

Automatska detekcija i raspoznavanje registarskih oznaka vozila korištenjem neuronskih mreža

Merlić, Nikola

Master's thesis / Diplomski rad

2025

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:137:198438>

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

Download date / Datum preuzimanja: **2025-03-21**



Repository / Repozitorij:

[Digital Repository Juraj Dobrila University of Pula](#)



Sveučilište Jurja Dobrile u Puli
Tehnički fakultet



NIKOLA MERLIĆ

**AUTOMATSKA DETEKCIJA I RASPOZNAVANJE REGISTARSKIH OZNAKA
VOZILA KORIŠTENJEM NEURONSKIH MREŽA**

Diplomski rad

Pula, ožujak 2025. godine

Sveučilište Jurja Dobrile u Puli
Tehnički fakultet

NIKOLA MERLIĆ

**AUTOMATSKA DETEKCIJA I RASPOZNAVANJE REGISTARSKIH OZNAKA
VOZILA KORIŠTENJEM NEURONSKIH MREŽA**

Diplomski rad

JMBAG: 0303090413, redoviti student

Studijski smjer: Računarstvo

Kolegij: Neuronske mreže i genetski algoritmi

Znanstveno područje: Tehničke znanosti

Znanstveno polje: Računarstvo

Znanstvena grana: Obrada informacija

Mentor: doc. dr. sc. Nikola Lopac

Komentor: izv. prof. dr. sc. Nicoletta Saulig

Pula, ožujak 2025. godine

Sadržaj

1. Uvod.....	1
2. Tehnike i razvoj prepoznavanja registarskih tablica.....	2
2.1. Pregled općih pristupa.....	2
2.2. Digitalna obrada slike.....	3
2.3. Povijesni razvoj.....	6
3. Arhitekture neuronskih mreža za ALPR.....	8
3.1. Konvolucijske neuronske mreže (CNN).....	8
3.2. Rekurentne neuronske mreže (RNN).....	15
3.3. Kombinirane arhitekture.....	17
4. Implementacija i analiza algoritama za detekciju i raspoznavanje registarskih oznaka.....	18
4.1. Izazovi u detekciji i raspoznavanju registarskih oznaka.....	18
4.2. Korišteni skup podataka i postupci predobrade.....	21
4.3. Implementacija YOLO algoritma za detekciju registarskih oznaka.....	21
4.4. Implementacija Faster R-CNN modela za detekciju registarskih oznaka.....	61
4.5. Implementacija Single Shot Multibox Detector (SSD) algoritma.....	82
4.6. Razvoj i evaluacija vlastite neuronske mreže.....	97
5. Usporedba rezultata korištenih algoritama.....	117
6. Zaključak.....	120
7. Literatura.....	121
8. Popis slika.....	123
9. Popis tablica.....	126
Sažetak.....	127
Abstract.....	129

1. Uvod

Automatska detekcija i raspoznavanje registarskih oznaka vozila (*Automatic Number Plate Recognition*, ANPR) je ključna komponenta velikog broja raznih sustava koji se koriste za nadzor prometa i sigurnost. Navedeni sustavi rasprostranjeni su širom svijeta te se koriste za različite primjene, kao što su upravljanje parkiralištima, naplaćivanje cestarina, kontrola brzine, te sve češću primjenu nalaze u privatnom sektoru kod kontrole pristupa zaposlenika i evidenciju posjeta. Sve veću primjenu ove tehnologije prepisuje se brzom tehnološkom napretku i širokoj potražnji u državnom i privatnom sektoru. Prepoznavanje registarskih tablica suočava se s brojnim izazovima kao što su: različiti kutovi snimanja kamera, diferencijacije u osvjetljenju, različite boje i vrste automobilskih tablica, prisutnosti šumova na slici koje mogu stvarati kamere ovisno o njihovoj poziciji i kvaliteti te nepogodni vremenski uvjeti kao što su kiša, snijeg i magla. Navedeni izazovi uvelike utječu na preciznost prepoznavanja te točnost izvedbe korištenih algoritama. Algoritmi za prepoznavanje registarskih tablica vozila generalno se mogu podijeliti u četiri koraka: detekcija vozila, detekcija registarske tablice, segmentacija simbola i prepoznavanje simbola tablice. Najveću uspješnost u navedenom segmentu pokazalo se korištenjem dubokog učenja i neuronskih mreža koje dominiraju u ovom području zbog svojih samoučećih sposobnosti. Najčešće korištene arhitekture u ovome području su konvolucijske neuronske mreže (CNN), rekurentne neuronske mreže (RNN) i kombinacije dviju arhitektura. Cilj ovog rada je istražiti učinkovitost nekoliko „*state – of – the – art*“ pristupa u različitim scenarijima koji se koriste pri prepoznavanju registarskih tablica te programski razviti vlastiti model neuronske mreže za istu primjenu i naposljetku usporediti rezultate vlastitog modela s prije spomenutim modelima koji se koriste u svijetu.

2. Tehnike i razvoj prepoznavanja registarskih tablica

Razvoj tehnika za prepoznavanje registarskih tablica kroz godine pratila su značajna unapređenja u području računalnog vida i umjetne inteligencije. U ovom poglavlju prikazat će se uspješni pristupi i metode koji se primjenjuju pri izradi automatskih sustava za detekciju i prepoznavanje registarskih tablica vozila.

2.1. Pregled općih pristupa

Prepoznavanje registarskih tablica vozila (ALPR) obuhvaća više različitih tehnika koje uključuju obradu slike, detekciju objekata i prepoznavanje znakova. Navedeni sustavi razvijaju se već godinama, od samih početaka računalnoga vida pa sve do danas gdje za najbolji rezultat koriste neuronske mreže. ALPR sustavi sastoje se od nekolicine ključnih koraka:

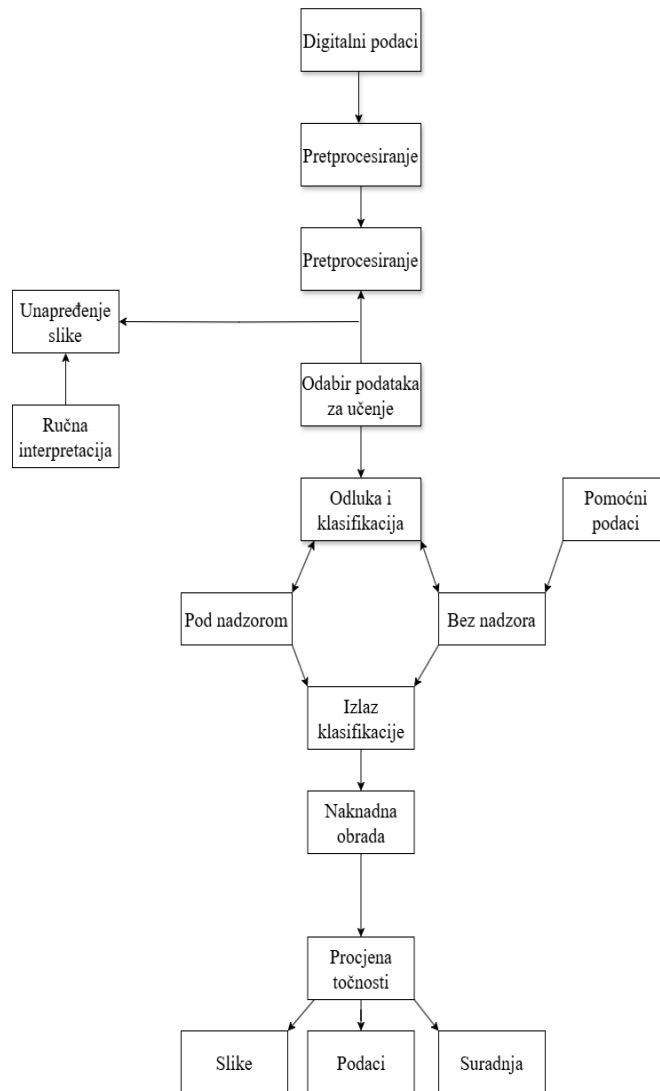
- **Obrada slike:** Prvi korak u cijelome procesu je obrada slike koja služi za poboljšanje kvalitete slike. Navedeni korak uključuje tehnologije filtriranja, povećavanja kontrasta i segmentaciju slike. Glavni cilj je izdvajanje registarske tablice vozila iz pozadine te maksimalno moguće smanjenje šuma sa slike. Neke od tehnika korištenih u ovom koraku su Gaussovo filtriranje, *Canny edge* detekcija i binarizacija.
- **Detekcija registarskih tablica:** Drugi korak u procesu je detekcija tablice, to jest njena precizna identifikacija. U starijim, više tradicionalnim pristupima pretežito su korištene Houghove transformacije za detekciju linija i pravokutnika. Houghova transformacija je tehnika procjenjivanja parametara koja se temelji na načelu glasanja, svaki element slike (piksel) glasa za parametre svih oblika gdje bi točno mogao pripadati. Navedeni glasovi se naknadno zbrajaju u akumulacijskom polju, te traženi oblici se pronalaze kao maksimumi navedenog polja. Danas se koriste tehnologije strojnog učenja kao što su algoritmi Haarove kaskade i *Support*

Vector Machines (SVM), te ostale tehnike dubokog učenja koje sadržavaju konvolucijske neuronske mreže.

- **Prepoznavanje znakova:** Posljednji korak u procesu je prepoznavanje znakova na registarskoj oznaci automobila. Problemu se pristupa implementiranjem tehnika OCR-a, to jest optičkog prepoznavanja znakova koji omogućuje automatsko čitanje i interpretaciju znakova s registarske tablice automobila. Prvi korak je segmentacija znakova, gdje se odvajaju svi znakovi na tablici zasebno, kako bi algoritam mogao pojedinačno prepoznati svaki znak sa što većom sigurnošću. Nakon uspješno izvršene segmentacije potrebno je prepoznati svaki znak na registarskoj tablici, navedeni korak se izvršava pomoću raznih algoritama dubokog i strojnog učenja. (Gonzalez, Rafael C. and Woods, Richard E. 2018).

2.2. Digitalna obrada slike

Digitalna slika se matematički definira kao dvodimenzionalna diskretna funkcija: $\mathcal{F}(x, y)$ gdje su x i y prostorne koordinate, a vrijednost funkcije predstavlja intenzitet ili razinu sive na određenoj točki u slici. Kada obje koordinate i vrijednost intenziteta funkcije sadržavaju konačne i diskretne veličine, sliku nazivamo digitalnom. Digitalna slika se sastoji od određenog (konačnog) broja elemenata, od kojih svaki element ima svoju poziciju i vrijednost na digitalnoj slici. Piksel je termin koji se najčešće koristi za označavanje elemenata digitalne slike. (Gonzalez, Rafael C. and Woods, Richard E., 2018. Digital Image Processing, Global Edition. United Kingdom: Pearson Education).



Slika 1. Tijek rada za obradu i klasifikaciju digitalnih podataka

izvor: Autor modificirao prema (Boyat & Joshi. 2015)

S obzirom na analognu obradu slike, digitalna obrada dopušta puno veću primjenu različitih algoritama koji se mogu primijeniti na ulazne podatke te uvelike smanjuje problem šuma na slici i distorzije tijekom obrade. Kod digitalne obrade slike ključno je dobro i kvalitetno rješavanje koraka kao što su filtriranje i segmentacija slike te detekcija rubova.

Filtriranje slike je vrlo značajan korak u predobradi podataka, cilj navedenog koraka je što bolje uklanjanje šuma i poboljšanje kvalitete slike. Ovisno o upotrebi i specifičnim zahtjevima, koriste se različite vrste filtara. Dva jednostavna i često primjenjivana filtra u praksi su Gaussovo filtriranje i Median filtriranje. Gaussov filter je niskopropusni filter kojemu je osnovni cilj smirivanje slike i uklanjanje šuma uz minimalan gubitak detalja. S druge strane, Median filter, koji je nelinearan, smanjuje impulsni šum bez zamućivanja rubova slike.

Segmentacija slike je proces gdje se slika dijeli na određene segmente kako bi olakšali daljnju analizu. Cilj ovog procesa je izdvajanje bitnih objekata iz pozadine. Postoje različite metode segmentacije slika, a neke od popularnijih jesu: pragovanje, *region – based* segmentacija i *edge – based* segmentacija.

Region – based segmentacija grupira piksele na slici po sličnim svojstvima poput grupacija po intenzitetu boje. Pragovanje (engl. *Thresholding*) objektivno najjednostavnija metoda koja se koristi pri binarizaciji slike. Pragovanje razdvaja objekte pomoću intenziteta piksela. *Edge – based* koristi informacije o rubovima slike pomoću kojih segmentira objekt u slici.

Detekcija rubova je fundamentalna tehnika u digitalnoj obradi slika koja služi za identificiranje te lociranje rubova određenog objekta na slici. Rubovi objekata slike su mjesta gdje se intenzitet svjetlosti određenog objekta mijenja naglo, što određuje granicu. Razlikujemo više različitih vrsta tehnika koje se koriste pri detekciji rubova:

- *Sobel Edge Detection*
- *Canny Edge Detection*
- *Scharr Edge Detection*
- *Roberts Cross Edge Detection*
- *Prewitt Edge Detection*
- *Palplacin Edge Detection*



Slika 2. Detekcija registarske tablice automobila

Izvor: Izradio autor

Cilj algoritama za detekciju rubova jest identificirati rubove unutar slike te ih povezati na način da se formira smislen prikaz granica objekata, što rezultira segmentiranom slikom koja sadrži jedno ili više područja. Detekcija rubova primjenjuje se u različitim scenarijima računalnog vida, a posebno je korisna u prepoznavanju uzoraka na registarskim tablicama vozila. (Gonzalez, Rafael C. and Woods, Richard E., 2018. Digital Image Processing, Global Edition. United Kingdom: Pearson Education.)

2.3. Povijesni razvoj

Povijest sustava za automatsko prepoznavanje registarskih pločica (ANPR) seže u 1976. godinu u Ujedinjenom Kraljevstvu, gdje je razvijen u sklopu Policijskog razvojnog ogranka (PSDB). Prvi praktični sustavi bili su raspoređeni duž autoceste A1 prema Dartfordovom tunelu, a prvo uspješno uhićenje dogodilo se 1981. godine kada je sustav prepoznao ukradeno vozilo i proslijedio informaciju policiji. Navedeni sustavi koristili su osnove tehnike obrade slika, uključujući detekciju rubova i binarizaciju, ali je njihova točnost ovisila o uvjetima osvjetljenja i tehničkim ograničenjima poput razlikovanja objekata u pozadini. Tijekom 1980-ih i 1990-ih godina tehnologija je napredovala, što je rezultiralo uvođenjem novih algoritama i tehnika strojnog učenja koje uvelike

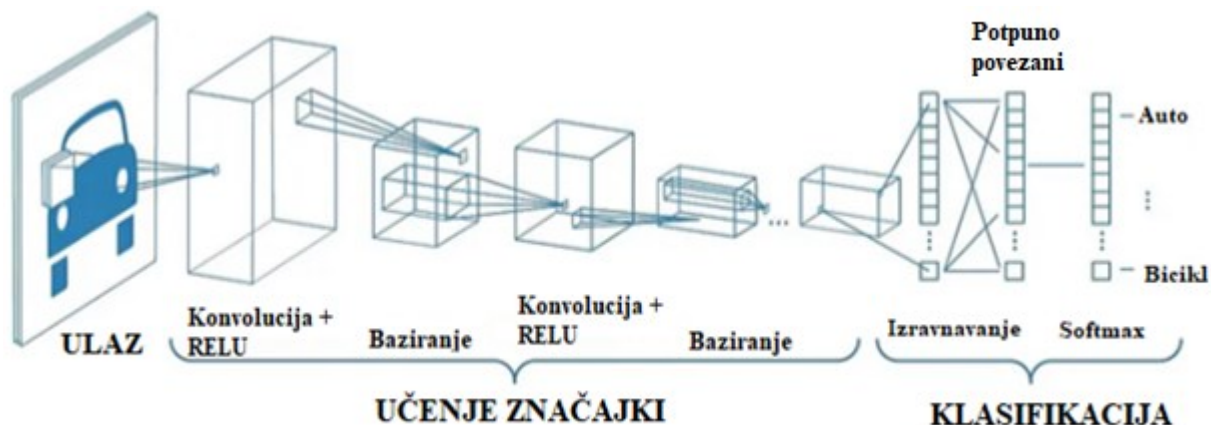
poboljšavaju robusnost i točnost navedenih sustava. Tehnike poput analiza povezanih komponenti i višeslojnog perceptrona omogućuju bolje prepoznavanje znakova. U navedenim godinama policija razvija novu jedinicu koja umjesto dosadašnjeg lokaliziranog skupljanja podataka širi svoje granice te prikuplja informacije iz svih regija države što omogućuje rad s više podataka te ujedno povećava točnost sustava. Početkom 2000-ih godina dolazi do razvoja strojnog učenja koje postaju široko prihvaćene u području analize podataka. Razvijaju se novi algoritmi koji su temeljeni na metodama kao što su *support vector machines* (SVM) i kNN (*k-nearest neighbors*) koji omogućuju automatsko učenje iz podataka. Navedeni algoritmi nisu novitet nastali u tom razdoblju, algoritam potpornih vektora temelji se na statističkim okvirima koje su razvili istraživači poput Vapnika još u drugoj polovici 20. stoljeća, dok je algoritam najbližih susjeda osmišljen još 1951. godine. U kontekstu primjene tih metoda početkom 21. stoljeća, ovi algoritmi značajno doprinose obradi i analizi podataka zahvaljujući povećanoj dostupnosti računalnih resursa i sve većim skupovima podataka. Ranih 2000-ih godina London postavlja gotovo 700 kamera kojima je primarna zadaća bila smanjivanje kongestivnog prometa koji se kolao najprometnijim ulicama. Početkom 2010-ih godina sve više se spominje pojam dubokog učenja i konvolucijskih neuronskih mreža (CNN) kojima je najveća prednost bila kako uz automatsko učenje značajki nije bilo potrebe za ručnim dizajniranjem što je uvelike smanjilo potrebu za ljudskim upletanjem u repetitivne zadatke. U ranim godinama korištenja tehnologije pristup velikim skupovima podataka bilo je ograničeno, većinom su to bili mali homogeni skupovi koji pri primjeni u realnom okruženju su imali mnogo problema s preciznošću i generalizacijom. Sve daljnjim prolaskom vremena te ulazom u eru velikih podataka, skupovi postaju sve veći i raznolikiji. Ovi skupovi podataka sadrže tisuće, pa čak i milijune slika koje su snimljene u različitim vremenskim uvjetima te pod različitim kutevima što ultimativno omogućuje da algoritam nauči što širi spektar varijacija te naposljetku ima što je moguće veću točnost. (ANPR International, 2023.)

3. Arhitekture neuronskih mreža za ALPR

Arhitekture neuronskih mreža igraju ključnu ulogu kod automatskog prepoznavanja registarskih tablica vozila. Postoje puno vrsta različitih arhitektura kod kojih je svaka posebna i specifična za različite namjene, ovisno o parametrima kao što su brzina, točnost i složenost pojedinog zadatka. U navedenim poglavljima analizirat će se tri ključne arhitekture koje se pretežito primjenjuju u sustavima za prepoznavanje registarskih tablica vozila, a to su: konvolucijske neuronske mreže (CNN), rekurentne neuronske mreže (RNN) i kombinirane arhitekture koje spajaju prethodno nabrojene neuronske mreže. (LeCun et al., 2015)

3.1. Konvolucijske neuronske mreže (CNN)

Konvolucijska neuronska mreža je vrsta neuronske mreže koja je specifična po tome što sama uči značajke putem optimizacije filtra. Navedena mreža sastoji se od ulaznog sloja, nevidljivog sloja i izlaznog sloja. Inspiracija navedene mreže dolazi s biološke strane, preciznije, sastav neurona unutar mreže vrlo je sličan organizaciji neurona životinjskog vizualnog korteksa. (LeCun et al., 2015)



Slika 3. Arhitektura konvolucijske neuronske mreže

Izvor: Autor modificirao prema (Towards Data Science, 2020)

Konvolucijska neuronska mreža sastoji se od ključnih komponenti kao što su ulazni sloj, konvolucijski sloj, aktivacijske funkcije, *pooling* slojeva, izravnavanja (engl. *flattening*) i potpuno povezanih slojeva. Navedene komponente omogućuju mreži prilagodbu podataka te prepoznavanje karakterističnih značajki potrebnih za zadatak, poput prepoznavanja registarskih oznaka vozila. U procesu prilagodbe, mreža prolazi kroz više slojeva koji se međusobno nadovezuju, s ciljem bolje interpretacije i prepoznavanja složenih uzoraka na slikama. Konvolucijski slojevi i ostale komponente međusobno se nadovezuju tijekom procesa obrade podataka, čime se omogućuje postupno prepoznavanje složenih uzoraka na slikama. U kontekstu treniranja mreže, ove komponente surađuju kako bi optimizirale prepoznavanje uzoraka putem iterativnog postupka prilagodbe, dok se faza evaluacije koristi za procjenu uspješnosti naučenog modela na nepoznatim podacima.

Kod konvolucijske neuronske mreže, u ulaznome sloju pretežito se nalaze slike koje se predstavljaju kao trodimenzionalni nizovi piksela. Slika može biti u različitim oblicima kao što je: *grayscale*, RGB, HSV i CMYK. Ako konvolucijska neuronska mreža prima sliku koja ima parametre 32 x 32 piksela s RGB kanalom boja (crvena, zelena, plava) predstavlja se kao 32 x 32 x 3 niz. Naravno, veličina same slike to jest njezina

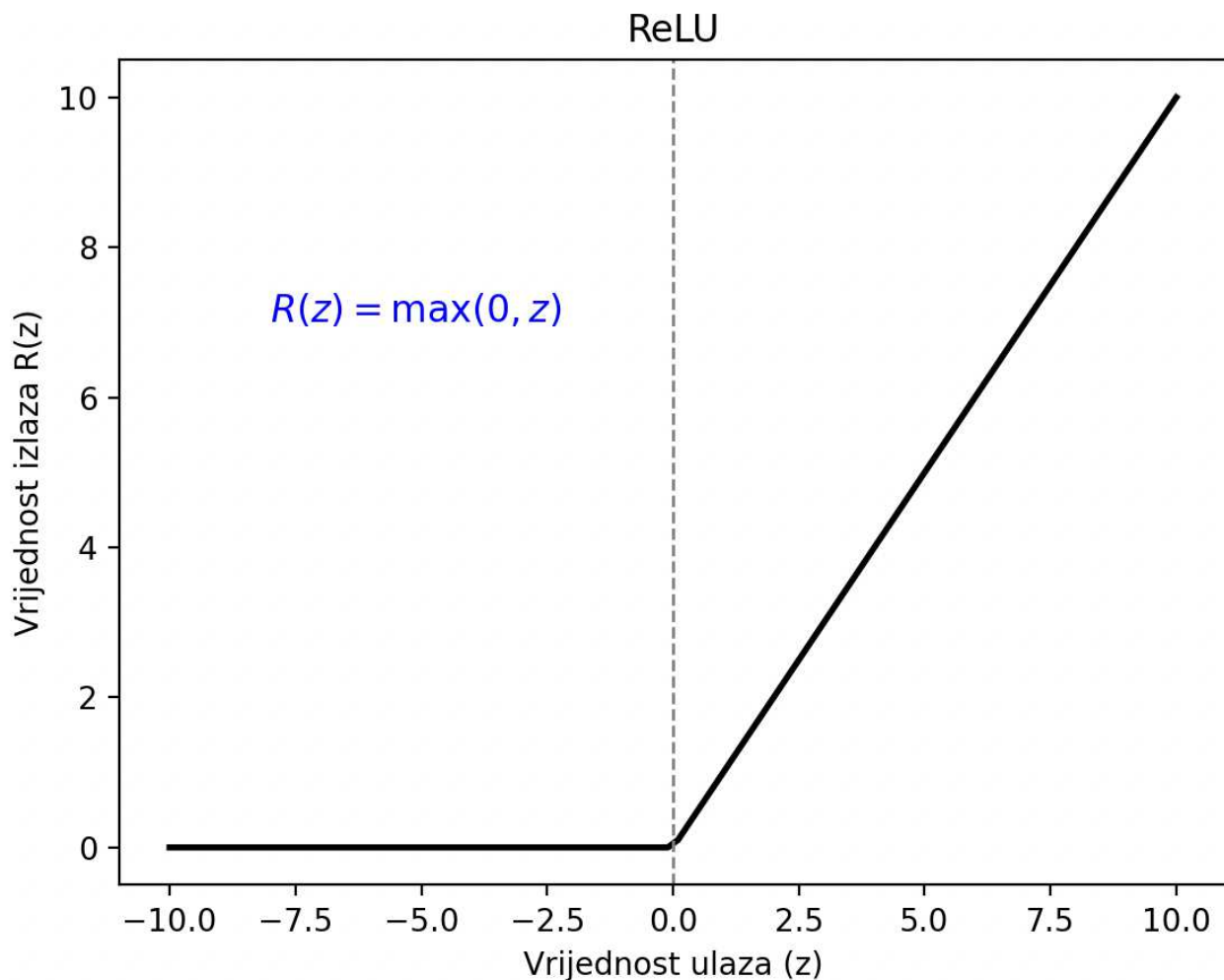
dimenzionalnost igra veliku ulogu u samoj obradi slike, veće dimenzije slike zahtijevaju veću računalnu moć.

Nakon ulaznog sloja, nalazi se konvolucijski sloj koji služi za izdvajanje određenih značajki iz ulaznih slika. Bit navedenog sloja je da prepozna specifična strukture unutar slike kao što su rubovi i ostali oblici. Navedeni sloj koristi filtre (engl. *kernel*) koje u matematičkom obliku mogu biti prepoznate kao matrice određenih dimenzija (ovisno o ulaznoj slici). Konvolucija je matematička operacija koja omogućuje lokalnu analizu podataka. Za dvodimenzionalnu sliku $I(x, y)$ i kernel $K(u, v)$, operacija konvolucije može se definirati kao:

$$O(x, y) = \sum_{u=-m}^m \sum_{v=-n}^n I(x + u, y + v) * K(u, v) \quad (1)$$

Gdje se izlaz $O(x, y)$ računa kao skalarni produkt između lokalne regije slike i kernela. Ova operacija omogućuje mreži da identificira lokalne značajke slike, poput rubova, tekstura i oblika. (Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.). Dimenzije kernela određuju koliko piksela će se obrađivati u zadanome periodu. Dubina i broj filtera biti će određeni kompleksnošću slike, to jest njenoj veličini, te broju kanala koju sadržava. Pri svakom pomaku filtra računa se skalarni umnožak ili *dot-product* što je zbroj svih produkata između odgovarajućih elemenata filtera i slike. Iskorak (engl. *stride*) je druga vrlo bitna komponenta procesa koja definira korak pomicanja filtera preko slike, veličina pomaka, što je iskorak veći to je pomak to jest preskok između piksela proporcionalno veći. Veličina iskoraka uvelike utječe na detaljnost izlazne mape značajki; manji iskorak, to jest manji preskok piksela rezultira većom detaljnošću slike, no zahtjeva i veću računalnu složenost. Prije primjene aktivacijske funkcije, potrebno je implementirati tehniku *padding* koja dodaje dodatne piksele oko rubova kako bi nakon konvolucije dimenzija ostala ista. Nakon *padding-a* slijedi primjena aktivacijske funkcije, najčešće ReLU. ReLU funkcionira na način ako je ulazna vrijednost pozitivan broj, navedena funkcija vraća ulaznu vrijednost, a ako je ulazna vrijednost negativna ili nula, funkcija vraća nulu. ReLU funkcija je specifična zbog svoje računske jednostavnosti za razliku od sigmoidnog ili hiperboličkog tangensa,

uvodi nelinearnost u neuronsku mrežu i ublažava problem nestajanja gradijenta. (Qadri and Asif, 2009)

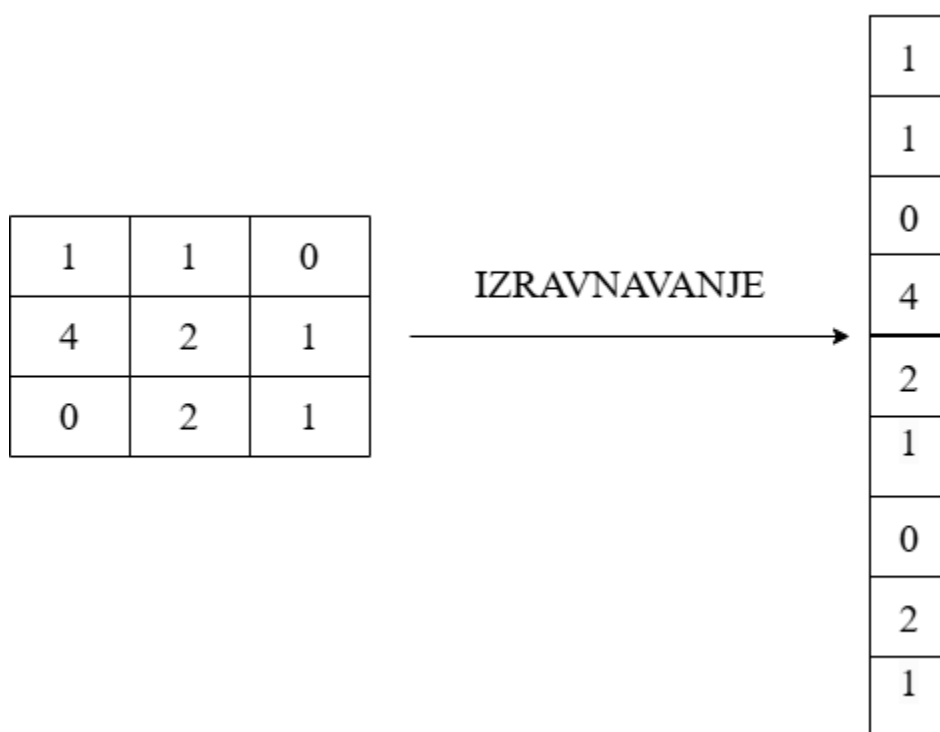


Slika 4. Prikaz ReLU funkcije na grafu

Izvor: Izradio autor

Nakon konvolucijskih slojeva slijedi *pooling* sloj koji je ključni dio neuronske mreže (CNN) i ima primarnu zadaću smanjiti dimenzionalnost značajki dobivenih iz konvolucijskih slojeva. Navedena stavka se postiže zadržavanjem najvažnijih informacija dok se redundantni podaci uklanjaju. *Pooling* slojevi olakšavaju računalnu obradu i smanjuju rizik od prekomjernog prilagođavanja (engl. *overfitting*). Razlikuje se više vrsta *pooling* slojeva: *max pooling*, *global pooling*, *average pooling*, *stochastic*

pooling, *Lp pooling*. Najčešće korištena metoda u praksi je *max pooling* koja radi tako da unutar definirane regije uzima maksimalnu vrijednost te zadržava samo najistaknutiju značajku iz svake pojedine regije. Sljedeći korak je izravnavanje, izravnavanje je prilično jednostavan korak kojemu je cilj pretvoriti višedimenzionalnu strukturu u jedinstveni vektor. Prije izravnavanja, značajke su prikazane u trodimenzionalnom nizu, nakon korištenja spomenute tehnike navedeni niz pretvara se u jedinstveni vektor koji služi kao ulaz u daljnju obradu unutar konvolucijske neuronske mreže.



Slika 5. Prikaz izravnavanja

Izvor: Autor modificirao prema (SuperDataScience, 2018)

Nakon prije spomenutog sloja slijede potpuno povezani slojevi (engl. *fully connected layers*) koji se odnose na „neuronsku mrežu u kojoj svaki neuron primjenjuje linearnu transformaciju na ulazni vektor kroz matricu težinskih vrijednosti.“ Kao rezultat toga, prisutne su sve moguće veze, što znači da svaki ulaz ulaznog vektora utječe na svaki

izlaz izlaznog vektora. (Built In. (2024). Fully Connected Layer in Machine Learning. *Built In*. Dostupno na: <https://builtin.com/machine-learning/fully-connected-layer>).

U navedenom koraku poželjno je korištenje aktivacijskih funkcija zbog unošenja linearnosti. Sljedeći korak je implementacija *softmax* funkcije koja služi za pretvaranje logita posljednjeg sloja mreže u distribucije vjerojatnosti, osiguravajući da izlazne vrijednosti predstavljaju normalizirane vjerojatnosti klase, što ga čini prikladnim za zadatke klasifikacije s više klasa. (GeeksforGeeks. (2023). Why Should SoftMax Be Used in CNN?. *GeeksforGeeks*. Dostupno na: <https://www.geeksforgeeks.org/why-should-softmax-be-used-in-cnn/>).

Softmax funkcija je specifična po svojoj distribuciji vjerojatnosti gdje navedeni logit predstavlja neobrađeni izlaz modela za svaku klasu prije normalizacije. Eksponencijalna transformacija normalizira vrijednosti tako da budu unutar raspona od 0 do 1, čineći ih razumljivima kao vjerojatnosti pripadanja pojedinoj klasi. Ova funkcija je pogodna pri problemima višeklasne klasifikacije, jer pruža jasnu i normaliziranu distribuciju vjerojatnosti za sve klase. Klasa s najvećom vjerojatnošću određuje se kao predviđena klasa, čime se pojednostavljuje interpretacija izlaza konvolucijske neuronske mreže i jasno prikazuje kojem ulazu predikcija pripada. Poželjno je kombinirati *softmax* funkciju u kombinaciji s funkcijom gubitka unakrsne entropije (engl. *cross-entropy loss*) koja mjeri nesličnost između predviđene distribucije vjerojatnosti i stvarne distribucije klasa.

Treniranje konvolucijske neuronske mreže bitan je korak u kojem se optimiziraju težinske vrijednosti mreže kako bi se minimizirala pogreška u predikcijama. Navedeni proces je iterativan, što omogućuje navedenoj mreži da uči iz danih podataka i kroz svaku iteraciju poboljšava svoje sposobnosti prepoznavanja i klasifikacije objekata na slikama. Najbitniji koraci kod treniranja su: propagacija naprijed, propagacija unatrag, izračun pogreške i optimizacija težinskih vrijednosti. Prvi proces koji se izvršava je propagacija naprijed gdje ulazna slika prolazi kroz sve prije spomenute slojeve i funkcije koje primjenjuju svoje operacije nad njome te prosljeđuju informaciju dalje, nakon ovog koraka generira se funkcija gubitka koja mjeri kolika je točnost predikcije, što je veća vrijednost gubitka, gora će predikcija biti. Slijedi propagacija unazad u kojem se dobivena vrijednost funkcije gubitka unatrag kroz mrežu, te se računaju gradijenti

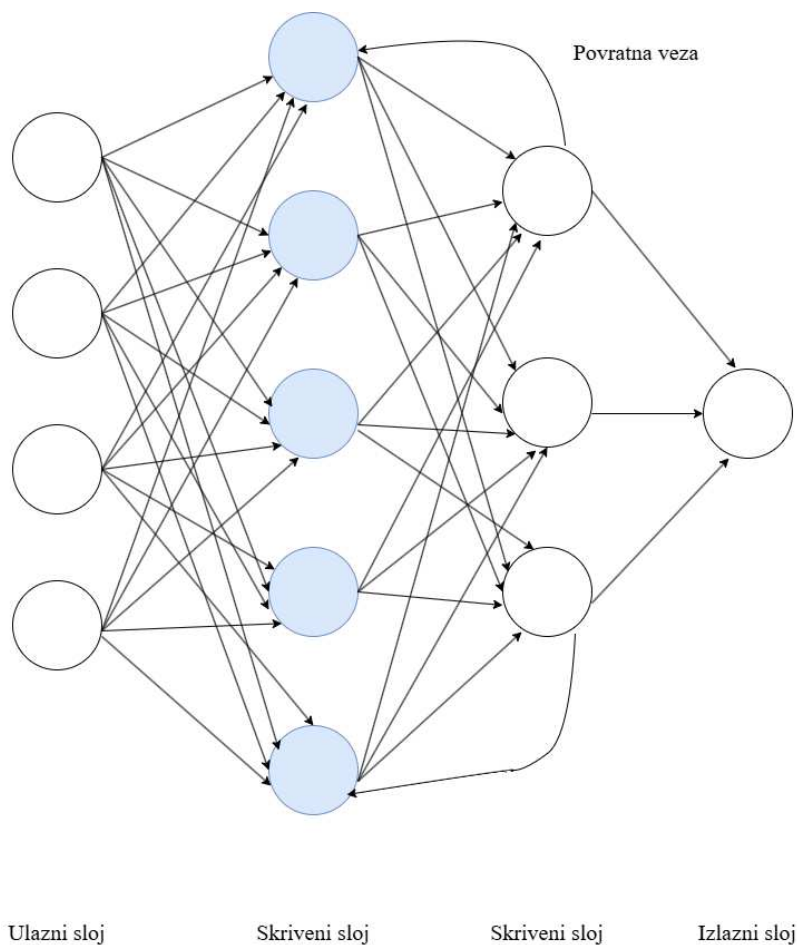
pogreške u odnosu na težine slojeva mreže. Željena točnost postiže se korištenjem različitih optimizacijskih algoritama, jedan od učestalo korištenih je stohastički gradijentni spust koji prilagođava težine u smjeru suprotnom od gradijenta, što smanjuje vrijednost funkcije gubitka. Smisao procesa treniranja je optimizacija parametara, kako bi na stvarnim podacima točnost bila što veća, što je zapravo ključ uspjeha primjene modela. Posljednji korak u cijelom procesu je evaluacija performansi. Njen cilj je utvrđivanje točnosti modela s pomoću različitih metrika performansi, tehnika validacije i analize robusnosti modela. Prva metrika koja se koristi za analizu je točnost. Točnost se računa kao omjer broja točnih predikcija i ukupnog broja predikcija, navedena predikcija je intuitivna i jednostavna, no kod neuravnoteženih klasa njena točnost može uvelike varirati. Preciznost je sljedeća metrika koja mjeri omjer ispravno klasificiranih pozitivnih uzoraka u odnosu na ukupan broj uzoraka koji su klasificirani kao pozitivni, vrlo je korisna gdje je potrebno minimizirati broj lažno pozitivnih predikcija, no njena mana je kako ne daje uvid u cjelokupnu uspješnost modela. Sljedeća metrika je odziv koja računa koliko često korišteni model identificira pozitivne instance uzevši u obzir sve pozitivne uzorke unutar korištenog seta podataka, glavni nedostatak je moguće smanjenje preciznosti koje rezultira velikim brojem lažno pozitivnih predikcija. F1 mjera sljedeća je metrika koja se izračunava kao harmonijska sredina prijašnje objašnjenih metrika preciznosti i odziva, navedena metrika pruža objektivno uravnoteženu mjeru uspješnosti modela. Često je korištena kod neuravnoteženih klasa, no teža je za interpretaciju od prije objašnjenih metrika. Posljednja relativno jednostavna metrika je specifičnost koja mjeri omjer ispravno identificiranih negativnih uzoraka u odnosu na ukupan broj stvarno negativnih uzoraka, njena primjena se nalazi u zadacima gdje lažno pozitivni rezultati mogu imati ozbiljne posljedice, kao što su kriminalistička istraživanja. Korištenje pravih metrika uvelike ovisi o samoj primjeni i problemu te ishodu koji se očekuje. (LeCun, Y., Bengio, Y., & Hinton, G., 2015. Deep learning. *Nature*, 521(7553), pp.436-444).

3.2. Rekurentne neuronske mreže (RNN)

Rekurentna neuronska mreža (RNN) vrsta je neuronske mreže gdje se izlaz iz prethodnog koraka unosi kao ulaz u trenutni korak. (GeeksforGeeks. (2023).

Introduction to Recurrent Neural Network. *GeeksforGeeks*. Dostupno na: <https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/>).

Pri korištenju tradicionalnih neuronskih mreža, pretežito svi ulazi i izlazi su neovisni jedni o drugima, no postoje specifični slučajevi primjerice gdje je potrebno predvidjeti sljedeću riječ rečenice, u ovome slučaju u obzir dolaze rekurentne neuronske mreže koje s pomoću svojeg skrivenog sloja pamte informacije o određenom nizu. Navedeno stanje se isto naziva stanje memorije pošto pamti prethodni unos u mrežu.



Slika 6. Rekurentna neuronska mreža

Izvor: Autor modificirao prema (BotPenguin, 2024)

Struktura rekurentne neuronske mreže sadržava: ulazni sloj, skriveni sloj i izlazni sloj. Kod ulaznog sloja podaci dolaze u obliku sekvence, dok se sekvence sastoje od elemenata kao što je riječ u rečenici. Ovisno o primjeni neuronske mreže, sekvence mogu imati različite dimenzije. Skriveni sloj ključan je korak jer se u njemu događa obrada podataka uz pomoć težina i aktivacijskih funkcija. Glavna stvar koja razlikuje rekurentnu neuronsku mrežu od tradicionalnih mreža je rekurentna veza koja omogućuje prijenos izlaza iz trenutnog vremenskog koraka t u sljedeći vremenski korak $t + 1$. To omogućava mreži da pamti informacije iz prethodnih vremenskih koraka i koristi ih za daljnju obradu. Stanje skrivenog sloja se izračunava sumom ulaznih vrijednosti pomnoženih pripadajućim težinama, trenutnog stanja te primijenjene aktivacijske

funkcije. Najčešće korištene aktivacijske funkcije u ovome slučaju su *ReLU* i hiperbolički tangens. Na kraju neuronske mreže nalazi se prije navedeni izlazni sloj koji daje konačni rezultat mreže. Rezultat može biti u obliku klasifikacije ili regresije, a često se koristi *softmax* funkcija za višeklasnu klasifikaciju. Postoje razne varijacije rekurentnih neuronskih mreža koje se koriste ovisno o zadatku, glavna upotreba mreža pronalazi se kod: prijevoda riječi, generiranja i modeliranja teksta (NLP), prepoznavanja slika i detekcije lica. Mana mreže se nalazi kod propagacije unazad gdje gradijenti postaju sve manji, što mreža ide više unazad to mreža potencijalno može biti nesposobnija prenositi informacije.

