



UNIVERZITET CRNE GORE  
ELEKTROTEHNIČKI FAKULTET



Vuk Slavić

**INTEGRACIJA *YOLOV8* I *PYTESSERACT-A* U  
CILJU POBOLJŠANJA DIGITALIZACIJE  
DOKUMENATA U BANKARSTVU**

- MASTER RAD -

Podgorica, 2024. godine

## PODACI I INFORMACIJE O STUDENTU

Ime i prezime: **Vuk Slavić**

Datum i mjesto rođenja: **2. avgust 2000. godine u Podgorici**

Naziv završenog osnovnog studijskog programa i godina završetka studija:

**Elektronika, telekomunikacije i računari, 2022. godine**

## INFORMACIJE O MASTER RADU

Naziv master studija: **Postdiplomske akademske studije, odsjek**

**Elektronika, telekomunikacije i računari, smjer Računari**

Naslov rada: **Integracija YOLOv8 i PyTesseract-a u cilju poboljšanja digitalizacije dokumenata u bankarstvu**

Fakultet na kojem je rad odbranjen: **Elektrotehnički fakultet Podgorica**

## UDK, OCJENA I ODBRANA MASTER RADA

Datum prijave master rada: **14. maj 2024. godine**

Datum sjednice Vijeća na kojoj je prihvaćena tema: **16. septembar 2024. godine**

Mentor: **Prof. dr Vesna Popović Bugarin**

Komisija za ocjenu/odbranu rada:

1. **Prof. dr Miloš Daković**
2. **Prof. dr Vesna Popović Bugarin**
3. **Doc. dr Miloš Brajović**

Lektor: **Jelena Šaković**

Datum odbrane:

Datum promocije:

## Etička izjava

(U skladu sa članom 22 Zakona o akademskom integritetu i članom 18 Pravila studiranja na master studijama)

Potpisani Vuk Slavić

Broj indeksa: 2/22

Izjavljujem pod krivičnom i materijalnom odgovornošću da je magistarski rad pod naslovom: „**Integracija YOLOv8 i PyTesseract-a u cilju poboljšanja digitalizacije dokumenata u bankarstvu**” moje originalno djelo.

Podnosilac izjave:



Vuk Slavić

Podgorica, 29. oktobar 2024. godine

# Predgovor

Ova teza predstavlja kulminaciju mojih studija na Elektrotehničkom fakultetu u Podgorici i bila je akademski obogaćujuće iskustvo. Interesovanje za vještačku inteligenciju počelo je tokom angažmana u Uhura Solutions firmi, gdje sam se prvi put susreo sa modelima za detekciju objekata i prepoznavanje teksta.

Želio bih da izrazim najdublju zahvalnost mentorki, Vesni Popović-Bugarin, na njenim mnogobrojnim savjetima, konstruktivnim povratnim informacijama i podršci. Posebnu zahvalnost dugujem i Uhura Solutions firmi na pružanju neophodnih resursa i podataka za ovo istraživanje.

Pisanje ove teze bilo je putovanje ispunjeno učenjem i izazovima, ali mi je pomoglo da rastem i kao inženjer mašinskog učenja i kao osoba. Nadam se da će ovo istraživanje doprinijeti automatizaciji digitalizacije dokumenata i inspirisati buduće studije na ovu temu.

## Sažetak

Bankarski sektor se često suočava sa značajnim ručnim poslom prilikom obrade osjetljivih dokumenata, posebno onih u tabelarnim formatima, gdje su tačnost i usklađenost sa propisima najvažniji. Ova teza istražuje primjenu vještačke inteligencije u svrhu poboljšanja efikasnosti i preciznosti obrade dokumenata u bankama. Predloženo rješenje integriše savremeni model detekcije objekata *YOLOv8*, sa *PyTesseract OCR* modelom kako bi se unaprijedila digitalizacija tabelarnih dokumenata, prevazilazeći mogućnosti tradicionalnih ručnih metoda i samostalnih *OCR* rješenja.

Istraživanje pokazuje da kombinovanje *YOLOv8* sa *PyTesseract*-om primjetno podiže i tačnost i efikasnost digitalizacije dokumenata u bankarskoj industriji. Ovaj automatizovani pristup značajno smanjuje potrebu za ručnim pregledom, što dovodi do skraćenog vremena obrade dokumenata i omogućava preraspodjelu ljudskih resursa ka strateškim zadacima, čime se povećava ukupna produktivnost. Implementacija sistema ima značajan pozitivan uticaj na operativnu efikasnost što ga čini vrijednim alatom za banke.

Integracija predstavljena u ovom radu označava značajan napredak u tehnologiji digitalizacije dokumenata. Rezultati ne samo da potvrđuju efikasnost predloženog pristupa, već ukazuju i na moguće puteve za dalja poboljšanja. Uz tekući razvoj i prilagođavanje, takvi inteligentni sistemi obećavaju da će značajno smanjiti ručno opterećenje i pojednostaviti operaciju obrade dokumenata. Ovaj napredak otvara put za šire usvajanje naprednih rješenja baziranih na vještačkoj inteligenciji, a za upravljanje dokumentima u različitim industrijama.

*Ključne riječi:* digitalizacija dokumenata, *YOLO*, detekcija objekata, *PyTesseract*, *OCR*

## Abstract

The banking sector often faces substantial manual workloads in processing sensitive documents, especially those in tabular formats, where accuracy and regulatory compliance are paramount. This thesis explores the application of artificial intelligence to enhance document processing efficiency and precision within banks. The proposed solution integrates the modern object detection model *YOLOv8* with the *PyTesseract* OCR model to enhance the digitization of tabular documents, surpassing the capabilities of traditional manual methods and standalone OCR solutions.

The research successfully demonstrates that combining *YOLOv8* with *PyTesseract* OCR significantly elevates both the accuracy and efficiency of document digitization in the banking industry. This automated approach markedly reduces the need for manual review, leading to decreased document processing times and allowing for the reallocation of human resources to more strategic tasks, thereby boosting overall productivity. The system's implementation has a notable positive impact on operational efficiency, which makes it a valuable tool for banks.

The integration presented in this thesis signifies a meaningful advancement in document digitization technology. The results not only confirm the effectiveness of the proposed approach, but also indicate possible directions for further improvements. With ongoing development and adaptation, such intelligent systems promise to substantially lessen manual workloads and streamline document processing operations. This progress opens the door to the broader adoption of advanced document management solutions based on artificial intelligence, across various industries.

*Keywords:* document digitalization, YOLO, object detection, PyTesseract, OCR

# Sadržaj

1. UVOD .....	9
1.1. Hipoteza .....	13
1.2. Očekivani rezultati istraživanja .....	14
2. DETEKCIJA OBJEKATA NA SLICI I PRIMJENA <i>OCR</i> TEHNOLOGIJE ZA DETEKCIJU TEKSTA.....	15
2.1. Neuralne mreže, osnova dubokog učenja.....	16
2.2. <i>YOLO (You Only Look Once)</i> algoritam .....	22
2.2.1. Kompjuterska vizija .....	22
2.2.2. Uvod u detekciju objekata .....	23
2.2.3. Uvod u <i>YOLO</i> algoritam .....	25
2.2.4. <i>YOLO</i> verzije: .....	29
2.2.5. <i>YOLOv8</i> .....	30
2.3. <i>PyTesseract OCR</i> algoritam .....	35
2.3.1. Uvod u <i>OCR</i> .....	35
2.3.2. <i>PyTesseract: Python</i> alatka za <i>OCR</i> .....	38
3. PRIMJENA VJEŠTAČKE INTELIGENCIJE U OBRADI DOKUMENATA IZ BANKARSKOG SEKTORA 40	
3.1. Aktuelna rješenja za obradu dokumenata u bankarstvu .....	41
3.2. Izazovi specifični za bankarski sektor .....	41
3.3. Ograničenja postojećih rješenja za automatizaciju.....	42
3.4. Potreba za integrisanim <i>AI</i> pristupom.....	43
4. OPIS SISTEMA ZA DIGITALIZACIJU DOKUMENATA .....	44
4.1. Priprema skupa podataka .....	46
4.2. Trening modela i optimizacija .....	47
4.3. Kriterijumi za evaluaciju performansi .....	53
5. DIZAJN I IMPLEMENTACIJA SISTEMA .....	54
5.1. Pregled sistema .....	54
5.2. Detaljan tok procesa .....	54
5.2.1. Početno podešavanje i učitavanje slike.....	54
5.2.2. Otkrivanje strukture dokumenta .....	54
5.2.3. Uklanjanje pogrešnih detekcija .....	55
5.2.4. Prepoznavanje teksta .....	56
5.2.5. Analiza i obrada tabele.....	56
5.2.6. Napredno rukovanje tabelama .....	58
5.2.7. Strukturiranje podataka i <i>JSON</i> izlaz .....	58

