Sustav za prepoznavanje rukom pisanog teksta

Zidarić, Matija

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Matija Zidarić

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:
doc. dr. sc. Tomislav Stipančić, dipl. ing.

Student:
Matija Zidarić

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Srdačno zahvaljujem svojem mentoru doc. dr. sc. Tomislavu Stipančiću na stručnim savjetima, strpljenju i motivaciji za završetak preddiplomskog studija i završnog rada. Razgovor sa dobrim mentorom uvijek ostavi utisak na studenta.

Hvala mojoj obitelji na podršci i strpljenju kada je bilo potrebno. Male stvari se jako cijene.

Želim se zahvaliti svim prijateljima koji su bili dio ove Zmajske odiseje i koji su sudjelovali u njoj. Ne želim nikoga posebno navoditi, jer oni koji znaju da sam im iskreno zahvalan od srca to već znaju i bez riječi.

M. Zidarić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagreb Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Matija Zidarić** JMBAG: **0035204441**

Naslov rada na hrvatskom jeziku: **Sustav za prepoznavanje rukom pisanog teksta**

Naslov rada na engleskom jeziku: **Handwriting recognition system**

Opis zadatka:

U radu je potrebno osmisliti i napraviti programsku aplikaciju – sustav za prepoznavanje rukom napisanog teksta. U sklopu softverskog rješenja potrebno je:

- proučiti računalne biblioteke OpenCV, Keras i TensorFlow,
- trenirati računalni model za optičko prepoznavanje znakova na skupu podataka,
- oblikovati prikladno korisničko sučelje za interakciju korisnika i sustava,
- implementirati računalni model za prepoznavanje rukom napisanog teksta koristeći Python programsko okruženje.

Razvijeno cjelovito rješenje potrebno je eksperimentalno evaluirati. U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:

30. 11. 2021.

Datum predaje rada:

1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:

1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Zadatak zadao:

Doc. dr. sc. Tomislav Stipančić

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

SAŽETAK

Razvojem računala, danas su algoritmi neuronskih mreža postali izuzetno moćni i rapidno se razvijaju. Rukom pisani tekst je zbog svoje unikatnosti jako zanimljivo područje za primjenu strojnog učenja. Suvremene duboke neuronske mreže nude nova rješenja za poznate probleme. U ovome radu je osmišljena programska aplikacija korištenjem biblioteka OpenCV, Keras i TensorFlow. Korišteni programski jezik je Python i ResNet kao arhitektura neuronske mreže. Provedena je i evaluacija dobivenog modela na primjerima rukom pisanog teksta.

Ključne riječi: rukom pisani tekst, umjetna inteligencija, strojno učenje, Python, Keras, OpenCV, ResNet

SUMMARY

The development of computers has resulted in extremely powerful neural network algorithms which continue to develop rapidly. Due to its uniqueness, handwritten text is a very interesting area for the application of machine learning. Modern deep neural networks offer new solutions to existing problems. In this thesis, a software application was designed using OpenCV, Keras and TensorFlow libraries. Python was used as the programming language and ResNet as the neural network architecture. An evaluation of the obtained model was also carried out on the examples of handwritten text.

Keywords: handwritten text, artificial intelligence, machine learning, Python, Keras, TensorFlow, OpenCV, ResNet

1. UVOD

1.1. Umjetna inteligencija i OCR

Dio računalne znanosti koji se bavi problematikom obavljanja zadaća koje zahtjevaju neki oblik inteligencije, odnosno da neki neživi sustav pokazuje karakteristike inteligentnog bića naziva se umjetna inteligencija.[1] Moderni svijet je pun sustava koji imaju implementiran oblik umjetne inteligencije u sebi, od kamera koje broje vozila na križanjima, sustava za praćenje ljudi koji ulaze i izlaze iz objekata, pa sve do umjetne inteligencije u računalnim igrama. Kao zasebna znanstvena disciplina, umjetna inteligencija se izdvojila sredinom 20. stoljeća, a od onda je imala više uzleta, posebice sa razvojem moćnijih računala i njihovim širim pristupom. Sve to je rezultiralo korištenjem novih metoda umjetne inteligencije koje su prije bile nezamislive.

Ovaj rad se bavi temom izrade aplikacije za prepoznavanje rukom pisanoga teksta korištenjem alata umjetne inteligencije. Sama problematika nije nova, ali je svakako i dalje aktualna. Velika razlika se pojavljuje između standardiziranoga teksta na računalu i onog ljudskog. Ne postoje dva identična otiska prsta, pa tako vjerojatno ne postoje niti dva identična rukopisa.

Mogućnost prepoznavanja rukom pisanoga teksta otvara jako veliki broj mogućnosti primjene. Digitalizacija starih tekstova, modernizacije poduzeća i automatizacija poslovanja, zdrastvo, školstvo, administracija. Čovjek gotovo svaki dan napiše barem nekoliko riječi, a sposobnost računala da dobro prepozna te informacije bi bile jako korisne.

1.2. Struktura rada

U prvome dijelu rada nalazi se teorijska podloga s najvažnijim pojmovima te kratki opis korištenih alata i programa.

Drugi dio rada se sastoji od opisa rada modela za prepoznavanje rukom pisanoga teksta, te opisa treninga duboke konvolucijske mreže korištenjem *ResNet* arhitekture. Opisano je i grafičko korisničko sučelje.

Treći dio rada opisuje eksperimentalni dio u kojem se evaluirao rad aplikacije i komentiraju rezultati.

U zadnjem dijelu rada nalazi se kritički osvrt, te moguća poboljšanja i zaključak.

Na kraju rada je prilog sa python računalnim kodom.

2. TEORIJSKA PODLOGA RADA

2.1. Strojno učenje (eng. *Machine learning*)

Strojno učenje (Machine Learning) je vrsta umjetne inteligencije koja omogućava računalnim aplikacijama da postanu preciznije u predviđanju ishoda bez da su za to eksplicitno programirane. Algoritmi strojnog učenja koriste podatke iz prošlosti kao ulazne podatke za predviđanje novih izlaznih vrijednosti. Klasično strojno učenje se kategorizira prema načinu na koji algoritam uči predviđanja. Četiri su osnovna pristupa: nadgledano učenje, nenadgledano učenje, polunadgledno učenje i pojačano učenje.

Nadgledano učenje je oblik strojnog učenja u kojem se markirani podaci za trening unose u algoritam te se definiraju varijable za koje se želi da algoritam procijeni korelacije. Specificirani su ulaz i izlaz algoritma. Nadgledano učenje je korisno za binarnu klasifikaciju, regresijsko modeliranje, razvrstavanje klasa itd.

Nenadgledano učenje koristi nemarkirane podatke. Algoritam prolazi kroz skupove podataka i traži bilo kakvu smislenu vezu. Podaci za treniranje algoritma, te predviđanja ili preporuke koje izlaze, unaprijed su određeni. Većina vrsta dubokog učenja, uključujući neuronske mreže, algoritmi su nenadgledanog učenja. Područja u kojima se koristi su grupiranje, otkrivanje anomalija, asocijacijsko rudarenje, smanjenje dimenzionalnosti itd.

Polunadgledano učenje je pristup strojnom učenju koje koristi kombinaciju nadgledanog i nenadgledanog učenja. Algoritam uglavnom dobiva malu količinu markiranih podataka, ali je slobodan samostalno razviti razumijevanje skupa podataka, te može to primijeniti na neoznačene podatke. Performanse algoritma se obično poboljšavaju kada se trenira na označenim podacima, no markiranje podataka može biti dugotrajno i skupo. Zato je polunadgledano za neke primjene dobar odabir između nadgledanog i nenadgledanog učenja. Područja u kojima se koristi polunadgledano učenje su strojno prevođenje, otkrivanje prevara, markiranje podataka itd.

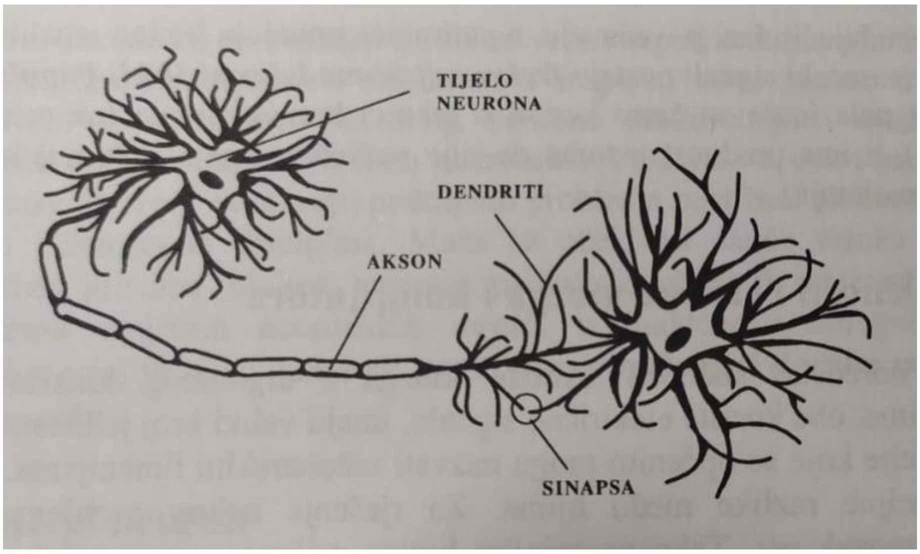
Podržano učenje radi na način da se isprogramira algoritam s jasnim ciljem i propisanim skupom pravila za postizanje toga cilja. Drugi algoritam služi za davanje nagrada kada se izvrši radnja koja je korisna za krajnji cilj i za davanje kazni ukoliko se izvršena radnja udaljava od krajnjeg cilja. Podržano strojno učenje se koristi u područjima kao što su robotika, računalne igre, evolucijski algoritmi, upravljanje resursima itd. [2]

2.2. Umjetne neuronske mreže

Umjetne neuronske mreže podskup su strojnog učenja i inspirirane su ljudskim mozgom, odnosno oponašaju način na koji biološki neuroni signaliziraju jedni drugima. Sastoje se od čvorova koji sadrže ulazni sloj, jedan ili više skrivenih slojeva i izlazni sloj. Svaki čvor ili umjetni neuron povezuje se s drugima i ima pridruženu težinu i prag. U slučaju da je vrijednost pojedinačnog čvora iznad navedene vrijednosti, taj se čvor aktivira te šalje podatke idućem sloju mreže, u protivnom se podaci ne prosljeđuju. Neuronske mreže oslanjaju se na podatke za obuku kako bi poboljšale i naučile svoju točnost tijekom vremena. Nakon nekog vremena, ti algoritmi učenja su fino podešeni za točnost i postaju moćni alati u računalnoj znanosti i umjetnoj inteligenciji što omogućuje grupiranje i klasificiranje podataka velikom brzinom. [3]

2.2.1. Građa i struktura bioloških i umjetnih neurona

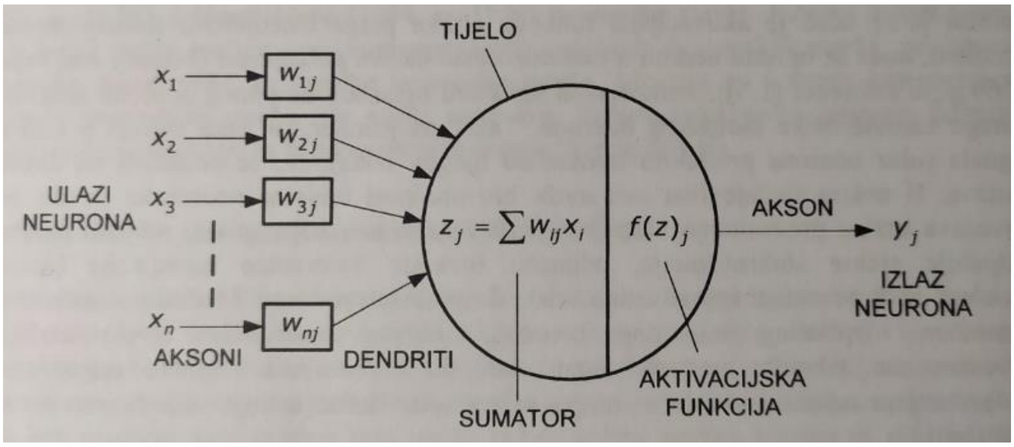
Biološki neuron se sastoji od tijela neurona te mnoštva dendrita koji ga okružuju. Akson je tanka cjevčica koja je jednim krajem povezana s tijelom neurona, a drugim dijelom se dijeli na niz grana. Krajevi grana završavaju zadebljanjima koja diraju dendrite ili rjeđe druge krajeve neurona. Sinapsa je mali razmak između završetka aksona prethodnog neurona i dendrita ili sljedećeg neurona. Impulsi, odnosno izlazi, putuju kroz akson do sinapsi odakle se signali različitih težina šalju kroz dendrite ili direktno na tijelo drugih neurona. Neuron će poslati impuls kroz svoj akson u slučaju da je doveden u stanje dovoljne uzbude.



