Prepoznavanje šahovske ploče i figura temeljeno na metodama dubokog učenja

Filek, Karlo

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Karlo Filek

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

Izv. prof. dr. sc. Tomislav Stipančić, dipl. ing. Karlo Filek

Zagreb, 2023.

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

Zahvaljujem se izv. prof. dr. sc. Tomislavu Stipančiću na pomoći i strpljenju tokom pisanja rada. Također zahvaljujem se obitelji i bliskim prijateljima na podršci tijekom studija.

Karlo Filek

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 Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 23 – 6 / 1**	
Ur.broj: **15 - 1703 - 23 -**	

ZAVRŠNI ZADATAK

Student: **Karlo Filek** JMBAG: **0035209710**

Naslov rada na hrvatskom jeziku: **Prepoznavanje šahovske ploče i figura temeljeno na metodama dubokog učenja**

Naslov rada na engleskom jeziku: **Chessboard and figure recognition based on deep learning methods**

Opis zadatka:

Računalni modeli temeljeni na algoritmima umjetne inteligencije omogućavaju automatsko prepoznavanje objekata koristeći vizijske senzore smještene u stvarnu okolinu. Prepoznavanje je moguće temeljiti na različitim značajkama objekata te na različitim metodama iz područja strojnog vida.

U okviru završnog rada je potrebno:

- proučiti metode za detekciju šahovskih figura u stvarnom vremenu,
- upoznati se s OpenCV programskom bibliotekom otvorenog koda te koristiti prikladne metode u svrhu prepoznavanja šahovskih figura,
- analizirati učinkovitost razvijenog modela u ovisnosti o kutu gledanja na šahovsku ploču.

Razvijeno programsko rješenje je potrebno temeljiti na OpenCV biblioteci implementiranoj kroz Python programski jezik. Model je potrebno eksperimentalno evaluirati koristeći vizijski sustav u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava. U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:

30. 11. 2022.

Zadatak zadao:

Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:

1. rok: 20. 2. 2023.
2. rok (izvanredni): 10. 7. 2023.
3. rok: 18. 9. 2023.

Predviđeni datumi obrane:

1. rok: 27. 2. – 3. 3. 2023.
2. rok (izvanredni): 14. 7. 2023.
3. rok: 25. 9. – 29. 9. 2023.

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

SAŽETAK

Zadnjih 2 desetljeća metode računalnog vida i strojnog učenja su se drastično razvile i modernom arhitekturom računala omogućili *real-time* izvedbu zadanih algoritama prepoznavanja i detekcije objekata. Tema ovog rada je izrada računalnog programa za detekciju i prepoznavanje šahovske ploče i figura te usporedba u učinkovitosti modela s obzirom o kutu gledanja na šahovsku ploču. Rješenje je bazirano na implementaciji duboke neuronske mreže doučene na naš set podataka te je izrađeno u programskom jeziku *Python* uz korištenje prograskih biblioteka otvorenog koda *OpenCV* za područje obrade slika i računalnog vida i *TensorFlow* za područje učenje neuronske mreže i obradu podataka. Učenje mreža je ubrzano primjenom *GoogleColab* kolaborativnog okruženja gdje se omogućava pristub *Google*-ovim radnim jedinicama.

Ključne riječi: *Python, OpenCV, TensorFlow*, duboke neuronske mreže, prepoznavanje objekata, VGG16

SUMMARY

For the past 2 decades the methods for computer vision and object recognition have been rapidly developing. With the evolution of modern computer processing units real-time object detection and recognition has been made a possibility. The goal of this thesis is to create a computer program for chess board detection and figure recognition based on deep neural network recognition. Operating program is built in *Python* 3.10 and uses open source libraries such as *OpenCV* for image manipulation and *TensorFlow* for neural network fitting and image evaluation. The training of the neural network was sped up by implementing Googles server resources through *GoogleColab* platform.

Key words: *Python*, *OpenCV*, *TensorFlow*, deep neural network, object recognition, VGG16

1. UVOD

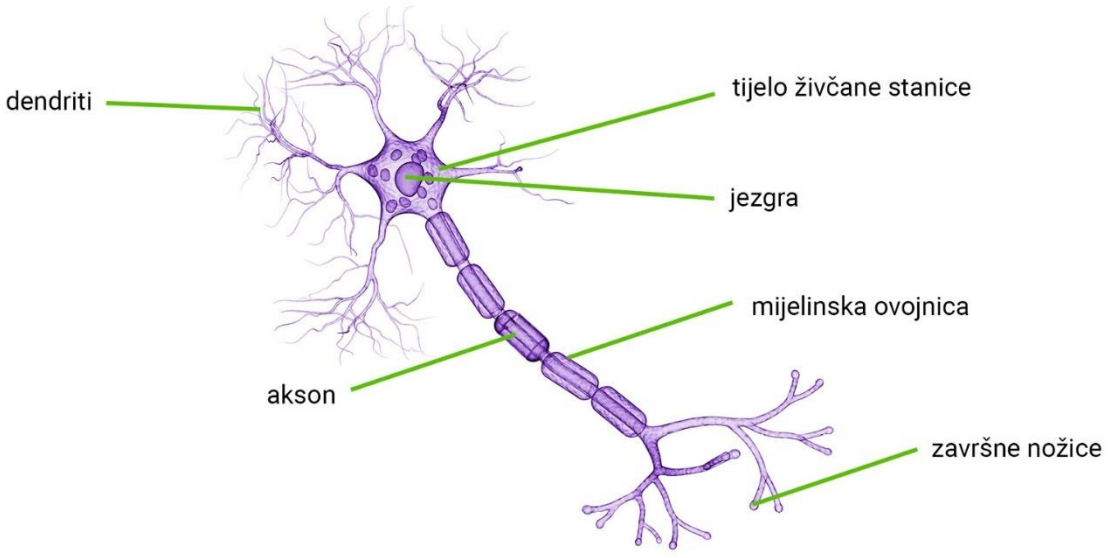
Detekcije i prepoznavanje predmeta u realnom svijetu zasniva se na 2 problema. Prvi problem je učenje algoritma umjetne inteligencije na prepoznavanje oblika, a drugi je pravilno segmentiranje slike i detekcija objekta da bi algoritam prepoznavanja mogao obaviti uspješni postupak prepoznavanja. Problem prepoznavanja se rješava putem strojnog učenja, odnosno jedne od metoda umjetne inteligencije. Neke od metoda učenja umjetne inteligencije u području računalnog vida su nadgledano učenje, nenadgledano učenje, duboke neuronske mreže i konvolucijske neuronske mreže [1]. U ovom radu problem detekcije objekta riješen je implementacijom jednostavnih algoritama dijeljenja slike unutar *OpenCV* biblioteke otvorenog koda.

Za ovaj rad odabrana je metoda prepoznavanja sa dubokom konvolucijskom neuronskom mrežom *VGG16* [7] u biblioteci otvorenog koda *TensorFlow* radi predučenosti mreže na detekciju predmeta putem *Imagenet* baze podataka i radi lakšeg doučavanja na odabrani set podataka šahovskih figura. Za programski jezik odabran je Python radi prethodnog poznavanja rada u programskom jeziku.

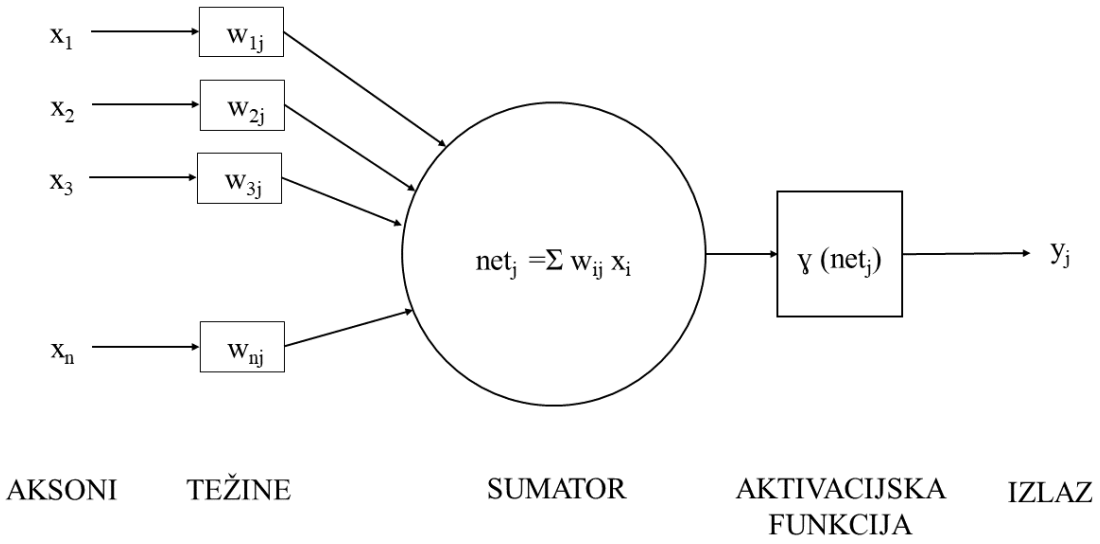
2. NEURONSKE MREŽE

Umjetne neuronske mreže razvile su se kao grana strojnog učenja računalnih znanosti. U pristupu umjetnih neuronskih mreža specifično je što se pokušava emulirati biološka struktura neurona te njihove interpovezanosti u veću funkcionalnu strukturu. Prvo analiziranje rada mozga i neurona je zabilježio William James kraje 19. stoljeća [2] razmišljanjem o strukturi i načinu ispaljivanja informacija neurona u mozgu. Njegova razmišljanja su 1943. McCulloch i Pitts [16] pretvorili u matematički model neurona koji se primjenjuje i danas.

Struktura umjetnog neurona je usporediva s onom biološkog neurona u vidu primitka, obrade i proslijeđivanja podataka [2]. Biološki neuron sastoji od dendrita, aksona, tijela neurona i sinapsi. Uloga dendrita je da prima informacije od drugih stanica, uloga tijela neurona da odluči da li proslijediti informaciju dalje, uloga aksona prenosi informaciju do aksonskih sinapsi gdje proslijeđuje tu informaciju na druge neurone. Kada bi usporedili strukturu biološkog neurona s umjetnim neuronom ulazni podaci (x_i) i njihove težine (w_i) bi bili dendriti, funkcija sume tih signala pomnoženih s njihovim težinama i aktivacijska funkcija bi bilo tijelo neurona te akson i sinapse na kraju aksona bi bila izlazna vrijednost umjetnog neurona (y_i) koja se proslijeđuje dublje u mrežu. Kod prepoznavanja slika i segmenata slika ulazne vrijednosti neuronske mreže su pikseli slike, a izlazne vrijednosti su klase predmeta na slici u obliku vjerojatnosti prikazane od 0 do 1, odnosno od 0 do 100% vjerojatnosti da je slika određene klase.



Slika 1. Biološki neuron [3]



Slika 2. Umjetni neuron [2]

Duboke neuronske mreže se sastoje od više slojeva međusobno povezanih neurona. Učenje neuronske mreže svodi se na optimizaciju težina ulaznih vrijednosti neurona za smanjenje greške u predviđanju rješenja. Težine optimiziramo tako što reduciramo funkciju gubitka koja je suma grešaka za svaku klasu izlaznih vrijednosti neuronske mreže. Za prepoznavanje slika i obilježja na slikama preporuča se korištenje konvolucijskih neuronskih mreža. [1]

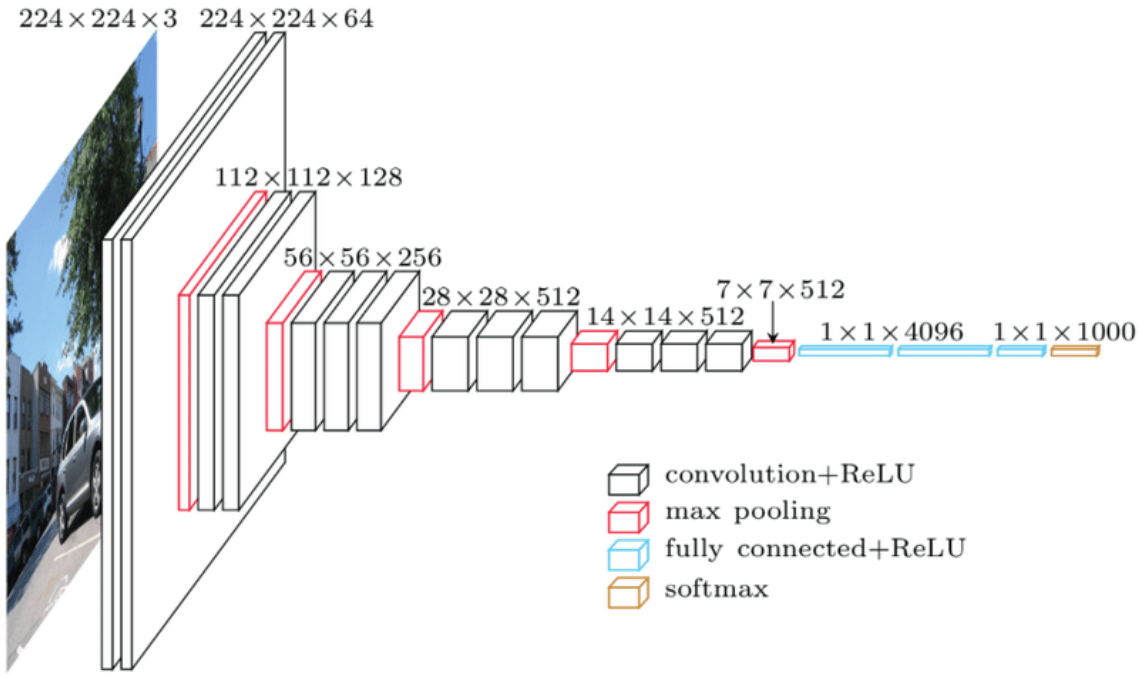
Konvolucijske neuronske mreže funkcioniraju na način da dijele sliku u podgrupe značajki koje spajaju u veću cjelinu.