5.3. Šematski prikaz sistema .....	59
6. EKSPERIMENTALNI REZULTATI I ANALIZA .....	61
6.1. Evaluacija performansi <i>YOLO</i> modela .....	61
6.2. Preciznost ekstrakcije teksta .....	65
6.3. Osvrt na istraživačka pitanja .....	67
7. ZAKLJUČAK I BUDUĆI RAD .....	68
7.1. Rezime ključnih doprinosa i rezultata .....	68
7.2. Ograničenja predloženog sistema i budući rad .....	68
7.4. Zaključak .....	69
RJEČNIK.....	70
DODATAK.....	84
LITERATURA.....	104

# 1. UVOD

Era digitalne transformacije zahtijeva inovativna rješenja zasnovana na naprednim tehnologijama za poboljšanje operativne efikasnosti u svim industrijama, posebno u sektorima opterećenim ručnim rukovanjem podacima, kao što je bankarstvo. Motivacija ovog istraživanja proizlazi iz značajnog obima ručnog rada potrebnog za obradu bankarskih dokumenata, posebno onih u tabelarnom formatu, kao i zbog osjetljivosti ovih podataka i značaja tačnosti sporevedene obrade. Zadaci u obradi dokumenata, iako ključni za donošenje odluka i praćenje klijenata, oduzimaju mnogo vremena i podložni su greškama, što dovodi do neefikasne alokacije ljudskih resursa za repetitivne zadatke koji se mogu automatizovati.

Ovo istraživanje se bavi izazovima digitalizacije dokumenata u bankarskom sektoru, gdje su brzina, preciznost i poštovanje regulatornih standarda od izuzetne važnosti. Banke svakodnevno rukuju velikim brojem različitih dokumenata (od jednostavnih obrazaca do složenih finansijskih izvještaja), a ključni problem nije samo njihova digitalizacija, već i tačno izvlačenje podataka. Mašinsko učenje (poglavlje 2), naročito kroz napredne modele kao što su *YOLOv8* (poglavlje 2.2.5) za detekciju objekata i *PyTesseract* (poglavlje 2.3.2) za optičko prepoznavanje teksta, nudi potencijalna rješenja za ove izazove.

Integracija *YOLOv8* i *PyTesseract*-a omogućava precizno prepoznavanje i izdvajanje relevantnih informacija iz dokumenata, što značajno poboljšava brzinu i tačnost obrade. Glavni doprinos ovog istraživanja jeste primjena ovih tehnologija na specijalno pripremljenom skupu podataka<sup>1</sup>, koji uključuje različite tipove bankarskih dokumenata sa složenim strukturama. Time se osigurava da sistem može efikasno da prepozna i izvuče kako tekstualne podatke, tako i složene tabele, specifične za bankarstvo.

Posebna pažnja posvećena je prilagođavanju algoritama različitim vrstama dokumenata, čime se postiže fleksibilnost i visoka tačnost bez obzira na složenost ili varijabilnost u njihovom dizajnu. Ovo istraživanje doprinosi razvoju naprednih metoda za obradu dokumenata u bankarskom sektoru, otvarajući put za dalju digitalnu transformaciju i povećanje operativne efikasnosti kroz automatizaciju.

---

<sup>1</sup> Pojmovi podvučeni na ovaj način su pojašnjeni na kraju rada, u rječniku.

Složenost struktura dokumenata, koje se značajno razlikuju među zemljama, poslovnim sektorima, pa čak i pojedinačnim kompanijama, zahtijeva primjenu sofisticiranih tehnika za njihovu digitalizaciju i izdvajanje ključnih informacija. Digitalizacija ovakvih dokumenata nosi niz izazova, obuhvata precizno prepoznavanje teksta i komponenti tabela, smanjenje šuma (smetnji za prepoznavanje karaktera) na površini dokumenta i izvršavanje optičkog prepoznavanja karaktera – *OCR* (poglavlje 2.3.1). Ovi zadaci su stimulisali širok spektar akademskih i industrijskih istraživačkih poduhvata u cilju usavršavanja procesa digitalizacije dokumenata. Da bi poboljšali kvalitet slike dokumenta za dalje faze obrade kao što je *OCR*, naučnici su primjenjivali skup morfoloških tehnika – uključujući dilataciju, eroziju i otvaranje – demonstrirajući efikasnost ovih metoda u uklanjanju zamućenja i ispravljanju izobličenja slike. Ove tehnike igraju vitalnu ulogu u izolovanju dijelova slike u kojima je sadržan tekst za analizu [1]. Proces razgraničenja tekstualnih regiona primjenom operacija dilatacije i erozije je sistematski istražen, što je dovelo do razvoja algoritama dizajniranih za provjeru strukture dokumenata [2]. Paralelno, zadatku identifikacije tekstualnog regiona u slikama dokumenata pristupilo se analizom razlika između morfoloških operacija otvaranja i zatvaranja [3].

Literatura dalje naglašava korišćenje tehnika obrade slike u svim aspektima sistema za obradu dokumenata. Inovacije u ovoj oblasti uključuju razvoj sistema sposobnih za otkrivanje i izdvajanje podataka o fakturama koristeći tehnike podudaranja šablona. Početni koraci obrade u ovim sistemima uključuju eliminaciju suvišnih elemenata pozadine i korekciju orijentacije dokumenata, postavljajući temelj za konverziju slika u mašinski čitljiv tekst putem *OCR* tehnologije [4]. Analitičko istraživanje strukture dokumenata dovelo je do mogućnosti klasifikacije, poput paragrafa, grafika, slika i tabela, uotrebnom metoda obrade slike, a naglašavajući prilagodljivost ovih tehnika različitim formatima dokumenata [5].

Domen dubokog učenja (poglavlje 2) je iskorišćen za rješavanje izazova u otkrivanju tabela u dokumentima – zadatak komplikovan zbog varijabilnosti u formatima i stilovima tabela, kao i zbog povremenog odsustva ivica tabele. Modeli detekcije objekata, kao i algoritmi dubokog učenja posebno prilagođeni detekciji tabela, su primijenjeni sa zapaženim uspjehom [6]. Na primjer, napravljene su adaptacije modela za detekciju objekata *YOLOv3* kako bi se prilagodile njegove mogućnosti za otkrivanje

tabele, uključujući prilagođavanja parametara modela na osnovu dimenzija tabele utvrđenih algoritmima za grupisanje. Ove adaptacije su dopunjene koracima naknadne obrade koji imaju za cilj poboljšanje preciznosti razgraničavanja tabele [7]. Konvolucione neuralne mreže i modeli dubokog učenja (poglavlje 2), kao što je *Mask R-CNN*, pokazali su se efikasnim za detekciju tabele, pokazujući visoku tačnost na različitim skupovima podataka [8]-[11].