Slika 2.1. Prikaz pojednostavljene strukture biološkog neurona [4]

Jedan neuron može generirati impuls koji će u neuronskoj mreži smiriti ili pak aktivirati mnoštvo drugih neurona, od kojih se svaki može istovremeno uzbuđivati od mnoštva drugih neurona. Time se realizira visoki stupanj međusobne povezanosti neurona u neuronskoj mreži. Drugim riječima, kompleksnost funkcija neuronske mreže se ostvaruje kompleksnošću veza, to jest težina među neuronima, prije nego kompleksnošću svakog neurona posebno. [4]

Umjetni neuron se izvodi s idejom oponašanja osnovne funkcije biološkog neurona. Tijelo biološkog neurona se zamjenjuje sumatorom, ulaze u sumator preuzimaju dendriti, a aksoni su izlazi iz sumatora, dok se prag osjetljivosti bioloških neurona preslikava na takozvane aktivacijske funkcije. Funkcije sinaptičke veze biološkog neurona s okolinom se preslikavaju u težinske faktore preko kojih se ostvaruje veza sa okolinom. [4]



Slika 2.2. Prikaz strukture umjetnog neurona [4]

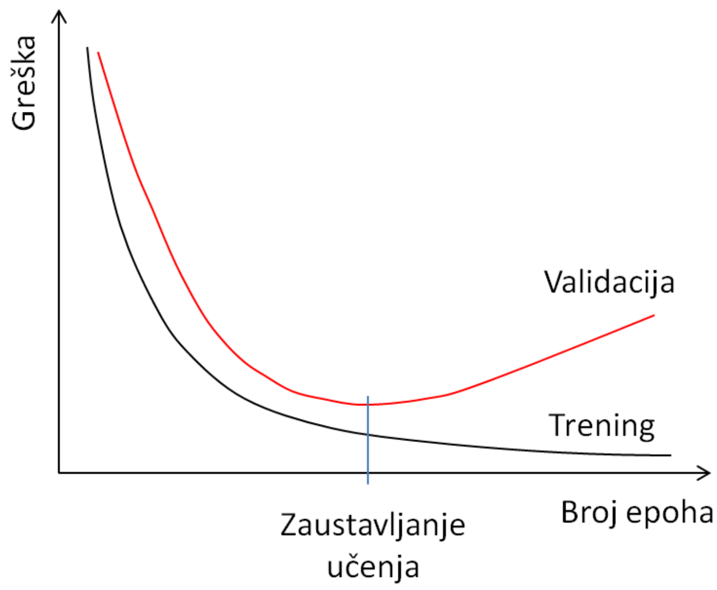
2.2.2. Težinski faktori i aktivacijske funkcije

Težinski faktori mogu biti pozitivni ili negativni broj, odnosno mogu poprimiti vrijednost neke funkcije (varijabilan težinski faktor). Vrijednost težinskoga faktora 0 označava nepostojanje veze s okolnim neuronima. Intenzitet veze ovisi o vrijednosti težinskog faktora (modulu), a karakteristika veze o predznaku (pozitivni ili negativni). Težinski faktori povezuju izlaze iz okoline neurona i drugih neurona (aksona) s ulazima sumatora (dendriti). Izlaz iz sumatora se povezuje na ulaz aktivacijske funkcije, koja na svome izlazu reproducira izlaz umjetnoga neurona. [4]

Aktivacijske funkcije se dijele na linearne i nelinearne. Kod linearnih, izlaz sumatora se množi sa nekim faktorom (pojačanjem) i tako dobiva izlaz neurona. Nelinearne funkcije mogu poprimiti različite oblike, a najčešće se koriste funkcije praga osjetljivosti, sigmoidalne, hiperbolične i harmoničke funkcije. [4]

2.2.3. Učenje umjetne neuronske mreže

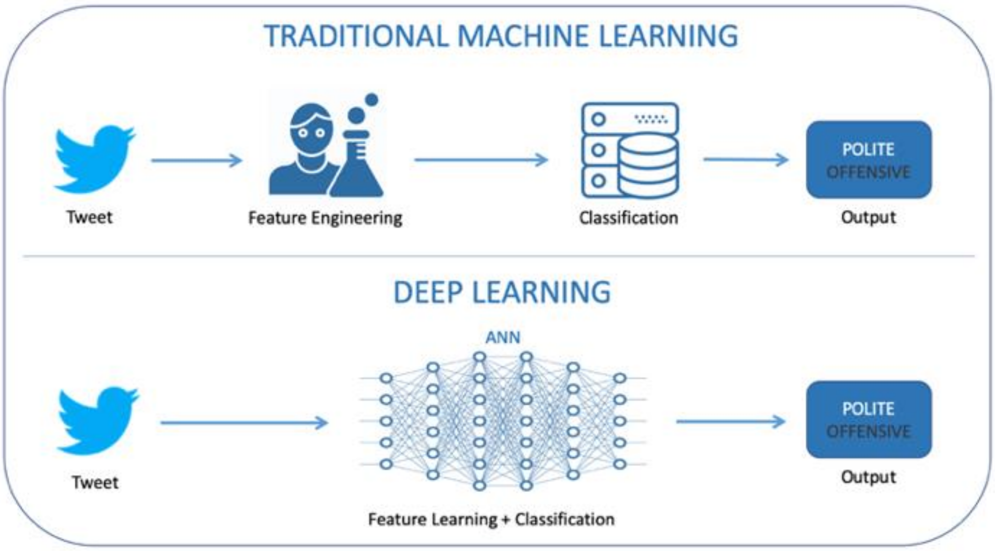
Učenje ili treniranje umjetne neuronske mreže iterativni je proces podešavanja težinskih faktora u mreži. Širenje unatrag (*backpropagation*) otkriva ispravne težinske faktore koje treba primijeniti na čvorove, odnosno uspoređuje izlaz iz mreže sa referentnom vrijednosti i traži minimalnu grešku, odnosno odstupanje. Algoritam se nakon usporedbe izlazne vrijednosti sa stvarnom vrijednošću vraća u slojeve neuronske mreže i podešava vrijednosti težinskih faktora. Proces se odvija tako dugo dok se ne dobije minimalno odstupanje. Zbog mogućnosti takozvane „pretreniranosti" mreže (*overfitting*), učenje se prekida u trenutku kada pogreška u modelu počinje rasti. Pretreniranost je situacija kada mreža daje jako dobre rezultate na skupu podataka na kojima je razvijena, a van tog skupa pokazuje dosta lošije podatke. U svrhu razvoja modela koriste se tri skupa podataka: skup za učenje, skup za provjeru i skup za testiranje. Podaci iz skupa za učenje se stavljaju u mrežu, te se uspoređuju sa skupom za provjeru na kraju jedne iteracije. Na kraju se dodatno provjerava model sa skupom za testiranje. [5]



Slika 2.3. Prikaz učenja neuronske mreže sa zaustavljanjem

2.3. Duboko učenje (eng. *Deep learning*)

Podskup strojnog učenja, odnosno neuronska mreža s tri ili više slojeva. Ovaj tip umjetnih neuronskih mreža pokušava simulirati ponašanje ljudskog mozga, to jest, na taj način može „učiti" iz velike količine podataka. Mreža s jednim slojem može dati dobre rezultate, ali dodatni skriveni slojevi mogu znatno poboljšati optimizaciju i točnost. Duboko učenje se razlikuje od strojnog učenja po vrsti podataka koje koristi i metodama kojima uči. Algoritmi strojnog učenja uglavnom koriste strukturirane, markirane podatke za modeliranje predviđanja. To nužno ne mora značiti da ne koriste nestrukturirane podatke, već da, ako ih koriste, da prolaze kroz svojevrsnu predobradu kako bi se podaci organizirali u strukturirani format. Duboko učenje eliminira dio predobrade podataka, a to omogućuje da takvi algoritmi mogu unijeti i obraditi nestrukturirane podatke poput teksta i slike, što pak omogućuje automatizaciju izvlačenja značajki koja uklanja dio ovisnosti o ljudskome radu. Algoritmi dubokog učenja mogu sami odrediti koje su značajke važne, dok u strojnome učenju tu hijerarhiju značajki ručno postavlja čovjek. Dalje se kroz procese gradijentnog spuštanja i povratne propagacije, algoritam dubokog učenja prilagođava za točnost, što mu daje preciznija predviđanja na izlazu. Ulazni i izlazni slojevi duboke neuronske mreže su vidljivi slojevi, dok su slojevi između skriveni. Algoritmi dubokog učenja mogu biti izuzetno složeni, a jedni od najšire korištenih su konvolucijske neuronske mreže (*Convolutional Neural Networks* - *CNN*) te povratne neuronske mreže (*Recurrent neural Network* - *RNN*). [6]



Slika 2.4. Usporedba strojnog učenja i dubokih neuronskih mreža [7]

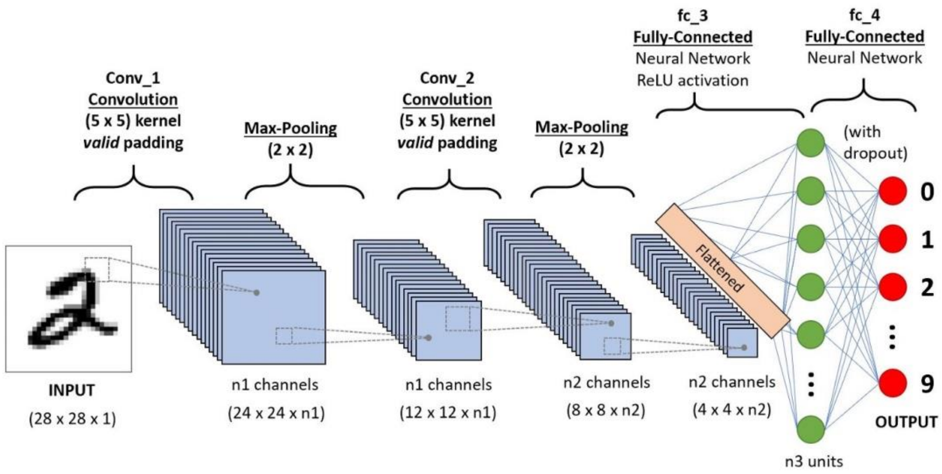
2.4. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže razlikuju se od drugih neuronskih mreža svojim superiornim performansama s ulaznim signalima slika, govora ili zvuka. Sastoje se od tri glavna sloja: konvolucijski sloj (*convolution layer*), sloj udruživanja (*pooling layer*) i potpuno povezani sloj (*Fully-connected – FC layer*).

Konvolucijski sloj je prvi sloj konvolucijske mreže, a kasnije se može kombinirati s drugim konvolucijskim slojevima ili slojevima udruživanja. Završni sloj je potpuno povezani sloj. Sa svakim slojem konvolucijska neuronska mreža postaje sve kompleksnija, identificirajući veće dijelove slike. Raniji slojevi fokusirani su na jednostavnije značajke poput boja i rubova. Kako podaci napreduju kroz mrežu, prepoznaju se sve veći elementi ili oblici objekta dok se konačno ne identificira željeni objekt. Konvolucijski sloj je temeljni građevni sloj i u njemu se odvija većina izračuna, a zahtjeva nekoliko komponenti poput ulaznih podataka, filtera i mapa značajki. Detektor značajki je 2D niz težina koje predstavljaju dio slike. Iako mogu varirati u veličini, filteri su obično matrice veličine 3x3, što također određuje veličinu receptivnog polja. Filter se primjenjuje na područje slike, a skalarni umnožak se izračunava između ulaznih piksela i filtera. Taj skalarni umnožak se zatim ubacuje u izlazni niz. Nakon toga filter se pomiče korak po korak te se postupak ponavlja sve dok kernel ne prođe kroz čitavu sliku. Konačni izlaz iz niza skalarnih umnožaka iz ulaza i filtera poznat je kao mapa značajki, aktivacijska mapa ili konvolvirana mapa. Nakon svake operacije konvolucije, mreža primjenjuje *Rectified Linear Unit* (ReLU) transformaciju na mapu značajki te time uvodi nelinearnost u model. Dijagram hijerarhije značajki u konvolucijskim neuronskim mrežama u konačnici osigurava pretvorbu slike u numeričke vrijednosti nakon prolaza kroz konvolucijski sloj te na taj način omogućava neuronskoj mreži jednostavniju interpretaciju i izdvajanje relevantnih uzoraka.

Sloj udruživanja (*pooling layer*) objedinjuje slojeve, odnosno smanjuje uzorkovanje i provodi smanjenje dimenzioniranosti te tako smanjuje broj parametara u ulazu. Slično kao i kod konvolucijskog sloja, operacija udruživanja prebacuje filter preko cijelog ulaza, ali sada taj filter nema nikakve težine. Umjesto toga, kernel primjenjuje funkciju agregacije na vrijednosti unutar receptivnog polja i tako popunjuje izlazni niz. Postoje dvije glavne vrste udruživanja, maksimalno udruživanje (*Max pooling*) i prosječno udruživanje (*Average pooling*). Maksimalno udruživanje se vrši na način da se filter pomiče preko ulaza i odabire piksel s maksimalnom vrijednošću za slanje u izlazni niz. Ovaj pristup se često koristi s prosječnim udruživanjem. Prosječno udruživanje računa prosječnu vrijednost unutar receptivnog polja dok se filter kreće preko ulaza i šalje u izlazni niz. Iako se puno informacija gubi u sloju udruživanja, on ima i određene prednosti poput smanjenja složenosti, poboljšavanja učinkovitosti i određena zaštita od pretreniranosti.

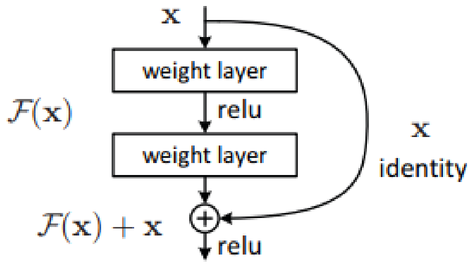
Potpuno povezani sloj povezuje svaki čvor u izlaznom sloju izravno s čvorom u prethodnom sloju. Ovaj sloj obavlja zadatak klasifikacije na temelju značajki estrahiranih kroz prethodne slojeve i njihove različite filtere. Dok konvolucijski sloj i sloj udruživanja obično koriste ReLu funckije, potpuno povezani slojevi obično koriste softmax aktivacijsku funkciju za odgovarajuću klasifikaciju ulaza te se na taj način dobiva vjerojatnost od 0 do 1. [8]



Slika 2.5 Primjer CNN za klasifikaciju rukom pisanog teksta [9]

2.5. ResNet

ResNet (*Residual Network*) je jedan on najpopularnijih i najuspješnijih modela dubokog učenja danas. Problem učenja vrlo dubokih neuronskih mreža je olakšan korištenjem zaostalih blokova (*residual blocks*) od kojih je ResNet sastavljena. Kao što je prikazano na slici 2.6, postoji izravna veza koja preskače neke slojeve modela. Ta veza se naziva „vezom za preskakanje“ (*skip connection*) i srce je zaostalih blokova. Ona mijenja izlaz iz mreže. Bez veze za preskakanje, ulaz X se množi s težinskim faktorima sloja što daje izraz pristranosti. Dimenzija ulaza može varirati od dimenzije izlaza što se može dogoditi u konvolucijskome sloju ili sloju udruživanja. Taj problem se može riješiti na dva načina. Prvi način je nula koja je kombinirana s vezom za preskakanje kako bi se povećala dimenzija ulaza. Drugi način su konvolucijski slojevi 1x1 dodani ulazu kako bi se dobile odgovarajuće dimenzije. Tehnika veza za preskakanje u ResNetu rješava problem nestajanja gradijenta u dubokim neuronskim mrežama tako da dopušta alternativni prečac kojim taj gradijent putuje. Veza za preskakanje može pomoći i na način da sloj koji šteti performansama arhitekture bude preskočen regularizacijom. [10]



Slika 2.6. Residualni građevni blok [10]

2.6. Računalni vid (eng. Computer Vision)

Računalni vid je područje umjetne inteligencije koje omogućava računalima i sustavima da izvuku korisne informacije iz digitalnih slika, videa i drugih vizualnih ulaza, te da poduzmu određene radnje ili daju preporuku na temelju tih informacija. Treniranje računalnog vida mora biti u razumnom vremenu koristeći kamere, podatke i algoritme za razliku od čovjeka koji ima prednost doživotnog konteksta za učenje. Velika količina podataka je potrebna za dobro učenje računalnog vida, a najčešće se koriste tehnologije dubokog učenja i konvolucijskih neuronskih mreža. Konvolucijske neuronske mreže pomažu algoritmu strojnoga učenja ili dubokog učenja na način da razlažu sliku na piksele kojima se daje marking. Te markinge koristi u konvolucijama kako bi predvidjela što vidi. Neuronska mreža pokreće konvolucije i provjerava točnost svojih predviđanja u nizu ponavljanja sve dok se predviđanja ne počnu ostvarivati. Slično kao što čovjek stvara sliku na daljinu, neuronska mreža prvo razaznaje oštre rubove i jednostavne oblike, a zatim popunjava informacije ponavljanjem konvolucija i predviđanja. Konvolucijske neuronske mreže se koriste za razumijevanje pojedinačnih slika, a povratne neuronske mreže (RNN) se koriste na sličan način za obradu video aplikacija za niz slika koje su povezane. [11]

