SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 131

STROJNO RASPOZNAVANJE STUDENTSKIH IDENTIFIKACIJSKIH BROJEVA IZ MATRIČNOG PREDLOŠKA

Filip Pavičić

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 131

STROJNO RASPOZNAVANJE STUDENTSKIH IDENTIFIKACIJSKIH BROJEVA IZ MATRIČNOG PREDLOŠKA

Filip Pavičić

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

DIPLOMSKI ZADATAK br. 131

Pristupnik:	**Filip Pavičić (0036514970)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	izv. prof. dr. sc. Marko Čupić

Zadatak: **Strojno raspoznavanje studentskih identifikacijskih brojeva iz matričnog predloška**

Opis zadatka:

Automatsko raspoznavanje studentskih identifikacijskih brojeva (poput JMBAG-a) korisno je prilikom automatizirane obrade studentskih ispita. Pri tome se identifikator može unositi na različite načine, a jedan od njih je uporabom matričnog predloška. U okviru ovog diplomskog rada potrebno je pripremiti početni skup primjeraka za učenje te na temelju tog skupa napraviti prototipnu implementaciju modela strojnog učenja koji obavlja posao raspoznavanja identifikacijskog broja na temelju slike popunjenog matričnog predloška. Napravljeni model potrebno je vrednovati, a pripremljeni početni skup obogatiti dodatnim primjercima koji trebaju biti automatski generirani i dovoljno realistični. Potrebno je citirati svu korištenu literaturu.

Rok za predaju rada: 23. lipnja 2023.

SADRŽAJ

1. Uvod

U današnjem obrazovnom sustavu sve veći broj studenata predstavlja značajne izazove u provođenju provjere znanja i ispravljanja ispita. Veliki obujam posla povezan s ručnim unosom pojedinačnih rezultata u sustav, pri čemu ispitivači moraju samostalno identificirati osobne podatke studenata s ispita, nije samo vremenski zahtjevno, već je i podložno pogreškama. Ovaj diplomski rad oslanja se na ideju iz članka [5] iz 2011. godine u kojem je predložen zapis studentskog identifikacijskog broja u matrični predložak. U ovome radu, prepoznavanje unosa u matrični zapis postiže se pomoću treniranog modela dubokog učenja, koji pokazuje visoku preciznost u identificiranju unesenog jedinstvenog matičnog broja akademskog građanina (JMBAG-a), pomoću kojeg se automatski određuje identitet studenta na ispitu.

Umjesto konvencionalne metode ručnog unosa osobnih podataka, studenti pojedinačno upisuju svaku znamenku svog JMBAG-a sjenčanjem odgovarajućeg kvadrata na za to predviđenom dijelu ispitnog papira. Ovakav sustavan pristup unosu podataka osigurava dosljednost, bez obzira na rukopis pojedinca, a istovremeno omogućuje automatsku detekciju studenta kroz neuronsku mrežu. Nadalje, ova metodologija olakšava identifikaciju netočnih unosa u matrični prikaz. U slučaju da student ne ispoštuje pravila unosa osobnog identifikacijskog broja u predložak matrice, ispitivač ima mogućnost ručno identificirati studenta i ispraviti grešku.

Ključno je naglasiti da ovaj pristup nije ograničen samo na obrazovni sektor, već ima potencijal za implementaciju u raznim industrijama u kojima korisnici posjeduju jedinstvene identifikacijske brojeve, poput zdravstva ili administracijskih sustava. Integracija ove tehnologije nudi intuitivnu i učinkovitu metodu identifikacije u različitim domenama.

U nastavku rada bit će detaljno predstavljena implementacija prepoznavanja studenskih osobnih brojeva. Bit će dan opis problema te napravljena analiza rješenja iz prethodno spomenutog članka. Također, bit će predstavljene prednosti i nedostatci pristupa iz članka u usporedbi s pristupom predloženim u ovome radu. Nadalje, bit će predstavljen način prikupljanja, anotiranja i provjere skupa podataka koji je korišten

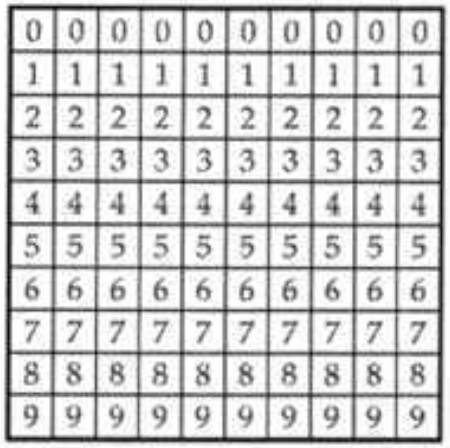
za treniranje mreže, te detaljno objašnjena arhitektura mreže koja se koristi za prepoznavanje identifikacijskih brojeva. U radu će također biti predstavljena mogućnosti automatiziranog obogaćivanja skupa podataka korištenjem tehnika argumentiranja podataka te će biti analiziran njihov utjecaj na performanse modela. Bit će predstavljeni rezultati evaluacije različitih isprobanih mreža koje se razlikuju po veličini i složenosti, bit će napravljena usporedba kvalitete modela pomoću izabranih metrika te će biti opisane mogućnosti daljnjeg unaprijeđenja sustava. Na kraju će biti opisana računalna aplikacija koja koristi naučeni model te pruža mogućnost predikcije studenskih JMBAG-ova na temelju ulazne slike.

Istražujući navedeni pristup identifikaciji studenata, ovaj diplomski rad nastoji pojednostaviti i ubrzati ispitni proces, poboljšati točnost i potaknuti učinkovitost u obrazovanju, s potencijalnom primjenom u drugim sektorima koji se oslanjaju na jedinstvenu identifikaciju korisnika.

2. Usporedba pristupa detekciji studentskog identifikacijskog broja iz matričnog zapisa

2.1. Opis problema

Kao što je navedeno u članku [5], ispiti na sveučilištima često su oblikovani kao skupovi zadataka višestrukog izbora [6][8][10], a automatsko ocjenjivanje često se koristi kao efikasna metoda ispravljanja ispita [3][4]. U članku se naglašava razlika između standardnih, najavljenih ispita, kao što su ispitni rokovi, te kratkih provjera znanja koje se provode unutar termina predavanja. Kratke provjere trebaju biti provedene brzo i efikasno u trajanju od 5 do 10 minuta, kako bi ostalo dovoljno vremena za redovno predavanje. Podjela ispita pojedinačno svakom studentu (primjerice, ispit koji sadrži bar kôd povezan s identitetom studenta) nije praktična jer znatno produljuje postupak podjele ispita i otežava brzu provedbu provjere znanja. Stoga je u članku predstavljen inovativan pristup identifikaciji studenata tijekom ispita i provjera znanja koji omogućuje ručni unos vlastitog studentskog identifikacijskog broja uz održavanje automatske čitljivosti. Predloženi pristup temelji se na konceptu prepoznavanja jedinstvenog matičnog broja akademskog građana, unutar posebno dizajniranog matričnog predloška (slika 2.1).



Slika 2.1: Primjer nepopunjenog matričnog predloška

2.2. Rješenje predstavljeno u članku za rješavanje problema prepoznavanja JMBAG-a unutar matričnog predloška

Prepoznavanje JMBAG-a unutar predloška matrice postiže se kroz proces koji se sastoji od više etapa opisanih u članku. Prvo, precizno pozicioniranje matrice određuje se pomoću različitih tehnika obrade slike. Dok je matrica konceptualno smještena na istom položaju na svakom ispitu, postupci ispisa i skeniranja mogu dovesti do pomicanja, skaliranja ili rotacije, zahtijevajući razvoj robusnog mehanizma za otkrivanje položaja matrice. Ovaj proces detekcije oslanja se na identifikaciju horizontalnih i vertikalnih crnih linija, kao što je objašnjeno u članku, što omogućuje točno određivanje vanjskih granica matrice. Nakon što su vanjski rubovi uspješno otkriveni, algoritam nastavlja locirati unutarnje rubove te se na temelju toga točno određuje pozicija svake ćelije unutar matrice. Na posljetku, izračunavanjem omjera crnih piksela prema ukupnom broju piksela unutar ćelije, algoritam odlučuje koje su ćelije označene. U članku su dodano objašnjene tehnike koje pomažu pri detekciji te omogućavaju da algoritam s 100%-tnom uspješnošću identificira JMBAG-ove na 296 primjera.

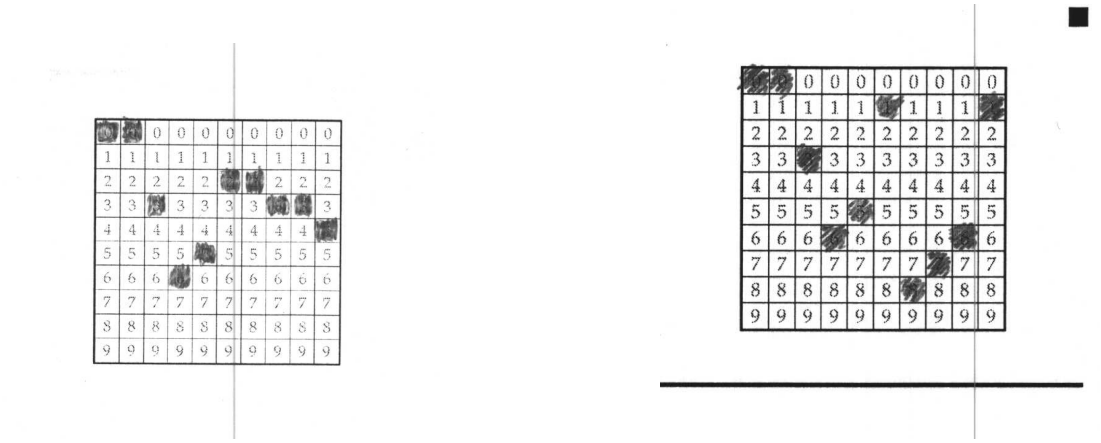
2.3. Potencijalni problemi s predloženim pristupom

Implementacija matričnog predloška i tehnika detekcije studenskih JMBAG-ova, kako je objašnjeno u članku, omogućila je automatizirano ispravljanje kratkih provjera zna-

nja. Međutim, kako se skup podataka i slika popunjenih matričnih predložaka povećavao kroz primjenu u nastavi, pojavili su se primjeri koje su studenti točno označili, ali ih sustav nije uspio točno prepoznati. U nastavku će biti navedene situacije i primjeri koje potencijalno mogu dovesti do pogrešne detekcije studenskih JMBAG-ova.

2.3.1. Ispis i skeniranje predložaka

Prilikom ispisa i skeniranja ispita mogu se pojaviti određene mrlje na predlošku u obliku crta, što može uzrokovati pogrešnu detekciju. Slika 2.2 pokazuju kako tragovi mrlja prekrivaju matrični predložak.



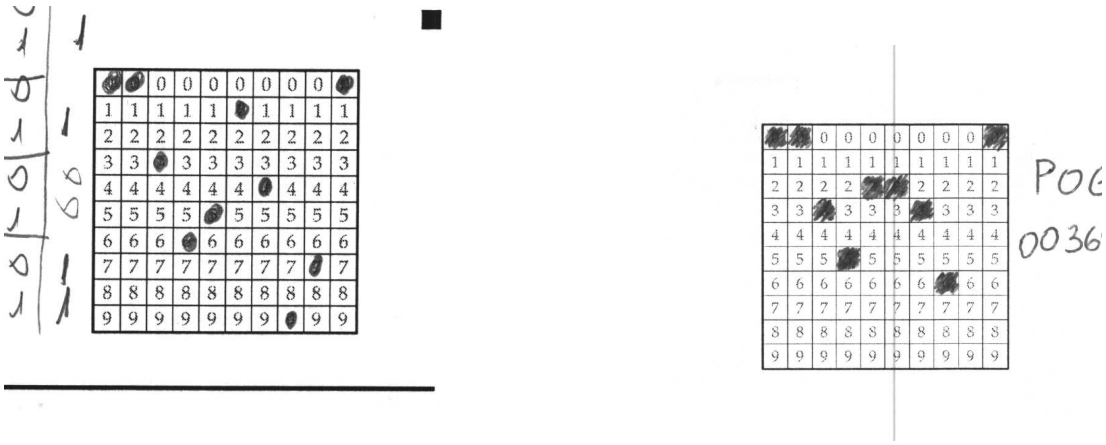
Slika 2.2: Primjeri matričnih predložaka koji sadrže mrlje

2.3.2. Pisanje i crtanje oko matričnog predloška

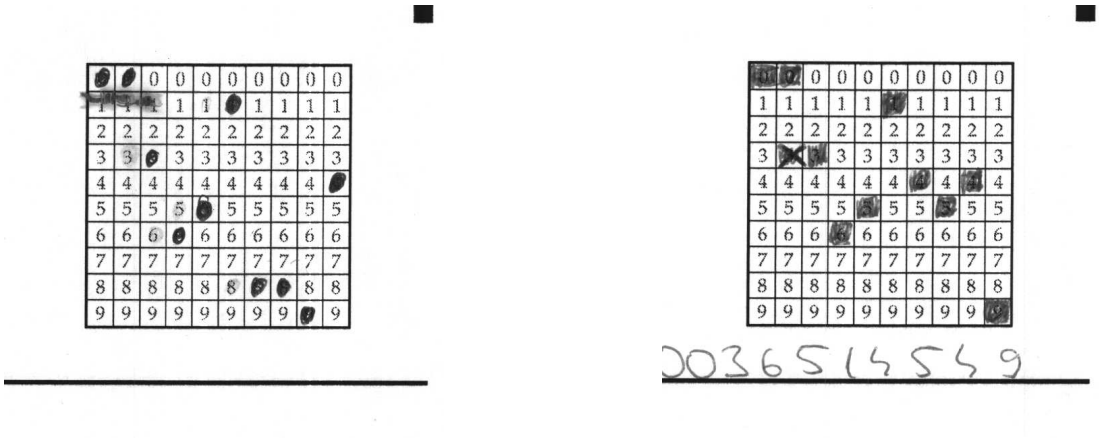
Pisanje i crtanje oko matričnog predloška može predstavljati izazove u otkrivanju vanjskih granica matrice, što kasnije može rezultirati pogrešnim detektiranjem oznaka. Studenti često koriste margine ispita za dodatne bilješke ili izračune, zbog ograničenog prostora (slika 2.3). Takva pisanja i crtanja mogu smanjiti jasnoću predloška.

2.3.3. Pogrešaka ili brisanje unutar matričnog predloška

U žurbi i stresu, studenti često naprave pogreške prilikom unosa JMBAG-a u matrični predložak. Te se pogreške mogu manifestirati kao tragovi brisanja ili nepotpunog zacrnjenja (slika 2.4), što dovodi do problema u točnom otkrivanju JMBAG-a. Prisutnost takvih pogrešaka ili propusta unutar predloška matrice predstavlja potencijalne poteškoće u kasnijem procesu prepoznavanja.



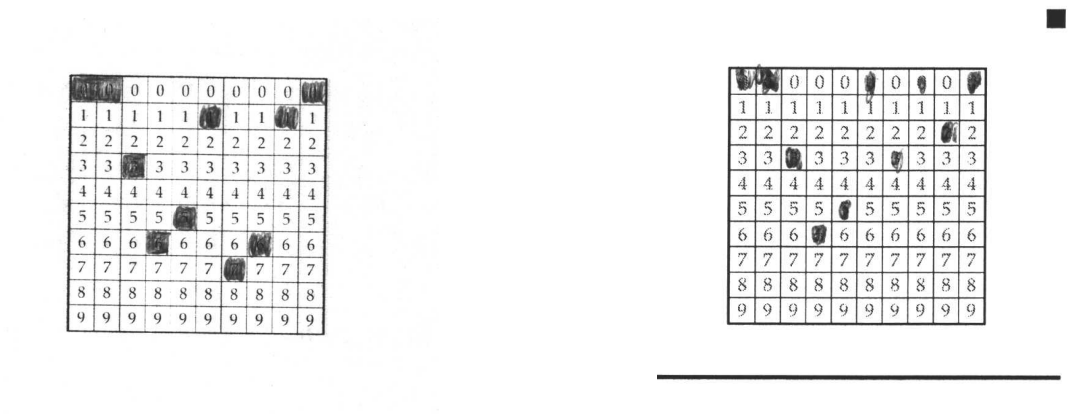
Slika 2.3: Primjeri pisanja i crtanja oko matričnog predloška



Slika 2.4: Primjeri brisanja i pogrešaka unutar matričnog predloška

2.3.4. Promjena izgleda kratkih provjera

Kroz vrijeme, izgled kratkih provjera može se mijenjati ili prilagođavati. Ako vizualne promjene kratke provjere utječu na područje blizu matričnog predloška, logika algoritma treba biti prilagođen tim promjenama. To može povećati kompleksnost algoritma kako bi se osigurala ispravna detekcija za sve primjere kratkih provjera. Slika 2.5 pokazuje promjenu dizajna u kojem se dodaje horizontalna crta ispod matričnog predloška.