2.1. *TensorFlow*

TensorFlow [6] je biblioteka otvorenog koda koja se koristi za rad sa neuronskim mrežama, manipuliranjem podacima učenja i testiranja te spremanjem i korištenjem dobivenog modela. Osnovan je 2015. godine od strane *Google*-a. Primjena biblioteke je moguća u različitim programskim jezicima poput *Python, C++*, *JavaScript* i *Java*. Struktura funkcija biblioteke omogućuje rad procesorskoj, grafičkoj i TPU procesnim jedinicama za izračun vrijednosti. Za ovaj završni rad učenje konvolucijske neuronske mreže izvršeno je pomoću grafičke jedinice na *Google*-ovim serverima putem *GoogleColab* servisa što je drastično ubrzalo učenje. Na privatnom računalu isti broj epoha učenja mreže je trajao 10h, dok na *GoogleColab* sučelju je trajao 15 min što je smanjenje vremena obrade i učenja približno 40 puta.

Za ovaj završni rad odabrana je *VGG16* konvolucijska neuronska mreža pred-učena na *ImageNet* setu podataka. *VGG16* [7] konvolucijska neuronska mreža je rezultat rada

K. Simonyana i A. Zissermana sa sveučilišta u Oxfordu 2014. godine. Strukturalno se sastoji od 16 međusobno povezanih slojeva, ulaza koji je slika dimenzija 224x224 piksela u 3 kanala. Dimenzija najmanjeg konvolucijskog sloja je 3x3 što omogućava veliku razlučivost detalja na slikama.



Slika 3. Struktura VGG16 konvolucijske neuronske mreže

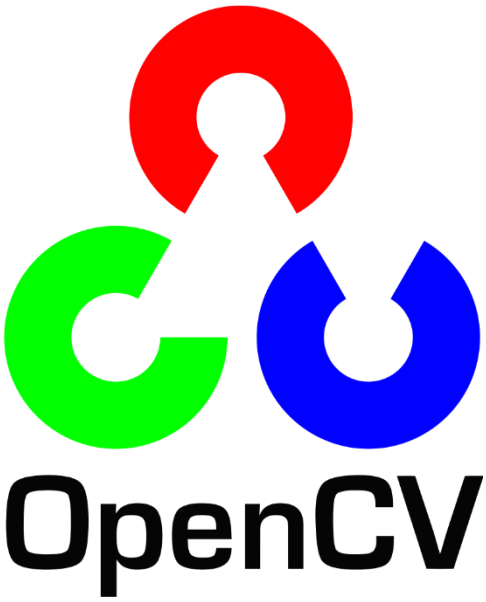
ImageNet je baza podataka slika s preko 14 milijuna slika i preko 21 tisuću klasa slika. Često se primjenjuje u strojnom učenju prepoznavanja slika radi kvalitete baze podataka i opširnog uzora po klasi što je otprilike 1000 slika po klasi u prosjeku.

3. RAČUNALNI VID

Računalni vid je disciplina računalnih znanosti koja se služi za davanje semantičke vrijednosti segmentima digitalnih slika u stvarnom životu. Primjenjuje se u industriji u područjima kvalitete kontrole proizvoda na proizvodnim trakama, nerazornim ispitivanjima [11]; na javnim površinama za prepoznavanje i praćenje počinitelja krivičnih djela; u mobilnim uređajima za detekciju lica i izoštravanje slike prilikom slikanja, skeniranje barkodova i QR kodova [10]; u autoindustriji pri razvoju samovozećih automobila [9]; u zdravstvenoj industriji za detekciju tumora [8]. U zadnjim desetljećima drastično se ubrzao rast područja računalnog vida zato što se omogućio pristup ogromnoj količini podataka razvojem tehnologija. U ovom radu fokusira se na rad računalnog vida u segmentiranju slike i prepoznavanju sadržaja segmenata.

3.1. *OpenCV*

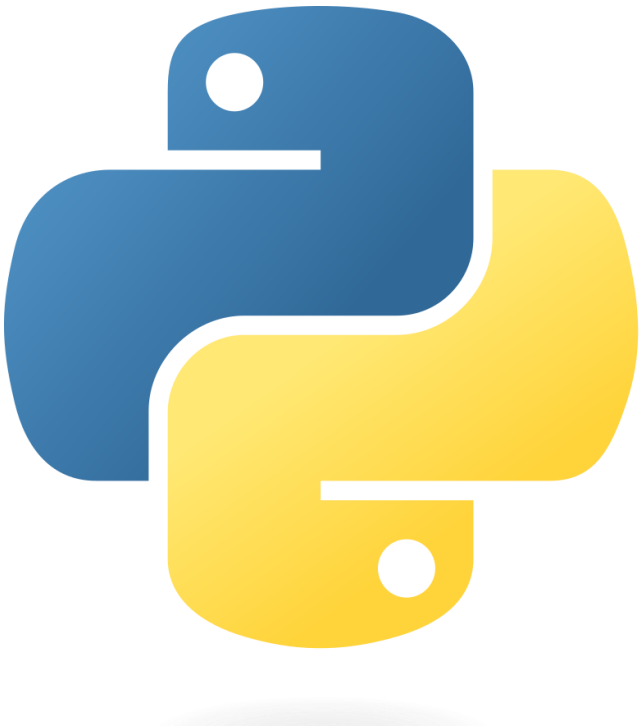
OpenCV je biblioteka otvorenog koda koja ugrađenim funkcijama drastično olakšava rad u okolini računalnog vida. Osnovana je 1999. godine od strane *Intel Research* inicijative [12]. Omogućila je optimizaciju metoda računalnog vida otvorenim pristupom. Prvenstveno se koristi za rad u realnom vremenu (en. r*eal-time*). Pisana je u programskom jeziku *C++*, ali podržava rad u drugim prgramskim jezicima (*Python*, *Java*) [12]. U radu je korištena *OpenCV* biblioteka zbog optimiziranosti funkcija, zahtjeva rada u realnom vremenu i odabira *Python*-a kao programskog jezika.



Slika 4. Logo *OpenCV* biblioteke [12]

4. *PYTHON*

Python je programski jezik razvijen 1991. godine. Zadnje stabilno izdanje je *Python 3.11* objavljeno 2022. godine [13]. Danas je jedan od najpopularnijih programskih jezika radi jednostavne sintakse i objektno orijentiranosti programskog jezika. To je programski jezik otvorenog koda opće namjene i izvodi se sekvencijalno liniju po liniju. Uslijed popularnosti i pristupačnosti prilikom rada u *Python*-u omogućen je pristup mnogim bibliotekama. Biblioteke programskih jezika su zbirke funkcija i operacijskih kodova za specifične probleme osmišljeni u svrhu olakšavanja programiranja bez potrebe da se kreće ispočetka. Služe unutar programa kao potprogrami i funkcije. Neke od najpoznatijih biblioteka koje se primjenjuju uz programski jezik *Python* su *NumPy* (omogućava rad u velikim multidimenzijskim matricama i array-evima), *SciPy* (služi za rješavanje matematičkih funkcija, linearnu algebru, interpolaciju i druge matematičke funkcije), *Keras* (omogućava suradnju *Python* sučelja sa *TensorFlow* bibliotekom), *PyTorch* (sučelje za rad sa strojnim učenjem), *TensorFlow* (biblioteka za strojno učenje fokusirana na učenje i rad s dubokim neuronskim mrežama), *OpenCV* (biblioteka funkcija za obradu slika i računalni vid u realnom vremenu), *Matplotlib* (biblioteka za izradu grafova, diagrama i prikaz matematičkih podataka). Ovaj rad je napravljen na programskom jeziku *Python* verzije 3.10.



Slika 5. Logo programskog jezika Python [13]

5. RAČUNALNI PROGRAM

Rad je napravljen u sklopu 3 računalna programa. Prvi je skripta za učenje neuronske mreže i spremanje evaluacijskog modela, drugi je biblioteka funkcija niže složenosti koje primjenjujemo pri izvršavanju trećeg programa koji izvršava korisničko sučelje i snimanje slika za obradu.

5.1. Prvi računalni program

Prvi računalni program naziva „*tensorflow_colab.py*“ izvršava učenje neuronske mreže *VGG16* na prepoznavanje šahovskih figura i praznih polja u *GoogleColab* okruženju. Da bi napravili specifičnu *VGG16* neuronsku mrežu za prepoznavanje šahovskih figura učitali smo *VGG16* s težinama naučenim na *ImageNet* setu podataka, zamrznuli težine, definirali izgled ulaznih vrijednosti i maknuli izlazni sloj mreže. Potom smo definirali izlazni sloj kao 13 neurona s 1 izlazom, za svaku figuru i prazno polje. Vršili smo učenje dvaju modela, jednog samo na slikama kose perspektive spremljenog pod nazivom „*model_VGG16_final.h5*“ i model učen na miješavini slika iz kose perspektive i vertikalne perspektive spremljenog pod nazivom „*model_mix_VGG16.h5*“. Da bi omogućili kvalitetno učenje izvršili smo povećanje broja slika za učenje manipulacijom rotacije slika i horizontalnim okretanjem slika. Ukupan broj zasebnih slika prije manipulacije je 2569, od čega je 2383 slike kose perspektive [4], a 186 vertikalne perspektive. Zatim je izvršeno testiranje oba modela na bazi podataka samo kosih slika, vertikalnih slika i miješavine slika čime smo dobili 3 matrice konfuzije po modelu. Za testiranje modela uzeto je nasumično 20% početne količine slika, jednakog omjera iz vertikalne i kose perspektive.

Funkcije primjenjene u prvom računalnom programu su sve ugrađene iz biblioteka *TensorFlow* i *Keras.*

5.2. Matrice konfuzije i metrike evaluacije

Matrice konfuzije su rezultirajuće matrice nakon testiranja naučene neuronske mreže. One su jedan od načina provjere kvalitete naučenosti mreže. Na ordinati matrice konfuzije stoje klase koje očekujemo u prepoznavanju, a na apscisi matrice se nalazi rezultat prediktivnog modela. Poželjna je dijagonalna raspodjela brojeva dok raspršivanje podataka van dijagonale matrice ukazuje na neuspješnu detekciju, odnosno lažne detekcije. Matrice konfuzije čine samo dio metoda testiranja kvalitativnog odziva mreže na pobudu. Ostale evaluacijske metrike su preciznost, točnost i *F1 Score*.