Postoje razni problemi s kojima se računalni vid susreće poput segmentacije, osvjetljenja, skaliranja, deformiranja, okluzija i preklapanja, pozadinske buke, kretanja, varijacija unutar iste klase, višeznačnosti, percepcije itd. Dio tih problema se rješava prilikom prikupljanja samih podataka poput drugog načina osvjetljenja (direktno, pozadinsko kupolno...) ili pak korištenjem kamera koje mogu vidjeti u drugome spektru svjetla (infracrvene kamere). Drugi način za rješavanje problema je programski, točnije, koriste se različiti filteri poput box filtera, gaussian filtera, median filtera i slično. Problemi segmentacije i varijacija unutar iste klase se mogu riješiti korištenjem umjetnih neuronskih mreža, ali ta rješenja ne moraju nužno biti jednostavna. [12]

2.7. Baze podataka za učenje

2.7.1. MNIST

MNIST baza podataka (Modified National Institute of Standards and Technology database – Modificirana baza podataka Nacionalnog instituta za standarde i tehnologiju) velika je zbirka rukom pisanih brojeva. Sadrži set za obuku od 60 000 primjera i set za testiranje od 10 000 primjera. To je podskup veće NIST posebne baze podataka 3 koju su zapisali zaposlenici Ureda za popis stanovništva Sjedinjenih Država i posebne baze podataka 1 koju su zapisali srednjoškolci. Znamenke su normalizirane prema veličini i centrirane na slici fiksne veličine. Izvorne crno-bijele (dvorazinske) slike iz NIST-a bile su normalizirane po veličini kako bi stale u okvir od 20x20 piksela uz očuvanje omjera slika. Rezultirajuće slike sadrže razine sive kao rezultat tehnike zaglađivanja koju koristi algoritam normalizacije. Slike su centrirane na slici 28x28 piksela izračunavanjem središta mase piksela i translacijom slike da se ta točka postavi u središte polja 28x28 piksela. [13]

Slika 2.7. Primjer MNIST baze simbola [14]

2.7.1. Kaggle A-Z

Kaggle A-Z je skup podataka koji sadrži 26 rukom pisanih znakova engleskog alfabeta. Bazira se na NIST posebnoj bazi podataka 19 koja sadrži preko 800 000 rukom pisanih slova od 3600 pisaca. Značajke te baze podataka je niz slika 28x28 piksela i svako slovo je centrirano u okvir 20x20 piksela. Slike su spremljene u nijansama sive. [15]

2.8. Data Augmentation

Data Augmentation (Povećanje podataka) je postupak dobivanja većeg broj podataka za treniranje umjetne neuronske mreže. Suvremene neuronske mreže mogu sadržavati parametre reda veličine milijuna, što zahtjeva proporcionalnu količinu primjera za učenje. Kako bi se dobila veća količina podataka od ovih koje već imamo, vrše se određene manipulacije podatcima. Konvolucijska neuronska mreža može robusno klasificirati objekte čak i ako su oni različite orijentacije, odnosno ima svojstvo invarijantnosti. Drugim riječima, KNM može biti invarijantna prema translaciji, točki gledišta, veličini ili osvjetljenju, odnosno kombinaciji navedenog. Korištenje tih tehnika manipulacije podacima za učenje može se izbjeći da mreža nauči krivu karakteristiku ili krive obrasce (npr., ne prepoznaje razliku između A i B automobila, već da je automobil A uvijek okrenut u lijevo) i krivo daje rezultate, iako je po izlazu točnost recimo veća od 95%. Popularne metode augumentacije podataka su [16]:

- preokretanje - vertikalno i horizontalno
- rotacija
- skaliranje - prema van ili prema unutra
- obrezivanje - nasumično rezanje dijelova slike
- translacija - pomicanje po x i/ili y osi
- gaussov šum - uklanjanje visokih frekvencija korištenjem filtera kako bi se izbjegla pretreniranost, odnosno dodavanje šuma za raznolikost podataka

2.9. Python

Python je interpretacijski viši programski jezik opće namjene. Stvorio ga je Guido van Rossum 1990. godine, a ime je dobio po televizijskoj seriji *Monty Python's Flying Circus*. U pythonu se može programirati objektno orijentirano, strukturno i aspektno orijentirano što čini korištenje jako fleksibilnim. To je jedan od glavnih razloga koji čine python jednim od najpopularnijih programskih jezika današnjice. Zbog toga što je python interpretacijski jezik, programi napisani u njemu se vrše sporije nego kod kompajlerskih jezika kao što su C, C++ i slični. Iznimno bogate biblioteke čine python jako zanimljivim i korisnim znanstvenicima, učenicima, ali i u komercijalne svrhe. [17]

2.10. TensorFlow

TensorFlow je platforma otvorenog računalnog koda za strojno učenje. Sadrži fleksibilan i sveobuhvatan sustav alata, programskih knjižnica i računalnih kodova znanstvene zajednice koji omogućuju korisnicima korištenje najsuvremenijeg strojnog učenja uz jednostavnu izradu i implementaciju u aplikacijama. Izvorno su ga razvili Google Brain istraživači i inženjeri unutar Googleove organizacije za istraživanje umjetne inteligencije za provođenje istraživanja strojnog učenja i dubokih neuronskih mreža. Dovoljna općenitost sustava omogućuje njegovu primjenu u velikom broju različitih domena gdje je strojno učenje koristan alat. TensorFlow pruža stabilna programska sučelja za aplikacije poput Python-a i C++, a može se koristiti i u drugim programskim jezicima. [18]

2.11. Keras

Keras je API (*Application Programming Interface* – programsko sučelje aplikacije) za duboko učenje temeljen na platformi TensorFlow i napisan je u programskome jeziku Python. Razvijen je na fokusu brzog eksperimentiranja. Ključ dobrog istraživanja je mogućnost da se brzo dođe od ideje do rezultata. Keras omogućuje jednostavnost, iako nije jednostavan, odnosno smanjuje kognitivno opterećenje korisnika kako bi se isti mogao usredotočiti na rješavanje samog problema. Fleksibilnost kerasa je u načelu progresivnog otkrivanja složenosti: jednostavne operacije moraju biti brze i lako izvedene, a proizvoljne napredne operacije su moguće zbog već naučenih postupaka. Industrijska razina performansi i skaliranje operacija, odnosno učinkovito izvršavanje operacija na različitim procesorima čini keras izuzetno moćnim alatom koji koriste NASA, YouTube itd. [19]

2.12. OpenCV

OpenCV (*Open Source Computer Vision Library*) biblioteka otvorenog koda za računalni vid i strojno učenje. Napravljen je kao zajednička infrastruktura za aplikacije računalnog vida i ubrzavanje korištenja strojne percepcije u komercijalnim proizvodima. Biblioteka ima preko 2500 optimiziranih algoritama, od skupa klasičnih do najsuvremenijih algoritama strojnog učenja i računalnog vida. Ti algoritmi se mogu koristiti za identifikaciju objekata, prepoznavanje lica i klasifikaciju ljudskih radnji u videozapisima, praćenje pokreta kamere i objekata, stvaranje 3D oblaka točaka iz stereo kamera i izdvajanje 3D modela itd. Biblioteka se intenzivno koristi u istraživanjima, ali i u komercijalne svrhe. Neke od primjena biblioteke su otkrivanje upada u videonadzor, praćenje rudarske opreme, robotska navigacija, otkrivanje nesreća utapanja, pokretanje interaktivne umjetnosti, pregled avionskih pista za krhotine, vizualna inspekcija itd. OpenCV ima C++, Python, Java i MATLAB sučelje, te podržava sve glavne operacijske sustave, a uglavnom teži primjeni u vizijskim aplikacijama u stvarnom vremenu. [20]

2.13. Google colaboratory

Google colaboratory (skraćeno *Colab*) je web stranica koja omogućuje korištenje Python programskoga jezika u internet pregledniku. Prednost Colab-a je u tome što ne zahtijeva nikakvu konfiguraciju, omogućava besplatan pristup GPU-ima i jednostavan je za podijeliti sa drugim korisnicima. Besplatan pristup je omogućen studentima i znanstvenicima. Colab omogućava i kombinaciju računalnog koda i teksta u jednome dokumentu, tako da se mogu objediniti slike, HTML, LaTeX i drugi, a kreirani dokumenti se spremaju na Google Drive. Velika prednost Colaba je korištenje Google-ovih grafičkih i procesorskih jedinica, što osigurava velikom broju ljudi da, bez obzira na računalo kod kuće, mogu raditi na algoritmima umjetne inteligencije. [21]

2.14. Predobrada slika

U svrhu boljeg rada aplikacije, prije samog unosa u model koji prepoznaje rukom pisane znakove, odrađena je određena obrada slike. U nastavku će kratko biti opisani ti postupci.

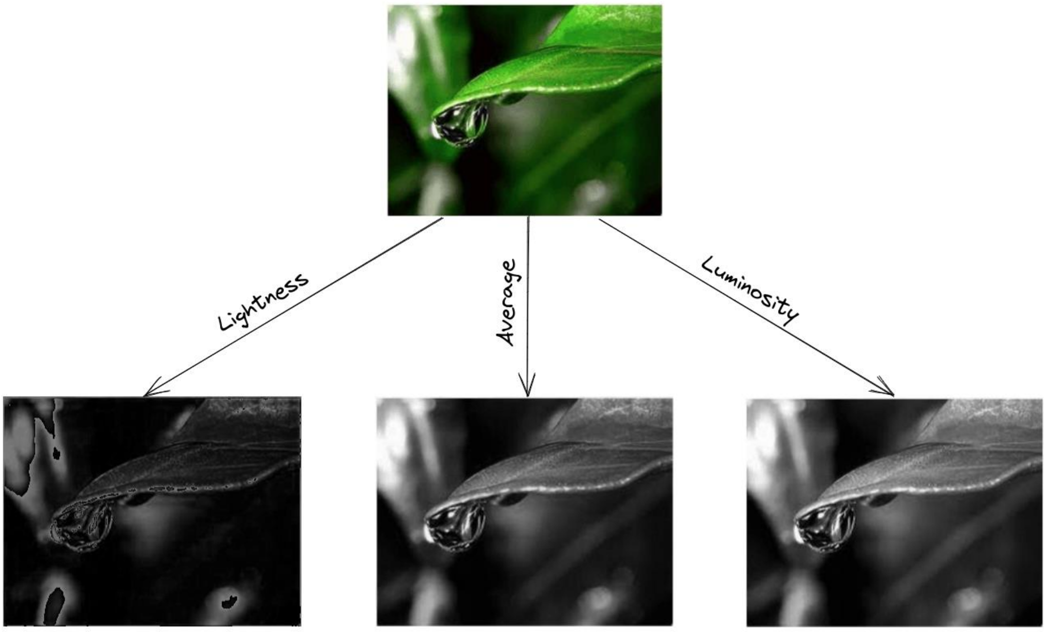
2.14.1. Colour to grayscale

Grayscale slika je ona čiji pikseli imaju samo vrijednosti intenziteta svjetla gdje je 0 crna i 255 bijela. Najčešće metode za konvertiranje slike u boji u crno bijele slike su: [22]

- **Lightness method** – uzima se srednja vrijednost najveće i najmanje vrijednosti. Mana je što se jedna komponenta izostavlja (od tri komponente - RGB)
- **Metoda srednjih vrijednosti** – uzima se srednja vrijednost od sve tri komponente (RGB). Problem ove metode je što uzima jednaku težinu od sve tri komponente, što nije slučaj kod ljudskog oka. Osjetljivost ljudskog oka je najveća na zelenu, zatim na crvenu i na kraju na plavu.

- **Metoda osvjetljenja** (*eng. Luminosity Method*) – ova metoda uspješno rješava problem prethodne dvije metoda na način da stavlja težine na vrijednosti u skladu s osjetljivosti ljudskog oka.

$$\text{grayscale} = 0.3 \times R + 0.59 \times G + 0.11 \times B$$

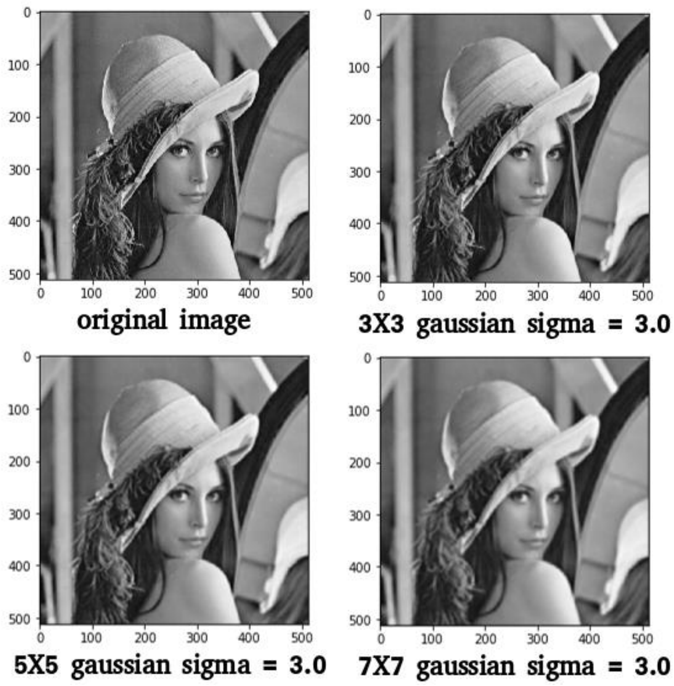


Slika 2.8. Primjer tri metode grayscale filtera [22]

Grayscale filter se koristi jer zahtjeva manje memorije i brže se obrađuje, a to pogotovo dolazi do izražaja kod velikih podataka i kompleksnih izračuna uz gubitak informacija koje imaju slike u boji.