3.3. Kombinirane arhitekture

Kombinirane arhitekture ili hibridne neuronske mreže teže spajanju najboljih značajki konvolucijskih i rekurentnih neuronskih mreža. Hibridnost mreža koristi se pri specifičnim zadacima gdje je potrebno obraditi sekvencijalne, vremenske i prostorne informacije. Video analiza je specifičan zadatak gdje je hibridna neuronska mreža pokazala visoke rezultate uspješnosti. Korištenjem konvolucijske neuronske mreže analiziraju se prostorne informacije kao što su objekti unutar slika, dok se rekurentna neuronska mreža bavi obradom sekvencijalnih podataka kao što je redoslijed događaja i riječi. U navedenom zadatku kombiniranjem tih dviju arhitektura, modeli mogu puno bolje te učinkovitije obraditi složene podatke. Dalju primjenu kombiniranih arhitektura pronalazi se kod prepoznavanja rukopisa, obrade prirodnog jezika s vizualnim podacima i slično. Navedene mreže bolje razumiju složenost zadataka te su fleksibilnije to jest prilagođavaju se različitim problemima, s druge strane korištenjem istih podiže se računalna složenost, to jest mreža zahtjeva značajno veće računalne resurse od tradicionalnih, te optimizacija samih modela može biti složena zbog međusobne integracije istih. (LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553)).

4. Implementacija i analiza algoritama za detekciju i raspoznavanje registarskih oznaka

U ovom dijelu rada prikazana je implementacija različitih algoritama za detekciju i prepoznavanje registarskih oznaka, s naglaskom na specifične izazove koji su se pojavili tijekom procesa. Cilj je prikazati glavne prepreke i tehničke zahtjeve koji su se pojavili u praktičnoj primjeni sustava za automatsko prepoznavanje registarskih oznaka, kao i analizirati učinkovitost odabranih metoda u stvarnim uvjetima.

4.1. Izazovi u detekciji i raspoznavanju registarskih oznaka

Detekcija i raspoznavanje registarskih oznaka vozila u današnjem svijetu suočava se s mnogobrojnim problemima te izazovima koji uvelike utječu na funkcionalnost i točnost navedenih sustava. Problemi nastaju iz različitih scenarija kao što su kutovi snimanja, vrsta vozila koje prolazi, hardverska ograničenja kamera, korištenih algoritama i posebno raznim vremenskim nepogodama i promjenama u osvjetljenju.

Prvi problem koji može stvarati probleme je zapravo veličina i oblik registracijske oznake koja nije standardizirana u svijetu, već svaka država ili regija može imati svoj propisani oblik i dimenziju koja je prepisana po zakonu. Uz navedeno, fontovi tablica mogu biti različitog oblika i boje.



Slika 7. Prikaz različitih automobilskih tablica ovisno o regiji i državi

Izvor: Izradio autor

Zbog različitih nagiba i pozicija cesta na kojima sustav snima, potrebno ga je postaviti pod različitim kutovima, što uvelike može utjecati na kvalitetu slike te može proizvesti raznolika izobličenja i ultimativno otežati detekciju i prepoznavanje. Uvjeti variraju o trenutačnoj poziciji sunca, osvjetljenja ceste i raznih noćnih scena. Promjene u osvjetljenju potencijalno uzrokuju sjene, vidljivost i refleksije. Uz sve navedeno vrlo je i bitna brzina kojom se vozilo približava, proporcionalno što se vozilo kreće većom brzinom teže je prepoznati tablicu zbog smanjenja kvalitete slike.

Prisutnost šuma isto tako može stvarati velike probleme pri čitljivosti slike. Šum može nastati iz različitih izvora, kao što je elektronički šum senzora kamere, prašine i različitih atmosferskih uvjeta (snijeg, led, kiša, magla). Šum može nastati i dalje kod procesa kompresije gdje uvelike smanjuje jasnoću i daljnji proces prepoznavanja slike.



Slika 8. Nastanak šuma zbog neadekvatnog osvjetljenja kamere

Izvor: <https://www.ubicept.com/blog/license-plate-recognition-in-low-light> [pristupljeno 28. kolovoza 2024.]

Daljnji problemi uključuju prepreke i opstrukcije koje mogu djelomično ili potpuno blokirati registarske oznake vozila. Navedene prepreke uključuju smetnje fizičkih objekata kao što su stabla, grane, listovi, nosači, blatobrani automobila i ostalo.

Ovisno o vrsti sustava i njegovoj proizvodnji, on može imati određena hardverska ograničenja to jest limitacije koje utječu na rezoluciju slike. Pretežito je bitna kvalitetna leća kamere koja utječe na oštrinu slike. Osim hardvera, performanse algoritma igraju ključnu ulogu u prepoznavanju, algoritmi moraju biti dovoljno brzi da omoguće pravovremenu detekciju registarske oznake vozila.

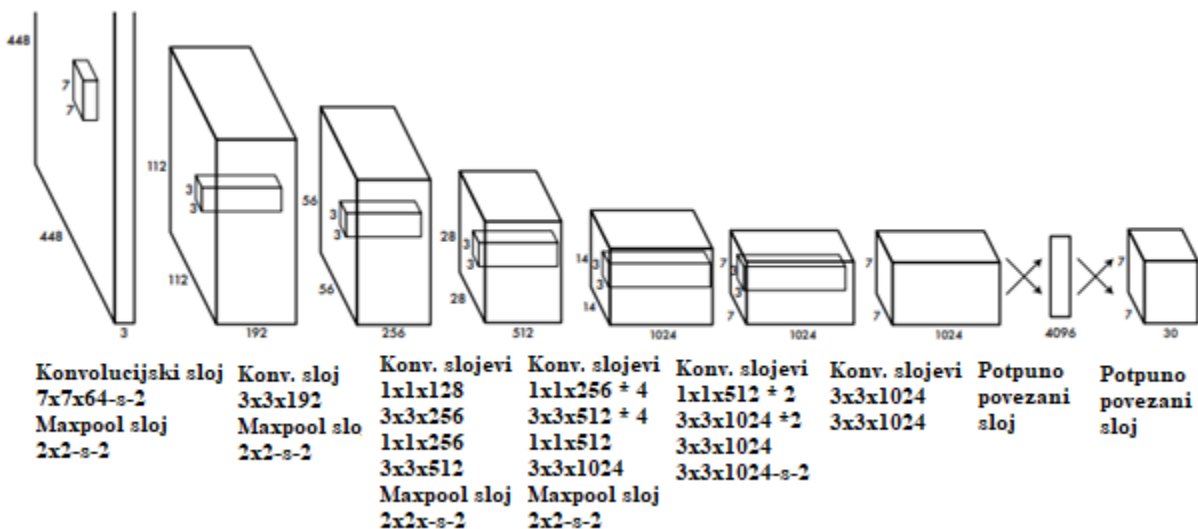
Ovaj diplomski rad istražuje kako se različite neuronske mreže, uključujući i vlastiti model, mogu potencijalno nositi s nabrojenim izazovima u pravome svijetu, te koliko su učinkovite u detekciji i prepoznavanju registarskih oznaka automobila u različitim uvjetima.

4.2. Korišteni skup podataka i postupci predobrade

Za potrebe ovog diplomskog rada korišten je javno dostupan skup podataka s internetske platforme Kaggle. Navedeni skup podataka sadrži 433 slike automobilskih registracijskih oznaka slikanih u različitim uvjetima i 45 negativnih slika koje ne sadrže registrarske oznake. Skup podataka je popraćen anotacijama koje sadržavaju XML datoteke s informacijama o poziciji registrarskih oznaka na slici.

4.3. Implementacija YOLO algoritma za detekciju registrarskih oznaka

YOLO (*You Only Look Once*) je *state-of-the-art, real – time object detection* algoritam koji je implementiran u ovome radu zbog svoje impresivne brzine i točnosti, te je poznat po brzini i točnosti detekcije objekata u polju računalnog vida. Algoritam je predstavljen 2015. godine, a izumitelj je Joseph Redmon. Navedeni model je specifičan po tome što koristi *bounding box* i klase objekta u jednoj prolaznoj iteraciji kroz sliku za razliku od ostalih R-CNN koji koriste regionalne prijedloge.



Slika 9. Arhitektura YOLO algoritma

Izvor: Autor modificirao prema (Redmon i dr., 2016).

Za potrebe treniranja YOLO algoritma na skupu podataka, bilo je potrebno konvertirati anotacije registarskih oznaka iz XML formata u YOLO format.

```
import os
import glob
import random
import shutil
import xml.etree.ElementTree as ET
from PIL import Image

annotations_path = '/Users/nikolamerlic/Desktop/Diplomski
Rad/YOLO_sve/KonacnoTestiranje/yolov5/CistiPodaci/dataset/annotations'
images_path = '/Users/nikolamerlic/Desktop/Diplomski
Rad/YOLO_sve/KonacnoTestiranje/yolov5/CistiPodaci/dataset/images'
dataset_path = '/Users/nikolamerlic/Desktop/Diplomski
Rad/YOLO_sve/KonacnoTestiranje/yolov5/CistiPodaci/rezultat'

for folder in ['images/train', 'images/val', 'images/test',
'labels/train', 'labels/val', 'labels/test']:
    os.makedirs(os.path.join(dataset_path, folder), exist_ok=True)

images = glob.glob(os.path.join(images_path, '*.png'))

random.shuffle(images)
train_split = int(0.7 * len(images))
val_split = int(0.85 * len(images))

train_images = images[:train_split]
val_images = images[train_split:val_split]
test_images = images[val_split:]

def convert_annotation(xml_file, txt_file, image_width, image_height):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    with open(txt_file, 'w') as f:
        for obj in root.iter('object'):
            cls = obj.find('name').text
            if cls != 'licence':
                continue
            cls_id = 0 # 'licence' je jedina klasa
```

```

        xmlbox = obj.find('bndbox')
        xmin = int(xmlbox.find('xmin').text)
        xmax = int(xmlbox.find('xmax').text)
        ymin = int(xmlbox.find('ymin').text)
        ymax = int(xmlbox.find('ymax').text)
        # Konvertiranje u YOLO format
        x_center = (xmin + xmax) / 2 / image_width
        y_center = (ymin + ymax) / 2 / image_height
        width = (xmax - xmin) / image_width
        height = (ymax - ymin) / image_height
        f.write(f"{cls_id} {x_center} {y_center} {width}
{height}\n")

def process_images(image_list, subset):
    for img_path in image_list:

        img_name = os.path.basename(img_path)
        shutil.copy(img_path, os.path.join(dataset_path,
f'images/{subset}', img_name))

        xml_name = img_name.replace('.png', '.xml')
        xml_file = os.path.join(annotations_path, xml_name)
        txt_file = os.path.join(dataset_path, f'labels/{subset}',
xml_name.replace('.xml', '.txt'))

        # Provjera postoji li XML datoteka
        if os.path.exists(xml_file):
            # Učitavanje veličine slike
            img = Image.open(img_path)
            width, height = img.size

            convert_annotation(xml_file, txt_file, width, height)

# Obrada skupova
process_images(train_images, 'train')
process_images(val_images, 'val')
process_images(test_images, 'test')

print("Dataset uspješno podijeljen na trening, validaciju i
testiranje.")

```

Slika 10. Pretvorba anotacija iz XML formata u YOLO format

Izvor: Izradio autor

U sklopu ovog rada razvijena je skripta koja automatizira pripremu podatkovnog skupa za treniranje, validaciju i testiranje modela za detekciju registarskih pločica koristeći algoritam YOLOv5. Skripta osigurava podjelu podataka te konverziju anotacija iz Pascal VOC formata u YOLO format. Yolo format je specifični format za označavanje graničnih okvira objekata na slikama. U Yolo formatu, anotacije su zapisane u tekstualnim .txt datotekama, pri čemu svaka slika ima svoju pripadajuću .txt datoteku istog imena. Svaka linija u .txt datoteci predstavlja jedan objekt na slici. Za razliku od Pascal VOC formata, koji koristi XML datoteke i apsolutne koordinate, Yolo koristi normalizirane vrijednosti u rasponu od [0,1]. Skup podataka dijeli se na tri podskupa u sljedećim omjerima: 70 % trening skup, 15 % validacijski skup i 15 % Testni skup.

Sljedeći korak je dodavanje dva različita šuma: *salt & pepper* i *Gaussov* šum. Navedeni šumovi se koriste kako bi simulirali realne uvjete u kojima algoritam funkcionira, te testirali njegovu otpornost.

Navedena skripta koristi funkciju `add_salt_and_pepper_noise` koja nasumično na slike dodaje zrnca bijelih i crnih točkica. `process_images_with_salt_pepper_noise` prolazi kroz sve slike koje pronalazi u direktorijima te primjenjuje navedeni šum i pohranjuje ga.

```
import os
import glob
import random
import shutil
import xml.etree.ElementTree as ET
from PIL import Image
import numpy as np
import cv2

def add_salt_and_pepper_noise(image, amount=0.1, s_vs_p=0.5):
    row, col, ch = image.shape
    out = np.copy(image)

    # Dodavanje "soli"
    num_salt = np.ceil(amount * image.size * s_vs_p).astype(int)
```

```

    coords = [np.random.randint(0, i - 1, int(num_salt / ch)) for i in
image.shape[:2]]
    for i in range(ch):
        out[coords[0], coords[1], i] = 255

    # Dodavanje "papra"
    num_pepper = np.ceil(amount * image.size * (1. -
s_vs_p)).astype(int)
    coords = [np.random.randint(0, i - 1, int(num_pepper / ch)) for i
in image.shape[:2]]
    for i in range(ch):
        out[coords[0], coords[1], i] = 0

    return out

def convert_annotation(xml_file, txt_file, image_width, image_height):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    with open(txt_file, 'w') as f:
        for obj in root.iter('object'):
            cls = obj.find('name').text
            if cls != 'licence':
                continue
            cls_id = 0 # 'licence' je jedina klasa
            xmlbox = obj.find('bndbox')
            xmin = int(float(xmlbox.find('xmin').text))
            xmax = int(float(xmlbox.find('xmax').text))
            ymin = int(float(xmlbox.find('ymin').text))
            ymax = int(float(xmlbox.find('ymax').text))
            # Konvertiraj u YOLO format
            x_center = ((xmin + xmax) / 2) / image_width
            y_center = ((ymin + ymax) / 2) / image_height
            width = (xmax - xmin) / image_width
            height = (ymax - ymin) / image_height
            f.write(f"{cls_id} {x_center} {y_center} {width}
{height}\n")

def process_images_with_salt_pepper_noise(image_paths,
annotations_path, output_base_dir):
    # Kreiranje direktorija
    image_output_dir = os.path.join(output_base_dir, 'dataset',
'images')
    annotations_output_dir = os.path.join(output_base_dir, 'dataset',
'annotations')

```

```

os.makedirs(image_output_dir, exist_ok=True)
os.makedirs(annotations_output_dir, exist_ok=True)

for img_path in image_paths:
    img_name = os.path.basename(img_path)
    image = cv2.imread(img_path)

    # Dodavanje Salt & Pepper šuma
    noisy_image = add_salt_and_pepper_noise(image, amount=0.1) #
Jačina šuma

    # Spremanje slike s šumom
    cv2.imwrite(os.path.join(image_output_dir, img_name),
noisy_image)

    xml_name = img_name.replace('.png', '.xml').replace('.jpg',
'.xml')
    xml_file = os.path.join(annotations_path, xml_name)
    xml_output_file = os.path.join(annotations_output_dir,
xml_name)

    if os.path.exists(xml_file):
        shutil.copy(xml_file, xml_output_file)

def split_dataset_and_convert_annotations(base_dir):
    images_dir = os.path.join(base_dir, 'dataset', 'images')
    annotations_dir = os.path.join(base_dir, 'dataset', 'annotations')

    images = glob.glob(os.path.join(images_dir, '*.png')) +
glob.glob(os.path.join(images_dir, '*.jpg'))
    images.sort()
    random.shuffle(images)

    train_split = int(0.7 * len(images))
    val_split = int(0.85 * len(images))

    subsets = {
        'train': images[:train_split],
        'val': images[train_split:val_split],
        'test': images[val_split:]
    }

    for subset, image_list in subsets.items():

```

```

        images_output_dir = os.path.join(base_dir, 'dataset', 'images',
subset)
        labels_output_dir = os.path.join(base_dir, 'dataset', 'labels',
subset)
        os.makedirs(images_output_dir, exist_ok=True)
        os.makedirs(labels_output_dir, exist_ok=True)

        for img_path in image_list:
            img_name = os.path.basename(img_path)
            shutil.move(img_path, os.path.join(images_output_dir,
img_name))

            # Konverzija anotacija
            xml_name = img_name.replace('.png', '.xml').replace('.jpg',
'.xml')
            xml_file = os.path.join(base_dir, 'dataset', 'annotations',
xml_name)
            txt_file = os.path.join(labels_output_dir,
img_name.replace('.png', '.txt').replace('.jpg', '.txt'))

            if os.path.exists(xml_file):
                # Učitavanje veličine slike
                image = cv2.imread(os.path.join(images_output_dir,
img_name))
                height, width, _ = image.shape

                # Konvertiranje u YOLO
                convert_annotation(xml_file, txt_file, width, height)

                # Opcionalno: Obrisati XML anotaciju nakon konverzije
                # os.remove(xml_file)

            if os.path.exists(annotations_dir):
                shutil.rmtree(annotations_dir)

def prepare_salt_pepper_data(input_image_dir, input_annotation_dir,
output_base_dir):
    # Kreiranje glavnih direktorija
    os.makedirs(output_base_dir, exist_ok=True)
    os.makedirs(os.path.join(output_base_dir, 'dataset'),
exist_ok=True)
    os.makedirs(os.path.join(output_base_dir, 'dataset', 'images'),
exist_ok=True)

```

```

    os.makedirs(os.path.join(output_base_dir, 'dataset',
'annotations'), exist_ok=True)
    os.makedirs(os.path.join(output_base_dir, 'dataset', 'labels'),
exist_ok=True)

    # Učitavanje svih slika
    image_paths = glob.glob(os.path.join(input_image_dir, '*.png')) +
glob.glob(os.path.join(input_image_dir, '*.jpg'))
    image_paths.sort()

    process_images_with_salt_pepper_noise(image_paths,
input_annotation_dir, output_base_dir)

    split_dataset_and_convert_annotations(output_base_dir)

    print("Dataset s Salt & Pepper šumom uspješno pripremljen i
podijeljen.")

input_image_dir = '/Users/nikolamerlic/Desktop/Diplomski
Rad/YOLO_sve/KonacnoTestiranje/yolov5/CistiPodaci/dataset/images'
input_annotation_dir = '/Users/nikolamerlic/Desktop/Diplomski
Rad/YOLO_sve/KonacnoTestiranje/yolov5/CistiPodaci/dataset/annotations'

output_base_dir_sp = '/Users/nikolamerlic/Desktop/Diplomski
Rad/YOLO_sve/KonacnoTestiranje/yolov5/SaltPepperPodaci'

# Pokretanje pripreme podataka s Salt & Pepper šumom
prepare_salt_pepper_data(input_image_dir, input_annotation_dir,
output_base_dir_sp)

```

Slika 11. Skripta za dodavanje salt & pepper šuma

Izvor: Izradio autor

Sljedeća skripta vrlo je slična prethodnoj, koristi funkciju `add_gaussian_noise` koja dodaje navedeni šum na slike. Šum funkcionira tako što dodatno zamućuje slike, te otežava detekciju registarskih oznaka. U skriptama moguće je povećanje ili smanjenje razine šuma na slikama ovisno o potrebama samog algoritma.


```

import os
import glob
import random
import shutil
import xml.etree.ElementTree as ET
from PIL import Image
import numpy as np
import cv2

def add_gaussian_noise(image, mean=0, var=1000):
    sigma = var ** 0.5
    gaussian = np.random.normal(mean, sigma, image.shape)
    noisy_image = image + gaussian
    noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)
    return noisy_image

def convert_annotation(xml_file, txt_file, image_width, image_height):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    with open(txt_file, 'w') as f:
        for obj in root.iter('object'):
            cls = obj.find('name').text
            if cls != 'licence':
                continue
            cls_id = 0 # 'licence' je jedina klasa
            xmlbox = obj.find('bndbox')
            xmin = int(float(xmlbox.find('xmin').text))
            xmax = int(float(xmlbox.find('xmax').text))
            ymin = int(float(xmlbox.find('ymin').text))
            ymax = int(float(xmlbox.find('ymax').text))
            x_center = ((xmin + xmax) / 2) / image_width
            y_center = ((ymin + ymax) / 2) / image_height
            width = (xmax - xmin) / image_width
            height = (ymax - ymin) / image_height
            f.write(f"{cls_id} {x_center} {y_center} {width}
{height}\n")

def process_images_with_gaussian_noise(image_paths, annotations_path,
output_base_dir):
    # Kreiranje direktorija
    image_output_dir = os.path.join(output_base_dir, 'dataset',
'images')
    annotations_output_dir = os.path.join(output_base_dir, 'dataset',
'annotations')

```

```

os.makedirs(image_output_dir, exist_ok=True)
os.makedirs(annotations_output_dir, exist_ok=True)

for img_path in image_paths:
    img_name = os.path.basename(img_path)
    image = cv2.imread(img_path)

    # Dodavanje Gaussovog šuma
    noisy_image = add_gaussian_noise(image)

    # Spremanje slike s šumom
    cv2.imwrite(os.path.join(image_output_dir, img_name),
noisy_image)

    # Kopiranje anotacija
    xml_name = img_name.replace('.png', '.xml').replace('.jpg',
'.xml')
    xml_file = os.path.join(annotations_path, xml_name)
    xml_output_file = os.path.join(annotations_output_dir,
xml_name)

    if os.path.exists(xml_file):
        shutil.copy(xml_file, xml_output_file)

def split_dataset_and_convert_annotations(base_dir):
    images_dir = os.path.join(base_dir, 'dataset', 'images')
    annotations_dir = os.path.join(base_dir, 'dataset', 'annotations')

    images = glob.glob(os.path.join(images_dir, '*.png')) +
glob.glob(os.path.join(images_dir, '*.jpg'))
    images.sort()
    random.shuffle(images)

    train_split = int(0.7 * len(images))
    val_split = int(0.85 * len(images))

    subsets = {
        'train': images[:train_split],
        'val': images[train_split:val_split],
        'test': images[val_split:]
    }

    for subset, image_list in subsets.items():

```

```

        images_output_dir = os.path.join(base_dir, 'dataset', 'images',
subset)
        labels_output_dir = os.path.join(base_dir, 'dataset', 'labels',
subset)
        os.makedirs(images_output_dir, exist_ok=True)
        os.makedirs(labels_output_dir, exist_ok=True)

        for img_path in image_list:
            img_name = os.path.basename(img_path)
            shutil.move(img_path, os.path.join(images_output_dir,
img_name))

            # Konverzija anotacija
            xml_name = img_name.replace('.png', '.xml').replace('.jpg',
'.xml')
            xml_file = os.path.join(base_dir, 'dataset', 'annotations',
xml_name)
            txt_file = os.path.join(labels_output_dir,
img_name.replace('.png', '.txt').replace('.jpg', '.txt'))

            if os.path.exists(xml_file):
                image = cv2.imread(os.path.join(images_output_dir,
img_name))
                height, width, _ = image.shape

                convert_annotation(xml_file, txt_file, width, height)

        if os.path.exists(annotations_dir):
            shutil.rmtree(annotations_dir)

def prepare_gaussian_data(input_image_dir, input_annotation_dir,
output_base_dir):
    os.makedirs(output_base_dir, exist_ok=True)
    os.makedirs(os.path.join(output_base_dir, 'dataset'),
exist_ok=True)
    os.makedirs(os.path.join(output_base_dir, 'dataset', 'images'),
exist_ok=True)
    os.makedirs(os.path.join(output_base_dir, 'dataset',
'annotations'), exist_ok=True)
    os.makedirs(os.path.join(output_base_dir, 'dataset', 'labels'),
exist_ok=True)

```

```

    image_paths = glob.glob(os.path.join(input_image_dir, '*.png')) +
glob.glob(os.path.join(input_image_dir, '*.jpg'))
    image_paths.sort()

    process_images_with_gaussian_noise(image_paths,
input_annotation_dir, output_base_dir)

    split_dataset_and_convert_annotations(output_base_dir)

    print("Dataset s Gaussovim šumom uspješno pripremljen i
podijeljen.")

input_image_dir = '/Users/nikolamerlic/Desktop/Diplomski
Rad/YOLO_sve/KonacnoTestiranje/yolov5/CistiPodaci/dataset/images'
input_annotation_dir = '/Users/nikolamerlic/Desktop/Diplomski
Rad/YOLO_sve/KonacnoTestiranje/yolov5/CistiPodaci/dataset/annotations'

output_base_dir_gauss = '/Users/nikolamerlic/Desktop/Diplomski
Rad/YOLO_sve/KonacnoTestiranje/yolov5/GaussPodaci'

# Pokretanje pripreme podataka s Gaussovim šumom
prepare_gaussian_data(input_image_dir, input_annotation_dir,
output_base_dir_gauss)

```

Slika 12. Skripta za dodavanje Gaussovog šuma

Izvor: Izradio autor

Nakon uspješno provedenog testiranja i validacije, uspoređuju se određene metrike koje objašnjavaju uspješnost provedenog algoritma. Metrike koje se specifično prate su: Matrica zabune, f1 krivulja, preciznost – odziv krivulja (*engl. precision-recall curve*), preciznost – pouzdanost krivulja (*engl. precision – confidence curve*) i odziv – pouzdanost krivulja (*engl. recall – confidence curve*).

Matrica zabune je tablica koja prikazuje performanse klasifikacijskog modela na skupu podataka za koji su unaprijed poznate stvarne vrijednost, specifično daje na pregled točne i netočne predikcije modela.

F1 krivulja kombinira dvije različite metrike preciznost i točnost te pruža harmonijsku sredinu koja se koristi kada je potrebno balansiranje između istih.

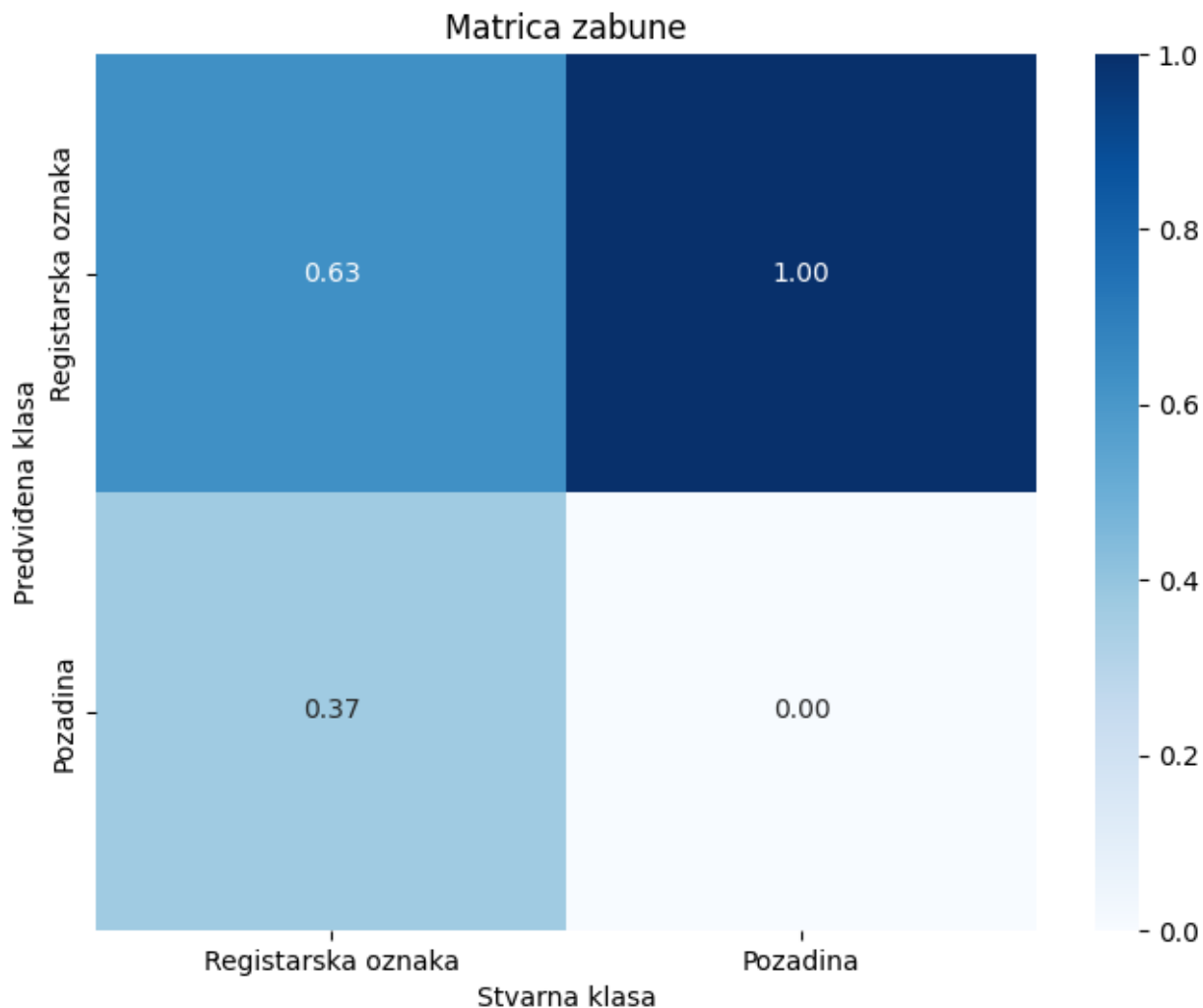
Preciznost - odziv krivulja slično prije navedenome, prikazuje odnos između preciznosti i odziva na različitim razinama granice pouzdanosti.

Preciznost - pouzdanost krivulja prikazuje kako se točnost to jest sama preciznost mijenja s promjenom granice pouzdanosti. Bitno je pronaći balans jer pretjeranim povećavanjem jednog model može biti selektivniji, no može smanjiti broj predikcija što ultimativno nije dobro za odziv.

Odziv - pouzdanost krivulja u ovom specifičnom slučaju služi da odredi koliko dobro model prepoznaje sve stvarne registarske tablice u zadanom skupu podataka. Što je niža granica pouzdanosti to će model biti popustljiviji te moći će identificirati više objekata.

Slijedi prikaz dobivenih rezultata YOLO algoritma, provedenih na prije spomenutom skupu podataka. U sljedećim slikama prikazat će se djelotvornost YOLO algoritma na običnom skupu podataka, koji nema većih opstrukcija te bez umjetno dodanih šumova. Zbog količine generiranih slika tijekom treniranja i validacije modela, u navedenom radu bit će prikazani reprezentativni primjeri rezultata.

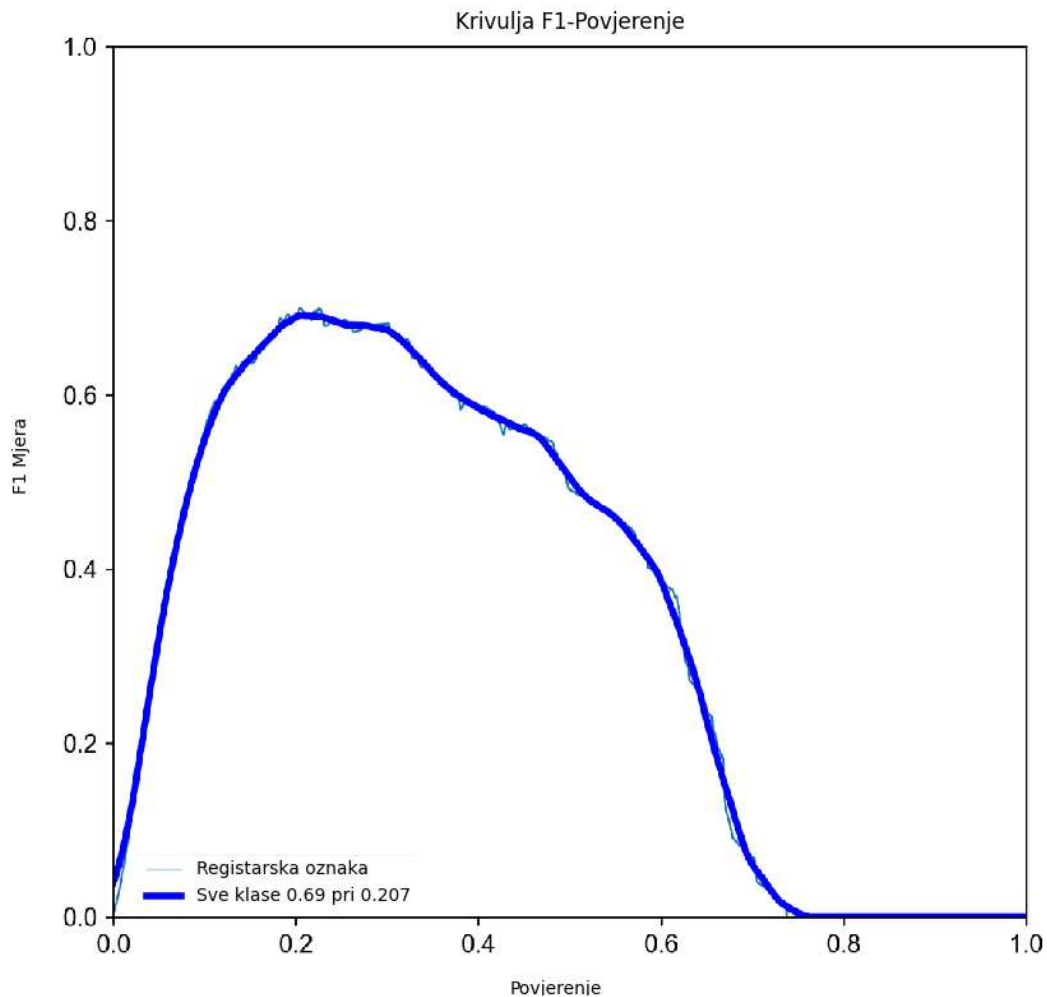
Prva metrika kojom je algoritam podvrgnut na treningu mreže je matrica zabune, čitanjem njenih rezultata može se zaključiti kako je navedeni algoritam ispravno prepoznao registarsku tablicu u 63 %, dok je u 37% slučajeva registarske oznake krivo klasificirao kao pozadinu. Vrijednosti unutar matrice predstavljaju normalizirane udjele klasifikacija.



Slika 13. Matrica zabune

Izvor: Izradio autor

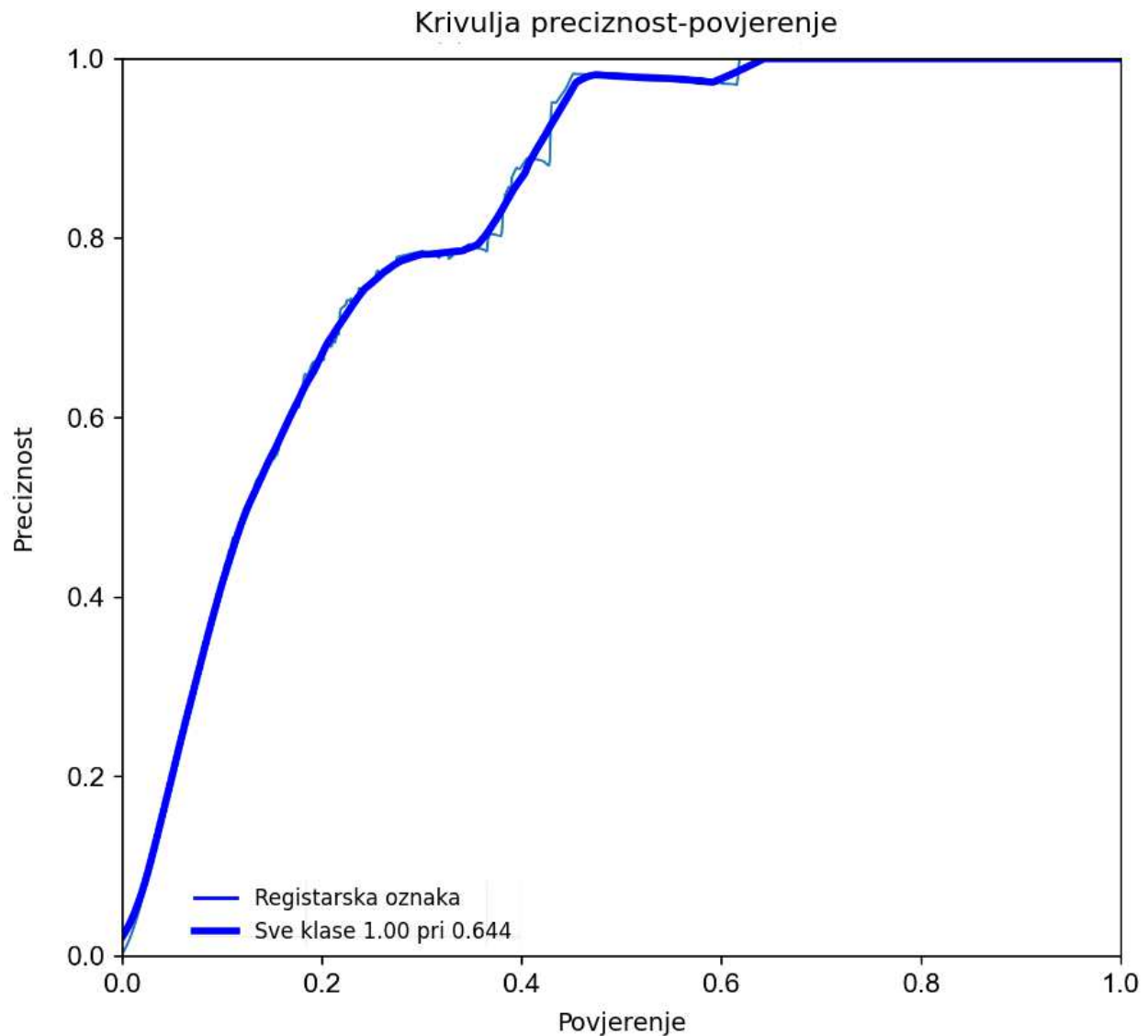
Sljedeća korištena metrika pri treniranju je $F1$ – prag povjerenja (samopouzdanja), iz navedenog grafa možemo očitati kako je najviša vrijednost 0.69 pri pragu povjerenja od 0.207, što prikazuje optimalan prag za klasifikaciju registarskih oznaka. Navedena analiza pomaže u podešavanju modela i odabiru optimalnog praga povjerenja kako bi se postigao najbolji balans između točnih i netočnih detekcija. Na kraju iz grafa se može očitati nagli pad krivulje što označuje konzervativnost modela, moguće je povećanje granice ovisno o potrebi.



Slika 14. F1 – prag samopouzdanja krivulje

Izvor: Izradio autor

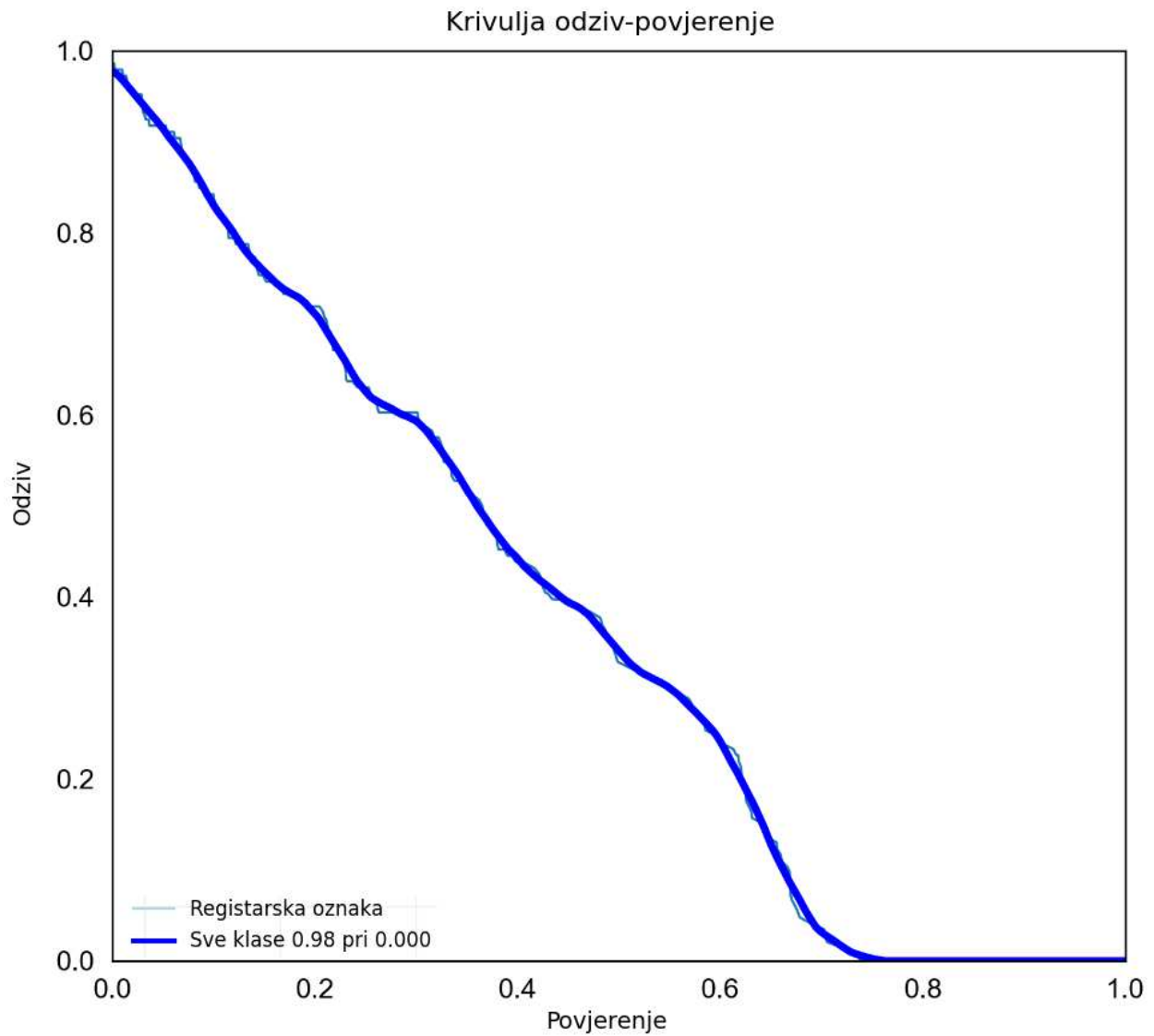
precision – confidence curve prikazuje kako se preciznost mijenja s povećanjem praga povjerenja modela. Detaljnije se može iščitati kako pri pragu povjerenja od 0.644, model postiže maksimalnu preciznost od 1.00 (100 %). Prema navedenom gravu, sve pozitivne detekcije su točne kada je prag povjerenja veći od 0.644. Međutim, niži prag povjerenja omogućuje modelu da detektira više objekata, uz potencijalno veći broj lažno pozitivnih rezultata.