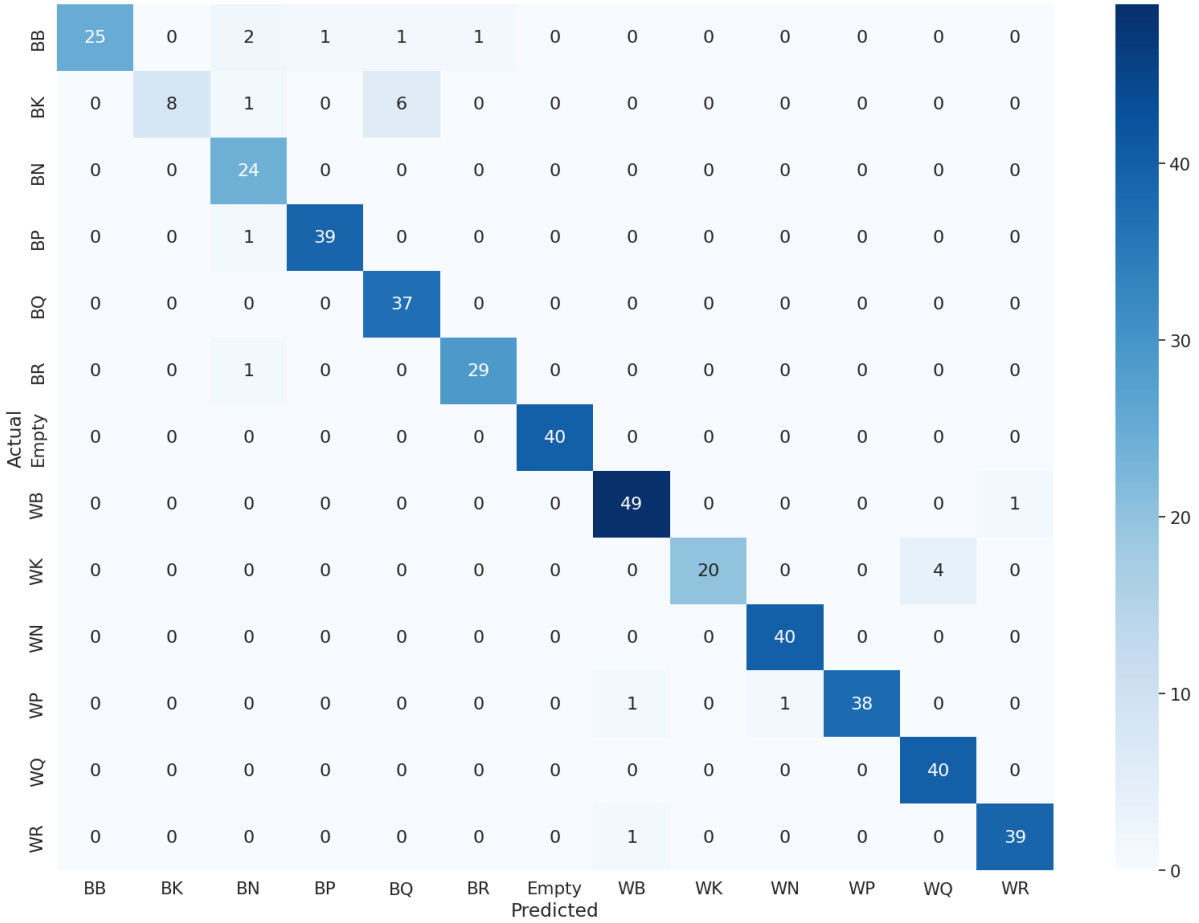
Preciznost neuronske mreže ukazuje na koliko model neuronske mreže napravi točnih klasifikacija na testnom setu podataka, dok točnost ukazuje na koliko je puta model napravio točnu predikciju unutar cijelog seta testnih podataka. Također jedan od metričkih kriterija koji se koristi je ponovljivost. Ponovljivost mreže ovisi o kriterijima prepoznavanja i najčešće s većom ponovljivosti opada preciznost i obrnuto, zato što ako su kriteriji prepoznavanja oštriji mreža kada prepoznaje predmet, manje će griješiti, ali radi oštrijih kriterija u manjem broju slučajeva će u dovoljnoj mjeri biti sigurna da opredjeli klasu predmetu koji prepoznaje.

F1 Score je nastao kao kombinacija metrike preciznosti i ponovljivosti, gdje porastom *F1 Score* funkcije se zahtjeva povećanu preciznost i povećanu ponovljivost modela. Razilaženje tih vrijednosti smanjuje *F1 Score.* Vrijednosti *F1 Score*-a mogu ići od 0 do 1, odnosno od 0 do 100%. Jedna od glavnih metoda provjere kvalitete modela prepoznavanja je upravo *F1 Score.*

Kod setova podataka s nesrazmjernim brojem primjeraka unutar klasa podataka koristi se težinski *F1 Score* (en. *weighted F1 Score*) koji uzima u obzir broj uzoraka unutar klasa. Rezultat težinskog F1 Score-a je suma F1 Score-a po klasi seta podataka pomnoženih s težinom koja se dobiva dijeljenjem broja uzoraka klase s ukupnim brojem uzoraka. [17]

Evaluacijske metrike dobivene su primjenom funkcije *classification_report()* biblioteke otvorenog koda *scikit-learn* koja nudi alate za rad s modelma prediktivnom analizom podataka.

Matrica konfuzije modela učenog na slikama iz kose perspektive je prikazana na slici niže [Slika 6]. Iz rezultata F1 Score-a u tom slučaju [Slika 7] vidimo da je mreža dobro naučena za prepoznavanje tog seta podataka s težinskim F1 Score-om od 0,95, odnosno 95%.



Slika 6. Matrica konfuzije prvog modela za bazu kosih podataka

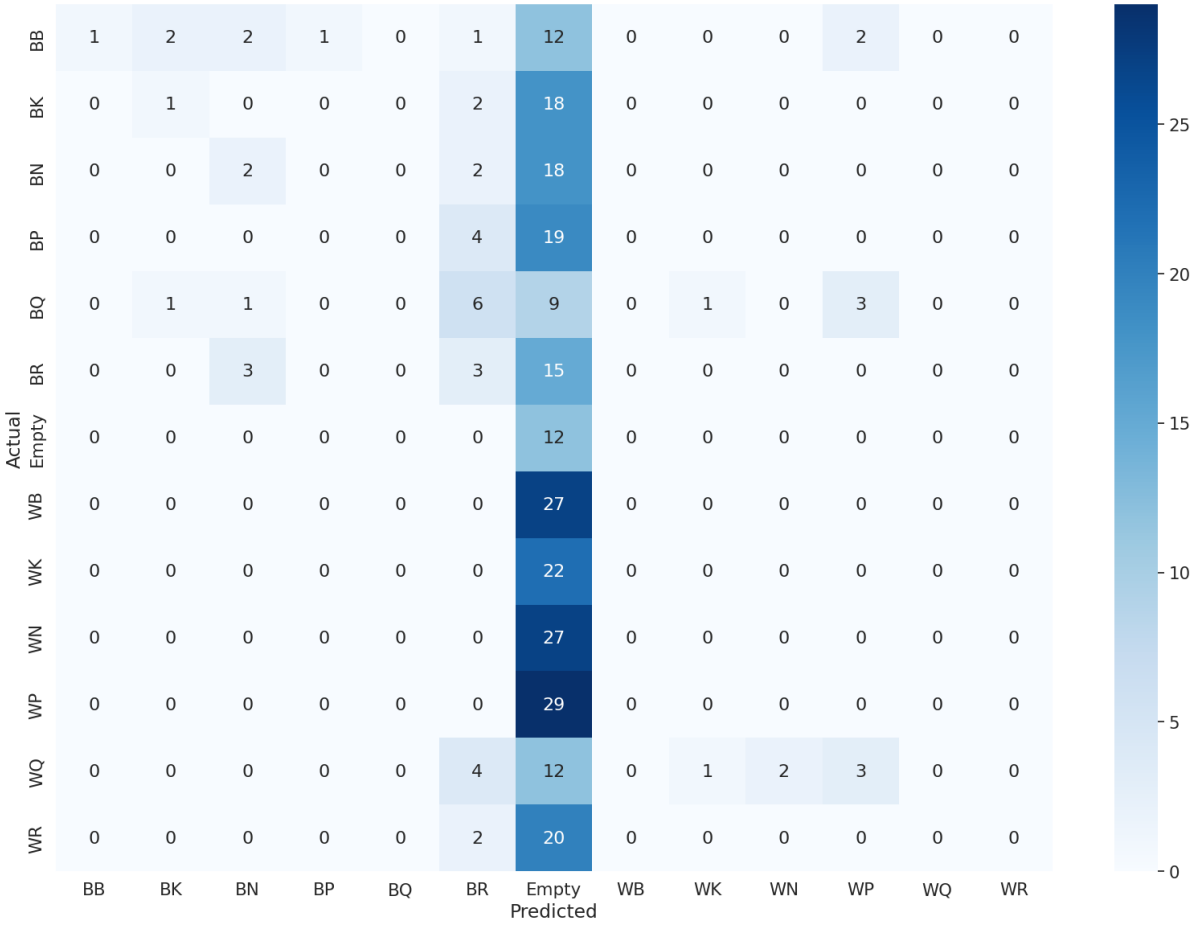
```
Confusion Matrix
[[25  0  2  1  1  1  0  0  0  0  0  0  0]
 [ 0  8  1  0  6  0  0  0  0  0  0  0  0]
 [ 0  0 24  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  1 39  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0 37  0  0  0  0  0  0  0  0]
 [ 0  0  1  0  0 29  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 40  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 49  0  0  0  0  1]
 [ 0  0  0  0  0  0  0  0 20  0  0  4  0]
 [ 0  0  0  0  0  0  0  0  0 40  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  1 38  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 40  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  0 39]]
Classification Report
              precision    recall  f1-score   support

          BB       1.00      0.83      0.91        30
          BK       1.00      0.53      0.70        15
          BN       0.83      1.00      0.91        24
          BP       0.97      0.97      0.97        40
          BQ       0.84      1.00      0.91        37
          BR       0.97      0.97      0.97        30
       Empty       1.00      1.00      1.00        40
          WB       0.96      0.98      0.97        50
          WK       1.00      0.83      0.91        24
          WN       0.98      1.00      0.99        40
          WP       1.00      0.95      0.97        40
          WQ       0.91      1.00      0.95        40
          WR       0.97      0.97      0.97        40

    accuracy                           0.95       450
   macro avg       0.96      0.93      0.93       450
weighted avg       0.96      0.95      0.95       450
```

Slika 7. Metrički rezultati kosog modela na kosim podacima

Rezultati predikcija istog modela kod vertikalnog seta podataka pokazuju da mreža učena na kosim slikama ima problema u prepoznavanju vertikalnih podataka što se vidi unutar matrice konfuzije [Slika 8] i *F1 Score*-a [Slika 9]. Iz matrice konfuzije možemo isčitati da je većina slika krivo prepoznato kao prazno polje osim u nekoliko instanci. Nepreciznost se vidi u očitanju preciznosti pojedinih klasa i unutar težinskog *F1 Score*-a.



Slika 8. Matrica konfuzije modela učenog na slikama kose perspektive za bazu vertikalnih podataka

Confusion Matrix

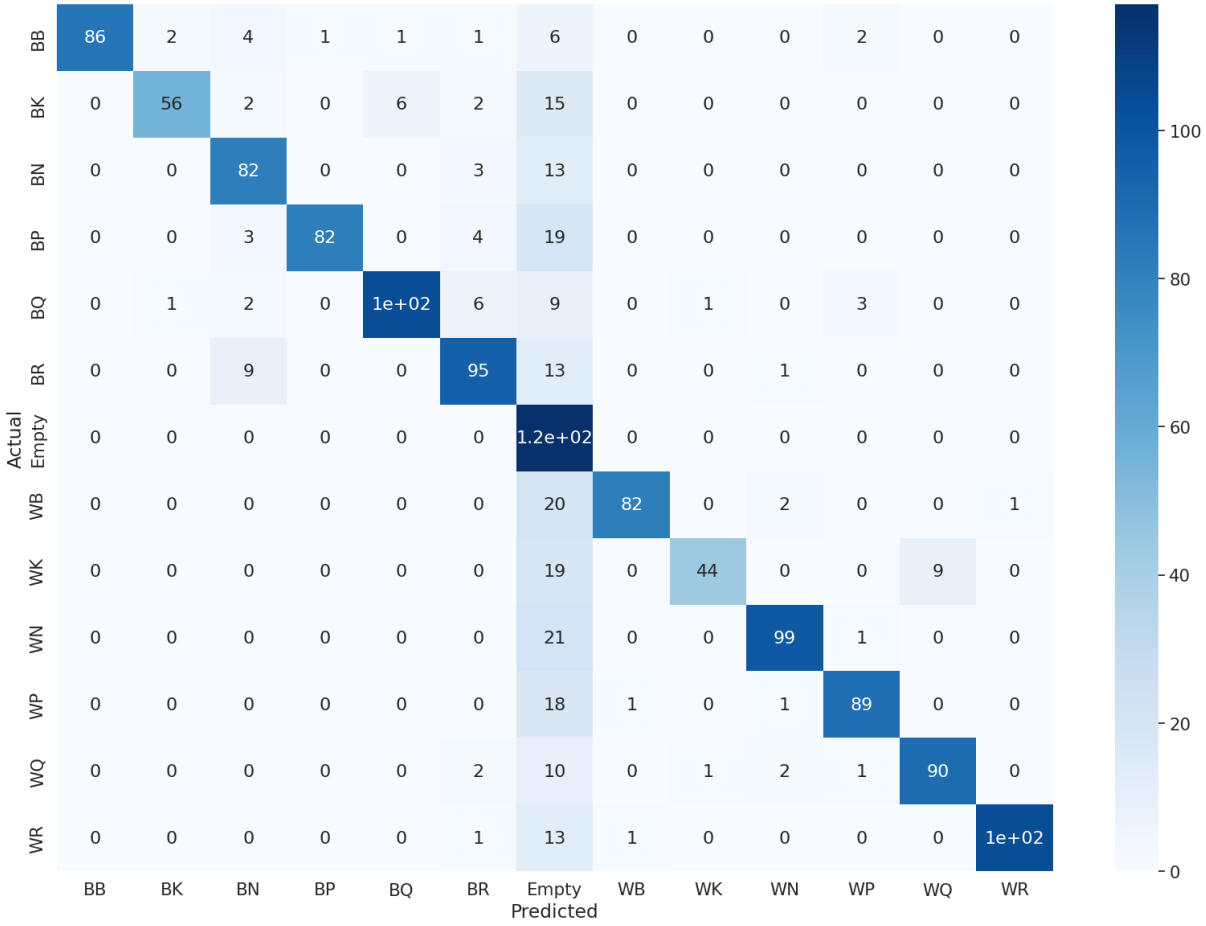
```
[[ 1  2  2  1  0  1 12  0  0  0  2  0  0]
 [ 0  1  0  0  0  2 18  0  0  0  0  0  0]
 [ 0  0  2  0  0  2 18  0  0  0  0  0  0]
 [ 0  0  0  0  0  4 19  0  0  0  0  0  0]
 [ 0  1  1  0  0  6  9  0  1  0  3  0  0]
 [ 0  0  3  0  0  3 15  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 12  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 27  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 22  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 27  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 29  0  0  0  0  0  0]
 [ 0  0  0  0  0  4 12  0  1  2  3  0  0]
 [ 0  0  0  0  0  2 20  0  0  0  0  0  0]]
```