2.14.2. Gaussov filter (eng. Gaussian blur)

Filter za obradu digitalnih slika gdje se dobiva efekt zamućenja (*blurring*) je Gaussov filter u svrhu uklanjanja visokih frekvencija u slici. Pikseli koji grade tipičnu digitalnu sliku imaju tri vrijednosti, odnosno iznose RGB intenziteta, podijeljene u tri zapisa. Naravno, slike u sivim tonovima imaju jednu vrijednost po pikselu, odnosno intenzitet svjetline. Osnovni princip rada Gaussovog filtera je isti bez obzira na sliku. Svaki piksel slike koji želimo zamutiti se gleda neovisno, a njegova vrijednost se mijenja ovisno o vlastitoj vrijednosti i onima u okolini na temelju matrice filtera koja se naziva kernel. Kernel je pravokutni niz brojeva koji prate Gaussovu ili normalnu distribuciju. Veličina kernela (npr. 3x3 ili 5x5) odgovara broju piksela koje uzimamo u obzir prilikom zamućenja pojedinačnog piksela. Okolne vrijednosti piksela se koriste za izračun ponderiranog prosjeka za novu vrijednost originalnog piksela na temelju Gaussove distribucije u kernelu. U slučaju rubova, nepostojeći neposredni okolni pikseli dobivaju istu vrijednost kao njihov najbliži susjed ili vrijednost koja odgovara zrcalno suprotnom pikselu u uzorkovanom području. [23]



Slika 2.9. Primjer različitih Gaussovih filtera [24]

2.14.3. Canny edge detektor

Canny edge detector je najvjerojatnije najšire korišteni detektor ruba u računalnom vidu. Sastoji se od „pokvarenih“ rubova s nadodanom Gaussovom bukom. Canny je pokazao da prva derivacija Gaussian filtera približno aproksimira operator (koji je relativno kompleksan – opisan je sumom četiriju eksponencijalnih uvjeta). Algoritam se može opisati pomoću 5 koraka:

1. Primjena Gaussian filtera (zaglađivanje slike u svrhu smanjenja buke).
2. Pronalazak gradijenta intenziteta kako bi se izdvojili potencijalni rubove (edge detektor operator vraća prvu derivaciju funkcije intenziteta u x i y smjerovima).
3. Primjena *non-maximum suppression* metode, u svrhu smanjenja broja mogućih rješenja (dokida tanke rubove; filter prolazi kroz sliku i ako vrijednosti prelaze prag, rubovi ostaju).
4. Primjena metode *double threshold* kako bi se detektirali potencijalni rubovi (miče “sumnjive” piksele koji su nastali zbog utjecaja buke i boje – high value (stvaranje ruba) and low value (povezivanje) vrijednosti).

5. Funkcijom histereze uklanjaju se zaostali slabi rubovi (ako su slabi rubovi povezani s jakim ostaju, u suprotnom se i oni uklanjaju – 8_way connectivity pristup). [12]

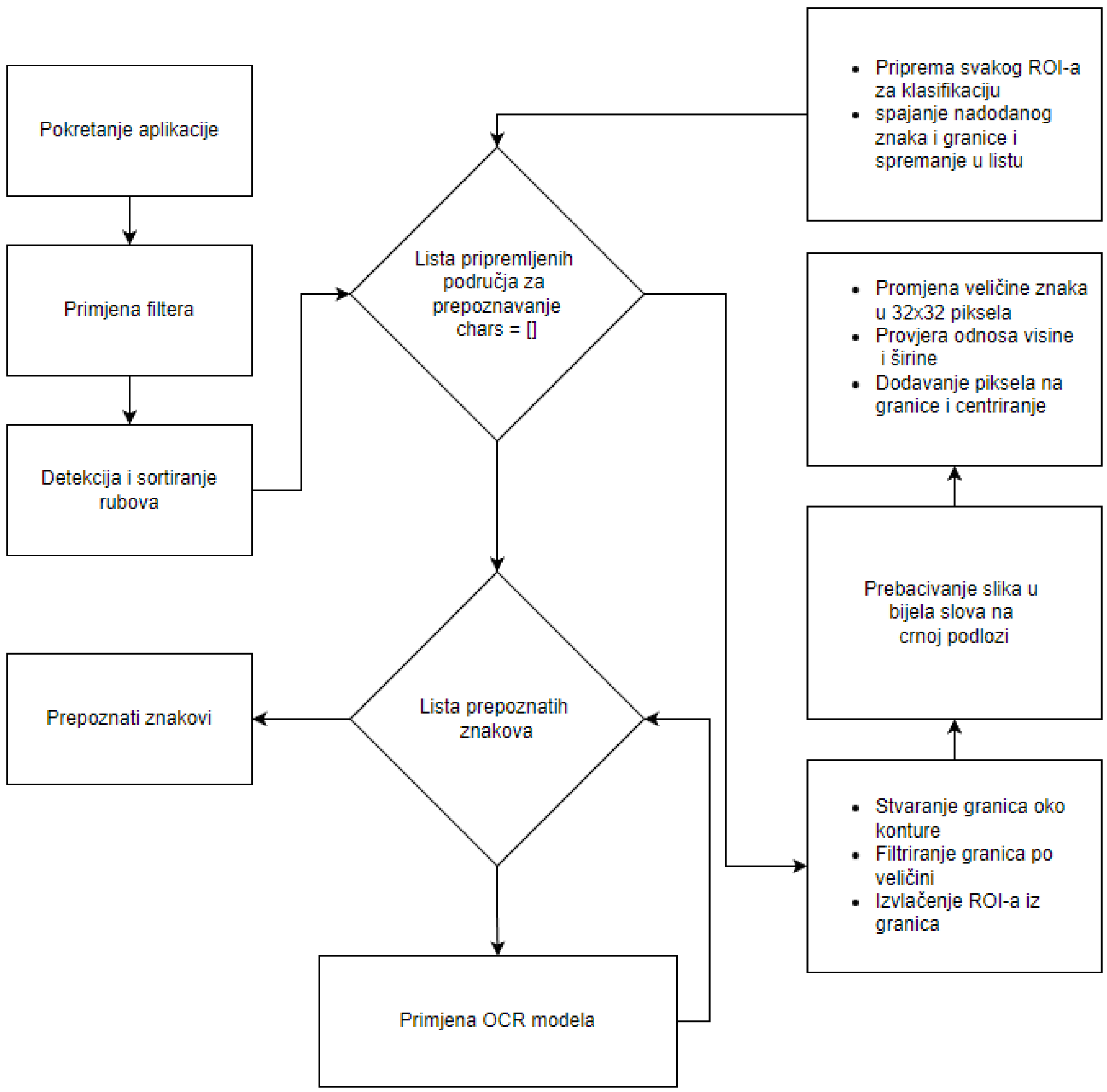
Slika 2.10. Primjer Canny Edge detekcije [24]

3. TRENIRANJE MODELA I IZRADA APLIKACIJE

Zadatak je napraviti aplikaciju koja koristi OCR treniran na konvolucijskim neuronskim mrežama kako bi prepoznavao rukom pisani tekst. Za to je potrebno trenirati model na ResNet neuronskoj mreži koristeći MNIST i Keggle A-Z baze podataka za brojeve i slova. Nakon toga se preko jednostavnog GUI-a napravljenog u Pythonu radi eksperimentalna evaluacija prepoznavanja rukom pisanoga teksta.

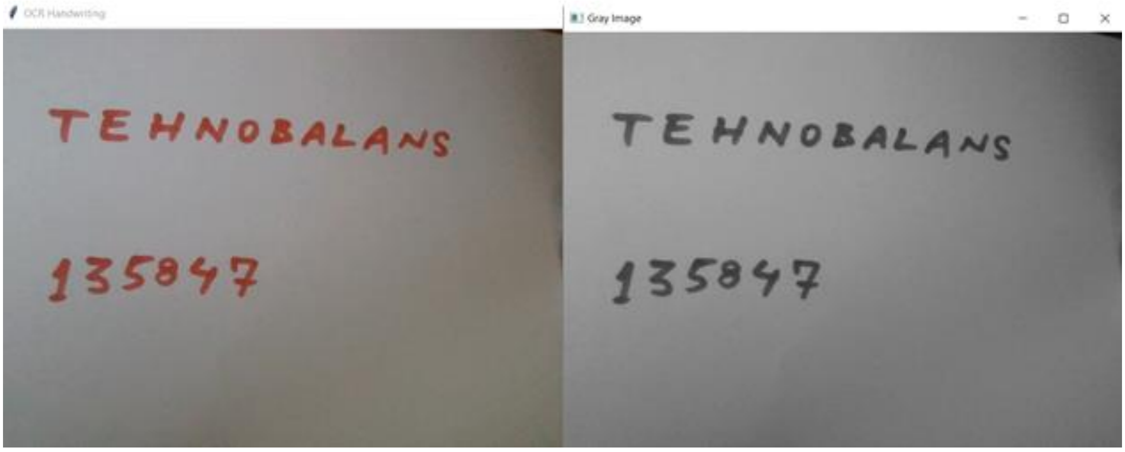
3.1. Model

Na slici 3.1. je prikazan dijagram toka kako radi OCR model za prepoznavanje rukom pisanoga teksta.

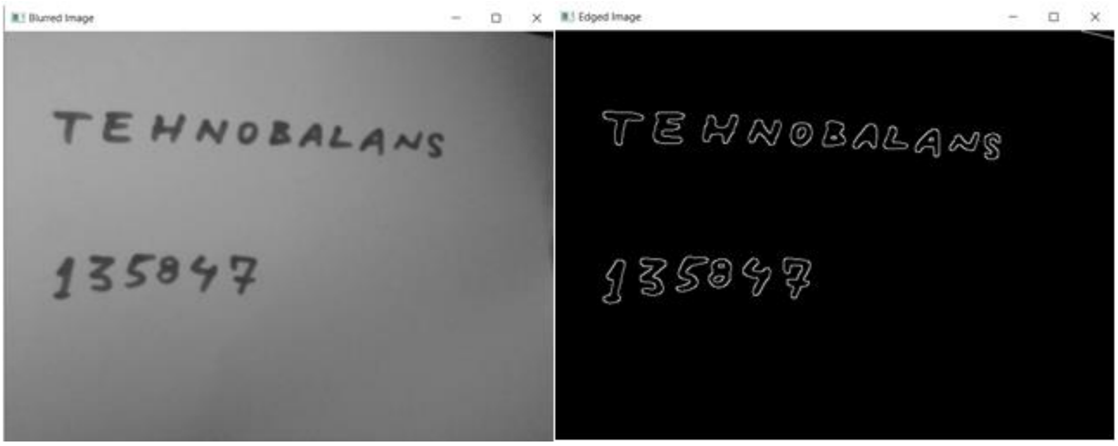


Slika 3.1. Dijagram toka rada modela za OCR rukopisa

Na slici 3.2. je prikazan originalan tekst pisan rukom i primjena prvog grayscale filtera na sliku kao uvod u digitalnu obradu. Slika 3.3. prikazuje daljnju primjenu gaussian filtera za smanjenje pozadinske buke i primjenjuje se detektora rubova. Nakon obrade filterima, vrši se detekcija i sortiranje rubova, te se podaci spremaju u listu za prepoznavanje *char = []*. U toj listi se vrti petlja za prepoznate rubove. Stvaraju se granice oko kontura, filtriraju se po veličini (prevelike i premale se odbacuju), te se izvlači ROI (*Region Of Interest* – interesantna regija) . Nakon toga se prebacuju slike u bijela slova na crnoj podlozi (Slika 3.4.) i provjerava se odnos visine i širine. Idući korak je dodavanje piksela na granice (eng. *padding*) što osigurava da je znak centriran i da je standardne veličine 32x32 piksela (Slika 3.4.). Zadnji korak petlje je priprema svakog ROI-a za klasifikaciju spajanjem nadodanih karakteristika i granica, te spremanje u listu. Ta pripremljena lista znakova ide u listu prepoznatih znakova koja se vrti u petlji sa istreniranim modelom koji vraća vjerojatnosti predikcije za pojedini simbol i na kraju se izbacuje u prepoznate znakove. Na slici 3.5. je prikaz gotovih rezultata predikcije.

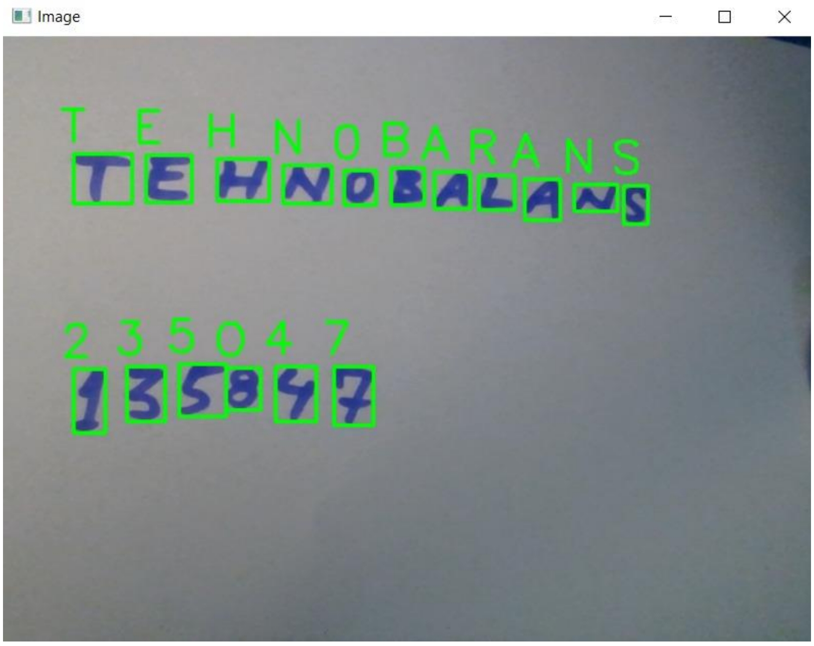


Slika 3.2. Uzimanje screenshota sa kamere i primjena grayscale filtera



Slika 3.3. Primjena gaussovog filtera i detektora rubova

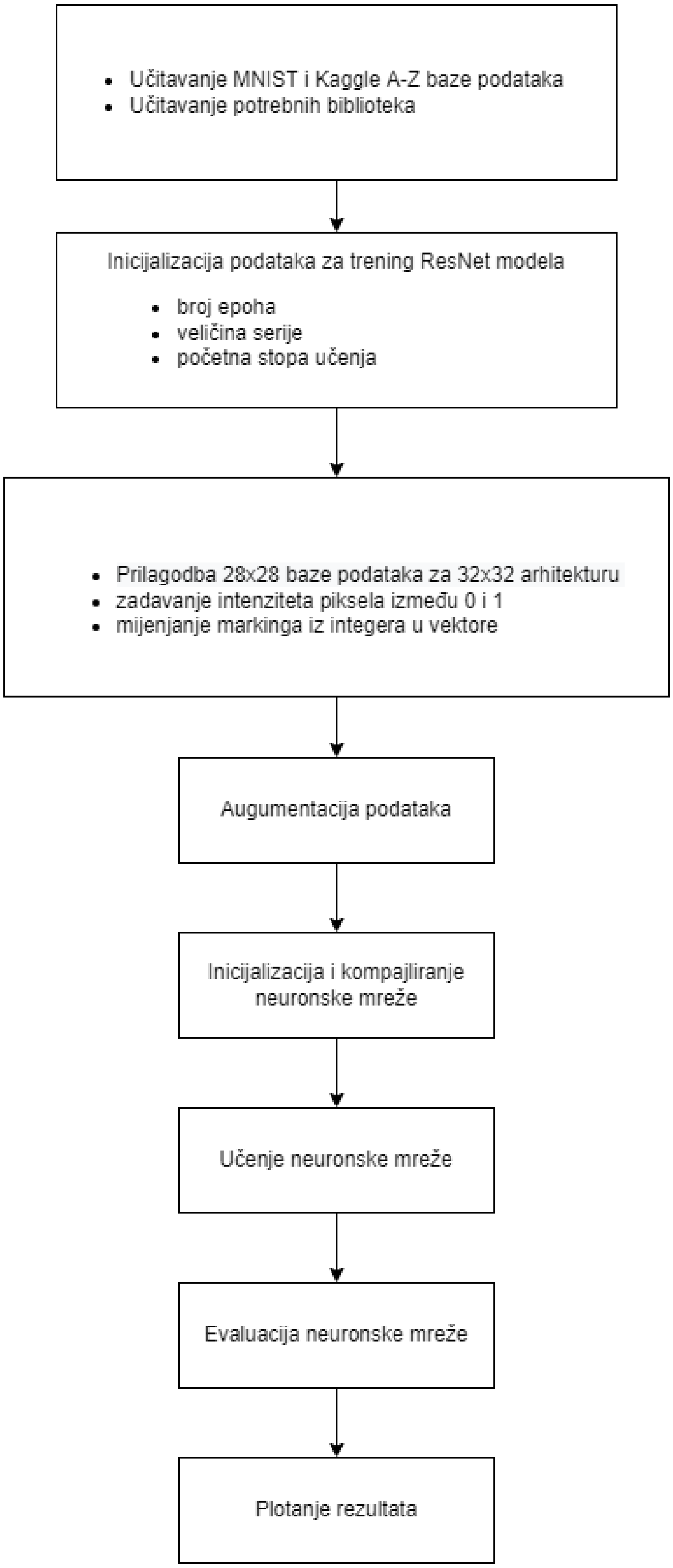
Slika 3.4. Nekoliko znakova u pripremi za OCR