Istraživanja sistema za obradu faktura za komercijalnu upotrebu dovela su do stvaranja sistema zasnovanih na *vebu*, dizajniranih da upravljaju fakturama izdvajanjem i sistematskim skladištenjem informacija izvedenih iz slika faktura. Ovi sistemi koriste i tehnike obrade slika i *OCR* tehnologije koje omogućavaju biblioteke kao što su *OpenCV* i *Tesseract*, demonstrirajući njihovu pogodnost za male poslovne operacije [12]. *OCR* faza je naglašena kao ključni element procesa digitalizacije, sa prilagođenim algoritmima koji postižu visok nivo tačnosti u prepoznavanju karaktera [13]-[14].

Imperativ da se iz podataka generisanih *OCR*-om izdvoje korisne informacije podstakao je razvoj sistema dizajniranih da minimiziraju ljudsku intervenciju. Ovi sistemi su u početku trenirani samo na jednom uzorku na kom bi korisnik označio šta od teksta treba biti izdvojeno. Prilikom obrađivanja novih dokumenata, ti sistemi se postepeno usavršavaju čime se povećava njihova svestranost [15]. Metodologije mašinskog učenja su iskorišćene za izdvajanje ključnih informacija iz faktura koristeći algoritme koji obrađuju višeslojne, nestrukturirane dokumente, bez oslanjanja na unaprijed definisane šablone. Ovaj pristup je proširen upotrebom tehnika izdvajanja teksta i karakteristika, zajedno sa neuralnim mrežama, a sa ciljem da se precizno odrede ključna polja fakture [16]. Pored toga, konvolucioni modeli zasnovani na grafovima, koji su efikasni i robusni u rukovanju složeno strukturiranim dokumenata, koriste se za izdvajanje i obradu ključnih informacija iz bilo kog dijela dokumenta. Zajedno sa tekstom i njegovom lokacijom, originalna slika se koristi kao ulaz na model dubokog učenja koji uključuje *CNN*, *BiLSTM* (eng. *Bidirectional Long Short-Term Memory*) i *Graph Convolution* [17] slojeve. Ovi modeli su testirani na različitim tipovima dokumenata, uključujući medicinske fakture i karte za voz, pokazujući njihov kapacitet da isporuče visok nivo tačnosti u identifikaciji ključnih polja.

Obrada dokumenata od značaja obuhvata mnoge faze. Međutim, sistemi za obradu dostupni u literaturi se najčešće fokusiraju na jednu temu kao što je detekcija tabele,

detekcija teksta i primjena *OCR* tehnologije na slici dokumenta. Pored toga, razvijeni sistemi za obradu dokumenata ne pokrivaju sve zahtjeve preduzeća. Inspirisan ovim nedostatkom u literaturi, razvijen je sistem s ciljem da pokrije sve potrebe za digitalizacijom dokumenata u bankama, koristeći zajedno tehnike detekcije objekata i *OCR*-a.

Inovativnost ovog istraživanja zasnovana je na prilagođenom treningu *YOLOv8* modela na pažljivo odabranom skupu podataka od 10 000 slika, čime se uzima u obzir raznolikost i složenost bankarskih dokumenata. Cilj treninga je optimizacija tačnosti modela u otkrivanju pasusa i tabela unutar dokumenata, što je ključno za precizno izdvajanje informacija. U tu svrhu, cilj je maksimiziranje tačnosti *YOLO* modela, s obzirom na to da će *OCR* model biti korišćen u svom postojećem stanju, bez daljeg treninga i podešavanja. Ova odluka da fokus bude na poboljšanju performansi *YOLO* modela proistekla je iz početnih istraživanja koja su ukazala na značaj i kritičnu ulogu preciznog otkrivanja pasusa i tabela u uspjehu predloženog sistema za obradu dokumenata koji se srijeću u bankarstvu.

Postizanje visoke tačnosti u performansama *YOLO* modela je imperativ iz nekoliko razloga. Prvo, obezbjeđuje da sistem može da se nosi sa raznovrsnim i složenim tipovima dokumenata koji se obrađuju u bankarskom sektoru, uključujući različite tabelarne formate. Drugo, minimiziraju se potencijalne greške u sljedećoj fazi sistema (obradi tabela). Treće, doprinosi skalabilnosti rješenja, omogućavajući njegovu primjenu na različita bankarska dokumenta bez potrebe za značajnim prilagođavanjima. Fokusirajući se na optimizaciju *YOLO* modela za bankarska dokumenta, istraživanje naglašava važnost odabira adekvatnih skupova podataka u fazi obuke modela mašinskog učenja za poboljšanje njegovih performansi.

Konačno, ova teza predstavlja napor da se pomjere granice vještačke inteligencije u digitalizaciji dokumenata u bankarskom sektoru sa ciljem postavljanja novih standarda u operativnoj efikasnosti, tačnosti i optimizaciji ljudskih resursa. Kroz sveobuhvatno eksperimentisanje i analizu, ovo istraživanje nastoji da kreira robustan sistem, zasnovan na mašinskom učenju, koji ne samo da ispunjava trenutne zahtjeve bankarskog sektora, već i postavlja osnovu za buduće inovacije u automatizovanoj obradi dokumenata.

## 1.1. Hipoteza

Centralna hipoteza i istraživačka pitanja su predstavljani na sljedeći način:

- **Prva hipoteza:** Prilagođavanje i integracija savremenog modela detekcije objekata *YOLOv8* (poglavlje 2.2.5) sa *PyTesseract OCR* (poglavlje 2.3.2) modelom će dovesti do značajnog poboljšavanja tačnosti i efikasnosti digitalizacije tabelarnih dokumenata u bankarstvu, u poređenju sa tradicionalnim ručnim metodama i samostalnim *OCR* sistemima.
- **Druga hipoteza:** Kvalitet digitalizacije dokumenata se može kvantifikovati preciznošću u otkrivanju rasporeda paragrafa i tabela u dokumentima i prepoznavanju teksta.
- **Treća hipoteza:** Povećanjem kvaliteta digitalizacije dokumenata smanjuju se zahtjevi za ručnim pregledom što dovodi do značajno manjeg vremena obrade dokumenata.

Istraživačka pitanja:

- Kako se preciznost prilagođenog *YOLOv8* modela u otkrivanju rasporeda teksta poredi sa starijim verzijama *YOLO* algoritma za otkrivanje objekata?
- Kako se preciznost i brzina obrade modela *YOLOv8* različite veličine (*nano*, *small*, *medium*, *large*, *extra large*) međusobno porede?
- Kako predloženi sistem utiče na ukupan tok digitalizacije dokumenata u bankama, posebno u pogledu alokacije ljudskih resursa i stope grešaka?

Ovdje je fokus na procjeni operativnih implikacija implementacije predloženog sistema, uključujući njegov potencijal da smanji radno opterećenje ručnog unosa podataka i prateće stope grešaka.

Kroz rigoroznu empirijsku analizu, ovo istraživanje ima za cilj da odgovori na istraživačka pitanja i potvrdi hipotezu da kombinovana primjena *YOLOv8* i *PyTesseract*-a, optimizovana kroz prilagođenu obuku i integraciju, nudi superiorno rješenje za izazove digitalizacije sa kojima se suočava bankarski sektor, posebno u rukovanju tabelarnim dokumentima.

## 1.2. Očekivani rezultati istraživanja

Predloženo istraživanje ima za cilj da značajno unaprijedi digitalizaciju dokumenata kroz inovativnu integraciju *YOLO* (poglavlje 2.2) modela za detekciju objekata sa *PyTesseract OCR* (poglavlje 2.3.2) modelom. Očekuje se nekoliko ključnih rezultata, koji će poduprijeti, kako teorijski napredak, tako i praktične primjene u automatizovanoj obradi dokumenata.