Slika 15. Preciznost – prag povjerenja krivulja

Izvor: Izradio autor

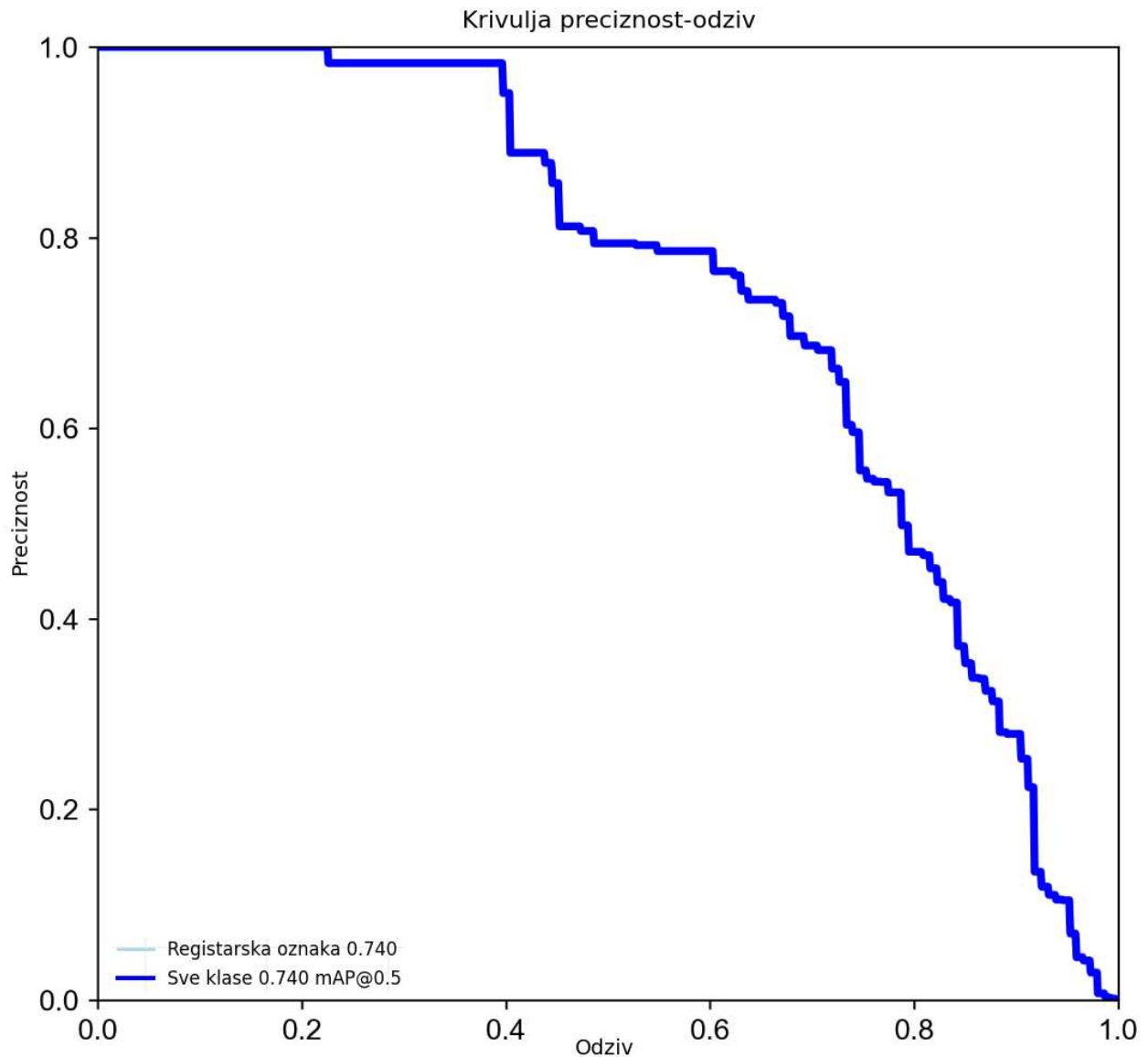
Odziv – povjerenje krivulja pokazuje kako na niskim vrijednostima praga pouzdanosti, odziv je vrlo visok, preciznije 98 %. Najveći odziv od 0.98 postignut je pri povjerenju od 0.000, što ukazuje da model pri vrlo niskom pragu prepoznaje skoro sve registarske oznake, no potencijalno može i generirati lažno pozitivne primjere.



Slika 16. Odziv - pouzdanost krivulja

Izvor: Izradio autor

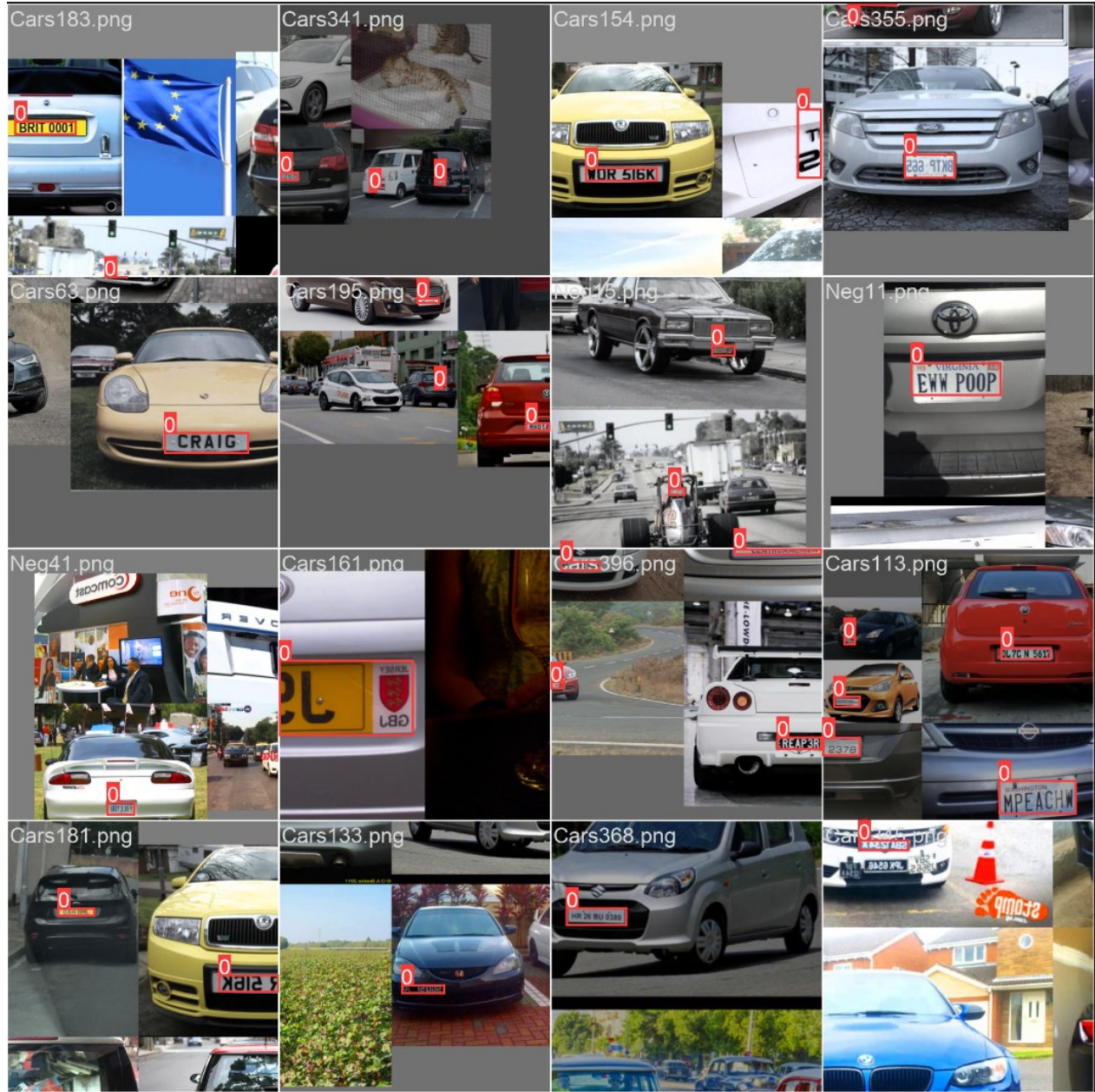
Posljednja korištena metrika je *Precision – Recall Curve* koja pokazuje kako mAP (engl. *Mean Average Precision*) iznosi 0.740, što znači kako model dobro identificira registarske oznake kada je siguran u predikcije, ali pri višim odzivima preciznost postupno pada.



Slika 17. Preciznost – odziv krivulja

Izvor: Izradio autor

Na priloženoj slici prikazuju se validacijski rezultati modela koji je detektirao automobilske tablice na autima. Iz priloženog se može zaključiti kako je model odlično prepoznao registarske tablice automobila na točno prikazanim područjima iz različitih kutova, osvjetljenja te uvjeta. Kao što je prije rečeno zbog velikog broja slika, bit će prikazani reprezentativni uzorci koji pokazuju objektivnu uspješnost modela.

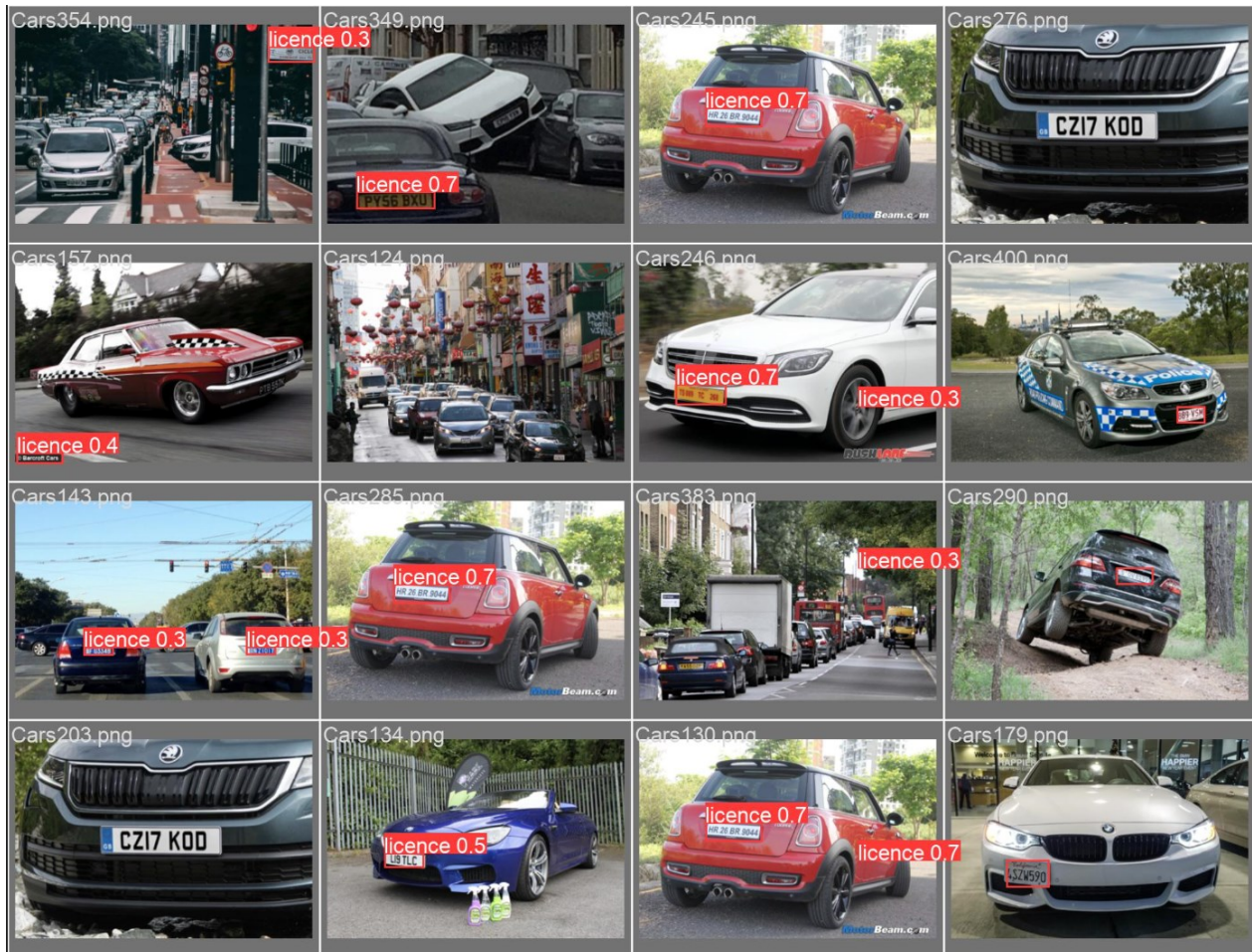


Slika 18. Primjeri slika iz skupa podataka korištenog za treniranje

Izvor: Izradio autor

Na sljedećoj slici možemo vidjeti metrike kao što su *confidence score* koji se u ovom slučaju kreće od 0.3 pa sve do 0.7 (ovisno o *batch-u*). Sama nesigurnost modela

proizlazi iz različitih kutova, mogućeg pozadinskog šuma, refleksije te ostalih varijacija. Usprkos različitom rezultatu metrike, vizualno se može pretpostaviti kako je model vrlo dobro detektirao tablice.



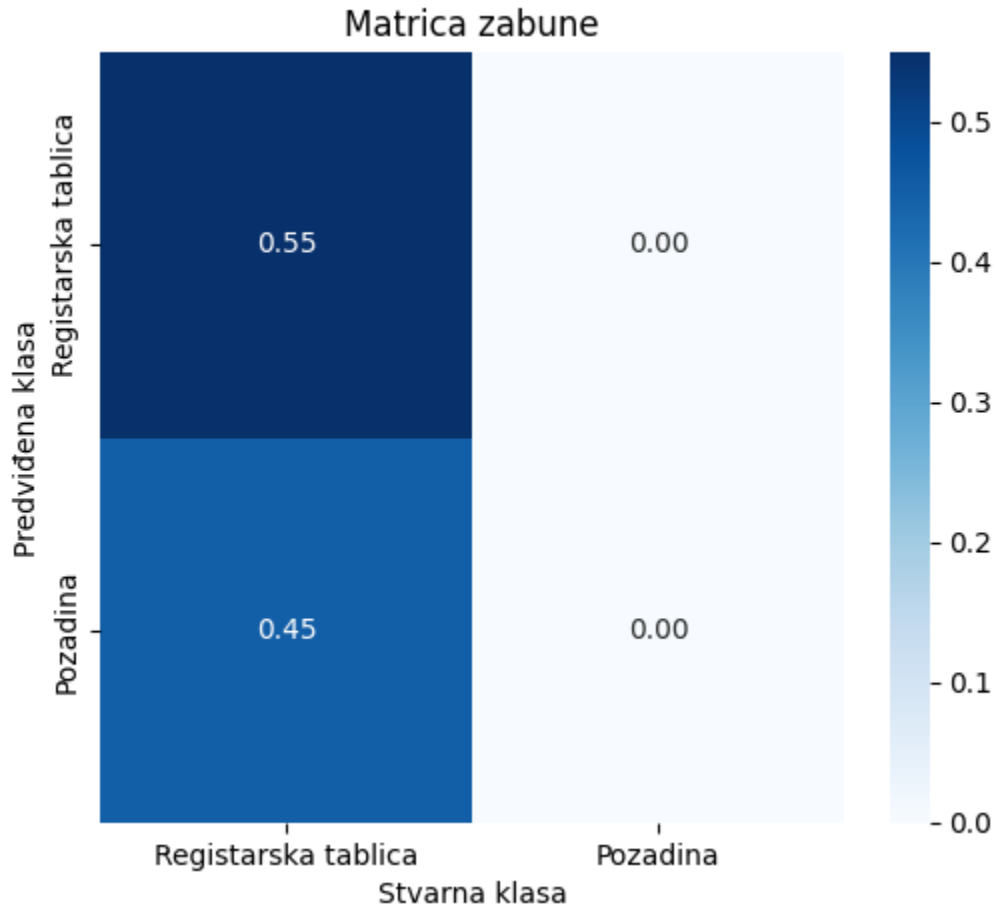
Slika 18. Točnost predikcije modela po tablici

Izvor: Izradio autor

Kao što je prije navedeno, za simulaciju prirodnih nepogoda i uvjeta koristiti će se Gaussov i *salt & pepper* šum. Gaussov šum implementiran je pomoću *numpy* biblioteke unutar Python-a, varijanca koja je ključna vrijednost pri definiranju širina raspona vrijednosti šuma je postavljena na 1000, što znači da je razina šuma na slikama

značajna kako bi što bolje testirala djelotvornost samog algoritma. Nakon uspješnog treninga na čistim podacima (slikama bez umjetno dodanog šuma), dobiveni model koristimo za validaciju i testiranje na slikama sa šumom.

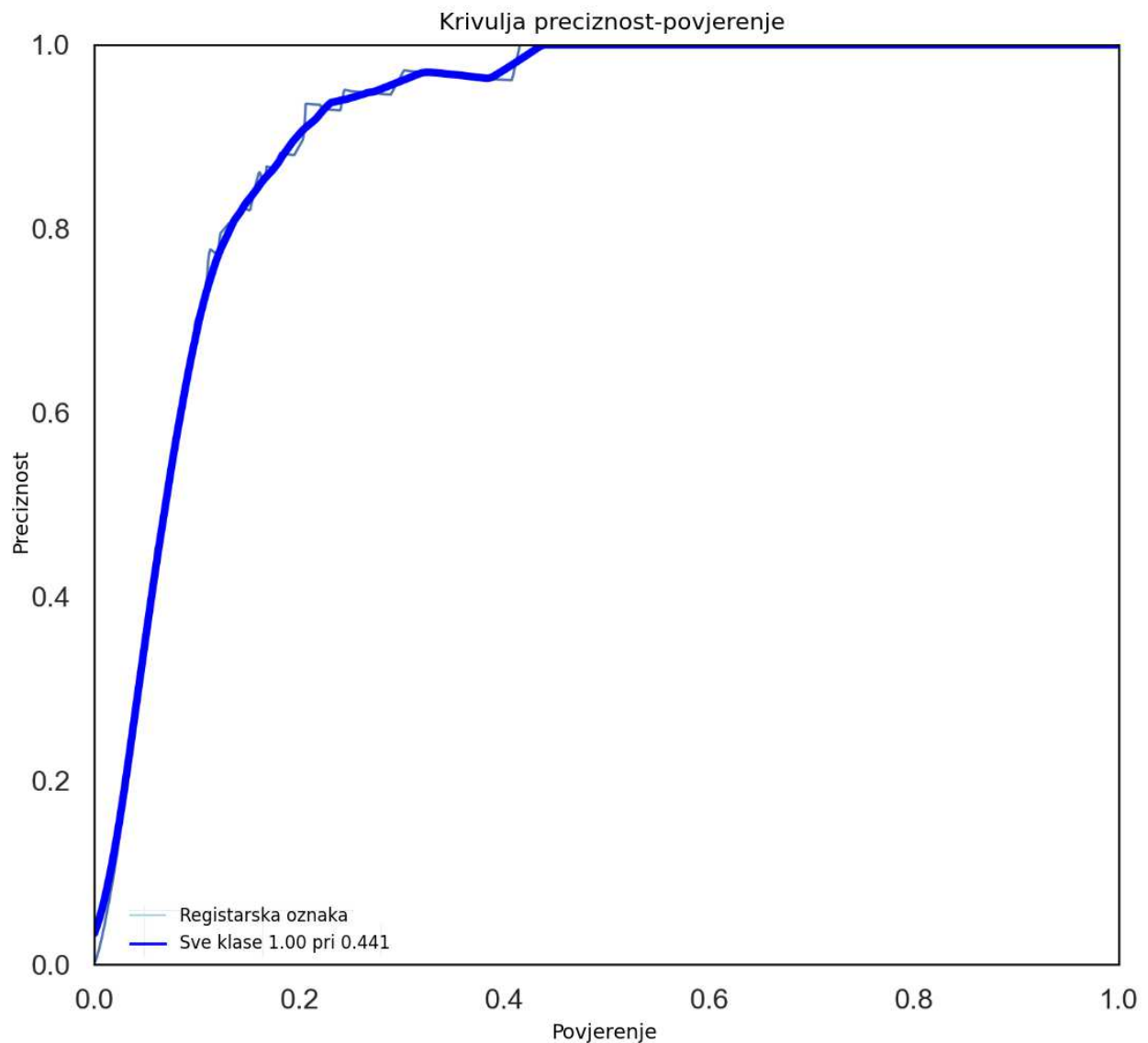
Prva korištena metrika je matrica zabune validacije na podacima s Gausovim šumom. 55% uzoraka registarskih oznaka tablica je ispravno klasificirano, dok preostalih 45% uzoraka je pogrešno klasificirano kao pozadina. Navedena matrica pokazuje da je model sposoban prepoznati registarske tablice, no primjećuje se osjetljivost modela na dodani Gaussov šum.



Slika 19. Matrica zabune validacije na podacima s Gausovim šumom

Izvor: Izradio autor

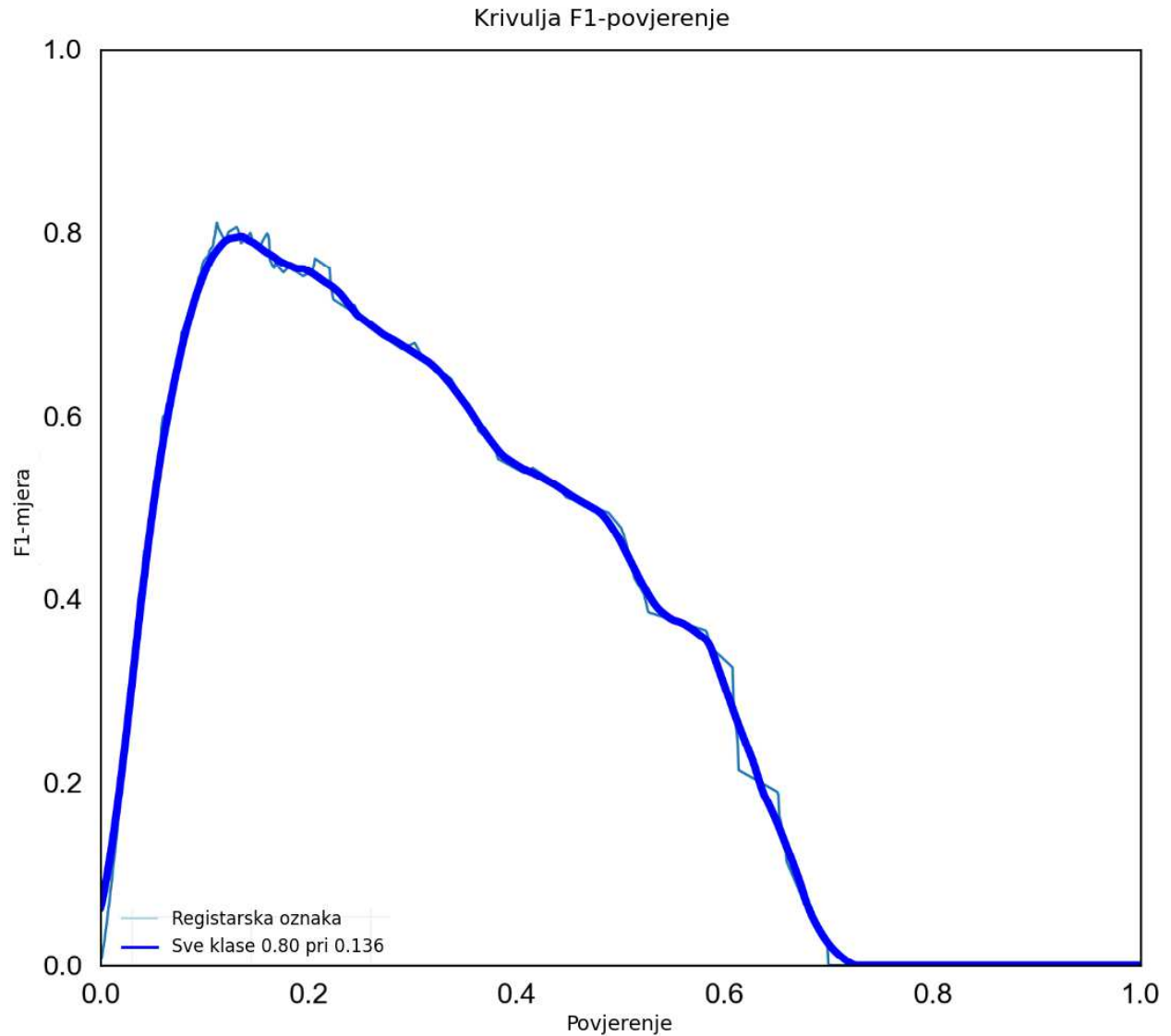
Sljedeća slika prikazuje krivulju preciznosti modela u ovisnosti o pragu pouzdanosti. Krivulja prikazuje kako model postiže visoku preciznost za niže vrijednosti povjerenja. Navedeni rezultat ukazuje na to da model relativno dobro prepoznaje oznake u prisutnosti šuma, do degradacija performansi je očita.



Slika 20. preciznost – prag pouzdanosti krivulja

Izvor: Izradio autor

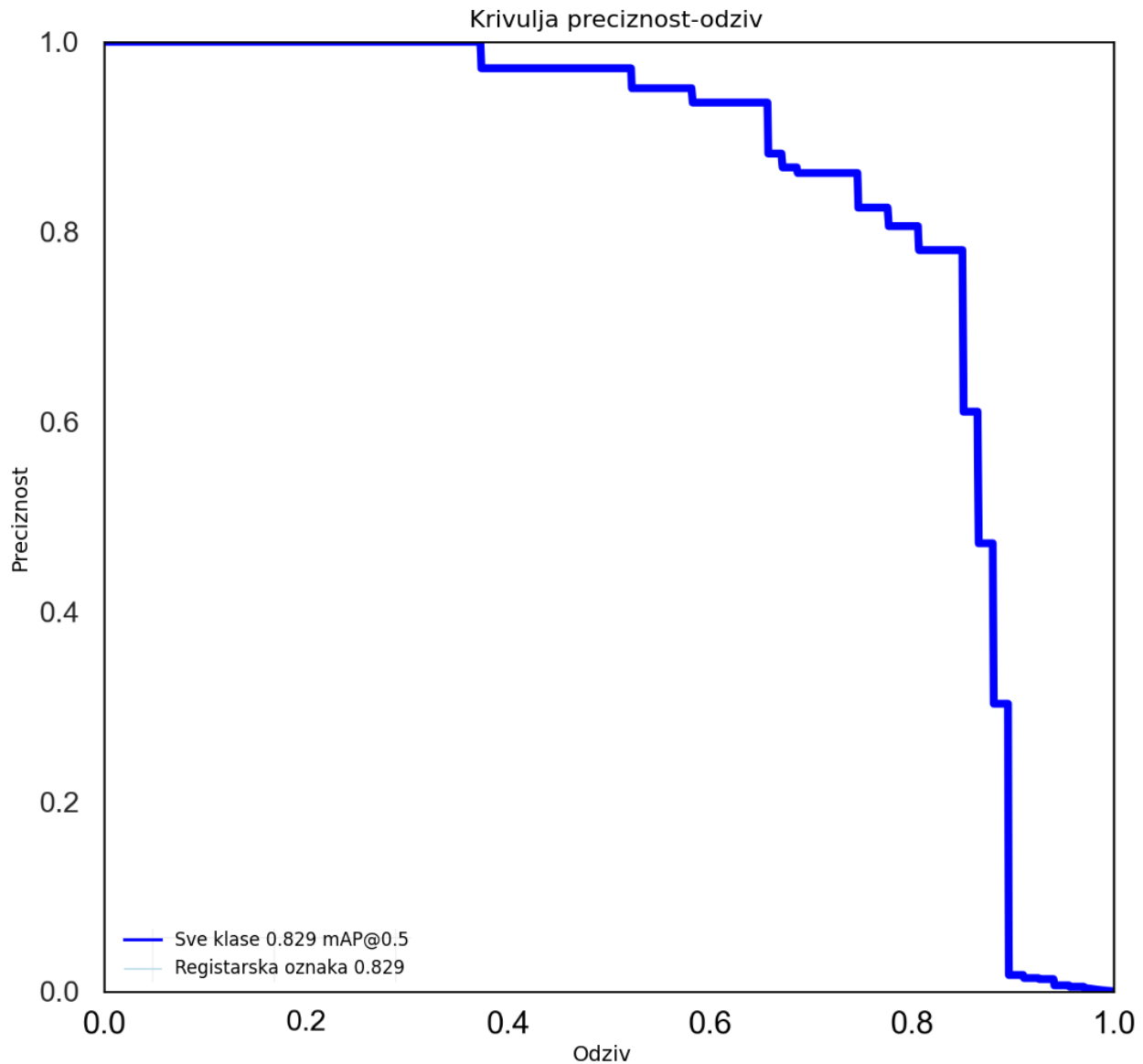
Sljedeća metrika prikazuje f1 krivulju na model ovisnosti o pragu povjerenja. Graf prikazuje visoku f1 vrijednost modela, što znači vrlo dobar balans između dvije navedene metrike, najviša mjera iznosi 0.80 pri povjerenju 0.136.



Slika 21. F1 – prag pouzdanosti krivulja

Izvor: Izradio autor

Krivulja preciznosti – odziv prikazuje odnos između navedenih dviju metrika. Visoka ukupna vrijednost mAP (82.9 %) ukazuje na solidne performanse u balansiranju, no definitivno prisutnost šuma utječe na krajnji rezultat.

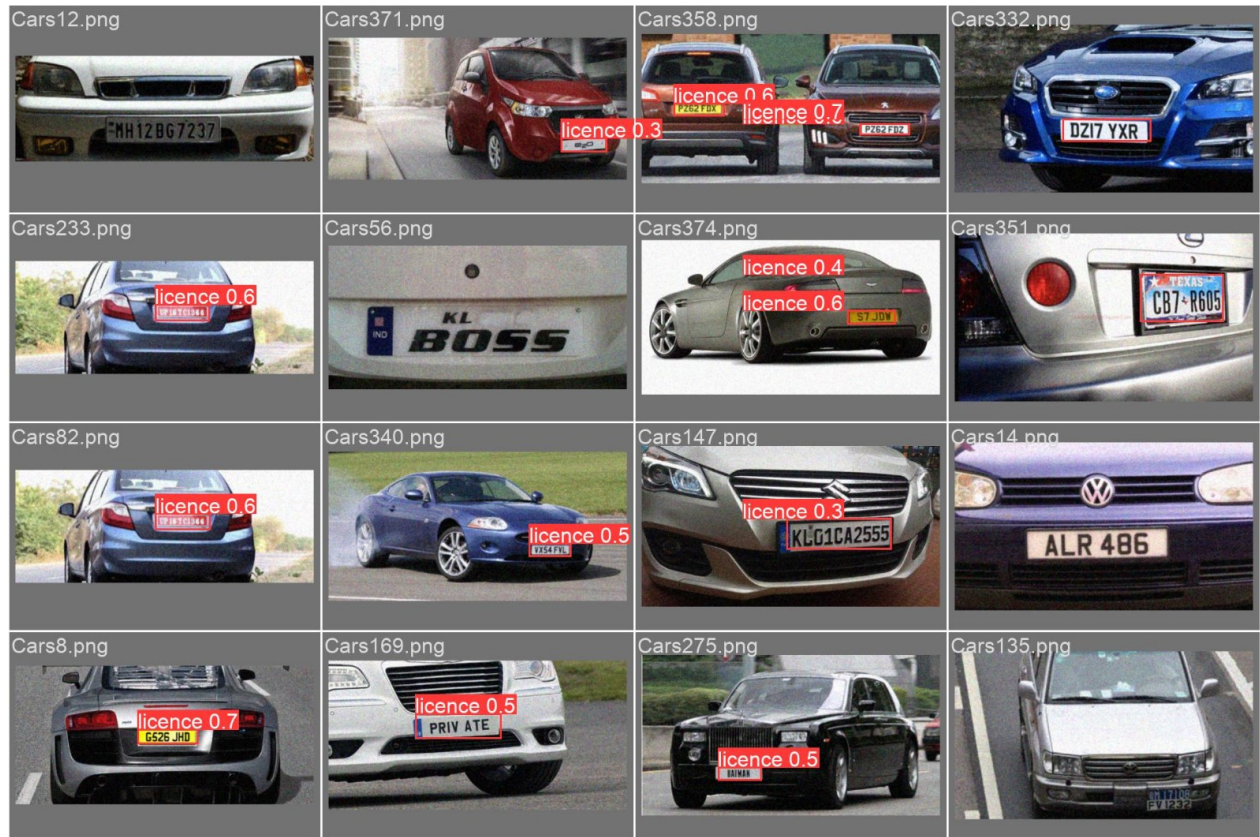


Slika 22. Preciznost - odziv krivulja

Izvor: Izradio autor

Iz prikazane slike može se vidjeti kako model uspješno detektira automobilske tablice unatoč relativno visokoj razini šuma. Detekcije variraju od 0.7 do 0.3, što označuje kako

Gaussov šum itekako ometa sliku i preciznost detekcije. No, usprkos tome model vrlo dobro očituje tablice, što pokazuje djelotvornost u težim uvjetima.



Slika 23. Validacijski rezultat primjene Gaussovog šuma

Izvor: Izradio autor

Sljedeća slika prikazuje jačinu dodanog Gaussovog šuma koji simulira loše uvjete snimanja poput slabog osvjetljenja, kiše ili niske kvalitete kamere.

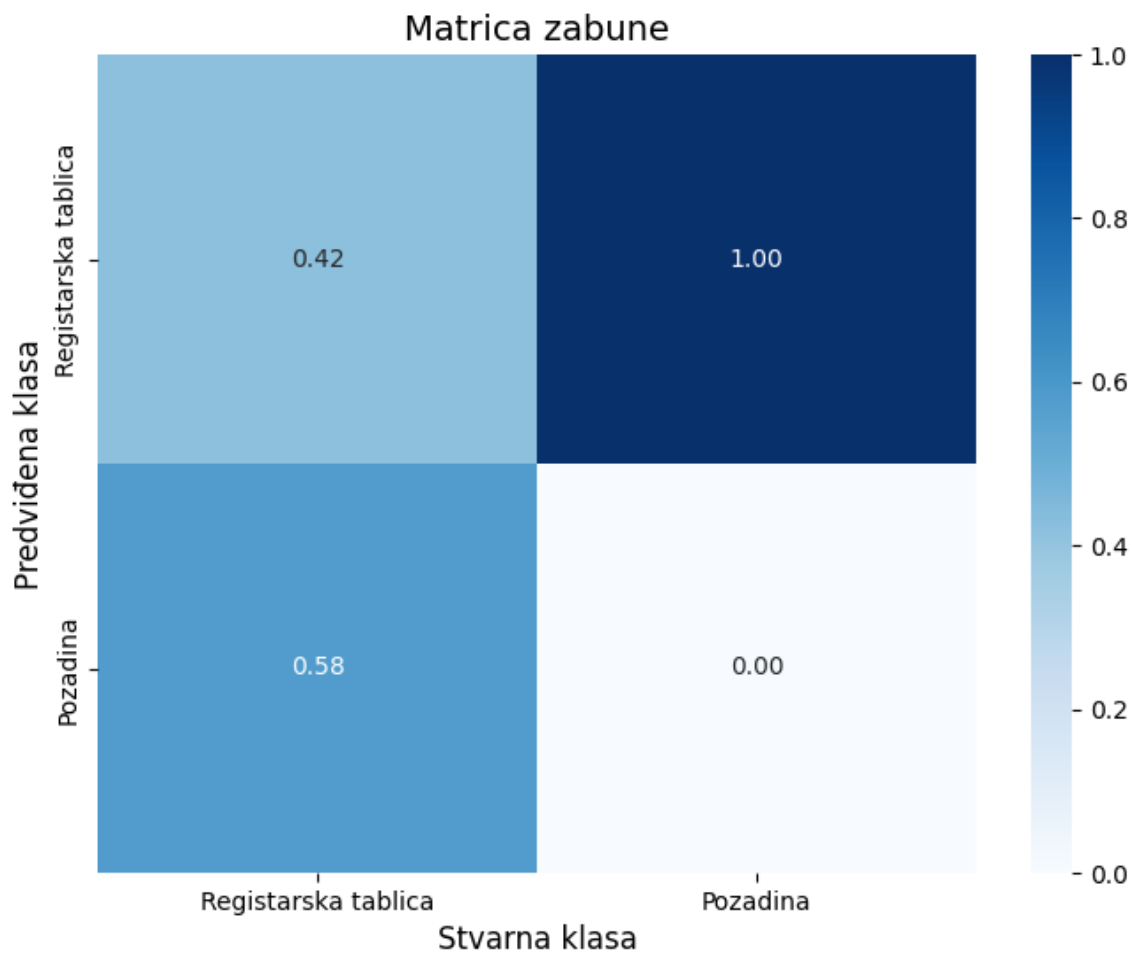


Slika 24. Prikaz jačine Gaussovog šuma

Izvor: Izradio autor

Nakon uspješno provedene validacije na slikama s Gausovim šumom, potrebno je provesti testiranje modela na istim slikama. Kao što je rečeno, koristi se model koji je dobiven treningom na čistim slikama.

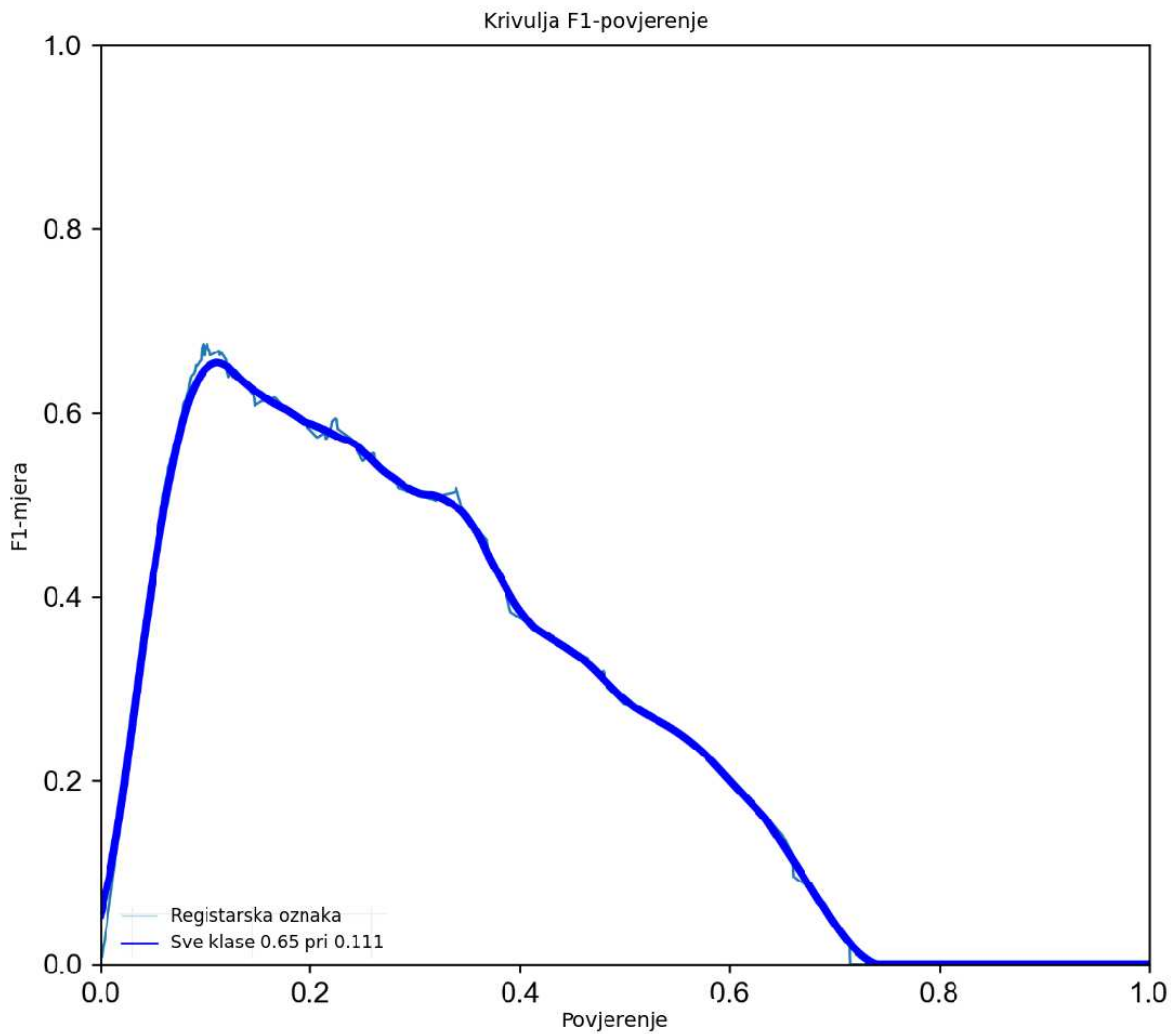
Pri testiranju slika s Gausovim šumom matrica zabune klasificira sljedeće rezultate. Udio slika s registarskim tablicama koje je model ispravno klasificirao iznosi 42%, dok je pogrešno klasificirao 58%. Zaključuje se kako pri testiranju modela u prisustvu visokog šuma model iako uspješno klasificira slike, isto tako ima problema kod klasifikacije određenog broja slika.