Slika 2.5: Primjer promjena dizajna matričnog predloška

2.4. Pristup rješavanju problema prepoznavanja JMBAG-a unutar matričnog predloška temeljen na modelu dubokog učenja

Analizom prethodno navedenih izazova koji se javljaju pri detektiranju JMBAG-ova korištenjem pristupa temeljenog na skupu pravila, ukazala se potreba za razvojem novog modela koji će biti otporan na navedene izazove. U ovom diplomskom radu predstavljen je model dubokog učenja (engl. *deep learning model*) dizajniran za postizanje vrlo preciznog i robusnog prepoznavanja JMBAG-a unutar predloška matrice. U vrijeme pisanja prethodno spomenutog članka iz 2011. godine, modeli dubokog učenja još su bili u početnim fazama razvoja, a njihova široka primjenjivost u raznim domenama računalne znanosti još nije bila u potpunosti realizirana. Modeli dubokog učenja obično zahtijevaju veliki broj primjera za učenje, čije prikupljanje predstavlja vrlo dugotrajan i skup proces. Međutim, za potrebe ovoga rada korišten je gotov skup podataka koji se sastoji od označenih matričnih predložaka i odgovarajućih JMBAG oznaka akumuliran tijekom provedbe kratkih provjera znanja na kojima su studenti ispunjavali matrični predložak.

Temeljna razlika između pristupa temeljenog na skupu pravila opisanog u članku i predloženog modela dubokog učenja leži u njihovim odgovarajućim mehanizmima za otkrivanje ključnih značajki svojstvenih problemu. Dok implementacija opisana u članku ručno definira pravila i koncepte na kojima se temelji ekstrakcija značajki, model dubokog učenja samostalno uči izdvajati diskriminirajuće značajke na temelju označenih primjera popunjenih matrica.

2.5. Prednosti i nedostaci oba pristupa

Usporedba opisanih pristupa prikazana je u tablici 2.1, ističući njihove prednosti i ograničenja. Model dubokog učenja pokazuje značajne prednosti zbog svoje inherentne sposobnosti da autonomno izdvaja relevantne značajke i svoje sposobnosti razlikovanja između značajnih informacija i šuma unutar slike. Koristeći svoju slojevitu arhitekturu, model može učinkovito naučiti ključne aspekte potrebne za točnu detekciju, čime se ublažavaju gore navedeni izazovi. Nasuprot tome, model temeljen na pravilima nudi razinu izražajnosti koja olakšava preciznu identifikaciju čimbenika koji uzrokuju pogreške u predviđanju. Ovaj atribut dopušta potencijalna poboljšanja u skupu pravila ili prilagodbe matričnog predloška, omogućujući lakše usavršavanje izvedbe.

Tablica 2.1: Usporedba rješenja temeljenog na eksplicitnim pravilima i arhitekture koja koristi model dubokog učenja

Rješenje temeljeno na pravilima	Arhitektura temeljena na modelu dubokog učenja
Pravila za ekstrakciju značajki definira programer	Mreža samostalno uči bitne značajke koje karakteriziraju unos u matrični predložak
Promjene matričnog predloška i izazove koji se javljaju u stvarnim situacijama potrebno je ručno analizirati i eventualno doraditi pravila	Model koji je treniran na raznovrsnim primjerima sposoban je razlikovati šum od matričnog predloška
Potrebno je znanje eksperta za definiranje kvalitetnih pravila	Potreban je veliki broj primjera za treniranje
Vrlo jasan i ekspresivan način odlučivanja prilikom predikcije	Model se ponašao kao crna kutija, nije moguće jasno definirati način odlučivanja
Prilikom promjena potrebno je refaktoriranje kôda i pravila	Promjene zahtijevaju ponovno treniranje modela, potrebno je anotirati nove primjere
Pristup ne zahtjeva velike računalne resurse	Ovisno o veličini modela, proces treninga i predikcije može biti vremenski zahtjevan

3. Korištene tehnologije i alati

U ovom poglavlju bit će predstavljene tehnologije i alati koji su korišteni u izradi ovog diplomskog rada. Najprije će biti predstavljene tehnologije koje su korištene u procesu anotiranja podataka, izrade i treniranje modela, a zatim će biti predstavljeni alati korišteni za razvoj računalnog programa koji koristi model za stvaranje predikcija.

3.1. Tehnologije i alati korišteni u procesu anotiranja podataka, izradi i treniranju modela

U ovome radu korišten je programski jezik Pythion za proces označavanja podataka, izradu i treniranje modela. Alati za označavanje podataka su implementirani kao Python programi koji se pokreću iz komandne linije, dok se izrada i trening provode unutar interaktivnog okruženja Jupyter bilježnice (engl. *Jupyter Notebook*).

3.1.1. Programski jezik Python

Unatoč tome što je skriptni jezik koji ne pokazuje istu razinu brzine izvršavanja kôda kao drugi programski jezici, Python[1] nudi veliki broj programskih biblioteka (engl. *software library*) koje iskorištavaju učinkovite matematičke i složene operacije implementirane u jezicima kao što su C i C++. Posljedično, ove biblioteke, uključujući Numpy, TensorFlow, Keras i Scikit-learn, omogućuju brza i učinkovita izračunavanja. Popularnost Pythona kao programskog jezika za duboko učenje, strojno učenje (engl. *machine learning*), računalni vid (engl. *computer vision*) i razne druge primjene u području umjetne inteligencije može se pripisati njegovoj kombinaciji jednostavne i čitljive sintakse s moćnim i brzim programskim bibliotekama.

[1] `https://www.python.org/`

3.1.2. Računalno okruženje Jupyter Notebook

Jupyter Notebook[2] popularno je interaktivno računalno okruženje koje se koristi u raznim područjima, uključujući znanost o podacima, istraživanje i obrazovanje. Pruža fleksibilnu platformu za pisanje i izvođenje kôda, dokumentiranje radnih procesa i vizualizaciju podataka. Sučelje Jupyter Notebook-a omogućuje korisnicima organiziranje kôda, teksta i vizualizacija u ćelije, promičući iterativni i istraživački pristup analizi. U ovome radu, Jupyter Notebook korišten je za provođenje eksperimenata prilikom podešavanja parametara modela i vizualizaciju prilikom učitavanja i augumentiranja podataka. Izvršavanje kôda u ćelijama omogućuje znatno brže pokretanje dijelova kôda, što omogućuje lakšu analizu zasebnih logičkih cjelina kôda.

3.1.3. Knjižnica TensorFlow i aplikacijsko programsko sučelje Keras

TensorFlow[3], razvijen od strane Google-a, otvorena je biblioteka koja pruža sveobuhvatan okvir za izgradnju i implementaciju modela strojnog učenja. Pruža iznimno fleksibilnu i skalabilnu arhitekturu koja omogućuje učinkovito računanje na CPU-u i GPU-u. TensorFlow koristi računalne grafove, gdje čvorovi predstavljaju matematičke operacije, a bridovi označavaju protok podataka, omogućujući učinkovito paralelno procesiranje i optimizaciju. Dodatno, TensorFlow nudi širok spektar unaprijed izgrađenih slojeva neuronskih mreža, aktivacijskih funkcija i algoritama optimizacije, olakšavajući razvoj složenih modela dubokog učenja.

S druge strane, Keras[4] je aplikacijsko programsko sučelje (engl. *Application Programming Interface, API*) visoke razine za neuronske mreže napisano u Pythonu. Izgrađen je nad TensorFlow programskom bibliotekom i pruža pojednostavljeno sučelje za konstrukciju modela dubokog učenja. Keras pruža jednostavnost korištenja i lako eksperimentiranje, što ga čini posebno prikladnim za početnike i istraživače kojima je potrebna jednostavna izgradnja prototipa i evaluacija modela. Nudi modularni i intuitivni API koji omogućuje laku integraciju različitih arhitektura mreža, funkcija gubitaka i optimizatora. Keras također podržava sekvencijalni i funkcionalni dizajn modela, omogućavajući stvaranje složenih struktura mreža.

U ovome radu, TensorFlow i Keras korišteni su za izradu i definiranje modela. Zbog arhitekture mreže koja nije sekvencijalna, što znači da slojevi nisu isključivo po-

[2] `https://jupyter.org/`
[3] `https://www.tensorflow.org/`
[4] `https://keras.io/`

vezani s prethodnim i sljedećim slojem, model je dizajniran pomoću Keras funkcionalnog dizajna modela. Funkcionalni dizajn modela omogućava povezivanje nesusjednih slojeva unutar mreže. Detaljna implementacija modela i programski kôd za treniranje mreže biti će predstavljeni u kasnijim poglavljima.

3.1.4. Albumentations

Programska biblioteka Albumentations[5] je moćan alat koji se široko koristi u istraživanjima i primjenama računalnog vida za augmentaciju slika. Pruža sveobuhvatnu kolekciju raznovrsnih i prilagodljivih tehnika augmentacije, uključujući geometrijske transformacije, manipulacije bojama i napredne filtre slika. Svojom učinkovitom implementacijom i integracijom s popularnim programski knjižnicama dubokog učenja, Albumentations olakšava proces augmentacije, poboljšava raznolikost skupa podataka te unaprjeđuje generalizaciju i robusnost modela strojnog učenja. Njegova svestranost i učinkovitost čine ga vrijednim alatom za istraživače u području računalnog vida.

3.1.5. Tkinter

Biblioteka Tkinter[6] široko je korišten alat za razvoj grafičkog korisničkog sučelja (engl. *graphical user interface, GUI*) unutar programskog jezika Python. Sa svojim raznolikim skupom funkcionalnosti, Tkinter pruža intuitivan i učinkovit okvir za stvaranje interaktivnih i vizualno privlačnih desktop aplikacija. Svojim objektno orijentiranim pristupom, Tkinter olakšava stvaranje i upravljanje različitim GUI elementima, kao što su prozori, gumbi, izbornici i tekstni okviri. Nadalje, Tkinter nudi opsežnu podršku za programiranje vođeno događajima, omogućujući interakciju korisnika s funkcionalnošću aplikacije. U ovome radu, Tkinter je korišen za izradu GUI aplikacije čija je svrha pružiti jednostavan alat za anotiranje podataka i provjeru postojećih anotacija. Korištenje Tkinter-a omogućilo je stvaranje intuitivnog i jednostavnog sučelja, pružajući korisnicima jednostavnu navigaciju kroz označene podatke koristeći isključivo tipkovnicu.

[5] `https://albumentations.ai/`

[6] `https://docs.python.org/3/library/tkinter.html`

3.2. Tehnologije i alati korišteni za razvoj računalnog programa koji koristi model za stvaranje predikcija

Zbog činjenice da je rješenje predloženo u članku na koji se nadovezuje ovaj diplomski rad napisano u programskom jeziku Java, računalni program koji koristi izrađeni model također je napisan u Javi. Ta činjenica omogućava kompatibilnost i primjenjivost rješenja predstavljenog u ovom radu s već postojećim dijelovima kôda koji se koriste za detekciju studenskih JMBAG-ova na kratkim provjerama znanja. Također, korištene su programske biblioteke Deeplearning4j i Nd4j za Javu koje omogućavaju učitavanje modela dubokog učenja napisanih u drugim programskim jezicima.

3.2.1. Java

Java[7], široko korišteni programski jezik, poznat po svojoj platformskoj neovisnosti i robusnosti, pruža svestranu okolinu za izgradnju pouzdanih i skalabilnih aplikacija. Jedna od ključnih prednosti Jave je njena opsežna standardna biblioteka, koja pruža širok spektar unaprijed izgrađenih klasa i funkcija za uobičajene programske zadatke. Ova biblioteka pojednostavljuje razvojni proces nudeći gotove komponente za zadatke, poput rukovanja datotekama, umrežavanja i manipulacije podacima. Kompatibilnost Jave s brojnim platformama i operacijskim sustavima omogućuje programerima stvaranje aplikacija koje se mogu izvoditi na različitim uređajima. Nadalje, podrška zajednice i obilje dokumentacije olakšavaju programerima pronalazak resursa i učinkovito rješavanje problema.

3.2.2. Deeplearning4j i Nd4j

DeepLearning4j (DL4j) i ND4j[8] su moćne biblioteke koje se koriste u području dubokog učenja, posebno u kontekstu Java baziranih aplikacija. DL4j pruža sveobuhvatnu podršku za arhitekture dubokih neuronskih mreža, nudeći širok spektar ugrađenih algoritama i alata za treniranje, evaluaciju i implementaciju modela dubokog učenja. ND4j, s druge strane, služi kao računalni okvir (engl. *computational framewor*) za DL4j. To je biblioteka visokih performansi posebno dizajnirana za višedimenzionalne nizove i numeričke operacije. Učinkovita implementacija ND4j-a podržava paralelno izvođe-

[7] `https://www.java.com/en/`

[8] `https://deeplearning4j.konduit.ai/`

nje na CPU i GPU, omogućujući ubrzano treniranje i izvođenje na velikim skupovima podataka. Njegova opsežna kolekcija matematičkih funkcija i operacija pojednostavljuje kompleksne izračune i poboljšava optimizaciju performansi. Još jedna značajna karakteristika Deeplearning4j i ND4J biblioteka je njihova kompatibilnost s modelima napisanima u Keras-u, što omogućava integraciju Keras modela u Java projekte.

4. Skup podataka

Problem prepoznavanja studenskih JMBAG-ova iz matričnog predloška predstavlja problem klasifikacije s više oznaka (engl. *multi-label classification*). Svaka pojedinačna znamenka predstavlja jednu oznaku koju treba ispravno detektirati. Klasifikacija s više oznaka spada u probleme nadziranog strojnog učenja (engl. *supervised machine learning*) u kojima se primjeri iz skupa podataka sastoje od ulaznih uzoraka zajedno s odgovarajućim oznakama. Konkretno, u ovome radu, ulazni uzorak je slika dijela skeniranog ispita gdje se nalazi matrični predložak, dok oznaka predstavlja JMBAG koji odgovara unosu u matrični predložak.

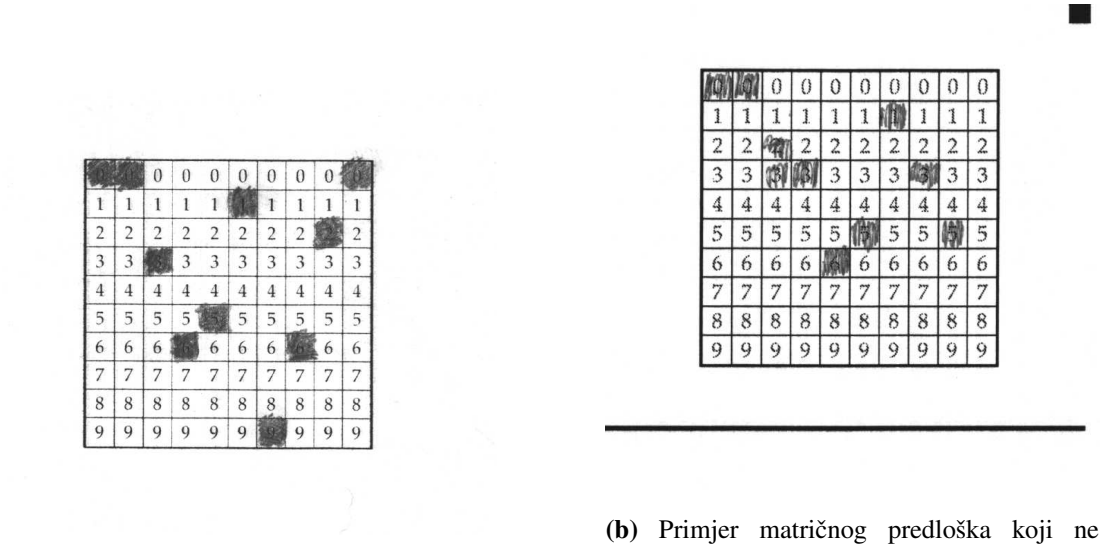
4.1. Opis skupa podataka

Skup podataka korišten u ovom radu sastoji se od 1703 primjera označnih podataka koji su prikupljeni tijekom provođenja kratkih provjera znanja na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu. Izvorni podaci sastoje se od 1703 slika u .png formatu i devet .txt datoteka koje sadrže točan JMBAG za svaku sliku. Važno je istaknuti da izvorne oznake sadrže ispravan JMBAG studenata, što znači da u slučaju da matrični predložak na slici nije ispravno ispunjen, oznaka svejedno sadrži očekivani JMBAG studenta kojemu taj ispit pripada. Takvi primjeri i oznake iz postojećeg skupa podataka morali su biti prilagođeni potrebama ovoga rada. Za treniranje modela dubokog učenja potrebno je označiti slučajeve u kojima matrični zapis nije ispravno popunjen kako bi model mogao naučiti prepoznavati takve slučajeve na temelju primjera iz skupa za učenje. Stoga su primjeri prvo morali biti provjereni kako bi se dodijelile stvarne oznake sa slike matričnog predloška. Prilagođeni skup podataka sastoji se od 1658 ispravno popunjenih matrica, dok ostali primjeri sadrže netočno popunjene matrice. Ispravno popunjeni matrični predložak (IPMP) sadrži točno jednu zacrnjenu ćeliju u svakom stupcu, što predstavlja odgovarajuću znamenku. Neispravno popunjeni matrični predlošci uključuju slučajeve u kojima se nalaze dvije oznake u istom stupcu, niti jedna oznaka u nekom stupcu ili kada su matrični predlošci očito pogrešno

označeni (npr. prekriženi ili precrtani). U nastavku rada kratica IPMP označavat će ispravno popunjen matrični predložak

