

Implementacija konvolucijskih neuronskih mreža u optičkom prepoznavanju znakova

Strunje, Ana

Undergraduate thesis / Završni rad (sveučilišni)

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split, Faculty of Science / Sveučilište u Splitu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:166:806248>

Rights / Prava: [Zaštićeno autorskim pravom.](#) / [In copyright.](#)

Download date / Datum preuzimanja: **2026-09-07**



Repository / Repozitorij:

[Digital Repository of Faculty of Science in Split / Digitalni repozitorij Prirodoslovno-matematičkog fakulteta u Splitu](#)



Dabar

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

**IMPLEMENTACIJA KONVOLUCIJSKIH
NEURONSKIH MREŽA U OPTIČKOM
PREPOZNAVANJU ZNAKOVA**

Ana Strunje

Split, rujan 2023.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za Informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

IMPLEMENTACIJA KONVOLUCIJSKIH NEURONSKIH MREŽA U OPTIČKOM PREPOZNAVANJU ZNAKOVA

Ana Strunje

SAŽETAK

Optičko prepoznavanje znakova (engl. *Optical Character Recognition*, OCR) je tokom desetljeća značajno napredovalo. U radu se opisuje kako je OCR nastao, gdje se primjenjuje, te koje su glavne faze izrade. Umjetna neuronska mreža, tip strojnog učenja, može se implementirati u OCR sustav u fazi klasifikacije za prepoznavanje znakova u dokumentu. Za primjenu teorije rada izrađena je mobilna aplikacija za učitavanje slike Sudoku igre i prepoznavanje brojki putem OCR sustava s konvolucijskom neuronskom mrežom.

Ključne riječi: optičko prepoznavanje znakova, umjetne neuronske mreže, konvolucijske neuronske mreže, strojno učenje, duboko učenje, Expo, Flask, OpenCV, TensorFlow

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 41 stranica, 33 grafička prikaza, 0 tablica i 17 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Doc. dr. sc. Goran Zaharija**, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **Doc. dr. sc. Goran Zaharija**, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Doc. dr. sc. Monika Mladenović, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Dino Nejašmić, *predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: **rujan 2023.**

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Informatics
Ruđera Boškovića 33, 21000 Split, Croatia

CONVOLUTIONAL NEURAL NETWORKS IN OPTICAL CHARACTER RECOGNITION

Ana Strunje

ABSTRACT

Optical Character Recognition (OCR) has significantly advanced over the decades. The thesis describes how OCR originated, where it is applied, and its' main phases of production. An Artificial Neural Network (ANN), a type of machine learning, can be implemented in an OCR system at the classification phase for character recognition in documents. To apply the theory of the paper, a mobile application was developed for capturing an image of a Sudoku game and recognizing numbers using an OCR system with a convolutional neural network.

Key words: Optical Character Recognition, Artificial Neural Networks, Convolutional Neural Networks, Machine Learning, Deep Learning, Expo, Flask, OpenCV, TensorFlow

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 41 pages, 33 figures, 0 tables and 17 references

Original language: Croatian

Mentor: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Reviewers: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Monika Mladenović, Ph.D. *Assistant Professor of Faculty of Science, University of Split*

Dino Nejašmić, Ph.D. *Lecturer of Faculty of Science, University of Split*

Thesis accepted: **September 2023.**

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam ravršni rad s naslovom IMPLEMENTACIJA KONVOLUCIJSKIH NEURONSKIH MREŽA U OPTIČKOM PREPOZNAVANJU ZNAKOVA izradio/la samostalno pod voditeljstvom doc.dr.sc. Goran Zaharija. U radu sam primijenio/la metodologiju znanstvenoistraživačkog rada i koristio/la literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući naveo/la u diplomskom radu na uobičajen, standardan način citirao/la sam i povezao/la s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Student/ica

Ana Strunje

Sadržaj

UVOD	1
1. OPTIČKO PREPOZNAVANJE ZNAKOVA (OCR).....	3
1.1. POVIJEST OCR-A	3
1.2. PRIMJENA OCR SUSTAVA	4
1.3. GLAVNE FAZE OCR-A.....	4
1.3.1. Pretprocesiranje	5
1.3.2. Segmentacija.....	5
1.3.3. Normalizacija.....	5
1.3.4. Ekstrakcija značajki	6
1.3.5. Klasifikacija.....	6
1.3.6. Postprocesiranje.....	6
2. NEURONSKE MREŽE	7
2.1. PRIMJENA UMJETNIH NEURONSKIH MREŽA	7
2.2. PODJELA UMJETNIH NEURONSKIH MREŽA	8
2.3. DUBOKO UČENJE.....	8
2.3.1. Konvolucijske neuronske mreže.....	9
3. IMPLEMENTACIJA	13
3.1. SUČELJE (FRONTEND).....	13
3.1.1. Korišteni alati	13
3.1.2. Arhitektura koda	14
3.2. OCR SUSTAV (BACKEND)	19
3.2.1. Korišteni alati	19
3.2.2. Arhitektura koda	20
3.3. POKRETANJE I IZGLED APLIKACIJE	34
ZAKLJUČAK.....	37
TABLICA SLIKA	38
LITERATURA	40

Uvod

U današnje doba, digitalna interpretacija informacija tekstualnog oblika iz različitih izvora postala je vrlo bitna za različite primjene, od digitalizacije dokumenata do pretraživača temeljenih na tekstu. Razvoj tehnologije zadnjih desetljeća doveo je i do razvoja alata i tehnika za automatizaciju procesa interpretacije, to jest alata za optičko prepoznavanje znakova (engl. *Optical Character Recognition*, OCR). OCR je područje umjetne inteligencije koje uključuje prepoznavanje znakova, slova i simbola iz različitih izvora, poput slika, bilješki, ili skeniranih dokumenata. Pojavio se zbog povećane uporabe automatizacije u modernom digitalnom svijetu te omogućuje brže procesiranje velike količine podataka. OCR zapravo omogućuje integraciju fizičkog teksta u digitalni svijet ponašajući se kao most između analognog i digitalnog, što daje znatan potencijal za tvrtke, istraživače i pojedince u modernizaciji obrade informacija.

U svijetu OCR-a, čiji je glavni aspekt transformacija fizičkog teksta u digitalni, umjetne neuronske mreže (engl. *Artificial Neural Networks*, ANNs) pokazale su se prikladnim načinom prepoznavanja i kategorizacije znakova. Umjetne neuronske mreže vrsta su strojnog učenja inspirirana načinom rada ljudskoga mozga. Posjeduju sposobnost učenja i prilagodbe, što ih čini odličnim izborom za kompleksno prepoznavanje znakova. Duboko učenje, pod koje spadaju konvolucijske neuronske mreže, vrsta su strojnog učenja te su postali popularan izbor za izgradnju alata računalnog vida i obrade prirodnoga jezika. Konvolucijske neuronske mreže (engl. *Convolutional Neural Networks*, CNNs) koriste se u izgradnji OCR sustava radi njihovih sposobnosti prepoznavanja složenih uzoraka iz vizualnih podataka. Razvile su se po inspiraciji na vizualni korteks ljudskoga mozga te funkcioniraju kao digitalne oči koje „čitaju“ znakove sa znatnom točnošću.

Ovaj rad podublje će objasniti što je to OCR, kako i zašto je nastao, te kako se zapravo postiže prepoznavanje znakova. Također će se objasniti umjetne neuronske mreže, i specifično konvolucijske neuronske mreže, u kojim područjima se koriste te koji su glavni elementi obične konvolucijske neuronske mreže. Naposljetku, kao primjena dviju tema, objasnit će se postupak izrade aplikacije koja koristi konvolucijsku neuronsku mrežu za izgradnju OCR sustava. Specifično, izgradit će se mobilna aplikacija za prepoznavanje znakova unutar Sudoku mreže. Sudoku je svjetski poznata logička igra koja se igra na mreži od 9 redaka i 9 stupaca, čiju 81 ćeliju treba popuniti brojevima u rasponu od 1 do 9 logičkom dedukcijom. Tradicionalno, Sudoku igre rješavale su se u tiskanom obliku, bilo to u novinama, časopisima, i sl. Međutim,

kako digitalna tehnologija napreduje i integrirana je u svakodnevni život, raste potražnja za Sudoku u elektroničnim formatima, kao što su internetske igre ili mobilne aplikacije.

1. Optičko prepoznavanje znakova (OCR)

Optičko prepoznavanje znakova, poznato po kratlici OCR (engl. *Optical Character Recognition*), tehnika je skeniranja i usporedbe namijenjena za identifikaciju tiskanog teksta ili brojčanih podataka. Sustav za OCR pokušava identificirati znakove uspoređujući njihove oblike s onima pohranjenim u bazu sustava. [1] Naravno, što je podatkovna baza u kojoj su pohranjeni znakovi za usporedbu veća, to je sustav više prilagodljiv odnosno daje točnije rezultate jer ima velik broj različitih situacija. Na primjer, ako baza znakova sadrži samo primjere tiskanoga teksta, teže će se snaći u prepoznavanju ručno pisanoga teksta. Isto tako, kvaliteta same slike utječe na točnost prepoznavanja. OCR tehnologija pojavila se zbog sve većeg zahtjeva digitalnih biblioteka, poštanskih i bankovnih usluga, te mnogih automatiziranih sustava procesiranja dokumenata.

1.1. Povijest OCR-a

Prvi patent OCR sustava datira 1929. godine kada je Gustav Tauschek, austrijski inženjer, osmislio uređaj za optičko prepoznavanje, kojeg je kasnije patentirao u SAD-u 1935. godine. Uređaj koristi kružni disk s izrezanom šablonom znakova da svjetlo može proći kroz otvore. Slika znakova se postavi ispred navedenog diska te se svjetlo reflektira kroz sliku i fokusira kroz otvor, nakon čega fotosenzor prepoznaje to svjetlo. [2]

Komercijalno dostupni OCR sustavi mogu se klasificirati u 4 generacije, temeljeno na njihovoj efikasnosti, robusnosti i prilagodljivosti. Sustavi prve generacije OCR-a, korišteni u ranim 1960-ima, mogli su raspoznati samo određene fontove i oblike znakova. Najpoznatiji sustav prve generacije je IBM 1418, koji se koristio tehnikom logičkog podudaranja šablone. U drugoj generaciji sustavi su bili napredniji, tj. mogli su prepoznati i tiskane i ručno pisane znakove, no bili su limitirani samo na prepoznavanje brojki. Bili su dostupni sredinom 1960-ih i u ranim 1970-ima, a prvi uređaj druge generacije je IBM 1287 koji je kombinirao digitalnu i analognu tehnologiju. Tokom treće generacije razvijalo se raspoznavanje ručno pisanih znakova i tiskanih znakova s lošijom kvalitetom, te su takvi sustavi korišteni u periodu od 1975. do 1985. godine. U četvrtoj generaciji OCR sustavi bili su sposobni skenirati i prepoznati znakove iz kompleksnih dokumenata koji sadržavaju tekst, tablice i matematičke simbole. Također su bili sposobni prepoznati znakove s dokumenata lošije kvalitete, npr. faks i fotokopije. [2]

1.2. Primjena OCR sustava

Sustavi za optičko prepoznavanje znakova danas se koriste u raznim komercijalnim primjenama. Chaudhuri i ostali navode tri glavne upotrebe OCR sustava: unos podataka (engl. *data entry area*), unos teksta (engl. *text entry*), te automatizacija procesa (engl. *process automation*). [3]

Unos podataka pokriva sustave za unos velike količine određenih podataka. Prvotno su takve uređaje koristile banke. Karakterizira ih čitanje samo određenih tiskanih znakova, obično brojki i specifičnih simbola. Dizajnirani su za čitanje korisničkih brojeva, brojeva artikala, količinu novca i dr. Iako prepoznaju samo određene znakove na specifičnim formatima papira, tolerantni su na lošu kvalitetu tiska. Zbog toga što su trenirani samo na određene znakove, postotak greški im je oko 0,01%.

Uređaji za unos teksta koriste se kao čitači velike količine teksta u uredskoj automatizaciji. Nisu ograničeni na određene znakove i formate kao uređaji za unos podataka, no zato su osjetljiviji na različite fontove i kvalitetu tiska. Postotak greški za ovakve uređaje je oko 0,1%.

Automatizacija procesa fokusira se na upravljanje određenog procesa, odnosno na automatsko čitanje adrese u svrhu razvrstavanja pošte. Funkcionira na način da pročita sve dostupne informacije i koristi poštanski broj za provjeru istih. Što se može i zaključiti, točnost prepoznavanja ručno pisanih adresa varira za razliku od tiskanih, no sustav odbaci adrese koje ne može iščitati pa je postotak greške blizu nuli.