Classification Report

	precision	recall	f1-score	support
BB	1.00	0.05	0.09	21
BK	0.25	0.05	0.08	21
BN	0.25	0.09	0.13	22
BP	0.00	0.00	0.00	23
BQ	0.00	0.00	0.00	21
BR	0.12	0.14	0.13	21
Empty	0.05	1.00	0.10	12
WB	0.00	0.00	0.00	27
WK	0.00	0.00	0.00	22
WN	0.00	0.00	0.00	27
WP	0.00	0.00	0.00	29
WQ	0.00	0.00	0.00	22
WR	0.00	0.00	0.00	22
accuracy			0.07	290
macro avg	0.13	0.10	0.04	290
weighted avg	0.12	0.07	0.04	290

Slika 9. Metričke vrijednost odziva na vertikalnom setu podataka modela učenog na kosom setu podataka

Kod odziva modela učenog na kosim podacima u slučaju miješanog testnog seta podataka možemo vidjeti utjecaj nemogućnosti točnog prepoznavanja vertikalnih slika u matrici konfuzije [Slika 10]. Unutar miješanog seta podataka za testiranje oko 15% čine slike iz vertikalne perspektive što se vidi da utječe na preciznost i *F1 Score* prepoznavanja modela na tom setu podataka [Slika 11].



Slika 10. Matrica konfuzije prvog modela za bazu miješanih podataka

Confusion Matrix

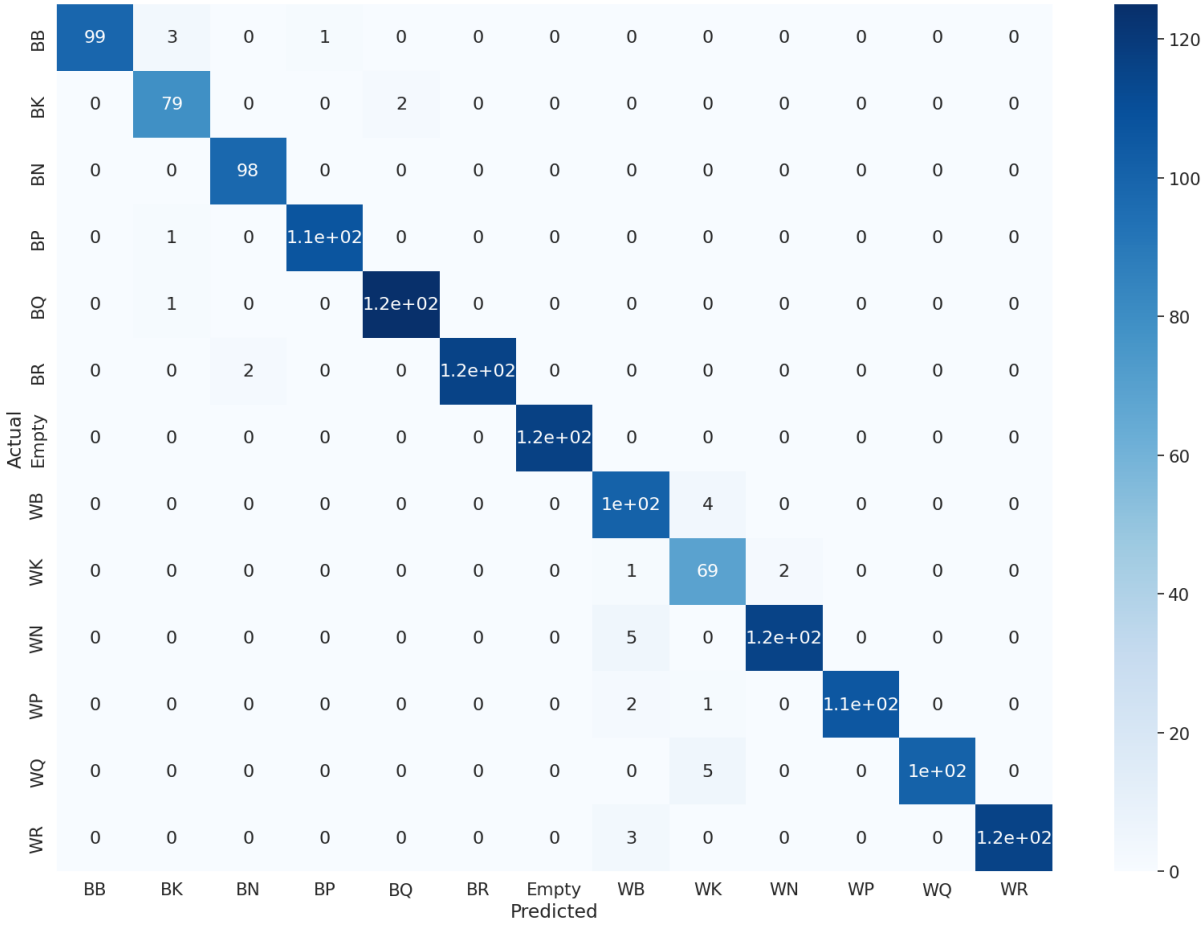
```
[[ 86   2   4   1   1   1   6   0   0   0   2   0   0]
 [  0  56   2   0   6   2  15   0   0   0   0   0   0]
 [  0   0  82   0   0   3  13   0   0   0   0   0   0]
 [  0   0   3  82   0   4  19   0   0   0   0   0   0]
 [  0   1   2   0 104   6   9   0   1   0   3   0   0]
 [  0   0   9   0   0  95  13   0   0   1   0   0   0]
 [  0   0   0   0   0   0 117   0   0   0   0   0   0]
 [  0   0   0   0   0   0  20  82   0   2   0   0   1]
 [  0   0   0   0   0   0  19   0  44   0   0   9   0]
 [  0   0   0   0   0   0  21   0   0  99   1   0   0]
 [  0   0   0   0   0   0  18   1   0   1  89   0   0]
 [  0   0   0   0   0   2  10   0   1   2   1  90   0]
 [  0   0   0   0   0   1  13   1   0   0   0   0 104]]
```

Classification Report

	precision	recall	f1-score	support
BB	1.00	0.83	0.91	103
BK	0.95	0.69	0.80	81
BN	0.80	0.84	0.82	98
BP	0.99	0.76	0.86	108
BQ	0.94	0.83	0.88	126
BR	0.83	0.81	0.82	118
Empty	0.40	1.00	0.57	117
WB	0.98	0.78	0.87	105
WK	0.96	0.61	0.75	72
WN	0.94	0.82	0.88	121
WP	0.93	0.82	0.87	109
WQ	0.91	0.85	0.88	106
WR	0.99	0.87	0.93	119
accuracy			0.82	1383
macro avg	0.89	0.81	0.83	1383
weighted avg	0.89	0.82	0.83	1383

Slika 11. Metričke vrijednost odziva na miješanom setu podataka modela učenog na kosom setu podataka

Kod modela učenog na miješanom setu podataka kose i vertikalne perspektive primjećuje se manjak nepreciznosti u odzivu u matrici konfuzije [Slika 12]. Odstranjen je dio netočnih predikcija vertikalnih slika koji se javljao u prethodnom modelu. Također, težinski *F1 Score* je ostao unutar vrijednosti od 90 do 100 posto što ukazuje na dobro naučenu mrežu za zadani set testnih podataka [Slika 13].



Slika 12. Matrica konfuzije modela učenog na bazi miješanih podataka za bazu miješanih podataka

```
Confusion Matrix
[[ 99   3   0   1   0   0   0   0   0   0   0   0   0]
 [  0  79   0   0   2   0   0   0   0   0   0   0   0]
 [  0   0  98   0   0   0   0   0   0   0   0   0   0]
 [  0   1   0 107   0   0   0   0   0   0   0   0   0]
 [  0   1   0   0 125   0   0   0   0   0   0   0   0]
 [  0   0   2   0   0 116   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0 117   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 101   4   0   0   0   0]
 [  0   0   0   0   0   0   0   1  69   2   0   0   0]
 [  0   0   0   0   0   0   0   5   0 116   0   0   0]
 [  0   0   0   0   0   0   0   2   1   0 106   0   0]
 [  0   0   0   0   0   0   0   0   5   0   0 101   0]
 [  0   0   0   0   0   0   0   3   0   0   0   0 116]]
Classification Report
              precision    recall  f1-score   support

          BB       1.00      0.96      0.98       103
          BK       0.94      0.98      0.96        81
          BN       0.98      1.00      0.99        98
          BP       0.99      0.99      0.99       108
          BQ       0.98      0.99      0.99       126
          BR       1.00      0.98      0.99       118
       Empty       1.00      1.00      1.00       117
          WB       0.90      0.96      0.93       105
          WK       0.87      0.96      0.91        72
          WN       0.98      0.96      0.97       121
          WP       1.00      0.97      0.99       109
          WQ       1.00      0.95      0.98       106
          WR       1.00      0.97      0.99       119

    accuracy                           0.98      1383
   macro avg       0.97      0.98      0.97      1383
weighted avg       0.98      0.98      0.98      1383
```

Slika 13. Metričke vrijednost odziva na miješanom setu podataka modela učenog na miješanom setu podataka

5.3. Drugi računalni program

Drugi računalni program naziva „*cv_chess_functions.py*“ služi kao biblioteka operacija niže složenosti manipulacije slikama i podacima. Funkcije definirane unutar drugog računalnog programa su:

- *read_img(file)* — učitava radnu sliku i izlazne vrijednosti su originalna slika i zamućena siva slika
- *canny_edge(img)* — učitava zamućenu sivu sliku, zatim ispisuje sliku s iscrtanim rubovima prijelaza između segmenata, primjenjuje Canny-ev algoritam detekcije rubova [14]
- *hough_line(edges)* — učitava izlaznu sliku od *canny_edge* funkcije i vrši detekciju ravnih linija koje zapisuje u sfernom koordinatnom sustavu, primjenuje Hough-ovu transformaciju [15]
- *h_v_lines(lines)* — s obzirom na parametre linija dobivenih funkcijom *hough_line* dijeli linije na vertikalne i horizontalne s obzirom na kut theta u sfernim koordinatama
- *line_intersections(h_lines, v_lines)* — pronalazi presjecišta vertikalnih i horizontalnih linija pomoću ugrađene funkcije *Numpy* biblioteke *linalg.solve* koji rješava sustave linearnih jednadžbi. Izlazne vrijednosti su koordinate presjecišta linija
- *cluster_points(points)* — učitava dobivene točke presjecišta i opredjeljuje im grupacije s obzirom na udaljenost. Izlazna vrijednost je vanjeske koordinate grupacija točaka
- *merge_points(points)* — briše višak točaka oko vanjskih točaka grupacija s obzirom na udaljenost. Izlazna vrijednost su koordinate preostalih točaka
- *write_crop_image(img, points, img_count = 0, folder_path='./Data/raw_data/')* — uzima koordinate preostalih točaka, reže glavnu sliku u segmente s obzirom na koordinate točaka i pohranjuje izrezane slike u folder *'raw_data'* iz kojeg kasnije uzima .png datoteke za prepoznavanje
- *classify_cells(model, img_filename_list)* — uzima slike iz foldera i pomoću prijašnje naučenog modela ih kategorizira i zapiše u FEN šahovsku notaciju.

5.4. Treći računalni program

Treći program naziva „*chess.py*“ se koristi kao korisničko sučelje pri kojem pritiskom na tipku razmaka snimamo sliku i ako su uspješno detektirane točke ploče provodimo ostatak programa.

Putem stiska tipke „q“ izlazimo iz programa i gasimo sve otvorene prozore.