4.2. Reprezentacija JMBAG-a i matričnog predloška

Matrični predložak predstavlja tablicu s brojem stupaca jednakim duljini identifikacijskog broja, pri čemu svaki stupac sadrži 10 redaka koji predstavljaju znamenke od 0 do 9. Konkretno, jedinstveni matični broj akademskog građanina u Republici Hrvatskoj sastoji se od 10 znamenki, što rezultira matričnim prikazom veličine 10x10. Matrični predložak u programskom kôdu predstavljen je dvodimenzijskim poljem brojeva (engl. *2D array*) u kojem svaki stupac predstavlja jednu znamenku, pri čemu je svaka zacrnjena pozicija unutar matričnog predloška predstavljena brojem 1, dok su sve ostale pozicije označene brojem 0. Kako bi se definirala jedinstvena relacijska veza između matričnog prikaza i JMBAG-a, definirana su pravila pretvorbe svakog stupca u znamenku, odnosno simbol. Ako određeni stupac sadrži samo jednu zacrnjenu ćeliju, tada se on predstavlja znamenkom koja odgovara položaju zacrnjene ćelije. Ako stupac ne sadrži niti jednu zacrnjenu ćeliju, tada se on reprezentira simbolom X. U slučaju kada je zacrnjeno više ćelija u jednom stupcu, taj stupac se reprezentira svim zacrnjenim znamenkama unutar uglatih zagrada (npr. [45]). Pseudokôd algoritma koji obavlja pretvorbu matričnog zapisa pokazan je u Algoritmu 1. Ovako definirana oznaka omogućuje jedinstvenu pretvorbu JMBAG-a unesenog zacrnjavanjem ćelija matrice u tekstualni zapis i obrnuto. Primjeri ispravno i neispravno popunjenog matričnog predloška s pripadajućim tekstualnim zapisom prikazani su na slici 4.1



(a) Primjer matričnog predloška koji zadovoljava IPMP, tekstualna oznaka: 0036519621

(b) Primjer matričnog predloška koji ne zadovoljava IPMP, tekstualna oznaka: 00[23]365135X

Slika 4.1

Algorithm 1 Algoritam za pretvorbu zapisa unutar matričnog predloška u tekstualni zapis

Ulaz: A – dvodimenzionalno polje brojeva, zapisa unutar matričnog predloška.
Izlaz: tekstualni zapis matričnog predloška.
$result := ""$
for ($column$ in A) **do**
  /* $findOnes$ - pronalazi indekse na kojima se nalazi 1 */
  $ones := findOnes(column)$
  if $ones.size() == 0$ **then**
    $result := result + "X"$
  else if $ones.size() == 1$ **then**
    $result := result + ones[0]$
  else
    /* $concatenateArray$ - spaja elemente polja u String reprezentaciju */
    $result := result + "[" + concatenateArray(ones) + "]"$
  end if
end for
return $result$

4.3. Program za anotiranje podataka i provjeru anotacija

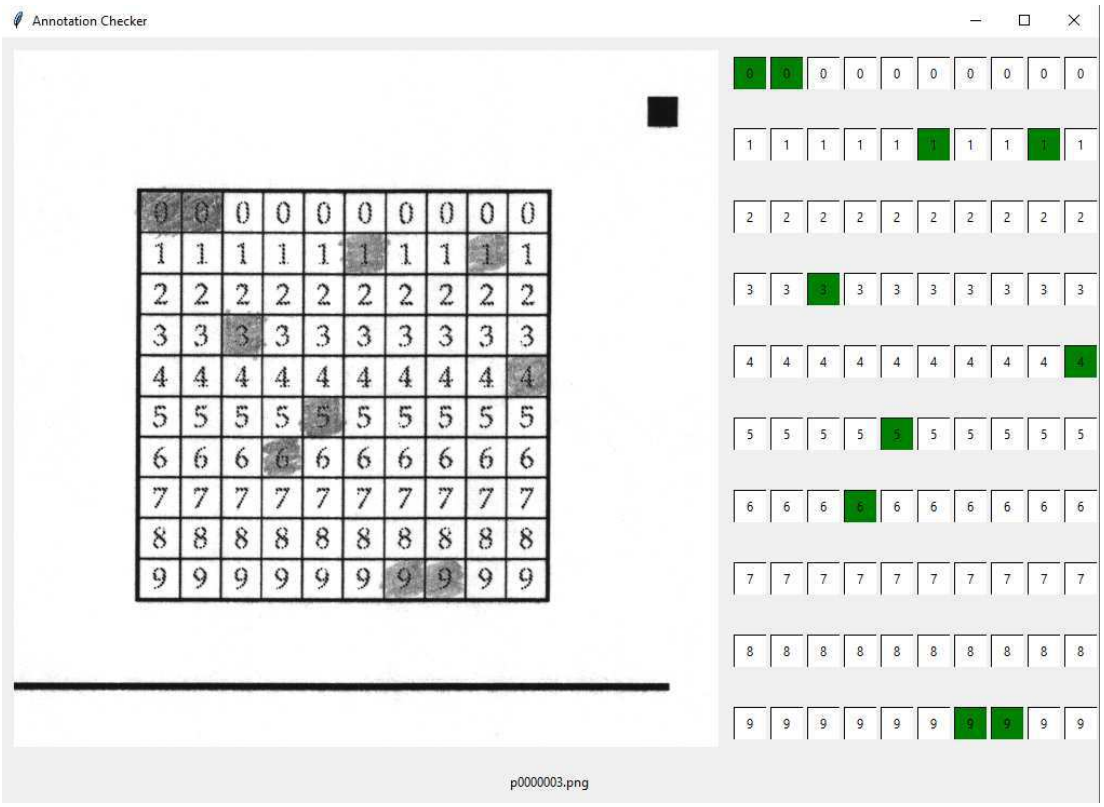
S ciljem ubrzavanja procesa označavanja primjera i verifikacije označenih primjera, razvijen je Python program koji sadrži grafičko korisničko sučelje (GUI) implementirano korištenjem programskog paketa Tkinter. Ovaj program ima za cilj olakšati označavanje podataka koji će se koristiti za treniranje neuronskih mreža, pružajući korisnicima jednostavno i intuitivno iskustvo (slika 4.2).

GUI prozor je pažljivo dizajniran i sastoji se od dva dijela. Prvi dio prikazuje sliku matričnog predloška, dok drugi dio vizualizira sam predložak matrice pomoću gumba koji se mogu prebacivati između označenih stanja (zeleno) i neoznačenih (bijelo). Ovi su gumbi raspoređeni u strukturu nalik matrici i označeni su odgovarajućim brojevima koji predstavljaju znamenke matrice. Ispod navedena dva dijela nalazi se informacija o imenu datoteke koja je trenutno u fokusu.

Interakcija s programom je jednostavna te je dovoljan klik mišem da se označi ili ukloni oznaka s gumba. Program najprije učitava sve postojeće anotacije, ako su dostupne. Osim toga, korisnici se mogu kretati kroz različite primjere pomoću tipkovnice, uz automatsko spremanje promjena koje se odvijaju prilikom prijelaza na novu sliku. Ovaj dizajn omogućuje pojednostavljeni proces označavanja, povećavajući produktivnost i brzinu označavanja podataka.

Nakon označavanja svakog primjera, podaci se spremaju u obliku parova od dvije vrijednosti: naziv datoteke koji predstavlja odgovarajuću sliku i dvodimenzionalni NumPy[1] niz koji označava oznaku matričnog zapisa. Ovi parovi pohranjeni su u lokalnom datotečnom sustavu koristeći format Pickle[2], koji omogućava serijalizaciju i deserijalizaciju Python objekata te fleksibilno spremanje raznih vrsta podataka i složenih objekata, kao što su liste, rječnici ili čak instance klasa, bez potrebe za ručnim čitanjem i pisanjem podataka.

[1] `https://numpy.org/doc/stable/index.html`
[2] `https://docs.python.org/3/library/pickle.html`



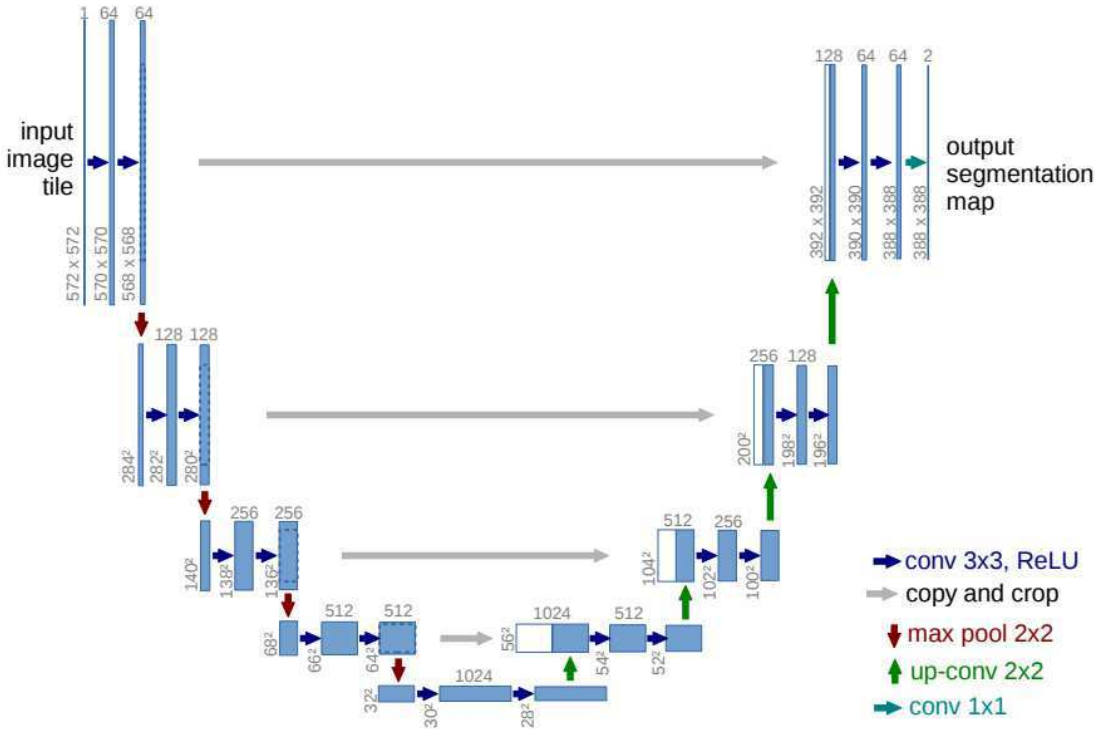
Slika 4.2: Snimka zaslona programa za anotiranje podataka

5. Arhitektura mreže

Problem prepoznavanja i detekcije JMBAG-ova predstavlja izazov koji se ne može jednostavno svrstati u klasične probleme klasifikacije slika. Zbog specifične prirode problema, u ovom radu je posebno dizajnirana arhitektura prilagođena i optimizirana kako bi se precizno riješio taj problem. Arhitektura koja se koristi u ovom radu temelji se na modificiranoj U-Net [9] arhitekturi. U-Net arhitektura pokazala je iznimne performanse u zadacima kao što je semantička segmentacija, koja uspješno uči lokacije i međusobne prostorne odnose među elementima na slici te uspješno zadržava naučene odnose na rezultantnoj slici. Ovakva arhitektura je izabrana za rješavanje problema detekcije JMBAG-a upravo iz razloga što su, položaji matričnih predložaka unutar slike direktno povezani s položajima unutar polja dvodimenzionalne oznake. Iskorištavanjem međusobno povezanih slojeva, U-Net arhitektura omogućuje poboljšane mogućnosti predviđanja. Radi razumljivosti, u ovome radu veličina sloja će se definirati pomoću rezolucije sloja i broja kanala. Kada bi se ulazna slika promatrala kao jedan sloj mreže, tada bi dimenzije slike, odnosno njezina širina i duljina, predstavljale rezoluciju sloja, a broj kanala bi bio 3 ako se radi o RGB slici, odnosno 1 ako se radi o crno-bijeloj slici. S druge strane, rezolucija oznaka predstavljena je dvjema vrijednostima. Prva vrijednost označava broj znamenki u identifikacijskom broju studenta, a druga vrijednost je jednaka 10 i označava raspon vrijednosti koje se mogu pojaviti u identifikacijskom broju (znamenke 0-9).

U-Net arhitektura sastoji se od dvije glavne cjeline. Enkoderski dio odgovoran je za progresivno smanjenje rezolucije u svakom sljedećem sloju. Kako se veličina svakog sloja smanjuje, mreža selektivno prenosi relevantne značajke sa slike na sljedeći sloj, učinkovito utječući na proces donošenja odluka u mreži. U konačnici, rezolucija se smanjuje na veličinu od 1×1, što se naziva uskim grlom (engl. *bottleneck*). Zbog svoje kompaktne veličine, sloj uskog grla sadrži najbitnije značajke i služi kao početna točka za postupak rekonstrukcije slike, koji se obavlja u dekoderskom dijelu arhitekture. Dekoderski dio obuhvaća slojeve koji postupno povećavaju razlučivost od najniže razine do rezolucije slike iz ulaznog sloja. Svaki sloj dekoderskoga dijela najprije se

konkatenira s odgovarajućim slojem iz enkoderskoga dijela koji ima rezoluciju jednaku njemu. Slojevi dobiveni opisanom konkatenacijom koriste se za povećanje rezolucije po razinama. Međusobno povezani slojevi uspostavljaju vezu između izvorne slike i rezultata modela (slika 5.1)[2].



Slika 5.1: Prikaz originalne arhitekture U-net mreže [9].

5.1. Opis arhitekture mreže

Arhitektura modela sastoji se od logičkih blokova, gdje svaki blok sadrži više slojeva odgovornih za transformaciju ulaznog tenzora. Ova podjela poboljšava organizaciju i jasnoću kôda te prikazuje jasnu segmentaciju funkcionalnosti. U nastavku će biti objašnjeni pojedinačni slojevi mreže, nakon čega slijedi prikaz blokova koji međusobno povezuju slojeve u logičke cjeline. Uz to, bit će priloženi pripadni isječci kôda koji ostvaruju implementaciju opisanih blokova. Nakon toga, bit će objašnjena mreža kao cjelina kroz povezane blokove.

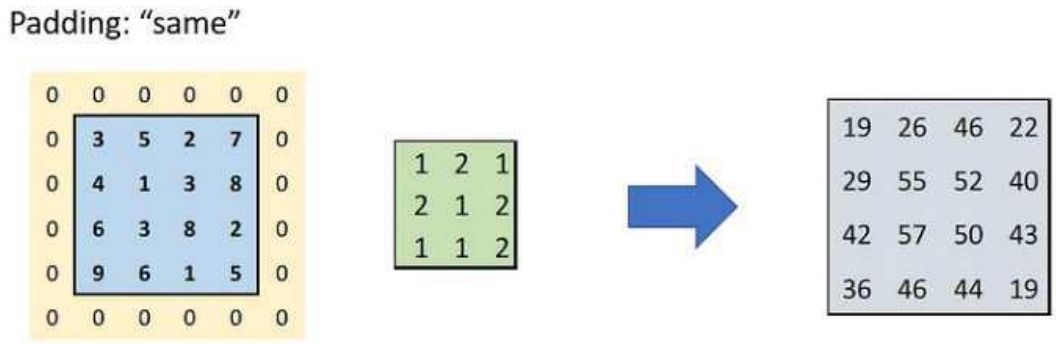
5.1.1. Opis slojeva

U izgradnji mreže korišteni su slojevi koji su dio programske biblioteke Keras. Potrebni moduli iz programske biblioteke Keras navedeni su u sljedećem isječku kôda.

```
from keras import layers, Model, activations
```

Conv2D

`Conv2D` sloj u biblioteci Keras ključan je za obradu slika konvolucijskim neuronskim mrežama. Ovaj sloj primjenjuje 2D konvoluciju na ulazni tenzor (slika 5.2). Parametar `filters` definira broj kanala u izlaznom tenzoru i određuje broj naučenih značajki, parametar `kernel_size` određuje veličinu konvolucijskog filtera, a `padding` postavljen na vrijednost `same` omogućuje očuvanje rezolucije ulaznih podataka uslijed transformacija.



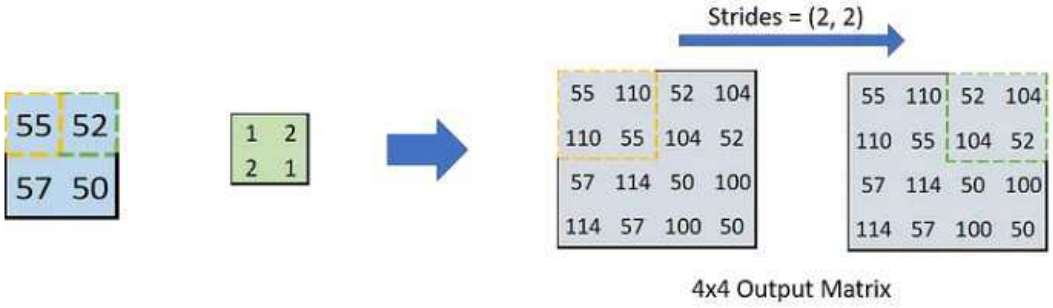
Slika 5.2: Primjena konvolucijskog sloja na jednostavnom primjeru [1].