Slika 3.5. Prepoznavanje znakova korištenjem neuronske mreže

3.2. Treniranje modela

Na slici 3.6. je prikaz dijagrama treninga OCR modela koristeći ResNet duboku neuronsku mrežu, koja u svojim bibliotekama primarno koristi *Keras* i *TensorFlow*. Trening započinje učitavanjem MNIST i Kaggle A-Z baza podataka i učitavanjem potrebnih python biblioteka. Nakon toga se inicijaliziraju podaci za ResNet, odnosno zadaje se broj epoha, veličina serije i početna stopa učenja. Idući korak je prilagodna podataka iz bazi podataka za učenje koji su 28x28 piksela na 32x32 piksela zbog arhitekture neuronske mreže koja zahtjeva te podatke. Vrši se augumentacija podataka, odnosno stvaranje više podataka za učenje korištenjem rotacije, translacije, zumiranja itd. Nakon toga dolazi do inicijalizacije i kompajliranja duboke neuronske mreže. U ovome slučaju koristi se *categorical crossentropy loss* funkcija koja je dobra za zadatke višeklasne klasifikacije. Ta funkcija kaže da primjer može pripadati samo jednoj od više kategorija i da model mora odlučiti koja. Na primjer ako imamo 0-9 znamenke, *categorical crossentropy* dati će visoku vjerojatnost samo za jedan broj, a za sve ostale nisku. [26] Treniranje neuronske mreže se vrši po epohama (Slika 3.7.), te nakon toga dolazi evaluacija koja govori o uspješnosti treninga (Slika 3.8.) i plotanje rezultata (Slika 3.9.). Na slici 3.7. gdje su prikazani treninzi nekoliko zadnjih epoha, može se primjetiti da točnost jako sporo raste, odnosno da je u 7 epoha točnost narasla za samo 0.1 %.



Slika 3.6. Dijagram toka treninga OCR duboke neuronske mreže

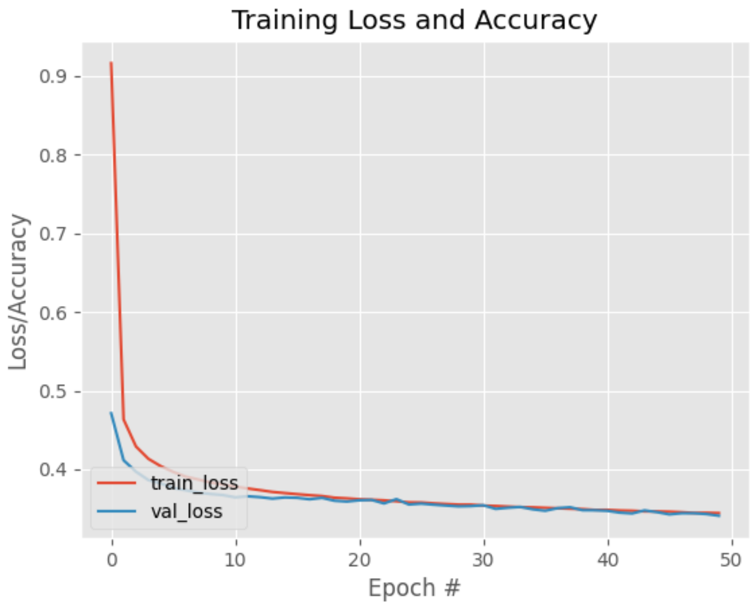
```
Epoch 44/50
2765/2765 [==============================] - 6824s 2s/step - loss: 0.5941 - accuracy: 0.9647 - val_loss: 0.3692 - val_accuracy: 0.9680
Epoch 45/50
2765/2765 [==============================] - 6872s 2s/step - loss: 0.5908 - accuracy: 0.9647 - val_loss: 0.3696 - val_accuracy: 0.9677
Epoch 46/50
2765/2765 [==============================] - 6792s 2s/step - loss: 0.5885 - accuracy: 0.9652 - val_loss: 0.3766 - val_accuracy: 0.9653
Epoch 47/50
2765/2765 [==============================] - 6811s 2s/step - loss: 0.5876 - accuracy: 0.9650 - val_loss: 0.3727 - val_accuracy: 0.9660
Epoch 48/50
2765/2765 [==============================] - 6815s 2s/step - loss: 0.5862 - accuracy: 0.9657 - val_loss: 0.3745 - val_accuracy: 0.9657
Epoch 49/50
2765/2765 [==============================] - 6869s 2s/step - loss: 0.5830 - accuracy: 0.9656 - val_loss: 0.3709 - val_accuracy: 0.9670
Epoch 50/50
2765/2765 [==============================] - 6804s 2s/step - loss: 0.5825 - accuracy: 0.9657 - val_loss: 0.3660 - val_accuracy: 0.9686
[INFO] evaluating network...
```

Slika 3.7. Trening CNN po epohama

Prema rezultatima treninga točnost neuronske mreže je u prosjeku 96%. Najslabiji rezultat ima nula (0) sa točnošću 62%, dok slova A, E, P i T imaju čak 100% točnost.

	precision	recall	f1-score	support
0	0.62	0.43	0.50	1381
1	0.98	0.99	0.99	1575
2	0.93	0.95	0.94	1398
3	0.99	0.99	0.99	1428
4	0.93	0.97	0.95	1365
5	0.81	0.92	0.86	1263
6	0.97	0.97	0.97	1375
7	0.98	0.99	0.98	1459
8	0.97	0.99	0.98	1365
9	0.99	0.98	0.98	1392
A	1.00	0.99	0.99	2774
B	0.98	0.99	0.99	1734
C	0.99	0.99	0.99	4682
D	0.91	0.98	0.94	2027
E	1.00	0.99	0.99	2288
F	0.98	0.99	0.99	232
G	0.96	0.96	0.96	1152
H	0.98	0.98	0.98	1444
I	0.98	0.99	0.98	224
J	0.98	0.98	0.98	1699
K	0.98	0.99	0.98	1121
L	0.98	0.99	0.98	2317
M	0.99	1.00	0.99	2467
N	0.99	0.99	0.99	3802
O	0.93	0.95	0.94	11565
P	1.00	0.99	0.99	3868
Q	0.95	0.99	0.97	1162
R	0.99	0.99	0.99	2313
S	0.99	0.97	0.98	9684
T	1.00	0.99	0.99	4499
U	0.99	0.99	0.99	5802
V	0.98	0.99	0.99	836
W	0.98	0.99	0.99	2157
X	0.99	0.99	0.99	1254
Y	0.99	0.95	0.97	2172
Z	0.95	0.95	0.95	1215
accuracy			0.97	88491
macro avg	0.96	0.96	0.96	88491
weighted avg	0.97	0.97	0.97	88491

Slika 3.8. Prikaz rezultata nakon završetka treninga CNN-a



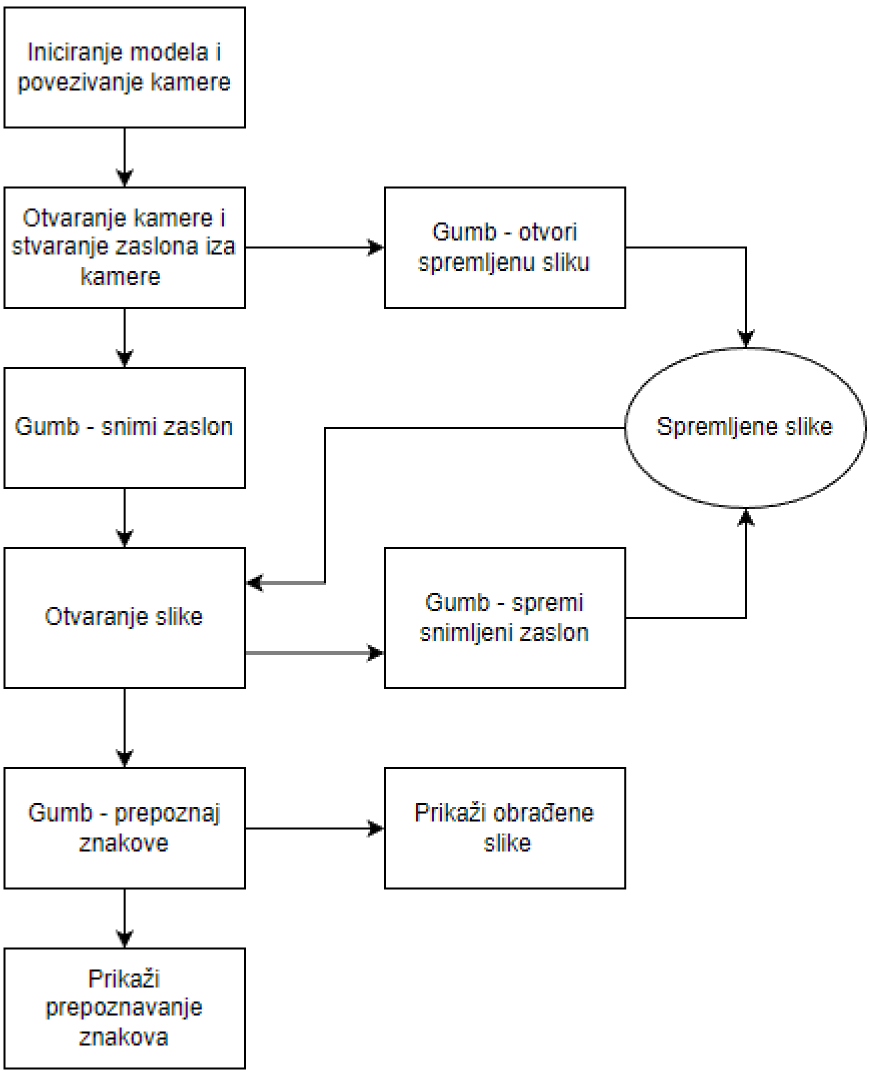
Slika 3.9. Graf točnosti po epohama

Na grafu (Slika 3.9.) možemo vidjeti da je train_loss mali, te da je malo znakova pretreniranosti modela što ukazuje na ispravan rad OCR modela.

Izrada modela i treniranje neuronske mreže su temeljeni na pyimagesearch člancima i kodu u njima. [27][28]

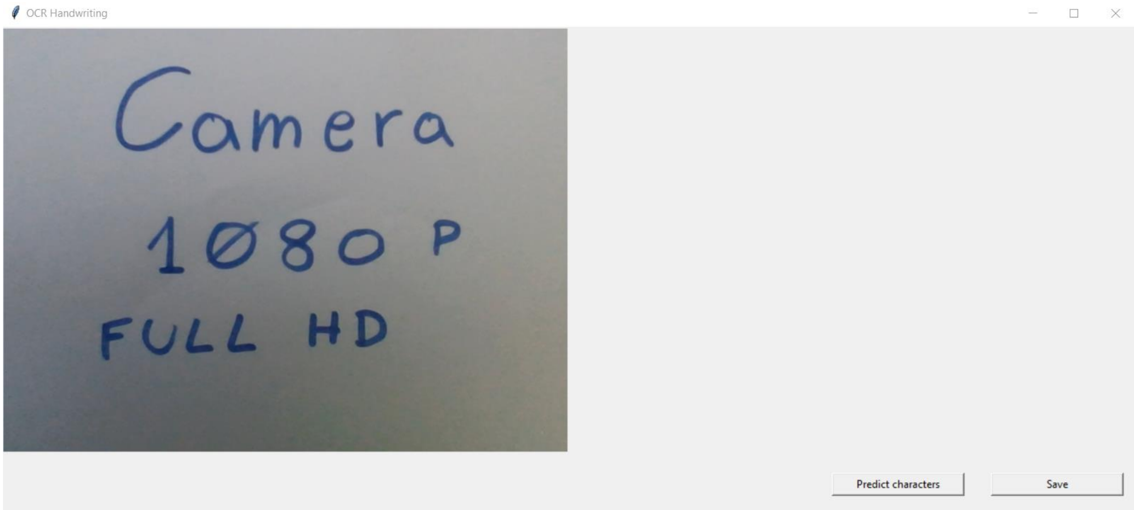
3.3. Oblikovanje korisničkog sučelja

Korištenjem *Tkinter* biblioteke u Pythonu izrađeno je jednostavno korisničko sučelje sa nekoliko opcija čiji se dijagram može vidjeti na slici 3.10. Najprije se pokreću potrebne biblioteke, te se inicira model i povezuje se kamera. Nakon otvaranja kamere se otvara i zaslon u veličini prozora kamere (Slika 3.11.). Postoje dva gumba sa opcijama *Open Image* – otvaranje postojeće slike na računalu i *Snapshot* koji uzima trenutni frame kamere. Nakon otvaranja slike pojavljuje se opcija da se spremi slika na računalo i da se prepoznaje rukom pisani tekst na slici (Slika 3.12.). Nakon opcije da se prepozna tekst, otvaraju se slike koje su obrađene filterima i detektorom ruba, te se prikazuju rezultati prepoznavanja teksta (Slika 3.13.).