Slika 25. Matrica zabune za testiranje - Gaussov šum

Izvor: Izradio autor

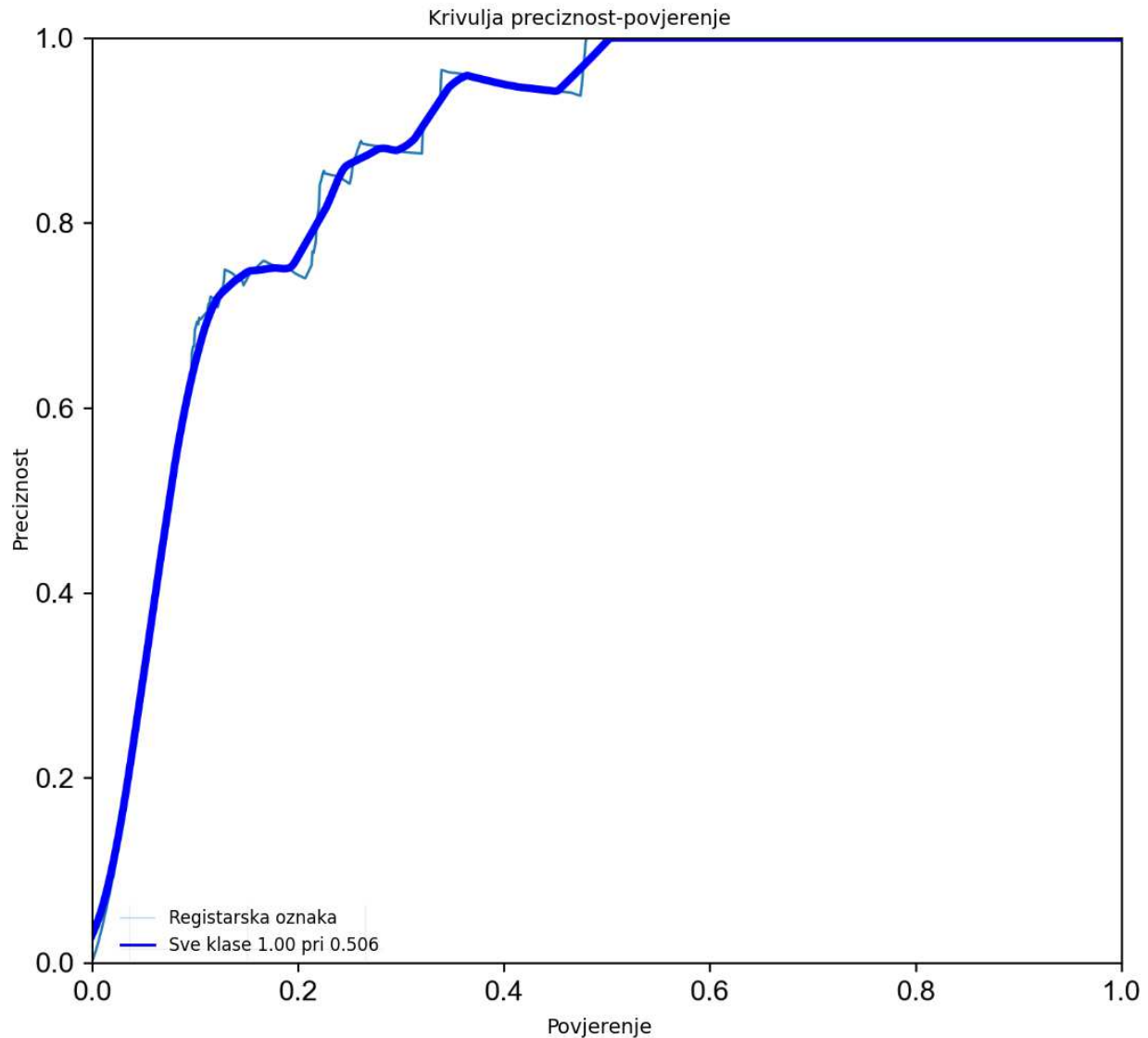
Ova slika prikazuje F1-povjerenje krivulju za testiranje modela na slikama s Gausovim šumom. Maksimalna F1-mjera iznosi 0.65 pri pragu povjerenja 0.111, što ukazuje na kompromis između preciznosti i odziva. Kako se prag povjerenja povećava, model postaje sigurniji, ali smanjuje broj uspješnih detekcija, što dovodi do pada F1-mjere. Gaussov šum negativno utječe na performanse modela, smanjujući njegovu sposobnost prepoznavanja registarskih oznaka u otežanim uvjetima.



Slika 26. Krivulja f1 - povjerenje pri testiranju

Izvor: Izradio autor

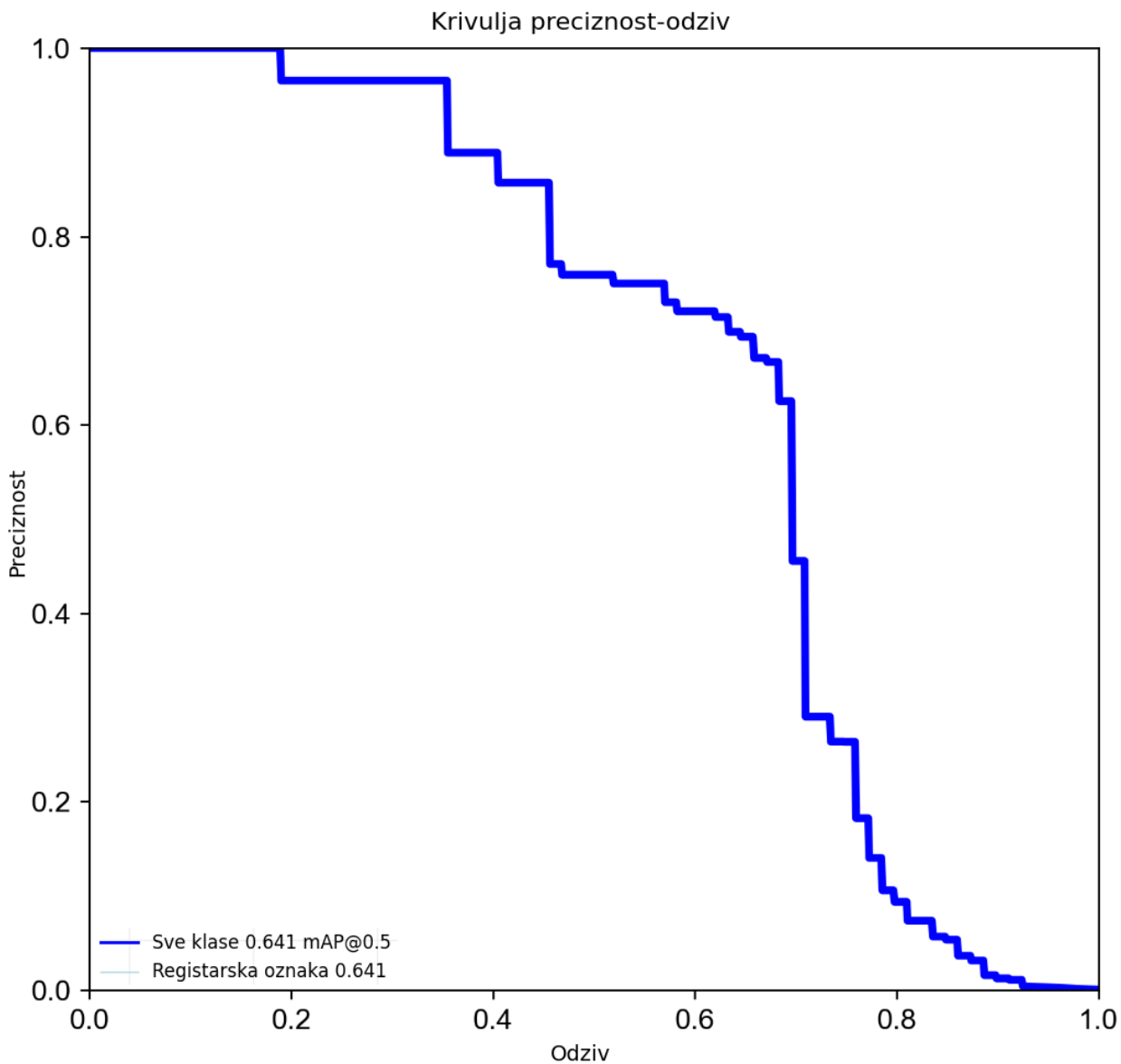
Krivulja preciznost-povjerenje prikazuje odnos između preciznosti modela i povjerenja detekcije na testnim podacima s Gausovim šumom. Maksimalna preciznost od 1.00 postiže se pri povjerenju 0.506, što sugerira da model može točno klasificirati većinu pozitivnih primjera pri višim vrijednostima povjerenja. Međutim, za niže pragove povjerenja, preciznost varira zbog povećane nesigurnosti detekcija.



Slika 27. Krivulja preciznost – povjerenje pri testiranju šuma

Izvor: Izradio autor

Ova krivulja preciznost-odziv prikazuje performanse modela tijekom testiranja na slikama s Gausovim šumom. Prosječna preciznost pri IoU pragu od 0.5 (mAP@0.5) iznosi 0.641, što ukazuje na smanjenu točnost modela u uvjetima povećanog šuma. Vidljivo je da model zadržava visoku preciznost pri nižim vrijednostima odziva, ali preciznost značajno opada kako se povećava odziv, što sugerira poteškoće u otkrivanju registarskih oznaka u otežanim uvjetima.

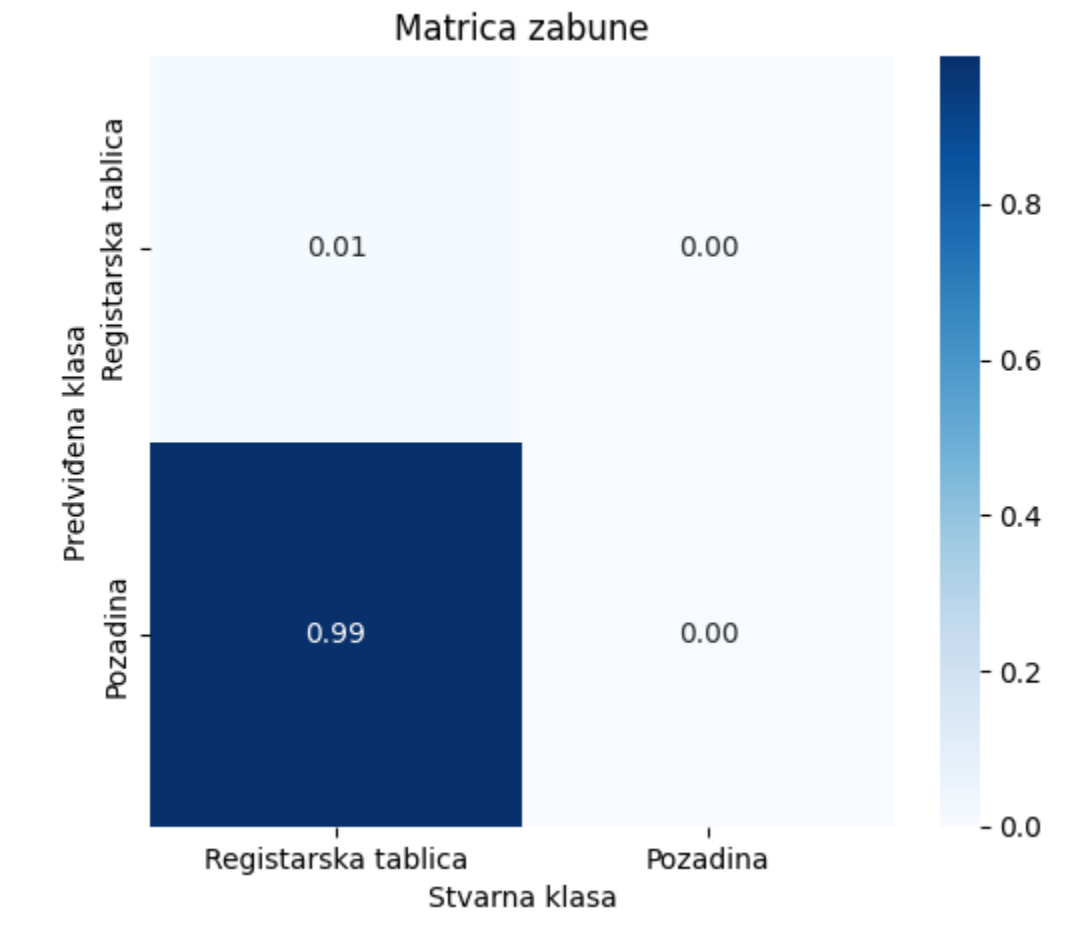


Slika 28. Krivulja preciznost - odziv pri testiranju šuma

Izvor: Izradio autor

Daljnja testiranja i evaluacija provest će se na prije spomenutom *salt & pepper* šumu, postotak šuma koji je manualno postavljen iznosi 0.1 (10 %) po slici, omjer soli i papra je 0.5 što ukazuje da će broj zrnca biti isti. Model za validaciju i testiranje koristimo onakav kakav je dobiven treniranjem na čistim slikama.

Ova matrica zabune prikazuje rezultate validacije modela na slikama s dodanim *salt & pepper* šumom. Model je točno predvidio klasu "Registarska tablica" u samo 1% slučajeva, dok je 99% stvarnih registarskih tablica pogrešno klasificirano kao "Pozadina". Ovi rezultati ukazuju na značajno smanjenje performansi modela u uvjetima visoke razine šuma, što potvrđuje osjetljivost modela na degradaciju kvalitete slike.



Slika 29. Matrica zabune - validacija kod sp šuma

Izvor: Izradio autor

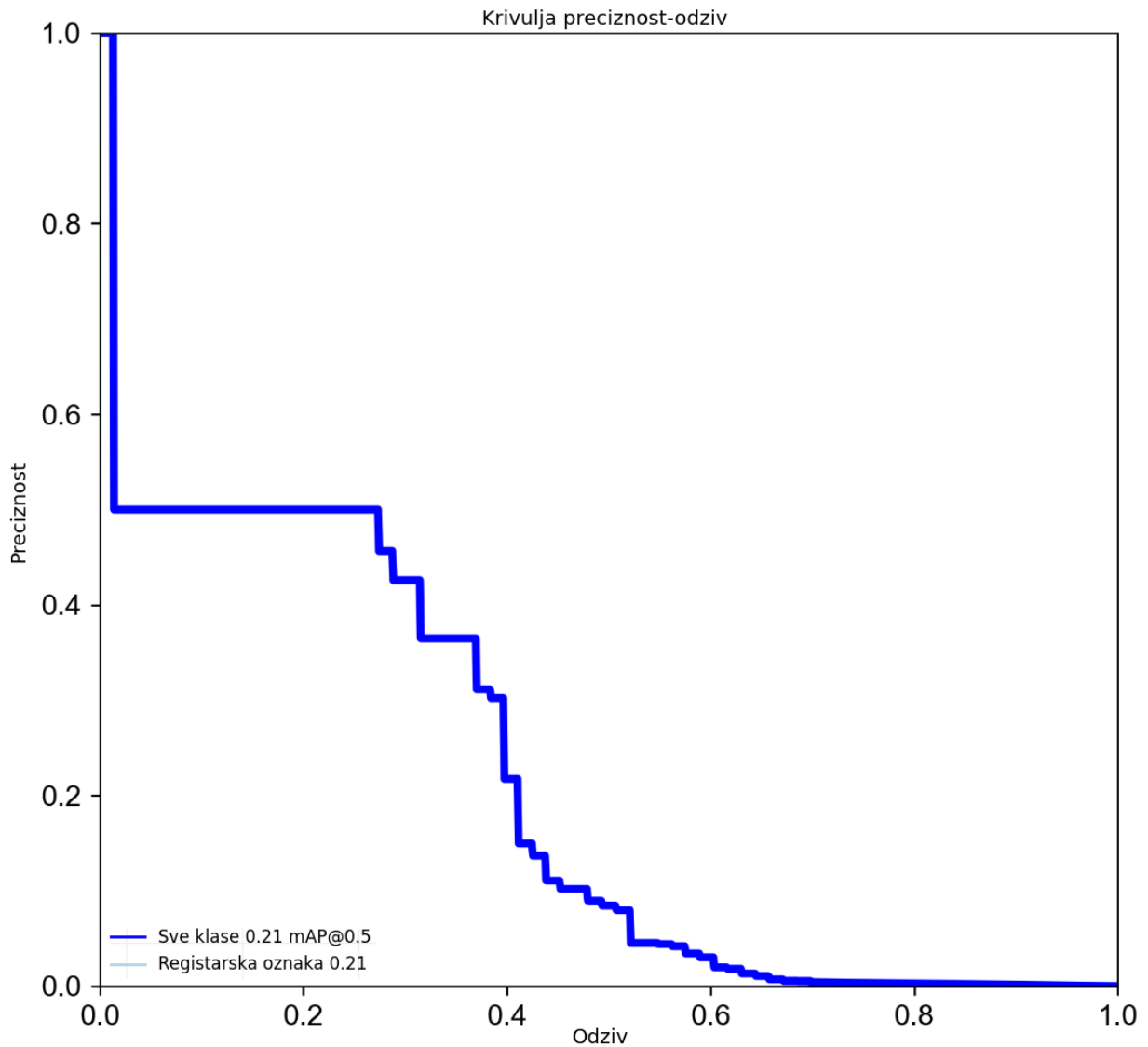
Sljedeća slika jasno prikazuje izazove koje šumovi postavljaju pred algoritme detekcije, te ujedno ukazuje na učinkovitost modela u takvim uvjetima. Dodavanje *salt & pepper* šuma simulira potencijalno najteže uvjete u kojima se može kamera pronaći.



Slika 30. Prikaz jačine salt & pepper šuma

Izvor: Izradio autor

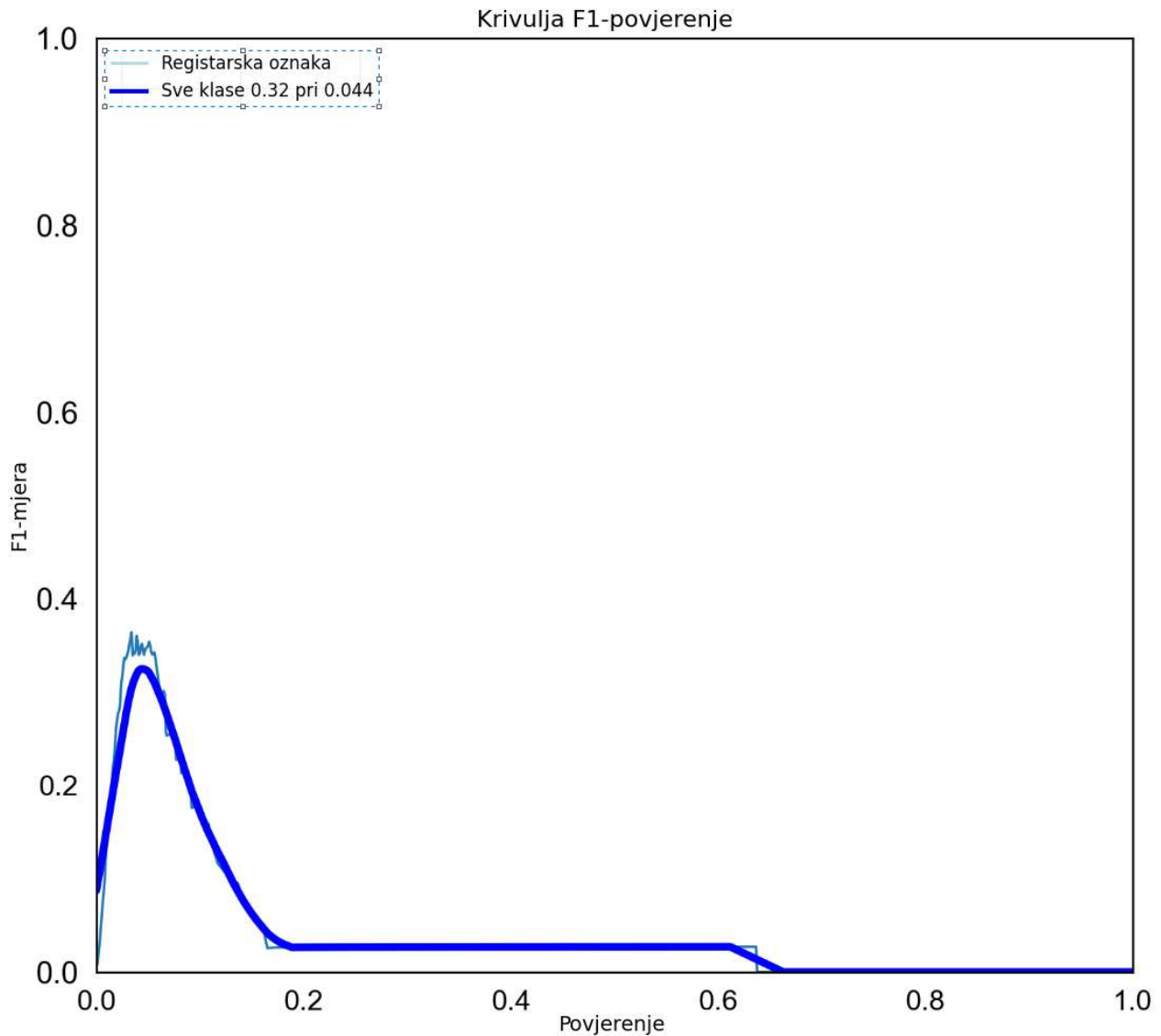
Dodani *salt & pepper* šum prikazuje velik utjecaj na performance modela, što se primjećuje iz smanjenja preciznosti i odziva. Maksimalne vrijednosti iznose 0.21 pri mAP@0.5.



Slika 31. Krivulja preciznost - odziv pri validaciji (salt & pepper šum)

Izvor: Izradio autor

Sljedeća krivulja prikazuje značajan pad performansi modela pri testiranju na slikama sa dodanim šumom. Maksimalna f1 – mjera od 0.32 postiže se pri vrlo niskom pragu od 0.044.

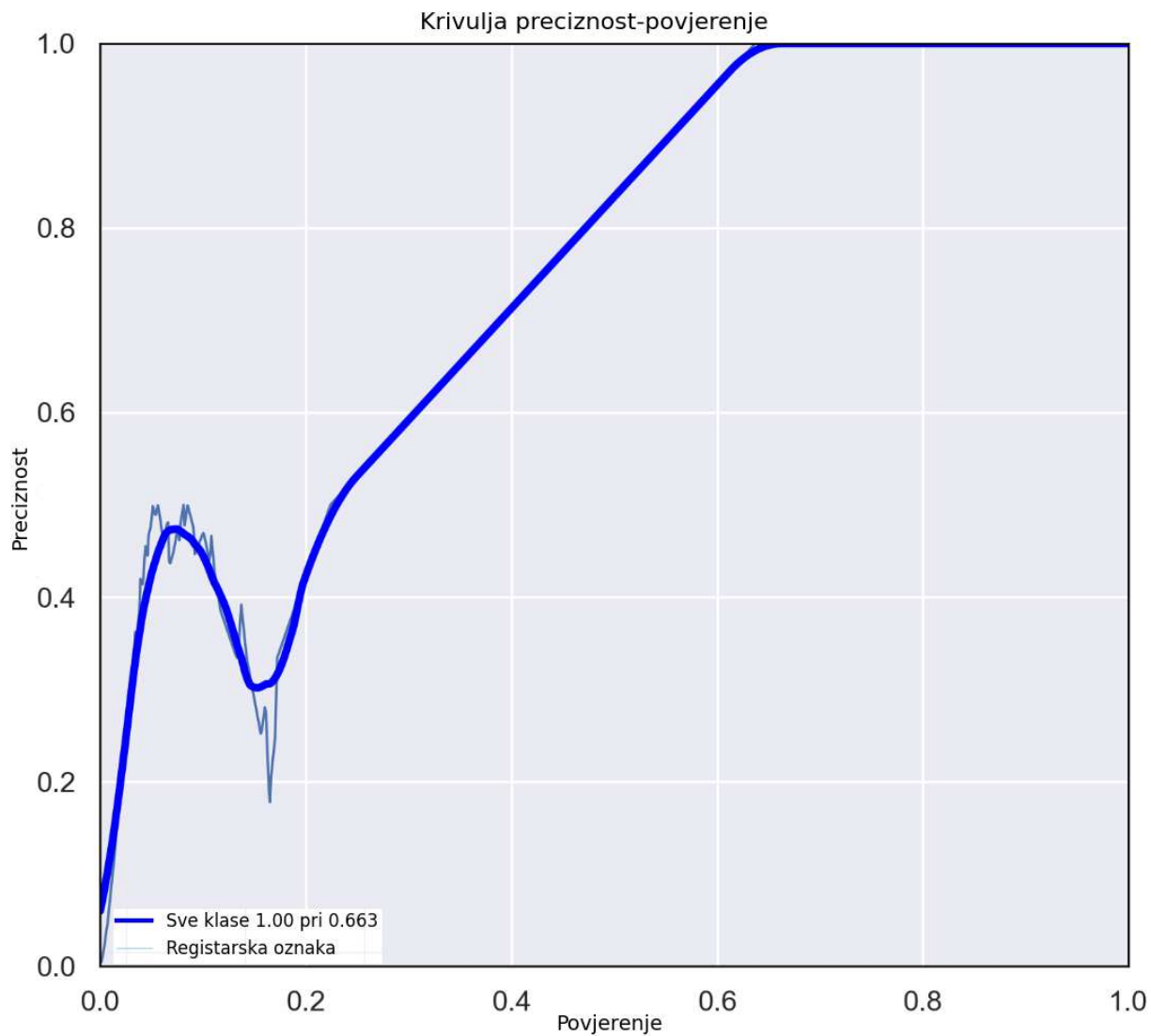


Slika 32. Krivulja f1 - povjerenje validacija (salt & pepper šum)

Izvor: Izradio autor

Ovaj graf prikazuje krivulju preciznosti u odnosu na prag povjerenja na validacijskom skupu s dodanim *salt & pepper* šumom. Vidljivo je da model postiže maksimalnu preciznost od 1.00 pri povjerenju od 0.663, no u nižim pragovima dolazi do oscilacija i

pada preciznosti. To sugerira da je model osjetljiv na šum u podacima te da preciznost značajno varira ovisno o postavljenom pragu povjerenja.

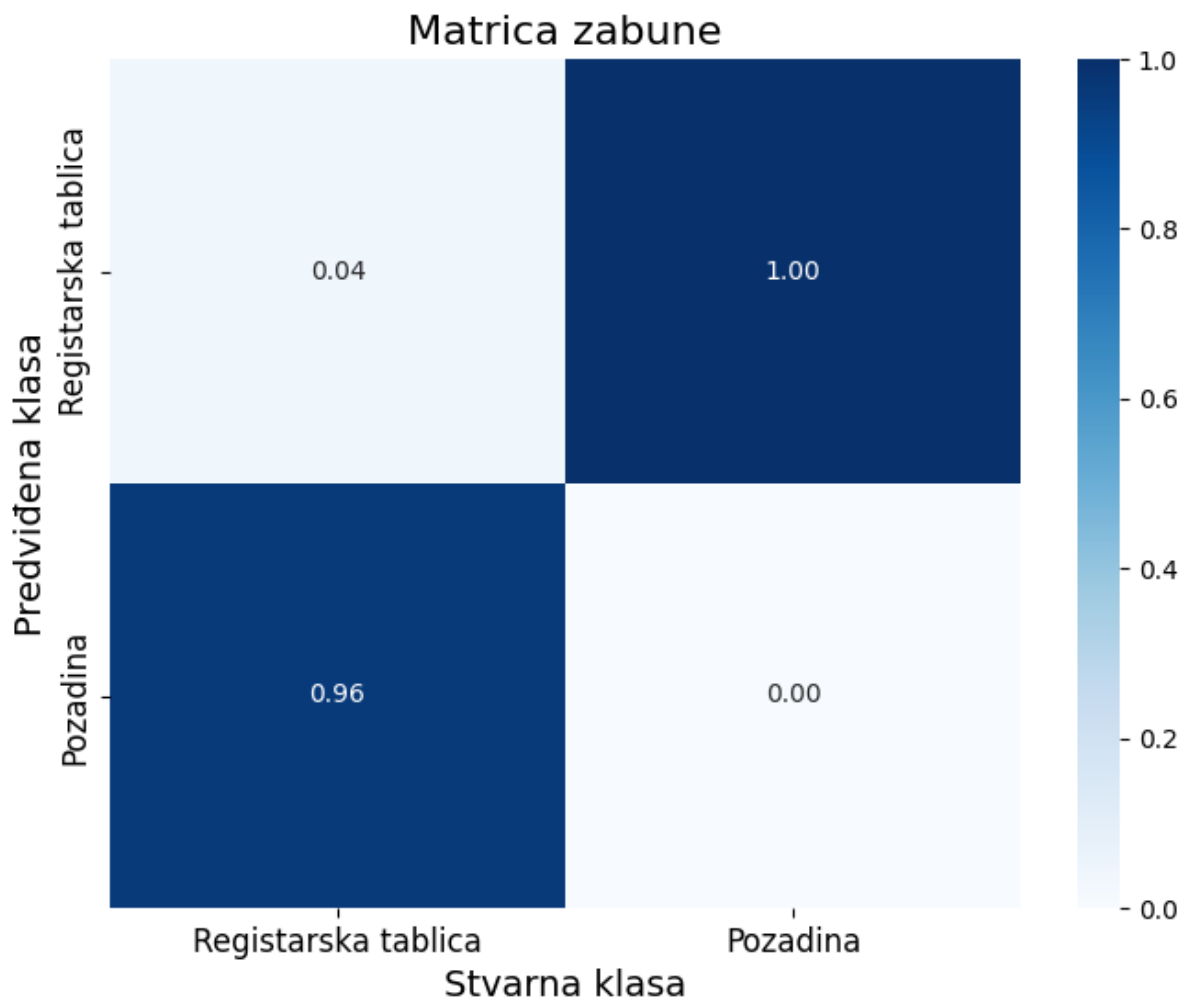


Slika 33. Krivulja preciznost- povjerenje (salt & pepper šum)

Izvor: Izradio autor

Sljedeća matrica zabune prikazuje performanse modela pri testiranju na slikama s dodanim *salt & pepper* šumom. Model klasificira registarske oznake u samo 4% slučajeva,

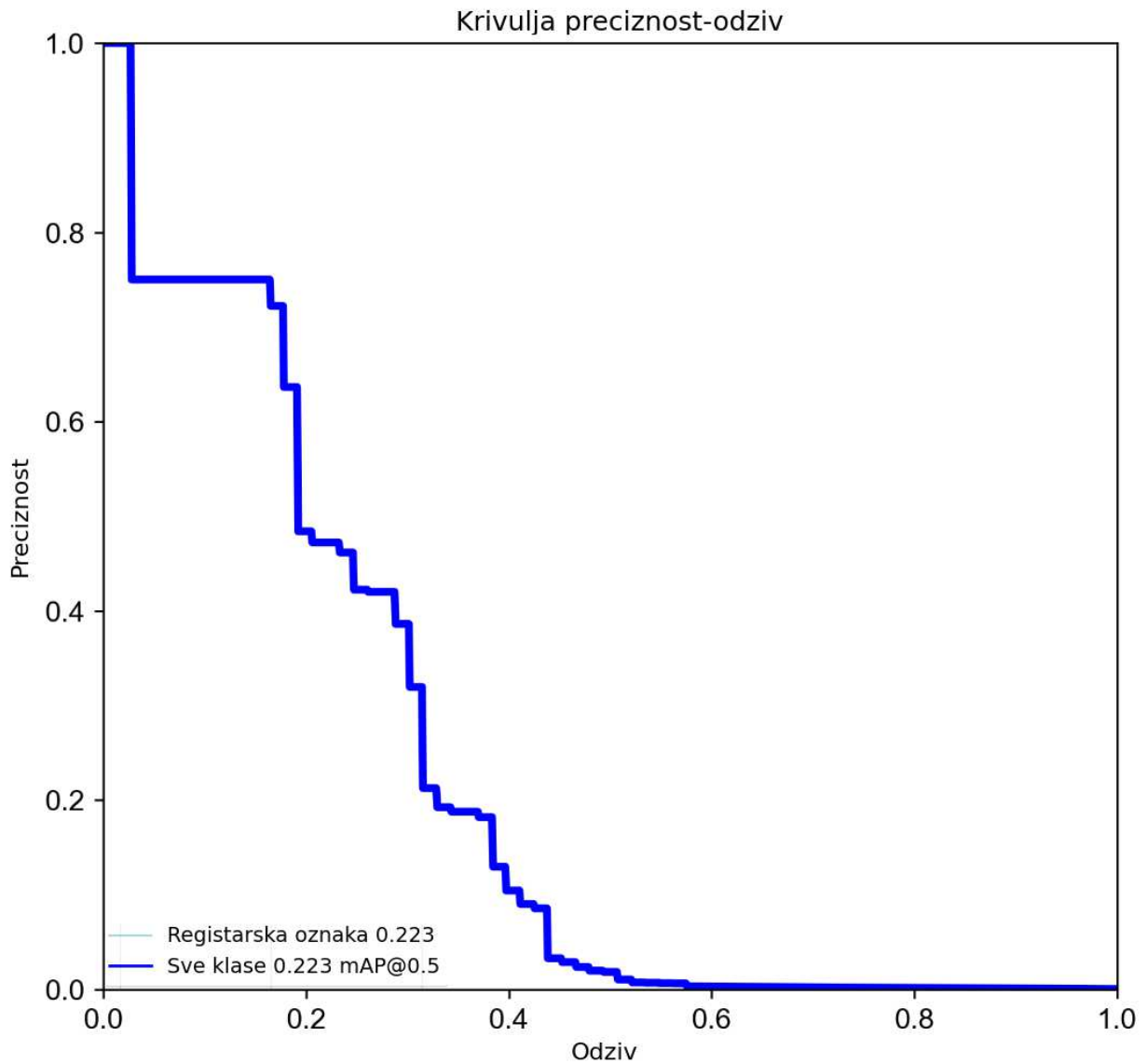
dok ih u 96% slučajeva pogrešno označava kao pozadinu. Navedeni rezultati ukazuju kako je model izrazito osjetljiv na *salt & pepper* šum.



Slika 34. Matrica zabune - testiranje (*salt & pepper* šum)

Izvor: Izradio autor

Krivulja preciznost-odziv na testnim podacima s dodatnim *salt & pepper* šumom pokazuje značajan pad performansi modela. Preciznost brzo opada s porastom odziva, što ukazuje na poteškoće modela u prepoznavanju registarskih oznaka u uvjetima visokog šuma

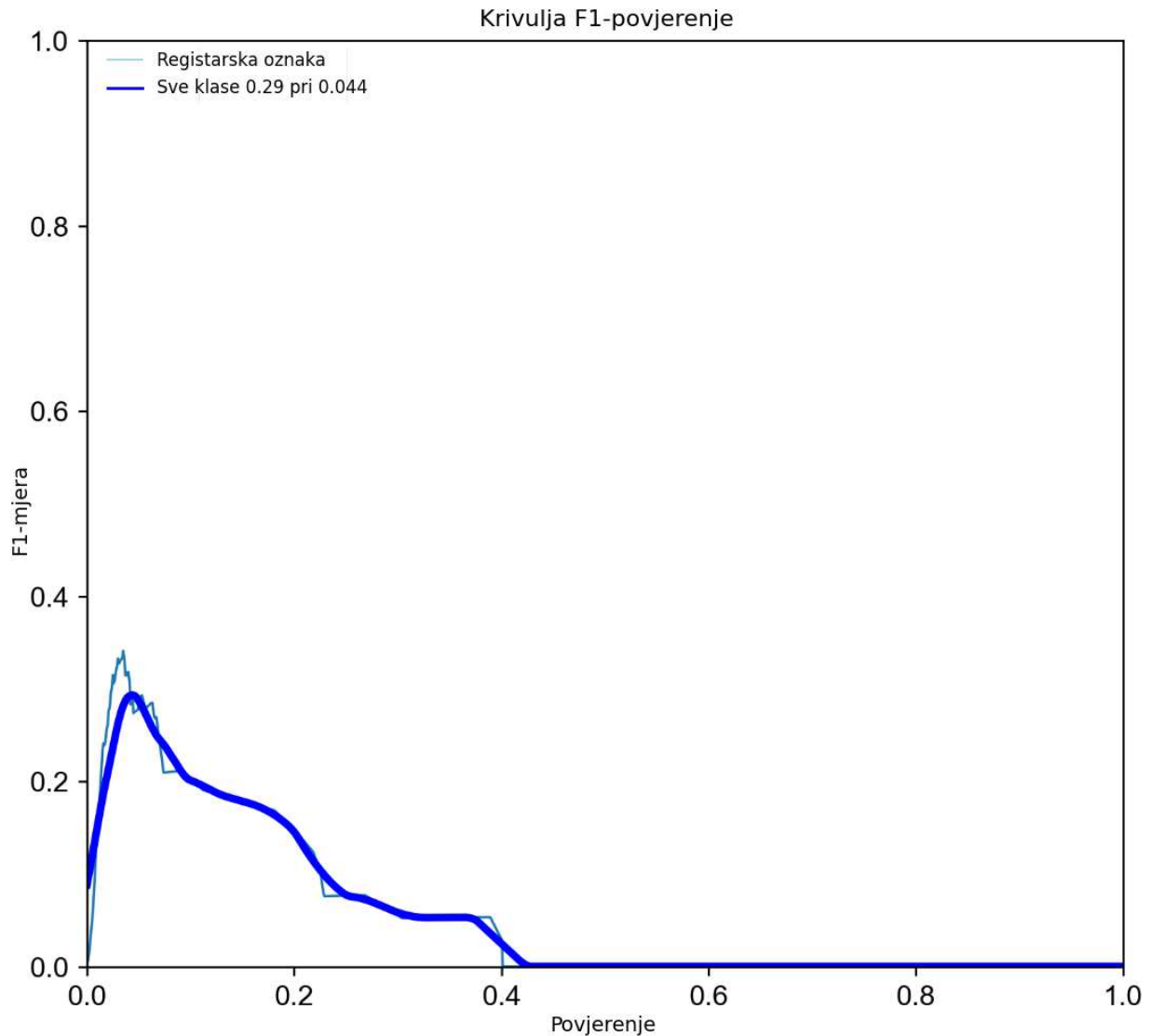


Slika 35. Krivulja preciznost - odziv - testiranje (salt & pepper šumom)

Izvor: Izradio autor

Ova krivulja F1-mjere u odnosu na povjerenje prikazuje kako se performanse modela smanjuju na testnim podacima sa *salt & pepper* šumom. Maksimalna F1-mjera iznosi 0.29 pri povjerenju od 0.044, što ukazuje na značajno smanjenje prepoznavanja registarskih oznaka u uvjetima visokog šuma. Model postiže najbolje rezultate pri niskim

vrijednostima povjerenja, dok se F1-mjera naglo smanjuje za veće vrijednosti povjerenja.

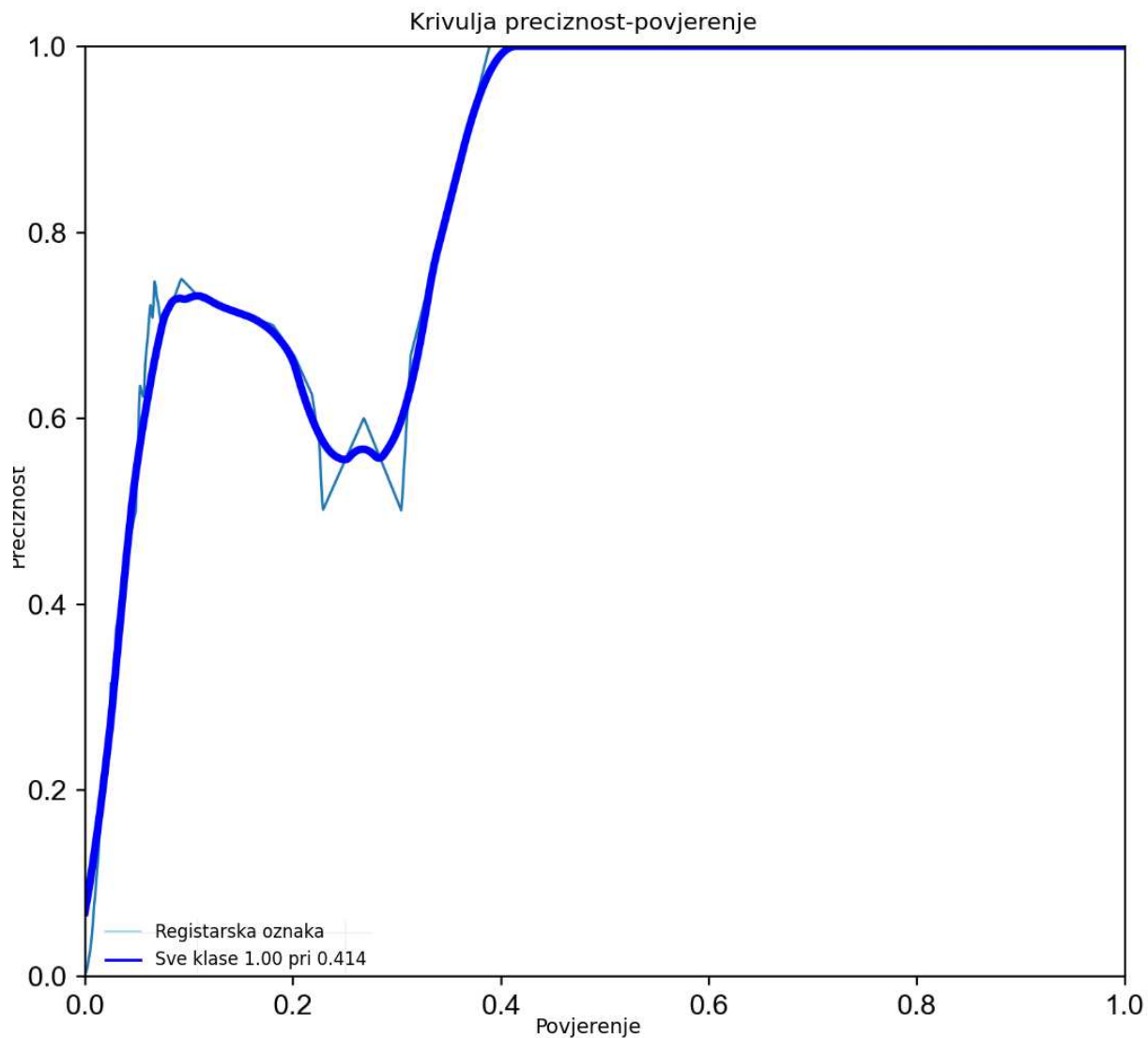


Slika 36. Krivulja f1 - povjerenje (testiranje - salt & pepper šum)

Izvor: Izradio autor

Krivulja preciznost-povjerenje prikazuje odnos između povjerenja modela i njegove preciznosti. Vidljivo je da se preciznost povećava s višim pragovima povjerenja, dostižući maksimalnu vrijednost pri 0.414. Međutim, oscilacije pri nižim vrijednostima

povjerenja ukazuju na nesigurnost modela pri donošenju odluka u uvjetima visoke nesigurnosti.