Očekivani rezultati:

- Visoka tačnost otkrivanja paragrafa i tabela u bankarskim dokumentima, čemu će svjedočiti veća preciznost, odziv i *F1* skor (poglavlje 4) za prilagođeni *YOLOv8* (poglavlje 2.2.5) model u poređenju sa osnovnim modelima.
- Drugi *YOLO* model, treniran na *Microsoft*-ovom obimnom skupu podataka, će postići visoku tačnost u prepoznavanju složenih struktura tabela, uključujući redove i kolone, olakšavajući napredniju analizu strukture dokumenta.
- Značajno će se povećati efikasnost i tačnost ekstrakcije teksta iz identifikovanih dijelova dokumenta usljed sinergije između *YOLOv8* modela i *PyTesseract OCR*-a, posebno u okviru složenih tabelarnih podataka. Ova integracija ima za cilj da smanji vrijeme obrade i da detektovanom tekstu dodijeli pripadnost paragrafu odnosno ćeliji tabele, nudeći robusno rješenje za tačnu digitalizaciju tekstualnog sadržaja.
- Metodologija i pronalasci će se moći ekstrapolirati na druge sektore koji se u velikoj mjeri oslanjaju na obradu dokumenata, poput pravnih, zdravstvenih i vladinih agencija, promovišući šire usvajanje rješenja za digitalizaciju dokumenata vođenih vještačkom inteligencijom.
- Ova teza će doprinijeti aktuelnom polju *ML*-a u obradi dokumenata pružanjem empirijskih dokaza o efikasnosti prilagođenih *YOLOv8* modela integrisanih sa *OCR* tehnologijom. Biće izložena procedura optimizacije modela mašinskog učenja za specifične zadatke obrade dokumenata, posebno u rukovanju složenim tabelarnim podacima, koji su manje istraženi u postojećoj literaturi.

## 2. DETEKCIJA OBJEKATA NA SLICI I PRIMJENA OCR TEHNOLOGIJE ZA DETEKCIJU TEKSTA

**Vještačka inteligencija** (eng. *Artificial Intelligence - AI*) obuhvata široku oblast računarske nauke koja ima za cilj stvaranje mašina sposobnih da obavljaju zadatke koji obično zahtijevaju ljudsku inteligenciju. Ovi zadaci uključuju, ali nisu ograničeni na prepoznavanje govora, donošenje odluka, vizuelnu percepciju i prevod jezika. *AI* sisteme karakteriše njihova sposobnost da se prilagode različitim ulazima i uslovima, obavljajući zadatke na način koji oponaša ljudske kognitivne funkcije.

**Mašinsko učenje** (eng. *Machine Learning - ML*) je oblast vještačke inteligencije koja se fokusira na omogućavanje mašinama da uče iz podataka, identifikuju obrasce i donose odluke uz minimalnu ljudsku intervenciju. U mašinskom učenju, algoritmi su dizajnirani da modeluju i uče složene obrasce iz velikih skupova podataka. Ovi algoritmi su kategorisani u tri osnovna tipa na osnovu njihovog pristupa učenju:

- nadgledano učenje (eng. *supervised learning*), gdje model predviđa ishode na osnovu prošlih primjera sa poznatim ishodima;
- nenadgledano učenje (eng. *unsupervised learning*), koje uključuje pronalaženje skrivenih obrazaca ili unutrašnjih struktura u ulaznim podacima i
- učenje sa podrškom (eng. *reinforcement learning*), gdje agent uči da se ponaša u okruženju tako što izvršava radnje i koriguje ih na osnovu ostvarenih rezultata.

**Duboko učenje** (eng. *Deep Learning*), dalja specijalizacija *ML*-a (slika 1), koja se oslanja na slojevitou strukturu algoritama. Najčešće, ali i u kontekstu ovog rada, to podrazumijeva stvaranje „vještačke neuralne mreže” (eng. *Artificial Neural Network - ANN*) koja može da uči i donosi autonomne odluke (više o neuralnim mrežama u narednom potpoglavlju). Duboko učenje je posebno poznato po svojoj efikasnosti u obradi velikih količina podataka, ali i sposobnosti da izvrši ekstrakciju karakteristika bez potrebe za ručnom intervencijom, što nije slučaj kod klasičnog mašinskog učenja.



Slika 1. Odnos *AI*-ja, *ML*-a i dubokog učenja, prikazan Venovim dijagramom.

U kontekstu mašinskog učenja, specifične aplikacije kao što su detekcija objekata i optičko prepoznavanje karaktera unose fundamentalne promjene u različitim industrijskim sektorima.

Integracija *AI* i *ML* tehnologija pravi revoluciju u industriji povećanjem efikasnosti i omogućavanjem automatizacije složenih zadataka. U zdravstvu, *ML* modeli olakšavaju dijagnostiku i robotske operacije; u finansijama, poboljšavaju osiguranje i otkrivanje prevara; u automobilskoj industriji, oni su ključni za razvoj tehnologija autonomne vožnje; a u maloprodaji optimizuju upravljanje zalihama i prilagođavaju usluge preferencama kupaca. Kako ove tehnologije evoluiraju, raste im potencijal da pokrenu značajan napredak, postavljajući nova mjerila za ono što mašine mogu da postignu.

## 2.1. Neuralne mreže, osnova dubokog učenja

Neuralne mreže su vrsta algoritma za mašinsko učenje i čine kičmu većine algoritama dubokog učenja. Inspirisane su biološkom strukturom i funkcijom ljudskog mozga, posebno njegovom sposobnošću da prepozna obrasce i obrađuje podatke na paralelan, međusobno povezan način preko neurona.

Strukturu neuralnih mreža čine:

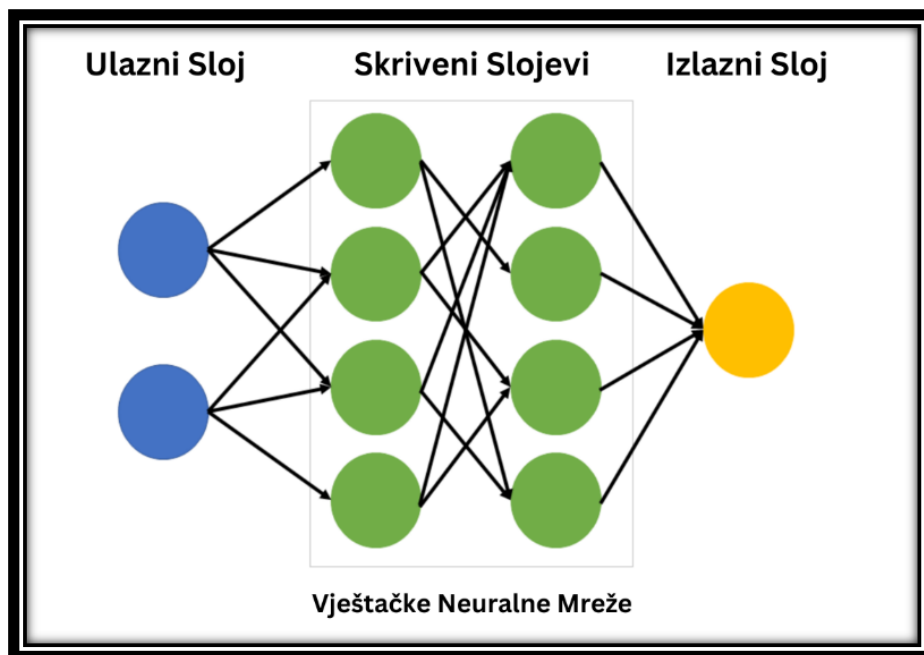
- **Neuroni:** Osnovne jedinice neuralne mreže, neuroni primaju ulaze, obrađuju ih i generišu izlaze. U vještačkim neuralnim mrežama, svaki neuron vrši zbir svojih ulaza, modifikovanih težinskim koeficijentima, dodaje koeficijente pristrasnosti, te zatim ovu sumu propušta kroz aktivacionu funkciju kako bi odredio izlaz.
- **Slojevi:** Neuralne mreže su strukturisane u slojevima: ulazni sloj, jedan ili više skrivenih slojeva i izlazni sloj (slika 2). Svaki sloj se sastoji od neurona koji su povezani sa narednim slojevima. Složenost i sposobnosti neuralne mreže u velikoj mjeri su određeni brojem slojeva i brojem neurona u svakom sloju.
- **Težinski koeficijenti** (eng. *weights*) **i koeficijenti pristrasnosti** (eng. *biases*): Ovo su parametri unutar neurona koji se optimizuju tokom procesa treniranja neuralne mreže. Težinski koeficijenti kontrolišu jačinu ulaznih signala, dok koeficijenti pristrasnosti predstavljaju dodatni parametar koji prilagođava izlaz zajedno sa težinskom sumom.
- **Aktivacione funkcije:** Aktivacione funkcije su ključne jer uvode nelinearne osobine u mrežu - ova nelinearnost omogućava neuralnim mrežama da obrađuju sve složenije funkcije. Uobičajene funkcije aktivacije uključuju *ReLU* (eng. *Rectified Linear Unit*), *Sigmoid* i *Tanh*.