Postoje i druge primjene OCR sustava, kao što su pomoć slijepima, automatsko čitanje registracijskih tablica, automatska kartografija, čitači obrazaca, verifikacija potpisa, i sl. [3]

1.3. Glavne faze OCR-a

Prema autorima Karić i ostalima, sustav za OCR obično se sastoji od pet procesa: dohvaćanja slike, pretprocesiranja, segmentacije, ekstrakcije značajki, te klasifikacije. [4] Islam et al. navode i šesti proces – postprocesiranje. [5] Hamad i Kaya također spominju šest faza opričkog prepoznavanja: pretprocesiranje, segmentaciju, normalizaciju, ekstrakciju značajki, klasifikaciju, te postprocesiranje. [6] U ovom radu opisat će se sistem podjele faza OCR-a prema Hamad i Kaya.

1.3.1. Pretprocesiranje

Cilj pretprocesiranja je poboljšanje kvalitete slike, odnosno eliminiranje neželjenih karakteristika na slici bez gubljenja značajnih informacija. Proces pretprocesiranja potreban je za slike u boji, sivim tonovima ili binarnim slikama koje sadrže tekst i/ili grafiku. Budući da je obrada slika u boji računalno zahtjevnija, većina sustava za prepoznavanje znakova koristi binarne ili sive slike. Pretprocesiranje poboljšava sliku i priprema je za sljedeće faze optičkog prepoznavanja znakova. Da bi procesiranje slike u sljedećim fazama bilo lakše i efektivnije, u ovoj fazi se slika preoblikuje u pogodan format.

Jedne od glavnih operacija u fazi pretprocesiranja su binarizacija, redukcija šuma, ispravljanje kosine i rotacije, morfološke operacije, određivanje praga (engl. *thresholding*), zaglađivanje, kompresija, te istanjivanje. Pomoću binarizacije se razdvajaju pikseli slike kao tekst ili pozadina. Smanjenje šuma, kako i sami naziv govori, pomoću smanjena šuma zaglađuje komponente slike, popunjava rupe ili uklanja nepravilnosti na slikama. Ispravljanje kosine i rotacije je potrebno jer npr. klijent može fotografirati papir s tekstom pod određenim kutom ili zarotirati isti papir, što otežava uređaju prepoznavanje. Pomoću morfoloških operacija se dodaju ili uklanjaju pikseli na znakovima koji imaju viška ili manjka piksela. *Thresholding* ili određivanje praga odvaja informaciju od njene pozadine. [6]

1.3.2. Segmentacija

U fazi segmentacije slika se razdvaja u znakove prije klasifikacije. Segmentacija se može izvesti eksplicitno ili implicitno, te ostale faze OCR-a mogu dati kontekst koristan za provođenje segmentacije slike. [5] Prema Hamad i Kaya, segmentacija je jedna od glavnih komponenti OCR sustava koja razdvaja linije teksta od slike. Segmentacija teksta od slike obično uključuje segmentaciju linije, segmentaciju riječi, pa segmentaciju znaka. U ovom procesu se slika klasificira u homogene zone, odnosno svaka zona sadrži samo jednu vrstu informacije, kao što su tekst, lik, ili tablica. U mnogim slučajevima stopa točnosti OCR sustava direktno ovisi o točnosti algoritma za segmentaciju. [6]

1.3.3. Normalizacija

Rezultat procesa segmentacije je matrica informacija koju treba normalizirati da bi olakšala rad sljedeće faze, ekstrakcije značajki. Matrice se normaliziraju tako da im se smanjuje veličina i eliminiraju se nepotrebne informacije bez gubitka bitnih. [6]

1.3.4. Ekstrakcija značajki

U fazi ekstrakcije značajki izdvajaju se značajke koje čine pojedini znak jedinstvenim. [5] Postoje dvije glavne klase značajki: statističke značajke i strukturalne značajke. U matrici znakova statističke se značajke dobivaju pomoću statističke distribucije svake točke pomoću zoniranja, momenata, preklapanja, Fourierove transformacije, te histograma projekcije. Statističke značajke se nazivaju i globalnim značajkama jer se obično ekstrahiraju iz podslika (npr. mreža). Strukturalne ili topološke značajke vezane su uz geometriju znakova. Pod geometriju spadaju linije znakova, konveksnost/konkavnost, broj rupa, broj krajeva, i sl. [6]

1.3.5. Klasifikacija

Klasifikacija je proces svrstavanja znakova u prikladne kategorije. [5] U ovoj fazi se distribuiraju znakovi u određene klase u svrhu kreiranja grupa s homogenim osobinama, odnosno odvajanja različitih znakova. [6] Islam et al. dijele principe klasifikacije na strukturalni i statistički. Strukturalni princip klasifikacije temelji se na vezama među komponentima slike. Statistički princip koristi diskriminantnu funkciju za klasifikaciju slike. Neki od statističkih načina klasifikacije su Bayesov klasifikator, te klasifikatori po principu stabla odlučivanja (engl. *decision tree classifier*), neuronskih mreža, najbližeg susjeda, itd. Postoji i sintaktički princip klasifikacije koji pretpostavlja gramatički pristup značajki. [5] Prema Hamad i Kaya, principi klasifikacije dijele se na statističke principe, neuronske mreže, podudaranje predloška, algoritme uređaja potpornih vektora (engl. *Support Vector Machine*, SVM), i klasifikator kombinacija. [6]

1.3.6. Postprocesiranje

Zadnja faza, postprocesiranje, bazira se na povećanju točnosti ishoda OCR-a te postoje razni pristupi. Jedno od pristupa jest korištenje više od jednog načina klasifikacije slike. Također, može se obraditi i kontekstualna analiza, u kojoj kontekst dokumenta može smanjiti mogućnost grešaka. [5] Iako pretprocesiranje pokušava „očistiti“ nepotrebne informacije, isto tako može ukloniti bitne podatke jer kontekst slike nije dostupan u toj fazi. Pokazalo se da ljudi točno mogu pročitati ručno pisan tekst po kontekstu do 60%. Povećanje točnosti OCR sustava može se ostvariti davanjem semantičkog konteksta, te korištenjem rječnika za smanjivanje grešaka u pravopisu. [6]

2. Neuronske mreže

U posljednje vrijeme umjetne neuronske mreže postale su popularan model za klasifikaciju, grupiranje, prepoznavanje uzoraka i predikciju u raznim disciplinama. Danas se neuronske mreže najčešće koriste za univerzalnu aproksimaciju funkcija i numeričke paradigme zbog njihovih mogućnosti samoučenja, prilagodljivosti, tolerancije grešaka i nelinearnosti. [7]

Umjetne neuronske mreže vrsta su strojnog učenja (engl. *Machine Learning*) i postale su konkurencija uobičajenoj regresiji i statističkim modelima što se tiče korisnosti. [7] U osnovi su paralelni računalni modeli koji imitiraju rad ljudskoga mozga. Sastoje se od velikog broja jednostavnih procesora povezanih opterećenim vezama. Svaki izlaz procesorskog čvora ovisi o informaciji koja je lokalno dostupna čvoru, bilo da je pohranjena unutar čvora ili dostupna preko opterećenih veza. Čvorovi dobivaju ulaz od drugih čvorova, te svoj izlaz šalju dalje drugim čvorovima. [8]

Prednost neuronskih mreža je što se modeli mogu lako koristiti i točniji su kada su u pitanju kompleksni sustavi s velikim brojem ulaza. Neuronske mreže prihvaćene su kao koristan model strojnoga učenja i zbog sposobnosti rješavanja problema. [7]

2.1. Primjena umjetnih neuronskih mreža

Umjetne neuronske mreže koriste se u raznim područjima rada: [7]

- sigurnost,
- znanost,
- zdravstvo,
- bankarstvo,
- školstvo,
- meteorologija,
- i sl.

Također se koriste za optimizaciju i klasifikaciju podataka. Neuronske mreže su isto odlične za identifikaciju trendova i uzoraka u podacima. Mogu predvidjeti financijsku stabilnost, što je korisno za analizu monetarne vrijednosti u ekonomiji. Isto tako mogu uspješno predvidjeti uspjeh, odnosno neuspjeh banki i procijeniti tržište dionica. Neuronske mreže se koriste se i za predviđanje vremenske prognoze. [7] Sigurnost plaćanja kreditnim karticama

značajno se poboljšala primjenom neuronskih mreža. Utvrđivanje zlouporabe kreditne kartice prosječno je trajalo 60 do 90 dana. Neuronske mreže mogu prepoznati ekstremna odstupanja te smanje utvrđivanje na samo nekoliko dana, što je moguće sustavom koji pregledava uzorke kupovine vlasnika kreditne kartice. [9]

2.2. Podjela umjetnih neuronskih mreža

Postoji velik broj različitih vrsta umjetnih neuronskih mreža, stoga ih nije jednostavno klasificirati. Kada gledamo strukturu, neuronske mreže dijele se na statičke unaprijedne (engl. *feedforward*) i dinamičke ili povratne (engl. *feedback*). [10] Statički unaprijedne neuronske mreže prate samo jedan smjer i put, tj. rezultat će uvijek putovati od ulaza do izlaza. Najčešće se koriste u prepoznavanju uzoraka. Svi izlazi se procjenjuju i tek onda šalju u sljedeći sloj neurona, i tako dalje. S druge strane, statički dinamičke neuronske mreže ne prate samo jedan put slanja signala, tj. signali mogu putovati u oba smjera. Postižu dinamičnost jer konstantno uspoređuju stavke i ulaze/izlaze te sukladno time mijenjaju svoj proces. [11]

Za procesiranje znatnog broja podataka razvijaju se neuronske mreže koje imaju stotine slojeva, što se danas naziva duboko učenje (engl. *Deep Learning, DL*). [10]

2.3. Duboko učenje

Umjetna inteligencija je zadnjih desetljeća znatno napredovala, stoga je postala jako široko polje računarstva. Unutar umjetne inteligencije spada i strojno učenje, a unutar strojnog učenja spada duboko učenje. Duboko učenje su umjetne neuronske mreže s kompleksnim slojevima. [7] Prema Kelleher, duboko učenje je potpolje umjetne inteligencije koje se fokusira na stvaranje modela neuronskih mreža znatnih veličina sposobnih stvaranju točnih odluka vođenih podacima. Prikladno je za obrađivanje kompleksnih podataka i baza s velikom količinom podataka. [12] Metode dubokog učenja postale su popularan izbor za izgradnju sustava prepoznavanja glasa, prepoznavanja uzoraka, računalnoga vida, te obrade prirodnoga jezika. [7]



Slika 1 Odnos umjetne inteligencije, strojnog učenja, i dubokog učenja. Duboko učenje je potpolje strojnog učenja, a strojno učenje potpolje umjetne inteligencije.