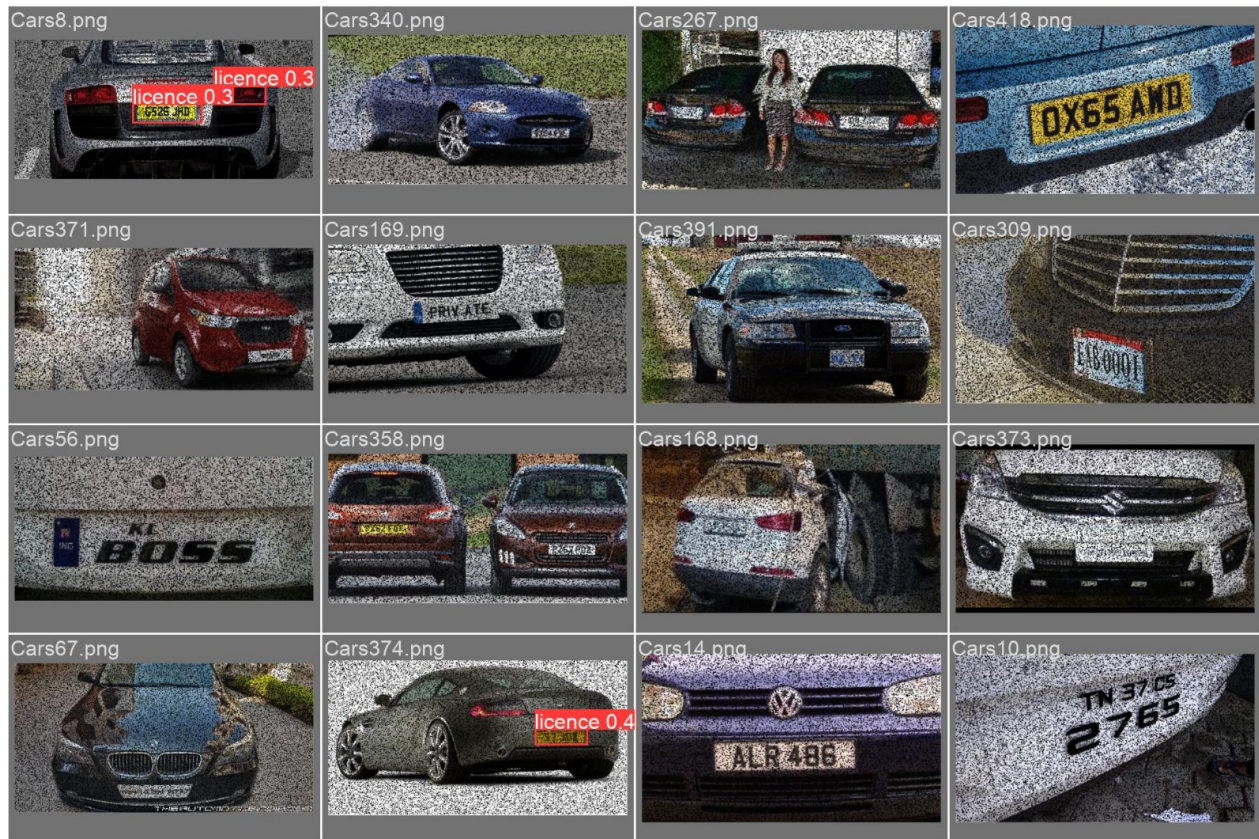


Slika 37. Krivulja preciznost - povjerenje (testiranje - salt & pepper šum)

Izvor: Izradio autor

Usporedba s *ground truth* oznakama pokazuje da model ima problema s točnim lociranjem registarskih oznaka na degradiranim slikama, što je posljedica visokog stupnja šuma. U mnogim slučajevima povjerenje u predikcije je nisko (npr. 0.3 ili 0.4),

što ukazuje na nesigurnost modela prilikom detekcije. Ovi rezultati pokazuju da *salt & pepper* šum značajno smanjuje performanse modela, što se također vidi u sniženim vrijednostima preciznosti i F1-mjere analiziranim kroz evaluacijske metrike.



Slika 38. Prepoznavanje slike s salt & pepper šumom

Izvor: Izradio autor

Kod Yolo algoritma, utjecaj šuma na performanse je uvelike primjetan, dok se model uspije koliko toliko držati uspješnih rezultata kod Gaussovog šuma, *dodavanjem salt & pepper* šuma rezultati metrika uvelike padaju te model ima puno lošiju generalnu sliku.

4.4. Implementacija Faster R-CNN modela za detekciju registarskih oznaka

Faster R-CNN (engl. *Regional Convolutional Neural Network*) jedan je od najnaprednijih algoritama za detekciju objekata, koji kombinira visoku preciznost i efikasnost zahvaljujući vlastitoj arhitekturi koja se temelji na dubokim konvolucijskim mrežama.

Faster R-CNN se pretežito sastoji od dvije primarne komponente: bazne konvolucijske mreže koja izvlači značajke iz slike i RPN mreže koja generira potencijalne objekte.

Naposljetku, ROI (engl. *Region Of Interest*) *pooling* prilagođava regije različitih veličina kako bi ih klasifikator i regresor mogli precizno obraditi. U ovome radu, navedena mreža je implementirana za detekciju registarskih oznaka, gdje je model treniran na čistim slikama te je kasnije validiran i testiran na slikama s Gaussovim i *salt & pepper* šumom.

Navedeni dio koda omogućuje montiranje *drive*-a kako bi se moglo pristupiti datasetu i rezultatima treninga. Učitavaju se standardne biblioteke te se uvozi *Faster R-CNN*. Glavni dio koda implementira klasu `CarsDatasetYolo`, koja omogućuje učitavanje YOLO anotacija spremljenih u .txt formatu. Klasa parsira *bounding box* koordinate, pretvara ih u apsolutne vrijednosti i vraća slike zajedno s odgovarajućim oznakama.

```
import os
from google.colab import drive

drive.mount('/content/drive')

results_folder =
"/content/drive/MyDrive/Diplomski/FasterRCNN/clean_results"
os.makedirs(results_folder, exist_ok=True)

!pip install --upgrade torch torchvision matplotlib

# 1) Import biblioteka

import torch
import torch.nn as nn
import torch.optim as optim
```

```

import matplotlib.pyplot as plt
from PIL import Image
import xml.etree.ElementTree as ET
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as T
import torchvision.transforms.functional as F
import numpy as np

import os
import math
import shutil

from torchvision.utils import draw_bounding_boxes

from torchvision.models.detection.faster_rcnn import (
    fasterrcnn_resnet50_fpn,
    FastRCNNPredictor
)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 2) Dataset za train (YOLO .txt)

class CarsDatasetYolo(Dataset):
    """
    Čita YOLO .txt ( <class> <x_c> <y_c> <w> <h> ), normalizirano
    [0..1].
    Ako je .txt prazan => box=[0,4], negative sample.
    """
    def __init__(self, root, transforms=None):
        self.root = root
        self.transforms = transforms
        self.img_dir = os.path.join(root, "JPEGImages")
        self.label_dir = os.path.join(root, "labels")

        self.images = []
        for fname in sorted(os.listdir(self.img_dir)):
            if fname.lower().endswith((".png", ".jpg", ".jpeg")):
                label_name = os.path.splitext(fname)[0] + ".txt"
                label_path = os.path.join(self.label_dir, label_name)
                if os.path.exists(label_path):
                    self.images.append(fname)

```

```

def __getitem__(self, idx):
    img_name = self.images[idx]
    img_path = os.path.join(self.img_dir, img_name)
    img = Image.open(img_path).convert("RGB")
    w, h = img.size

    label_name = os.path.splitext(img_name)[0] + ".txt"
    label_path = os.path.join(self.label_dir, label_name)

    boxes_list = []
    labels_list = []

    with open(label_path, "r") as f:
        lines = [ln.strip() for ln in f if ln.strip()]

    for line in lines:
        parts = line.split()
        cls = int(parts[0])
        x_c = float(parts[1]) * w
        y_c = float(parts[2]) * h
        w_abs = float(parts[3]) * w
        h_abs = float(parts[4]) * h

        xmin = x_c - w_abs/2
        xmax = x_c + w_abs/2
        ymin = y_c - h_abs/2
        ymax = y_c + h_abs/2

        # Jedna klasa => 0 => label=1
        labels_list.append(cls + 1)
        boxes_list.append([xmin, ymin, xmax, ymax])

    if len(boxes_list) == 0:
        boxes = torch.zeros((0,4), dtype=torch.float32)
        labels = torch.zeros((0,), dtype=torch.int64)
    else:
        boxes = torch.tensor(boxes_list, dtype=torch.float32)
        labels = torch.tensor(labels_list, dtype=torch.int64)

    target = {"boxes": boxes, "labels": labels}

    if self.transforms:
        img = self.transforms(img)
    return img, target

```

```

def __len__(self):
    return len(self.images)

with open(label_path, "r") as f:
    lines = [ln.strip() for ln in f if ln.strip()]

for line in lines:
    parts = line.split()
    cls = int(parts[0])
    x_c = float(parts[1]) * w
    y_c = float(parts[2]) * h
    w_abs = float(parts[3]) * w
    h_abs = float(parts[4]) * h

    xmin = x_c - w_abs/2
    xmax = x_c + w_abs/2
    ymin = y_c - h_abs/2
    ymax = y_c + h_abs/2

    # 0 => label=1
    labels_list.append(cls + 1)
    boxes_list.append([xmin, ymin, xmax, ymax])

if len(boxes_list) == 0:
    boxes = torch.zeros((0,4), dtype=torch.float32)
    labels = torch.zeros((0,), dtype=torch.int64)
else:
    boxes = torch.tensor(boxes_list, dtype=torch.float32)
    labels = torch.tensor(labels_list, dtype=torch.int64)

target = {"boxes": boxes, "labels": labels}

if self.transforms:
    img = self.transforms(img)
return img, target

def __len__(self):
    return len(self.images)

```

Slika 39. Montiranje i priprema okruženja

Izvor: Izradio autor

Sljedeći dio koda definira klasu `VocTestDataset`, koja omogućuje učitavanje iz testnog skupa podataka iz *Pascal* VOC formata. Ako slika nema anotacija, smatra se negativnim uzorkom i dodjeljuje joj se prazan bounding box.

```
#####
# 3) Dataset za test (VOC .xml)
#####
class VocTestDataset(Dataset):
    """
    Čita Pascal VOC .xml ( <object><bndbox> ... ), klasa => 1
    Ako nema obj. => negative => [0,4].
    """
    def __init__(self, root, transforms=None):
        self.root = root
        self.transforms = transforms
        self.img_dir = os.path.join(root, "JPEGImages")
        self.ann_dir = os.path.join(root, "Annotations")

        self.images = []
        for fname in sorted(os.listdir(self.img_dir)):
            if fname.lower().endswith((".png", ".jpg", ".jpeg")):
                xml_name = os.path.splitext(fname)[0] + ".xml"
                xml_path = os.path.join(self.ann_dir, xml_name)
                if os.path.exists(xml_path):
                    self.images.append(fname)

    def __getitem__(self, idx):
        img_name = self.images[idx]
        img_path = os.path.join(self.img_dir, img_name)
        img = Image.open(img_path).convert("RGB")

        xml_name = os.path.splitext(img_name)[0] + ".xml"
        xml_path = os.path.join(self.ann_dir, xml_name)

        boxes_list = []
        labels_list = []

        tree = ET.parse(xml_path)
        root = tree.getroot()

        for obj in root.findall("object"):
            bnd = obj.find("bndbox")
            xmin = float(bnd.find("xmin").text)
```

```

    ymin = float(bnd.find("ymin").text)
    xmax = float(bnd.find("xmax").text)
    ymax = float(bnd.find("ymax").text)

    labels_list.append(1)
    boxes_list.append([xmin, ymin, xmax, ymax])

if len(boxes_list) == 0:
    boxes = torch.zeros((0,4), dtype=torch.float32)
    labels = torch.zeros((0,), dtype=torch.int64)
else:
    boxes = torch.tensor(boxes_list, dtype=torch.float32)
    labels = torch.tensor(labels_list, dtype=torch.int64)

target = {"boxes": boxes, "labels": labels}

if self.transforms:
    img = self.transforms(img)
return img, target

def __len__(self):
    return len(self.images)

```

Slika 40. Učitavanje iz testnog skupa

Izvor: Izradio autor

Za detekciju koristi se Faster R-CNN model s unaprijed naučenim težinama. Model se prilagođava za prepoznavanje samo jedne klase objekata, uz pozadinsku klasu.

```

model = fasterrcnn_resnet50_fpn(weights="DEFAULT")
num_classes = 2 # 1 klasa + background
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
num_classes)
model.to(device)

```

Slika 41. Inicijalizacija Faster R-CNN modela

Izvor: Izradio autor

Model se trenira kroz 15 epoha koristeći SGD optimizator s momentumom i regulizacijom. Svaka epoha računa gubitak, a model se nakon treniranja sprema u označeni direktorij.

```
params = [p for p in model.parameters() if p.requires_grad]
optimizer = optim.SGD(params, lr=0.005, momentum=0.9,
weight_decay=0.0005)

num_epochs = 15
losses_per_epoch = []

for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0

    for images, targets in train_loader:
        igpu = [img.to(device) for img in images]
        tgpu = [{k:v.to(device) for k,v in t.items()} for t in targets]

        loss_dict = model(igpu, tgpu)
        losses = sum(loss for loss in loss_dict.values())

        optimizer.zero_grad()
        losses.backward()
        optimizer.step()

        total_loss += losses.item()

    avg_loss = total_loss / len(train_loader)
    losses_per_epoch.append(avg_loss)
    print(f"[{epoch+1}/{num_epochs}] Gubitak: {avg_loss:.4f}")

# Spremi trenirani model
saved_model_path = os.path.join(results_folder,
"fasterrcnn_custom.pth")
torch.save(model.state_dict(), saved_model_path)
print(f"Trening završen. Model spremljen u '{saved_model_path}'.")
```

Slika 42. Trening modela

Izvor: Izradio autor

Postavlja se radno okruženje za provođenje evaluacije i testiranja na slikama s Gaussovom šumom

```
from google.colab import drive
drive.mount('/content/drive')

import os

# Gdje je spremljeni model (fasterrcnn_custom.pth)
model_path =
"/content/drive/MyDrive/Diplomski/FasteRCNN/clean_results/fasterrcnn_custom.pth"

# Gdje se nalaze gauss slike/dataset:
#  gaussimg/test/ (images/, labels/)
#  gaussimg/val/ (images/, labels/)
gauss_root =
"/content/drive/MyDrive/Diplomski/FasteRCNN/dataset/gaussimg"

# Spremanje rezultata:
results_folder =
"/content/drive/MyDrive/Diplomski/FasteRCNN/gauss_results"
test_out_folder = os.path.join(results_folder, "test")
val_out_folder = os.path.join(results_folder, "val")
os.makedirs(test_out_folder, exist_ok=True)
os.makedirs(val_out_folder, exist_ok=True)

!pip install --upgrade torch torchvision matplotlib seaborn
```

Slika 43. Postavljanje okruženja za evaluaciju

Izvor: Izradio autor

Evaluacija modela provodi se izračunavanjem Map, IoU, preciznost, odziv i f1 – rezultata. Navedene metrike pokazuju nam uspješnost detekcije objekata kod slika s umjetno dodanim Gaussovom šumom.


```

metrics = evaluate_metrics(model, dl, iou_thresh, score_thresh)
    print(f" [Metrike] mAP={metrics['mAP']:.3f},
IoU={metrics['meanIoU']:.3f}, " \
        f"Prec={metrics['precision']:.3f},
Odziv={metrics['odziv']:.3f}, F1={metrics['F1']:.3f}")

```

Slika 44. Isječak koda za evaluaciju

Izvor: Izradio autor

Prethodno istrenirani Faster R-CNN model učitavamo u memoriju. Model je treniran da detektira jednu klasu (registarske oznake vozila), uz dodatnu klasu pozadinu.

```

#####
# 5) Učitavanje modela i evaluacija (test i val)
#####

model = fasterrcnn_resnet50_fpn(weights=None)
num_classes=2
in_feats = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_feats,num_classes)

model.load_state_dict(torch.load(model_path, map_location="cpu"))
model.to(device)
model.eval()
print("Učitani model:", model_path)

```

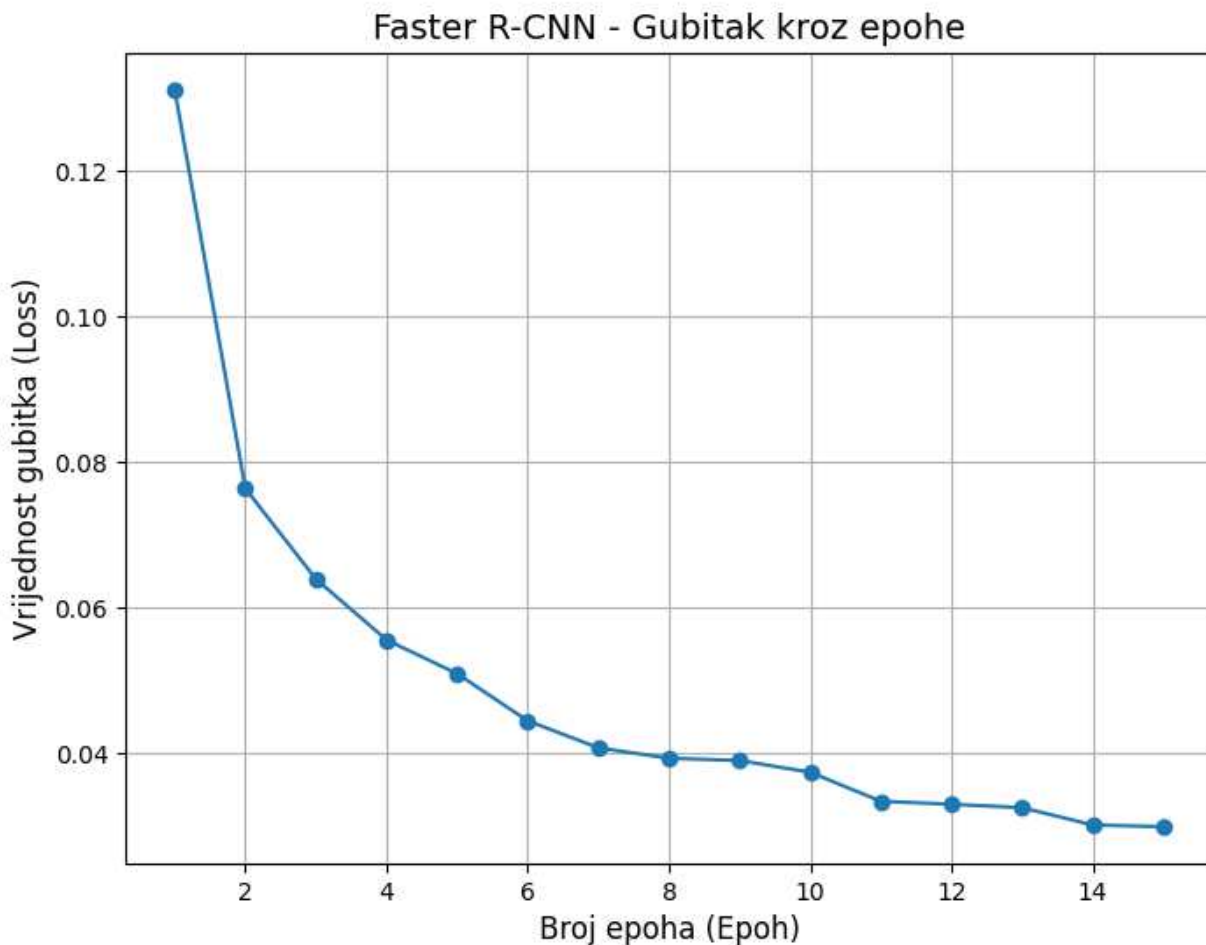
Slika 45. Slika. Učitavanje treniranog modela

Izvor: Izradio autor

Prikazani proces je identičan za *salt & pepper* šum, jedino što se razlikuje su putanje do slika i naravno, krajnji rezultat.

Na sljedećoj slici prikazano je smanjenje funkcije gubitka tijekom 15 epoha trening modela *Faster R-CNN*. Na samome početku gubitak iznosi 0.13, dok je na kraju trening pao ispod 0.04. Navedeni pad pokazuje na postupnu optimizaciju modela to jest težinskih parametara modela i njegovo postepeno prilagođavanje dobivenim podacima.

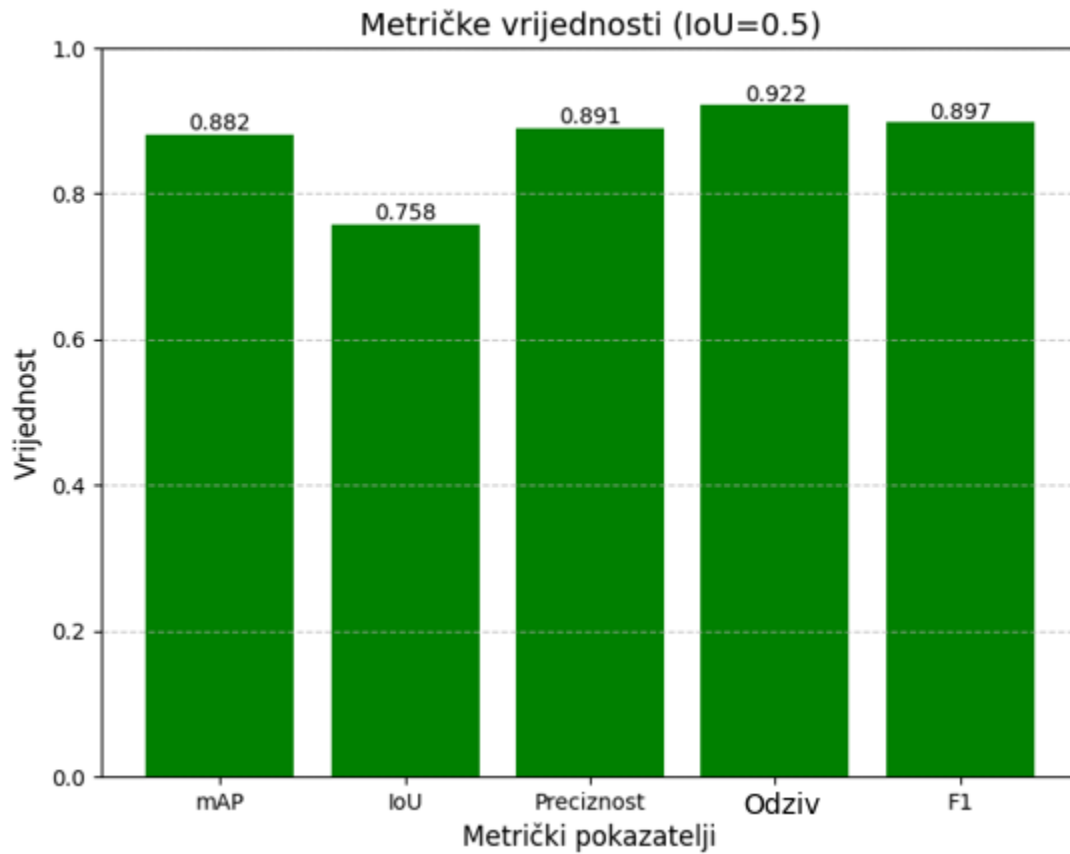
Krivulja prikazuje stabilizaciju pri kraju treninga, što sugerira da je model dosegnuo idealno stanje u kojem daljnje epohe ne bi značajno poboljšale performanse



Slika 46. Faster R-CNN gubitak kroz epohe

Izvor: Izradio autor

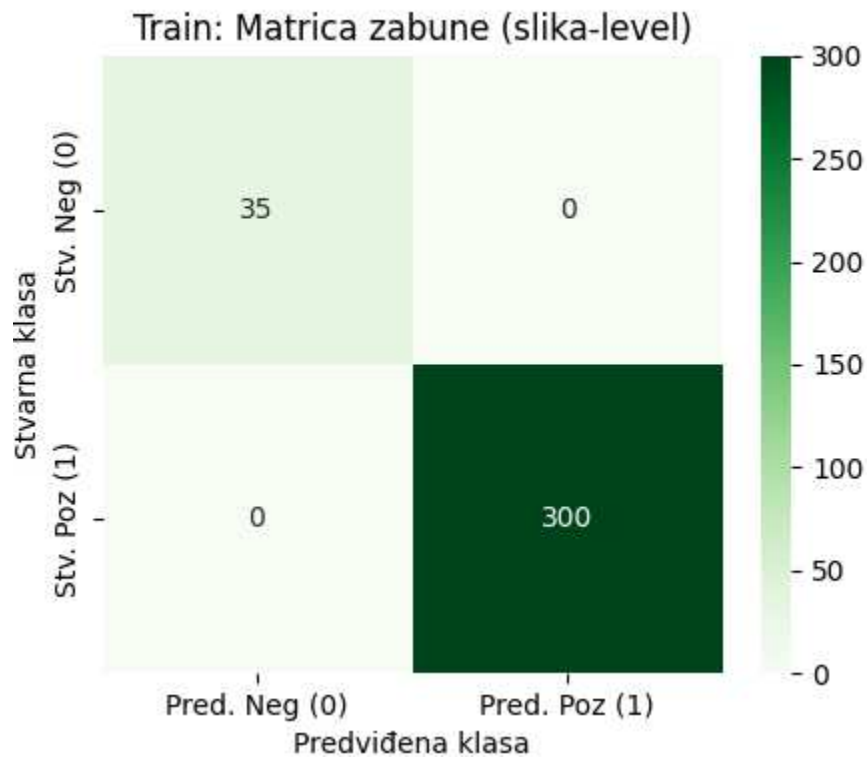
Graf prikazuje različite metričke pokazatelje, izvršena je evaluacija modela na testnom skupu (čiste slike). Model postiže mAP od 0.882 i IoU od 0.758, što ukazuje na vrlo dobro preklapanje predviđenih i stvarnih *bounding boxova*. Visoke vrijednosti preciznosti, odziva i f1 rezultata, potvrđuju učinkovito prepoznavanje modela s minimalnim pogreškama.



Slika 47. Metrički pokazatelji Faster R-CNN

Izvor: izradio autor

Matrica zabune pokazuje vrlo visoku pouzdanost modela pri detekciji registarskih oznaka u testnim podacima bez šuma. Model postiže savršenu klasifikaciju na navedenim slikama.

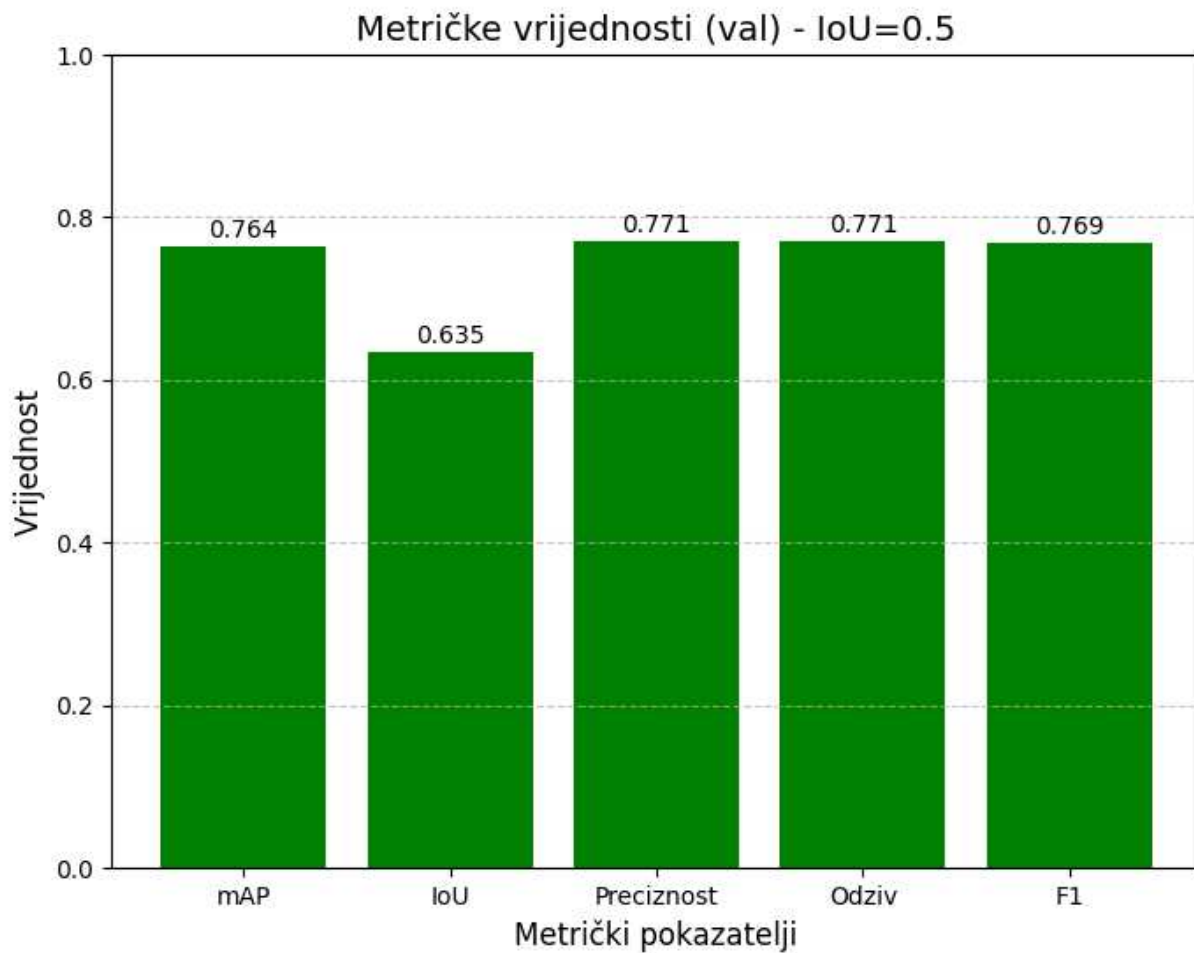


Slika 48. Matrica zabune (trening)

Izvor: Izradio autor

Sljedeći dobiveni rezultati odnose se na provedenu validaciju na skupu podataka na koji je dodan Gaussov šum, čime je testirana otpornost modela na degradaciju kvalitete slike.

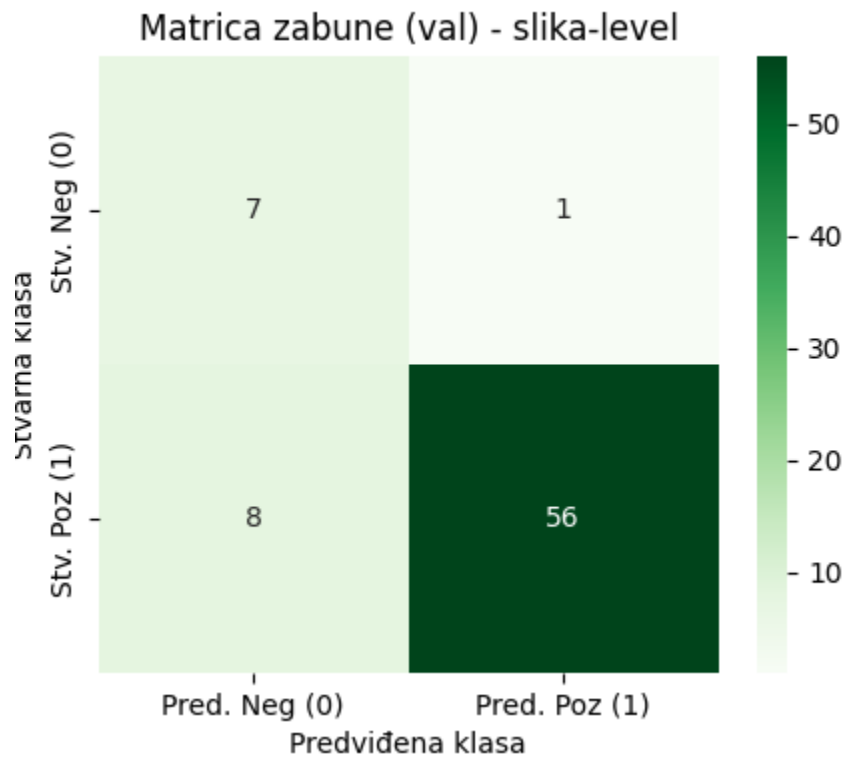
Validacijski rezultati na Gaussovom šumu prikazuju blagi pad performansi modela u odnosu na čisti testni skup, što se može pripisati smanjenoj vidljivosti registarskih oznaka uzrokovanoj Gaussovim šumom. mAP iznosi 0.764, a IoU je 0.635, što je i dalje u prihvatljivim granicama točnosti.



Slika 49. Metričke vrijednosti validacija (Gauss šum)

Izvor: Izradio autor

Kod matrice zabune uočava se povećan broj lažno negativnih slučajeva u odnosu na validaciju na čistim podacima, što ukazuje na to da model u nekim slučajevima nije uspio prepoznati registracijske oznake zbog šuma.

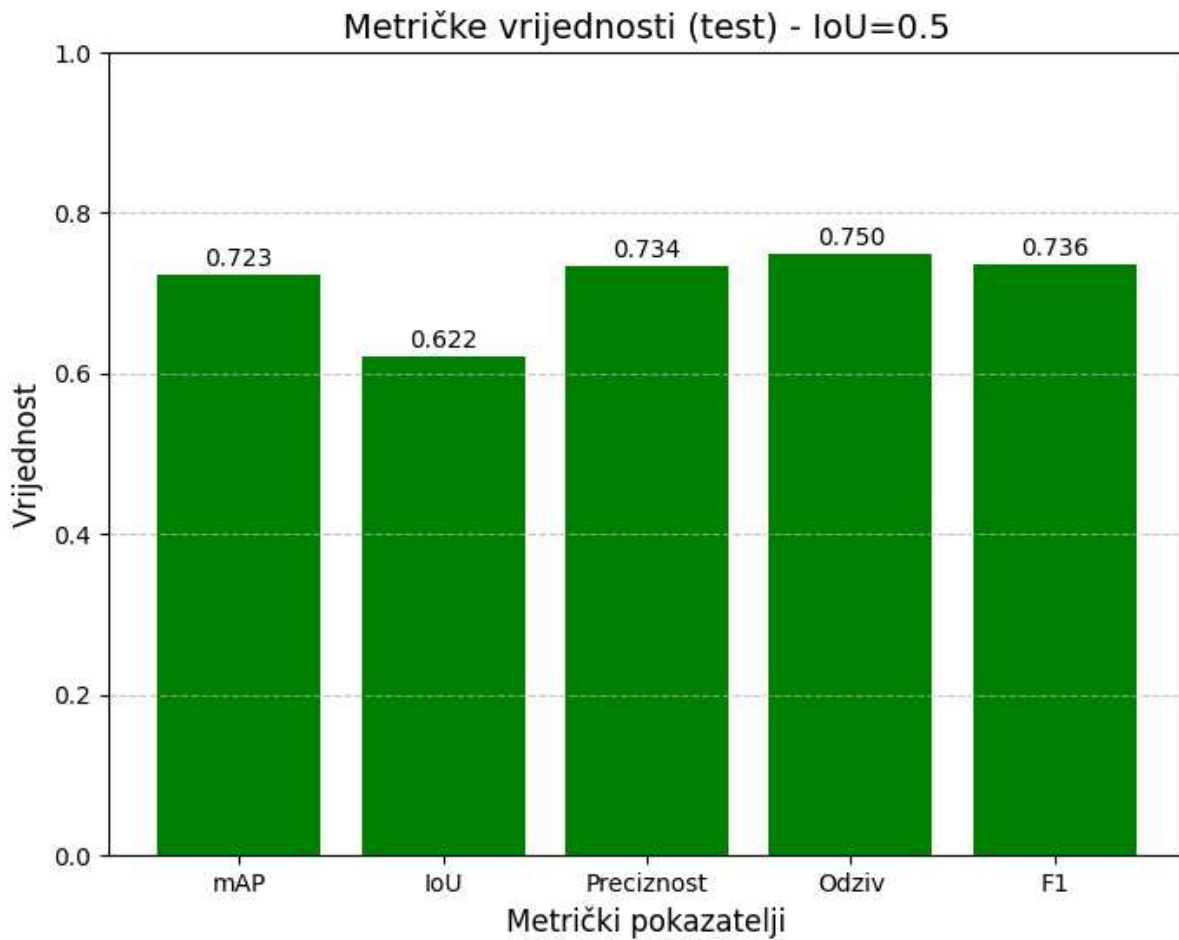


Slika 50. Matrica zabune (validacija) - Gauss šum

Izvor: Izradio autor

Sljedeće dobiveni rezultati odnose se na provedeno testiranje na skupu podataka s Gausovim šumom.

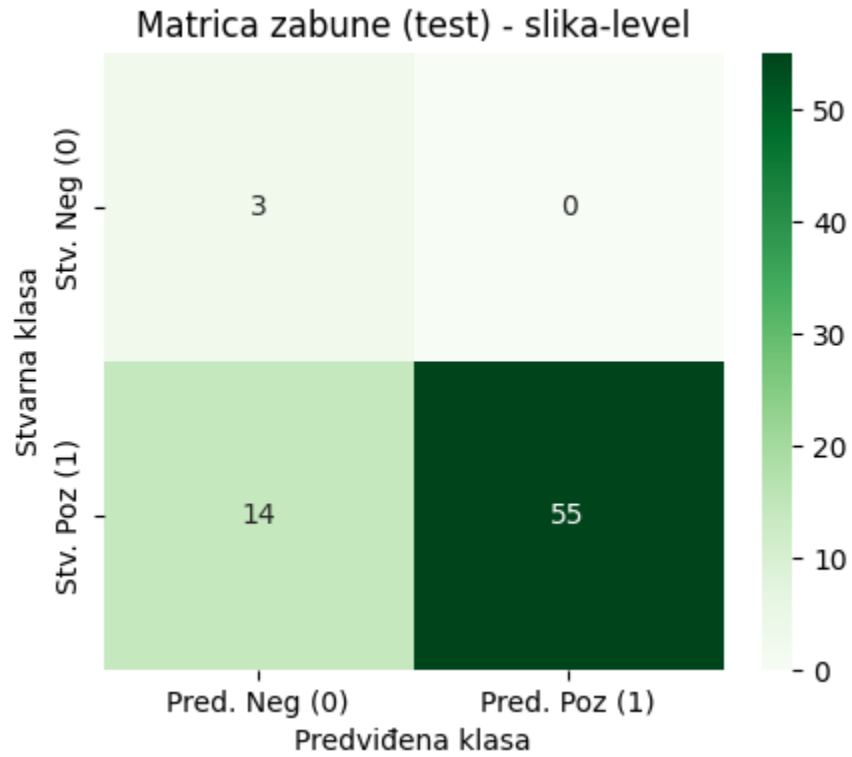
Mjerni pokazatelji na testnom skupu ukazuju na dodatni pad u odnosu na validacijski skup, potvrđujući teoriju da šum negativno utječe na detekcijske sposobnosti modela. Primjećuje se slabije preklapanje detekcijskih okvira s točnim oznakama, no i dalje vrijednosti su relativno visoke točnosti.



Slika 51. Metričke vrijednosti (testiranje) - Gaussov šum

Izvor: Izradio autor

Matrica zabune pokazuje kako je model propustio 14 pozitivnih primjera, što se može protumačiti kao nešto slabija generalizacija uzrokovana primjenom šuma na slike. 55 točno klasificiranih uzoraka pokazuje da model i dalje prepoznaje većinu registarskih oznaka.