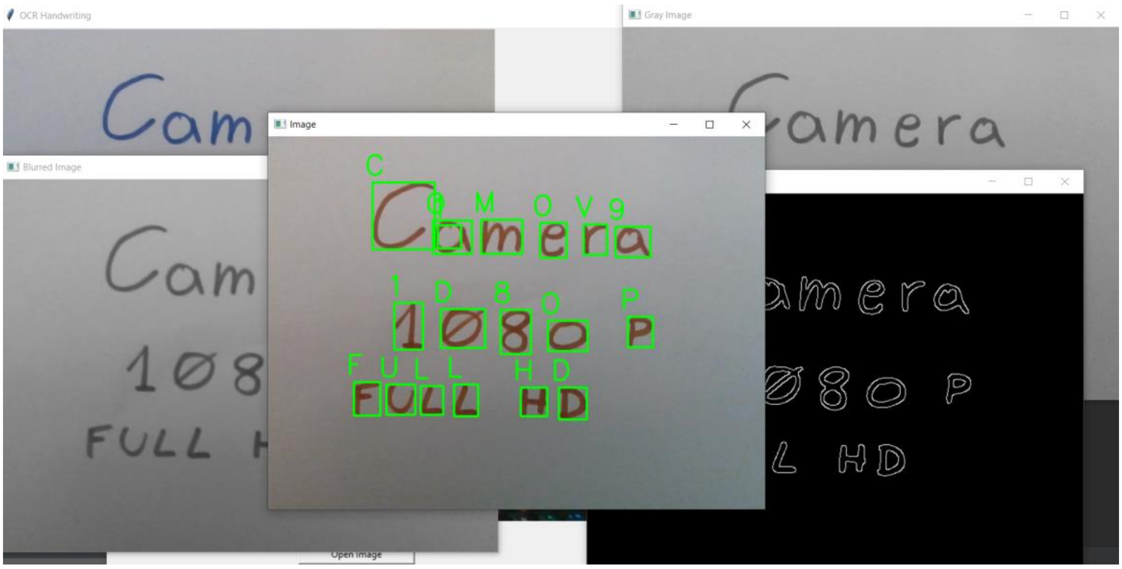


Slika 3.10. Dijagram rada grafičkog korisničkog sučelja

Slika 3.11. Prikaz inicijalnog korisničkog sučelja



Slika 3.12. Sučelje prilikom odabira predikcije ili spremanja



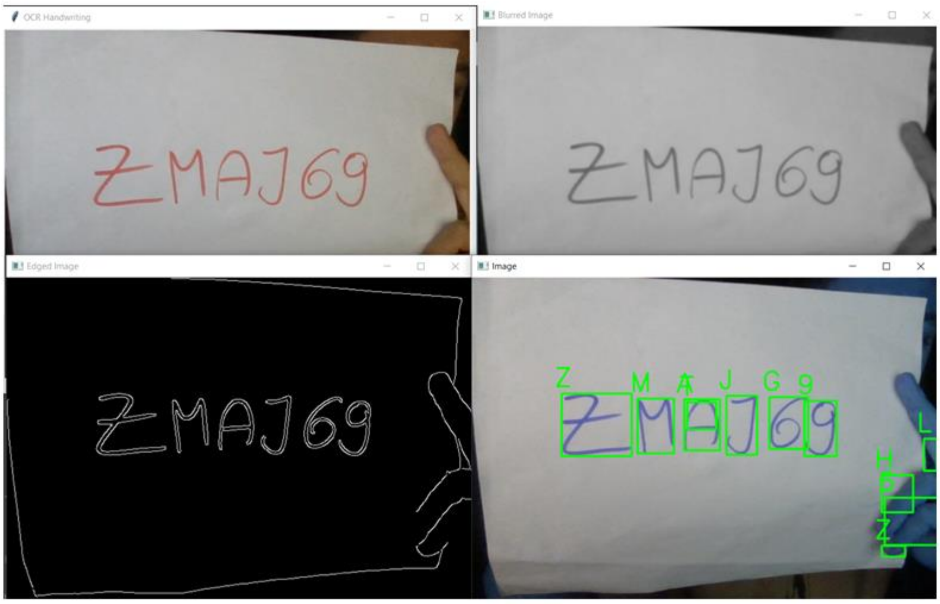
Slika 3.13. GUI sa prikazom rezultata i obrada slika

4. EKSPERIMENTALNA EVALUACIJA

Evaluacija rada aplikacije je napravljena snimanjem niza rukom pisanih tekstova koji su zatim unešeni u aplikaciju korištenjem kamere. Rezultati su prikazani (u smjeru kazaljke na satu, od gornje lijeve slike) kao: originalna slika, slika sa grayscale i gaussian filterom, prepoznati znakovi i rubovi slike. Razmatrano je više situacija poput crvenog i crnog markera na papiru, osvjetljenja R/G/B spektrom, neosvijetljenih znakova, pisanih slova, sličnih znakova i više rukopisa na istome papiru.

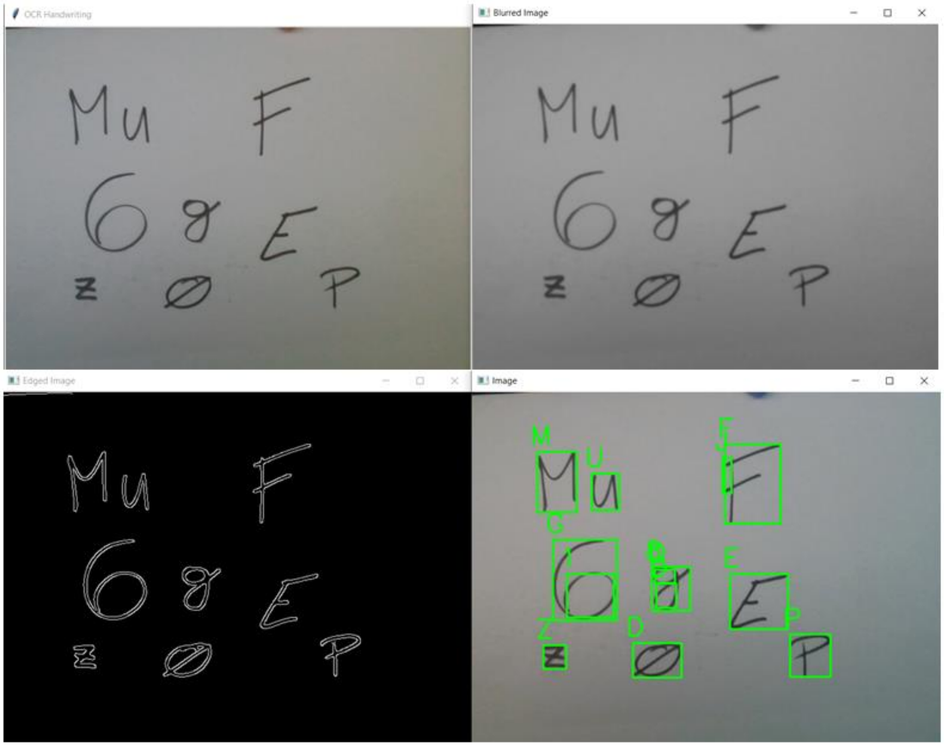
4.1. Rezultati evaluacije

Rezultati evaluacija su jako raznoliki i zanimljivi. Na slici 4.1. je većina znakova dobro prepoznata, ali su detektirani i rubovi prstiju što je kasnije prepoznato kao neki znak. Pojavio se i problem prepoznavanja slova u slovu.



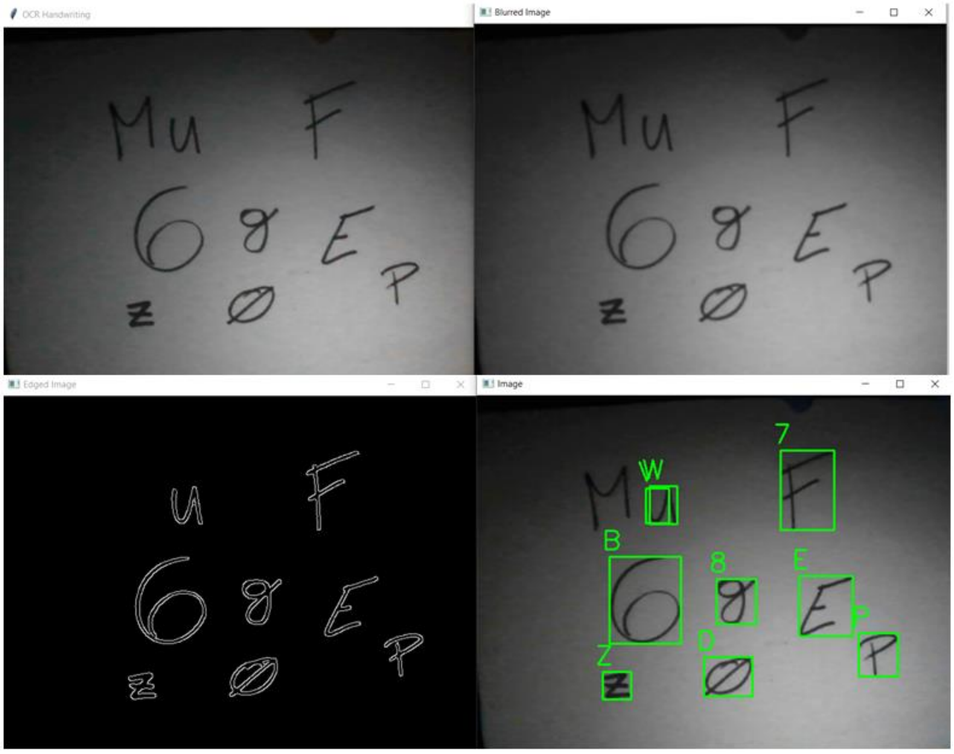
Slika 4.1. Test – crveni marker

Na slici 4.2. je prepoznata većina znakova. Neki znakovi su krivo prepoznati i ima više slučajeva prepoznavanja slova u slovu.



Slika 4.2. Test – referentna slika

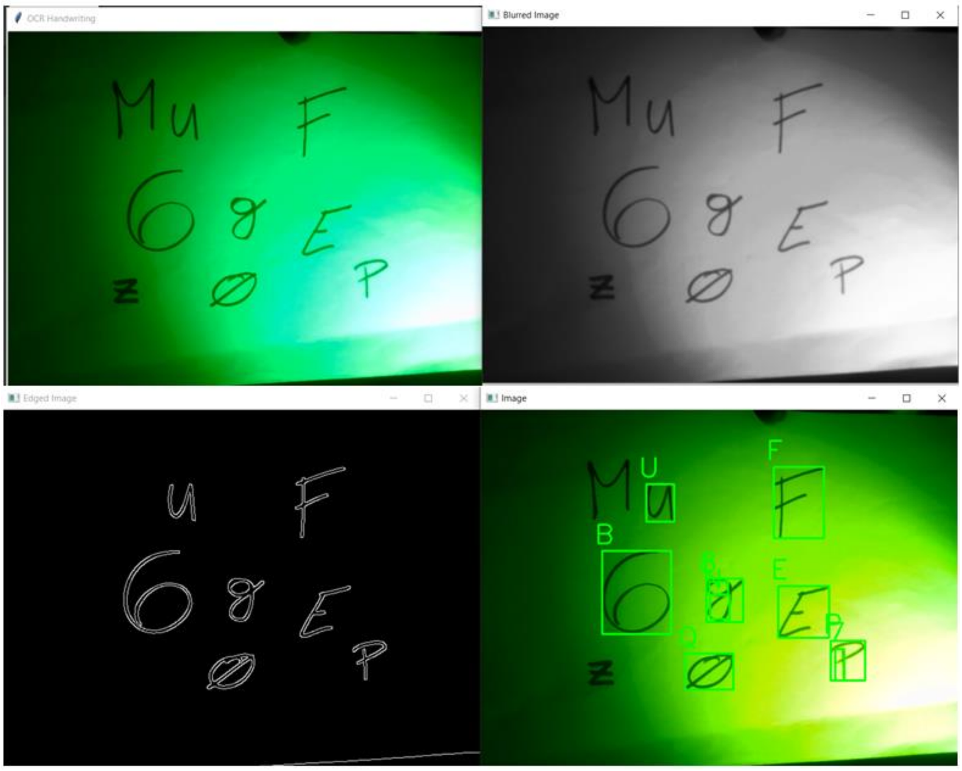
Slika 4.3. gdje je bilo samo pozadinsko svjetlo ekrana jedan simbol uopce nije prepoznat, a više njih je krivo prepoznato. Moguće objašnjenje je loša osvjetljenost koju još dodatno naglasi gaussian filter, pa čak do te granice da canny edge uopće ne prepozna rub.



Slika 4.3. Test – ugašeno svjetlo

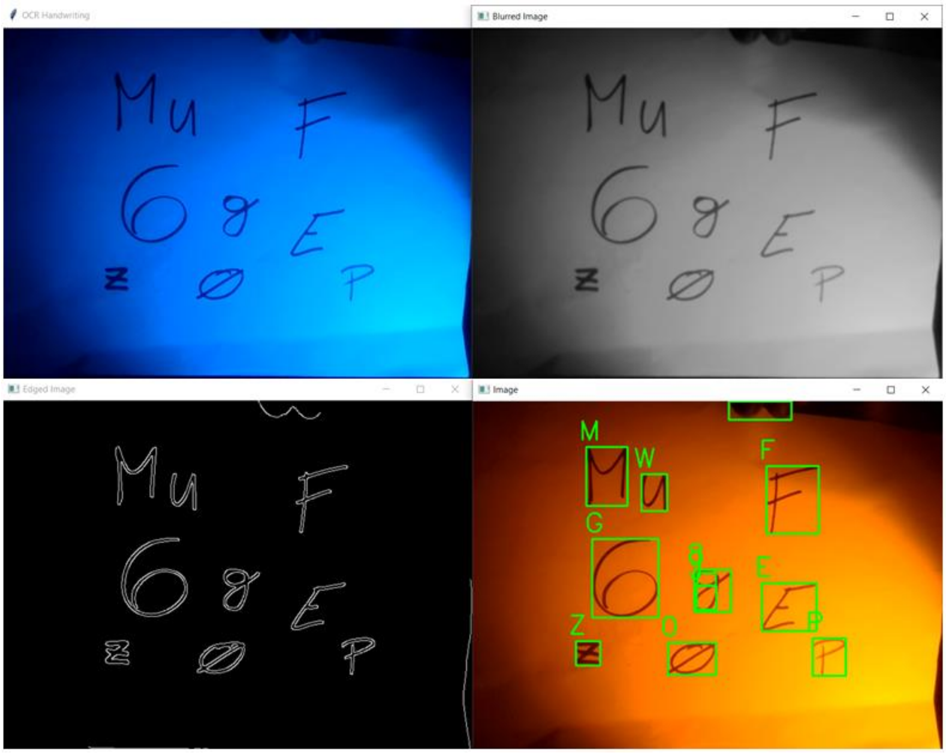
Testovi sa crvenim i zelenim svjetlom na slikama 4.4. i 4.5. imaju slične probleme kao i slika 4.3.