Razlika između neuronskih mreža dubokog učenja i neuronskih mreža kao što su statičke unaprijedne i dinamične je u njihovim karakteristikama. Duboko učenje koristi kompleksnije načine spajanja slojeva te ima veći broj neurona. Dakle, duboko učenje je definirano kao neuronska mreža s raznovrsnim varijablama i slojevima temeljenim na specifičnoj arhitekturi mreže prethodno treniranih mreža bez nadzora, konvolucijskih neuronskih mreža (engl. *Convolutional Neural Network*, CNN), rekurzivnih neuronskih mreža (engl. *Recursive Neural Network*, RvNN), i rekurentnih neuronskih mreža (engl. *Recurrent Neural Network*, RNN). [7]

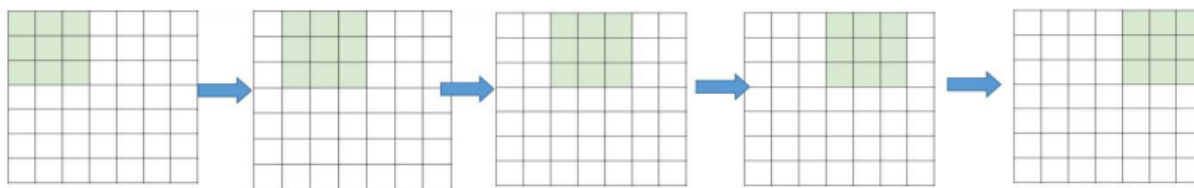
2.3.1. Konvolucijske neuronske mreže

U posljednje vrijeme konvolucijske neuronske mreže pokazale su se izvrsnim alatom za prepoznavanje uzoraka. Jedan od najznačajnijih aspekata konvolucijskih neuronskih mreža je što smanjuju broj parametara u umjetnim neuronskim mrežama. Zbog smanjenja broja parametara moguće je lakše obraditi velik broj podataka za rješavanje kompleksnih zadataka, što nije moguće ostvariti s klasičnim neuronskim mrežama. [13]

2.3.1.1. Elementi konvolucijske neuronske mreže

Prema Daboviću i Tartalji, konvolucijske neuronske mreže sastoje se od sljedećih glavnih elemenata: [14]

- 1) Ulazna zapremnina (engl. *input volume*): Konvolucijske neuronske mreže najčešće koriste slike kao ulazni podatak. Te slike su zapravo matrice piksela koje imaju svoju širinu, visinu, i vrijednosti piksela u rasponu 0-255.
- 2) Karakteristike (engl. *feature*): Putem ulaznog sloja unosi se slika koja sadrži zasebne karakteristike. Ono što konvolucijske neuronske mreže pregledavaju jesu same karakteristike ulaznih slika pomoću kojih se neuronska mreža uvježbava.
- 3) Filteri: Naziva se još i konvolucijska jezgra (engl. *kernel*). Konvolucijski filter u jednostavnim konvolucijskim neuronskim mrežama je generalizirani linearni model (engl. *Generalized Linear Model*, GLM) koji se primjenjuje na regiju slike. Filter se predstavlja dvodimenzionalnom matricom malih dimenzija i sastoji se od realnih vrijednosti. Odnose se na konvolucijski sloj arhitekture gdje se filteri primjenjuju na sliku. Sam postupak primjene filtera na sliku naziva se konvolucija.
- 4) Receptivno polje: Osnovna ideja receptivnih polja je simulacija načina funkcioniranja vizualnog korteksa životinja, gdje su ćelije osjetljive na male subregije vidnog polja poznate kao receptivna polja. Umjesto da svaki neuron bude povezan sa svim neuronima u susjednom sloju kao u tradicionalnim MLP (engl. *Multilayer Perceptron*), u konvolucijskim neuronskim mrežama neuroni se raspoređuju u trodimenzionalnu strukturu i povezuju se s određenim receptivnim poljem. Ovaj pristup smanjuje kompleksnost povezivanja i izračunavanja tako da se povezuje samo na dio slike koji je vidljiv filteru u određenom trenutku.
- 5) Korak (engl. *stride*) i dopuna (engl. *padding*): Korak se odnosi na korak pomicanja filtera preko ulazne slike. Kako bi se izvukle sve karakteristike, uključujući i karakteristike niske razine (engl. *low-level features*), cilj je da se u konvolucijskoj neuronskoj mreži očuva što više informacija. Stoga, uvodi se dopuna. Na primjer, ako je ulazna slika dimenzija 7x7, filter dimenzija 3x3, a korak 1, izlaz će biti slika dimenzija 5x5. Najčešća vrijednost piksela kojima se slika nadopunjuje je 0, odakle dolazi pojam dopune nulama (engl. *zero-padding*), jedna od najčešćih načina dopune. Ako se primijeni dopuna nulama za jedan sloj, izlazna slika dobiva dimenzije 7x7, što je jednako dimenziji ulazne slike.



Slika 2 Prikaz pomicanja filtera dimenzija 3x3 s korakom 1 [13]

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Slika 3 Dopuna nulama na jednom sloju [13]

- 6) Sažimanje maksimumom (engl. *max pooling*): Sažimanje maksimumom vrsta je nelinearnog smanjivanja broja piksela (engl. *down-sampling*). Sloj sažimanja (engl. *pooling layer*) koristi se za smanjenje veličine slike, odnosno broja karakteristika, što pruža jednostavnije računanje. Kod sažimanja bira se piksel s određenom vrijednošću u određenom polju. Metoda sažimanja maksimumom najčešće je korištena metoda sažimanja koja uzima piksel s najvećom vrijednošću. Korisna je za uklanjanje nemaksimalnih vrijednosti, što pojednostavljuje računanje, te pruža nepromjenjivost translacije (engl. *translation invariance*), odnosno prepoznavanje objekta ne zavisi o transformaciji translacije objekta unutar slike. Pored sažimanja maksimumom postoje i: prosječno sažimanje (engl. *average pooling*), mješovito sažimanje (engl. *mixed pooling*), i dr.

12	20	30	0	2×2 Max-Pool	20	30
8	12	2	0		112	37
34	70	37	4			
112	100	25	12			

Slika 4 Primjer sažimanja maksimumom na matrici dimenzija 4x4 s filterom 2x2 i korakom 2. Sažimanjem se dobiva matrica dimenzija 2x2 s maksimalnim vrijednostima svakog polja. [15]

- 7) Hiperparametri (engl. *hyperparameters*): Hiperparametri čine arhitekturu konvolucijske neuronske mreže jedinstvenom. Glavni hiperparametri su receptivno polje, dopuna

nulama, dimenzije ulazne zapremnine, te korak. Arhitektura konvolucijske mreže ovisi o izboru navedenih hiperparametara. Parametri koji utječu na veličinu izlazne zapremnine su dubina, korak i dopuna nulama.

2.3.1.2. Arhitektura konvolucijske neuronske mreže

Dabović i Tartalja spominju pet glavnih komponente arhitekture osnovne konvolucijske neuronske mreže: 1) ulazni sloj, 2) konvolucijski sloj, 3) sloj sažimanja, 4) sloj aktivacijske funkcije, i 5) potpuno povezani (engl. *Fully Connected*, FC) sloj. [14]

- 1) Ulazni sloj primjenjiv je za svaku neuronsku mrežu. Njime se podaci unose u mrežu, a u konvolucijskim neuronskim mrežama podaci su najčešće slike. Ulazna zapremnina direktno je povezana s ulaznim slojem.
- 2) Konvolucijski sloj realizira osnovnu operaciju učenja neurona mreže. [14] Uzima sliku iz ulaznog sloja te ju konvulira s filterima kako bi kreirao izlaznu sliku. [16]
- 3) Sloj sažimanja se periodično ubacuje između sukcesivnih konvolucijskih slojeva. U njemu se odrađuje operacija sažimanja, koja je objašnjena u 2.3.1.1. Smanjenjem broja parametara može se kontrolirati i pretreniranost (engl. *overfitting*) mreže.
- 4) U sloju aktivacijske funkcije bira se aktivacijska funkcija koja može značajno poboljšati performanse mreže. Jedna od najčešće korištenih aktivacijskih funkcija je ReLU (engl. *Rectified Linear Unit*). [14] Njezina uloga je dodavanje nelinearnosti u skrivene slojeve mreže, te uklanjanje zasićenost gradijenta za ulazne vrijednosti. [16] Pored ReLU postoje i druge aktivacijske funkcije: *Leaky ReLU (LReLU)*, *Parametric ReLU (PReLU)*, *Randomized ReLU (RReLU)*, *Exponential Linear Unit (ELU)*, i dr.
- 5) Kod potpuno povezanoga sloja svi neuroni su povezani sa svim izlazima prethodnog sloja. On je obično i posljednji sloj mreže. [14] U praksi se često koristi nekoliko potpuno povezanih slojeva na samom kraju neuronske mreže. Nakon potpuno povezanih slojeva slijedi rezultat, obično prikazan u vektorskom obliku. [16]

3. Implementacija

Cilj praktičnog dijela ovog rada bio je razviti sustav za optičko prepoznavanje znakova koji u procesu klasifikacije koristi duboko učenje odnosno konvolucijske neuronske mreže, te koji daje zadovoljavajuće rezultate. Izrađena je mobilna aplikacija kojom korisnik može učitati ili fotografirati sliku Sudoku igre, koja se potom procesuirala i kao rezultat daje brojke koje se nalaze u matrici Sudoku-a. Iako se OCR široko primjenjuje za prepoznavanje teksta, njegova prilagodba Sudoku mrežama postavlja jedinstvene izazove. Sudoku se sastoji od brojeva precizno raspoređenih u ćelije koje isto tako ne moraju sadržavati brojke, već mogu biti prazne. Taj precizan raspored brojki u ćelijama mora se točno interpretirati kako bi igra bila rješiva.

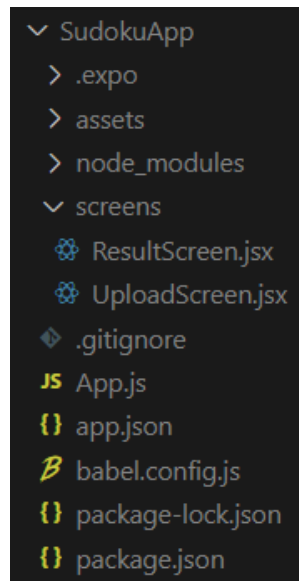
Aplikacija je izrađena u dva programska jezika. Programski jezik Python koristio se za razvoj OCR sustava, a JavaScript za izradu korisničkog sučelja mobilne aplikacije. U nastavku su opisani detalji izrade projekta.

3.1. Sučelje (frontend)

3.1.1. Korišteni alati

Sučelje, odnosno frontend mobilne aplikacije pisan je programskim jezikom JavaScript. Specifično, izrađen je u Visual Studio Code alatu za uređivanje izvornog koda, a za samo razvijanje mobilne aplikacije korišten je okvir Expo. Expo je okvir za razvoj React Native aplikacija koji pruža jednostavniju izradu projekta. Pogodan je za početnike jer pojednostavljuje kompleksne aspekte izrade projekta u React Native okruženju. Naravno, uz Expo korišteni su i okviri React i React Native, jer Expo koristi navedene biblioteke u izradi Expo aplikacija.

3.1.2. Arhitektura koda



Slika 5 Datoteke u frontend-u

Glavna datoteka u frontend dijelu aplikacije je `App.js`, iz koje se pokreće program i s njim „ekrani“ za učitavanje slike i pregled rezultata. Potrebna je za rukovođenje navigacijom između dva navedena ekrana. Navigacija se postiže bibliotekom *React Navigation* koja sadrži kontejner *Navigation Container* za praćenje stanja aplikacije i *Native Stack Navigator* za samu navigaciju između ekrana.

```

import React from "react";
import { NavigationContainer } from "@react-navigation/native";
import { createNativeStackNavigator } from "@react-navigation/native-stack";

import UploadScreen from "../screens/UploadScreen";
import ResultScreen from "../screens/ResultScreen";

const Stack = createNativeStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator
        screenOptions={{
          headerStyle: {
            backgroundColor: "#87a690",
          },
          headerTintColor: "#ebecda",
        }}
      >
        <Stack.Screen name="Home" component={UploadScreen} />
        <Stack.Screen name="Results" component={ResultScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;

```

Slika 6 Kod datoteke App.js

Kao što je već spomenuto, postoje dva ekrana koja se mogu prikazati u korisničkom sučelju mobilne aplikacije. Prvi ekran, *UploadScreen*, sadrži operacije za dohvaćanje slike, slanje slike u backend na obradu, te dohvaćanje rezultata koje backend pošalje nakon obrade. Drugi ekran, *ResultScreen*, prikazuje dohvaćeni rezultat.

3.1.2.1. Učitavanje slike

Kada se aplikacija pokrene, korisniku daje dvije opcije za učitavanje slike: učitavanje slike iz galerije i fotografiranje slike kamerom. Proces za obje opcije je isti: Kada se odabere opcija od korisnika se traži dopuštenje za otvaranje galerije, to jest korištenje kamere. Provjera dopuštenja omogućena je bibliotekom *Expo Image Picker*, u kojoj se nalazi metoda *requestMediaLibraryPermissionAsync* (*requestCameraPermissionAsync*). Ako je korisnik dopustio korištenje aplikaciji, sustav dohvaća sliku pomoću funkcije *launchImageLibraryAsync* (*launchCameraAsync*) koji sadrži vrstu rezultata, base64, i URI. Vrsta (engl. *type*), kao što i sam naziv govori, sadrži vrstu datoteke, na primjer slika, video, i slično. Base64 je

reprezentacija slike koja preoblikuje binarne podatke slike u tekstualne, osiguravajući prijenos slike bez gubljenja podataka. URI (engl. *Uniform Resource Identifier*) služi za identifikaciju resursa, u ovom slučaju slike. Nakon dohvaćanja slike, karakteristike slike šalju se funkciji *toServer* koji šalje podatke na obradu u backend.