Conv2DTranspose

Keras-ov `Conv2DTranspose` sloj ključna je komponenta za povećavanje rezolucije, odnosno dekonvoluciju u konvolucijskim neuronskim mrežama. Ovaj sloj može se promatrati kao operacija obrnuta `Conv2D` sloju. Parametri `filters`, `kernel_size` i `padding` imaju istu ulogu kao i u `Conv2D` sloju, a povećavanje rezolucije izlaznog tenzora omogućeno je postavljanjem dodatnog parametra `stride`, koji definira korak pri pomaku jezgre (engl. *kernel*). Za `stride=2`, jezgra se pomiče za 2 koraka te efektivno udvostručuje rezoluciju podataka (slika 5.3) [1].

LeakyReLU

`LeakyReLU` je nelinearna aktivacijska funkcija čije korištenje omogućuje sprječavanje ugašenih neurona, što rezultira izbjegavanjem iščezavajućeg gradijenta i boljom



Slika 5.3: Primjena transponiranog konvolucijskog sloja na jednostavnom primjeru [1].

generalizacijom u dubokim modelima učenja. `LeakyReLU` može se prikazati matematički izrazom 5.1, gdje je α zadan kao parametar sloja.

$$f(x) = \begin{cases} x & \text{ako } x \geq 0 \\ \alpha x & \text{inače} \end{cases} \tag{5.1}$$

`Dropout`

`Dropout` sloj postavlja vrijednosti unutar tenzora na 0 s određenom vjerojatnošću, koja je zadana kao argument sloja. Sloj `Dropout` pomaže u sprečavanju prenaučenosti mreže na primjerima iz skupa za treniranje.

`MaxPool2D`

`MaxPool2D` je sloj koji izvodi maksimalno uzorkovanje (engl. *max pooling*), što rezultira smanjenjem dimenzijonalnosti ulaznog tenzora. Rezolucija se smanjuje tako što se odabire maksimalna vrijednost unutar svakog prozora uzorkovanja. Prozor uzorkovanja zadaje se argumentom `pool_size`.

5.1.2. Opis blokova mreže

U ovome dijelu predstavljeni su gradivni blokovi mreže i njihovi programski kôdovi.

Enkoderski dio mreže

Enkoderski dio mreže zadužen je za smanjivanje rezolucije ulaznih podataka. Svakim sljedećim slojem rezolucija se smanjuje 2 puta. Enkoderski dio mreže sastoji se od blokova za smanjivanje rezolucije, čija implementacija je prikazana u sljedećem isječku kôda.

```
def downsample_block(x, n_filters, dropout=None):
  f = conv_block(x, n_filters)
  p = layers.MaxPool2D(pool_size=2)(f)
  if dropout is not None:
  p = layers.Dropout(dropout)(p)
  return f, p
```

Blok za smanjivanje rezolucije sastoji se od konvolucijskog bloka, `MaxPool2D` sloja, koji smanjuje rezoluciju postavljanjem argumenta `pool_size=2`, te `Dropout` sloja, ako je zadan. Blok za smanjivanje rezolucije vraća dva tenzora: tenzor prije i tenzor poslije prolaska kroz `MaxPool2D` sloj. Tenzor prije prolaska kroz `MaxPool2D` se kasnije koristi u dekoderskom dijelu mreže.

Konvolucijski blok

Implementacija konvolucijskog bloka prikazana je u isječku kôda.

```
def conv_block(x, n_filters):
  x = layers.Conv2D(n_filters, kernel_size=3,
  padding="same")(x)
  x = layers.LeakyReLU(alpha=0.2)(x)
  return x
```

Konvolucijski blok sastoji se od konvolucijskog sloja i aktivacijske funkcije. Ulazni tenzor se podvrgava operaciji konvolucijskog sloja s jezgrom veličine 3 i argumentom `padding="same"`, kako bi se nakon prolaza kroz konvolucijski sloj zadržala ista rezolucija podataka. Na izlaz iz konvolucijskog sloja primjenjuje se aktivacijska funkcija `LeakyReLU`.

Dekoderski dio mreže

Dekoderski dio mreže zadužen je za povećanje rezolucije. Svakim sljedećim slojem rezolucija se povećava 2 puta. Dekoderski dio mreže sastoji se od blokova za povećavanje rezolucije, čija je implementacija prikazana u sljedećem isječku kôda.

```
def upsample_block(x, conv_features, n_filters,
  dropout=None):
  x = layers.Conv2DTranspose(n_filters, kernel_size=3,
  strides=2, padding="same")(x)
```

```
    x = layers.concatenate([x, conv_features])
    x = conv_block(x, n_filters)
    if dropout is not None:
    x = layers.Dropout(dropout)(x)
    return x
```

U bloku za povećanje rezolucije, ulazni tenor `x` najprije prolazi kroz sloj `Conv2DTranspose`. Rezultanti tenzor konkatenira se s tenzorom `conv_features` koristeći `concatenate` metodu. Na tenzor dobiven konkatenacijom primjenjuje konvolucijski blok. Naposljetku, ako je zadan, obavlja se prolazak kroz `Dropout` sloj.

Završni blok

Zadaća završnog bloka je prilagoditi izlaz modela odgovarajućoj rezoluciji oznaka. Implementacija završnog bloka prikazana je u sljedećem isječku kôda:

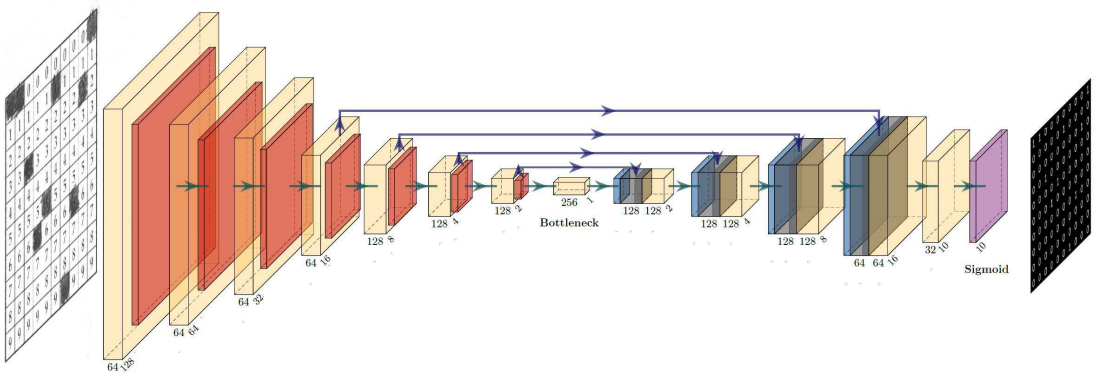
```
def final_block(x, final_channel, out_channels, digits,
    dropout=None):
    kernel_size = calculate_kernel_size((16, 16),
    (digits, 10))
    x = layers.Conv2D(final_channel,
    kernel_size=kernel_size)(x)
    x = layers.LeakyReLU(alpha=0.2)(x)
    if dropout is not None:
    x = layers.Dropout(dropout)(x)
    x = layers.Conv2D(out_channels, kernel_size=3,
    padding="same")(x)
    x = activations.sigmoid(x)
    return x
```

Završni blok sastoji se od kovolucijskoga sloja, čija veličina jezgre je izračunata pomoću metode `calculate_kernel_size` na temelju argumenta `target_shape`, koji predstavlja rezoluciju oznaka.

```
def calculate_kernel_size(input_shape, target_shape):
    kernel_size = []
    for i in range(len(input_shape)):
    kernel_size.append(input_shape[i]-target_shape[i]+1)
    return tuple(kernel_size)
```

Na dobiveni tenzor primjenjuje se aktivacijska funkcija `LeakyReLU` te, opcionalno, `Dropout` sloj. Završno, tenzor prolazi kroz `Conv2D` sloj, kojemu je broj izlaznih kanala određen parametrom `out_channels`. Također, obavlja se skaliranje vrijednosti između 0 i 1 koristeći sigmoidalnu aktivacijsku funkciju.

5.1.3. Opis strukture mreže



Slika 5.4: Prikaz strukture mreže

Razvijena arhitektura nastala je modifikacijom U-net mreže. Konkretno, modifikacija U-net mreže u ovoj arhitekturi odnosi se na dekoderski dio mreže. Tipično, dekoderski dio mreže se sastoji od slojeva koji postupno povećavaju rezoluciju od najniže razine do rezolucije ulaznog sloja. Međutim, u ovom modelu, dekoderski dio mreže se proširuje samo do sloja koji povećava rezoluciju do veličine 16x16. Nakon U-Net dijela arhitekture, slijedi završni blok mreže koji prilagođava rezoluciju izlaza modela rezoluciji oznaka (Slika 5.4).

Prvi korak u izgradnji modela je definiranje ulaza modela na temelju veličine i broja kanala ulazne slike. Potom slijedi enkoderski dio mreže, koji se sastoji od 7 enkoderskih blokova. Nakon enkoderskog dijela dobiva se sloj uskog grla, koji se koristi dalje u dekoderskom dijelu mreže. Dekoderski dio sastoji se od 4 bloka za povećanje rezolucije, koji na temelju prethodnog sloja i odgovarajućeg sloja iz enkoderskog dijela povećavaju rezoluciju. Na posljetku se izlaz iz dekoderskog dijela šalje kao ulaz u završni blok, koji generira izlaz modela. Konačno, funkcija vraća Keras Model [1], koji se instancira pomoću definiranih ulaza i izlaza.

Definicija modela implementirana je unutar sljedeće metode:

[1] `https://keras.io/guides/functional_api/`

```
def create_model(image_shape, in_channels, out_channels,
    digits, channels, final_channel,  dropout):
  inputs = layers.Input(
  shape=image_shape + (in_channels,))
  f1, p1 = downsample_block(inputs, channels[0],
  dropout=dropout)  # 128
  f2, p2 = downsample_block(p1, channels[0],
  dropout=dropout)  # 64
  f3, p3 = downsample_block(p2, channels[1],
  dropout=dropout)  # 32
  f4, p4 = downsample_block(p3, channels[1],
  dropout=dropout)  # 16
  f5, p5 = downsample_block(p4, channels[2],
  dropout=dropout)  # 8
  f6, p6 = downsample_block(p5, channels[2],
  dropout=dropout)  # 4
  f7, p7 = downsample_block(p6, channels[2],
  dropout=dropout)  # 2
  bottleneck = conv_block(p7, channels[3]) # 1
  u6 = upsample_block(bottleneck, f7, channels[2],
  dropout=dropout) # 2
  u7 = upsample_block(u6, f6, channels[2],
  dropout=dropout) # 4
  u8 = upsample_block(u7, f5, channels[2],
  dropout=dropout) #8
  u9 = upsample_block(u8, f4, channels[1],
  dropout=dropout) #16

  outputs = final_block(u9, final_channel, out_channels,
    digits, dropout)
  model = Model(inputs, outputs, name="Modified_U-Net")
  return model
```

Metoda prima sljedeće parametre:

- `image_shape` - određuje veličinu ulazne slike,
- `in_channels` - definira ulazni broj kanala slike,
- `out_channels` - označava izlazni broj kanala predikcije,
- `digits` - predstavlja broj znamenki identifikacijskog broja studenta,
- `channels` - polje vrijednosti koje definira broj kanala u određenom sloju,
- `final_channel` - broj kanala prvog konvolucijskog sloja unutar završnog bloka,
- `dropout` - ako je zadan, predstavlja parametar `Dropout` sloja koji određuje udio vrijednosti unutar tenzora koje se postavljaju na 0.

U nastavku je prikazan primjer poziva funkcije `create_model`.

```
model = create_model(image_size, in_channels=1,
    out_channels=1,
  channels=[32, 32, 64, 128], final_channel=16,
  dropout=0.1)
```

6. Proces treniranja modela

U ovome poglavlju bit će detaljno objašnjeno učitavanje, predprocesiranje i augmentacija podataka, bit će definirani korišteni hiperparametri te navedeni koraci u procesu treniranja Keras modela. Kôd za treniranje modela napisan je u obliku Juypter Notebook bilježnice.

6.1. Učitavanje podataka

6.1.1. Učitavanje popisa primjera i anotacija

Nakon procesa anotiranja podataka, označeni podaci, koji se sastoje od naziva slika i odgovarajućih anotacija, pohranjuju se u memoriju u Pickle formatu. Za učinkovito lociranje i pristup ovim podacima, raspoređenima u više direktorija, razvijena je prilagođena metoda `loadAnnotation`. Ova metoda koristi funkciju `glob` iz programske knjižnice `glob` za rekurzivno traženje Pickle datoteka. Podaci iz tih datoteka zatim se agregiraju u jedan skup podataka, predstavljen kao popis parova. Svaki par sadrži apsolutnu putanju do slikovne datoteke i njezinu odgovarajuću anotaciju pohranjenu u dvodimenzionalnom NumPy polju.

Sljedeći isječak kôda prikazuje opisanu metodu:

```
import glob
folder = "D:/FER/diplomski_rad/data"

def loadAnnotation(input_folder):
  data = []
  paths = glob.glob(input_folder + "/**/*.pkl",
  recursive=True)
  for path in paths:
  with open(path, "rb") as file:
  annotations = pickle.load(file)
```

```
    annotationsArray = [
    (str(Path(path).parent) + "\\" + name, ant)
    for (name, ant) in annotations]
    data += annotationsArray
    return data

annotation = loadAnnotation(folder)
```

Primjer jednog učitanog para podataka prikazan je u nastavku:

```
('D:\\FER\\diplomski_rad\\data\\studentidmatrix-dataset01
    \\dataset01\\p0000001.png',
array([
[1, 1, 0, 0, 0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 1, 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, 1, 0, 0, 0]], dtype=int8))
```

6.1.2. Učitavanje podataka za treniranje

Kako bi se olakšalo treniranje podataka pomoću Keras modela, nužno je kreirati prilagođenu klasu koja nasljeđuje klasu `keras.utils.Sequence`[1]. Ovaj pristup osigurava učinkovito učitavanje podataka za potrebe treniranja.

Motivacija za korištenje klase `Sequence`, kako je navedeno u Keras dokumentaciji, leži u njenoj sposobnosti omogućavanja sigurne paralelne obrade podataka. Struktura klase jamči da će neuronska mreža biti trenirana samo jednom na svakom uzorku unutar epohe.

Kada se kreira klasa koja nasljeđuje `Sequence`, potrebno je implementirati dvije ključne metode: `__getitem__` i `__len__`. Metoda `__getitem__` dohvaća seriju (engl. *batch*) podataka za određeni indeks, dok metoda `__len__` vraća ukupan

[1] `https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence`

broj dostupnih serija podataka. Dodatno, metoda `on_epoch_end` se može po želji implementirati kako bi izvršila određene zadatke na kraju svake epohe treniranja. U kontekstu ovoga rada, razvijena je prilagođena klasa `Dataset`, koja se koristi za učitavanje podataka i nasljeđuje `Sequence` klasu. Implementacija klase `Dataset` dana je u nastavku:

```
import random
from keras.utils import Sequence
import cv2
import numpy as np
import math

class Dataset(Sequence):
  def __init__(self, data, batch_size, image_size,
     transform = None):
    self.image_size = image_size
    self.data = data
    self.batch_size = batch_size
    self.transform = transform

  def __len__(self):
    return math.ceil(len(self.data) / self.batch_size)

  def __getitem__(self, idx):
    batch_data = self.data[idx * self.batch_size:
    (idx + 1) * self.batch_size]
    images = np.array([self.loadImage(name)
    for (name, ant) in batch_data])
    annotation = np.array(
    [np.expand_dims(ant.astype(np.float32), -1)
    for (name, ant) in batch_data])
    return images, annotation

  def on_epoch_end(self):
    random.shuffle(self.data)
```

```
def loadImage(self, path, transform=None):
    img = cv2.imread(str(path), cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, self.image_size,
    interpolation=cv2.INTER_AREA)
    img = img.astype(np.float32) / 255.0
    if self.transform is not None:
    img = self.transform(image=img)["image"]
    img = np.expand_dims(img, axis=-1)
    return img
```

Konstruktor klase Dataset prihvaća sljedeće argumente:

- `data` - lista parova koja se sastoji od putanja do slika i oznaka, koji zajedno čine učitani dataset
- `batch_size` - veličina serije u kojoj će se podaci učitati
- `image_size` - željena veličina na koju će se slike skalirati prilikom učitavanja, specificirana kao par vrijednosti
- `transform` - skup transformacija koje će se primijeniti na učitane slike (detalji i metode transformacija nad slikama bit će objašnjeni u kasnijim poglavljima)

Metoda `__len__` vraća rezultat dijeljenja ukupnog broja elemenata u `data` s veličinom serije, zaokružen na prvi veći cijeli broj.
Metoda `__getitem__` prima indeks serije i dohvaća odgovarajući podskup iz liste `data`. Za svaki element u podskupu učitavaju se slike i pripadne oznake, slike se učitavaju pomoću metode `load_image`. Broj kanala oznaka se proširuje kako bi odgovarala broju kanala izlaza iz mreže.
Metoda `load_image` odgovorna je za učitavanje slika pomoću programske biblioteke OpenCV[2], koja pruža funkcije za obradu i manipulaciju slika. Nakon što se slika učita, njezina rezolucija se skalira na specificiranu veličinu, a vrijednosti piksela se normaliziraju u rasponu od 0 do 1. Ako su specificirane, transformacije se primjenjuju na učitanu sliku. Dodatno, slici se dodaje dimenzija kanala, kako bi sadržavala odgovarajući broj kanala za treniranje, koji je u ovom slučaju jednak 1.
Na kraju svake epohe treniranja, metoda `on_epoch_end` nasumično miješa primjere u `data` listi. Ovo osigurava da je treniranje modela neovisno o redoslijedu podataka.