Slika 52. Matrica zabune (testiranje) - Gaussov šum

Izvor: Izradio autor

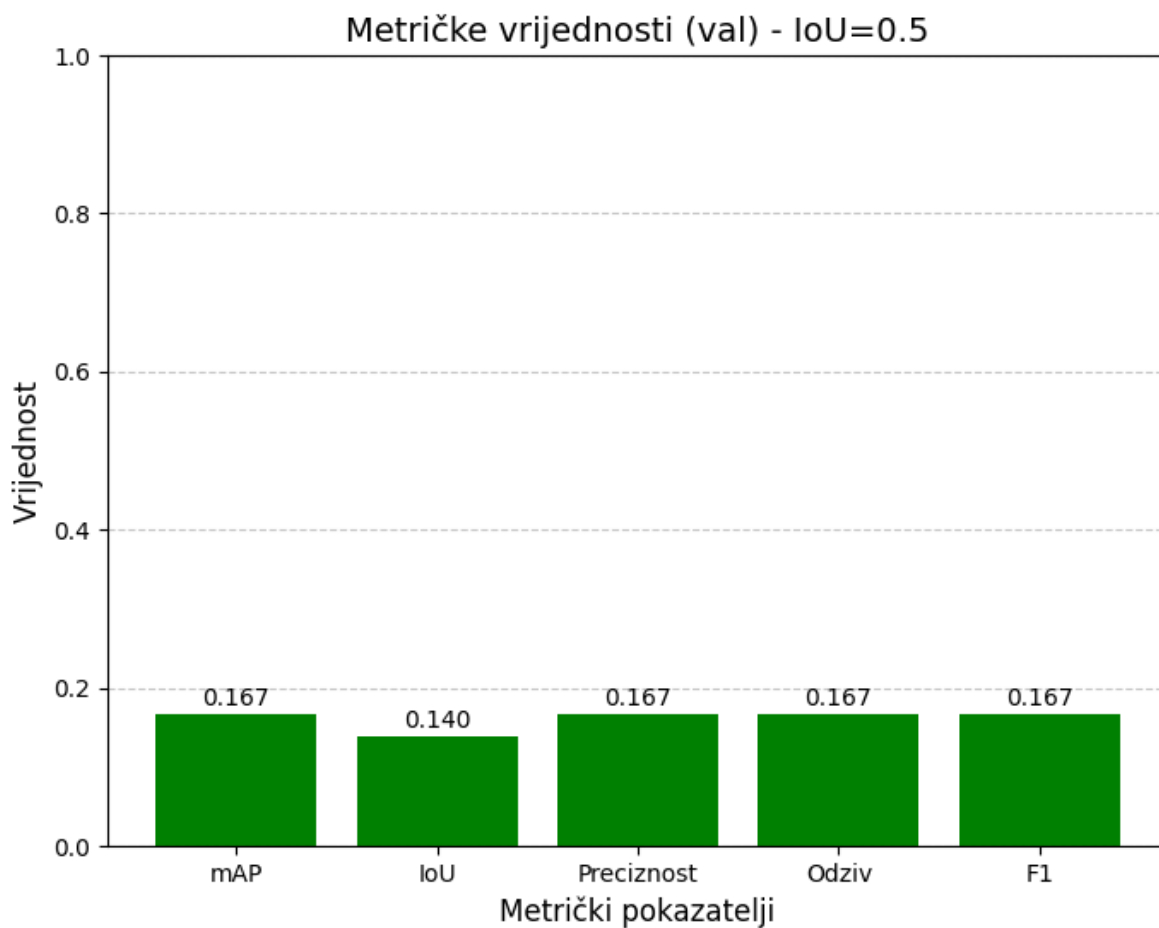
Na slici su prikazane uspješne predikcije modela uz prisutnost Gaussovog šuma. Crveni okviri označavaju detektirane registarske pločice.



Slika 53. Predikcije modela s Gausovim šumom

Izvor: Izradio Autor

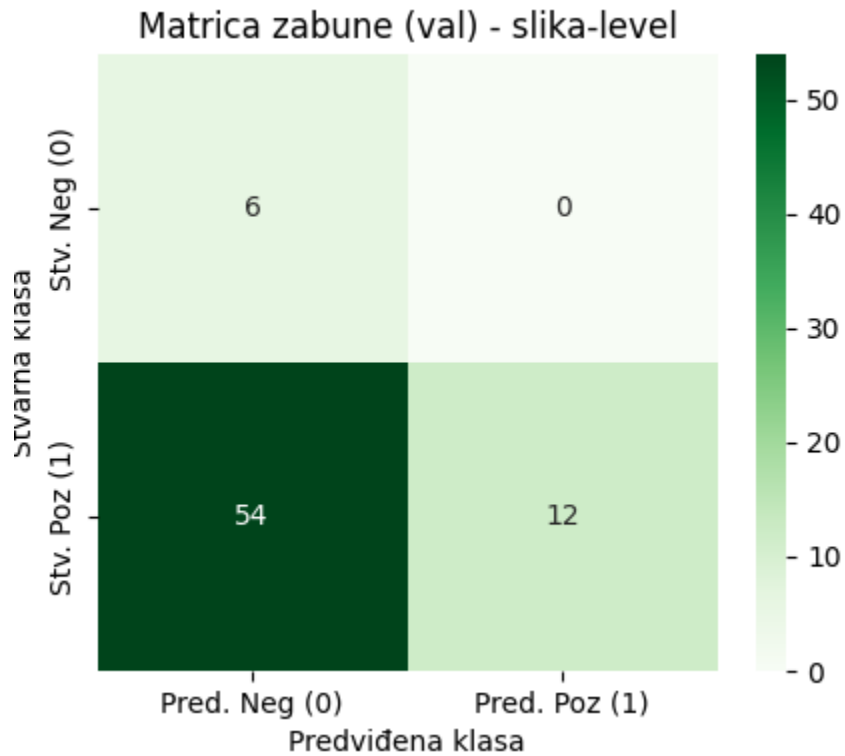
Nakon treniranja, model validiramo na skupu podataka s *salt & pepper* šumom kako bi se ispitala otpornost ekstremnih uvjeta. Rezultati pokazuju značajan pad svih ključnih metrika, mAP iznosi 0.167, dok je IoU smanjen na 0.140. Ostale mjere također su značajno smanjene s obzirom na Gaussov šum, što ukazuje na to kako *salt & pepper* šum uvelike smanjuje točnost algoritma.



Slika 54. Metričke vrijednosti (validacija - salt & pepper)

Izvor: Izradio autor

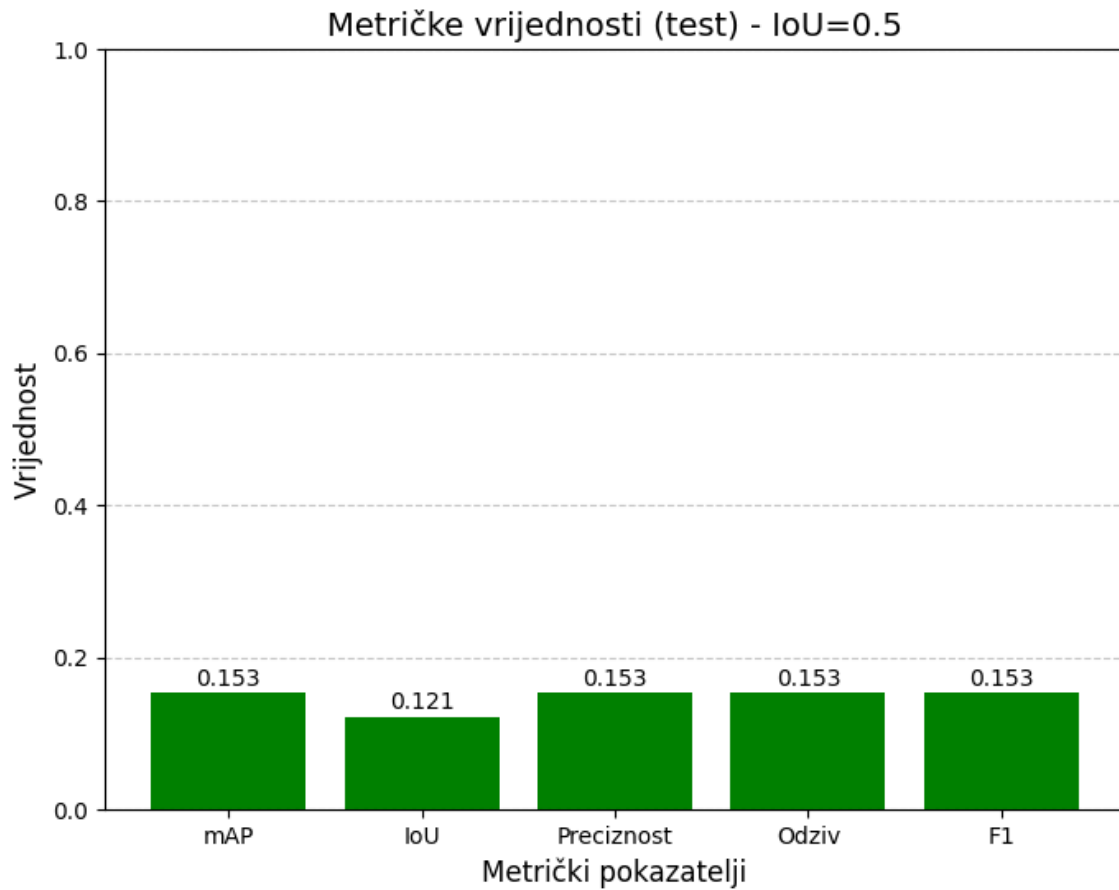
Matrica zabune pokazuje da model ima 54 lažno negativna primjera, što znači da u velikoj većini slučajeva nije prepoznao registarske pločice na slikama s *salt & pepper* šumom. 12 slika je model ispravno klasificirao, dok je 6 slika ispravno označio kao negativne.



Slika 55. Matrica zabune (validacija - salt & pepper)

Izvor: Izradio autor

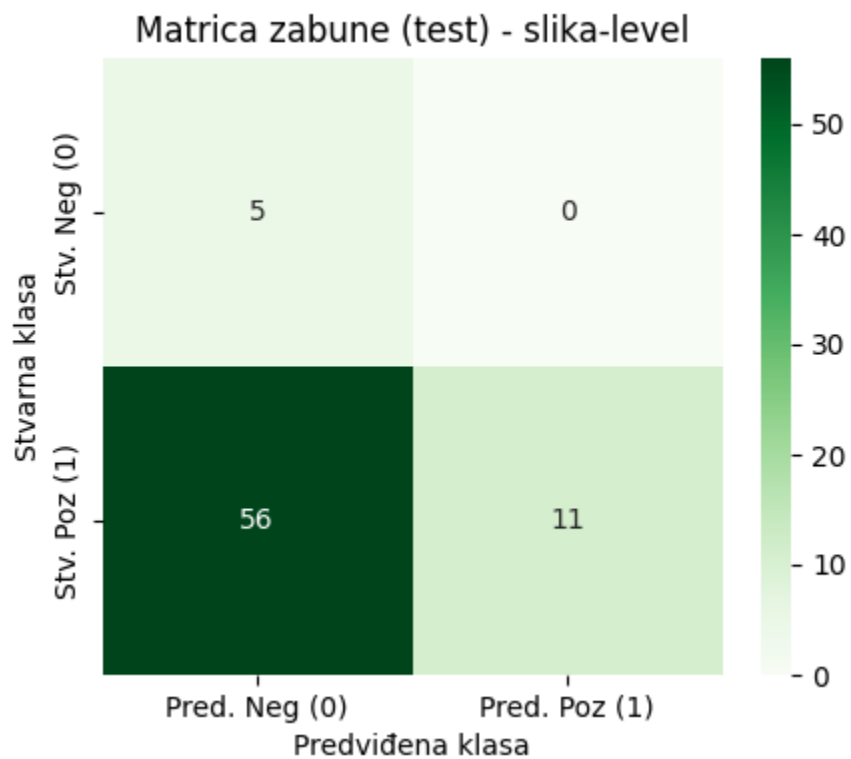
Metričke vrijednosti dobivene testiranjem na testnom skupu podataka sa dodanim *salt & pepper* šumom pokazuju na značajan pad performansi u odnosu na validaciju. Metrika mAP iznosi 0.153, a IoU iznosi 0.121, što ukazuje na loše prepoznavanje i kreplop između predikcija i stvarnih *bounding boxova*.



Slika 56. Metričke vrijednosti (test - salt & pepper šum)

Izvor: Izradio autor

Matrica zabune prikazuje kako od 61 stvarno pozitivne oznake, model klasificira samo 11, dok je 56 pogrešno klasificirao kao negativne, a 5 stvarno negativnih primjera model je točno klasificirao.



Slika 57. Matrica zabune (test - salt & pepper šum)

Izvor: Izradio autor

Na slici su prikazane neuspješne predikcije koje model nije uspio prepoznati. Rezultati algoritma pokazuju kako *salt & pepper* šum značajno više utječe na performanse *Faster* R-CNN modela u odnosu na Gaussov šum. Navedena degradacija ukazuje na osjetljivost modela kod šumova koji uzrokuju nagle promjene u pikselima.

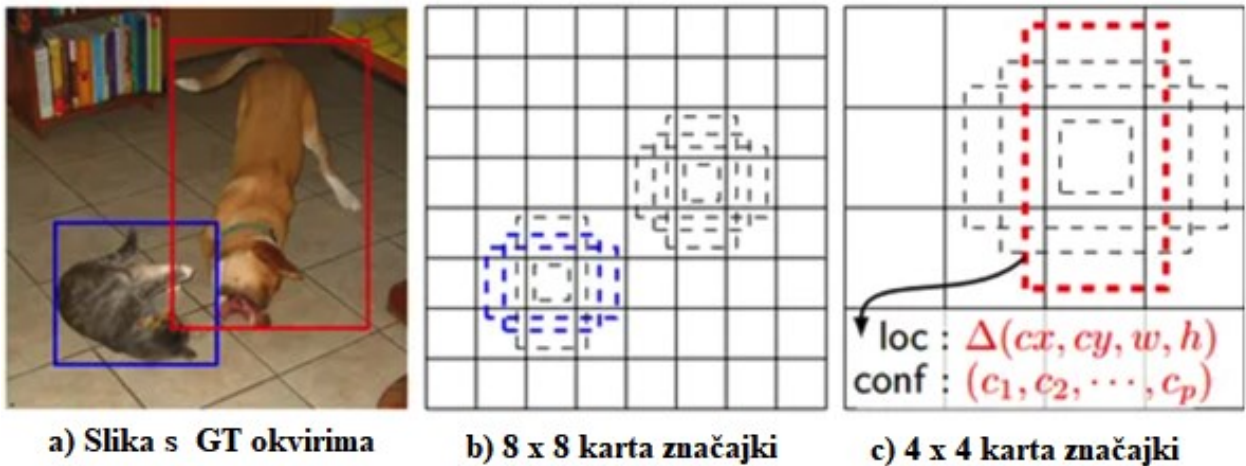


Slika 58. Neuspješno prepoznavanje slika (salt & pepper)

Izvor: Izradio autor

4.5. Implementacija Single Shot Multibox Detector (SSD) algoritma.

SSD ili *Single Shot Detector* je *state of the art* algoritam koji se koristi pri detekciji različitih vrsta objekata u računalnome vidu. Navedeni algoritam je specifičan po tome što izravno detektira više klasa objekata koristeći jednu prolaznu mrežu. Vrlina algoritma je visoka razina točnosti koju algoritam posjeduje. SSD koristi višerezolucijski pristup gdje ovisno o veličini objekta koristi određene razine mreže; što je veći objekt, proporcionalno se koriste veće razine. Algoritam koristi *anchor* koji je unaprijed definirani s različitim dimenzijama kutija od kojih svaka predviđa klase objekta te se prilagođava kako bi se bolje uklopila u realni objekt. Arhitektura mreže je duboka konvolucijska neuronska mreža (CNN), baza se sastoji od unaprijed treniranih mreža, npr. VGG16 ili *MobileNet*.



Slika 59. Prikaz anchor-a

Izvor: Autor modificirao prema (Analytics Vidhya, 2023)

Implementacija SSD-a izvršena je u *google collab* zbog brže izvedbe te mogućnosti korištenja CUDA jezgri koje rade veliku razliku pri treniranju, validaciji i testiranju modela. Korišteni skup podataka je identičan prijašnjima radi konzistentnije usporedbe rezultata. Prvi korak implementacije je kreiranje modela, u ovom slučaju pred treniranu VGG16 arhitekturu kao bazu s unaprijed postavljenim težinama. Model prolazi kroz 20 epoha s *batch size* 16, svaka slika ima svoju pripadajuću anotaciju. Trening skup iznosi 70 % (čiste slike), validacijski skup 15 % (Gauss i *salt & pepper*) i testni skup 15 % (Gauss i *salt & pepper*).

Koristimo *MobileNetV3-Large*, koji je prilagođen zadatku, model detektira dvije klase: pozadinu i registracijsku oznaku.

```
from torchvision.models.detection import (
    ssdlite320_mobilenet_v3_large,
    SSDLite320_MobileNet_V3_Large_Weights
)

model = ssdlite320_mobilenet_v3_large(
    weights=SSDLite320_MobileNet_V3_Large_Weights.DEFAULT
)
```

```

model.head.classification_head.num_classes = num_classes

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

```

Slika 60. Kreiranje SSD modela

Izvor: Izradio autor

Koristimo SGD optimizator i *step – based learning rate decay* kako bi model stabilnije učio.

```

params = [p for p in model.parameters() if p.requires_grad]
optimizer = optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)
lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.1)

```

Slika 61. Postavljanje optimizatora

Izvor: Izradio autor

U nastavku je prikazan dio koda korištenog za trening na jednoj epohi.

```

def treniraj_jednu_epohu(model, optimizer, data_loader):
    model.train()
    ukupni_gubitak = 0
    for images, targets in data_loader:
        images = list(img.to(device) for img in images)
        targets = [{k:v.to(device) for k,v in t.items()} for t in targets]

        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())

        optimizer.zero_grad()
        losses.backward()
        optimizer.step()

        ukupni_gubitak += losses.item()

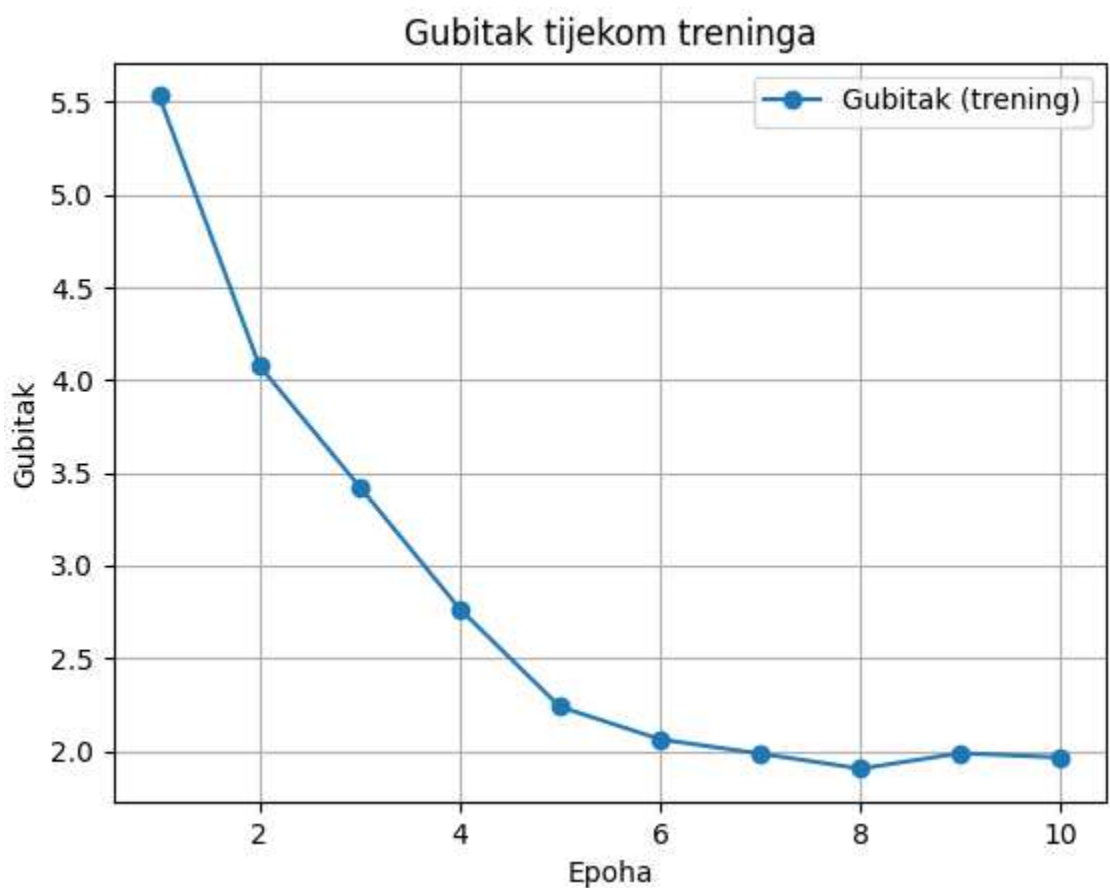
    return ukupni_gubitak / len(data_loader)

```

Slika 62. Primjer treninga za jednu epohu

Izvor: Izradio autor

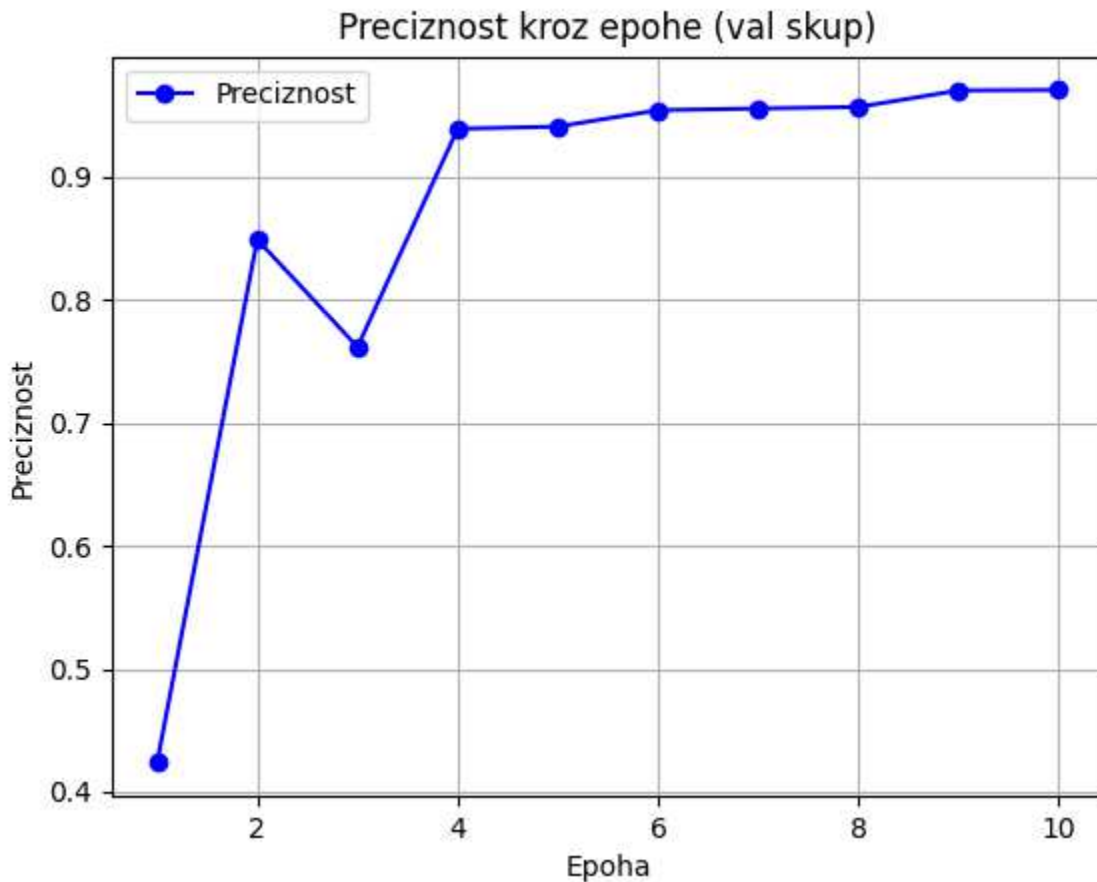
Graf gubitka tijekom treninga prikazuje smanjenje gubitka kroz epohe, što je očekivano jer model s vremenom postaje bolji u prepoznavanju uzoraka u podacima. Smanjenje gubitka ukazuje na prilagođavanje modela. Model je treniran kroz više pokušaja na raznom broju epoha, no pokazalo se kako je 10 epoha najbolja razina za navedenu primjenu.



Slika 63. Gubitak tijekom treninga

Izvor: Izradio autor

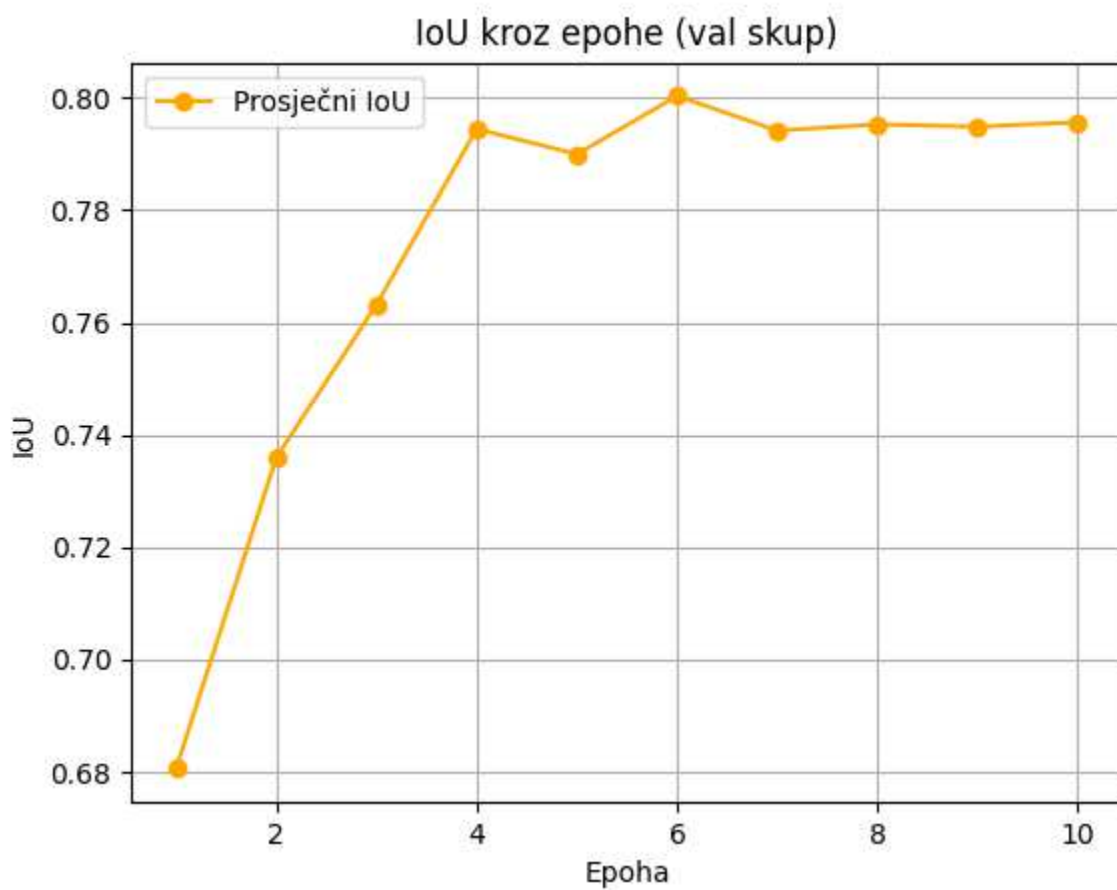
Sljedeći graf prikazuje omjer točnih pozitivnih predikcija u odnosu na sve pozitivne predikcije modela. Preciznost značajno raste tijekom prvih 7 epoha, nakon toga se stabilizira. Navedeni trend sugerira brzo učenje modela na podacima.



Slika 64. Preciznost kroz epohe

Izvor: Izradio autor

IoU kroz epohe mjeri kakvo je preklapanje predviđenih i stvarnih *bounding boxova*, kod navedenog algoritma primjećuje se kontinuirani rast IoU vrijednosti, što pokazuje da model poboljšava preciznost lokacije detektiranih pločica.



Slika 65. IoU kroz epohe

Izvor: Izradio autor

Graf odziva prikazuje oscilaciju u prvih par epoha treniranja, što ukazuje na nesigurnost modela u početnim fazama učenja. Kasnije dolazi do stabilizacije, no ukupna vrijednost i dalje ostaje niža, što sugerira na postojanje lažno negativnih slučajeva.



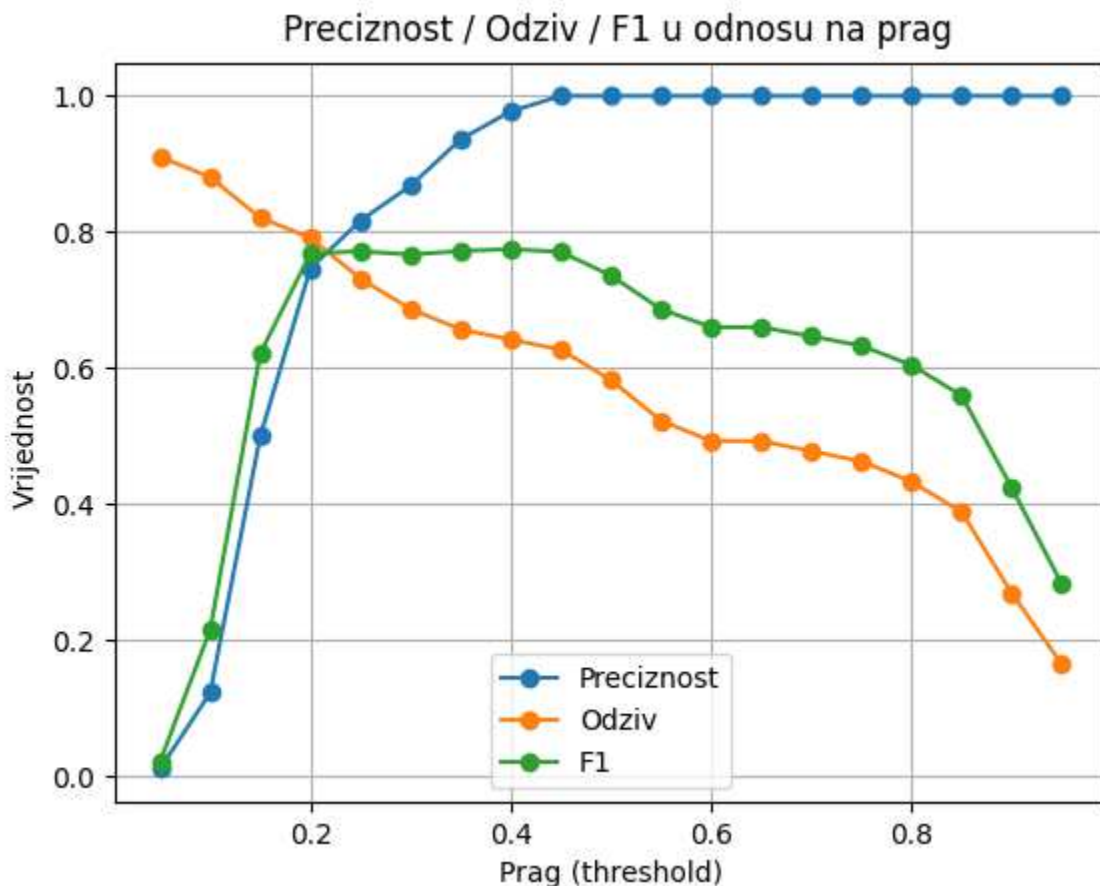
Slika 66. Odziv kroz epohe

Izvor: Izradio autor

Nakon treniranja modela i evaluiranja njegovih metrika, potrebna je dodatna evaluacija i testiranje na slikama s dodanim Gausovim šumom. Analiza omogućuje procjenu modela pri otežanim uvjetima detekcije.

Ovaj graf prikazuje odnos između preciznosti, odziva i F1-mjere u ovisnosti o pragu (*threshold*) tijekom validacije modela na slikama s Gausovim šumom. Preciznost naglo raste s povećanjem praga i doseže maksimalnu vrijednost oko 0.4, nakon čega ostaje stabilna. Odziv ima suprotan trend – veći prag rezultira njegovim padom, što znači da model propušta detekcije koje nisu dovoljno sigurne. F1-mjera, koja predstavlja balans između preciznosti i odziva, postiže svoj maksimum oko praga od 0.2 do 0.3, a zatim se postupno smanjuje. Ovi rezultati sugeriraju da niži prag omogućuje bolju detekciju

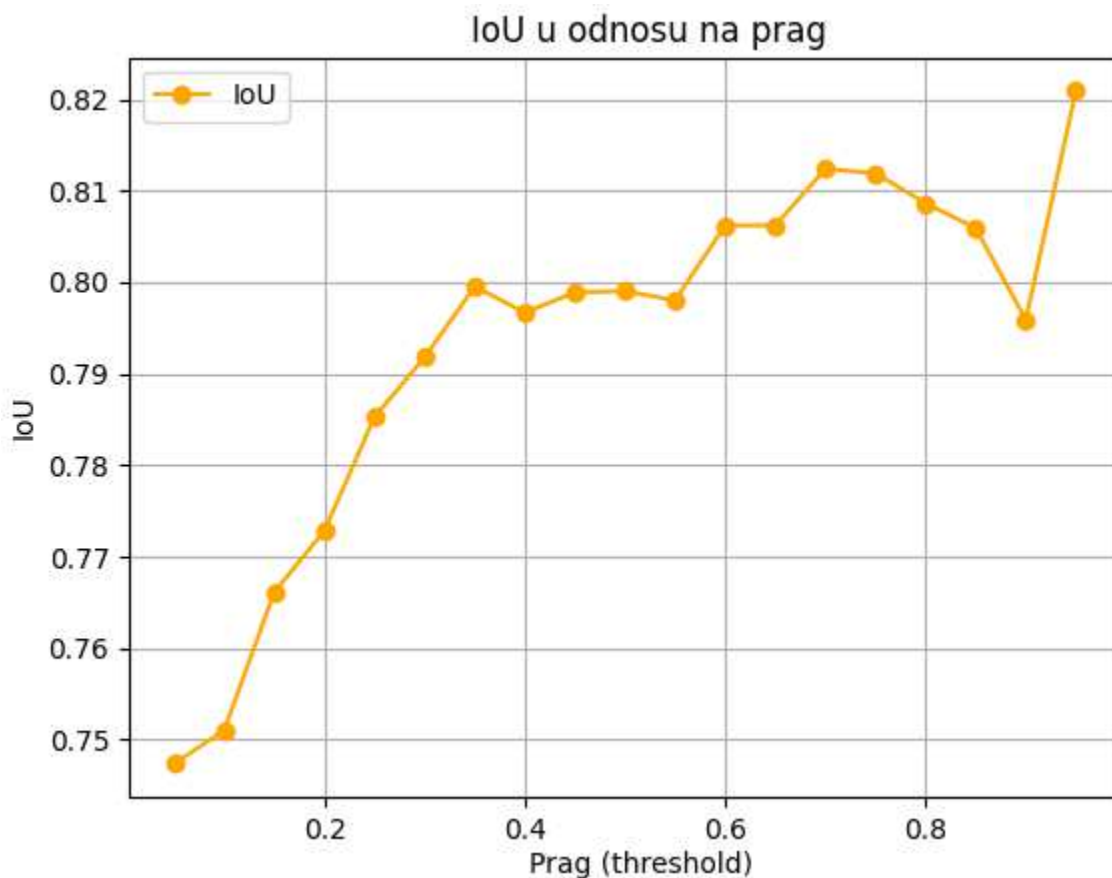
registracijskih pločica na slikama s Gausovim šumom, dok veći prag poboljšava preciznost, ali smanjuje ukupnu detekciju.



Slika 67. Preciznost/Odziv/F1 u odnosu na prag

Izvor: Izradio autor

Kako se prag povećava, vrijednost IoU također raste, što ukazuje na to da model postaje precizniji pri višim pragovima povjerenja. Međutim, nakon određene točke (~ 0.6 – 0.8), rast se usporava, a prema višim pragovima dolazi do blagog osciliranja. To sugerira da je model stabilan u određivanju lokacije registarskih pločica čak i u prisutnosti šuma, ali da viši prag smanjuje broj detekcija, što može dovesti do propuštanja nekih ispravnih predikcija.

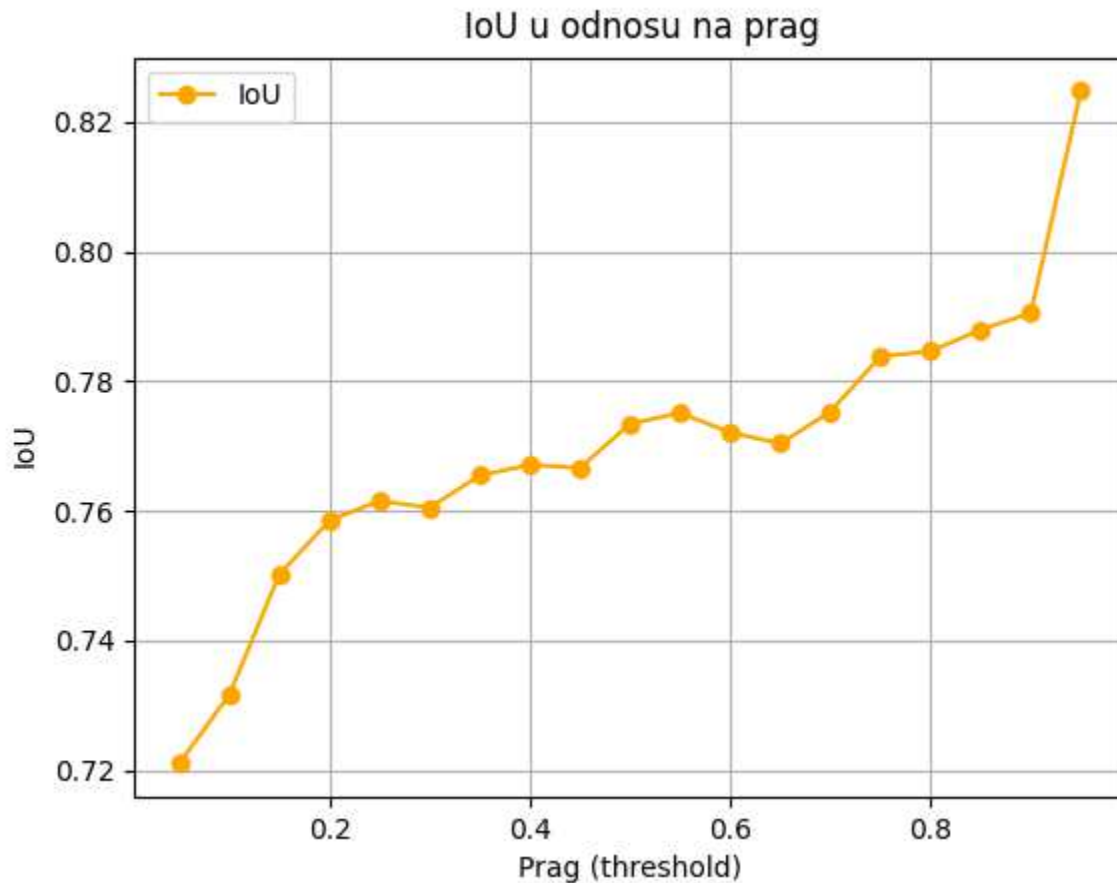


Slika 68. IoU u odnosu na prag (val - Gauss)

Izvor: Izradio autor

Nakon analize rezultata validacije na slikama s Gaussovim šumom, prelazi se na testiranje rezultata istog.

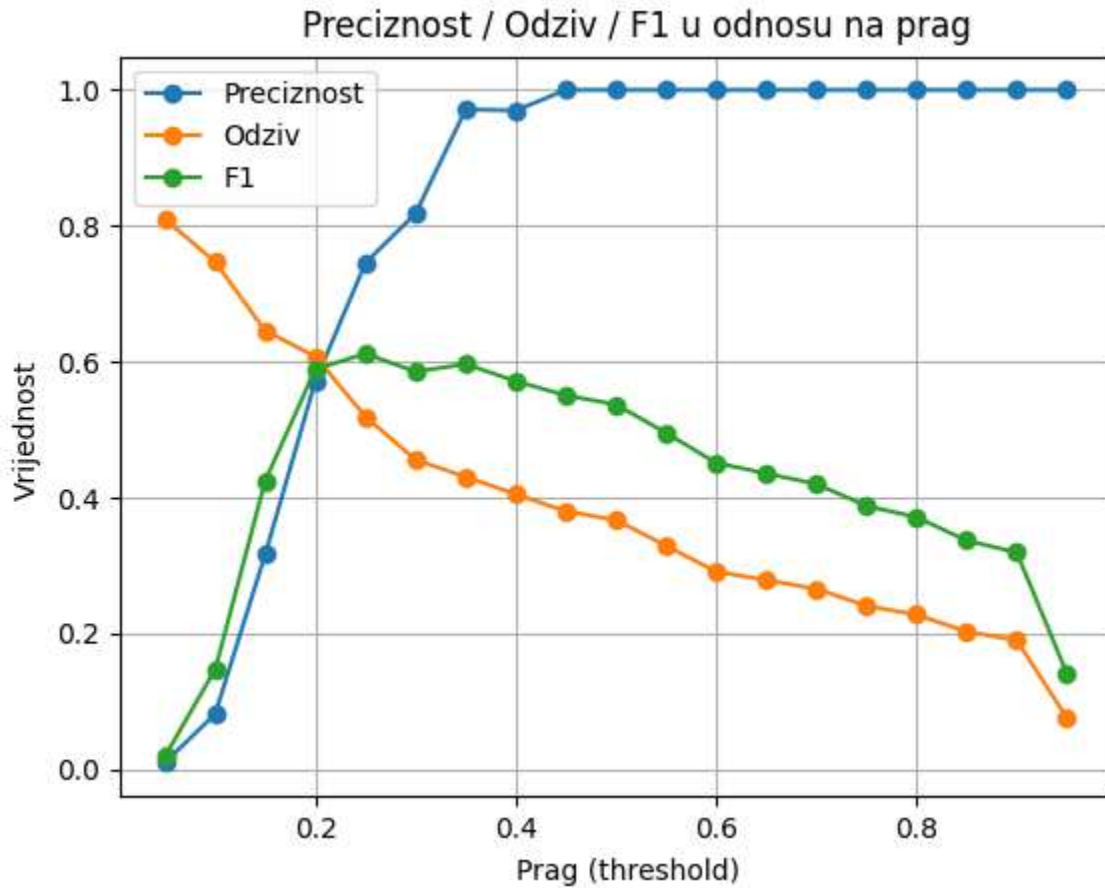
Na sljedećem grafu je vidljivo da IoU raste s povećanjem praga, što znači da postiže preciznije detekcije kada je prag viši. S obzirom na validaciju primjećuje se nešto viši krajnji rezultat, no rast više oscilira.



Slika 69. IoU u odnosu na prag

Izvor: Izradio autor

U usporedbi s validacijskim grafom, možemo primijetiti da su trendovi vrlo slični, ali su rezultati kod test skupa nešto slabiji – preciznost ostaje visoka, ali su odziv i f1 nešto niži. To ukazuje da model ima malo većih poteškoća pri testiranju nego kod validacije, što može značiti da slike u test skupu sadrže složenije ili teže primjere registarskih pločica.