Jedina razlika između funkcija za učitavanje iz galerije i fotografiranje je ta što funkcija za učitavanje slike iz galerije sliku uredi tako da ima jednaku visinu i širinu, te ju spremi u .jpeg format. Manipulacija slike postiže se funkcijom *manipulateAsync* iz biblioteke *Expo ImageManipulator*. Slika fotografirana kamerom unutar aplikacije već ima navedene karakteristike.

```
const openGallery = async () => {
  const checkPermission =
    await expoImagePicker.requestMediaLibraryPermissionsAsync();

  if (checkPermission.granted === false) {
    alert("Access denied");
    return;
  }

  const result = await expoImagePicker.launchImageLibraryAsync({
    base64: true,
  });

  console.log(result);

  if (!result.canceled) {
    const manipResult = await manipulateAsync(
      result.localUri || result.assets[0].uri,
      [{ resize: { height: 500, width: 500 } }],
      { base64: true, compress: 1, format: SaveFormat.JPEG }
    );

    setImageUri(manipResult);
    console.log(result.assets[0].uri);
    console.log("manipResult:");
    console.log(manipResult);

    await toServer({
      type: manipResult.type,
      base64: manipResult.base64,
      uri: manipResult.uri,
    });
  } else {
    setImageUri(null);
  }
};
```

Slika 7 Kod funkcije „openGallery“ za učitavanje slike iz galerije

```

const openCamera = async () => {
  const checkPermission =
    await expoImagePicker.requestCameraPermissionsAsync();

  if (checkPermission.granted === false) {
    alert("Access denied");
    return;
  }

  const result = await expoImagePicker.launchCameraAsync({
    base64: true,
  });

  console.log(result);

  if (!result.canceled) {
    setImageUri(result.assets[0].uri);
    console.log(result.assets[0].uri);

    await toServer({
      type: result.type,
      base64: result.base64,
      uri: result.uri,
    });
  } else {
    setImageUri(null);
  }
};

```

Slika 8 Kod funkcije „openCamera“ za učitavanje fotografirane slike

3.1.2.2. Povezivanje s backend-om

Nakon što je slika učitana u program, bilo iz galerije ili kamere, šalje se u metodu *toServer* kao ulazni podatak. Funkcija služi za slanje slike u backend dio programa koji vrši OCR nad slikom, te za dohvaćanje rezultata OCR-a. Funkcija šalje sliku na specificiranu IP adresu backend servera pomoću HTTP zahtjeva „POST“. Slanje podataka u server omogućen je funkcijom *fetch* koja vrši HTTP zahtjev navedenoj IP adresi. Kada se kod servera izvrši, funkcija dohvaća rezultat pomoću „*then()*“ metode. Dakle, funkcija *toServer* šalje na server podatke o slici Sudoku mreže, a dobiva natrag niz brojki iz odabrane slike.

```

toServer = async (file) => {
  let ip = "192.168.5.25";
  let content_type = "image/jpeg";
  let url = "http://" + ip + ":5000/image";

  const response = await fetch(url, {
    method: "POST",
    headers: {
      "content-type": content_type,
    },
    body: file,
  }).then((response) =>
    response.json().then((data) => {
      console.log("Data: ");
      console.log(data);
      sudokuDigits = data;
      console.log("Sudoku digits:");
      console.log(sudokuDigits);
    })
  );
};

```

Slika 9 Kod funkcije „toServer“ za komunikaciju s backend serverom

3.1.2.3. Prikaz rezultata

Posljednja stavka sučelja aplikacije je ekran za prikaz rezultata OCR sustava. Funkcija *ResultScreen* za prikaz ekrana prvo dohvaća podatke (brojke Sudoku mreže) s ekrana *UploadScreen* pomoću metode *useRoute* iz biblioteke *React Navigation* koja omogućava pristup objektu s rute. Da bi se dohvaćanje podataka ostvarilo, treba u ruti (*UploadScreen*) definirati slanje podataka (Slika 10). Ako podaci još nisu obrađeni, to jest ako je niz brojki prazan, na ekranu se pojavljuje prazna Sudoku mreža dimenzija 9x9, odnosno ako su podaci obrađeni, na ekranu se pojavljuje Sudoku mreža s brojkama na odgovarajućim mjestima. Prikaz mreže ostvaren je komponentom *FlatList* iz *React Native* biblioteke.

```

const checkResult = () => {
  navigation.navigate("Results", { sudokuDigits });
};

```

Slika 10 Funkcija "checkResult" u ekranu "UploadScreen" koja omogućava navigaciju na ekran „ResultScreen“ i slanje brojki Sudoku mreže.

```

const columns = 9;

const ResultScreen = () => {
  const route = useRoute();
  const sudokuDigits = route.params.sudokuDigits;

  renderItem = ({ item, index }) => {
    if (item === 0) {
      return <View style={[styles.item, styles.itemInvisible]} />;
    }

    return (
      <View style={styles.item}>
        <Text style={styles.itemText}>{item}</Text>
      </View>
    );
  };

  return (
    <FlatList
      data={sudokuDigits}
      style={styles.container}
      renderItem={renderItem}
      numColumns={columns}
    />
  );
};

export default ResultScreen;

```

Slika 11 Kod funkcije "ResultScreen" za prikaz prepoznatih brojki u Sudoku mreži

3.2. OCR sustav (backend)

3.2.1. Korišteni alati

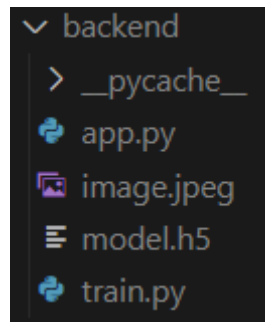
OCR sustav, odnosno server ili backend mobilne aplikacije pisan je programskim jezikom Python. Što se tiče samog razvoja koda, zbog jednostavnosti provjere rada koda na računalu korištena je Google Colaboratory okolina, potom je kod prenesen u Visual Studio Code, uz dodatak povezivanja servera sa sučeljem. Povezivanje servera sa sučeljem ostvareno je pomoću Flask-a. Flask je okvir koji pomaže pri realiziranju web aplikacija bez brige o detaljima kao što su protokoli, upravljanje nitima, itd.

Za samu izradu OCR sustava korištene su sljedeće biblioteke:

- *OpenCV* otvorena biblioteka za računalni vid,
- *NumPy* biblioteka za multi-dimenzionalne matrice i matematičke operacije,
- *Operator* biblioteka za operacije u Pythonu. U ovom projektu koristi se metoda *itemgetter* unutar biblioteke,

- *TensorFlow* biblioteka za strojno učenje,
- *os* biblioteka za interakciju s operacijskim sustavom. U ovom projektu koristi se metoda *listdir* za čitanje naziva datoteka na putanji.

3.2.2. Arhitektura koda



Slika 12 Datoteke u backend-u

Arhitektura samoga koda prilično je jednostavna. Server sadrži sliku koja se šalje sa sučelja na server (*image.jpeg*), datoteku *train.py* koja sadrži kod za stvaranje i uvježbavanje modela te napravljen model (*model.h5*), i glavnu datoteku *app.py* zaslužnu za cijeli OCR sustav i povezivanje sa sučeljem.

3.2.2.1. Povezivanje s frontend-om

Dohvaćanje slike sa sučelja i vraćanje rezultata prepoznavanja na sučelje omogućeno je upotrebnom okvira Flask. Za korištenje Flask-a prvo treba stvoriti Flask aplikaciju, što je omogućeno linijom koda `app = Flask(__name__)`. Nakon stvaranja aplikacije treba definirati CORS (engl. *Cross-Origin Resource Sharing*), tj. označiti koje domene imaju dopuštenje slati HTTP zahtjeve aplikaciji. U ovom slučaju dopušteno je svim domenama slati zahtjeve jer se radi o lokalnoj aplikaciji.

```
app = Flask(__name__)
cors = CORS(app, resources={r'/*': {'origins': '*'}})
```

Slika 13 Kod za stvaranje Flask aplikacije i određivanje domena s dopuštanjem slanja zahtjeva

Kao što je vidljivo na Slici 9, slika sa sučelja se šalje na određenu IP adresu i rutu. Navedenu rutu */image* treba definirati i u Flask aplikaciji, kao i HTTP zahtjeve koje prihvća. Omogućuje se kodom `@app.route („/image“, methods=[ 'GET', 'POST' ]` gdje se definira spomenuta ruta */image* te zahtjeve „GET“ i „POST“. Tek nakon stvaranja aplikacije i definiranja rute moguće je dohvatiti sliku sa sučelja. U metodi *image()* provjerava se je li poslan

HTTP zahtjev „POST“ (što je u ovom slučaju točno, ref. Slika 9). Ako je istina, slika se sprema u varijablu *bytesOfImage* nakon čega se pohranjuje na disk u datoteku pod nazivom *image.jpeg*.

```
@app.route("/image", methods=['GET', 'POST'])
def image():
    if (request.method == "POST"):
        bytesOfImage = request.get_data()
        with open('image.jpeg', 'wb') as out:
            out.write(bytesOfImage)
```

Slika 14 Kod za definiranje rute i HTTP zahtjeva, te dohvaćanje i pohranjivanje slike na disk

Sada je server spreman obraditi zadatke, u ovom slučaju izvršavanje koda za OCR sustav. Sastav koda za OCR objašnjen je u sljedećem poglavlju. Nakon što se slika obradi, Flask vraća dobivene rezultate natrag na sučelje pomoću linije koda `return result.tolist()` koja je završni korak metode *image()*.

3.2.2.2. Izrada OCR sustava

Kako je objašnjeno u Poglavlju 1.3., procesi OCR sustava prate određene korake svrstane u 6 kategorija: pretprocesiranje, segmentaciju, normalizaciju, ekstrakciju značajki, klasifikaciju, te postprocesiranje. U ovom poglavlju objasniti će se izrada OCR sustava za prepoznavanje brojki Sudoku mreže.

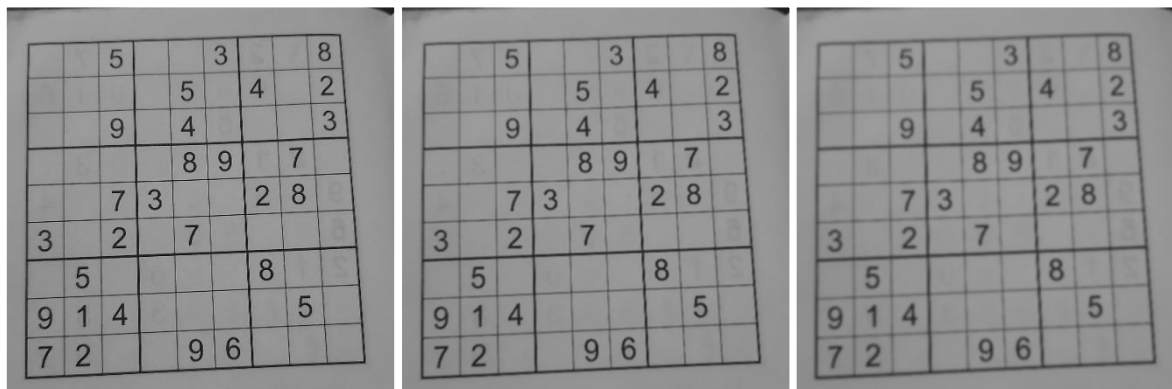
Važno je napomenuti da postoji više načina klasifikacije OCR sustava te da se sami procesi ne moraju nužno izvršavati sekvencijalno. Svaka različita implementacija OCR sustava ima specifične zahtjeve, pa tako i ovaj projekt. Devijacija od striktnog praćenja sekvence procesa OCR sustava nije nužno loša stvar ako se potrebni elementi točno implementiraju.

Pretprocesiranje

Nakon što se slika učitava, treba napraviti nad njom određene promjene da bi bila spremna za sljedeće korake OCR-a. Prvo se originalna slika, koja je u boji, preoblikuje u sliku sa sivim tonovima pomoću linije `gray = cv2.cvtColor(og, cv2.COLOR_RGB2GRAY)`, gdje je *gray* nova siva slika, a *og* učitana slika. Specifično, mijenjanje boje slike omogućeno je funkcijom *cvtColor* iz biblioteke *OpenCV*. Digitalna slika u boji je trodimenzionalni niz brojeva koji određuje zastupanje crvene, zelene i plave boje. Pretvaranje slike u boji u sive tonove pretvara trodimenzionalan niz u jednu vrijednost od 0 do 255 gdje je 0 crni, a 255 bijeli piksel.