[2] `https://pypi.org/project/opencv-python/`

U nastavku je prikazan programski kôd koji demonstrira korištenje `Dataset` klase te prikazuje dimenzije učitanih slika i anotacije iz prve serije podataka.

```
whole_dataset = Dataset(annotation, batch_size,
    image_size)
print(whole_dataset[0][0].shape,
whole_dataset[0][1].shape)
# output:
((64, 128, 128, 1), (64, 10, 10, 1))
```

6.1.3. Augmentacija podataka

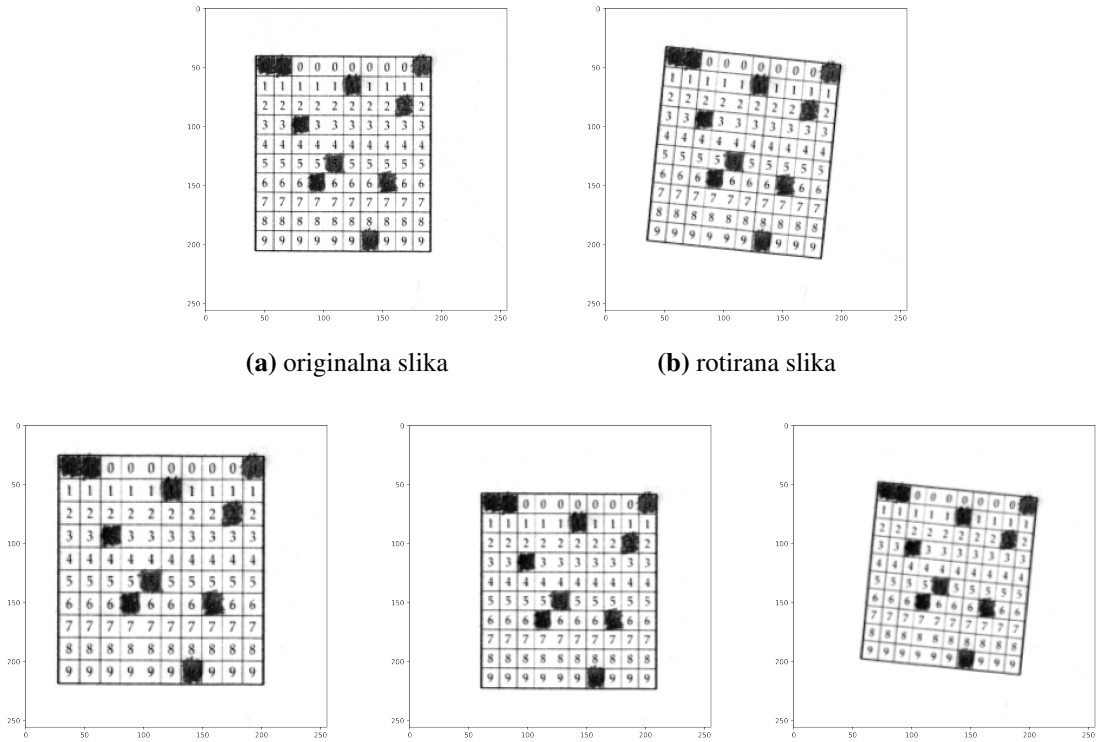
Augmentacija podataka je tehnika koja se široko koristi u strojnom učenju kako bi se umjetno povećala veličina i raznolikost skup podataka za treniranje. Primjenom različitih transformacija, kao što su rotacija, skaliranje i translacija, dobiva se prošireni skup podataka, koji pruža dodatne varijacije za treniranje modela. Na taj način poboljšava se robusnost i sposobnost generalizacije modela te se smanjuje prenaučenost, što dovodi do bolje izvedbe i preciznosti u stvarnim scenarijima. Augmentacija podataka može se obaviti prije samog procesa treniranja, a novogenerirane slike se spremaju u lokalnu memoriju. Osim toga, postoji tehnika online augmentacije podataka, koja omogućuje generiranje novih uzoraka tijekom samog procesa učenja.

Kako bi se poboljšao proces treniranja, u ovome radu korištena je online augmentacija podataka. Za implementaciju niza augmentacija korištena je Albumentations programska biblioteka. Korištena augmentacija sastoji se od više komponiranih transformacija, uključujući:
`Rotate` (rt) - rotira sliku za slučajno odabrani kut iz zadanog raspona.
`ShiftScaleRotate` (sc) - skalira sliku za slučajno odabrani faktor iz zadanog raspona, a pritom se matrični predložak povećava ili smanjuje unutar slike. Iako ime metode sugerira da navedena augmentacija obavlja i ostale augmentacije, postavljanjem argumenata funkcije provodi se samo nasumično skaliranje slike.
`ShiftScaleRotate` (sh) – pomiče (translatira) sliku horizontalno i vertikalno za slučajno odabrani faktor iz zadanog raspona, a nastale praznine popunjava zadanom vrijednošću. Iako ime metode sugerira da navedena augmentacija obavlja i ostale augmentacije, postavljanjem argumenata funkcije provodi se samo nasumični pomak slike.

Ispod je prikazan kôd za implementaciju augmentacija, a na slici 6.1 je prikazan

vizualni primjer svake augmentacije.

```
import cv2
import albumentations as A
train_transform_test = A.Compose([
A.Rotate(limit=(-6,6), p=1,mask_value=1),
A.ShiftScaleRotate(p=1, shift_limit=0.0,
   scale_limit=0.1, rotate_limit=0, mask_value=1,
   border_mode=cv2.BORDER_CONSTANT, value=1),
A.ShiftScaleRotate(p=1, shift_limit=0.1,
   scale_limit=0.00, rotate_limit=0, mask_value=1,
   border_mode=cv2.BORDER_CONSTANT, value=1)
])
```



(a) originalna slika **(b)** rotirana slika

(c) skalirana slika **(d)** pomaknuta slika **(e)** primjena svih augmetacija

Slika 6.1: Primjer augmentacija

Količina augmentacije kontrolira se pomoću hiperparametra `org_prob` (p_{org}), koji određuje očekivani postotak originalnih, neaugmentiranih primjera u svakoj seriji. Svaka augmentacija ima zadan prvobitni faktor augmentacije između 0 i 1 te se na temelju zadanog parametra p_{org} optimizacijskim postupkom zlatnog reza traži parametar

μ, koji zadovoljava izraz 6.1. Naposljetku, svaki faktor augmentacije se množi s dobivenim μ i ta vrijednost predstavlja konačnu vjerojatnost pojedine augmentacije. Ovim pristupom postiže se ravnoteža između originalnih slika i augmentiranih primjera u skupu za učenje. Korištenje određenog postotka neaugmentiranih slika osigurava se da model ima dovoljno podataka za učenje izvornih uzoraka, dok istovremeno koristi augmentirane primjere kako bi se omogućilo učenje varijacija i povećala robusnost. Kôd za optimizaciju vjerojatnosti augmentacija prikazan je u nastavku.

```
org_prob = 0.5
from scipy import optimize
probabilities = [ t.p for t in train_transform]
p = lambda x: abs( np.prod([1- x * p for p in
    probabilities]) -  originalImagesProbability)
minimum = optimize.golden(p, brack=(0, 1e-6))
print("Optimization factor", minimum)
for t in train_transform:
new_p = t.p * minimum
if new_p < 0 or new_p > 1:
raise ValueError("new P is outside of probability limits,
     new_p = "+ str(new_p))
t.p = new_p
print("Adjusted probablities" ,[ str(t.p) for t in
    train_transform])

# output
Optimization factor 0.6172705889627702
Adjusted probablities ['0.2469', '0.1851', '0.1851']
```

$$p_{\text{org}} = (1 - \mu\gamma_{\text{rt}})(1 - \mu\gamma_{\text{sh}})(1 - \mu\gamma_{\text{sc}}) \tag{6.1}$$

$$p_{\text{rt}} = \mu\gamma_{\text{rt}} \tag{6.2}$$

$$p_{\text{sh}} = \mu\gamma_{\text{sh}} \tag{6.3}$$

$$p_{\text{sc}} = \mu\gamma_{\text{sc}} \tag{6.4}$$

6.1.4. Podjela podataka na skup za učenje i skup za validaciju

Učitani skup podataka potrebno je podijeliti na skup za treniranje i skup za validaciju. U ovom radu korištena je k-dijelna podjela (engl. *k-fold*) za `k=5`, čime je ostvarena metoda za podjelu skupa podataka na 5 dijelova. Svaki model je treniran 5 puta, pri čemu je jedan od dijelova podjele korišten kao validacijski skup podataka, dok su preostala 4 dijela korištena za trening. K-dijelna podjela je moćna tehnika koja se koristi u strojnom učenju za procjenu performansi modela i smanjenje rizika od prenaučenosti. Podjelom skupa podataka na k podskupova te ponovljenim treniranjem i evaluacijom modela na različitim kombinacijama podataka, k-dijelna podjela pruža pouzdaniju procjenu točnosti i robusnosti modela. Programski kôd u nastavku prikazuje podjelu podataka k-dijelnom metodom.

```
from sklearn.model_selection import KFold
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=1)
fold_datasets = []
folds = kf.split(annotation)
for fold, (train_indices, test_indices)
  in enumerate(folds):
  train_data = [annotation[i] for i in train_indices]
  test_data = [annotation[i] for i in test_indices]
  train_dataset = Dataset(train_data, batch_size,
    image_size, transform=train_transform)
  test_dataset = Dataset(test_data, batch_size,
    image_size)
  fold_datasets.append((train_dataset, test_dataset))
```

6.2. Treniranje modela

Nakon učitavanja i podjele podataka na skup za učenje i skup za validaciju, obavlja se definiranje modela i njegovog procesa učenja. Definira se arhitektura modela, funkcija gubitka, optimizacijski postupak i metrike koje će se pratiti tijekom procesa učenja. Zatim se model sastavlja (engl. *compile*) tako da se specificira optimizacijski postupak, funkcija gubitka i potrebne metrike. Nakon što je model sastavljen, može se započeti s treniranjem neuronske mreže.

Tijekom treniranja modela koristi se prilagodljiva stopa učenja, koja se definira argumentima `decay_rate` i `decay_step`. Svakih `decay_step` epoha, stopa učenja se smanjuje za `decay_rate`.

```
from keras.callbacks import LearningRateScheduler

def lr_scheduler(epoch, lr):
  if epoch % decay_step == 0 and epoch:
    return lr * pow(decay_rate,
      np.floor(epoch / decay_step))
  return lr
  callbacks = [LearningRateScheduler(lr_scheduler,
    verbose=1)]
```

Proces treniranja započinje pozivom metode modela `fit`.

```
history = model.fit(train_dataset,
  validation_data=val_dataset, epochs=epochs,
  callbacks=callbacks)
```

Metoda `fit` prima skup podataka za učenje i validaciju, broj epoha i dodatne povratne metode koje implementiraju oblikovni obrazac promatrača. Povratne metode omogućuju funkcionalnosti poput definiranja prilagodljive stope učenja, spremanja modela tijekom treniranja i prikaza rezultata kroz epohe.

Za treniranje modela u ovome radu korišten je optimizacijski postupak Adam[3] (engl. *Adaptive Moment Estimation*)[7], koji je poznat po svojoj efikasnosti, brzoj konvergenciji i robusnosti. Za funkciju gubitka modela koristi se binarna unakrsna entropija[4] (engl. *binnary cross-entropy*).

Cjelokupni proces treniranja i dobivanja konačnih rezultata uključivao je učenje modela pet puta s različitim trening podacima za određeni skup hiperparametara. Rezultati i povijest svakog treniranja zabilježeni su u lokalnoj memoriji. Na kraju su izračunate srednje vrijednosti metrika za sve iteracije treniranja kako bi se rezultati mogli uspoređivati za različite vrijednosti hiperparametara.

Nakon analize rezultata i odabira konačnih hiperparametara, model je treniran na cijelom skupu podataka. Tako dobiveni model namijenjen je za korištenje u aplikaciji za detekciju JMBAG-a.

[3] `https://keras.io/api/optimizers/adam/`

[4] `https://keras.io/api/losses/probabilistic_losses/#binarycrossentropy-class`

6.3. Hiperparametri modela

Performanse modela uvelike ovise o različitim hiperparametrima. U tablici 6.1 navedeni su korišteni hiperparametri i njihove vrijednosti. Varijabilna vrijednost nekih hiperparametara označava da će se njihov utjecaj objasniti u dijelu rada koji se bavi analizom rezultata.

Tablica 6.1: Hiperaparametri modela

Naziv	Vrijednost	Opis
`image_size`	(128,128)	rezolucija slika na kojima se trenira model
`batch_size`	64	veličina serije (engl. *batch*) podataka na kojima se trenira mreža
`k_folds`	5	broj dijelova na koji se dijeli skup podataka
`org_prob`	varijabilna	očekivani postotak neaugmentiranih slika unutar serije
`channels`	varijabilna	veličine kanala u U-et dijelu arhitekture
`final_channel`	varijabilna	veličine kanala u završnom bloku
`droupout`	varijabilna	postotak gašenja neurona u mreži
`learning_rate`	0.0015	stopa učenja mreže na početku
`epochs`	150	broj epoha koji će se trenirati model
`decay_rate`	0.9	postotak smanjena stope učenje svakim korakom
`decay_step`	15	broj epoha nakon koje se smanjuje stopa učenja

7. Rezultati

U ovom dijelu rada bit će predstavljeni rezultati treniranja neuronske mreže za detekciju JMBAG-ova. Prvo će biti predstavljene metrike koje opisuju performanse modela. Nakon toga će biti prikazani rezultati u ovisnosti o veličini mreže te analiziran utjecaj implementiranih tehnika koje za cilj imaju poboljšati rezultate, kao što su adaptivna stopa učenja, augmentacija podataka i Dropout sloj. Na kraju će biti prikazani i komentirani primjeri u kojima je model uspješno i neuspješno detektirao JMBAG-ove.

7.1. Evaluacijske metrike

S obzirom na prirodu problema, evaluacijske metrike su prilagođene specifičnim zahtjevima zadatka. Neka $y^{(k)}$ predstavlja 2D vektor stvarne oznake za k-ti primjer, a $\hat{y}^{(k)}$ 2D vektor predviđene oznake za isti primjer. Definiramo jednakost ovih dvaju vektora ako su svi njihovi odgovarajući elementi jednaki, formalno izraženo kao:

$$y_{ij}^{(k)} = \hat{y}_{ij}^{(k)} \quad \forall i, j$$

U ovom radu korištene su tri metrike: točnost, alpha pogreška i beta pogreška.

7.1.1. Točnost

Točnost (ACC) predikcije definirana je kao srednja vrijednost indikatorske funkcije $I(y^{(k)}, \hat{y}^{(k)})$ (jednadžba 7.1), gdje je vrijednost I jednaka 1 ako su vektori jednaki, a 0 ako nisu, za sve primjere.

$$\text{ACC} = \frac{1}{N} \sum_{k=1}^{N} I(y^{(k)}, \hat{y}^{(k)}) \tag{7.1}$$

$$I(y^{(k)}, \hat{y}^{(k)}) = \begin{cases} 1, & \text{ako } y^{(k)} = \hat{y}^{(k)} \\ 0, & \text{inače} \end{cases} \tag{7.2}$$

7.1.2. Alfa pogreška

Alfa pogreška (kritična pogreška) predstavlja pogrešku modela u kojoj $y^{(k)} \neq \hat{y}^{(k)}$ i $\hat{y}^{(k)}$ zadovoljava IPMP (poglavlje 4.1). Ova pogreška se smatra kritičnom jer predviđene oznake koje zadovoljavaju IPMP ne zahtijevaju interakciju s ispravljačem nego su automatski procesuirane, što otežava otkrivanje pogrešaka. Stopa alfa pogreške predstavlja udio primjera s alfa pogreškom u odnosu na ukupan broj predikcija koje zadovoljavaju IPMP.

7.1.3. Beta pogreška

S druge strane, beta pogreška predstavlja pogrešku modela u kojoj $y^{(k)} \neq \hat{y}^{(k)}$ i $y^{(k)}$ zadovoljava IPMP. Iako manje kritična, beta pogreška ipak zahtijeva ručni pregled ispravljača, iako je student ispravno popunio predložak. Stopa beta pogreške predstavlja udio primjera s beta pogreškom u odnosu na ukupan broj oznaka koje zadovoljavaju IPMP.

7.2. Evaluacija performansi modela

Performanse modela bit će ispitane i prikazane kroz analizu tri različita modela, pri čemu svaki od njih ima različite razine složenosti. Složenost modela određena je brojem kanala prisutnih u njegovim slojevima. Broj kanala direktno utječe na sposobnost modela da nauči i predstavi kompleksne obrasce i značajke unutar ulaznih podataka.