$$ReLU(x) = \max(0, x) \quad (1)$$

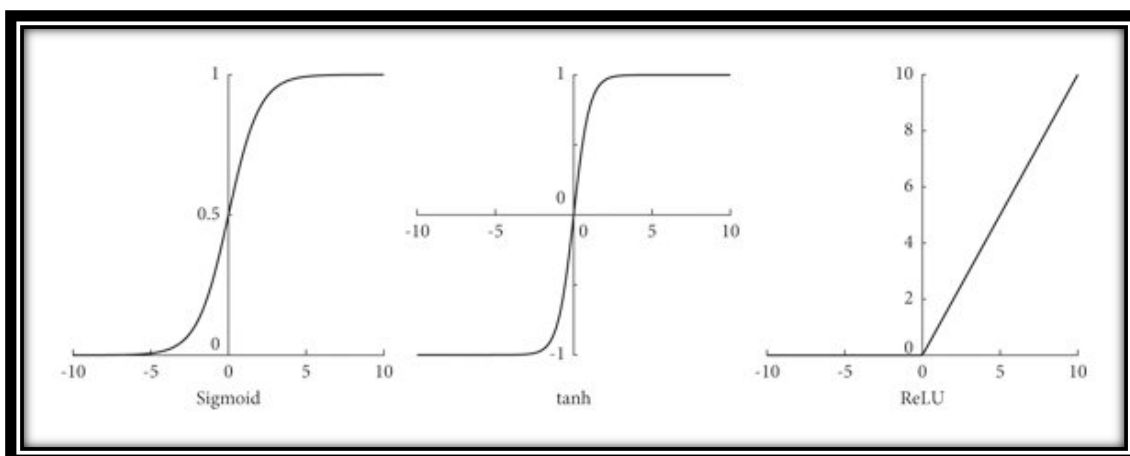
$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \quad (2)$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2 \cdot Sigmoid(2x) - 1 \quad (3)$$

*ReLU* (1) funkcija vraća ulaz ako je pozitivan, u suprotnom vraća nulu. *Sigmoid* (2) funkcija je glatka kriva koja mapira bilo koji ulaz u opseg vrijednosti između 0 i 1. Funkcija *tanh* (3) je slična *sigmoid* funkciji, ali mapira ulaze u opseg od -1 do 1. Njihove grafičke reprezentacije date su slikom 3.



Slika 2. Grafički prikaz strukture jednostavnih neuralnih mreža



Slika 3. Grafički prikaz popularnih aktivacionih funkcija

Proces učenja ili treniranja neuralne mreže čine koraci:

1. **Propagacija unaprijed:** Podaci se prenose kroz mrežu, od ulaznog sloja preko skrivenih slojeva do izlaznog sloja, gdje se izlazi generišu na osnovu trenutnih vrijednosti težinskih koeficijenata i koeficijenata pristrasnosti.
2. **Proračun gubitaka** (eng. *loss*): Performanse mreže se procjenjuju korišćenjem funkcije gubitaka (eng. *loss function*), koja mjeri razliku između stvarnog i

predviđenog trenutnog izlaza neuralne mreže. Cilj treniranja je da se minimizira ovaj gubitak.

3. **Propagacija unazad:** Ovo je ključna faza u kojoj mreža uči iz grešaka. Propagacija unazad uključuje izračunavanje gradijenta funkcije gubitaka u odnosu na svaki težinski koeficijent, po lančanom pravilu, idući unazad od izlaznog do ulaznog sloja.
4. **Ažuriranje težinskih koeficijenata:** Nakon što se tokom propagacije unazad izračunaju gradijenti funkcije gubitaka u odnosu na težinske koeficijente, ti gradijenti se koriste za ažuriranje težinskih koeficijenata i koeficijenata pristrasnosti. Ovi podaci o gradijentima se zatim koriste u optimizacionim algoritmima, kao što je gradijentni spust, koji prilagođavaju vrijednosti težinskih koeficijenata i pristrasnosti s ciljem minimizacije funkcije gubitaka. Ovaj ciklus se ponavlja kroz brojne iteracije (često i hiljadama puta) kako bi mreža postepeno poboljšala svoje performanse. Epoha (eng. *epoch*) je jedan prolazak algoritma, ili iteracija, kroz cijeli skup podataka za trening. Tokom jedne epohe model uči i to prilagođavajući svoje težinske koeficijente kako bi minimizirao greške u svojim predviđanjima. Tipično je potrebno više epoha da bi model efikasno učio i dobro generalizovao iz podataka, pri čemu svaka uzastopna epoha potencijalno poboljšava tačnost modela.

Za proces treniranja je od posebne važnosti odabir veličine **grupe podataka** (eng. *batch*). U mašinskom učenju, grupa podataka se odnosi na podskup skupa podataka koji se koristi za trening algoritma. Cjelokupan skup podataka podijeljen je u nekoliko grupa podataka, pri čemu se svaka grupa obrađuje zasebno tokom faze treniranja. Ovaj pristup, poznat kao treniranje na grupama podataka (eng. *batch training*), pomaže u optimizaciji procesa učenja tako što postepeno ažurira težinske koeficijente modela nakon obrade svake grupe podataka, umjesto da se čeka da se obradi cijeli skup podataka. Batch training je posebno koristan za velike skupove podataka koji ne mogu u potpunosti da stanu u memoriju, a takođe omogućava stabilniju i efikasniju optimizaciju gradijentnog spusta, usrednjavanjem gradijenta u grupi podataka, čime se smanjuje varijansa ažuriranja. Odabir veličine grupe podataka (eng. *batch size*) zavisi od nekoliko faktora, uključujući veličinu skupa podataka i raspoložive računarske resurse. Manje grupe podataka