Potom se izvršava redukcija šuma, u ovom slučaju ostvarena Gaussovim zaglađivanjem (engl. *Gaussian blur*). Gaussovo zaglađivanje, nazvano po matematičaru Carl Friedrich

Gaussu, vrsta je filtera koji zaglađuje neravne vrijednosti piksela tako što izbacuje ekstremna odstupanja. U biblioteci *OpenCV* nalazi se metoda *GaussianBlur* koja pojednostavljuje implementaciju zaglađivanja. Metoda prima 3 vrijednosti: sliku koju treba zagladiti, veličinu jezgre, te standardnu devijaciju jezgre horizontalno. Što je veličina jezgre veća, to je zaglađivanje više istaknuto (Slika 15).



Slika 15 Rezultati nastali primjenom Gaussovog zaglađivanja s različitim veličinama jezgre (s lijeva): (1,1), (5,5) (9,9)

Nakon redukcije šuma realizirano je određivanje praga (engl. *thresholding*) pomoću metode *adaptiveThreshold* u kojoj je vrijednost praga računata u malim regijama. Adaptivni prag uzima u obzir različite jačine osvjetljenja dijelova slike jer računa različite pragove za različite regije slike. Uz određivanje praga, metoda *adaptiveThreshold* također pretvara sliku u binarnu pomoću parametra *cv2.THRESH_BINARY*. Sada su elementi na slici predstavljeni crnom bojom, a pozadina bijelom. Crna boja je jednaka binarnoj vrijednosti nule, stoga se boje slike invertiraju da pikseli elemenata slike nisu jednaki nuli. Inverzija boja ostvarena je linijom koda `bitwisenot = cv2.bitwise_not(thresh, thresh)` u kojoj metoda *bitwise_not* uzima binarnu sliku s određenim pragom te invertiranu sliku pohranjuje u varijablu *bitwisenot*.

		5		3			8
				5		4	2
		9		4			3
				8	9		7
		7	3			2	8
3		2		7			
	5					8	
9	1	4					5
7	2			9	6		

		5		3			8
				5		4	2
		9		4			3
				8	9		7
		7	3			2	8
3		2		7			
	5					8	
9	1	4					5
7	2			9	6		

Slika 16 Slika nakon određivanja praga i binarizacije (lijevo), te nakon invertiranja binarne slike (desno)

Zadnji korak pretprocesiranja u ovom projektu je zadebljanje elemenata, što će pomoći prepoznavanju elemenata slike u sljedećim koracima. Zadebljanje se ostvaruje funkcijom *dilate* iz biblioteke *OpenCV*.

		5		3			8
				5		4	2
		9		4			3
				8	9		7
		7	3			2	8
3		2		7			
	5					8	
9	1	4					5
7	2			9	6		

		5		3			8
				5		4	2
		9		4			3
				8	9		7
		7	3			2	8
3		2		7			
	5					8	
9	1	4					5
7	2			9	6		

Slika 17 Ulazna slika (lijevo) i slika nakon pretprocesiranja (desno)

```

og = cv2.imread(img)
og = cv2.resize(og, (500,500))
gray = cv2.cvtColor(og, cv2.COLOR_RGB2GRAY)

#smanjenje šuma
blur = cv2.GaussianBlur(gray, (9, 9), 0)

#određivanje praga i binarizacija
thresh = cv2.adaptiveThreshold(
    blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

#inverzija boje
bitwisenot = cv2.bitwise_not(thresh,thresh)

#proširivanje elemenata
kernel = np.array([[0., 1., 0.], [1., 1., 1.], [0., 1., 0.]], np.uint8)
dilate = cv2.dilate(bitwisenot, kernel)

```

Slika 18 Kod procesa pretprocesiranja

Segmentacija

Nakon provođenja procesa preprocesiranja nad slikom potrebno je segmentirati sliku, to jest pronaći prvotno vanjske granice Sudoku mreže pa granice pojedinih ćelija unutar mreže, nakon čega se slika razdvaja u zasebne ćelije.

Pronalaženje vanjskih granica ostvareno je funkcijom *findContours* iz biblioteke *OpenCV* koja prima 3 parametra: ulaznu binarnu sliku *dilate*, metodu vraćanja *cv2.RETR_EXTERNAL* koja vraća vanjske granice, te metodu aproksimacije *cv2.CHAIN_APPROX_SIMPLE* koja sažima segmente i ostavlja samo njene krajnje točke. Nakon pronalaženja vanjskih granica treba pronaći četiri vanjske točke koja će kasnije pomoći pri rezanju slike. Za pronalazak točaka korištena je funkcija *operator.itemgetter* koja uzima maksimume, odnosno minimume zbroja točaka u vanjskom poligonu.

```

contours, q = cv2.findContours(dilate, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = sorted(contours, key=cv2.contourArea, reverse=True)
polygon = contours[0]

br, q = max(enumerate([pt[0][0] + pt[0][1] for pt in polygon]), key=operator.itemgetter(1))
tl, q = min(enumerate([pt[0][0] + pt[0][1] for pt in polygon]), key=operator.itemgetter(1))
bl, q = min(enumerate([pt[0][0] - pt[0][1] for pt in polygon]), key=operator.itemgetter(1))
tr, q = max(enumerate([pt[0][0] - pt[0][1] for pt in polygon]), key=operator.itemgetter(1))

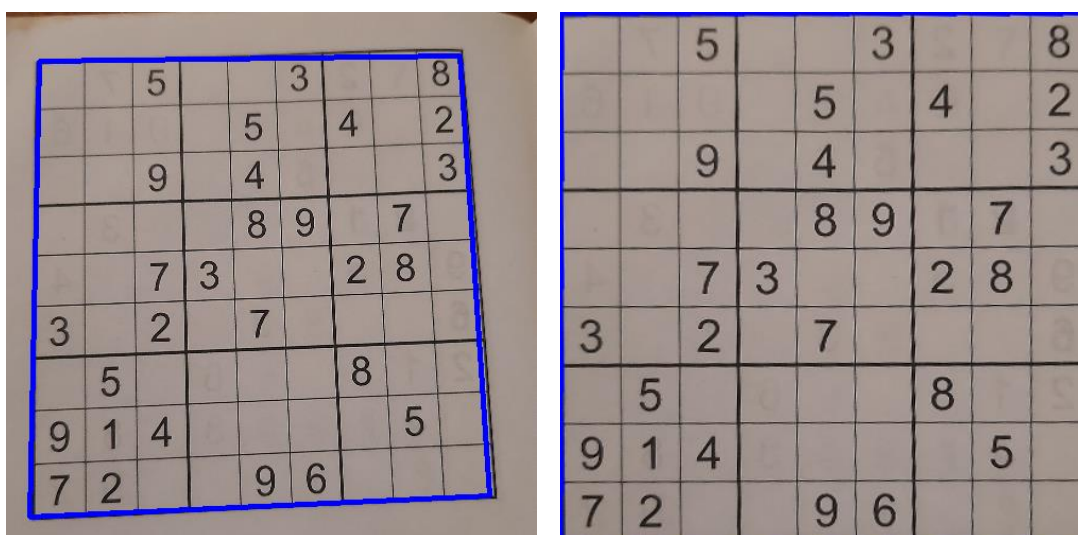
corners=[polygon[tl][0], polygon[tr][0], polygon[br][0], polygon[bl][0]]

```

Slika 19 Kod za pronalaženje vanjskih granica Sudoku mreže

Nakon pronalaska vanjskih točaka Sudoku mreže potrebno je maknuti sve nepotrebne informacije. Specifično, sliku treba izrezati tako da ostane sama Sudoku mreža, a izbriše

pozadina oko nje. U varijablu *max_line* pohranjuje se maksimalna duljina između četiri moguće strane Sudoku mreže. Duljina stranica računa se matematičkom formulom za računanje udaljenosti točaka pomoću koordinata $A(x_1, y_1)$ i $B(x_2, y_2)$ koja glasi $d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Pomoću duljine najveće strane definira se traženi oblik *dst* za obrezanu sliku. Prije samog obrezivanja slike, prvo se krajnje točke trebaju mapirati na prethodno definiran *dst* oblik. Naposljetku se slika obrezuje i ispravlja joj se kosina pomoću linije koda `warp = cv2.warpPerspective(og, transform_matrix, (int(max_line), int(max_line)))` gdje se u varijablu *warp* pohranjuje zakrivljena i razrezana slika. Dobra je praksa ispraviti kosinu slike jer korisnik može fotografirati Sudoku mrežu pod kutom, što se i po Slici 20 ispod može vidjeti.



Slika 20 Vizualizacija pronađenih granica Sudoku mreže (lijevo), te izrezana i zakrivljena Sudoku mreža (desno)

S centriranom i izrezanom slikom Sudoku mreže, sada se mreža može izrezati na 81 sličicu koje predstavljaju ćelije mreže. Redovi i stupci Sudoku mreže jednake su širine pa je jednostavno izračunati krajnje točke svake ćelije. Iz izrezane slike uzima se vertikalna duljina slike koja se potom dijeli s 9 da bi se dobila širina ćelije. Horizontalna duljina u ovom slučaju nije bitna jer je slika kvadratna. Za izračun koordinata krajnjih točaka ćelija koristi se ugniježđena *for* petlja koja za svaki stupac i red u stupcu pohranjuje u niz *sq* koordinate gornjeg lijevog i donjeg desnog kuta pomoću izračunate širine ćelije. Sada se ćelije mogu zasebno gledati za realizaciju ekstrakcije značajki.

```

sq = []
side = warp.shape[:1]
side = side[0]/9

for j in range(9):
    for i in range(9):
        p1 = (i * side, j * side) # Gornji lijevi kut
        p2 = ((i + 1) * side, (j + 1) * side) # Donji desni kut
        sq.append((p1, p2))

```

Slika 21 Kod za izračun granica pojedinih ćelija

Ekstrakcija značajki

U procesu ekstrakcije značajki pregledava se svaka ćelija i odvaja se brojka od pozadine. Brojka se očekuje u sredini ćelije, stoga se određuje margina *margin* pomoću aritmetičke sredine visine i širine ćelije koja potom određuje gornji lijevi kut *scan_tl* i donji desni kut *scan_br* središnjeg dijela ćelije.

```

dig = dilate[int(square[0][1]):int(square[1][1]), int(square[0][0]):int(square[1][0])]

h, w = dig.shape[:2]
margin = int(np.mean([h, w]) / 2.5)

scan_tl = [margin, margin]
scan_br = [w - margin, h - margin]

```

Slika 22 Kod za dohvaćanje sličice ćelije i definiranje središnjeg dijela ćelije

Nakon određivanja sredine ćelije koristi se ugniježđena ćelija u kojoj se provjerava svaki piksel unutar središnjeg dijela. Ako je piksel bijele boje, provodi se funkcija *floodFill* iz biblioteke *OpenCV* koja rekurzivno mijenja boju svih povezanih piksela iste boje u sivu, počevši od pronađenog piksela. Rezultat operacije sprema se u varijablu *area*, čija prva vrijednost sadrži količinu piksela koju je *floodFill* obojao. Ako je taj broj veći od prethodno pronađenog maksimalnog broja obojanog područja, ta brojka postaje novo maksimalno povezano područje. U konačnici, najveće područje pronađeno će biti brojka koja se tražila u ćeliji. Za svaki slučaj izvršava se provjera svih piksela u cijeloj ćeliji da se nađu eventualni pikseli koji nisu u središnjem području. Ako se pronađe najveće povezano područje, ono se boja u bijelu boju, dok manja područja ostaju siva. Sada ta manja područja treba promijeniti u crnu boju da se stope s pozadinom. Koristi se još jedna ugniježđena *for* petlja i *floodFill* funkcija za promjenu boje sivih područja u crnu. Također, u petlji se određuju granice područja brojke u koordinatama.

```

for x in range(scan_tl[0], scan_br[0]):
    for y in range(scan_tl[1], scan_br[1]):
        if dig.item(y, x) == 255 and x < w and y < h:
            area = cv2.floodFill(dig, None, (x, y), 64)
            if area[0] > max_area:
                max_area = area[0]
                seed_point = (x, y)

for x in range(w):
    for y in range(h):
        if dig.item(y, x) == 255 and x < w and y < h:
            cv2.floodFill(dig, None, (x, y), 64)

mask = np.zeros((h + 2, w + 2), np.uint8)

if all([p is not None for p in seed_point]):
    cv2.floodFill(dig, mask, seed_point, 255)

t, b, l, r = h, 0, w, 0

for x in range(w):
    for y in range(h):
        if dig.item(y, x) == 64:
            cv2.floodFill(dig, mask, (x, y), 0)

        if dig.item(y, x) == 255:
            t = y if y < t else t
            b = y if y > b else b
            l = x if x < l else l
            r = x if x > r else r

bbox = [[l, t], [r, b]]
bbox = np.array(bbox, dtype='float32')

dig = dig[int(bbox[0][1]):int(bbox[1][1]), int(bbox[0][0]):int(bbox[1][0])]

```

Slika 23 Kod za pronalazak najvećeg povezanog područja i njegovih graničnih koordinata

Nakon pronalaska brojke u ćeliji, dimenzije sličice se oblikuju da budu primjenjive u klasifikaciji.