Definirani su sljedeći modeli

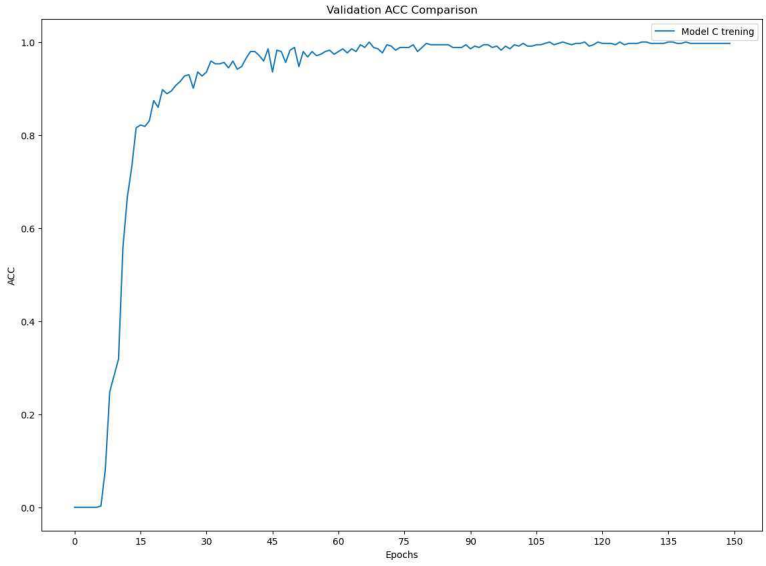
- Model A: `channels` = [16,16,32,64], `final_channels` = 16 (ukupni broj parametara: 163,057), najbolje performanse dobivene su za sljedeće hiperparametre: `Dropout` = 0.1, `prob_org` = 0.5
- Model B: `channels` = [32,32,64,128], `final_channels` = 32 (ukupni broj parametara: 650,721), najbolje performanse dobivene su za sljedeće hiperparametre: `Dropout` = 0.1, `prob_org` = 0.5
- Model C: `channels` = [64,64,128,256], `final_channels` = 64 (ukupni broj parametara: 2,599,873), najbolje performanse dobivene su za sljedeće hiperparametre: `Dropout` = 0.3, `prob_org` = 0.5

Tablica 7.1 prikazuje prosječne vrijednosti metrika za navedene modele tijekom treninga na k-dijelnom skupu podataka. Rezultati pokazuju da sva tri modela postižu iznimno visoku točnost i niske stope alfa i beta pogrešaka. Kao što je očekivano,

model s najvećim brojem parametara pokazuje najbolju izvedbu po svim metrikama. Eksperimentalna istraživanja potvrđuju da najbolji model postiže točnost od 98,65%. Osim toga, s alfa pogreškom od 0.18% i beta pogreškom od 0.49%, model pokazuje visoku pouzdanost i prikladnost za stvarne scenarije. Ti rezultati ističu praktičnu primjenjivost predloženog modela u preciznom predviđanju ispravnog JMBAG-a. Graf krivulje točnosti prikazuje vrijednosti točnosti kroz epohe. Iz slike 7.1 je vidljivo da model već nakon 50 epoha postiže visoku točnost na validacijskom skupu podataka. Međutim, analizom ponašanja modela utvrđeno je da točnost i dalje fluktuira sve do približno 150. epohe. Na temelju tih rezultata, odlučeno je da će se finalni model trenirati 150 epoha.

Tablica 7.1: Vrijednosti metrika za trenirane modele

model	**ACC**	**α Pogreška**	**β Pogreška**
Model A	0.9497	0.0062	0.0344
Model B	0.9719	0.0043	0.0170
Model C	0.9865	0.0018	0.0049



Slika 7.1: Točnost modela C kroz epohe

Tablica 7.2 prikazuje vremena treniranja i predikcije na procesorskoj i grafičkoj jedinici. U ovom radu, za treniranje modela korištena je grafička kartica GTX 1050

Ti mobile[1] (4GB VRAM memorije). Iz rezultata mjerenja vremena vidljivo je da za treniranje jedne epohe za sve modele treba otprilike 6 sekundi, što ukupno iznosi oko 15 minuta za treniranje. To predstavlja relativno kratko vrijeme treniranja za duboke modele. Važno je napomenuti da navedena grafička kartica pripada među najslabije grafičke kartice na tržištu i da bi vrijeme treniranja bilo značajno kraće na modernijim i moćnijim grafičkim karticama. Cijeli proces evaluacije skupa parametara kroz 5 treninga trajao je otprilike 80 minuta. Nadalje, u tablici su prikazana vremena predikcije serije podataka od 64 slike za sva tri modela. Vidljivo je da vrijeme predikcije značajno ovisi o veličini modela i uređaju na kojem se izvršava. Također, rezultati ukazuju da je vrijeme predikcije u programskom jeziku Java značajno veće nego u programskom jeziku Python. Tijekom rada na ovom diplomskom radu nije pronađeno rješenje za taj problem.

Tablica 7.2: Vremena izvođenja i predikcije modela

model	**vrijeme treniranja jedne epohe (GPU)**	**Vrijeme predikcije Python (GPU) (64 slike)**	**Vrijeme predikcije Python (CPU) (64 slike**	**Vrijeme predikcije Java (CPU) (64 slike)**
Model A	5.4530 s	248 ms	317 ms	710 ms
Model B	5.6717 s	261 ms	403 ms	1453 ms
Model C	6.0804 s	287 ms	648 ms	2850 ms

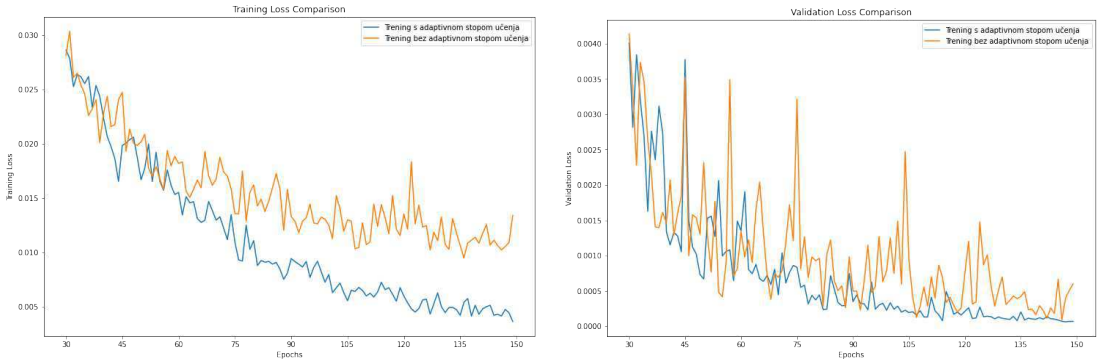
7.3. Evaluacija implementiranih tehnika za poboljšanje performansi

7.3.1. Analiza utjecaja adaptivne stope učenja

Koristeći iste hiperparametre, model C je ponovno treniran bez adaptivne stope učenja. Rezultati tog treninga prikazani su u tablici 7.4, otkrivajući pad performansi. Analizom funkcije gubitka kroz epohe za oba modela sa i bez adaptivne stope učenja, vidljivo je da model treniran bez adaptivne stope učenja pokazuje veći gubitak na skupu za učenje

[1] `https://www.techpowerup.com/gpu-specs/geforce-gtx-1050-ti-mobile.c2912`

kako epohe napreduju (slika 7.2a). Također, na validacijskom skupu gubitak je veći i nestabilniji kroz epohe (slika 7.2b), što rezultira lošijim performansama modela.



(a) Usporedba gubitka na trening setu modela sa i bez adaptivne stope učenja

(b) Usporedba gubitka na validacijskom setu modela sa i bez adaptivne stope učenja

Slika 7.2

Tablica 7.3: Usporedba modela sa i bez adaptivne stope učenja

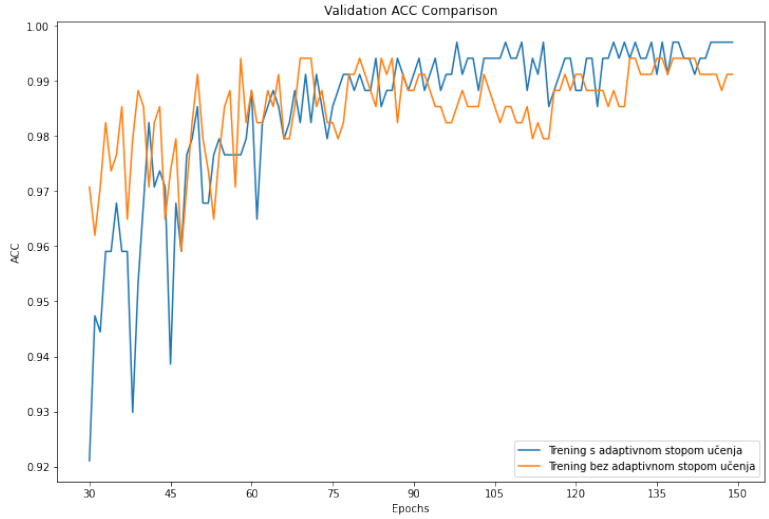
model	**ACC**	**α Pogreška**	**β Pogreška**
Model C s adaptivnom stopom učenja	0.9865	0.0018	0.0049
Model C bez adaptivne stope učenja	0.9719	0.0061	0.0194

7.3.2. Analiza utjecaja argumentiranja slika

Koristeći iste hiperparametre, model C je ponovno treniran bez online augmentiranja slika. Kao što je vidljivo na slici 7.3, model u ranijim epohama postiže visoku točnost na validacijskom skupu, ali ne uspijeva postići jednaku točnost kao model s augmentiranim slikama zbog slabije sposobnosti generalizacije. Usporedba rezultata prikazani su u tablici 7.4

Tablica 7.4: Usporedba modela sa i bez augumentiranja slika

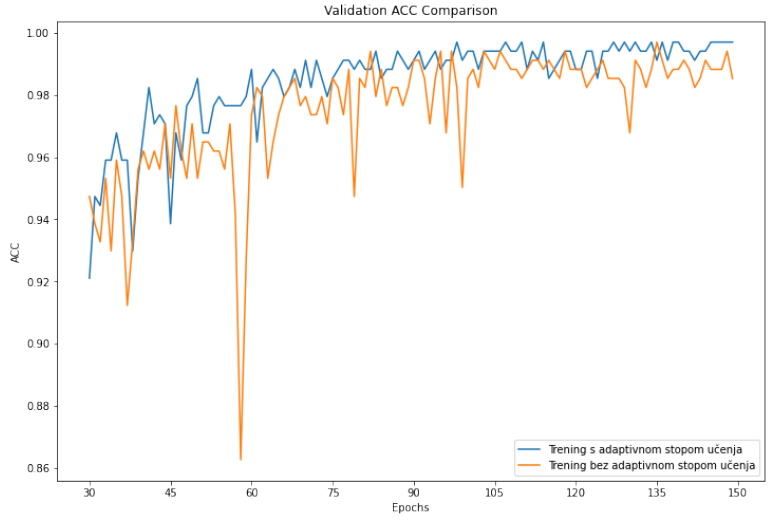
model	**ACC**	**α Pogreška**	**β Pogreška**
Model C s augmentiranjem slika	0.9865	0.0018	0.0049
Model C bez augmentiranja slika učenja	0.9656	0.0056	0.0212



Slika 7.3: Usporedba točnosti na validacijskom skupu modela sa i bez argumentiranja slika

7.3.3. Analiza utjecaja `Dropout` sloja

Kako bi se ispitao utjecaj `Drouput` sloja, ponovljen je trening modela C bez `Dropout` sloja. Iz slike 7.4 je vidljivo da je točnost na validacijskom skupu nestabilnija te dolazi do prenaučenosti modela. U Tablici 7.5 prikazana je usporedba modela sa i bez `Drouput` sloja.



Slika 7.4: Usporedba točnosti na validacijskom skupu modela sa i bez `Dropout` sloja

Tablica 7.5: Usporedba modela sa i bez `Dropout` sloja

model	ACC	α Pogreška	β Pogreška
Model C s `Dropout` slojem	0.9865	0.0018	0.0049
Model C bez `Dropout` sloja	0.9742	0.0049	0.0223

7.4. Analiza i evaluacija primjera predikcije u modelu za detekciju JMBAG-ova

U sljedećem dijelu će biti prikazani i komentirani primjeri za koje model generira ispravnu predikciju, kao i primjeri za koje model generira pogrešnu predikciju.

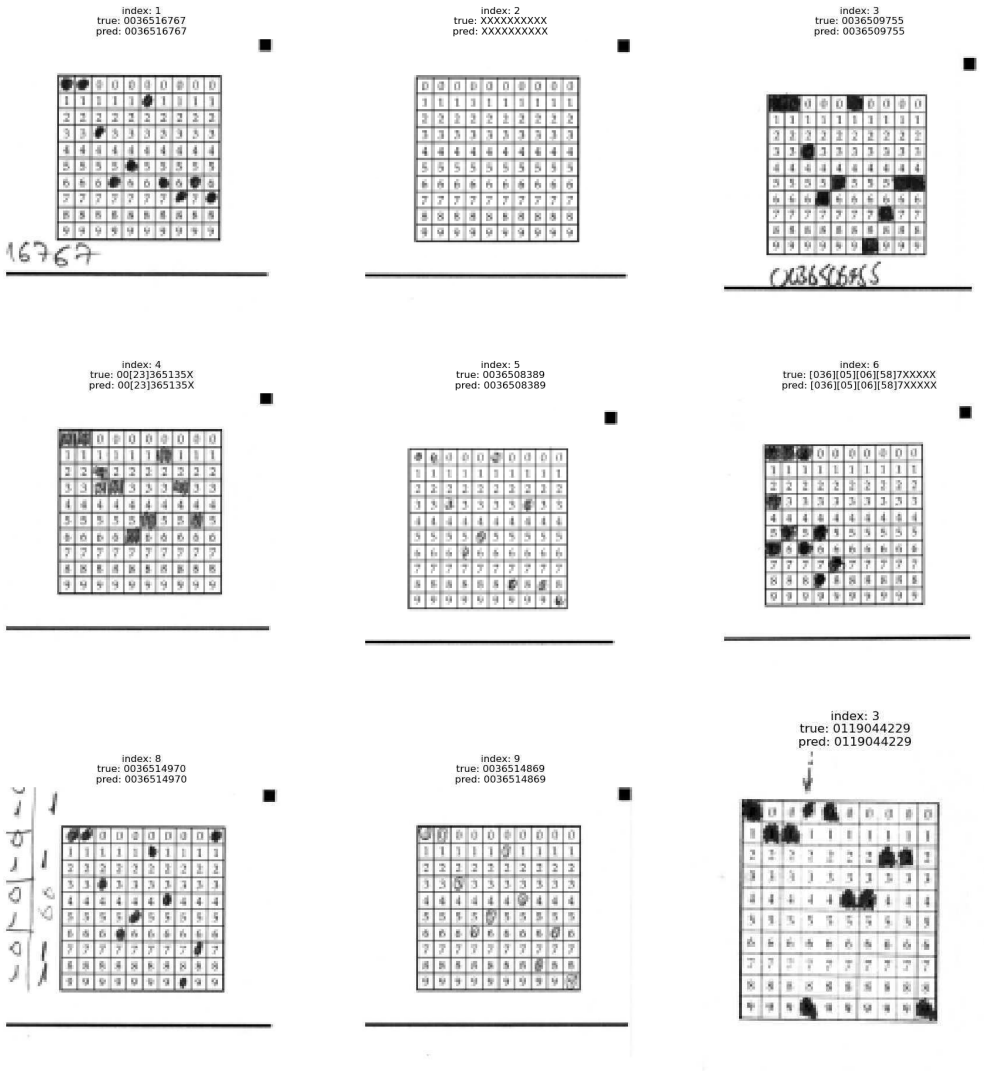
7.4.1. Primjeri s ispravnom predikcijom

Slika 7.5 prikazuje primjere iz validacijskih skupova iz k-dijelne podjele s ispravnom predikcjom.

7.4.2. Primjeri s neispravnom predikcijom

Slika 7.6 prikazuje netočne predikcije koje ne pripadaju ni alfa ni beta pogrešci. Primjeri prikazuju prekrižene ili precrtane matrične predloške za koje je je model detektirao neke unose, ali čija je stvarna anotacija zapravo jednaka praznom matričnom predlošku. Povećanjem primjera prekriženih ili precrtanih matričnih predložaka u skupu za učenje, model bi lakše detektirao takve primjere i dodijelio im ispravnu oznaku.