Slika 4.4. Test – crveno svjetlo



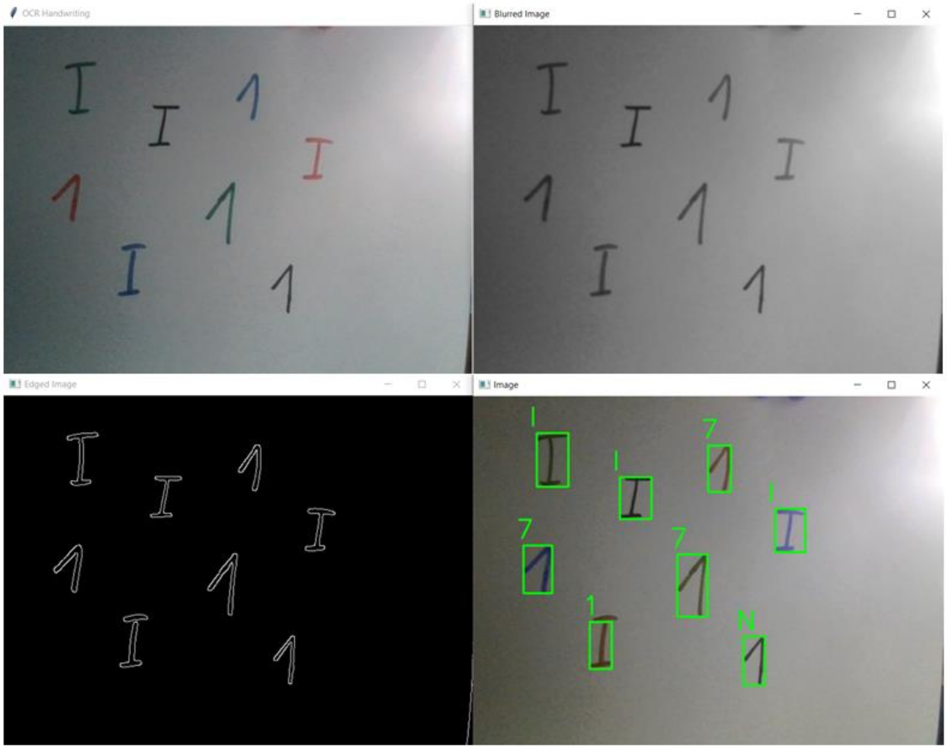
Slika 4.5. Test – zeleno svjetlo

Na slici 4.6. su bolji rezultati nego na prethodne tri slike. Svi znakovi su prepoznati uključujući i prste koji drže papir. Moguće objašnjenje rezultata je jednoličnije osvjetljenje, te se onda ne gube informacije prilikom obrade slike.



Slika 4.6. Test – plavo svjetlo

Slika 4.7. je test sa sličnim znakovima, odnosno broj jedan i veliko slovo I. Svi znakovi su prepoznati, ali su samo tri slova I točno prepoznata.



Slika 4.7. Test – slični znakovi u različitim bojama

Tekst napisan pisanim slovima na slici 4.8. očekivano nije dobro prepoznat. Vjerojatno objašnjenje je neprekinuta kontura, odnosno nema takvih slova u dataset-u na kojemu je neuronska mreža učila.

Slika 4.8. Test – pisana slova

Slika 4.9. ima više tipova rukopisa. Gotovo svi elementi su prepoznati, ali svaki rukopis je dao različite rezultate. Crna slova imaju preklapanja, te su nakon obrade slika znakovi stopljeni i neuronska mreža ima problema sa prepoznavanjem. Slična situacija je i sa crvenim slovima. Plavi znakovi su dobro prepoznati, ali prepoznata su i slova u slovima. Veličina slova je vjerojatno utjecala na to. Zelena slova su najbolje prepoznata, što potencijalno znači da je veličina slova bila povoljnija za model prepoznavanja.

Slika 4.9. Test – različiti rukopisi

5. KRITIČKI OSVRT

U evaluacijskom dijelu rada zapaženo je više stvari koje bi se mogle izmijeniti i na taj način poboljšati rad aplikacije. Neka od zapažanja su skup podataka za učenje, osvjetljenje i predobrada slika.

Kako je temelj najvažniji dio građevine, tako je i skup podataka za trening temelj dobrog modela neuronske mreže. Čak niti savršene neuronske mreže nisu u stanju dati dobre predikcije ako ulazni set podataka nije dobar. U ovome slučaju mogu se dodati dijakritički znakovi kako bi se proširilo prepoznavanje i na širi tekst. Skup podataka za učenje je vjerojatno širokog fonta, odnono pisan markerom što za posljedicu daje gotovo nemoguće prepoznavanje teksta pisanog tankom olovkom ili kemijskom.

U slučaju slabog osvjetljenja, filteri mogu sjenu dodatno proširiti te canny edge više ne može detektirati rub i u potpunosti se izgubi prepoznavanje znaka. Zanimljivo je da se znakovi u predobradi u potpunosti izgube iako čovjek to može sa lakoćom razaznati. To je jedan od indikatora da postoji široko područje za poboljšavanje filtera i algoritama.

Predobrada slika je jako važna jer sliku iscjepka i prilagodi na elemente koje model neuronske mreže može prepoznavati. Konkretno u slučaju velikih slova se događaju dvostruke detekcije na istome znaku. Taj problem bi se mogao algoritamski riješiti tako da se odbacuje dio prepoznatih znakova u preklapanju. Još jedan od problema je detekcija znakova na rubovima od ljudi ili prstiju i slično. Moguće rješenje je korištenje segmentacije ili dodatne neuronske mreže za bolju pripremu slike za obradu i prepoznavanje, odnosno za odbacivanje nepotrebnih dijelova slike.

6. ZAKLJUČAK

Ovaj završni rad je pokazao kako prepoznavanje rukom pisanog teksta nije nimalo jednostavan zadatak. Tijek razvoja aplikacije nije jednostavan. Potrebna je široka teorijska podloga iz raznih segmenata računalne znanosti, od vizijskih sustava do konvolucijskih neuronskih mreža, kako bi se problem mogao kvalitetno riješiti. Korištenjem ResNet neuronske mreže trenirane na MNIST i Keggle A-Z bazama podataka stvoren je model od 96% teoretske točnosti. Korisničko sučelje koje se povezuje sa kamerom omogućuje elegantno i fleksibilno korištenje programa. Evaluacijski dio je pokazao da u realnim uvjetima model ima problema sa radom. Utjecaj osvjetljenja, veličine slova, dodirivanje znakova i prepoznavanje rubova od objekata poput papira ili prstiju su veliki izazov u kvalitetnom radu modela.

Moguća poboljšanja su korištenje drugih baza podataka koje su ažurnije ili prikladnije za ResNet neuronsku mrežu. Ubrzani razvoj procesorske snage računala omogućuje primjenu složenih i zahtjevnih algoritama koji bi neke od navedenih problema mogli riješiti u programu, pa čak i u realnome vremenu.

Zaključak je da, iako OCR za rukopis nije nova ideja, postoje područja u kojima bi se moglo jako napredovati i približiti teoretskoj točnosti od 96% .

LITERATURA

[1] umjetna inteligencija. Hrvatska enciklopedija, mrežno izdanje. Leksikografski zavod Miroslav Krleža, http://www.enciklopedija.hr/Natuknica.aspx?ID=63150 2021. Pristupljeno: 17. 09. 2022.

[2] TechTarget: „Machine Learning“. https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML Pristupljeno: 18.09.2022.

[3] IBM: „Neural Networks“. https://www.ibm.com/cloud/learn/neural-networks Pristupljeno: 18.09.2022.

[4] Novaković, Majetić, Široki, „Umjetne neuronske mreže“, FSB, Zagreb 2011.

[5] N. Bolf, OSVJEŽIMO ZNANJE, Kem. Ind. 68 (5-6) (2019) 219–220

[6] IBM: „Deep Learning“. https://www.ibm.com/cloud/learn/deep-learning Pristupljeno: 19.09.2022.

[7] Usporedba strojnog učenja i dubokih neurosnkih mreža; https://thenewstack.io/demystifying-deep-learning-and-artificial-intelligence/ Pristupljeno: 20.9.2022.

[8] IBM: „Convolutional Neural Networks“. https://www.ibm.com/cloud/learn/convolutional-neural-networks Pristupljeno: 20.9.2022.

[9] Primjer CNN za klasifikaciju rukom pisanog teksta https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 Pristupljeno: 20.9.2022.

[10] Analytics Vidhya: „Build ResNet from Scratch With Python !„. https://www.analyticsvidhya.com/blog/2021/06/build-resnet-from-scratch-with-python/ Pristupljeno: 20.09.2022.

[11] IBM: „Computer Vision“. https://www.ibm.com/topics/computer-vision Pristupljeno: 20.09.2022.

[12] T. Stipančić: Podloge za predavanje iz kolegija „Vizijski sustavi“, Fakultet strojarstva i brodogradnje, Zagreb, 2022.

[13] Papers With Code: „MNIST“. https://paperswithcode.com/dataset/mnist Pristupljeno: 20.09.2022.

[14] primjer MNIST baze simbola. https://en.wikipedia.org/wiki/MNIST_database Pristupljeno: 20.9.2022.

[15] Kaggle: „A-Z Handwritten Alphabets in .csv format“. https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format Pristupljeno: 20.09.2022.

[16] Nanonets: „Data augumentation“. https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/ Pristupljeno: 21.09.2022.

[17] Wikipedia: „Python“. https://hr.wikipedia.org/wiki/Python_(programski_jezik) Pristupljeno: 20.09.2022.

[18] Github: „tensorflow“. https://github.com/tensorflow/tensorflow Pristupljeno: 16.09.2022.

[19] Keras. https://keras.io/about/ Pristupljeno: 16.09.2022.

[20] OpenCV. https://opencv.org/about/ Pristupljeno: 16.09.2022.

[21] Welcome to Colaboratory. https://colab.research.google.com/ Pristupljeno: 20.09.2022.

[22] Baeldung: „How to Convert an RGB Image to a Grayscale“. https://www.baeldung.com/cs/convert-rgb-to-grayscale Pristupljeno: 21.09.2022.

[23] Hackaday: „WHAT EXACTLY IS A GAUSSIAN BLUR?“. https://hackaday.com/2021/07/21/what-exactly-is-a-gaussian-blur/ Pristupljeno: 21.09.2022.

[24] Medium: „A Beginners Guide to Computer Vision (Part 1)-Filtering“. https://medium.com/@t.bharathchandra/a-beginners-guide-to-computer-vision-part-1-filtering-3d95a1d51fb1 Pristupljeno: 22.09.2022.

[25] logs alexander decurnou: „knest: part one“. https://log.alx.xyz/posts/knest_pt1/ Pristupljeno: 22.09.2022.

[26] Knowledge Center: „Categorical crossentropy“. https://peltarion.com/knowledge-center/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy Pristupljeno: 22.09.2022.

[27] PyImageSearch: „OCR with Keras, TensorFlow, and Deep Learning“. https://pyimagesearch.com/2020/08/17/ocr-with-keras-tensorflow-and-deep-learning/ Pristupljeno: 22.09.2022.

[28] PyImageSearch: „OCR: Handwriting recognition with OpenCV, Keras, and TensorFlow“. https://pyimagesearch.com/2020/08/24/ocr-handwriting-recognition-with-opencv-keras-and-tensorflow/?fbclid=IwAR2o-9WBX-_imvewu21mFpPAyrXqPPjZ3ojWWIBRJQqcMzq_KZYcxcE8fvQ Pristupljeno: 22.09.2022.

PRILOG

Helpers az_dataset

```
[ ] # import the necessary packages
    from tensorflow.keras.datasets import mnist
    import numpy as np
```

```
[ ] def load_az_dataset(datasetPath):
      # initialize the list of data and labels
      data = []
      labels = []

      # loop over the rows of the A-Z handwritten digit dataset
      for row in open(datasetPath):
        # parse the label and image from the row
        row = row.split(",")
        label = int(row[0])
        image = np.array([int(x) for x in row[1:]], dtype="uint8")

        # images are represented as single channel (grayscale) images
        # that are 28x28=784 pixels -- we need to take this flattened
        # 784-d list of numbers and repshape them into a 28x28 matrix
        image = image.reshape((28, 28))

        # update the list of data and labels
        data.append(image)
        labels.append(label)

      # convert the data and labels to NumPy arrays
      data = np.array(data, dtype="float32")
      labels = np.array(labels, dtype="int")

      # return a 2-tuple of the A-Z data and labels
      return (data, labels)
```

```
def load_mnist_dataset():
  # load the MNIST dataset and stack the training data and testing
  # data together (we'll create our own training and testing splits
  # later in the project)
  ((trainData, trainLabels), (testData, testLabels)) = mnist.load_data()
  data = np.vstack([trainData, testData])
  labels = np.hstack([trainLabels, testLabels])

  # return a 2-tuple of the MNIST data and labels
  return (data, labels)
```