		5			3			8
				5		4		2
		9		4				3
				8	9		7	
		7	3			2	8	
3		2		7				
	5					8		
9	1	4					5	
7	2			9	6			

Slika 24 Pregled ekstrahiranih ćelija i brojki u njima

Klasifikacija

Za klasifikaciju brojki Sudoku mreže izrađena je konvolucijska neuronska mreža, koja ima sekvencijalni model od osam slojeva. Model konvolucijske neuronske mreže realiziran je pomoću *TensorFlow*, popularne biblioteke za strojno učenje. Prvi konvolucijski sloj ima 32 izlazna filtera, konvolucijski prozor veličine 5x5 piksela te koristi aktivacijsku funkciju „ReLU“. Drugi sloj je sloj sažimanja maksimumom (engl. *Max Pooling*) koji uzima maksimalnu vrijednost područja dimenzije 2x2 piksela. Treći sloj, odnosno drugi konvolucijski sloj pak sadrži 16 izlaznih filtera s konvolucijskim prozorom veličine 3x3 piksela, nakon kojeg slijedi drugi sloj sažimanja maksimumom s područjem sažimanja dimenzije 2x2 piksela. Peti sloj je *dropout* sloj koji se brine o prekomjernom opremanju (engl. *overfitting*). *Dropout* je tehnika gdje se nasumično odabrani neuroni ignoriraju tokom treniranja (u ovom slučaju zadržat će se 20% čvorova). Šesti sloj je izravnavajući sloj (engl. *flatten layer*) koji ravna podatke čvorova tako da ih prikaže u jednoj dimenziji. Zadnji skriveni sloj je potpuno povezan (engl. *Fully Connected*) sloj koji ima 128 filtera i aktivacijsku funkciju „ReLU“. Konačno, izlazni sloj ima 10 izlaznih klasa (za svaki znak 0-9, gdje 0 predstavlja praznu ćeliju) i aktivacijsku funkciju „Softmax“ koja se koristi za višeklasnu klasifikaciju.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 1, 28, 32)	22432
max_pooling2d_1 (MaxPooling2D)	(None, 1, 14, 32)	0
conv2d_2 (Conv2D)	(None, 1, 14, 16)	4624
max_pooling2d_2 (MaxPooling2D)	(None, 1, 7, 16)	0
dropout (Dropout)	(None, 1, 7, 16)	0
flatten (Flatten)	(None, 112)	0
dense (Dense)	(None, 128)	14464
dense_1 (Dense)	(None, 10)	1290

=====
Total params: 42810 (167.23 KB)
Trainable params: 42810 (167.23 KB)
Non-trainable params: 0 (0.00 Byte)

Slika 25 Struktura modela neuronske mreže

Svaki model neuronske mreže treba podatke po kojima će se učiti. U ovom projektu korištena je baza podataka MNIST (engl. *Modified National Institute of Standards and Technology database*) koja sadrži velik broj ručno pisanih brojk. Baza podataka dijeli se na podatke za treniranje, kojom će se model neuronske mreže učiti, te podatke za testiranje, kojom se realizira provjera učenja mreže.



Slika 26 Primjer podataka u bazi MNIST [17]

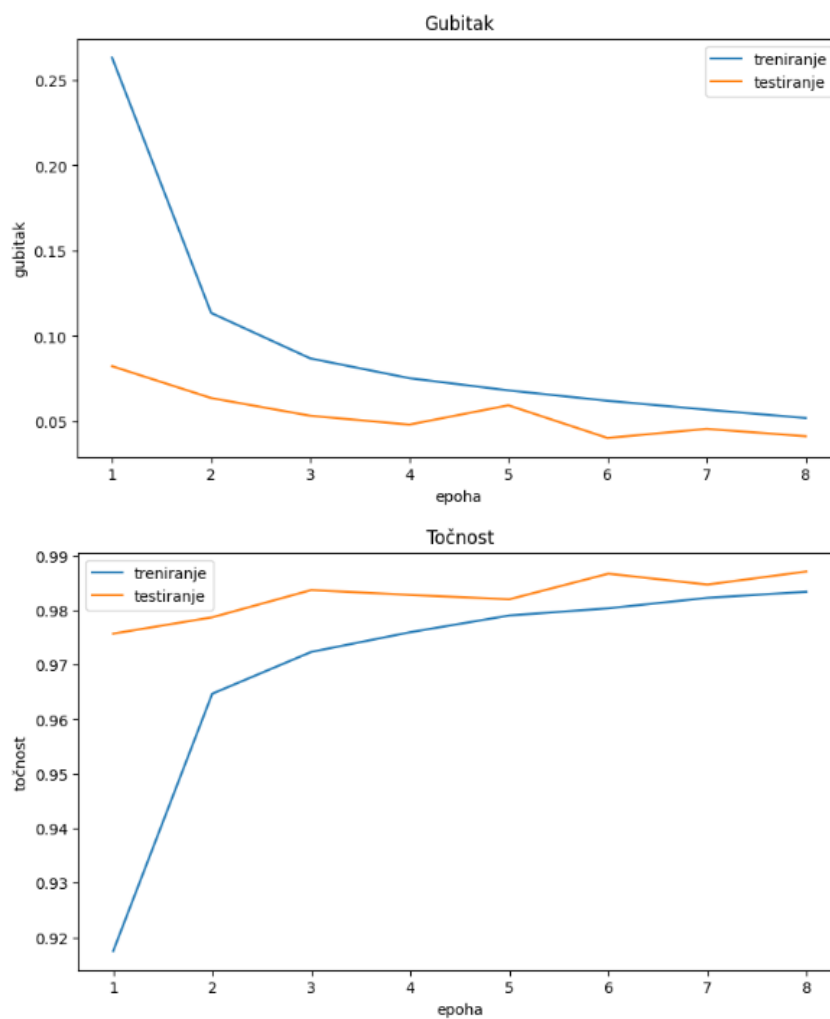
Da bi se mreža uvježbavala, mora se prvo kompilirati funkcijom `model.compile()`, specifično linijom koda `model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])`. Funkcija prima tri parametra: funkciju gubitka (engl. *loss function*), optimizator, te metrike. Funkcija gubitka mjeri koliko se uspješno model trenira, odnosno uspoređuje traženo rješenje sa rješenjem koje model pretpostavi. U ovom projektu korištena je funkcija gubitka *categorical_crossentropy* koja je popularan izbor za modele višeklasne klasifikacije. Optimizator, kao što i sam naziv kaže, povećava efikasnost učenja. Korišten je optimizator *adam* koji prilagođava stopu učenja za svaki parametar u modelu. Metrike određuju što točno model treba učiti i prate napredak prema tom cilju. U slučaju prepoznavanja brojki bitna je točnost, stoga se koristi metrika točnosti (engl. *accuracy*).

Metrika točnosti računa se tako da se broj točno predviđenih podataka podijeli s brojem svih podataka. Znači, ona pokazuje vjerojatnost točnoga predviđanja svih brojki. Ako je model točno prepoznao 80 od 100 podataka, točnost mu je 80%. Također postoji i metrika za računanje preciznosti, koja se računa tako da se broj točno predviđene brojke podijeli sa sumom točno predviđenih i netočno predviđenih brojki iste kategorije. Na primjer, baza podataka sadrži 10 instanci brojke 1, a model je prepoznao 8 od 10 jedinica te je 8 sedmica prepoznao kao jedinicu. Preciznost prepoznavanja brojke 1 u tom slučaju je 50% (8 točnih jedinica / 8 točnih jedinica + 8 netočnih jedinica). U ovom projektu koristila se samo metrika točnosti za prikazivanje koliko je prepoznavanje brojki blizu istinitoj vrijednosti.

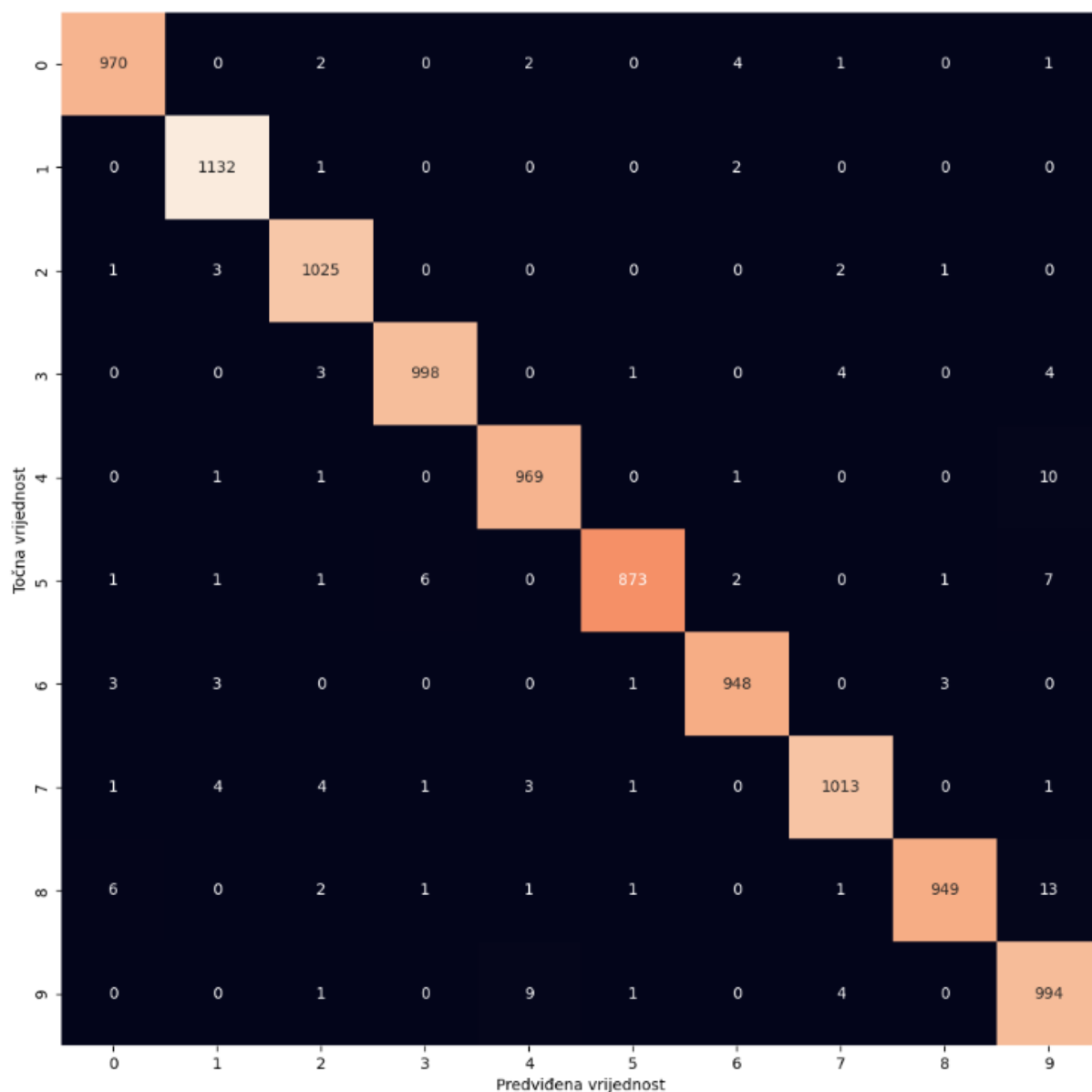
Model se trenira pomoću funkcije `model.fit()`, specifično linijom koda `history = model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 8, batch_size = 16, verbose = 2)`. Funkcija uzima parametre *X_train* i *y_train* kao podatke na kojima će se učiti, te podatke za provjeru *X_test* i *y_test*. Također prima broj epoha, a epoha označava jedan prolazak cijele baze podataka kroz model. U ovom slučaju broj epoha jednak je 8. Funkcija također prima i veličinu serije (engl. *batch size*), odnosno količinu podataka u jednoj seriji. Ako model ima 1000 podataka, a veličina serije je 10, modelu će trebati 100 iteracija da bi završio jednu epohu. U ovom slučaju veličina serije je 16.