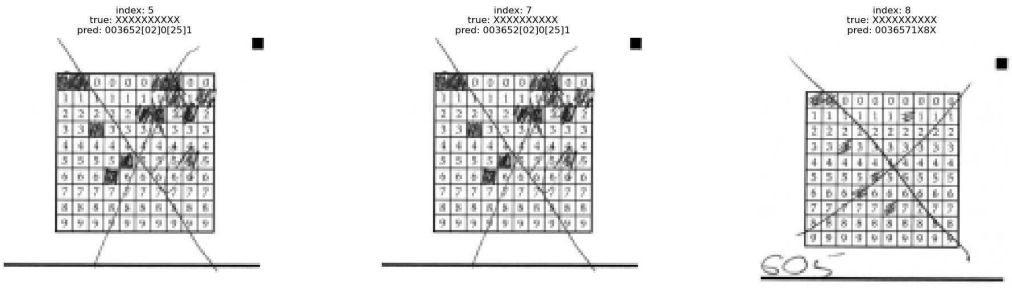
Na slici 7.7 su prikazani primjeri iz validacijskih skupova k-dijele podjele koji zadovoljavaju svojstvo beta pogreške. Primjeri s indeksom 2, 3 i 6 prikazuju primjere s ispravljanjem unosa u matrični zapis, odnosno situacije u kojima je prekrižena oznaku koju se želi ukloniti. Takvi primjeri predstavljaju „sivu zonu“ predikcije jer, iako je u matričnom predlošku zacrnjeno više oznaka za jednu znamenku, postoji jasan način raspoznavanja koja oznaka je pogrešna. Odluku o tome kako se takvi primjeri trebaju anotirati treba donijeti ekspert ili korisnik koji će koristiti model za predikciju. U ovom radu, takvi primjeri su označeni kao ispravan unos u matrični zapis, kako bi se ispitala mogućnost modela da se nosi s takvim primjerima. Zbog malog broja takvih primjera u skupu podataka, u nekim slučajevima model daje pogrešnu predikciju. Slična stvar je vidljiva u primjerima s indeksom 1 i 7, gdje je student/ica počeo/la zacrnjivati jednu



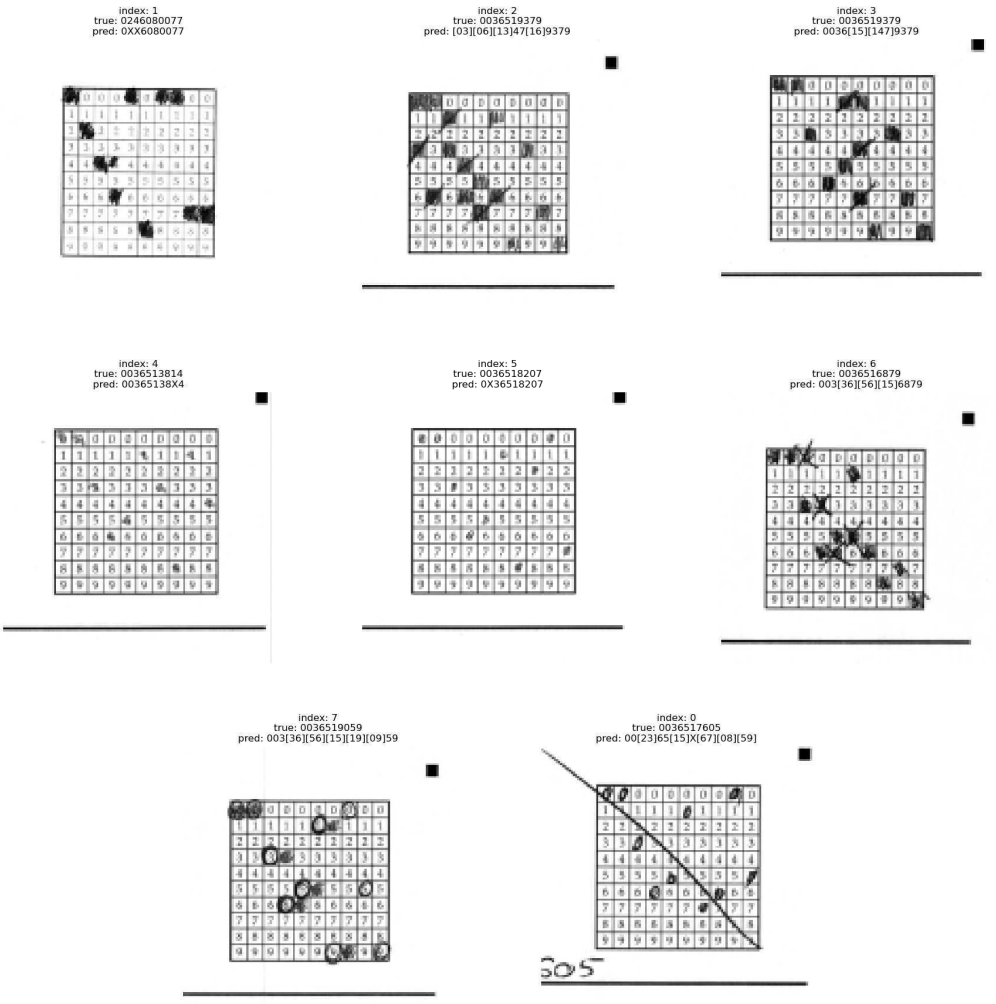
Slika 7.5: Primjeri s ispravnom predikcijom

oznaku, a potom je shvatio/la da označava pogrešnu ćeliju matrice te je zatim zacrnio/la drugu ćeliju u stupcu.

Opisani slučajevi utjecali su na konačne mjere uspješnosti modela. Iako ovakvi primjeri imaju anotaciju koja sugerira da se radi o ispravno popunjenim matričnim predlošcima, ipak bi bila potrebna intervencija ispravljača da potvrdi točnost predikcije za takve primjere.



Slika 7.6: Primjeri s prekriženim matričnim predloškom



Slika 7.7: Primjeri s ispravnom predikcijom

8. Računalna aplikacija za detekciju JMBAG-ova

U ovom dijelu bit će predstavljena računalna aplikacija za detekciju JMBAG-ova. Aplikacija je implementirana u programskom jeziku Java kao Maven[1] projekt te su korištene programske biblioteke Deeplearning4j i Nd4j za rad s Keras modelom i višedimenzionalnim vektorima. Najprije će biti prikazan općeniti način rada s Nd4j višedimenzionalnim vektorima, zatim će biti objašnjeno učitavanje skupa podataka i modela te će na kraju biti predstavljena dva primjera korištenja računalne aplikacije za predikciju JMBAG-ova. Za korištenje programskih biblioteka Deeplearning4j i Nd4j potrebno je navesti navedene biblioteke kao dodatke (engl. *dependencies*) u datoteku pom.xml, koja je sastavni dio Maven projekta.

```
<dependencies>
  <dependency>
    <groupId>org.deeplearning4j</groupId>
    <artifactId>deeplearning4j-core</artifactId>
    <version>1.0.0-M2.1</version>
  </dependency>
  <dependency>
    <groupId>org.nd4j</groupId>
    <artifactId>nd4j-native</artifactId>
    <version>1.0.0-M2.1</version>
  </dependency>
</dependencies>
```

[1] https://maven.apache.org/what-is-maven.html

8.1. ND4j višedimenzionalni vektori

ND4J pruža podršku za implementaciju N-dimenzionalnih vektora, koja omogućuje učinkovite numeričke izračune u Javi. Za stvaranje ND4J polja, prvo moramo uvesti potrebne razrede (engl. *classes*) iz ND4J biblioteke.

```
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.factory.Nd4j;
import org.nd4j.linalg.indexing.NDArrayIndex;
```

Nadalje, prikazano je kako stvoriti ND4J polje. Moguće je inicijalizirati 2D polje s određenim dimenzijama pomoću metode `Nd4j.create`.

```
int numRows = 3;
int numColumns = 4;
INDArray array = Nd4j.create(numRows, numColumns);
```

Moguće je pristupiti i mijenjati elemente polja koristeći metode `put` i `get`.

```
array.put(0, 0, 1.0);
array.put(0, 1, 2.0);
double element = array.getDouble(0, 0);
```

Također, ND4J omogućuje rad s podpoljima. Podpolje možemo izdvojiti koristeći metodu `get` i specificirajući raspon indeksa.

```
INDArray subArray = array.get(
  NDArrayIndex.interval(0, 2), NDArrayIndex.all());
```

Osim toga, mogu se izvoditi matematičke operacije nad ND4J poljima, poput zbrajanja i množenja elemenata te operacija s matricama.

```
INDArray array1 = Nd4j.ones(3, 3);
INDArray array2 = Nd4j.zeros(3, 3);
INDArray sum = array1.add(array2);
INDArray product = array1.mul(array2);
INDArray matrixProduct = array1.mmul(array2);
```

Nadalje, ND4J pruža praktične metode za statističke izračune, kao što su izračun srednje vrijednosti, standardne devijacije i zbroja.

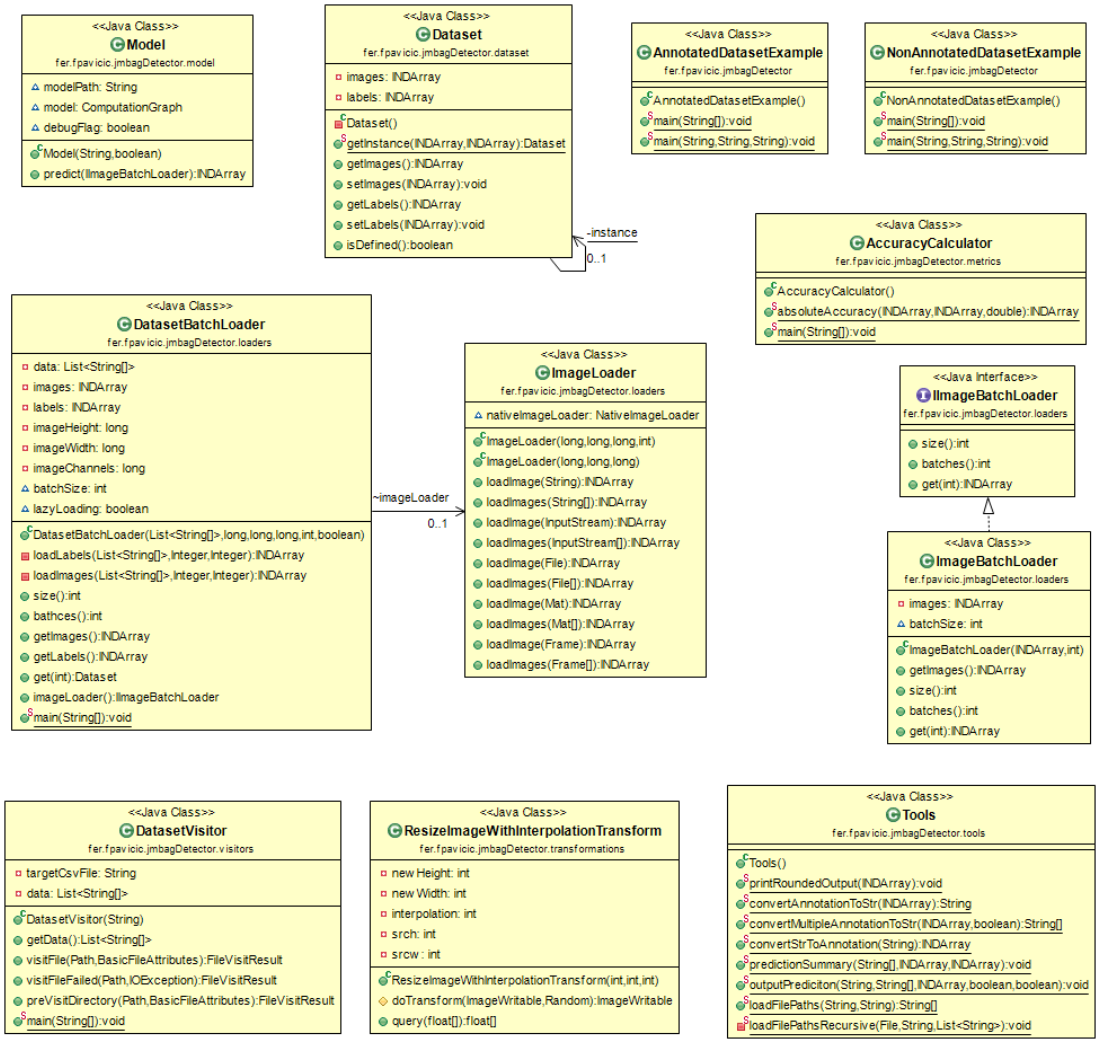
```
double mean = array.meanNumber().doubleValue();
double stdDev = array.stdNumber().doubleValue();
```

```
double sum = array.sumNumber().doubleValue();
```

Ovi primjeri kôda demonstriraju osnovnu upotrebu ND4J N-dimenzionalnih polja, uključujući stvaranje polja, pristup elementima, izdvajanje podpolja, matematičke operacije i statističke izračune. Iskorištavanjem ovih funkcionalnosti, moguće je izvoditi učinkovite numeričke izračune i manipulacije nad višedimenzionalnim podacima koristeći ND4J u Javi.

8.2. Dijagram razreda

Slika 8.1 prikazuje dijagram razreda koji su implementirani u izradi aplikacije.



Slika 8.1: Dijagram razreda

8.3. Učitavanje skupa podataka

U ovom dijelu bit će prikazan način učitavanja podataka. Razvijena je implementacija za učitavanje podatka koji se sastoje od slika i anotacija te implementacija za učitavanje slika iz mnogobrojnih izvora.

8.3.1. Učitavanje podataka koji sadrže slike

Prilikom korištenja ovog računalnog programa, očekivano ponašanje je da korisnik od aplikacije zatraži predikciju za skup slika popunjenih matričnih predložaka. Slike mogu već biti učitane ili se moraju moći učitati iz lokalne memorije. U tu svrhu, razvijen je razred `ImageLoader`, koji omogućuje učitavanje slika iz različitih izvora, kao što su putanje do datoteka i tokovi podataka. Prilikom učitavanja slika, rezolucija se prilagođava zadanoj rezoluciji podataka na kojima je mreža trenirana, a učitane slike se pohranjuju u obliku Nd4j višedimenzionalnog polja. Na taj način, omogućeno je korištenje ovog program u već postojećoj aplikaciji koja radi sa slikama. Za učitavanje slike koristi se razred `NativeImageLoader`. Dio kôda za učitavanje slika iz lokalne memorije prikazan je u nastavku.

```
public INDArray loadImage(String f) throws IOException {
  return loadImages(f);
}
public INDArray loadImages(String... fs) throws
   IOException {
  INDArray images = null;
  for(var f : fs) {
    INDArray output = nativeImageLoader.
    asMatrix(f).div(255.0).permute(0, 2, 3, 1);
    if (images == null) {
      images = output;
    } else {
      images = Nd4j.concat(0, images, output);
    }
  }
  return images;
}
```

Razred `ImageBatchLoader` implementira sučelje `IImageBatchLoader` te os-

tvaruje funkcionalnost učitavanja slika u serijama radi korištenja prilikom predikcije modela. Prilikom inicijalizacije, razred prima 4-dimenzijonalno ND4j polje. Dimenzije su redom: veličina serije, visina slike, duljina slike i broj kanala slike. Pozivom metode `get`, razred vraća izdvojenu seriju podataka iz učitanog polja.

```
@Override
public INDArray get(int batchIndex) {
  if (batchIndex >= this.size()) throw new
     IndexOutOfBoundsException("");
  int batchStartIndex = batchIndex * this.batchSize;
  int batchEndIndex = (int) Math.min(batchStartIndex +
     this.batchSize, images.shape()[0]);
  INDArray batchImages = images.get(NDArrayIndex.interval
     (batchStartIndex, batchEndIndex),NDArrayIndex.all(),
      NDArrayIndex.all(), NDArrayIndex.all());
  return batchImages;
}
```

8.3.2. Učitavanje podataka koji sadrže slike i anotacije

Kako bi se ostvarilo učitavanje podataka koji sadrže slike i anotacije, prvo je potrebno pronaći sve tekstualne datoteke određenog imena unutar direktorija koji sadrži podatke. Tekstualne datoteke sadrže informacije o slikama i anotacijama. U tu svrhu, razvijena je klasa `DatasetVisitor`, koja nasljeđuje `SimpleFileVisitor` u skladu s oblikovnim obrascem posjetitelja. Razvijeni posjetitelj prima ime datoteke kao argument i pretražuje sve poddirektorije unutar direktorija s podacima kako bi pronašao zadanu datoteku. Ako je datoteka pronađena, iz nje se čitaju putanje do slika i anotacija. U nastavku su prikazani najvažniji dijelovi implementacije opisanoga posjetitelja.

```
public class DatasetVisitor extends SimpleFileVisitor<
   Path> {
  private String targetCsvFile;
  private List<String[]> data;
  // Constructor and getters
  @Override
  public FileVisitResult visitFile(Path file,
     BasicFileAttributes attrs) throws IOException {
```

```
    if (file.getFileName().toString().equals(
        targetCsvFile)) {
      List<String> lines = Files.readAllLines(file);
      for (String line : lines) {
        String[] values = line.split(",");
        if (values.length >= 2) {
          String annotation = values[1].trim();
          String path = values[0].trim();
          String fullPath = file.getParent().toString();
          String prefixedValue = fullPath + "\\" + path;
          String[] item = {annotation, prefixedValue};
          data.add(item);
        }
      }
    }
    return FileVisitResult.CONTINUE;
  }
  // implemented other method from SimpleFileVisitor
}
```

Učitani skup putanja i anotacija dostupan je kao lista koja sadrži niz stringova u kojemu su pohranjene putanje i tekstualne anotacije.

Za učitavanje slika i anotacija, razvijena je klasa `DatasetBatchLoader`, koja učitava slike i dvodimenzionale vektore anotacija u serijama na temelju učitanog skupa putanja i anotacija. Podaci se mogu učitati prilikom pozivanja klase ili lijenim načinom, odnosno samo kada se traže određeni podaci u seriji. Klasa implementira metode za učitavanje slika i anotacija.