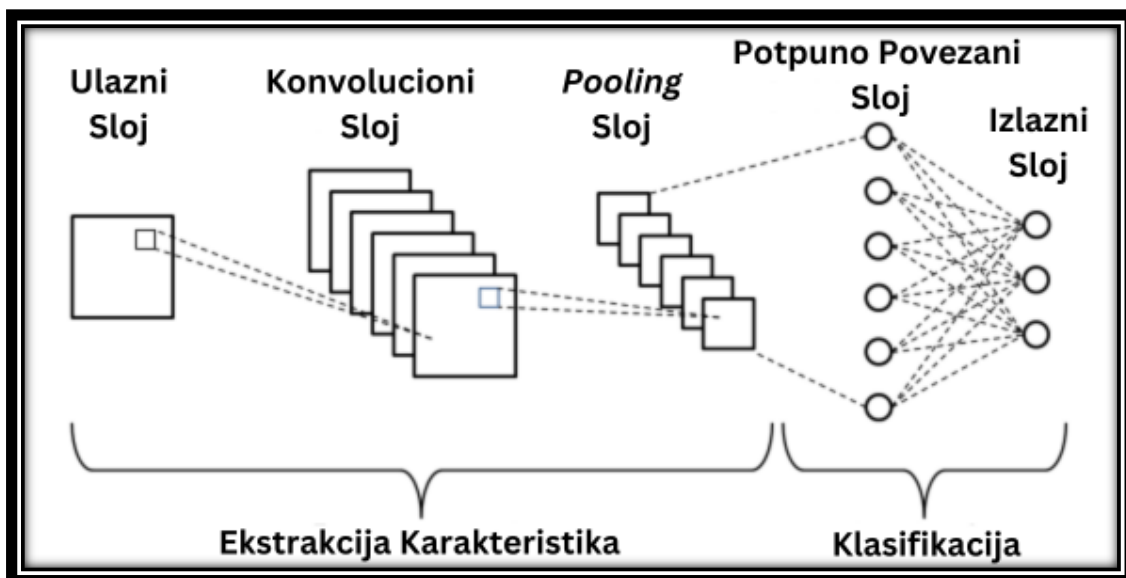
omogućavaju brže ažuriranje težinskih koeficijenata i često ubrzavaju treniranje, ali mogu imati veću varijansu u procjeni gradijenata, što može dovesti do nestabilnog učenja. Veće grupe podataka smanjuju tu varijansu, jer usrednjavaju gradijent na većem broju uzoraka, ali zahtijevaju više memorije i mogu dovesti do sporijih iteracija (epoha). Najčešće se eksperimentiše sa veličinom grupe, često počinjući od manjih grupa (od 16 uzorka npr.) i povećavajući u skladu sa dostupnim resursima.

Najpopularnije vrste neuralnih mreža su:

- **Neuralne mreže sa propagacijom unaprijed** (eng. *Feedforward Neural Networks*): Najjednostavniji tip vještačke neuralne mreže, gdje podaci teku u jednom pravcu, od ulaznog sloja preko jednog ili više skrivenih slojeva do izlaznog sloja. Svaki sloj se sastoji od neurona koji su u potpunosti povezani sa neuronima u sledećem sloju, bez ciklusa ili povratnih petlji, zbog čega se često nazivaju i **potpuno povezanim neuralnim mrežama**. U svakom skrivenom sloju, neuroni primjenjuju matematičke operacije, kombinujući težinske koeficijente i koeficijente pristrasnosti sa aktivacionom funkcijom, kako bi transformisali ulaz. Cilj mreže je da prilagodi ove koeficijente tokom treninga (kroz propagaciju unazad) kako bi se minimizirala greška između predviđenog izlaza i stvarnog cilja. Neuralne mreže sa propagacijom unaprijed se obično koriste u zadacima kao što su klasifikacija slika, regresija i prepoznavanje obrazaca i čine osnovu za složenije arhitekture neuralnih mreža.
- **Konvolucione neuralne mreže** (eng. *Convolutional Neural Networks - CNN*): Klasa neuralnih mreža, veoma efikasnih za analizu slika i video zapisa. *CNN*-ovi su posebno dizajnirani za obradu piksela i uveliko se koriste u zadacima prepoznavanja i obrade slika. Odlikuju se višeslojnom arhitekturom i sposobnošću da uhvate prostorne hijerarhije u podacima zahvaljujući konvolucionim slojevima. Arhitektura *CNN*-a se obično sastoji od nekoliko slojeva uključujući konvolucione slojeve, slojeve sažimanja i potpuno povezane slojeve (slika 4). Konvolucionni slojevi primjenjuju skup podesnih filtera na ulaznu sliku da bi kreirali mape karakteristika (eng. *feature mape*), koje obuhvataju važne aspekte, kao što su ivice i oblici. Slojevi sažimanja (eng. *pooling layers*) smanjuju dimenzionalnost svake mape karakteristika, ali zadržavaju najvažnije informacije.

Potpuno povezani slojevi zatim koriste ove karakteristike da klasifikuju ulaznu sliku u različite kategorije (klase) na osnovu skupa podataka za trening. CNN-ovi automatski otkrivaju važne karakteristike, bez ikakvog ljudskog nadzora (van podešavanja parametara obuke, poglavlje 4.2), što ih čini posebno pogodnim za zadatke kompjuterske vizije (poglavlje 2.2.1), kao što su prepoznavanje lica, detekcija objekata i segmentacija slike. Njihova sposobnost da razviju internu reprezentaciju dvodimenzionalne slike koje zadržavaju ključne informacije o slici (poput ivica, oblika, tekstura), dok zanemaruju manje važne detalje, omogućava im da efikasno i precizno rukovode složenim zadacima transformacije slike i klasifikacije.

- **Rekurentne neuralne mreže** (eng. *Recurrent Neural Networks - RNN*): Imaju povratne petlje (eng. *loops*) u sebi, koje omogućavaju da se informacije iz prethodnih vremenskih koraka prenose naprijed kroz mrežu. Na taj način, informacije o prethodnim stanjima „opstaju” unutar mreže i utiču na donošenje odluka u trenutnim i budućim vremenskim koracima, čineći ih idealnim za rukovanje sekvencijalnim podacima poput govora i teksta. Varijante kao što je *LSTM* su popularne u rješavanju problema gdje kontekst i redosljed događaja igraju ključnu ulogu, kao što je to slučaj u algoritmima obrade prirodnog jezika.



Slika 4. Arhitektura konvolucione neuralne mreže

Neuralne mreže pokreću mnoge moderne *AI* aplikacije, čineći mogućim napredak u različitim oblastima kao što su autonomna vozila, zdravstvo (za dijagnostičku tačnost), finansijske usluge (za prediktivnu analitiku) i još mnogo drugih. Njihova sposobnost da uče iz ogromne količine podataka i identifikuju obrasce koji ljudskim analitičarima nisu odmah očigledni jeste značajna prekretnica na putu vještačke inteligencije.

Proširujući mogućnosti mašina i omogućavajući im da obavljaju složene zadatke slične ljudima, neuralne mreže nastavljaju da budu na čelu *AI* revolucije, pomjerajući granice onoga što mašine mogu da postignu.

## **2.2. *YOLO (You Only Look Once)* algoritam**

### **2.2.1. Kompjuterska vizija**

**Kompjuterska vizija** je interdisciplinarno naučno polje koje se fokusira na omogućavanje računarima da tumače i razumiju vizuelne informacije iz svijeta, na isti način na koji to ljudi rade. Koristeći se digitalnim slikama sa kamera, video zapisima i modelima dubokog učenja, kompjuterska vizija nastoji da automatizuje zadatke koji zahtijevaju ljudsko vizuelno razumijevanje. Mogućnosti ove tehnologije se iskazuju kroz primjene u različitim aplikacijama, od jednostavnih zadataka poput prepoznavanja ručno pisanih cifara do složenih operacija poput autonomne vožnje, prepoznavanja lica za bezbjednosne sisteme i analiziranja medicinskih snimaka u dijagnostičke svrhe.

U osnovi, kompjuterska vizija upošljava algoritme i modele koji mogu prepoznati obrasce i donositi odluke na osnovu vizuelnih ulaza. Ovo uključuje brojne manje zadatke, kao što su klasifikacija slike, detekcija objekata, segmentacija slike i retuširanje slika. Naučno polje od interesa je vidjelo značajna unapređenja, zahvaljujući brzom razvoju mašinskog učenja i neuralnih mreža, koje su značajno poboljšale tačnost i pouzdanost računarskog vizuelnog prepoznavanja. Ova poboljšanja su gurnula računarsku viziju u prvi plan tehnološke inovacije, olakšavajući transformacije u zdravstvenom, automobilskom, zabavnom i bezbjednosnom sektoru. Tekući razvoj tehnologija računarske vizije i dalje pomjera granice onoga što se može automatizovati, obećavajući još veću integraciju u svakodnevna tehnološka rješenja.

