Shodno tome, za neko dalje unapređenje, bilo bi dobro razmišljati u ovom pravcu. Naravno, nije dovoljno samo proširiti modele, imajući u vidu da dodavanje novih karaktera zahteva i doradu segmentacije, koja nije nužno jednostavna jer dovodi do novih problema.

Glava 5

Zaključak

Iz svega do sad prikazanog, možemo zaključiti da je problem prepoznavanja rukom pisanog teksta veoma složen i da zahteva raznovrsne algoritme i heuristike da bi se mogao primeniti u praksi. U ovom radu, bavili smo se ovom zadatkom uz dosta ograničenja zbog težine problema i, pored toga, naišli smo na mnogo problema. Neki od njih su, nažalost, ostali neprevaziđeni, a neke smo, u izvesnoj meri, rešili.

Da bi se napravio upotrebljiv program koji transformiše sliku u tekst, potrebno je sklopiti više celina koje pojedinačno moraju dobro funkcionisati. Tako, na primer, ako segmentacija nije dovoljno dobra, ne možemo očekivati ni uspešno prepoznavanje teksta sa slike, bez obzira na kvalitet modela koji klasifikuju komponente. Takođe, problem predstavlja i činjenica da je svaki od ovih zadataka teško, ili gotovo nemoguće rešiti bez greške, te je potrebno dizajnirati rešenje koje će biti „najkorisnije".

U radu smo se bavili prepoznavanjem rukom pisanog teksta u programu Xournal++ i, pritom, koncentrisali smo se na određeni rukopis, tj. stil. Pod stilom podrazumevamo način na koji su pojedinačna slova napisana, učestalost spajanja dva i više slova, veličinu razmaka između slova i redova teksta, verovatnoću spajanja susednih redova itd. Definisanje stila kojim ćemo se baviti nam je uvelo izvesna ograničenja, ali i pružilo mogućnost za uvođenje određenih pretpostavki koje smo koristili prilikom dizajniranja rešenja. Takođe, oslanjanje na jedan program za pisanje teksta i zanemarivanje šuma može se opravdati činjenicom da je ideja autora od početka bila integracija ovde opisanog programa za prepoznavanje teksta u Xournal++. Međutim, sama integracija prevazilazi opseg ovog rada.

Većina prikazanih ideja može se koristiti za slične probleme sa drugačijim ograničenjima, dok je u nekim slučajevima potrebno razviti nova rešanja prilagođena specifičnostima problema koji se rešava.

Bibliografija

[1] Alireza Alaei, P. Nagabhushan, and Umapada Pal. Piece-wise painting technique for line segmentation of unconstrained handwritten text: A specific study with persian text documents. *Pattern Anal. Appl.*, 14:381–394, 11 2011.

[2] Jelena Mrdak. Letters recognition. `https://github.com/mrdakj/letters`, 2021.

[3] Jelena Mrdak. Line segmentation. `https://github.com/mrdakj/line_segmentation`, 2021.

[4] Jelena Mrdak. Text recognition. `https://github.com/mrdakj/master_text_recognition`, 2021.