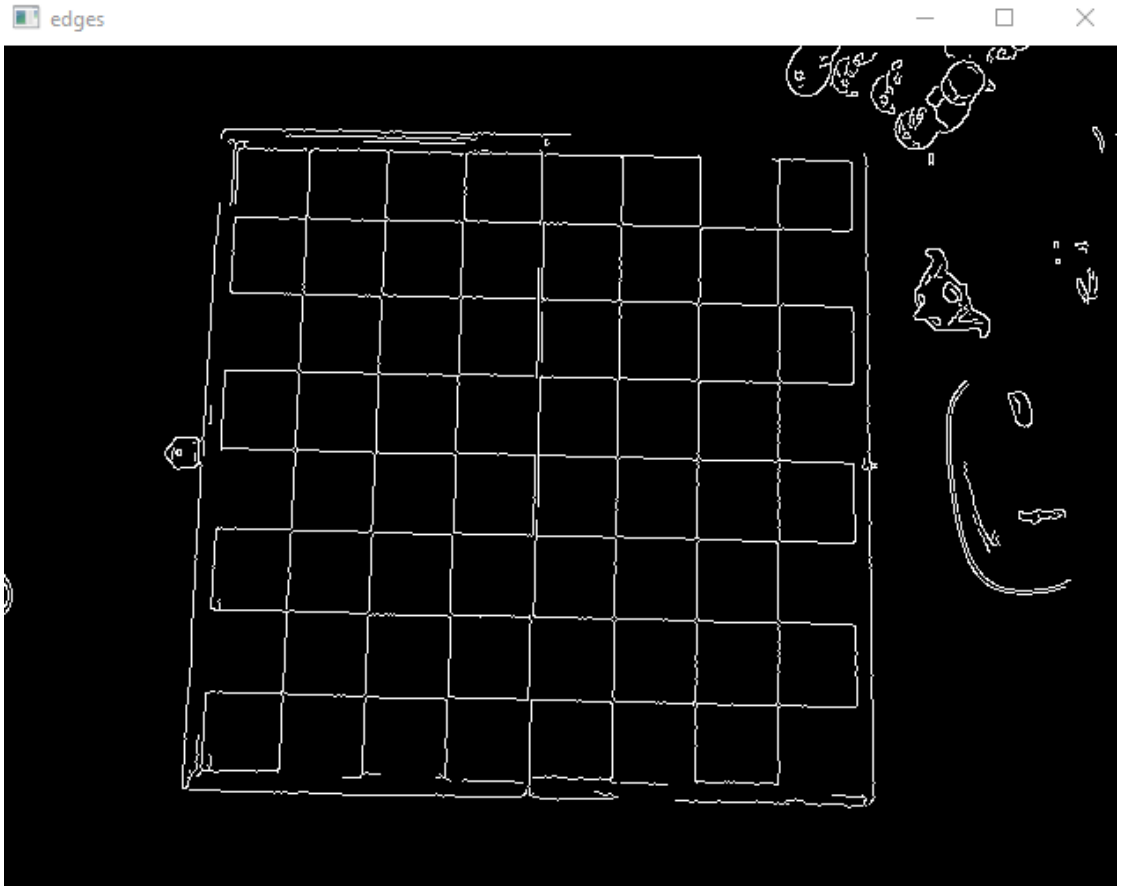
Izlaz trećeg programa je slika šahovske ploče dobivene pomoću FEN šahovske notacije nakon prepoznavanja figura.

5.5. Primjer rezultata

Rezultati pojedinih funkcija su zadovoljavajući. Dobivene su točne koordinate točaka i dobro je namješteno ograničenje na Canny algoritmu za traženje rubova.

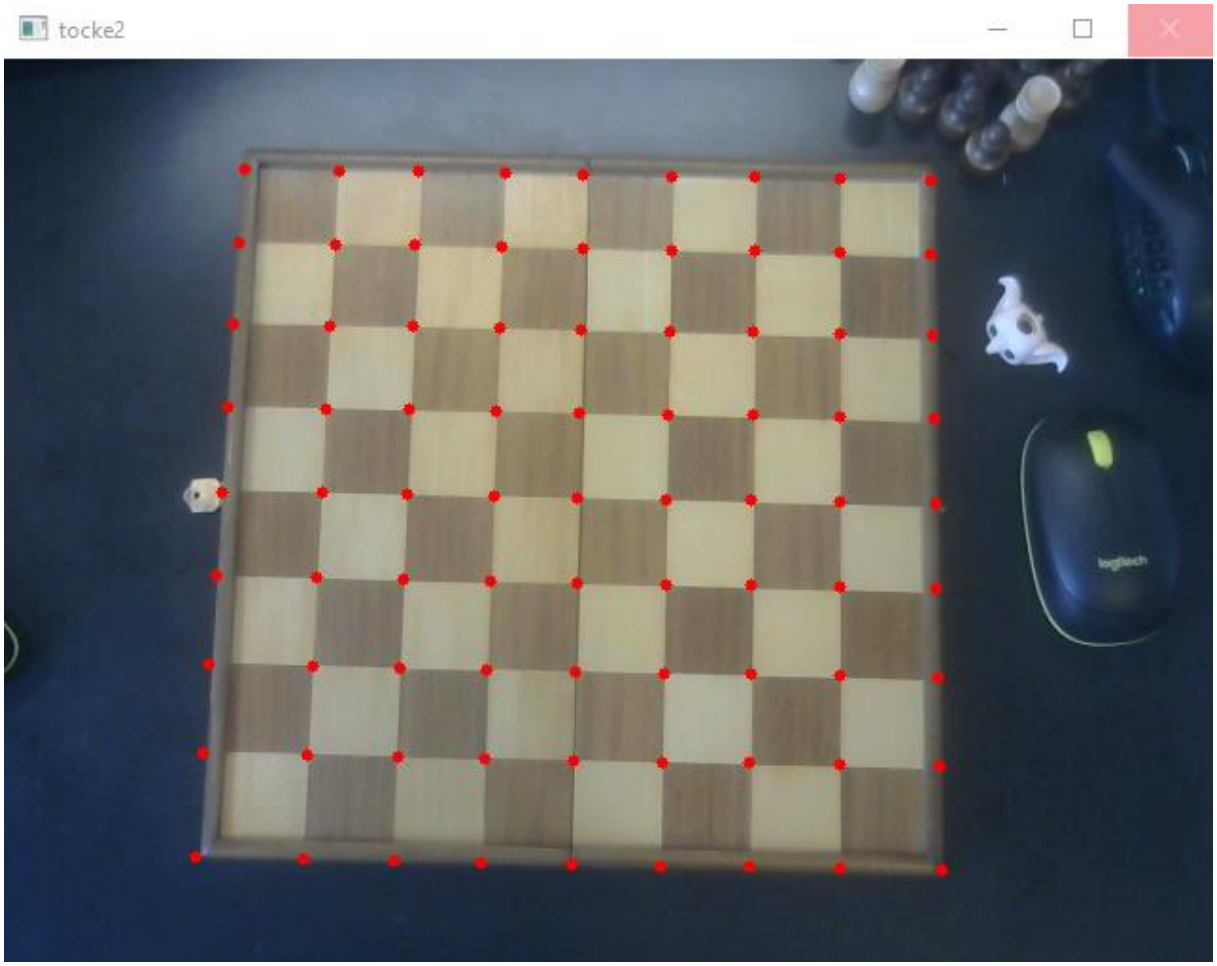
5.6. Primjer rezultata 1

Na slici [Slika 14] primjera 1 primjećujemo da se nije cijela kontura svakog polja detektirala, ali vidimo da nije bitno zato što pomoću produljenja linija i dalje imamo cijelo polje točaka [Slika 15] koje nam definira polja šahovske ploče [Slika 16].



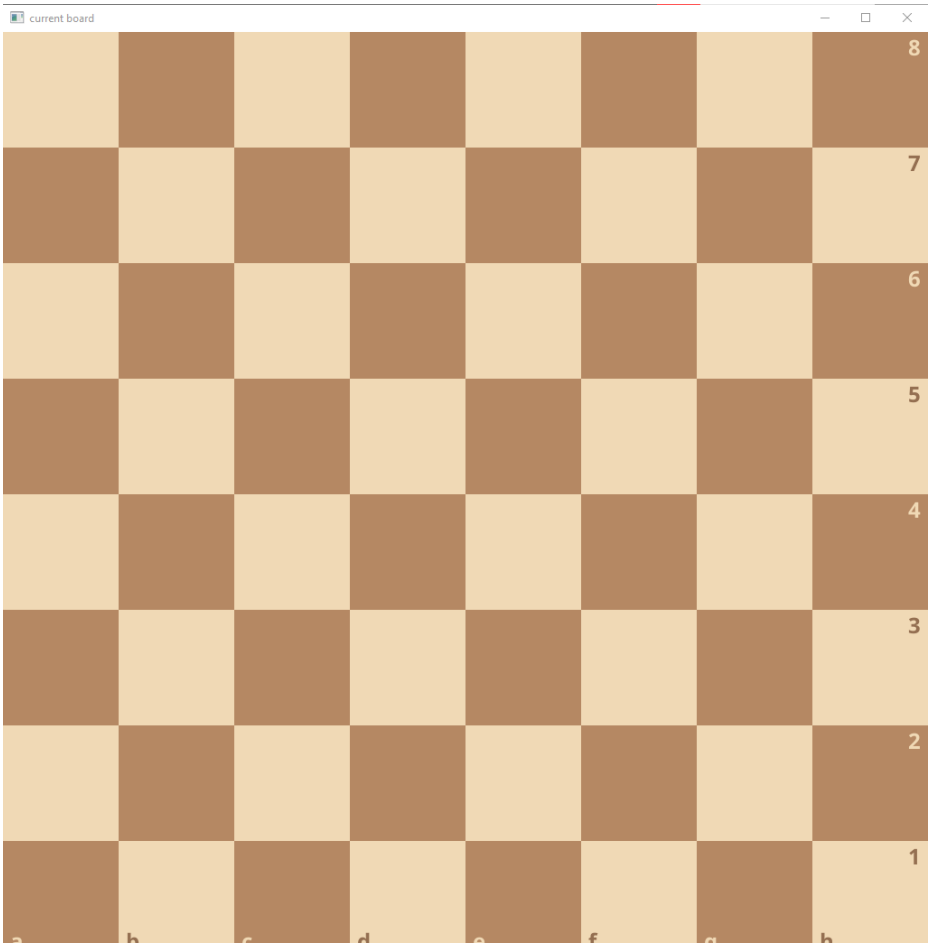
Slika 14. Canny_edge transformacija originalne slike primjer 1

Na prikazano slici [Slika 15] prikazane su preostale presjecišne točke nakon reduciranja šuma koji se javlja prilikom prve detekcije vertikalnih linija i njihovih presjecišta.



Slika 15. Točke presjecišta primjer 1

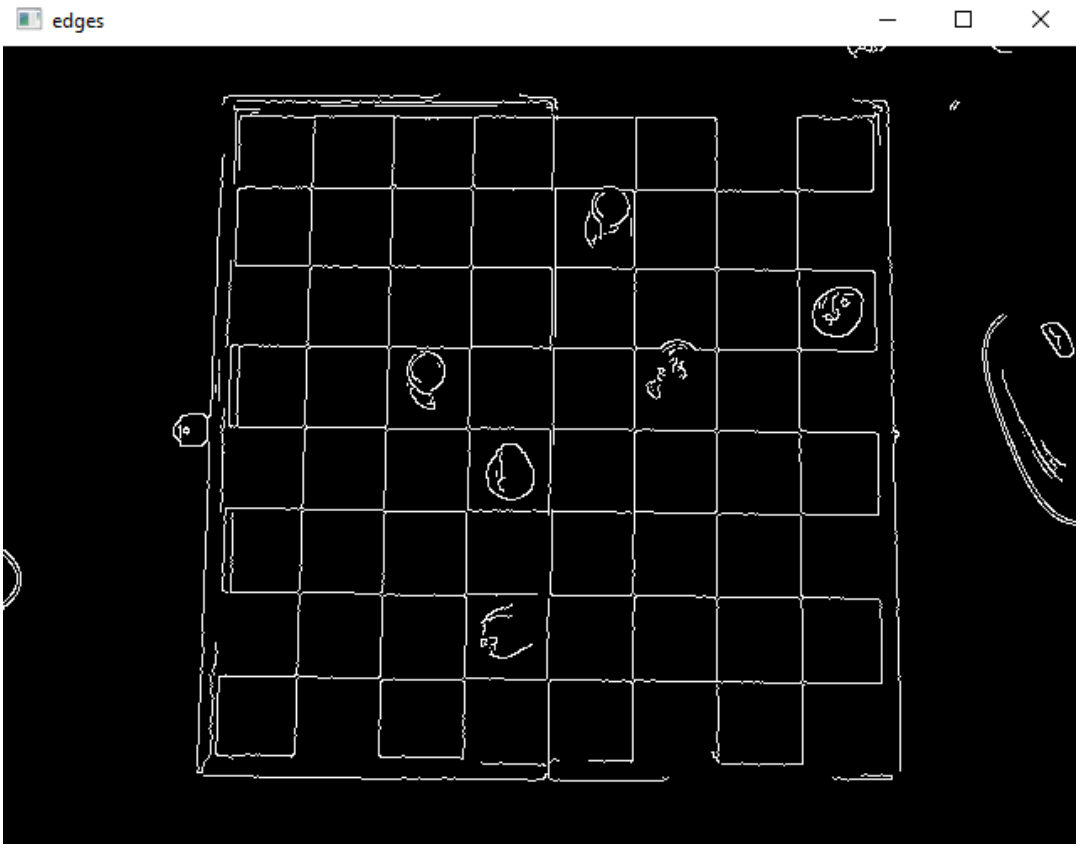
Slika šahovske ploče [Slika 16] je dobivena prebacivanjem aproksimacija modela u FEN šahovsku notaciju.