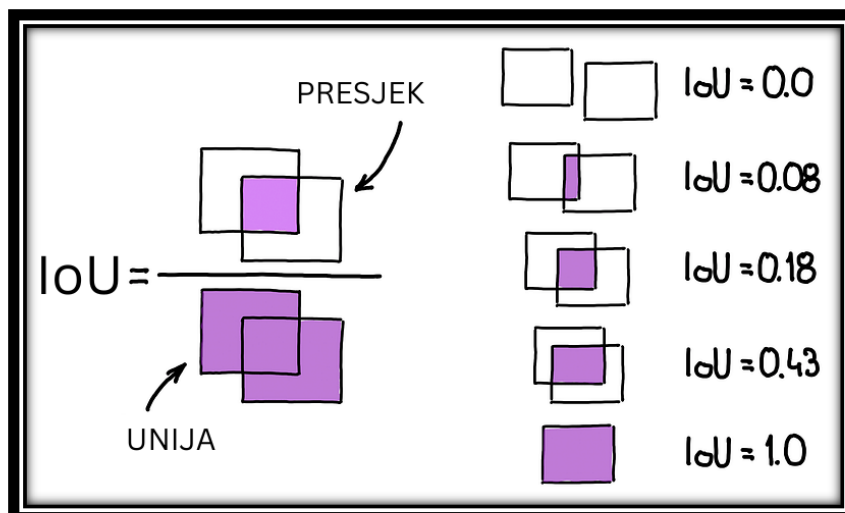
## 2.2.2. Uvod u detekciju objekata

**Detekcija objekta** stoji kao fundamentalni zadatak u polju računarske vizije, sa široko rasprostranjenim aplikacijama koje uključuju video nadzor, autonomnu vožnju i sisteme za pretraživanje slika. Ovaj proces uključuje identifikaciju i lokalizaciju objekata unutar digitalnih slika i dodjeljivanje svakog otkrivenog objekta određenoj kategoriji (klasi). Tradicionalne metode detekcije objekta, poput onih zasnovanih na kliznim prozorima i predlozima regiona [18], u velikoj mjeri su zamijenjene sofisticiranijim pristupima dubokog učenja koji nude značajna poboljšanja u tačnosti i brzini.

**Klizni prozori** (eng. *sliding windows*) su metoda detekcije objekata koja uključuje pomijeranje prozora fiksne veličine preko slike, na svaku moguću lokaciju. Na svakoj poziciji prozora koristi se klasifikator za utvrđivanje da li je predmet interesovanja prisutan u tom prozoru. Ova tehnika sistematski provjerava svaki dio slike, što može biti računski zahtjevno i sporo, posebno za velike slike sa mnogim potencijalnim veličinama i oblicima predmeta.

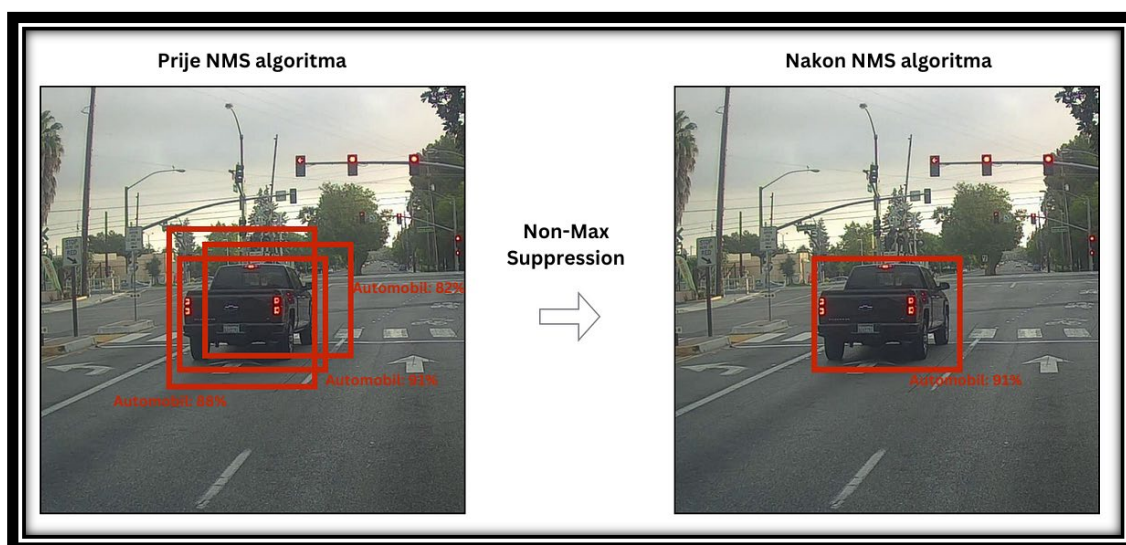
**Predlog regiona** (eng. *region proposal*) je metoda detekcije objekata koja predstavlja napredak u odnosu na metodu kliznih prozora tako što prvo identifikuje manji broj potencijalnih regiona koji će vjerovatno sadržati tražene objekte. To se često postiže algoritmima poput selektivne pretrage, koji segmentiraju sliku u regione na osnovu teksture, boja i drugih karakteristika, a zatim ih grupišu u veće regione koji mogu potencijalno da sadrže predmete. Ove predložene regije zatim obrađuju složeniji klasifikatori (često *CNN*, poglavlje 2.1) da bi utvrdili koji su predmeti (ako uopšte postoje) prisutni i da preciziraju njihove lokacije.

Za zadatak detekcije objekata, od suštinskog značaja je *NMS* algoritam. U njegovoj srži leži ***Intersection over Union (IoU)***, metrika zastupljena u modelima detekcije objekata. Najčešće se koristi za procjenu tačnosti predviđenog graničnog okvira mjerenjem njegovog preklapanja sa stvarnim graničnim okvirom (iz skupa podataka). Izračunava se tako što se površina preklapanja (presjeka) između predviđenih i tačnih graničnih okvira podijeli sa površinom njihove unije (slika 5), što rezultira vrijednošću između 0 i 1. Veći *IoU* ukazuje na veću tačnost predviđanja. Ova metrika je ključna za procjenu performansi detekcije i glavna primjena joj je u evaluaciji modela i *NMS* algoritmima.



Slika 5. Vizuelni prikaz računanja  $IoU$  metrike

**Non-Max Suppression (NMS)** je algoritam dominantno zastupljen u modelima za detekciju objekata. Služi za pojednostavljenje izlaza tako što obezbeđuje da svaki otkriveni objekat bude predstavljen samo jednim graničnim okvirom. Proces uključuje sortiranje svih pronađenih graničnih okvira prema pouzdanosti, zadržavanje onog sa najvišim stepenom pouzdanosti (slika 6) i uklanjanje svih drugih koji se značajno preklapaju sa njim na osnovu  $IoU$  metrike. Ovaj proces se ponavlja za svaki granični okvir, dajući na izlazu čist skup detekcija sa minimalnim preklapanjem, poboljšavajući tačnost i čitljivost rezultata.