Slika 70. Preciznost / odziv / f1 (test - Gauss)

Izvor: Izradio autor

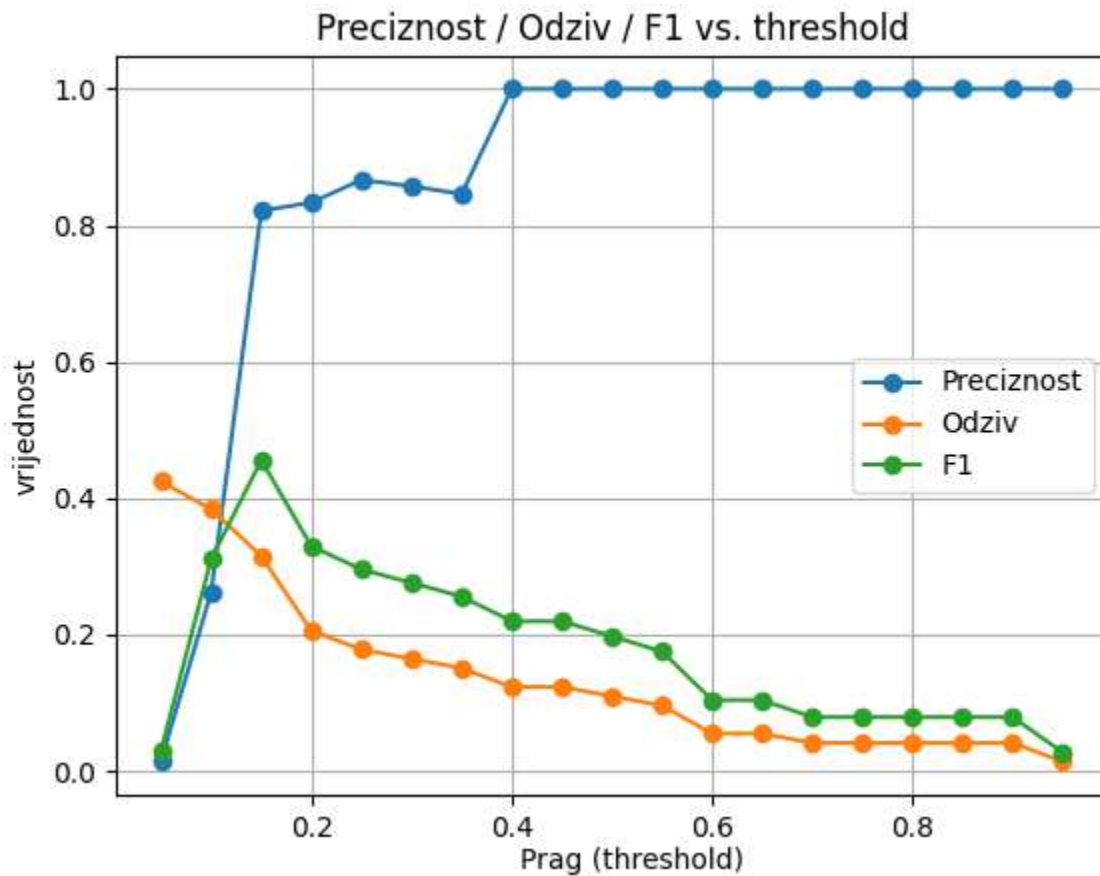
Sljedeća slika prikazuje predikcije modela na testnim podacima s Gausovim šumom. Na slici se mogu vidjeti registarske pločice koje model detektira, pripadno s postocima pouzdanosti predikcije.



Slika 71. Predikcije SSD modela na Gaussovom šumu

Izvor: Izradio autor

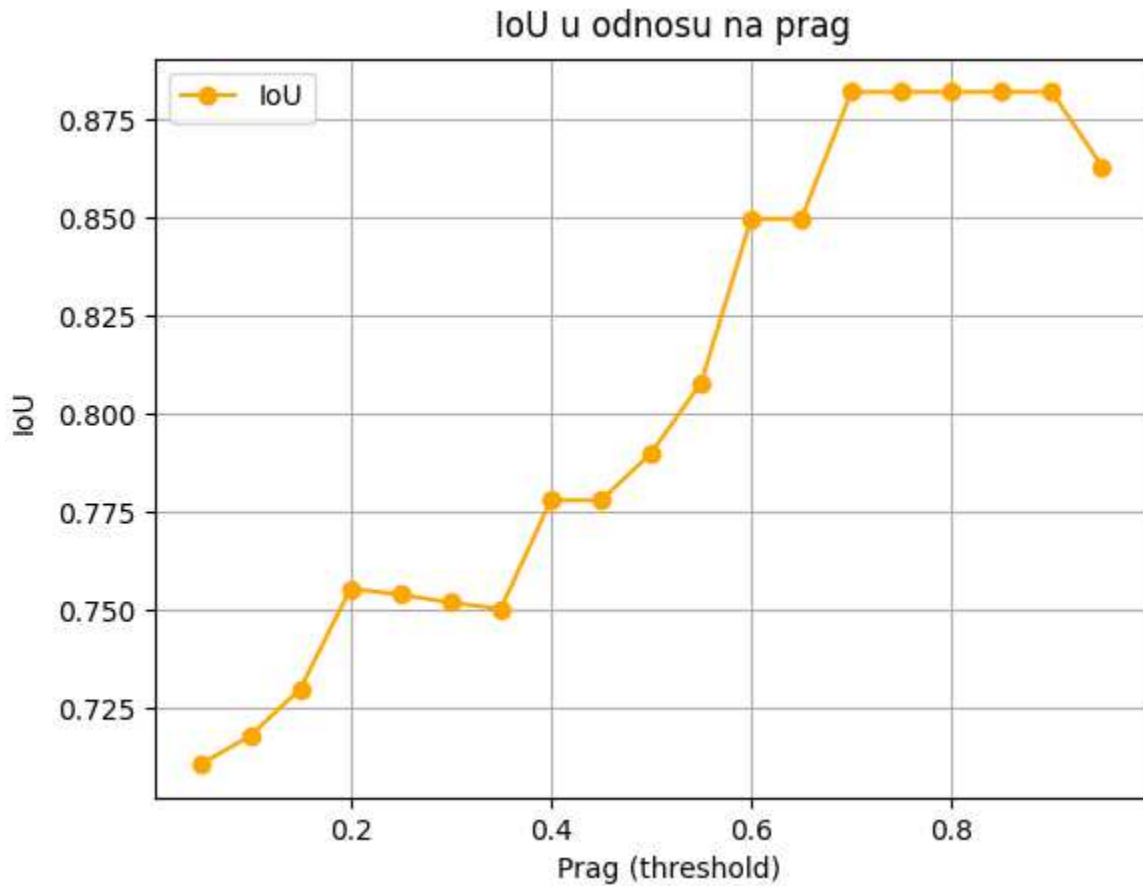
Nakon analize rezultata modela na Gaussovom šumu, potrebno je validirati (testirati) na slikama s *salt & pepper* šumom. U usporedbi s Gausovim šumom, model teže prepoznaje registarske pločice kada su one prekrivene *salt & pepper* šumom.



Slika 72. Preciznost / odziv / f1 (salt & pepper šum)

Izvor: Izradio autor

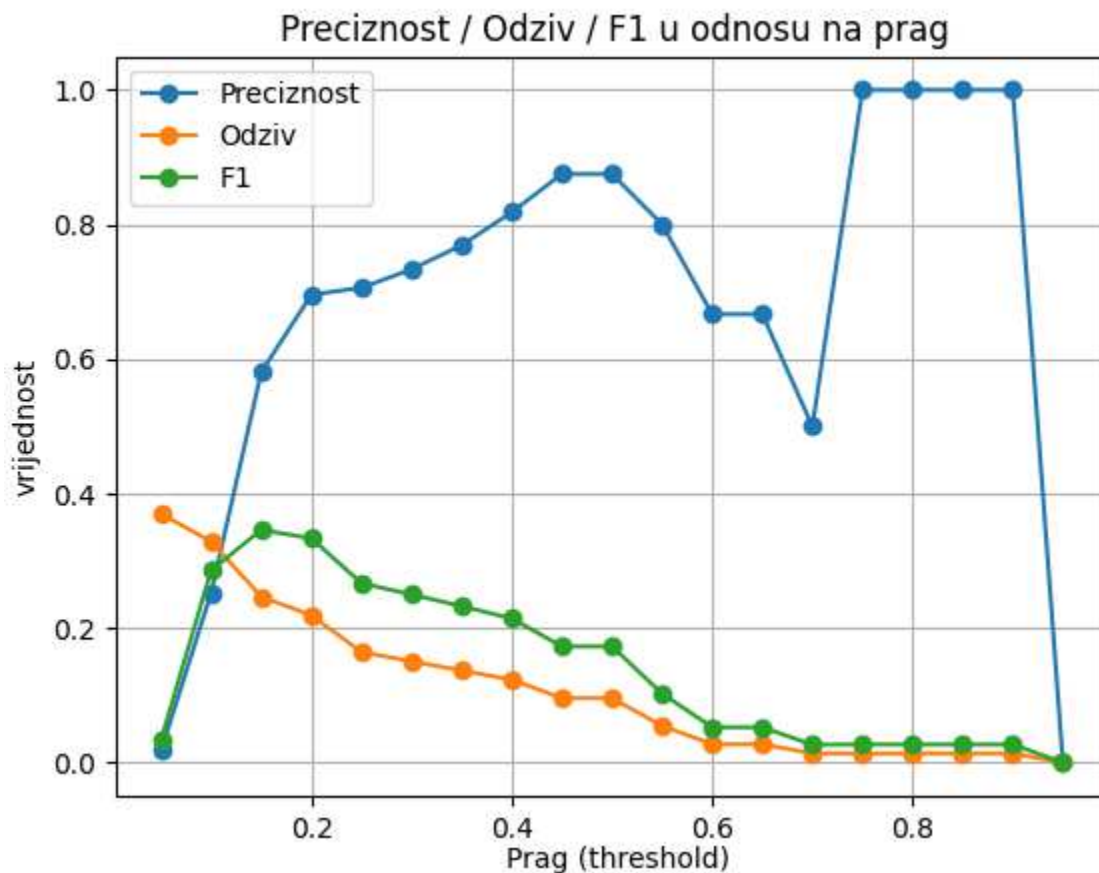
U usporedbi s validacijom Gaussovog šuma, IoU je stabilniji, što znači da model konzistentnije predviđa *bounding boxove*, usprkos ostalim metrikama koje su osjetno niže.



Slika 73. IoU u odnosu na prag (salt & pepper - val)

Izvor: Izradio autor

Graf prikazuje rezultate testiranja modela na podacima sa salt & pepper šumom, analizirajući preciznost, odziv i F1 mjeru u odnosu na prag klasifikacije. Uočava se da preciznost naglo raste do praga oko 0.2, nakon čega postupno opada i pokazuje značajne oscilacije, što ukazuje na nestabilnost modela u prisutnosti ovog tipa šuma. Odziv i F1 mjera kontinuirano opadaju s rastom praga, pri čemu je F1 mjera znatno niža nego kod testiranja na Gaussovom šumu. Ovo sugerira da model često krivo klasificira objekte (lažno pozitivni slučajevi), što dovodi do visokih varijacija u preciznosti pri višim pragovima.



Slika 74. Preciznost / Odziv / F1 (test - salt & pepper)

Izvor: Izradio autor

Ovaj graf prikazuje IoU u odnosu na prag klasifikacije prilikom testiranja modela na *salt & pepper* šumu. U usporedbi s testiranjem na Gaussovom šumu, primjećuje se da IoU započinje na nešto nižim vrijednostima i postupno raste, ali uz manje oscilacije nego kod Gaussovog šuma. Međutim, na višim pragovima dolazi do naglog pada IoU-a, što nije bio slučaj kod Gaussovog šuma, gdje su vrijednosti bile stabilnije.

Sljedeća slika prikazuje kako model nije uspio detektirati registarske pločice na automobilima zbog intenzivnog *salt & pepper* šuma. Navedeni šum jasno utječe na jasnoću rubova i tekstura, što otežava modelu prepoznavanje.



Slika 75. Neuspješne detekcije modela na slikama sa salt & pepper šumom

Izvor: Izradio autor

Testiranje SSD modela na podacima sa šumom pokazalo je da različiti tipovi šuma imaju značajan utjecaj na performanse detekcije registarskih pločica. Gaussov šum je uzrokovao blagi pad preciznosti i odziva, ali model je i dalje bio u stanju prepoznati većinu registarskih pločica. S druge strane, *salt & pepper* šum, posebno pri visokom intenzitetu, značajno je otežao detekciju, uzrokujući veći broj propuštenih detekcija i smanjenje F1 mjere.

4.6. Razvoj i evaluacija vlastite neuronske mreže

U ovom odlomku prikazan je razvoj i evaluacija vlastite neuronske mreže čiji je cilj rješavanje problema prepoznavanja registarskih oznaka vozila. Izvedba personalizirana neuronske mreže odrađena je u *google collab*-u radi mogućnosti korištenja eksternog GPU-a.

Prvi korak je uvoz potrebnih biblioteka i montiranje *google drive*-a

```
# Montiranje Google Drivea  
drive.mount('/content/drive', force_remount=True)
```

```

#!pip install tensorflow scikit-learn matplotlib opencv-python

import os
import glob
import numpy as np
import cv2
import matplotlib.pyplot as plt
import xml.etree.ElementTree as ET
from collections import Counter

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix,
precision_recall_fscore_support, roc_curve, auc, precision_recall_curve

# =====
# 1. Postavke direktorija
# =====
base_dir = '/content/drive/MyDrive/Diplomski/VlastitaNM/dataset'
train_images_dir = os.path.join(base_dir, 'train', 'images')
train_annotations_dir = os.path.join(base_dir, 'train', 'annotations')
results2_dir = os.path.join(base_dir, 'results2')
if not os.path.exists(results2_dir):
    os.makedirs(results2_dir)

```

Slika 76. Instalacija i unos potrebnih biblioteka

Izvor: Izradio autor

Sljedeći korak u kreiranju personalizirane neuronske mreže je učitavanje i parsiranje iz XML anotacija.

```

def load_data(image_dir, annotations_dir):
    """
    Učitava slike i XML anotacije (PASCAL VOC stil) iz zadanih
    direktorija.
    Ako je pronađena anotacija, parsiraju se bounding boxovi i slika se
    označava kao pozitivna (1);
    inače, slika se označava kao negativna (0).
    """

```

```

"""
slike = []
boksovi = []
oznake = []
image_files = sorted(glob.glob(os.path.join(image_dir, '*..*')))
for image_file in image_files:
    ime_slike = os.path.basename(image_file)
    annotation_file = os.path.join(annotations_dir,
os.path.splitext(ime_slike)[0] + '.xml')
    img = cv2.imread(image_file)
    if img is None:
        print(f"Upozorenje: Ne mogu učitati {image_file}")
        continue
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    slike.append(img)
    if os.path.exists(annotation_file):
        tree = ET.parse(annotation_file)
        root = tree.getroot()
        boks = []
        lbls = []
        for member in root.findall('object'):
            bndbox = member.find('bndbox')
            xmin = int(float(bndbox.find('xmin').text))
            ymin = int(float(bndbox.find('ymin').text))
            xmax = int(float(bndbox.find('xmax').text))
            ymax = int(float(bndbox.find('ymax').text))
            boks.append([xmin, ymin, xmax, ymax])
            lbls.append(1)
        if len(lbls) == 0:
            lbls = [0]
        boksovi.append(boks)
        oznake.append(lbls)
    else:
        boksovi.append([])
        oznake.append([0])
return slike, boksovi, oznake

slike, boksovi, oznake = load_data(train_images_dir,
train_annotations_dir)
print(f"Učitano je {len(slike)} slika.")

```

Slika 77. Učitavanje i pretprocesiranje slika

Izvor: Izradio autor

Podaci su preprocesirani tako da su slike promijenjene na fiksnu veličinu (128x128), a *bounding boxovi* su skalirani u skladu s promjenom veličine slike.

```
target_size = (128, 128)

def preprocess_data(slike, boksovi, oznake, target_size=(128,128)):
    proc_slike = []
    proc_boksovi = []
    proc_oznake = []
    for img, boks, lbls in zip(slike, boksovi, oznake):
        img_resized = cv2.resize(img, target_size)
        proc_slike.append(img_resized)
        scale_x = target_size[0] / img.shape[1]
        scale_y = target_size[1] / img.shape[0]
        boks_resized = []
        for bx in boks:
            xmin = bx[0] * scale_x
            ymin = bx[1] * scale_y
            xmax = bx[2] * scale_x
            ymax = bx[3] * scale_y
            boks_resized.append([xmin, ymin, xmax, ymax])
        proc_boksovi.append(boks_resized)
        proc_oznake.append(lbls)
    return np.array(proc_slike), proc_boksovi, proc_oznake

slike, boksovi, oznake = preprocess_data(slike, boksovi, oznake,
target_size=target_size)
slike = slike / 255.0 # Normalizacija
```

Slika 78. Preprocesiranje podataka

Izvor: Izradio autor

Podaci se dijele u tri skupa: 70 % podataka se koristi za treniranje mreže, 15 % za validaciju i 15 % za testiranje. Svaki skup sadržava nasumično odabrane primjere slika. 70 % podataka kojih se koristi za treniranje su slike koje ne sadrže šum to jest čiste slike, dok se preostalih 15 % validacije i 15 % testiranja prvo izvodi na podacima s Gausovim šumom pa na podacima sa *salt & pepper* šumom. Validacija se ujedno i

provodi nakon treniranja na čistim podacima kako bi se pratila ravnoteža i uspješnost učenja algoritma.

```
poz_ind = [i for i, lbl in enumerate(oznake_conv) if lbl[0]==1]
neg_ind = [i for i, lbl in enumerate(oznake_conv) if lbl[0]==0]
np.random.seed(42)
np.random.shuffle(poz_ind)
np.random.shuffle(neg_ind)

pos_train_end = int(0.7 * len(poz_ind))
pos_val_end = int(0.85 * len(poz_ind))
train_pos = poz_ind[:pos_train_end]
val_pos = poz_ind[pos_train_end:pos_val_end]
test_pos = poz_ind[pos_val_end:]

neg_train_end = int(0.7 * len(neg_ind))
neg_val_end = int(0.85 * len(neg_ind))
train_neg = neg_ind[:neg_train_end]
val_neg = neg_ind[neg_train_end:neg_val_end]
test_neg = neg_ind[neg_val_end:]

train_ind = train_pos + train_neg
val_ind = val_pos + val_neg
test_ind = test_pos + test_neg
```

Slika 79. Podjela podataka

Izvor: Izradio autor

Model je konvolucijska neuronska mreža s tri konvolucijska sloja i dvije izlazne grane: jedna za binarnu klasifikaciju i druga za regresiju *bounding boxova*.

```
def create_custom_model(input_shape=(128,128,3)):
    ulaz = keras.Input(shape=input_shape)
    x = layers.Conv2D(32, (3,3), activation='relu')(ulaz)
    x = layers.MaxPooling2D((2,2))(x)
    x = layers.Conv2D(64, (3,3), activation='relu')(x)
    x = layers.MaxPooling2D((2,2))(x)
    x = layers.Conv2D(128, (3,3), activation='relu')(x)
```

```

x = layers.MaxPooling2D((2,2))(x)
x = layers.Flatten()(x)
x = layers.Dense(64, activation='relu')(x)
x = layers.Dropout(0.5)(x)
izlaz_klasa = layers.Dense(1, activation='sigmoid',
name='class_output')(x)
izlaz_bbox = layers.Dense(4, activation='linear',
name='bbox_output')(x)
model = keras.Model(inputs=ulaz, outputs=[izlaz_bbox, izlaz_klasa])
return model

model = create_custom_model()
model.summary()

```

Slika 80. Arhitektura neuronske mreže

Izvor: Izradio autor

Model je treniran 50 epoha koristeći „Adam“ optimizator i ponderirane gubitke za klasifikaciju i *bounding boxove*.

```

epochs = 50
history = model.fit(train_gen,
                    steps_per_epoch=train_steps,
                    validation_data=val_gen,
                    validation_steps=val_steps,
                    epochs=epochs)

# Spremanje treniranog modela
model_save_path = os.path.join(results2_dir,
"custom_model_treniran.h5")
model.save(model_save_path)

```

Slika 81. Treniranje modela

Izvor: Izradio autor

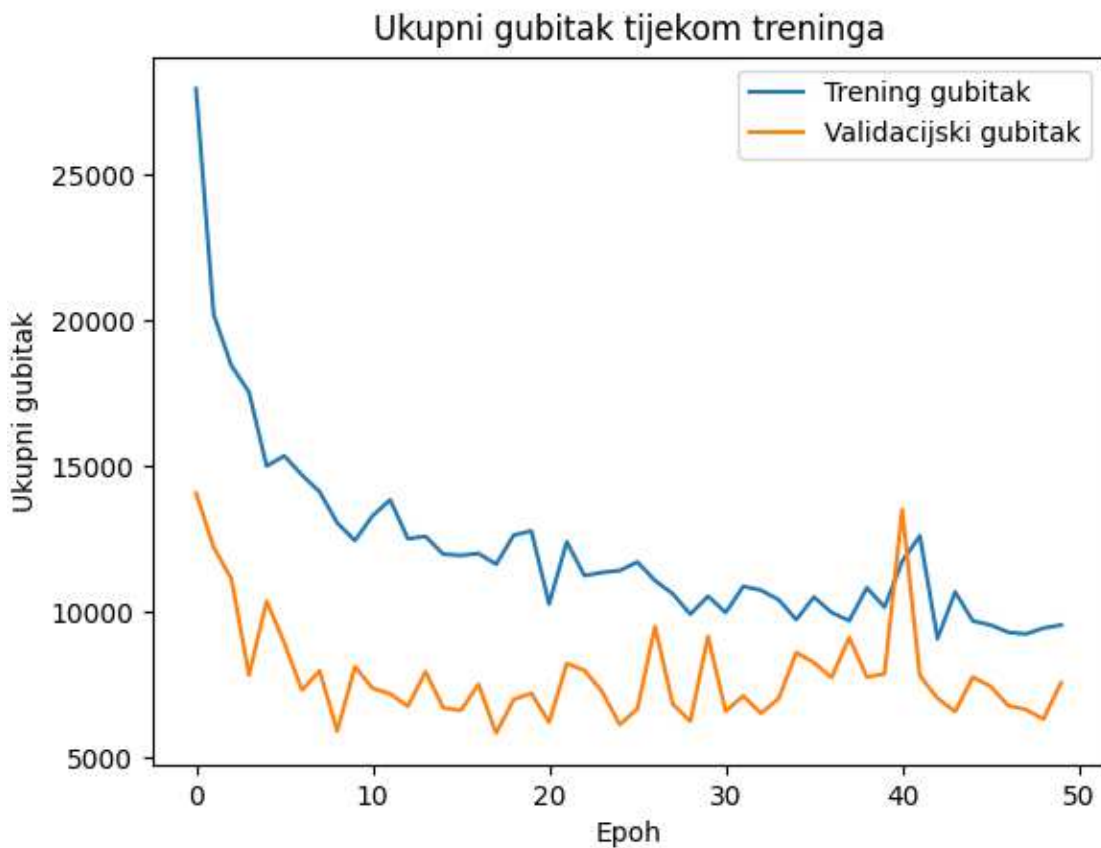
Model se evaluira na test skupu s metrikama točnosti, preciznosti, odziva i gubitka.

```
test_results = model.evaluate(test_slike, {'class_output': test_oznake,  
'bbox_output': test_bbox})  
print("Test rezultati:", test_results)
```

Slika 82. Evaluacija modela

Izvor: Izradio autor

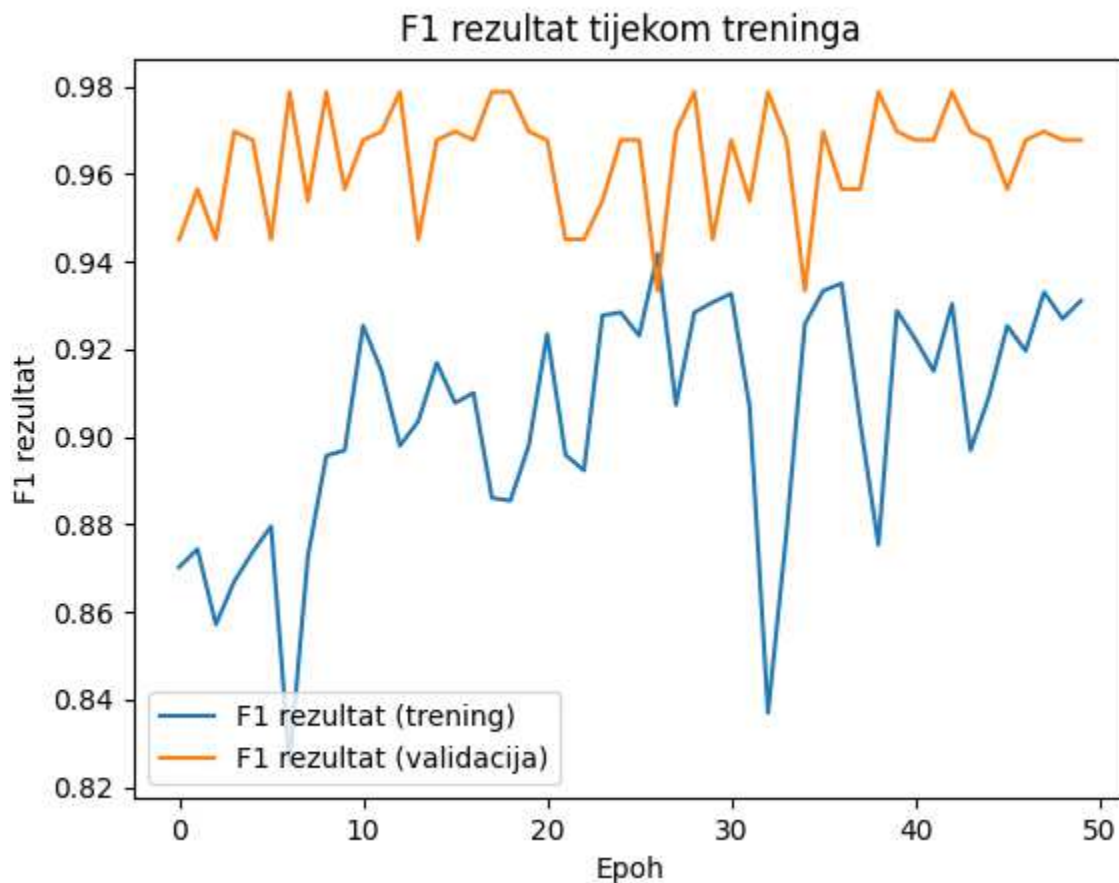
Sljedeći graf prikazuje ukupni gubitak tijekom treninga za trening i validacijski skup podataka. Gubitak se smanjuje kroz epohe, što označuje učenje modela. Vidljiv je validacijski gubitak koji se prikazuje u karakterističnim oscilacijama, što može ukazivati na određeni stupanj varijabilnosti.



Slika 83. Ukupni gubitak tijekom treninga

Izvor: Izradio autor

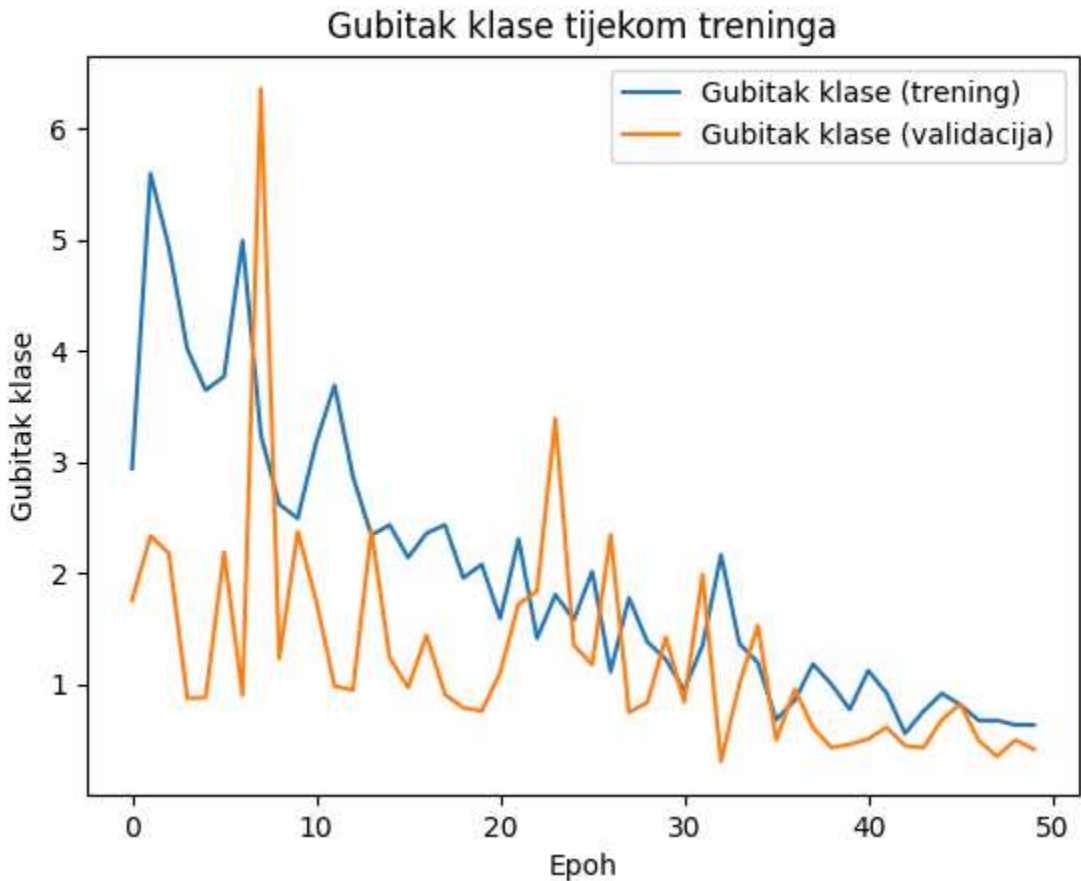
Sljedeći graf prikazuje f1 rezultat tijekom treninga i validacije. Primjećuje se rast varijable f1 kroz trening, što ukazuje na poboljšanje modela. Validacijski f1 rezultat ostaje na relativno visokoj razini, što ukazuje na dobru generalizaciju.



Slika 84. F1 rezultat tijekom treninga

Izvor: Izradio autor

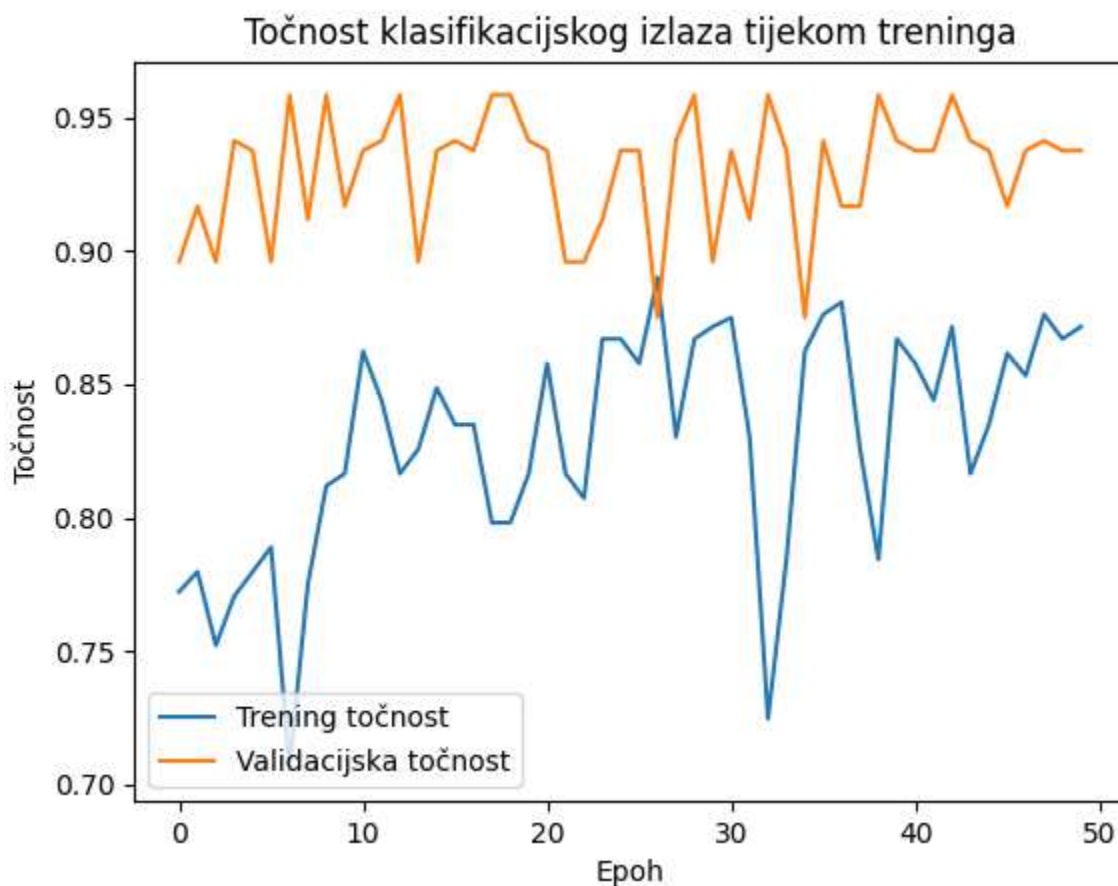
Graf gubitka klase tijekom treninga ukazuje kako gubitak klase opada s vremenom kroz epohe, što ukazuje na dobro učenje modela. U kasnijim epohama, primjećuje se kako se gubici stabiliziraju, što sugerira konvergenciju modela.



Slika 85. Gubitak klase tijekom treninga

Izvor: Izradio autor

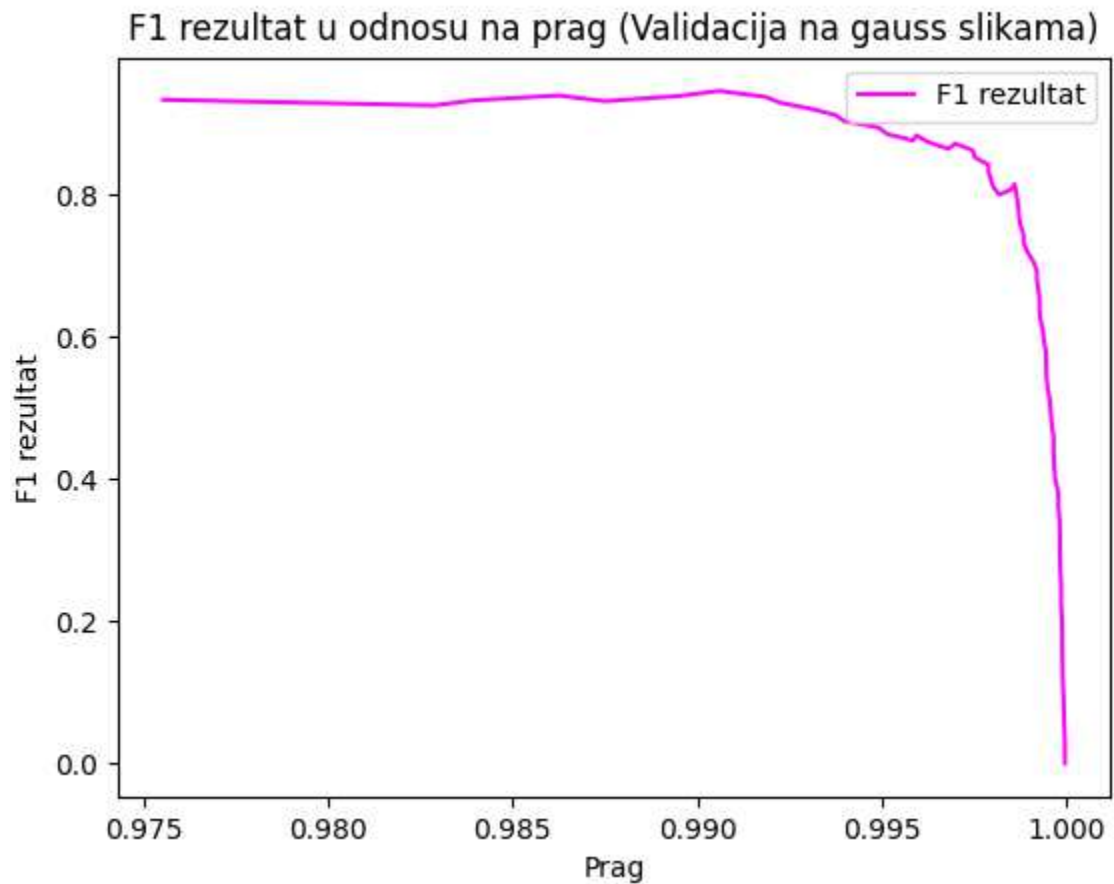
Sljedeći graf prikazuje točnost klasifikacijskog izlaza modela tijekom treninga. Vidljivo je da se točnost na trening skupu postupno povećava kako epohe napreduju, što ukazuje na uspješno učenje modela. Točnost na validacijskom skupu također pokazuje visoke vrijednosti, ali uz manje oscilacije, što sugerira da model općenito dobro generalizira na neviđene podatke.



Slika 86. Točnost klasifikacijskog izlaza tijekom treninga

Izvor: Izradio autor

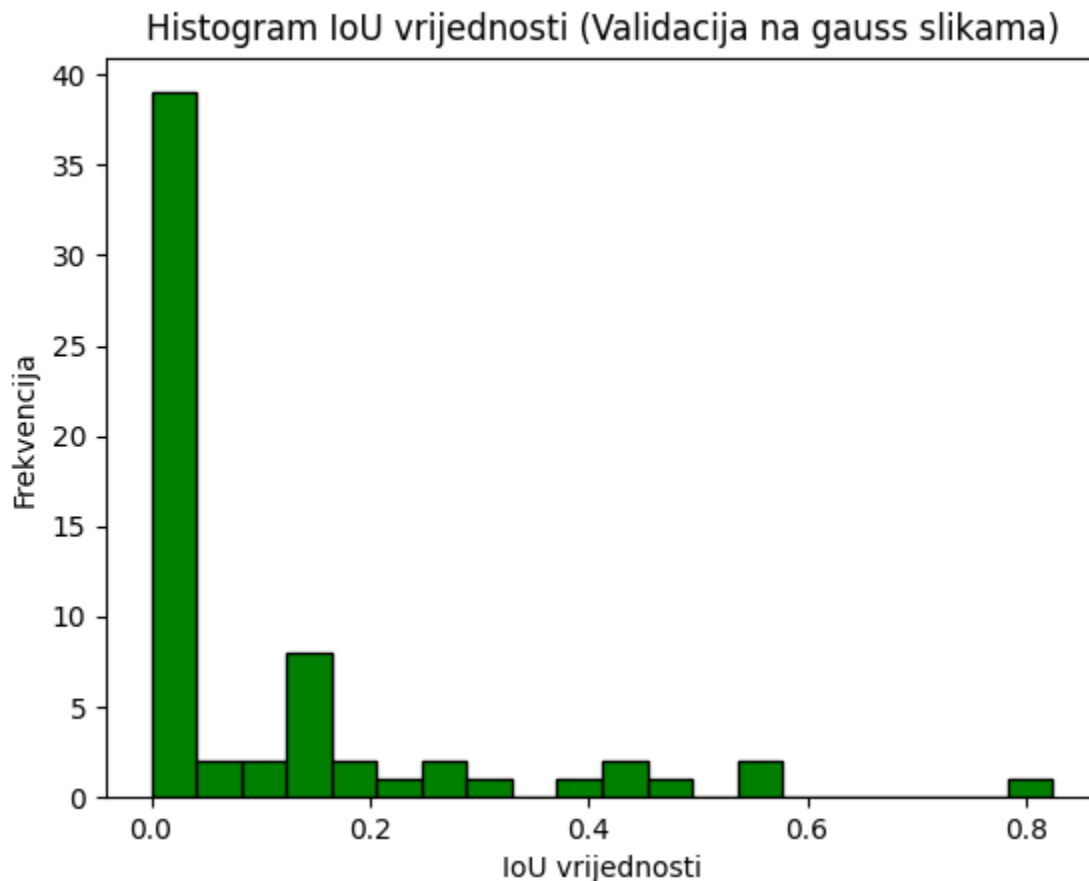
Nakon analize performansi treniranja modela na čistim podacima, potrebno je evaluirati i testirati na podacima s Gausovim šumom. Prvi graf prikazuje f1 rezultat modela kroz različite pragove detekcije. Primjećuje se da model održava visoku f1 vrijednost za većinu pragova, što ukazuje na dobru balansiranost. Primjećuje se značajan pad f1 rezultata pri visokim pragovima.