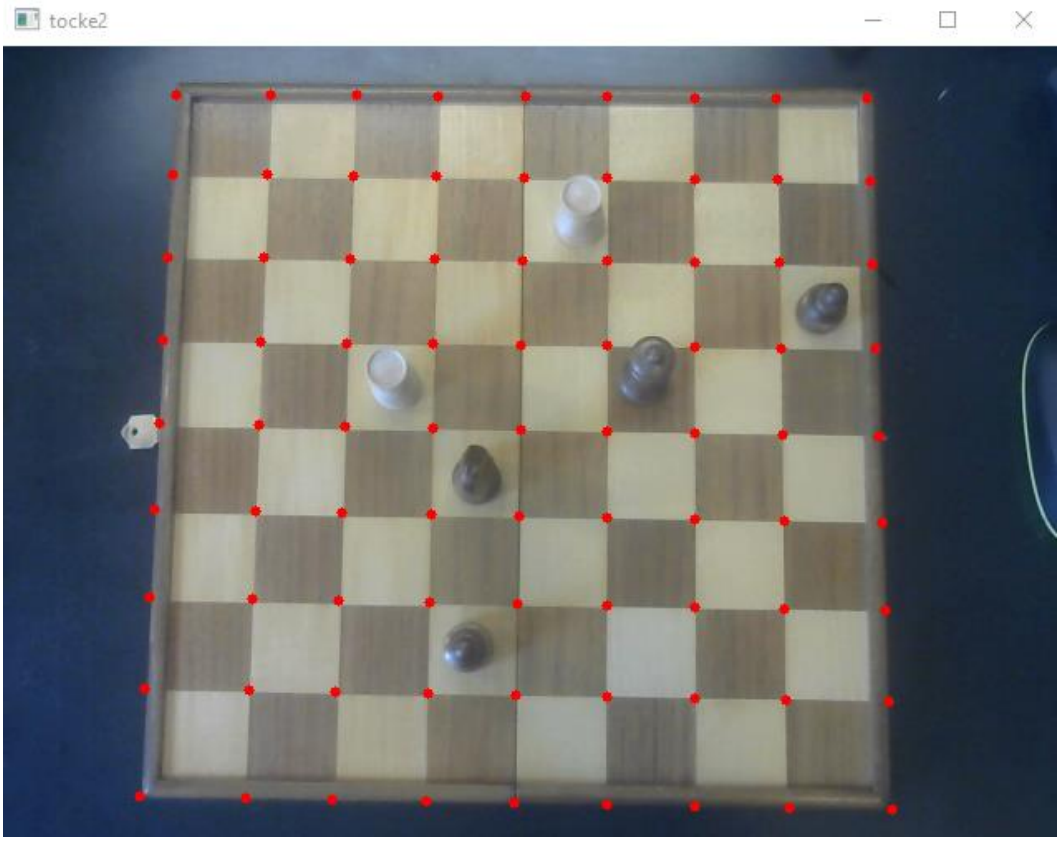
Slika 16. Rezultat algoitma prepoznavanja primjer 1

5.7. Primjer rezultata 2

Na primjeru 2 primjećujemo da je detekcija točaka na slici [Slika 18] i dalje potpuna iako se linije presjecaju figurama [Slika 17] i imamo dodatan šum figura na ploči, ali primjećujemo na krajnjem rezultatu prepoznavanja na slici [Slika 19] netočnost u prepoznavanju po poljima. Dubljom analizom nailazimo na grešku u algoritmu dijeljenja ploče na polja odnosno u detekciji objekata, algoritam ne grabi singularna polja uvijek nego u nekim instancama što vidimo na priloženoj slici [Slika 20], većinom hvata okolna polja što zbunjuje neuronsku mrežu.

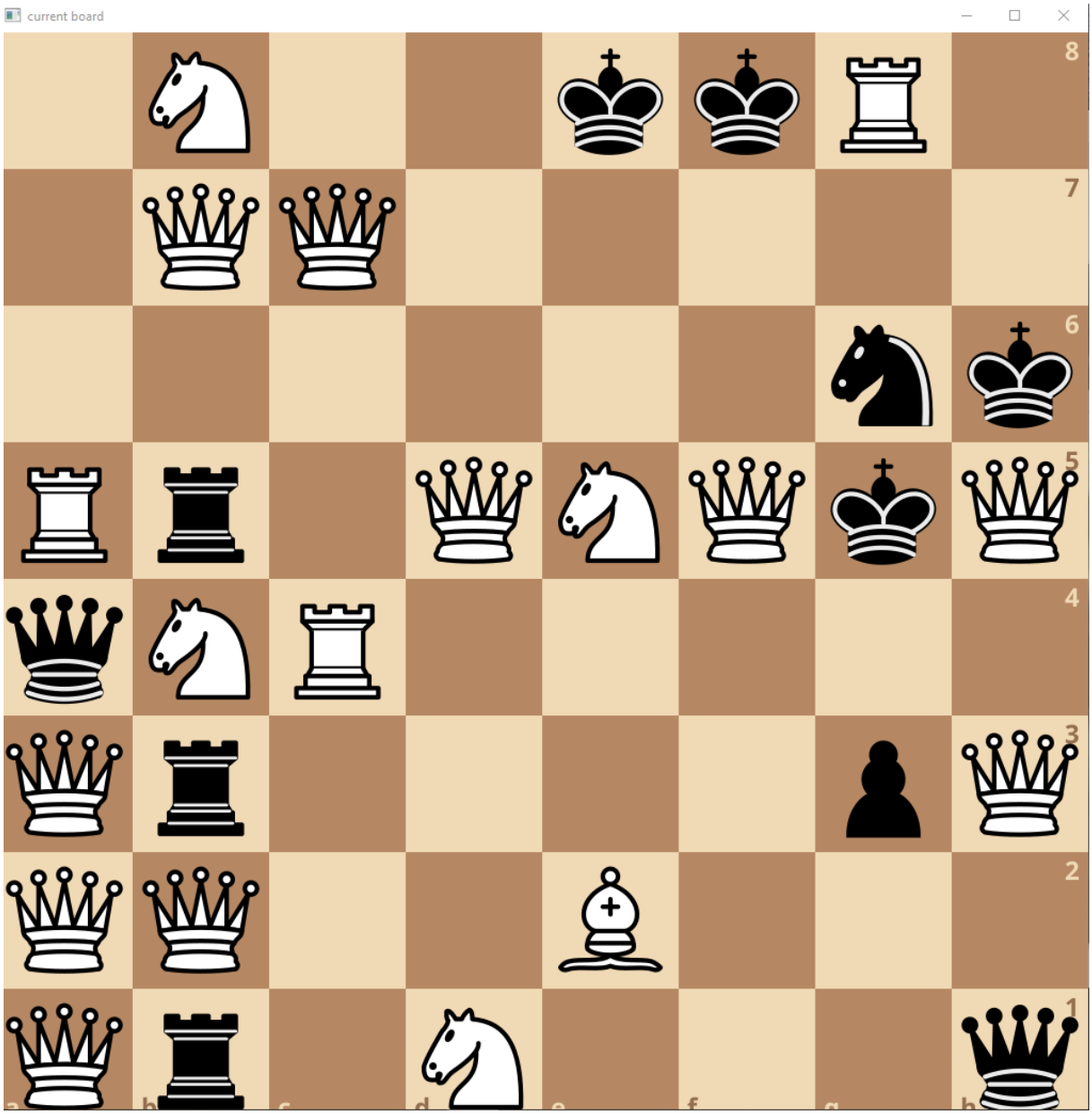


Slika 17. Rezultat Canny algoritma primjera 2



Slika 18. Dobivene točke podjele slike primjera 2

Prikaz šahovske ploče [Slika 19] dobiven programskim kodom prikazuje nerealno i nemoguće stanje šahovske ploče. Dok je u stvarnosti bilo samo 6 figura na ploči, dok je algoritam za prepoznavanje prepoznao 27 figura na terenu uzrokovano loše izrezanim slikama polja.



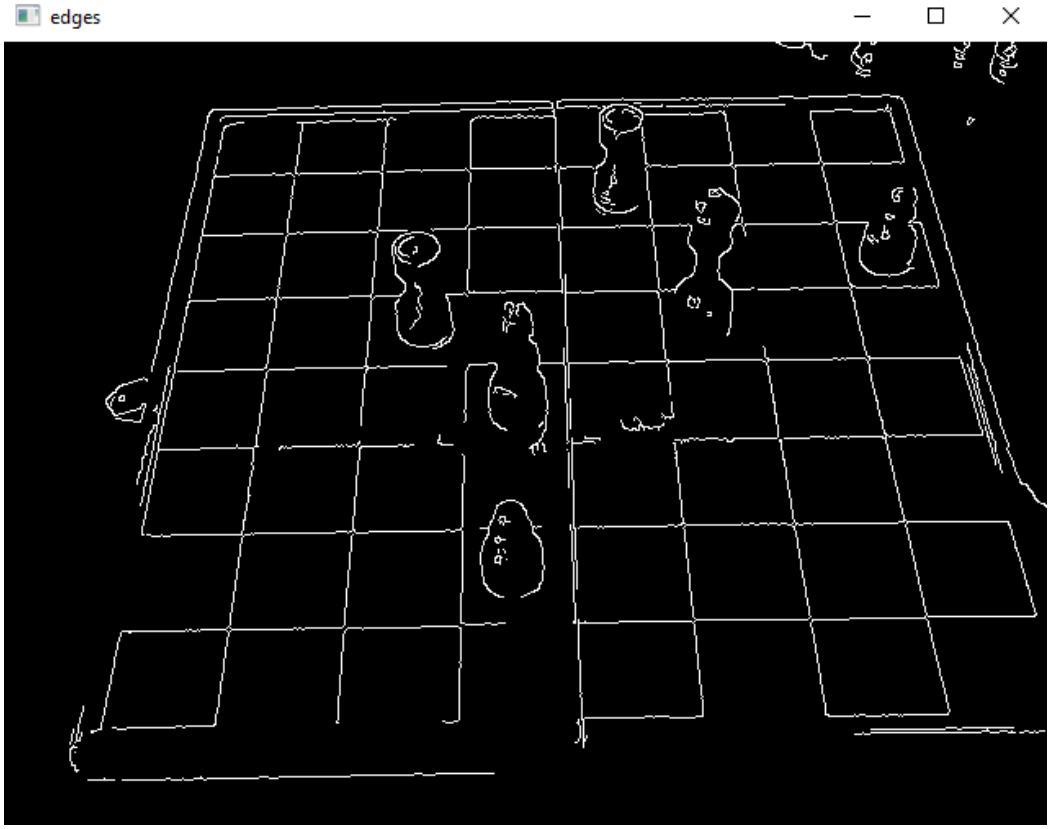
Slika 19. Rezultat prepoznavanja primjera 2

U isječenim slikama primjera 2 [Slika 20] primjećujemo da prilikom odsjecanja dolazi do greške gdje se odsjeca nepotrebni višak oko figure što netočno ukazuje na polje i time onemogućava pravilno prepoznavanje figure na šahovskom polju.