Helpers ResNET

```
[ ] # import the necessary packages
    from tensorflow.keras.layers import BatchNormalization
    from tensorflow.keras.layers import Conv2D
    from tensorflow.keras.layers import AveragePooling2D
    from tensorflow.keras.layers import MaxPooling2D
    from tensorflow.keras.layers import ZeroPadding2D
    from tensorflow.keras.layers import Activation
    from tensorflow.keras.layers import Dense
    from tensorflow.keras.layers import Flatten
    from tensorflow.keras.layers import Input
    from tensorflow.keras.models import Model
    from tensorflow.keras.layers import add
    from tensorflow.keras.regularizers import l2
    from tensorflow.keras import backend as K

    class ResNet:
      @staticmethod
      def residual_module(data, K, stride, chanDim, red=False,
        reg=0.0001, bnEps=2e-5, bnMom=0.9):
        # the shortcut branch of the ResNet module should be
        # initialize as the input (identity) data
        shortcut = data

        # the first block of the ResNet module are the 1x1 CONVs
        bn1 = BatchNormalization(axis=chanDim, epsilon=bnEps,
          momentum=bnMom)(data)
        act1 = Activation("relu")(bn1)
        conv1 = Conv2D(int(K * 0.25), (1, 1), use_bias=False,
          kernel_regularizer=l2(reg))(act1)
```

```
        # the second block of the ResNet module are the 3x3 CONVs
        bn2 = BatchNormalization(axis=chanDim, epsilon=bnEps,
          momentum=bnMom)(conv1)
        act2 = Activation("relu")(bn2)
        conv2 = Conv2D(int(K * 0.25), (3, 3), strides=stride,
          padding="same", use_bias=False,
          kernel_regularizer=l2(reg))(act2)

        # the third block of the ResNet module is another set of 1x1
        # CONVs
        bn3 = BatchNormalization(axis=chanDim, epsilon=bnEps,
          momentum=bnMom)(conv2)
        act3 = Activation("relu")(bn3)
        conv3 = Conv2D(K, (1, 1), use_bias=False,
          kernel_regularizer=l2(reg))(act3)
```

```
        # if we are to reduce the spatial size, apply a CONV layer to
        # the shortcut
        if red:
          shortcut = Conv2D(K, (1, 1), strides=stride,
            use_bias=False, kernel_regularizer=l2(reg))(act1)

        # add together the shortcut and the final CONV
        x = add([conv3, shortcut])

        # return the addition as the output of the ResNet module
        return x

    @staticmethod
    def build(width, height, depth, classes, stages, filters,
        reg=0.0001, bnEps=2e-5, bnMom=0.9, dataset="cifar"):
        # initialize the input shape to be "channels last" and the
        # channels dimension itself
        inputShape = (height, width, depth)
        chanDim = -1

        # if we are using "channels first", update the input shape
        # and channels dimension
        if K.image_data_format() == "channels_first":
          inputShape = (depth, height, width)
          chanDim = 1

        # set the input and then apply a BN followed by CONV
        inputs = Input(shape=inputShape)
        x = BatchNormalization(axis=chanDim, epsilon=bnEps,
          momentum=bnMom)(inputs)
        x = Conv2D(filters[0], (3, 3), use_bias=False,
          padding="same", kernel_regularizer=l2(reg))(x)
```

```
        # loop over the number of stages
        for i in range(0, len(stages)):
          # initialize the stride, then apply a residual module
          # used to reduce the spatial size of the input volume
          stride = (1, 1) if i == 0 else (2, 2)
          x = ResNet.residual_module(x, filters[i + 1], stride,
            chanDim, red=True, bnEps=bnEps, bnMom=bnMom)

          # loop over the number of layers in the stage
          for j in range(0, stages[i] - 1):
            # apply a ResNet module
            x = ResNet.residual_module(x, filters[i + 1],
              (1, 1), chanDim, bnEps=bnEps, bnMom=bnMom)
```

```
# apply BN => ACT => POOL
x = BatchNormalization(axis=chanDim, epsilon=bnEps,
  momentum=bnMom)(x)
x = Activation("relu")(x)
x = AveragePooling2D((8, 8))(x)

# softmax classifier
x = Flatten()(x)
x = Dense(classes, kernel_regularizer=l2(reg))(x)
x = Activation("softmax")(x)

# create the model
model = Model(inputs, x, name="resnet")

# return the constructed network architecture
return model
```

Train model

```
[ ] # USAGE
    # python train_ocr_model.py --az a_z_handwritten_data.csv --model handwriting.model

    # connect to google drive
    from google.colab import drive

    drive.mount("/content/gdrive/", force_remount=True)

    az_file_path = "/content/gdrive/MyDrive/Colab Notebooks/Matija_Zavrsni/a_z_handwritten_data.csv"
    model_path = "/content/gdrive/MyDrive/Colab Notebooks/Matija_Zavrsni/handwriting.model"
    plot_image_path = "/content/gdrive/MyDrive/Colab Notebooks/Matija_Zavrsni/plot.png"

    # set the matplotlib backend so figures can be saved in the background
    import matplotlib
    matplotlib.use("Agg")

    # import the necessary packages
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.optimizers import SGD
    from sklearn.preprocessing import LabelBinarizer
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import classification_report
    from imutils import build_montages
    import matplotlib.pyplot as plt
    import numpy as np
    import argparse
    import cv2
```

```
# construct the argument parser and parse the arguments
"""
ap = argparse.ArgumentParser()
ap.add_argument("-a", "--az", required=True,
  help="path to A-Z dataset")
ap.add_argument("-m", "--model", type=str, required=True,
  help="path to output trained handwriting recognition model")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
  help="path to output training history file")
args = vars(ap.parse_args())
"""

# initialize the number of epochs to train for, initial learning rate,
# and batch size
EPOCHS = 50
INIT_LR = 1e-1
BS = 128

# load the A-Z and MNIST datasets, respectively
print("[INFO] loading datasets...")
(azData, azLabels) = load_az_dataset(az_file_path)
(digitsData, digitsLabels) = load_mnist_dataset()

# the MNIST dataset occupies the labels 0-9, so let's add 10 to every
# A-Z label to ensure the A-Z characters are not incorrectly labeled
# as digits
azLabels += 10
```

```
# stack the A-Z data and labels with the MNIST digits data and labels
data = np.vstack([azData, digitsData])
labels = np.hstack([azLabels, digitsLabels])

# each image in the A-Z and MNIST digts datasets are 28x28 pixels;
# however, the architecture we're using is designed for 32x32 images,
# so we need to resize them to 32x32
data = [cv2.resize(image, (32, 32)) for image in data]
data = np.array(data, dtype="float32")

# add a channel dimension to every image in the dataset and scale the
# pixel intensities of the images from [0, 255] down to [0, 1]
data = np.expand_dims(data, axis=-1)
data /= 255.0

# convert the labels from integers to vectors
le = LabelBinarizer()
labels = le.fit_transform(labels)
counts = labels.sum(axis=0)
```

```
# account for skew in the labeled data
classTotals = labels.sum(axis=0)
classWeight = {}

# loop over all classes and calculate the class weight
for i in range(0, len(classTotals)):
  classWeight[i] = classTotals.max() / classTotals[i]

# partition the data into training and testing splits using 80% of
# the data for training and the remaining 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data,
  labels, test_size=0.20, stratify=labels, random_state=42)

# construct the image generator for data augmentation
aug = ImageDataGenerator(
  rotation_range=10,
  zoom_range=0.05,
  width_shift_range=0.1,
  height_shift_range=0.1,
  shear_range=0.15,
  horizontal_flip=False,
#Ignoring horizontal characters
  fill_mode="nearest")

# initialize and compile our deep neural network
print("[INFO] compiling model...")
opt = SGD(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model = ResNet.build(32, 32, 1, len(le.classes_), (3, 3, 3),
  (64, 64, 128, 256), reg=0.0005)
model.compile(loss="categorical_crossentropy", optimizer=opt,
  metrics=["accuracy"])
```

```
# train the network
print("[INFO] training network...")
H = model.fit(
  aug.flow(trainX, trainY, batch_size=BS),
  validation_data=(testX, testY),
  steps_per_epoch=len(trainX) // BS,
  epochs=EPOCHS,
  class_weight=classWeight,
  verbose=1)

# define the list of label names
labelNames = "0123456789"
labelNames += "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
labelNames = [l for l in labelNames]
```

```
# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=BS)
print(classification_report(testY.argmax(axis=1),
  predictions.argmax(axis=1), target_names=labelNames))

# save the model to disk
print("[INFO] serializing network...")
model.save(model_path, save_format="h5")

# construct a plot that plots and saves the training history
N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure()
plt.plot(N, H.history["loss"], label="train_loss")
plt.plot(N, H.history["val_loss"], label="val_loss")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(plot_image_path)

# initialize our list of output images
images = []

# randomly select a few testing characters
for i in np.random.choice(np.arange(0, len(testY)), size=(49,)):
  # classify the character
  probs = model.predict(testX[np.newaxis, i])
  prediction = probs.argmax(axis=1)
  label = labelNames[prediction[0]]

  # extract the image from the test data and initialize the text
  # label color as green (correct)
  image = (testX[i] * 255).astype("uint8")
  color = (0, 255, 0)
```

```
  # otherwise, the class label prediction is incorrect
  if prediction[0] != np.argmax(testY[i]):
    color = (0, 0, 255)

  # merge the channels into one image, resize the image from 32x32
  # to 96x96 so we can better see it and then draw the predicted
  # label on the image
  image = cv2.merge([image] * 3)
  image = cv2.resize(image, (96, 96), interpolation=cv2.INTER_LINEAR)
  cv2.putText(image, label, (5, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
    color, 2)
```

```
  # add the image to our list of output images
  images.append(image)

# construct the montage for the images
montage = build_montages(images, (96, 96), (7, 7))[0]

# show the output montage
cv2.imshow("OCR Results", montage)
cv2.waitKey(0)
```

GUI

```
import tkinter
import cv2
import PIL.Image, PIL.ImageTk
import time
from tensorflow.keras.models import load_model
import imutils
from imutils.contours import sort_contours
import numpy as np

class App:
    def __init__(self, window, window_title, video_source=0):
        self.model = load_model('handwriting.model')
        self.window = window
        self.window.title(window_title)
        self.video_source = video_source

        # open video source (by default this will try to open the computer webcam)
        self.vid = MyVideoCapture(self.video_source)

        # Create a canvas that can fit the above video source size
        self.canvas = tkinter.Canvas(window, width = self.vid.width, height = self.vid.height)
        self.canvas.pack()

        # Button that lets the user take a snapshot
        self.btn_snapshot = tkinter.Button(window, text="Snapshot", width=20, command=self.snapshot)
        self.btn_snapshot.pack(anchor=tkinter.CENTER, expand=True)

        # Button that lets the user open a stored image
        self.btn_open = tkinter.Button(window, text="Open Image", width=20, command=self.snapshot)
        self.btn_open.pack(anchor=tkinter.CENTER, expand=True)

        # After it is called once, the update method will be automatically called every delay milliseconds
        self.delay = 15
        self.update()

        self.window.mainloop()

    def _predict(self, image):
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        cv2.imshow("Gray Image", gray)
        blurred = cv2.GaussianBlur(gray, (5, 5), 0)
        cv2.imshow("Blurred Image", blurred)
        edged = cv2.Canny(blurred, 30, 150)
        cv2.imshow("Edged Image", edged)
        cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
                                cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        cnts = sort_contours(cnts, method="left-to-right")[0]

        chars = []


        # loop over the contours
        for c in cnts:
            # compute the bounding box of the contour
            (x, y, w, h) = cv2.boundingRect(c)

            # filter out bounding boxes, ensuring they are neither too small
            # nor too large
            if (w >= 5 and w <= 150) and (h >= 15 and h <= 120):
                # extract the character and threshold it to make the character
                # appear as *white* (foreground) on a *black* background, then
                # grab the width and height of the thresholded image
                roi = gray[y:y + h, x:x + w]
                cv2.imshow("Roi",roi)
                thresh = cv2.threshold(roi, 0, 255,
                                       cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
                (tH, tW) = thresh.shape

                # if the width is greater than the height, resize along the
                # width dimension
                if tW > tH:
                    thresh = imutils.resize(thresh, width=32)

                # otherwise, resize along the height
                else:
                    thresh = imutils.resize(thresh, height=32)
```

```
            # re-grab the image dimensions (now that its been resized)
            # and then determine how much we need to pad the width and
            # height such that our image will be 32x32
            (tH, tW) = thresh.shape
            dX = int(max(0, 32 - tW) / 2.0)
            dY = int(max(0, 32 - tH) / 2.0)

            # pad the image and force 32x32 dimensions
            padded = cv2.copyMakeBorder(thresh, top=dY, bottom=dY,
                                        left=dX, right=dX, borderType=cv2.BORDER_CONSTANT,
                                        value=(0, 0, 0))
            padded = cv2.resize(padded, (32, 32))

            cv2.imshow("Padded", padded)
            cv2.waitKey(0)

            # prepare the padded image for classification via our
            # handwriting OCR model
            padded = padded.astype("float32") / 255.0
            padded = np.expand_dims(padded, axis=-1)

            # update our list of characters that will be OCR'd
            chars.append((padded, (x, y, w, h)))

    # extract the bounding box locations and padded characters
    boxes = [b[1] for b in chars]
    chars = np.array([c[0] for c in chars], dtype="float32")

    preds = self.model.predict(chars)

    # define the list of label names
    labelNames = "0123456789"
    labelNames += "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    labelNames = [l for l in labelNames]

    # loop over the predictions and bounding box locations together
    for (pred, (x, y, w, h)) in zip(preds, boxes):
        # find the index of the label with the largest corresponding
        # probability, then extract the probability and label
        i = np.argmax(pred)
        prob = pred[i]
        label = labelNames[i]

        # draw the prediction on the image
        print("[INFO] {} - {:.2f}%".format(label, prob * 100))
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(image, label, (x - 10, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 255, 0), 2)
    cv2.imshow("Image", image)


def snapshot(self):
    # Get a frame from the video source
    ret, frame = self.vid.get_frame()

    if ret:
        top = tkinter.Toplevel(self.window)
        local_canvas = tkinter.Canvas(top, height=self.vid.height, width=self.vid.width * 2)
        local_canvas.pack()
        self.photo_l = PIL.ImageTk.PhotoImage(image=PIL.Image.fromarray(frame))
        image_on_canvas = local_canvas.create_image(0, 0, image=self.photo_l, anchor=tkinter.NW)

        def _save():
            cv2.imwrite("frame-" + time.strftime("%d-%m-%Y-%H-%M-%S") + ".jpg",
                        cv2.cvtColor(frame, cv2.COLOR_RGB2BGR))

        btn_save = tkinter.Button(top, text="Save", width=20,
                                  command=lambda: _save())
        btn_save.pack(side=tkinter.RIGHT, padx=15, pady=20)

        btn_predict = tkinter.Button(top, text="Predict characters", width=20,
                                     command=lambda: self._predict(frame))
        btn_predict.pack(side=tkinter.RIGHT, padx=15, pady=20)
```

```
    def update(self):
        # Get a frame from the video source
        ret, frame = self.vid.get_frame()

        if ret:
            self.photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(frame))
            self.canvas.create_image(0, 0, image = self.photo, anchor = tkinter.NW)

        self.window.after(self.delay, self.update)


class MyVideoCapture:
    def __init__(self, video_source=0):
        # Open the video source
        self.vid = cv2.VideoCapture(video_source)
        if not self.vid.isOpened():
            raise ValueError("Unable to open video source", video_source)

        # Get video source width and height
        self.width = self.vid.get(cv2.CAP_PROP_FRAME_WIDTH)
        self.height = self.vid.get(cv2.CAP_PROP_FRAME_HEIGHT)

    def get_frame(self):
        if self.vid.isOpened():
            ret, frame = self.vid.read()
            if ret:
                # Return a boolean success flag and the current frame converted to BGR
                return (ret, cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
            else:
                return (ret, None)
        else:
            return (ret, None)

    # Release the video source when the object is destroyed
    def __del__(self):
        if self.vid.isOpened():
            self.vid.release()

# Create a window and pass it to the Application object
App(tkinter.Tk(), "OCR Handwriting")
```