Testiranje modela pokazalo je zadovoljavajuće podatke. Postotak točnosti prepoznavanja brojki je 98.34%, a matrica konfuzije također prikazuje da su se u većini slučajeva točno prepoznale brojke (Slika 28). Najveće greške koju neuronska mreža radi su

predviđanje broja 9 umjesto broja 4 i broja 9 umjesto broja 8, no te greške su zanemarive jer je jako malen broj.

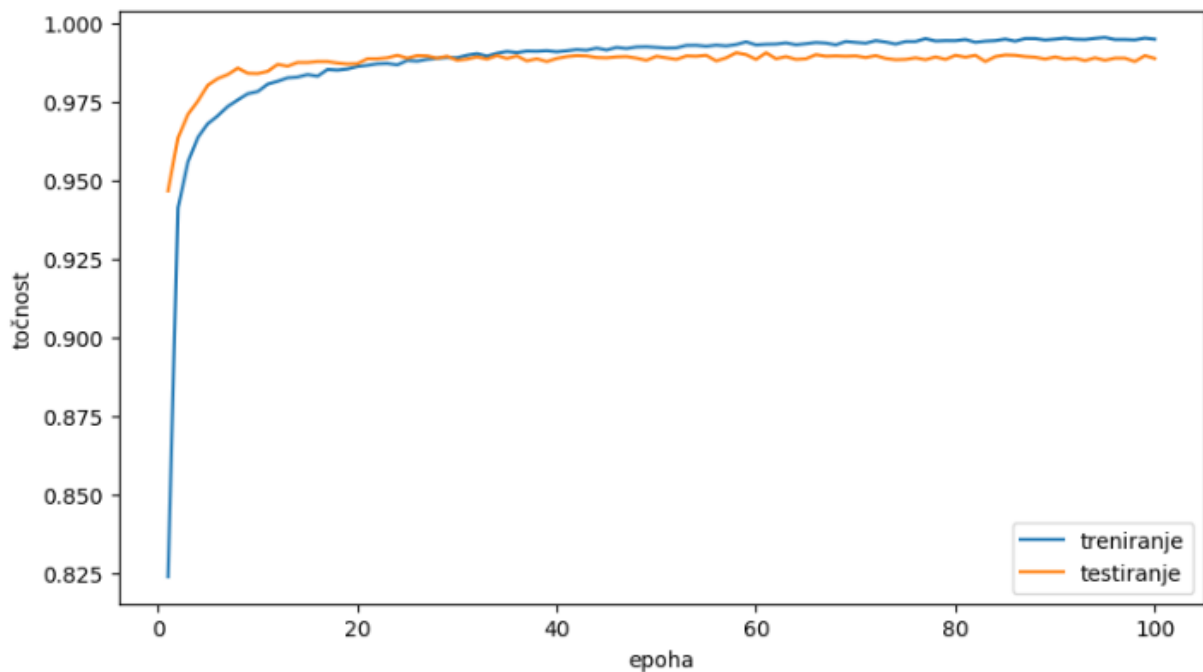


Slika 27 Grafovi prikaza gubitka (gore) i točnosti (dolje) pri treniranju i testiranju po epohama



Slika 28 Matrica konfuzije testirane mreže

Naravno, mora se uzeti u obzir da neuronska mreža ne smije biti previše istrenirana (engl. *overtrained*). Suvišno treniranje najlakše se može vidjeti iz grafa koji prikazuje točnost rezultata treniranja i testiranja modela, kada točnost treniranih vrijednosti raste, a točnost testiranih vrijednosti pada. Na primjer, ako podignemo broj epoha na 100 i veličinu serije na 200 postotak točnosti je 98.87%, no graf na Slici 29 pokazuje da zapravo točnost prepoznavanja podataka za testiranje pada što se neuronska mreža više trenira. Dakle, treba paziti koji broj epoha i veličinu serije koristiti.



Slika 29 Graf koji prikazuje točnost prepoznavanja podataka za treniranje i testiranje na modelu neuronske mreže sa 100 epoha i veličinom serije 200

Treniranje modela neuronske mreže može dugo trajati, pogotovo u ovom slučaju gdje je veličina serije mala pa svaka epoha ima mnogo iteracija. Da bi se izbjeglo treniranje svaki put pri pokretanju, model je pohranjen u datoteku na disku nazvanu *model.h5*.

3.2.2.3. Prepoznavanje brojki Sudoku mreže

Nakon izrade OCR sustava i treniranja modela za klasifikaciju brojki, potrebno je pomoću modela prepoznati brojke u Sudoku mreži. Pomoću petlje provjerava se svaka ćelija sudoku mreže koju prvo treba preoblikovati u dimenziju (1, 1, 28, 28) kakvu model očekuje. Nakon preoblikovanja model prepoznaje brojku koja se nalazi u ćeliji pomoću funkcije *model.predict*. Funkcija vraća niz vrijednosti koje označavaju sigurnost pretpostavke za svaku moguću brojku. Na primjer, ako funkcija vrati niz jednak [0, 0, 1, 0, 0, 0, 0, 0, 0, 0], onda je model siguran da je pronašao brojku 2. Prepoznata brojka sprema se u niz *row*, potom se red spremi u niz *sudoku*.

```

if 'model.h5' in os.listdir('/content/drive/MyDrive/faks'):
    model.load_weights('/content/drive/MyDrive/faks/model.h5')

dic = {}
char = '0123456789'

for i, c in enumerate(char):
    dic[i] = c

sudoku = []
row = []

for i, dig in enumerate(digits):
    img = cv2.resize(dig, (28,28))
    img = dig.reshape(1, 1, 28, 28)
    pred = (model.predict(img) > 0.5).astype("int32")
    print(pred)
    classes = np.argmax(pred)
    print(classes)
    character = int(dic[classes])
    print(character)
    row.append(int(character))

    if (i + 1) % 9 == 0:
        sudoku.append(row)
        row = []

sudoku = np.array(sudoku)

```

Slika 30 Kod za prepoznavanje brojke Sudoku mreže pomoću treniranog modela

Naposljetku je dobivena Sudoku mreža u obliku niza u kojem su brojke jednake onima s ulazne slike, a nule jednake praznim poljima.

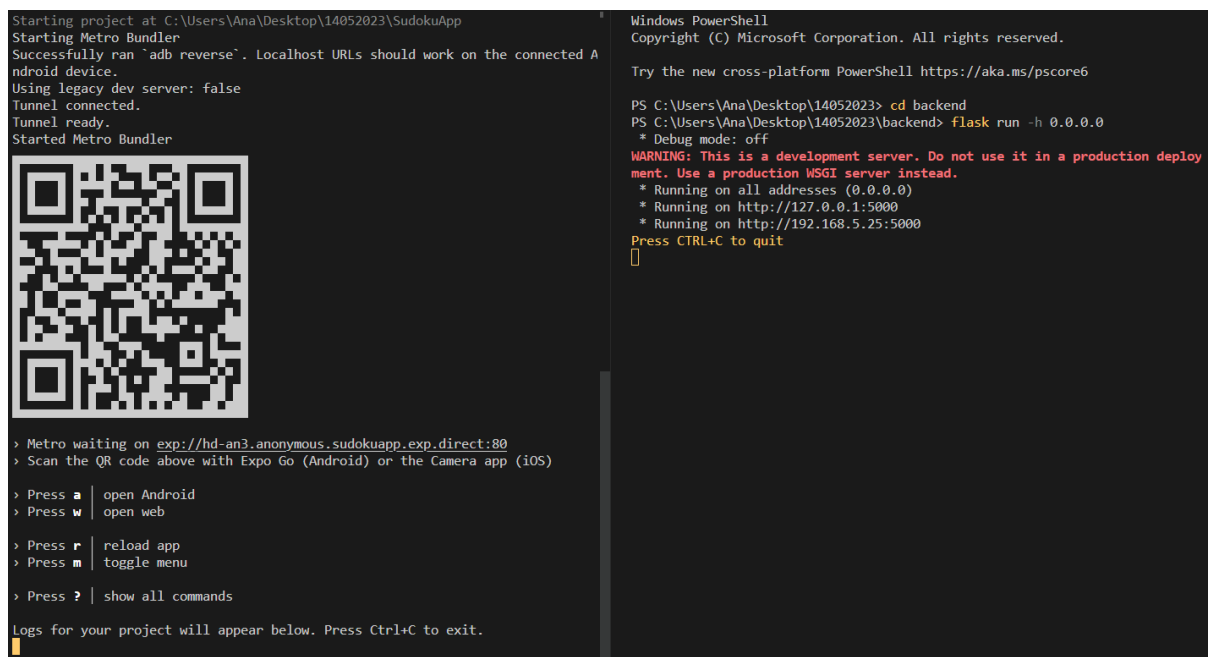
0	0	5		0	0	3		0	0	8
0	0	0		0	5	0		4	0	2
0	0	9		0	4	0		0	0	3
-----+-----+-----										
0	0	0		0	8	9		0	7	0
0	0	7		3	0	0		2	8	0
3	0	2		0	7	0		0	0	0
-----+-----+-----										
0	5	0		0	0	0		8	0	0
9	1	4		0	0	0		0	5	0
7	2	0		0	9	6		0	0	0

Slika 31 Prepoznate brojke s ulazne slike

3.3. Pokretanje i izgled aplikacije

Aplikacija se pokreće lokalno, unutar Visual Studio Code alata. Potrebno je pokrenuti zasebno sučelje i server, to jest frontend i backend. Nakon pokretanja servera pojavi se

informacija o IP adresi na kojoj je server pokrenut (u ovom slučaju `http://192.168.5.25:5000`), a nakon pokretanja sučelja pojavi se QR kod (Slika 32). QR kod je potreban jer se aplikacija, u ovom slučaju, ne pokreće na računalu, već na mobilnom uređaju. Specifično, za pregled rada aplikacije korištena je mobilna aplikacija „Expo Go“ koja služi kao emulator za mobilne aplikacije napravljene pomoću Expo okvira. Kada se QR kod skenira preko navedene aplikacije, sučelje se tada, preko IP adrese, šalje na mobilni uređaj.



```
Starting project at C:\Users\Ana\Desktop\14052023\SudokuApp
Starting Metro Bundler
Successfully ran `adb reverse`. Localhost URLs should work on the connected Android device.
Using legacy dev server: false
Tunnel connected.
Tunnel ready.
Started Metro Bundler

> Metro waiting on exp://hd-an3.anonymous.sudokuapp.exp.direct:80
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

> Press a | open Android
> Press w | open web

> Press r | reload app
> Press m | toggle menu

> Press ? | show all commands

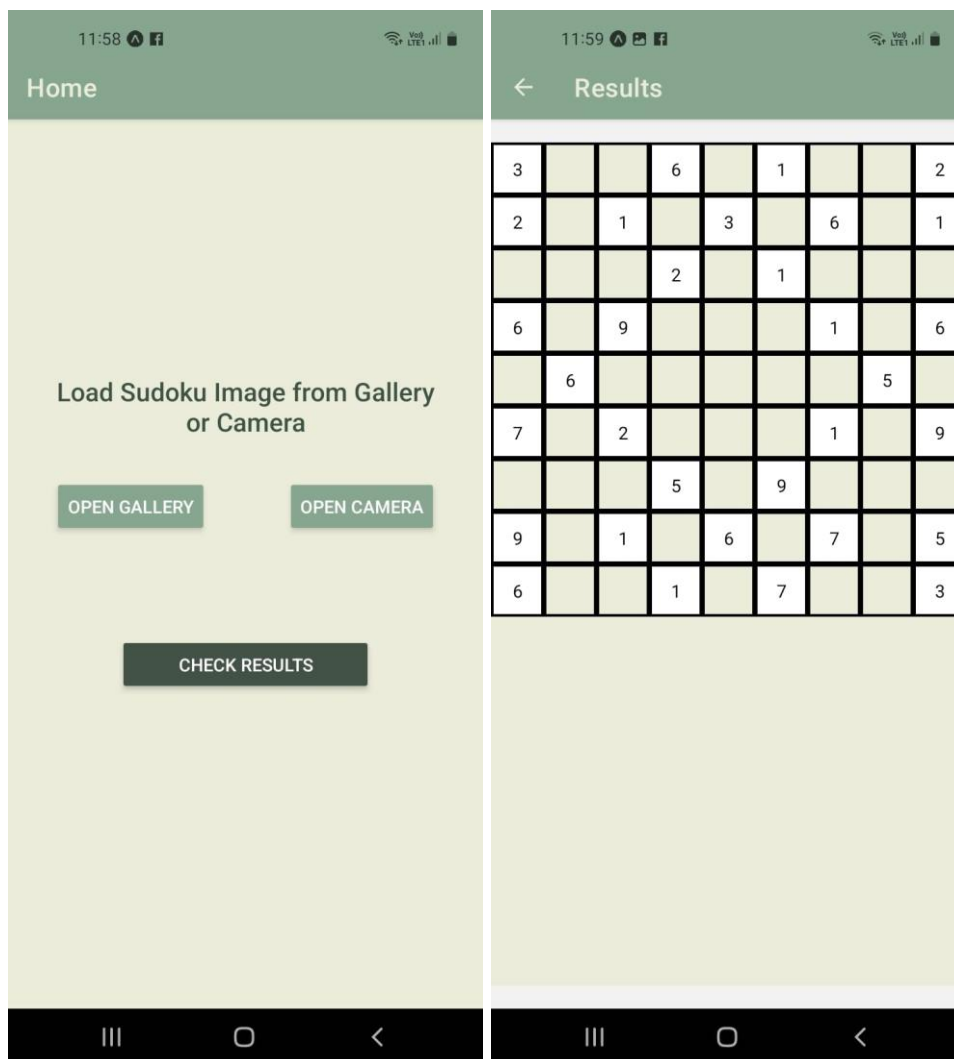
Logs for your project will appear below. Press Ctrl+C to exit.
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Ana\Desktop\14052023> cd backend
PS C:\Users\Ana\Desktop\14052023\backend> flask run -h 0.0.0.0
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.5.25:5000
Press CTRL+C to quit
```