Slika 20. Isječene slike iz originalne slike primjera 2

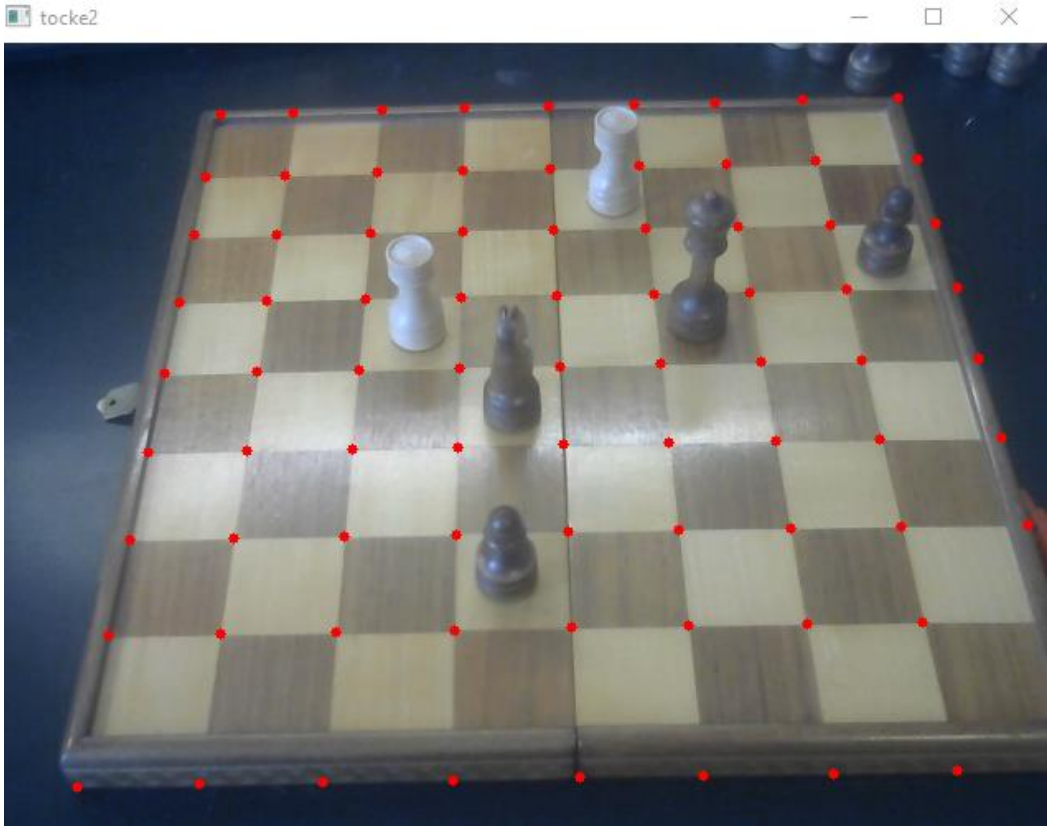
5.8. Primjer rezultata 3

Iz kosog pogleda Canny algoritam [Slika 21] i dalje prepoznaje linije šahovske ploče i algoritam pronalaženja točaka pronalazi sjecišta linija usprkos šumu [Slika 22].



Slika 21. Rezultat Canny algoritma primjera 3

Na prikazu presjecišnih točaka primjera 3 [Slika 22] primjećujemo da je algoritam krivo naznačio donji vanjski rub šahovske ploče kao rub polja. Takvo rezanje šahovske ploče utječe na rezultate prepoznavanja, ali u manjoj mjeri nego miješanje polja zato što u pravilnom algoritmu odsjecanja polja samo dio slike bi bio zahvaćen vanjskim rubom ploče. To bi i dalje omogućilo prepoznavanje figure, samo je pitanje preciznosti prepoznavanja.



Slika 22. Rezultat točaka ploče primjera 3

6. KRITIČKI OSVRT

Iz priloženih matrica konfuzije dvaju modela [Slika 6] [Slika 8] [Slika 10] [Slika 12] vidimo da su slike figura i ploče iz vertikalne perspektive i kose perspektive komplementarne. Model koji je primjenjivao i jedan i drugi set odataka tokom učenja nije izgubio mogućnost pravilne detekcije figura jednog od seta. Potpuni model za prepoznavanje šahovskih figura na ploči zahtjeva obe baze podataka da bi pozicija kamere za snimanje šahovske ploče mogla biti neovisna o kutu na samu ploču, a da bi figure bile kvalitetno prepoznate.

U primjeni prepoznavanja na primjerima potpune ploče naišli smo na problem detekcije objekata i detekcije samih polja šahovske ploče što je učinilo prepoznavanje figura na poljima nemoguće.

Time, iako je mreža naučena na miješanim podacima dobro naučena sa prihvatljivim rezultatima evaluacijskih metrika, njene sposobnosti prepoznavanja su neprimjenjive radi nepotpunosti programa.

To nam ukazuje na kritičnost ne samo kvalitete i opsežnosti baze podataka za učenje i testiranje kvalitetnog modela (podaci iz više kutova pogleda na kameru, pravilno označeni podaci i u dovoljnom broju za kvalitetno testiranje i učenje) nego i koliko je bitno pravilno segmentiranje slike za detekciju traženih objekata.

7. ZAKLJUČAK

Umjetna inteligencija i računalni vid su sve prisutniji u svakodnevnom životu čovjeka. U pronalaženju rješenja za nove probleme koji nastaju najveća prepreka primjene umjetne inteligencije je dolazak do kvalitetnih podataka kojih je kvantitativno dovoljno.

Neupitno je da će se računalni vid i umjetna inteligencija i dalje razvijati dok čovječanstvo postoji, nalaziti će se nove metode prikupljanja podataka i omogućiti će primjenu u širim poljima i na realnim problemima.

Ovim radom se pokazalo da je bitna opsežnost podataka za učenje modela da bi naučeni model bio potpun u svom djelovanju i neovisan o manjim promjenama parametara poput kuta gledanja i perspektive.

LITERATURA

[1] Szeliski, R.: Computer Vision: Algorithms and Applications 2nd Edition, Springer, 2022.

[2] Majetić, D.: predavanja iz kolegija Neuronske mreže, Fakultet strojarstva i brodogradnje, Zagreb, 2020.

[3] https://edutorij.e-skole.hr/share/proxy/alfresco-noauth/edutorij/api/proxy-guest/3b8a4b4e-84b0-4580-aa6f-e38efe028ed9/biologija-8/m03/j01/index.html, zadnji pristup 17.9.2023.

[4] https://www.dropbox.com/sh/8s4tvir5zbotseq/AACAQypmuFb6j-Yww9x9Q6Gta?dl=0, zadnji pristup 17.9.2023.

[5] https://www.tensorflow.org/, zadnji pristup 17.9.2023.

[6] https://en.wikipedia.org/wiki/TensorFlow, zadnji pristup 17.9.2023.

[7] https://arxiv.org/abs/1409.1556, zadnji pristup 17.9.2023.

[8] https://www.nature.com/articles/s41586-019-1799-6.epdf, zadnji pristup 17.9.2023.

[9] https://plainsight.ai/blog/autonomous-vehicles-computer-vision/, zadnji pristup 17.9.2023.

[10] https://mobisoftinfotech.com/resources/blog/how-is-mobile-computer-vision-changing-the-world/, zadnji pristup 17.9.2023.

[11] https://www.ndt.net/article/wcndt2016/papers/mo2h3.pdf, zadnji pristup 17.9.2023.

[12] https://en.wikipedia.org/wiki/OpenCV, zadnji pristup 17.9.2023.

[13] https://en.wikipedia.org/wiki/Python_(programming_language), zadnji pristup 17.9.2023.

[14] https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html, zadnji pristup 17.9.2023.

[15] https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html, zadnji pristup 17.9.2023.

[16] https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1, zadnji pristup 17.9.2023.

[17] https://www.v7labs.com/blog/f1-score-guide, zadnji pristup 20.9.2023.

PRILOZI

I. Programski kod s Google Colab servisa tensorflow_colab

II. Programski kod chess_functions.py

III. Programski kod chess.py

PYTHON KOD

```
# -*- coding: utf-8 -*-
"""tensor_test.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1VaGSvKbyrzU13ZChAkxG-y49M8-ulx4a
"""

from google.colab import drive
drive.mount('/content/drive')
#2 modela, bocni + mix, 3 testne grupe, bocni, vert i mix

import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

folder = '/content/drive/My Drive/Zavrsni_Data'
image_size = (224, 224)
batch_size = 32

from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        rotation_range=5,
        # width_shift_range=0.1,
        # height_shift_range=0.1,
        rescale=1./255,
        horizontal_flip=True,
        fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_gen = datagen.flow_from_directory(
    folder + '/train',
    target_size = image_size,
    batch_size = batch_size,
    class_mode = 'categorical',
    color_mode = 'rgb',
    shuffle=True
)

train_gen_mix = datagen.flow_from_directory(
    folder + '/train_mix',
    target_size = image_size,
    batch_size = batch_size,
    class_mode = 'categorical',
```

```
    color_mode = 'rgb',
    shuffle=True
)

test_gen = test_datagen.flow_from_directory(
    folder + '/test',
    target_size = image_size,
    batch_size = batch_size,
    class_mode = 'categorical',
    color_mode = 'rgb',
    shuffle=False
)

test_gen_vert = test_datagen.flow_from_directory(
    folder + '/test_vert',
    target_size = image_size,
    batch_size = batch_size,
    class_mode = 'categorical',
    color_mode = 'rgb',
    shuffle=False
)

test_gen_mix = test_datagen.flow_from_directory(
    folder + '/test_mix',
    target_size = image_size,
    batch_size = batch_size,
    class_mode = 'categorical',
    color_mode = 'rgb',
    shuffle=False
)

from keras.applications.vgg16 import VGG16
from keras.applications.imagenet_utils import decode_predictions

model = VGG16(weights='imagenet')
model.summary()

from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
from keras.models import Model

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,224,3))

# Freeze convolutional layers
for layer in base_model.layers:
    layer.trainable = False

# Establish new fully connected block
x = base_model.output
x = Flatten()(x)  # flatten from convolution tensor output
```

```
x = Dense(500, activation='relu')(x) # number of layers and units are hyperparameters, as
usual
x = Dense(500, activation='relu')(x)
predictions = Dense(13, activation='softmax')(x) # should match # of classes predicted

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])

model_mix = Model(inputs=base_model.input, outputs=predictions)
model_mix.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])

epochs = 10

history = model.fit(
   train_gen,
   epochs=epochs,
   verbose = 1,
   validation_data=test_gen
   )

model.save('/content/drive/My Drive/Zavrsni_Data/Modeli/model_VGG16_final.h5')


history_mix = model_mix.fit(
   train_gen_mix,
   epochs=epochs,
   verbose = 1,
   validation_data=test_gen_mix
   )

model_mix.save('/content/drive/My Drive/Zavrsni_Data/Modeli/model_mix_VGG16.h5')

import seaborn as sn
import matplotlib.pyplot as plt
import pandas as pd

plt.plot(history.history['categorical_accuracy'], 'ko')
plt.plot(history.history['val_categorical_accuracy'], 'b')

plt.title('Accuracy vs Training Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation']);

plt.plot(history_mix.history['categorical_accuracy'], 'yo')
plt.plot(history_mix.history['val_categorical_accuracy'], 'r')
```

```
plt.title('Accuracy vs Training Epoch mix')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation', 'Train_mix','Validation_mix']);

model = tf.keras.models.load_model('/content/drive/My
Drive/Zavrsni_Data/Modeli/model_VGG16_final.h5', compile=False)
model_mix = tf.keras.models.load_model('/content/drive/My
Drive/Zavrsni_Data/Modeli/model_mix_VGG16.h5', compile=False)

from sklearn.metrics import classification_report, confusion_matrix

target_names = ['BB', 'BK', 'BN', 'BP', 'BQ', 'BR', 'Empty', 'WB', 'WK', 'WN', 'WP', 'WQ',
'WR']

test_gen.reset()
test_gen_vert.reset()
test_gen_mix.reset()

Y_pred = model.predict_generator(test_gen)
Y_pred_vert = model.predict_generator(test_gen_vert)
Y_pred_mix = model.predict_generator(test_gen_mix)

Y_pred_mix_test = model_mix.predict_generator(test_gen)
Y_pred_mix_vert = model_mix.predict_generator(test_gen_vert)
Y_pred_mix_mix = model_mix.predict_generator(test_gen_mix)

classes = test_gen.classes[test_gen.index_array]
y_pred = np.argmax(Y_pred, axis= -1)
print(sum(y_pred==classes)/800)


data = confusion_matrix(classes, y_pred)
df_cm = pd.DataFrame(data, columns=target_names, index = target_names)
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (20,14))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16})# font size

classes = test_gen_vert.classes[test_gen_vert.index_array]
y_pred_vert = np.argmax(Y_pred_vert, axis= -1)
print(sum(y_pred_vert==classes)/800)

data_vert = confusion_matrix(classes, y_pred_vert)
df_cm = pd.DataFrame(data_vert, columns=target_names, index = target_names)
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (20,14))
sn.set(font_scale=1.4)#for label size
```

```
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16})# font size

classes = test_gen_mix.classes[test_gen_mix.index_array]
y_pred_mix = np.argmax(Y_pred_mix, axis= -1)
print(sum(y_pred_mix==classes)/800)

data_mix = confusion_matrix(classes, y_pred_mix)
df_cm = pd.DataFrame(data_mix, columns=target_names, index = target_names)
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (20,14))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16})# font size

classes = test_gen.classes[test_gen.index_array]
y_pred_mix_test = np.argmax(Y_pred_mix_test, axis= -1)
print(sum(y_pred_mix_test==classes)/800)

data_mix_test = confusion_matrix(classes, y_pred_mix_test)
df_cm = pd.DataFrame(data_mix_test, columns=target_names, index = target_names)
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (20,14))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16})# font size

classes = test_gen_vert.classes[test_gen_vert.index_array]
y_pred_mix_vert = np.argmax(Y_pred_mix_vert, axis= -1)
print(sum(y_pred_mix_vert==classes)/800)

data_mix_vert = confusion_matrix(classes, y_pred_mix_vert)
df_cm = pd.DataFrame(data_mix_vert, columns=target_names, index = target_names)
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (20,14))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16})# font size

classes = test_gen_mix.classes[test_gen_mix.index_array]
y_pred_mix_mix = np.argmax(Y_pred_mix_mix, axis= -1)
print(sum(y_pred_mix_mix==classes)/800)

data_mix_mix = confusion_matrix(classes, y_pred_mix_mix)
df_cm = pd.DataFrame(data_mix_mix, columns=target_names, index = target_names)
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (20,14))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16})# font size
```

```
print('Confusion Matrix')
print(data)
print('Classification Report')
print(classification_report(test_gen.classes[test_gen.index_array], y_pred,
target_names=target_names))

print('Confusion Matrix')
print(data_mix)
print('Classification Report')
print(classification_report(test_gen_mix.classes[test_gen_mix.index_array], y_pred_mix,
target_names=target_names))

print('Confusion Matrix')
print(data_mix_mix)
print('Classification Report')
print(classification_report(test_gen_mix.classes[test_gen_mix.index_array],
y_pred_mix_mix, target_names=target_names))

print('Confusion Matrix')
print(data_mix_vert)
print('Classification Report')
print(classification_report(test_gen_vert.classes[test_gen_vert.index_array],
y_pred_mix_vert, target_names=target_names))

print('Confusion Matrix')
print(data_mix)
print('Classification Report')
print(classification_report(test_gen.classes[test_gen.index_array], y_pred_mix,
target_names=target_names))
```