Metoda `loadImages` učitava slike za zadanu seriju te vraća Nd4j polje s 4 dimenzije: veličina serije, visina slike, duljina slike i broj kanala slike.

```
private INDArray loadImages(List<String[]> subset,
    Integer startIndex, Integer endIndex) {
  int numExamples = subset.size();
  if (startIndex == null) {
    startIndex = 0;
  }
  if (endIndex == null || endIndex > numExamples) {
```

```
    endIndex = numExamples;
  }
  int numImages = endIndex - startIndex;
  INDArray images = Nd4j.zeros(numImages, imageHeight,
     imageWidth, imageChannels);
  for (int i = startIndex; i < endIndex; i++) {
    String[] item = subset.get(i);
    String path = item[1];
    INDArray image = null;
    try {
      image = imageLoader.loadImage(path);
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    images.put(new INDArrayIndex[]{
      NDArrayIndex.point(i - startIndex),
      NDArrayIndex.all(), NDArrayIndex.all(),
      NDArrayIndex.all()}, image);
  }
  return images;
}
```

Anotacije se učitavaju tako da se svaka tekstualna anotacija iz zadane liste pretvori u višedimenzionalni prikaz anotacije. Pretvorba se obavlja korištenjem metode `convertStrToAnnotation`, koja vraća anotaciju u obliku Nd4j višedimenzionalnog polja za zadani tekstualni zapis anotacije. Implementacija metode `loadLabels` je slična implementaciji metode `loadImages`, uz razliku u dijelu gdje se obavlja učitavanje anotacije.

Razred `DatasetBatchLoader` također implementira metodu za dohvaćanje pojedine serije koja sadrži slike i anotacije, kao i metode koje vraćaju broj primjera i broj serija podataka. Dodatno, razred `DatasetBatchLoader` implementira metodu koja stvara objekt sučelja `IImageBatchLoader`, koji se koristi za učitavanje slika tijekom procesa treniranja modela.

```
public IImageBatchLoader imageLoader() {
  return new IImageBatchLoader() {
```

```
    @Override
    public int size() {
      return DatasetBatchLoader.this.size();
    }

    @Override
    public INDArray get(int batchIndex) {
      return DatasetBatchLoader.this.get(batchIndex).
          getImages();
    }

    @Override
    public int batches() {
      return DatasetBatchLoader.this.bathces();
    }
  };
}
```

8.4. Učitavanje modela

Za učitavanje modela i stvaranje predikcija, stvoren je razred Model, koji prima putanju do spremljenog modela u lokalnoj memoriji te ga učitava koristeći metodu `importKerasModelAndWeights` iz razreda `KerasModelImport`, koji je dio programske biblioteke Deeplearning4j. Metoda `predict` obavlja predikciju za podatke koji se učitavaju pomoću instance klase koja implementira sučelje `IImageBatchLoader`. Predikcija se radi u serijama, a rezultati se konkateniraju u jedno Nd4j polje. Stoga za `IImageBatchLoader`, koji ima 100 primjera podijeljenih u serije od 32 primjera, ulaz u model će biti Nd4j polje oblika (32,128,128,1), a model će na izlazu dati predikcije oblika (32,10,10,1). Dobivene predikcije se konkateniraju u jedno Nd4j polje oblika (100,10,10). Kôd razreda `Model` prikazan je u nastavku.

```
public class Model {
  String modelPath;
  ComputationGraph model = null;
  boolean debugFlag;
```

```
  public Model(String modelPath, boolean debugFlag) {
    this.modelPath = modelPath;
    this.debugFlag = debugFlag;
    try {
      model = KerasModelImport.importKerasModelAndWeights
          (modelPath);
    } catch (...) {...}
  }

  public INDArray predict(IImageBatchLoader loader) {
    INDArray concatenatedOutput = null;
    if (debugFlag) {...}
    for (int i = 0; i < loader.batches(); i++) {
      INDArray images = loader.get(i);
      INDArray output = model.outputSingle(images);
      if (concatenatedOutput == null) {
        concatenatedOutput = output;
      } else {
        concatenatedOutput = Nd4j.concat(0,
            concatenatedOutput, output);
      }
      if (debugFlag) {...}
    }
    if (debugFlag) {...}
    concatenatedOutput = concatenatedOutput.get(
        NDArrayIndex.all(), NDArrayIndex.all(),
        NDArrayIndex.all(), NDArrayIndex.point(
        concatenatedOutput.size(3) - 1));
    return concatenatedOutput;
  }
}
```

8.5. Upute za korištenje

Upute za korištenje ovog računalnog programa bit će demonstrirane kroz dva primjera. Prvi primjer prikazuje korištenje modela za predviđanje anotiranih primjera te služi za procjenu točnosti modela koristeći predviđanja i točne oznake za svaki primjer na standardnom izlazu. Pokretanje ovog primjera osigurava da model djeluje identično u okruženju u kojem je izgrađen i treniran (Python) te u okruženju u kojem se koristi (Java). Drugi primjer demonstrira korištenje modela za predviđanje neoznačenih primjera. Proučavanjem ovog primjera, korisnik može razumjeti cjelokupni postupak predviđanja i dobiti ideju za implementaciju predviđanja unutar vlastitog sustava. U svrhu demonstracije, odabran je testni skup podataka koji sadrži 98 primjera.

8.5.1. Primjer za predikciju anotiranih podataka

Primjer prikazuje sljedeće:

1. Učitavanje modela koristeći razred `Model`.
2. Stvaranje instance razreda `DatasetVisitor` i pretraživanje svih datoteka koje sadrže informacije o putanjama do slika i njihovih anotacija. Dohvaćeni podaci su dostupni kao lista podataka.
3. Stvaranje instance razreda `DatasetBatchLoader`, koji učitava podatke u serijama na temelju liste podataka.
4. Stvaranje predikcija pomoću učitanog modela na slikama iz `DatasetBatchLoader`.
5. Ispis točnosti modela, rezultata predikcija za svaki pojedini primjer i rezultata u obliku matrice za jedan primjer.

U nastavku će biti prikazana implementacija opisanog primjera i rezultat njegovog izvođenja (slika 8.2).

```
public class AnnotatedDatasetExample{

  public static void main(String[] args) {
    String modelPath = "resources\\test_model.h5";
    String labelfile = "dataset-info_corrected.csv";
    String dataPath = "resources\\test_data";
```

```
    main(modelPath, labelfile, dataPath);
  }
  public static void main(String modelPath, String
      labelfile, String dataPath) {
    Model model = new Model(modelPath, true);

    DatasetVisitor fileVisitor = new DatasetVisitor(
        labelfile);
    try {
      Files.walkFileTree(Paths.get(dataPath),
      fileVisitor);
    } catch (IOException e) {
      e.printStackTrace();
    }
    List<String[]> data = fileVisitor.getData();

    int imageHeight = 128;
    int imageWidth = 128;
    int imageChannels = 1;
    int batchSize = 32;
    boolean lazyLoading = false;

    DatasetBatchLoader datasetLoader = new
        DatasetBatchLoader(data, imageHeight, imageWidth
        , imageChannels, batchSize, lazyLoading);

    System.out.println("Dataset contains: " +
        datasetLoader.size()+" examples, divided into "
        + datasetLoader.bathces()+ " batches");

    INDArray concatenatedOutput = model.predict(
        datasetLoader.imageLoader());

    System.out.println("concatenatedOutput shape: " +
        Arrays.toString(concatenatedOutput.shape()));
    String[] paths = data.stream().map(s -> s[1].
```

```
            toString()).toArray(String[]::new);
        Tools.predictionSummary(paths, datasetLoader.
            getLabels(), concatenatedOutput);

        System.out.println("Example of one matrix");
        Tools.printRoundedOutput(concatenatedOutput.get(
            NDArrayIndex.point(0)));
    }
}
```

```
Dataset contains: 98 examples, divided into 4 batches
Starting prediction with 98 images.
Processed 1/4 batches (32/98), Total pass time: 1808ms (lap time: 1808ms)
Processed 2/4 batches (64/98), Total pass time: 3427ms (lap time: 1619ms)
Processed 3/4 batches (96/98), Total pass time: 4899ms (lap time: 1472ms)
Processed 4/4 batches (98/98), Total pass time: 5017ms (lap time: 118ms)
Prediction completed in 5017ms.
concatenatedOutput shape: [98, 10, 10]
Prediction summary:
Number of images: 98
Mean accuracy: 1.0
Prediction for each label:
[true] Image:...\p0000001.png, True label: 0036507841, predicted label: 0036507841
[true] Image:...\p0000002.png, True label: 0108069417, predicted label: 0108069417
...
[true] Image:...\p0000098.png, True label: 0036512058, predicted label: 0036512058
Example of one matrix
1 1 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0
```

Slika 8.2: Ispis pokretanja programa: primjer za predikciju anotiranih podataka

8.5.2. Primjer za predikciju neoznačenih podataka

Primjer za predikciju neanotiranih podataka prikazuje sljedeće:

1. Učitavanje modela korištenjem razreda `Model`.
2. Pronalazak svih putanja do slikovnih datoteka unutar zadanog direktorija korištenjem metode `loadFilePaths`.
3. Inicijalizacija razreda `ImageLoader` i učitavnje slika u Nd4j polje.

4. Predikcija korištenjem modela nad slikama koje su učitane u serijama koristeći `ImageBatchLoader`.

5. Ispis predikcije za svaki pojedinačni primer.

U nastavku će biti prikazana implementacija opisanog primjera i rezultat njegovog izvođenja (slika 8.3).

```
public class NonAnnotatedDatasetExample {
  public static void main(String[] args) {
    String modelPath = "resources\\test_model.h5";
    String fileExtension = "png";
    String dataPath = "resources\\test_data";
    main(modelPath, fileExtension, dataPath);
  }
  public static void main(String modelPath, String
     fileExtension, String dataPath) {
    Model model = new Model(modelPath, true);
    String[] imagesPaths = Tools.loadFilePaths(dataPath
       , fileExtension);
    System.out.println("Loaded " + imagesPaths.length
       + " images");

    int imageHeight = 128;
    int imageWidth = 128;
    int imageChannels = 1;
    int batchSize = 32;

    ImageLoader loader = new ImageLoader(imageHeight,
       imageWidth, imageChannels);
    INDArray images = null;
    try {
      images = loader.loadImages(imagesPaths);
    } catch (IOException e) {
      e.printStackTrace();
    }
    INDArray concatenatedOutput = model.predict(new
       ImageBatchLoader(images, batchSize));
```

```
        Tools.outputPrediciton("resources/prediction.csv",
            imagesPaths, concatenatedOutput, true, true);
    }
}
```

```
Loaded 98 images
Starting prediction with 98 images.
Processed 1/4 batches (32/98), Total pass time: 1678ms (lap time: 1678ms)
Processed 2/4 batches (64/98), Total pass time: 3406ms (lap time: 1728ms)
Processed 3/4 batches (96/98), Total pass time: 4863ms (lap time: 1456ms)
Processed 4/4 batches (98/98), Total pass time: 4982ms (lap time: 119ms)
Prediction completed in 4982ms.
...\p0000001.png,0036507841
...\p0000002.png,0108069417

...\p0000098.png,0036512058
```

Slika 8.3: Ispis pokretanja programa: primjer za predikciju neoznačenih podataka

9. Zaključak

U ovom diplomskom radu razvijena je nova metoda za detekciju jedinstvenog matičnog broja akademskog građanina (JMBAG) iz matričnog predloška koristeći model dubokog učenja. Koncept unosa JMBAG-a u matrični predložak i algoritam za automatsku predikciju unesenog JMBAG-a predstavljeni su u članku [5]. Ovaj rad predstavlja novi pristup rješavanju problema detekcije unosa u matrični predložak.

U ovom istraživanju pružena je izravna usporedba dvaju pristupa, ističući prednosti i nedostatke obiju metoda. Razvijeni algoritam dubokog učenja u ovom radu pokazuje visoku točnost u prepoznavanju unosa u matričnom formatu, uz učinkovito zanemarivanje šuma u podacima. Također, u ovom istraživanju je razvijena aplikacija u programskom jeziku Java, koja koristi razvijeni model za generiranje predikcija JMBAG-a iz matričnog predloška.

Predstavljeni se alati i tehnologije korišteni u ovom istraživanju, zajedno sa skupom podataka i procesom anotacije podataka. Detaljno je opisana arhitektura modela, uključujući mrežne blokove i pripadajuće isječke kôda. U radu je detaljno objašnjen cijeli proces treniranja, uključujući hiperparametre modela.

U poglavlju s rezultata prikazana je uspješnost triju modela s različitim složenostima. Sva tri modela postižu performanse visoke kvalitete. Najbolji model postiže točnost od 98.65% u detekciji unosa u matričnom predlošku, pri čemu vrijednosti alfa i beta pogreške ne prelaze 0.5%. Analizom slučajeva u kojima model daje netočne predikcije, otkriveno je da se većina pogreška događa za primjere kojima točna anotacija nije jasno definirana, nego je podložna raspravama. Za ispravnu detekciju takvih rubnih slučajeva, potrebno je povećati njihov broj unutar skupa podataka za učenje.

Budući rad trebao bi se fokusirati na daljnje poboljšanje efikasnosti modela i proširenje skupa podataka s dodatnim primjerima. Također, mogao bi se razviti model koji automatski popunjava matrični predložak željenim JMBAG-om, čime bi se automatizirano proširio skup podataka. Za unaprjeđenje programske aplikacije, bilo bi korisno temeljito istražiti razloge sporijih vremena predikcije u usporedbi s vremenima predikcije u okruženju u kojem je model razvijen.

Tijekom izrade ovog diplomskog rada, istražene su različite teme iz područja dubokog učenja i računalnog vida te su stječene praktične vještine u razvoju dubokog modela za specifični problem. Implementacija dubokog modela u različitim programskim okolinama pridonijela je stjecanju znanja o korištenju modela dubokog učenja bez obzira na programski jezik koji se koristi.

LITERATURA

[1] Understand transposed convolutions. https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-4f5d97b2967. Pristupljeno: 2023-06-01.

[2] U-net explained: Understanding its image segmentation architecture. https://towardsdatascience.com/u-net-explained-understanding-its-image-segmentation-architecture-56e4842e313a. Pristupljeno: 2023-06-01.

[3] Marko Cupic. A case study: using multiple-choice tests at university level courses–preparation and supporting infrastructure. *International Journal of Intelligent Defence Support Systems*, 3(1-2):90–100, 2010.

[4] Marko Čupić, Jan Šnajder, i Bojana Dalbelo Bašić. Post-test analysis of automatically generated multiple choice exams: a case study. U *Proceedings of ICL 2009*, stranice 1–1, 2009.

[5] Marko Čupić, Karla Brkić, Tomislav Hrkać, i Zoran Kalafatić. Supporting automated grading of formative multiple choice exams by introducing student identifier matrices. U *2011 Proceedings of the 34th International Convention MIPRO*, stranice 993–998. IEEE, 2011.

[6] Thomas M Haladyna. *Developing and validating multiple-choice test items*. Routledge, 2004.

[7] Diederik P Kingma i Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Anthony Rhodes, Karyn Bower, i Peter Bancroft. Managing large class assessment. U *Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30*, stranice 285–289, 2004.

[9] Olaf Ronneberger, Philipp Fischer, i Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. U *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, stranice 234–241. Springer, 2015.

[10] Karyn Woodford i Peter Bancroft. Multiple choice questions not considered harmful. U *Proceedings of the 7th Australasian conference on Computing education-Volume 42*, stranice 109–116, 2005.

Strojno raspoznavanje studentskih identifikacijskih brojeva iz matričnog predloška

Sažetak

Ovaj diplomski rad predstavlja novi pristup identifikaciji studenata tijekom ispita i provjera znanja. Predložena metoda koristi matrični predložak na ispitu, gdje studenti selektivno označavaju kvadratiće, unoseći znamenke koje odgovaraju njihovom jedinstvenom matičnom broju akademskog građanina (JMBAG). Razvijen je specijalizirani model dubokog učenja temeljen na modificiranoj U-Net arhitekturi, koji precizno prepoznaje i interpretira označene kvadratiće unutar matričnog predloška. Razvijeni model postiže visoku točnost u identifikaciji unesenih JMBAG-a unutar predloška uz zanemarivanje šuma u podacima. Nadalje, razvijena je aplikacija koja koristi model kako bi omogućila predviđanje osobnih brojeva studenata na temelju skeniranih dijelova ispita koji sadrže predložak. Uvođenje ovog inovativnog sustava identifikacije predstavlja značajan napredak u području ispita i provjere znanja, zamjenjujući konvencionalni ručni unos osobnih podataka učinkovitim, brzim i točnim postupkom identifikacije.

Ključne riječi: identifikacija studenata, detekcija s više oznaka, U-Net, autmatizirani proces identifikacije, model dubokog učenja.

Machine-based recognition of student identifiers from matrix template

Abstract

This master's thesis introduces a novel approach to student identification during exams and knowledge tests. The proposed method utilizes a matrix template on the exam sheet, where students selectively mark squares by entering digits corresponding to their unique academic citizen identification number. A specialized deep learning model, based on a modified U-Net architecture, has been developed to accurately recognize and interpret the marked squares within the matrix template. The designed model achieves high accuracy in identifying the entered student IDs within the template while disregarding data noise. Furthermore, an application has been developed that utilizes the trained model to provide the functionality of predicting student identification numbers based on scanned sections of the exam containing the template. The introduction of this innovative identification system represents a notable advancement in the field of exams and knowledge tests, replacing the conventional manual entry of personal data with an efficient, fast, and accurate identification procedure.

Keywords: student identification, multi-label detection, U-Net, automated identification process, deep learning model.