Slika 87. F1 rezultat u odnosu na prag

Izvor: Izradio autor

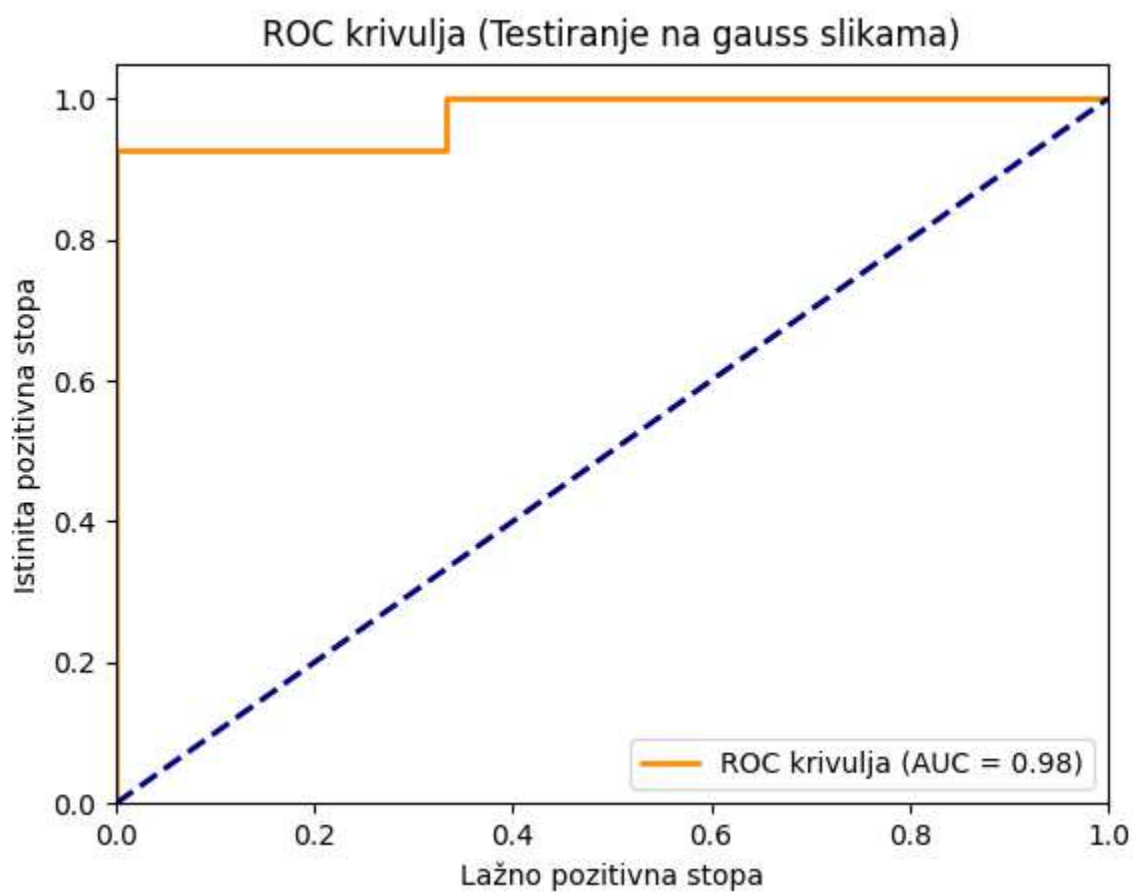
Histogram prikazuje distribuciju IoU vrijednosti između predikcija modela i stvarnih *bounding boxova*. Najveći nedostatak modela primjećuje se pri slaboj lokalizaciji na slikama s Gausovim šumom, većina predikcija nije dovoljno točna da bi se smatrala ispravnom.



Slika 88. Histogram IoU vrijednosti

Izvor: Izradio autor

Nakon što su analizirani validacijski rezultati, prelazi se na testne rezultate modela na slikama s dodanim Gausovim šumom. Budući da su grafovi testiranja relativno slični onima iz validacije, fokus će biti na dva ključna grafa: matricu zabune i ROC krivulju. Površina ispod krivulje iznosi 0.98, što ukazuje na visoku sposobnost modela da razlikuje pozitivne i negativne primjere, čak uz dodan šum.



Slika 89. ROC krivulja

Izvor: Izradio autor

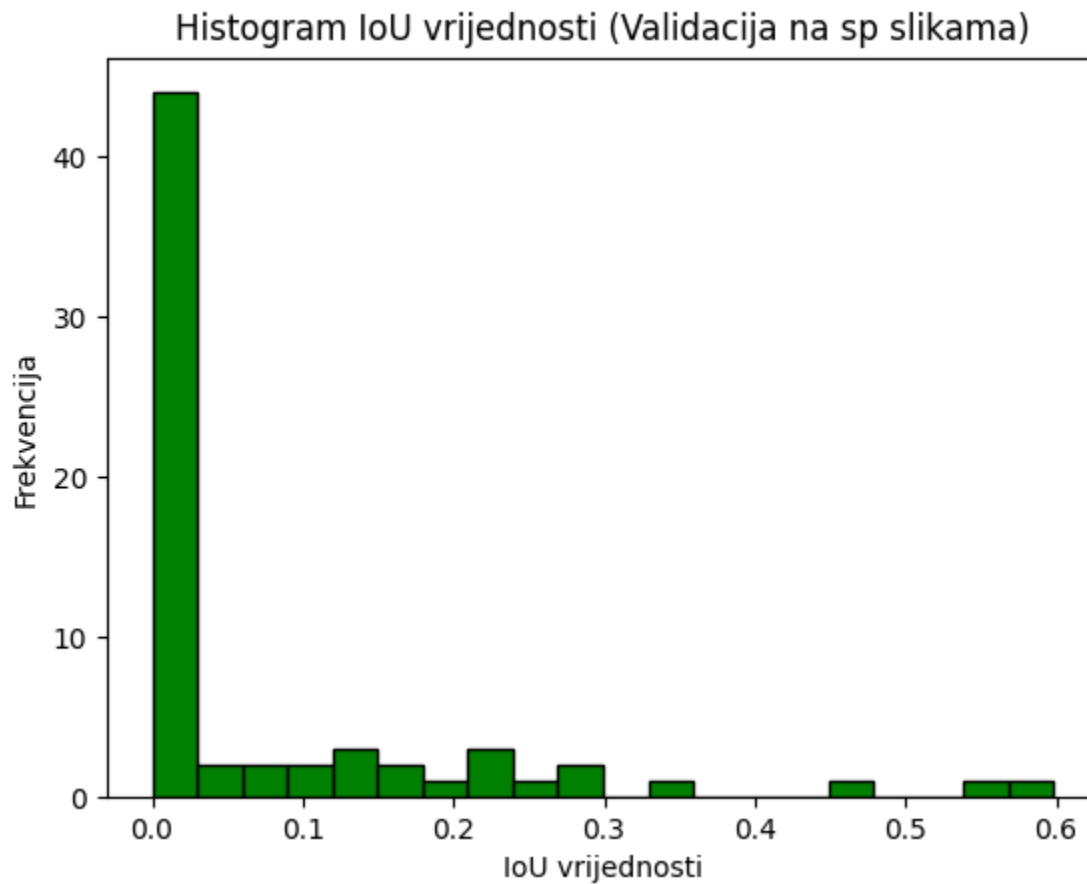
Matrica zabune prikazuje rezultate testiranja modela na slikama s dodanim Gaussovom šumom. Model je postigao 69 točno pozitivnih klasifikacija i 3 lažno pozitivna slučaja.



Slika 90. Matrica zabune

Izvor: Izradio autor

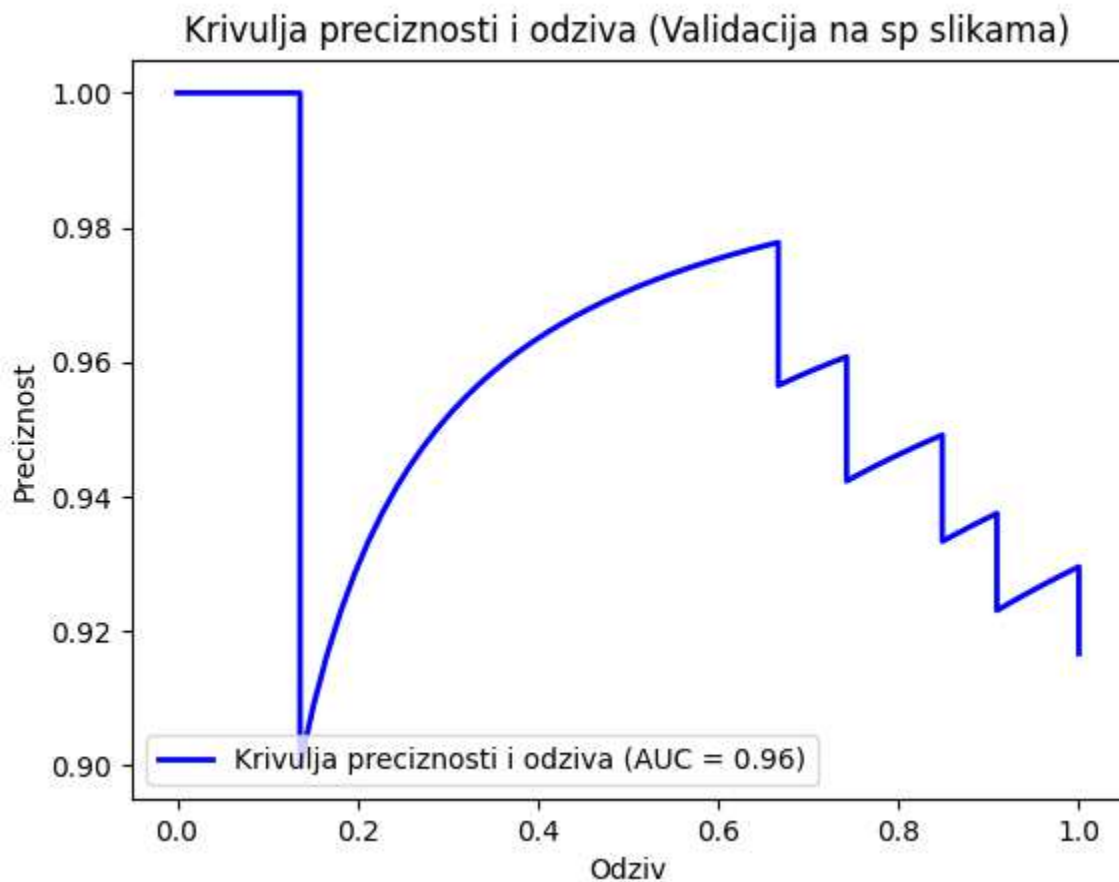
Nakon analize rezultata na Gaussovom šumu, potrebno je provesti identičnu analizu na slikama s *salt & pepper* šumom. Histogram prikazuje sličnu situaciju kao i kod slika sa Gausovim šumom. Model u velikom slučaju ne predviđa točno *bounding boxove* blizu stvarnim oznakama, a situacija je još više istaknuta kod slika s *salt & pepper* šumom.



Slika 91. Histogram IoU vrijednosti

Izvor: Izradio autor

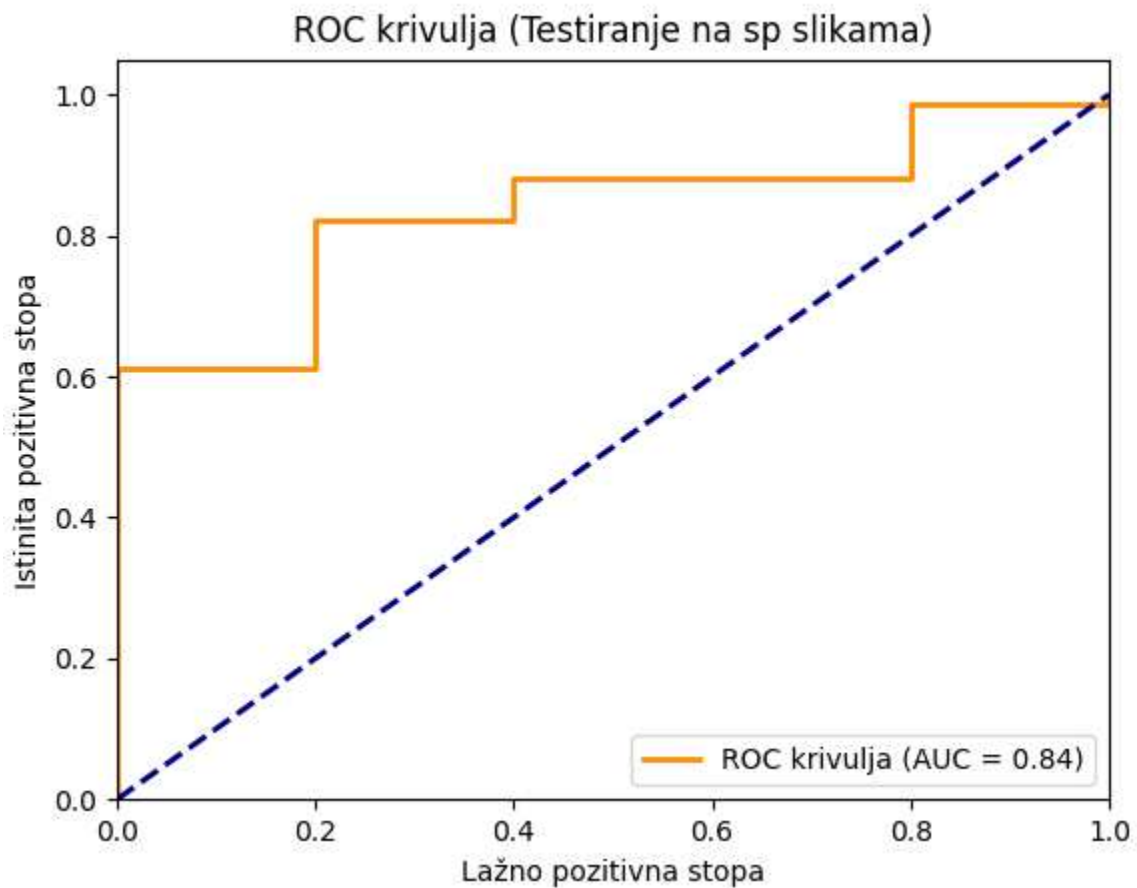
Krivulja preciznosti i odziva prikazuje mijenjanje metrika pri relativno različitim pragovima detekcije. Preciznost ostaje visoka, dok odziv nešto opada s vremenom. Uspoređujući s Gausovim šumom, vrijednost kod *salt & pepper* slika je niža.



Slika 92. Krivulja preciznosti i odziva

Izvor: Izradio autor

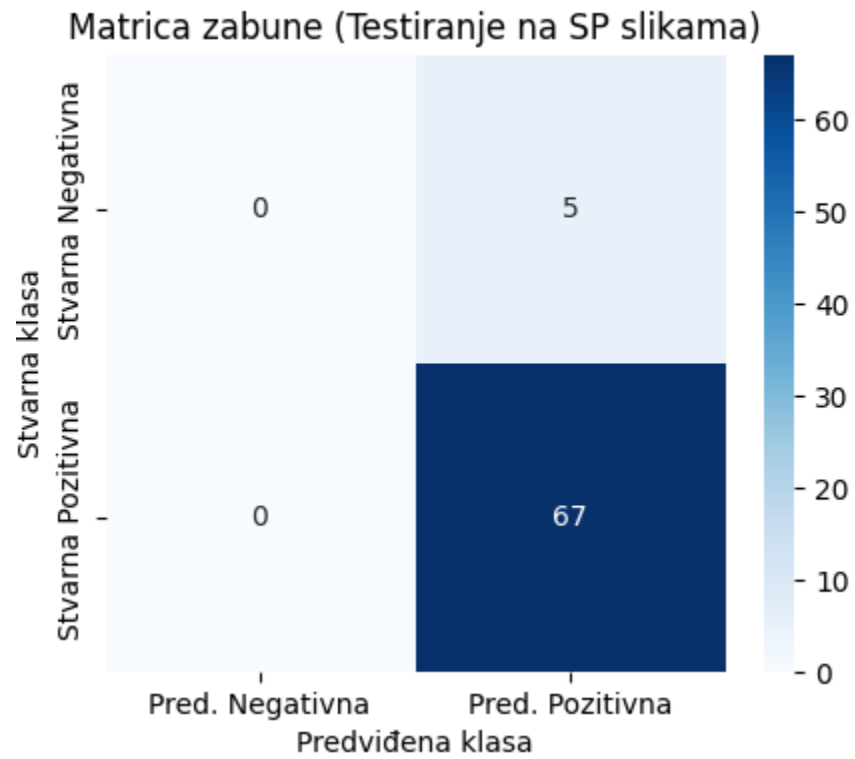
Prilikom testiranja modela, ROC krivulja od 0.84 sugerira dobru klasifikacijsku sposobnost modela, no i dalje slabiju nego što je bila kod Gaussovog šuma. Prilično je jasno kako *salt & pepper* definitivno više utječe na krajnji rezultat modela.



Slika 93. ROC krivulja

Izvor: Izradio autor

Matrica zabune prikazuje kako je model točno klasificirao 67 pozitivnih primjera, dok je imao 5 lažno pozitivnih predikcija. Odziv ostaje visok, no pojavljuje se nešto veći broj lažno pozitivnih slučajeva.



Slika 94. Matrica zabune (Testiranje na SP slikama)

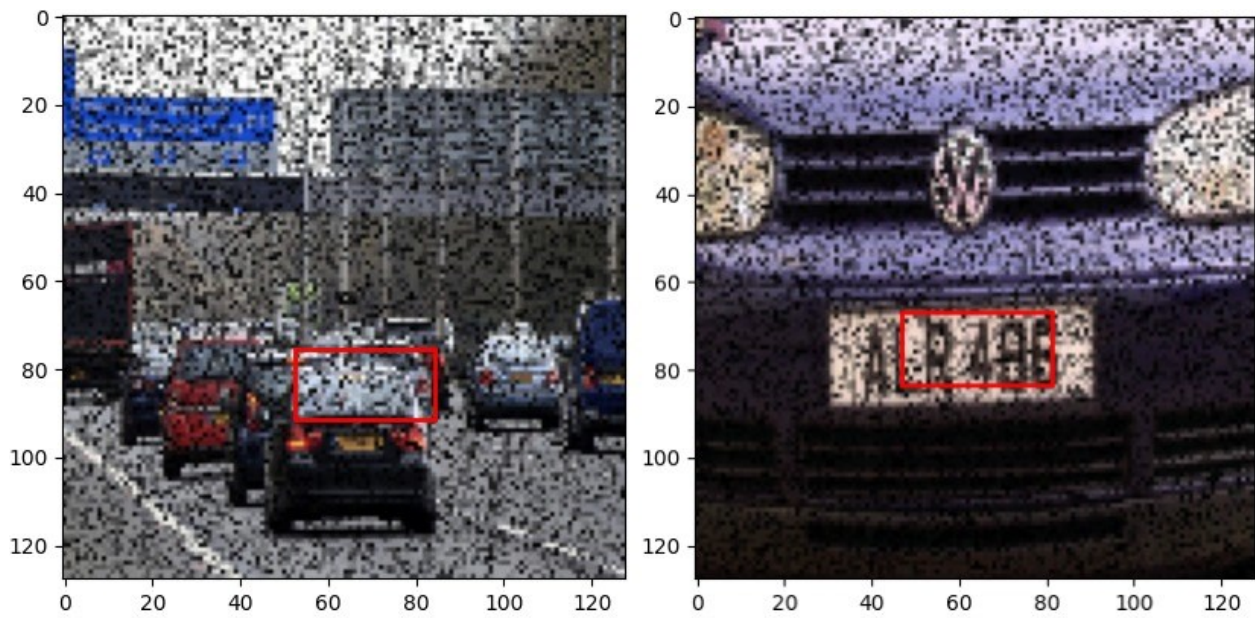
Izvor: Izradio autor

Slika 89. prikazuje detekciju vlastite neuronske mreže kod čistih slika, koje je u ovom slučaju vrlo precizno klasificirala, što se ne može reći za sliku 90. gdje pri dodavanju salt & pepper šuma, preciznost opada.



Slika 95. Prikaz detekcije vlastite neuronske mreže

Izvor: Izradio autor



Slika 96. Prikaz detekcije vlastite neuronske mreže (salt & pepper šum)

Izvor: Izradio autor

)

Rezultati evaluacije vlastite neuronske mreže pokazali su da model postiže visoku točnost u klasifikaciji slika, s vrlo dobrim odzivom i preciznošću na testnim skupovima, čak i uz prisutnost Gaussovog i *salt & pepper* šuma. ROC krivulje i F1 rezultati dodatno potvrđuju da model uspješno prepoznaje slike koje sadrže objekte od interesa.

Međutim, analiza IoU vrijednosti pokazuje da lokalizacija objekata nije na istoj razini kao klasifikacija – većina predikcija ima niske IoU vrijednosti, što znači da predloženi *bounding boxovi* često nisu precizno poravnati s ground truth oznakama. To sugerira da, iako model može ispravno prepoznati prisutnost objekta, njegova sposobnost preciznog lociranja objekata unutar slike ostaje ograničena.

Ovi rezultati naglašavaju važnost korištenja naprednijih, state-of-the-art modela poput Faster R-CNN, YOLO ili SSD, koji su optimizirani za istovremeno preciznu klasifikaciju i lokalizaciju objekata. Iako jednostavniji modeli mogu dati solidne rezultate u klasifikaciji, precizna detekcija objekata zahtijeva dublje arhitekture koje mogu efikasnije upravljati prostornim informacijama unutar slike.

5. Usporedba rezultata korištenih algoritama

Algoritam	Tip podataka	Preciznost	Odziv	F1	IoU	mAP	TP	FP	FN	TN
YOLO	Trening	0.72	0.73	0.69	0.75	0.76	63	1.0	37	0
YOLO	Validacija (Gauss)	0.83	0.94	0.80	0.6	0.83	55	0	45	0
YOLO	Testiranje (Gauss)	0.69	0.65	0.65	0.65	0.64	42	1.0	58	0
YOLO	Validacija (S&P)	0.66	0.6	0.32	0.20	0.21	1	0	99	0
YOLO	Testiranje (S&P)	0.41	0.49	0.29	0.20	0.22	4	1.0	96	0
Faster R-CNN	Trening	0.89	0.92	0.89	0.75	0.88	300	0	0	35
Faster R-CNN	Validacija (Gauss)	0.77	0.77	0.76	0.64	0.76	56	1	8	7
Faster R-CNN	Testiranje (Gauss)	0.77	0.77	0.77	0.64	0.76	56	1	8	7
Faster R-CNN	Validacija (S&P)	0.17	0.17	0.17	0.14	0.17	12	0	54	6
Faster R-CNN	Testiranje (S&P)	0.15	0.15	0.15	0.12	0.15	11	0	56	5
SSD	Trening	0.97	0.46	0.62	0.80	0.46	67	2	79	10
SSD	Validacija (Gauss)	1.0	0.58	0.74	0.80	0.58	39	0	28	8
SSD	Testiranje (Gauss)	1.0	0.37	0.54	0.77	0.37	29	0	50	3
SSD	Validacija (S&P)	1.0	0.11	0.20	0.79	0.11	8	0	65	6
SSD	Testiranje (S&P)	0.88	0.10	0.17	0.75	0.10	7	1	66	5
Personalizirana neuronska mreža	Trening	0.88	1.0	0.88	0.45	0.88	75	5	10	10
Personalizirana neuronska mreža	Validacija (Gauss)	0.89	1.0	0.94	0.10	0.89	64	0	8	0
Personalizirana neuronska mreža	Testiranje (Gauss)	0.96	1.0	0.98	0.07	0.96	69	0	3	0
Personalizirana neuronska mreža	Validacija (S&P)	0.91	1.0	0.96	0.07	0.91	66	0	6	0
Personalizirana neuronska mreža	Testiranje (S&P)	0.94	1.0	0.96	0.08	0.94	67	0	5	0

Tablica 1. Usporedba rezultata korištenih algoritama

Izvor: Izradio autor

YOLO algoritam pokazao je zadovoljavajuće rezultate na čistim podacima, s preciznošću od 72 %, odzivom od 73 % i F1 rezultatom od 69 %. U prisutnosti Gaussovog šuma, performanse su se čak poboljšale, pri čemu preciznost doseže 83 %, odziv 94 %, dok je F1 rezultat 80 %. Međutim, prisutnost *salt & pepper* šuma izazvala je drastičan pad performansi, pri čemu je F1 rezultat pao na samo 32 %, uz vrlo nizak IoU od 21 %, što ukazuje na osjetljivost YOLO algoritma na ovaj tip degradacije slike.

Faster R-CNN je na čistim podacima ostvario vrlo dobre rezultate, s preciznošću od 89 %, odzivom od 92 % i F1 rezultatom od 89 %. U prisutnosti Gaussovog šuma, performanse su ostale relativno stabilne, s F1 rezultatom od 76 % i IoU od 0.76. Međutim, dodavanje *salt & pepper* šuma uzrokovalo je značajan pad performansi, pri čemu F1 rezultat pada na samo 17 %, uz vrlo nizak odziv i IoU 15 %. Ovi rezultati ukazuju na to da je Faster R-CNN otporniji na Gaussov šum, ali se ne može nositi s agresivnijim degradacijama poput *salt & pepper* šuma.

SSD algoritam pokazao se kao najtočniji model, ali istovremeno i najsporiji u izvođenju detekcija. Na čistim podacima postigao je preciznost od 97 %, dok je F1 rezultat iznosio visokih 80 %. U prisutnosti Gaussovog šuma, F1 rezultat ostaje stabilan (oko 80 %), dok je preciznost nešto niža (58 %). Međutim, prisutnost *salt & pepper* šuma znatno narušava performanse, pri čemu F1 rezultat pada na samo 11 %, uz IoU od 0.79. Unatoč tome, SSD pokazuje izuzetnu konzistentnost u predikcijama i najmanju osjetljivost na promjene u kvaliteti slike među testiranim modelima, što ga čini vrlo pouzdanim u uvjetima manje degradacije slike.

Vlastita neuronska mreža pokazala je relativno dobre rezultate u smislu klasifikacije objekata, ali ima značajne probleme s predviđanjem bounding boxova, što se vidi po niskim IoU vrijednostima. Iako postiže visoku preciznost od 88-91 % i F1 rezultat od 88-96 % na svim testiranim skupovima podataka, IoU ostaje vrlo nizak, što ukazuje na to da mreža ne uspijeva točno locirati registarske oznake unutar slike. Ovakvi rezultati pokazuju da, iako vlastita neuronska mreža može biti korisna u određenim zadacima, specijalizirani algoritmi poput YOLO, SSD-a i *Faster R-CNN*-a bolje su prilagođeni problemima detekcije objekata, gdje precizno lociranje objekta unutar slike igra ključnu ulogu.

Analizirani rezultati temelje se na skupu od 433 slike koje sadrže registarske oznake, uz dodatnih 45 negativnih primjera koji ne sadrže registarske oznake, a korišteni su za provjeru otpornosti modela na detekcije lažno pozitivnih uzoraka. Rezultati ukazuju na to da je SSD algoritam najtočniji, ali istovremeno i najsporiji u izvođenju predikcija, dok je YOLO algoritam brži, ali osjetljiviji na degradaciju slike, osobito u slučaju *salt & pepper* šuma. *Faster R-CNN* pokazao se pouzdanim na čistim podacima, ali se teško nosi s dodatnim šumom, dok vlastita neuronska mreža, iako postiže solidne rezultate u klasifikaciji, nije idealna za precizno određivanje lokacije registarskih oznaka.

Ovi rezultati pružaju dublji uvid u otpornost modela na različite uvjete i sugeriraju da se optimizacija hiperparametara, povećanje skupa podataka i dodatne tehnike augmentacije mogu koristiti za daljnje poboljšanje performansi modela u zadatku detekcije registarskih oznaka.

6. Zaključak

Kroz diplomski rad prikazana je implementacija i usporedba četiri različita algoritma za prepoznavanje registarskih oznaka: YOLO, *Faster R-CNN*, SSD i vlastite neuronske mreže. Cilj diplomskog rada je implementacija tri *state-of-the-art* pristupa navedenom problemu i međusobna usporedba rezultata, odnosno njihovu učinkovitost u prepoznavanju tablica na temelju metrika koje se inače koriste za takve potrebe. Također izvedena je implementacija personalizirane (vlastite) neuronske mreže te je testirana u usporedbi s ostalima. Model je treniran na čistim slikama, dok je njegova validacija i testiranje provedeno na skupu slika koje su uključivale Gaussov i *salt & pepper* šum. Poanta testiranja na slikama s dodanim šumom je simulacija pravih životnih uvjeta kao što su grmljavina, kiša, tuča, led te sve ostale smetnje koje mogu otežati detekciju kamera. Rezultati pokazuju da je SSD algoritam ostvario najvišu razinu točnosti, s vrlo preciznim detekcijama čak i u prisutnosti šuma, no njegova izvedba je sporija u usporedbi s YOLO-om. YOLO se pokazao bržim, ali je osjetljiviji na degradaciju slike, osobito na prisutnost salt & pepper šuma, gdje su mu performanse značajno pale. *Faster R-CNN* postigao je vrlo dobre rezultate na čistim podacima, no pokazao je osjetljivost na dodavanje šuma, pri čemu se značajno smanjuje njegova sposobnost detekcije registarskih oznaka, posebno u uvjetima ekstremne degradacije slike. Vlastita neuronska mreža postigla je solidne rezultate kod klasifikacije objekta, no imala je problema s preciznim lociranjem registarskih oznaka. Iako mreža uspješno prepoznaje prisutnost oznaka, ne postiže uvijek precizna predviđanja u usporedbi s modelima koji su specijalizirani za ovu vrstu zadatka. Iako prikazani rezultati daju dobar uvid u performanse analiziranih algoritama, moguće ih je dodatno poboljšati prilagodbom modela, optimizacijom hiperparametara, povećanjem veličine skupa podataka te korištenjem tehnika augmentacije. Veći i raznovrsniji dataset mogao bi poboljšati generalizacijsku sposobnost modela i povećati njihovu otpornost na različite vrste šuma i degradacija slike.

7. Literatura

Patel, C., Shah, D. and Patel, A., 2013. Automatic number plate recognition system (anpr): A survey. *International Journal of Computer Applications*, 69(9).

Qadri, M.T. and Asif, M., 2009, April. Automatic number plate recognition system for vehicle identification using optical character recognition. In 2009 international conference on education technology and computer (pp. 335-338). IEEE.

Silva, S.M. and Jung, C.R., 2018. License plate detection and recognition in unconstrained scenarios. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 580-596).

Gonzalez, Rafael C., Woods, Richard E.. *Digital Image Processing*, Global Edition. United Kingdom: Pearson Education, 2018.

ANPR International, History of ANPR. <https://www.anpr-international.com/history-of-anpr/> [Mrežno] [Pokušaj pristupa 15 srpanj 2024].

Contributing Writer. (Jun 14, 2024). Edge Detection in Image Processing: An Introduction. Roboflow Blog: [Mrežno] Dostupno na: <https://blog.roboflow.com/edge-detection/> [Pokušaj pristupa 20 srpanj 2024].

ANPR International, *History of ANPR*. Dostupno na: <https://www.anpr-international.com/history-of-anpr/> [Mrežno] [Pokušaj pristupa 25 srpanj 2024].

Towards Data Science, *A Comprehensive Guide to Convolutional Neural Networks: The ELI5 Way*. Dostupno na: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> [Mrežno] [Pokušaj pristupa 26 srpanj 2024].

LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. *nature*, 521(7553), pp.436-444.

MarTech Zone HR, *ReLU*. Dostupno na: <https://hr.martech.zone/acronym/relu/>
[Mrežno] [Pokušaj pristupa 1 kolovoz 2024].

DeepChecks, *Pooling Layers in CNN*. Dostupno na:
<https://deepchecks.com/glossary/pooling-layers-in-cnn/#:~:text=What%20are%20Pooling%20Layers%20in,decreasing%20the%20input's%20spatial%20size.> [Mrežno] [Pokušaj pristupa 5 kolovoz 2024].

SuperDataScience, *Convolutional Neural Networks (CNN) Step 3: Flattening*. Dostupno na: <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening> [Mrežno] [Pokušaj pristupa 8 kolovoz 2024].

GeeksforGeeks, *Introduction to Recurrent Neural Network*. Dostupno na: <https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/> [Mrežno] [Pokušaj pristupa 20 kolovoz 2024].

Karpathy, A. and Fei-Fei, L., 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3128-3137).

Analytics Vidhya, *Real-Time Object Detection with SSDs (Single Shot Multibox Detectors)*. Dostupno na: <https://www.analyticsvidhya.com/blog/2023/11/real-time-object-detection-with-ssds-single-shot-multibox-detectors/> [Mrežno] [Pokušaj pristupa 26 kolovoz 2024].

8. Popis slika

Slika 1. Tijek rada za obradu i klasifikaciju digitalnih podataka	4
Slika 2. Detekcija registarske tablice automobila	6
Slika 3. Arhitektura konvolucijske neuronske mreže	9
Slika 4. Prikaz ReLU funkcije na grafu	11
Slika 5. Prikaz izravnivanja.....	12
Slika 6. Rekurentna neuronska mreža.....	16
Slika 7. Prikaz različitih automobilskih tablica ovisno o regiji i državi.....	19
Slika 8. Nastanak šuma zbog neadekvatnog osvjetljenja kamere	20
Slika 9. Arhitektura YOLO algoritma	21
Slika 10. Pretvorba anotacija iz XML formata u YOLO format	23
Slika 11. Skripta za dodavanje salt & pepper šuma	28
Slika 12. Skripta za dodavanje Gaussovog šuma.....	32
Slika 13. Matrica zabune	34
Slika 14. F1 – prag samopouzdanja krivulje	35
Slika 15. Preciznost – prag povjerenja krivulja	36
Slika 16. Odziv - pouzdanost krivulja	37
Slika 17. Preciznost – odziv krivulja.....	38
Slika 18. Točnost predikcije modela po tablici	40
Slika 19. Matrica zabune validacije na podacima s Gausovim šumom	41
Slika 20. preciznost – prag pouzdanosti krivulja	42
Slika 21. F1 – prag pouzdanosti krivulja	43
Slika 22. Preciznost - odziv krivulja	44
Slika 23. Validacijski rezultat primjene Gaussovog šuma	45
Slika 24. Prikaz jačine Gaussovog šuma.....	46
Slika 25. Matrica zabune za testiranje - Gaussov šum	47
Slika 26. Krivulja f1 - povjerenje pri testiranju	48
Slika 27. Krivulja preciznost – povjerenje pri testiranju šuma	49
Slika 28. Krivulja preciznost - odziv pri testiranju šuma	50
Slika 29. Matrica zabune - validacija kod sp šuma	51
Slika 30. Prikaz jačine salt & pepper šuma.....	52
Slika 31. Krivulja preciznost - odziv pri validaciji (salt & pepper šum).....	53
Slika 32. Krivulja f1 - povjerenje validacija (salt & pepper šum).....	54
Slika 33. Krivulja preciznost- povjerenje (salt & pepper šum).....	55
Slika 34. Matrica zabune - testiranje (salt & pepper šum)	56
Slika 35. Krivulja preciznost - odziv - testiranje (salt & pepper šumom)	57
Slika 36. Krivulja f1 - povjerenje (testiranje - salt & pepper šum).....	58
Slika 37. Krivulja preciznost - povjerenje (testiranje - salt & pepper šum)	59
Slika 38. Prepoznavanje slika s salt & pepper šumom	60
Slika 39. Montiranje i priprema okruženja.....	64

Slika 40. Učitavanje iz testnog skupa	66
Slika 41. Inicijalizacija Faster R-CNN modela.....	66
Slika 42. Trening modela	67
Slika 43. Postavljanje okruženja za evaluaciju	68
Slika 44. Isječak koda za evaluaciju	69
Slika 45. Slika. Učitavanje treniranog modela.....	69
Slika 46. Faster R-CNN gubitak kroz epohe	70
Slika 47. Metrički pokazatelji Faster R-CNN.....	71
Slika 48. Matrica zabune (trening).....	72
Slika 49. Metričke vrijednosti validacija (Gauss šum)	73
Slika 50. Matrica zabune (validacija) - Gauss šum	74
Slika 51. Metričke vrijednosti (testiranje) - Gaussov šum	75
Slika 52. Matrica zabune (testiranje) - Gaussov šum	76
Slika 53. Predikcije modela s Gausovim šumom	77
Slika 54. Metričke vrijednosti (validacija - salt & pepper).....	78
Slika 55. Matrica zabune (validacija - salt & pepper).....	79
Slika 56. Metričke vrijednosti (test - salt & pepper šum)	80
Slika 57. Matrica zabune (test - salt & pepper šum)	81
Slika 58. Neuspješno prepoznavanje slika (salt & pepper).....	82
Slika 59. Prikaz anchor-a	83
Slika 60. Kreiranje SSD modela	84
Slika 61. Postavljanje optimizatora	84
Slika 62. Primjer treninga za jednu epohu	84
Slika 63. Gubitak tijekom treninga	85
Slika 64. Preciznost kroz epohe	86
Slika 65. IoU kroz epohe.....	87
Slika 66. Odziv kroz epohe	88
Slika 67. Preciznost/Odziv/F1 u odnosu na prag	89
Slika 68. IoU u odnosu na prag (val - Gauss).....	90
Slika 69. IoU u odnosu na prag	91
Slika 70. Preciznost / odziv / f1 (test - Gauss)	92
Slika 71. Predikcije SSD modela na Gaussovom šumu	93
Slika 72. Preciznost / odziv / f1 (salt & pepper šum).....	94
Slika 73. IoU u odnosu na prag (salt & pepper - val)	95
Slika 74. Preciznost / Odziv / F1 (test - salt & pepper)	96
Slika 75. Neuspješne detekcije modela na slikama sa salt & pepper šumom	97
Slika 76. Instalacija i unos potrebnih biblioteka	98
Slika 77. Učitavanje i pretprocesiranje slika.....	99
Slika 78. Preprocesiranje podataka	100
Slika 79. Podjela podataka	101
Slika 80. Arhitektura neuronske mreže	102
Slika 81. Treniranje modela	102
Slika 82. Evaluacija modela.....	103

Slika 83. Ukupni gubitak tijekom treninga.....	103
Slika 84. F1 rezultat tijekom treninga.....	104
Slika 85. Gubitak klase tijekom treninga.....	105
Slika 86. Točnost klasifikacijskog izlaza tijekom treninga.....	106
Slika 87. F1 rezultat u odnosu na prag.....	107
Slika 88. Histogram IoU vrijednosti.....	108
Slika 89. ROC krivulja.....	109
Slika 90. Matrica zabune.....	110
Slika 91. Histogram IoU vrijednosti.....	111
Slika 92. Krivulja preciznosti i odziva.....	112
Slika 93. ROC krivulja.....	113
Slika 94. Matrica zabune (Testiranje na SP slikama).....	114
Slika 95. Prikaz detekcije vlastite neuronske mreže.....	115
Slika 96. Prikaz detekcije vlastite neuronske mreže (salt & pepper šum).....	115

9. Popis tablica

Tablica 1. Usporedba rezultata korištenih algoritama	117
--	-----

Sažetak

U ovom diplomskom radu detaljno je istražen problem prepoznavanja registarskih oznaka, pri čemu su implementirana i uspoređena tri različita algoritma: YOLO, *Faster R-CNN* i SSD, uz implementaciju vlastite neuronske mreže. Cilj rada bio je istražiti performanse ovih algoritama u zadatku detekcije registarskih oznaka te analizirati njihovu otpornost na degradirane slike. Modeli su trenirani na skupu od 433 slike koje sadrže registarske oznake, dok je dodatno uključeno 45 slika bez oznaka kao negativni primjeri, s ciljem poboljšanja otpornosti modela na lažne detekcije. Nakon inicijalnog treniranja na čistim podacima, modeli su validirani i testirani na slikama s dodanim Gausovim i *salt & pepper* šumom kako bi se procijenila njihova otpornost na realne uvjete degradacije slike.

Analiza rezultata pokazala je da SSD algoritam ostvaruje najviše rezultate u preciznosti, dok YOLO postiže bolju brzinu detekcije. SSD se pokazao otpornijim na šum, dok je YOLO osjetljiviji na degradirane slike, ali i dalje održava solidne rezultate. *Faster R-CNN* je dao balansirane rezultate između SSD-a i YOLO-a, no pokazao je osjetljivost na šum, što ga čini manje pouzdanim u uvjetima degradirane slike. Vlastita neuronska mreža postigla je solidne rezultate u prepoznavanju registarskih oznaka, no zbog slabijih performansi u određivanju *bounding boxova* (što se vidi po nižem IoU rezultatu), pokazalo se da je korištenje specijaliziranih arhitektura poput SSD-a, YOLO-a i *Faster R-CNN-a* bolja opcija za ovaj zadatak.

Uočeno je da otpornost modela na šum varira, pri čemu se SSD pokazao najstabilnijim, dok YOLO nudi brži, ali osjetljiviji pristup. Rezultati ukazuju na to da se performanse svih modela mogu dodatno poboljšati prilagodbom hiperparametara, augmentacijom podataka te korištenjem većih i raznovrsnijih skupova podataka. Ova istraživanja mogu poslužiti kao smjernice za buduća unaprjeđenja algoritama za detekciju registarskih oznaka, osobito u realnim uvjetima gdje degradacija slike može utjecati na konačne rezultate.

Ključne riječi: prepoznavanje registarskih oznaka, YOLO, *Faster* R-CNN, SSD, neuronske mreže, otpornost na šum, Gaussov šum, *salt & pepper* šum, prilagodba algoritma, praktična primjena.

Abstract

This thesis presents an in-depth study of the problem of license plate recognition by implementing and comparing three different algorithms: YOLO, Faster R-CNN, and SSD, alongside a custom neural network. The objective of this research was to evaluate the performance of these algorithms in license plate detection and analyze their robustness to image degradation. The models were trained on a dataset of 433 images containing license plates, with an additional 45 images without plates included as negative examples to improve model resilience against false detections. After the initial training on clean images, the models were validated and tested on images with added Gaussian and salt & pepper noise to assess their performance under real-world image degradation conditions.

The results indicate that SSD achieved the highest accuracy, while YOLO provided faster detection. SSD demonstrated greater robustness to noise, whereas YOLO exhibited higher sensitivity to degraded images but maintained solid performance. Faster R-CNN yielded balanced results between SSD and YOLO but showed vulnerability to noise, making it less reliable in degraded conditions. The custom neural network produced satisfactory results in license plate recognition, but its weaker performance in bounding box prediction (as reflected in its lower IoU score) suggests that specialized architectures such as SSD, YOLO, and Faster R-CNN are more suitable for this task.

The study highlights variations in model robustness to noise, with SSD emerging as the most stable, while YOLO offers a faster but more sensitive approach. The findings emphasize that all models could achieve improved results through hyperparameter tuning, data augmentation, and the use of larger and more diverse datasets. These insights serve as guidelines for future enhancements in license plate detection

algorithms, particularly in real-world scenarios where image degradation can impact final outcomes.

Keywords: license plate recognition, YOLO, Faster R-CNN, SSD, neural networks, noise robustness, Gaussian noise, salt & pepper noise, model adaptation, real-world applications.