cv_chess_functions.py

```
import os
import math
import cv2
import numpy as np
import scipy.spatial as spatial
import scipy.cluster as cluster
from collections import defaultdict
from statistics import mean
import chess
import chess.svg
from svglib.svglib import svg2rlg
from reportlab.graphics import renderPM
from PIL import Image
import re
import glob
import PIL


# Ucitaj sliku, transformacija u grayscale i zamucivanje slike
def read_img(file):
    img = cv2.imread(str(file))
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray_blur = cv2.blur(gray, (5, 5))
    return img, gray_blur


# Canny edge detection
def canny_edge(img, sigma=0.33):
    v = np.median(img)
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    edges = cv2.Canny(img, lower, upper)
    return edges


# Hough line detection
def hough_line(edges, min_line_length=10, max_line_gap=150):
    lines = cv2.HoughLines(edges, 1, np.pi / 180, 125, min_line_length, max_line_gap)
    lines = np.reshape(lines, (-1, 2))
    return lines


# Odvajanje linija u horizontalne i vertikalne
def h_v_lines(lines):
    h_lines, v_lines = [], []
    for rho, theta in lines:
        if theta < np.pi / 4 or theta > np.pi - np.pi / 4:
            v_lines.append([rho, theta])
```

```
        else:
            h_lines.append([rho, theta])
    return h_lines, v_lines


# Pronalazak presjecista linija
def line_intersections(h_lines, v_lines):
    points = []
    for r_h, t_h in h_lines:
        for r_v, t_v in v_lines:
            a = np.array([[np.cos(t_h), np.sin(t_h)], [np.cos(t_v), np.sin(t_v)]])
            b = np.array([r_h, r_v])
            inter_point = np.linalg.solve(a, b)
            points.append(inter_point)
    return np.array(points)


# Hijerarhijski cluster (po euklidiskoj udaljenosti) presjecnih tocaka
def cluster_points(points):
    dists = spatial.distance.pdist(points)
    single_linkage = cluster.hierarchy.single(dists)
    flat_clusters = cluster.hierarchy.fcluster(single_linkage, 15, 'distance')
    cluster_dict = defaultdict(list)
    for i in range(len(flat_clusters)):
        cluster_dict[flat_clusters[i]].append(points[i])
    cluster_values = cluster_dict.values()
    clusters = map(lambda arr: (np.mean(np.array(arr)[:, 0]), np.mean(np.array(arr)[:, 1])),
cluster_values)
    return sorted(list(clusters), key=lambda k: [k[1], k[0]])



# Smanjenje suma tocaka njihovim spajanjem po kriteriju udaljenosti
def merge_points(points, max_dist):
    num_points = len(points)
    max_dist = max_dist*max_dist

    for i in range(0, len(points)-1):
        for j in range(i+1, len(points)):
            if ((points[i][0]-points[j][0])*(points[i][0]-points[j][0])<max_dist) and ((points[i][1]-
points[j][1])*(points[i][1]-points[j][1])<max_dist):
                points.remove(points[j])

    return points

# Crop ploce u segmente i spremanje segmenata u zasebne slike i Data/raw_data/ folderu
def write_crop_images(img, points, img_count=0, folder_path='./Data/raw_data/'):

    #vidi dal ima 81 toc, ako nema slikaj opet, odaberi toc, nadi 3 najblize, nadi sredinu i uzmi
kvadrat okolo
```

```
    dist = 0
    pnts=points.copy()
    dist_list = []
    index_list = []
    lista_najbliz = []
    if len(points) <81:
        print("Nisi uhvatio cijelu plocu, slikaj opet")
    else:
        for i in range(0, len(pnts)-9):
            dist_list = []
            index_list = []
            for j in range(i+1, len(pnts)):
                dist = (pnts[i][0]-pnts[j][0])*(pnts[i][0]-pnts[j][0])+(pnts[i][1]-
pnts[j][1])*(pnts[i][1]-pnts[j][1])
                dist_list.append(dist)
            for n in range(0,3):
                index = points[dist_list.index(np.min(dist_list))+i]
                index_list.append(index)
                dist_list[dist_list.index(np.min(dist_list))]+=100000
            lista_najbliz.append(index_list) #lista 3 najbliz tocke za svaku tocku osim zadnjih 9
            pnts[i]+=(100000,100000)


        for i in range(0, len(lista_najbliz)-8):
            x_min = 10000
            x_max = 0
            y_min = 10000
            y_max = 0
            tocke = []
            tocke.append(points[i])
            for j in range(0, 3):
                tocke.append(lista_najbliz[i][j])
            for n in range(0,len(tocke)):
                if tocke[n][0]>x_max:
                    x_max = tocke[n][0]
                if tocke[n][0]<x_min:
                    x_min = tocke[n][0]
                if tocke[n][1]>y_max:
                    y_max = tocke[n][1]
                if tocke[n][1]<y_min:
                    y_min = tocke[n][1]
            cropped = img[int(y_min): int(y_max), int(x_min): int(x_max)]
            img_count += 1
            cv2.imwrite('./Data/raw_data/data_image' + str(img_count) + '.jpeg', cropped)

    return img_count


# Konverzija slike iz RGB u BGR
def convert_image_to_bgr_numpy_array(image_path, size=(224, 224)):
```

```
    image = PIL.Image.open(image_path).resize(size)
    img_data = np.array(image.getdata(), np.float32).reshape(*size, -1)
    # swap R and B channels
    img_data = np.flip(img_data, axis=2)
    return img_data


# Promjena dimenzija slika za neuronsku mrezu dimenzijama (1, 224, 224, 3)
def prepare_image(image_path):
    im = convert_image_to_bgr_numpy_array(image_path)

    im[:, :, 0] -= 103.939
    im[:, :, 1] -= 116.779
    im[:, :, 2] -= 123.68

    im = np.expand_dims(im, axis=0)
    return im


# Promjena znamenki u tekstu u intigere
def atoi(text):
    return int(text) if text.isdigit() else text


# Pronalazak znamenki u tekst stringu
def natural_keys(text):
    return [atoi(c) for c in re.split('(\d+)', text)]


# Ucitavanje crop-anih slika u listu za obradu
def grab_cell_files(folder_name='./Data/raw_data/*'):
    img_filename_list = []
    for path_name in glob.glob(folder_name):
        img_filename_list.append(path_name)
    return img_filename_list


# Klasifikacija svakog segmenta i spremanje klasifikacije u Forsyth-Edwards Notaciju (FEN)
def classify_cells(model, img_filename_list):
    category_reference = {0: 'b', 1: 'k', 2: 'n', 3: 'p', 4: 'q', 5: 'r', 6: '1', 7: 'B', 8: 'K', 9: 'N', 10: 'P',
                          11: 'Q', 12: 'R'}
    pred_list = []
    for filename in img_filename_list:
        img = prepare_image(filename)
        out = model.predict(img)
        top_pred = np.argmax(out)
        pred = category_reference[top_pred]
        pred_list.append(pred)

    fen = ''.join(pred_list)
```

```
    fen = fen[::-1]
    fen = '/'.join(fen[i:i + 8] for i in range(0, len(fen), 8))
    sum_digits = 0
    for i, p in enumerate(fen):
        if p.isdigit():
            sum_digits += 1
        elif p.isdigit() is False and (fen[i - 1].isdigit() or i == len(fen)):
            fen = fen[:(i - sum_digits)] + str(sum_digits) + ('D' * (sum_digits - 1)) + fen[i:]
            sum_digits = 0
    if sum_digits > 1:
        fen = fen[:(len(fen) - sum_digits)] + str(sum_digits) + ('D' * (sum_digits - 1))
    fen = fen.replace('D', '')
    return fen


# Konverzija FEN prikaza u PNG file
def fen_to_image(fen):
    board = chess.Board(fen)
    current_board = chess.svg.board(board=board)

    output_file = open('current_board.svg', "w")
    output_file.write(current_board)
    output_file.close()

    svg = svg2rlg('current_board.svg')
    renderPM.drawToFile(svg, 'current_board.png', fmt="PNG")
    return board
```

cv_chess.py

```
import os
import glob
import re
import cv2
import tensorflow as tf
from keras.models import load_model
from cv_chess_functions import (read_img,
                                canny_edge,
                                hough_line,
                                h_v_lines,
                                line_intersections,
                                cluster_points,
                                write_crop_images,
                                grab_cell_files,
                                classify_cells,
                                fen_to_image,
                                atoi,
                                merge_points)
import chess
from fentoimage.board import BoardImage
from PIL import Image

# Resize the frame by scale by dimensions
def rescale_frame(frame, percent=75):
    # width = int(frame.shape[1] * (percent / 100))
    # height = int(frame.shape[0] * (percent / 100))
    dim = (1000, 750)
    return cv2.resize(frame, dim, interpolation=cv2.INTER_AREA)


# Find the number(s) in the text
def natural_keys(text):
    return [atoi(c) for c in re.split('(\d+)', text)]


# ucitavanje naucenog modela
model = tf.keras.models.load_model('./nauceni_model_VGG16_3.h5', compile=False)

# Odabir izvora video prijenosa (0-ugradena & 1-vanjska)
cap = cv2.VideoCapture(0)

# Prikaz ploce na fen prikazu
# start = 'rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR'
blank = '8/8/8/8/8/8/8/8'
board = BoardImage(blank)
current_board = board.render()
current_board.save("current_board.png")
```

```
board_image = cv2.imread( 'current_board.png' )
cv2.imshow('current board', board_image)

while(True):
   # Snimanje frame-by-frame
   ret, frame = cap.read()

   # Promjena velicine svakog framea
   small_frame = rescale_frame(frame)

   # Prikaz prijenosa uzivo
   cv2.imshow('live', small_frame)

   if cv2.waitKey(1) & 0xFF == ord(' '):

      print('Working...')

         # Ciscenje radne direktorije od prethodnih slika
      files = glob.glob('./Data/raw_data/*')
      for f in files:
         os.remove(f)

      try:
         # Spremi sliku za analizu
         cv2.imwrite('frame.jpeg', frame)
         # Low-level CV techniques (grayscale & blur)
         img, gray_blur = read_img('frame.jpeg')

         # Canny algoritam
         edges = canny_edge(img)
         cv2.imshow('edges', edges)
         # Hough Transformacija
         lines = hough_line(edges)

         # Odvajanje linija u vertikalne i horizontalne
         h_lines, v_lines = h_v_lines(lines)

         # Pronalazenje, clusteranje i ciscenje krizanja hor i vert linija
         intersection_points = line_intersections(h_lines, v_lines)

         points = cluster_points(intersection_points)

         # Finalne koordinate prociscenih tocaka na ploci

         points = merge_points(points, 25)

         # Crop kocaka na ploci i stvaranje sortirane liste
         x_list = write_crop_images(img, points, 0)
         img_filename_list = grab_cell_files()
         img_filename_list.sort(key=natural_keys)
```

```
            # Klasificiranje cropanih slika i zapis u Forsyth-Edwards Notaciju (FEN)
            fen = classify_cells(model, img_filename_list)
            # Stvori i snimi dobivenu sliku u FEN prikazu
            board = BoardImage(fen)
            # Prikazi plocu u ASCII formatu
            print(board)
            # Prikazi i spremi sliku trenutne ploce
            current_board = board.render()
            current_board.save("current_board.png")
            board_image = cv2.imread('current_board.png')
            cv2.imshow('current board', board_image)
        except:
            print("Slikaj opet")
        print('Completed!')

    if cv2.waitKey(1) & 0xFF == ord('q'):
        # End the program
        break

# Izlazak iz petlje, gasenje prozora
cap.release()
cv2.destroyAllWindows()
```