Slika 32 Terminali nakon pokretanja sučelja (lijevo) i servera (desno)



Slika 33 Izgled ekrana za učitavanje slike (lijevo) i ekrana za provjeru rezultata (desno). Ekran za učitavanje slike sadrži tri dugmeta: dugme za otvaranje galerije "open gallery", dugme za otvaranje kamere "open camera", te dugme za prelazak na ekran rezultata "check results". Ekran za provjeru rezultata sadrži Sudoku mrežu s pronađenim brojkama na slici.

Mobilna aplikacija napravljena je u svrhu demonstracije primjene konvolucijskih neuronskih mreža u OCR sustavu. U kontroliranom okruženju, sa jednim ispitivačem na jednom uređaju, pokazala je zadovoljavajuće rezultate. Kako bi se saznala prava efikasnost i primjenjivost aplikacije, za daljnje testiranje trebalo bi angažirati više ispitanika na različitim mobilnim uređajima, što za lokalnu aplikaciju nije bilo moguće ostvariti. Da se aplikacija osposobi za uporabu preko različitih mreža, korisnici bi ju mogli upotrijebiti za vlastito optičko prepoznavanje znakova, pa čak i za igranje Sudoku igre, naravno uz implementaciju koda za igru.

Zaključak

U ovom radu opisano je optičko prepoznavanje znakova i kako se razvilo u ono što danas znamo kao OCR sustav. Prvotno su postojale jednostavne šablone koje bi svjetlom iščitavale znakove, no tokom desetljeća proces se digitalizirao i postao bitan aspekt u automatizaciji područja digitalizacije dokumenata. Danas, primjena OCR sustava integrirala se u razne industrije i aplikacije, te je promijenila način na koji vršimo interakciju s tekstom. Umjetna inteligencija potpomogla je poboljšavanju OCR sustava u prepoznavanju. Umjetne neuronske mreže učinile su OCR sustave prilagodljivijima te su omogućile bržu obradu velike količine informacija. Konvolucijske neuronske mreže, inspirirane ljudskim osjetilom vida, pokazale su se izvrsnim alatom za optičko prepoznavanje tiskanih ili ručno pisanih znakova zbog mogućnosti prilagodbe i točnosti. Kako tehnologija još uvijek rapidno napreduje, pogotovo u aspektu umjetne inteligencije, tako će i OCR sustavi pokazivati još veću točnost, brzinu, i prilagodljivost.

U primjeni teorije ovog rada razvijena je mobilna aplikacija za optičko prepoznavanje znakova unutar Sudoku mreže. U Sudoku mreži dobro obučene konvolucijske neuronske mreže mogu s lakoćom prepoznati i klasificirati brojeve u pojedinim ćelijama. OCR sustav s primjenom konvolucijskih neuronskih mreža razlučuje brojeve iz mreže i osigurava točnost interpretacije brojki. Točnost prepoznavanja brojki u aplikaciji približno je jednaka 98.5%, što je zadovoljavajuća stopa točnosti. Ova aplikacija ima potencijal kasnijeg osposobljavanja na rad ne samo lokalnih, nego svih mreža. Isto tako, aplikacija nudi mogućnost implementacije koda za samo rješavanje Sudoku igre, što bi joj dodalo još jednu praktičnu funkciju.

Tablica slika

Slika 1 Odnos umjetne inteligencije, strojnog učenja, i dubokog učenja. Duboko učenje je potpolje strojnog učenja, a strojno učenje potpolje umjetne inteligencije.	9
Slika 2 Prikaz pomicanja filtera dimenzija 3x3 s korakom 1 [13].....	11
Slika 3 Dopuna nulama na jednom sloju [13].....	11
Slika 4 Primjer sažimanja maksimumom na matrici dimenzija 4x4 s filterom 2x2 i korakom 2. Sažimanjem se dobiva matrica dimenzija 2x2 s maksimalnim vrijednostima svakog polja. [15]	11
Slika 5 Datoteke u frontend-u.....	14
Slika 6 Kod datoteke App.js	15
Slika 7 Kod funkcije „openGallery“ za učitavanje slike iz galerije.....	16
Slika 8 Kod funkcije „openCamera“ za učitavanje fotografirane slike	17
Slika 9 Kod funkcije „toServer“ za komunikaciju s backend serverom.....	18
Slika 10 Funkcija "checkResult" u ekranu "UploadScreen" koja omogućava navigaciju na ekran „ResultScreen“ i slanje brojki Sudoku mreže.	18
Slika 11 Kod funkcije "ResultScreen" za prikaz prepoznatih brojki u Sudoku mreži.....	19
Slika 12 Datoteke u backend-u	20
Slika 13 Kod za stvaranje Flask aplikacije i određivanje domena s dopuštenjem slanja zahtjeva	20
Slika 14 Kod za definiranje rute i HTTP zahtjeva, te dohvaćanje i pohranjivanje slike na disk	21
Slika 15 Rezultati nastali primjenom Gaussovog zaglađivanja s različitim veličinama jezgre (s lijeva): (1,1), (5,5) (9,9)	22
Slika 16 Slika nakon određivanja praga i binarizacije (lijevo), te nakon invertiranja binarne slike (desno)	23
Slika 17 Ulazna slika (lijevo) i slika nakon pretprocesiranja (desno)	23
Slika 18 Kod procesa pretprocesiranja	24
Slika 19 Kod za pronalaženje vanjskih granica Sudoku mreže	24
Slika 20 Vizualizacija pronađenih granica Sudoku mreže (lijevo), te izrezana i zakrivljena Sudoku mreža (desno)	25
Slika 21 Kod za izračun granica pojedinih ćelija	26
Slika 22 Kod za dohvaćanje sličice ćelije i definiranje središnjeg dijela ćelije	26

Slika 23 Kod za pronalazak najvećeg povezanog područja i njegovih graničnih koordinata .	27
Slika 24 Pregled ekstrahiranih ćelija i brojki u njima.....	28
Slika 25 Struktura modela neuronske mreže	29
Slika 26 Primjer podataka u bazi MNIST [17]	29
Slika 27 Grafovi prikaza gubitka (gore) i točnosti (dolje) pri treniranju i testiranju po epohama	31
Slika 28 Matrica konfuzije testirane mreže	32
Slika 29 Graf koji prikazuje točnost prepoznavanja podataka za treniranje i testiranje na modelu neuronske mreže sa 100 epoha i veličinom serije 200	33
Slika 30 Kod za prepoznavanje brojki Sudoku mreže pomoću treniranog modela.....	34
Slika 31 Prepoznate brojke s ulazne slike.....	34
Slika 32 Terminali nakon pokretanja sučelja (lijevo) i servera (desno)	35
Slika 33 Izgled ekrana za učitavanje slike (lijevo) i ekrana za provjeru rezultata (desno). Ekran za učitavanje slike sadrži tri dugmeta: dugme za otvaranje galerije "open gallery", dugme za otvaranje kamere "open camera", te dugme za prelazak na ekran rezultata "check results". Ekran za provjeru rezultata sadrži Sudoku mrežu s pronađenim brojkama na slici.	36

Literatura

- [1] »OCR,« Brittanica, 11. Veljača 2023.. [Mrežno]. Available: <https://www.britannica.com/technology/OCR>. [Pokušaj pristupa 7. rujan 2023.].
- [2] D. Berchmans i S. S. Kumar, »Optical character recognition: An overview and an insight,« u *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, 2014., pp. 1361-1365.
- [3] A. Chaudhuri, P. Badelia i S. K. Ghosh, »Optical Character Recognition Systems for Different Languages with Soft Computing,« u *Studies in Fuzziness and Soft Computing*, 2017., pp. 35-37.
- [4] M. Karić, Z. Krpić i G. Martinović, »Optičko prepoznavanje znakova na grid i višejezgrenim platformama – analiza performansi,« u *Tehnički vjesnik* 20, 2013., pp. 647-653.
- [5] N. Islam, Z. Islam i N. Noor, »A Survey on Optical Character Recognition System,« u *Journal of Information & Communication Technology*, 2016..
- [6] K. Hamad i M. Kaya, »A Detailed Analysis of Optical Character Recognition Technology,« u *International Journal of Applied Mathematics Electronics and Computers*, 2016..
- [7] O. Abiodun, A. Jantan, A. Omolara, K. Dada, N. Mohamed i H. Arshad, »State-of-the-art in artificial neural network applications: A survey,« u *Heliyon*, 2018..
- [8] A. Dongare, R. Kharde i A. Kachare, »Introduction to Artificial Neural Network,« u *International Journal of Engineering and Innovative Technology (IJEIT)*, 2012..
- [9] S. Dumančić, »Neuronske mreže,« u *Diplomski rad*, Osijek, Sveučilište Josipa Jurja Strossmayera u Osijeku, Odjel za fiziku, 2014..
- [10] Ž. Ujević Andrijić, »OSVJEŽIMO ZNANJE: Umjetne neuronske mreže,« u *Kemija u industriji: Časopis kemičara i kemijskih inženjera Hrvatske* 68, 2019., pp. 219-220.

- [1 S. Singh, »Feedforward and Feedback Neural Networks: What's the difference?,«
1] LinkedIn, 19 Listopad 2022.. [Mrežno]. Available:
<https://www.linkedin.com/pulse/feedforward-feedback-neural-networks-whats-difference-sreeshti-singh#:~:text=The%20feedforward%20neural%20network%20has,of%20a%20feedback%20network%20system..> [Pokušaj pristupa 09. Rujan 2023.].
- [1 J. Kelleher, »Deep learning,« u *MIT press*, 2019., p. 1.
2]
- [1 S. Albawi, T. Mohammed i S. Al-Zawi, »Understanding of a convolutional neural
3] network,« u *2017 International Conference on Engineering and Technology (ICET)*,
2017., pp. 1-6.
- [1 M. Dabović i I. Tartalja, »Duboke konvolucijske neuronske mreže–koncepti i aktuelna
4] istraživanja,« u *Zbornik 61. Konferencije za elektroniku, telekomunikacije, računarstvo, automatiku i nuklearnu tehniku, ETRAN 2017, Kladovo, 05. do 08. juna 2017*, 2017..
- [1 C. S. Wiki, »Max-pooling / Pooling,« 27 Veljača 2018.. [Mrežno]. Available:
5] https://computersciencewiki.org/index.php/Max-pooling/_/_Pooling. [Pokušaj pristupa 11
Rujan 2023.].
- [1 K. Džomba, »Konvolucijske neuronske mreže,« u *Diplomski rad*, Sveučilište u Zagrebu,
6] Prirodoslovno-matematički fakultet, 2018., p. 24.
- [1 Suvanjanprasai, »File:MnistExamplesModified.png,« Wikipedia, 24 Svibanj 2023..
7] [Mrežno]. Available: <https://en.wikipedia.org/wiki/File:MnistExamplesModified.png>.
[Pokušaj pristupa 14 Rujan 2023.].