Slika 6. Prikaz funkcionisanja  $NMS$  algoritma

### 2.2.3. Uvod u *YOLO* algoritam

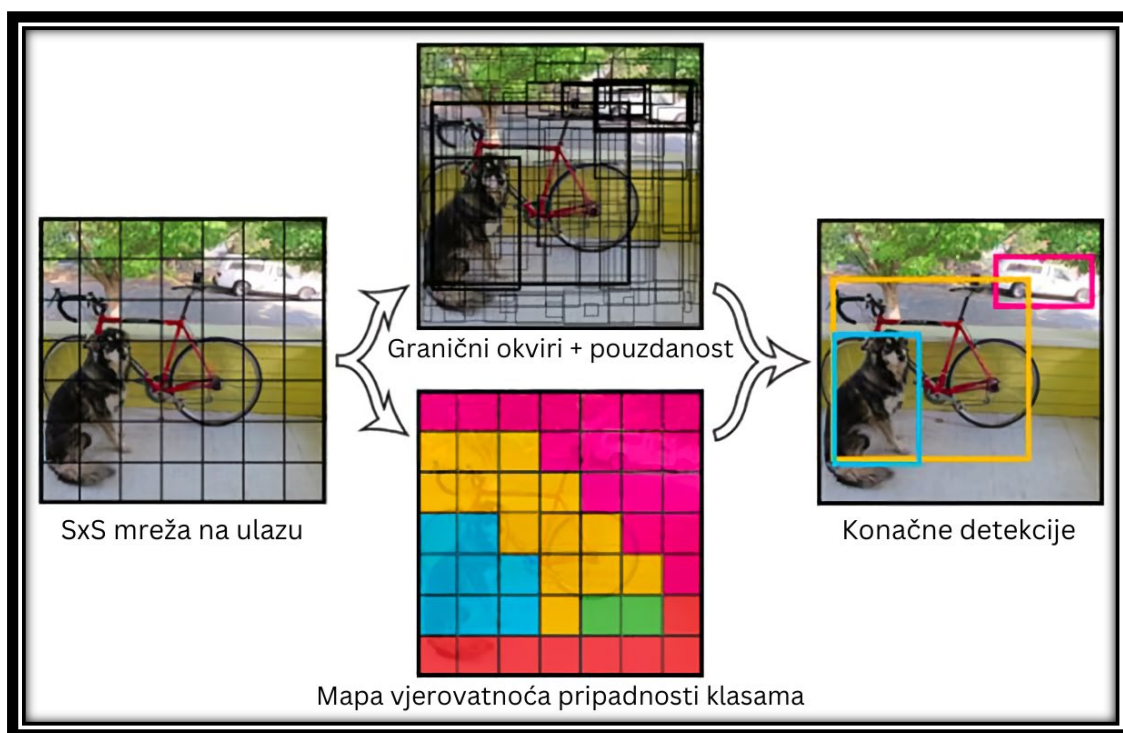
*YOLO* [19] je popularan algoritam mašinskog učenja za otkrivanje objekata. Poznat je po svojoj brzini i tačnosti u otkrivanju objekata na slikama u realnom vremenu. *YOLO* svodi detekciju objekta na jedinstveni regresioni problem, dajući koordinate graničnih okvira i vjerovatnoće klasa, polazeći od piksela slike. Ovaj pristup se značajno razlikuje od ostalih sistema za otkrivanje objekata, gdje sistem predlaže regione pa zatim pokreće klasifikator na tim regionima.

Prednosti *YOLO* algoritma u otkrivanju objekata su mnogobrojne, posebno u poređenju sa tradicionalnim metodama detekcije objekata. Prije pojave *YOLO*-a, većina sistema za detekciju objekata, kao što je *R-CNN* [18] (Regije sa *CNN* karakteristikama, poglavlje 2.1) i njegove varijante (*Fast R-CNN*, *Faster R-CNN*), funkcionisala je u dvije različite faze:

1. **Predlog regiona** (poglavlje 2.2.2): Često se radi korišćenjem selektivne pretrage ili drugih heurističkih tehnika u ranijim metodama, ili upotrebom zasebne mreže za predlaganje regiona (kao u slučaju *Faster R-CNN*-a).
2. **Klasifikacija**: Nakon što su predloženi potencijalni regioni, klasifikator (često konvoluciona neuralna mreža) se pokreće na svakom regionu posebno kako bi se utvrdilo koji objekat, ako ga ima, se nalazi u datom regionu. Ovaj korak takođe može da poboljša koordinate graničnih okvira kako bi preciznije opisivao otkriveni objekat.

Ovaj pristup je, iako efikasan, računarski veoma intenzivan pošto zahtijeva višestruko pokretanje kompleksnog klasifikatora za jednu sliku – po jednom za svaki predloženi region. To će izvjesno dovesti do sporije brzine obrade, što nije idealno za aplikacije koje zahtijevaju analizu u realnom vremenu. *YOLO* pojednostavljuje navedeni proces tretirajući detekciju objekata kao problem sa jednom regresijom koja od ulazne slike generiše koordinate graničnih okvira i vjerovatnoće pripadnosti klasama (slika 7). Originalna *YOLO* arhitektura je zasnovana na „dubokoj” konvolucionoj neuralnoj mreži, sastavljenoj od 24 konvoluciona sloja praćena sa dva potpuno povezana sloja [19]. Ovi slojevi istovremeno vrše ekstrakciju karakteristika i predviđanje. U ranim i srednjim slojevima, *YOLO* koristi konvolucione slojeve da izvuče važne karakteristike iz slike. Ovi

slojevi otkrivaju prostorne obrasce i karakteristike objekata, ali još uvijek ne predviđaju direktno klase. Poslednji konvolucioni sloj je odgovoran za predviđanja klasa zajedno sa koordinatama graničnih okvira i stepenom pouzdanosti. Tip aktivacione funkcije (poglavlje 2.1) koji propušta nemodifikovane izlaze neurona mreže – linearna aktivaciona funkcija, koristi se za generisanje koordinata graničnih okvira. Ona omogućava modelu da proizvodi kontinualne vrijednosti bez ograničenja, što je neophodno za preciznu lokalizaciju objekata unutar slike. S druge strane, *sigmoid* aktivaciona funkcija se koristi za stepene pouzdanosti klase i graničnih okvira, ograničavajući ove izlaze na opseg između 0 i 1. To osigurava da predviđanja modela odražavaju vjerovatnoću da granični okvir pripada određenoj klasi i sigurnost modela u predviđeni granični okvir.



Slika 7. Princip funkcionisanja *YOLO* algoritma

Princip funkcionisanja *YOLO* algoritma (slika 7):

1. **Podjela u mrežu:** Prvi korak u *YOLO* algoritmu je podjela ulazne slike u mrežni sistem. Konkretno, slika je podeljena na  $S \times S$  mrežu ćelija (dimenzija  $13 \times 13$  na primjer). Svaka ćelija mreže je odgovorna za otkrivanje objekata čiji centri spadaju u nju. Ovakav pristup omogućava *YOLO*-u da obradi cijelu sliku u jednom prolasku kroz mrežu, uzimajući u obzir ukupni kontekst uz dodjeljivanje

specifičnih zadataka detekcije svakoj ćeliji mreže. Dijeljenjem slike, *YOLO* smanjuje računsku složenost problema i omogućava bržu, paralelnu obradu. Svaka ćelija mreže radi nezavisno što modelu pojednostavljuje detekciju.

2. **Predviđanje graničnih okvira:** Kada se slika podijeli u mrežu, naredni korak je predviđanje graničnih okvira. Svaka ćelija mreže predviđa unaprijed definisani broj graničnih okvira. Poslednji sloj *YOLO*-a je konfigurisan da na izlazu daje tačan broj graničnih okvira po ćeliji mreže, koji zavisi od konfiguracije i verzije *YOLO* modela.

$$F = S \cdot S \cdot (B \cdot 5 + C) \quad (4)$$

Ovo se postiže filterima u poslednjem konvolucionom sloju, označenim sa  $F$  u (4), gdje svaki filter odgovara tačno jednoj predikciji graničnog okvira. Od ukupnog broja filtera poslednjeg sloja,  $SxSxBx5$  su odgovorni za granične okvire, od kojih se  $SxS$  filtera odnosi na broj ćelija mreže iz prvog koraka,  $B$  na definisani broj graničnih okvira po ćeliji i  $5$  na koordinate graničnih okvira. Predviđanjem većeg broja graničnih okvira po ćeliji mreže, *YOLO* povećava vjerovatnoću otkrivanja objekata različitih veličina i razmjera. Granični okvir je definisan sa pet komponenti:  $x$  i  $y$  koordinate – definišu poziciju centra graničnog okvira, širina i visina – predstavljaju veličinu graničnog okvira u odnosu na cijelu sliku i stepen pouzdanosti, izražen u procentima (eng. *confidence score*) – računa se kao kombinacija vjerovatnoće da on sadrži određeni objekat i vjerovatnoće da pripada određenoj klasi. Prilikom treniranja *YOLO* modela, stepenu pouzdanosti se dodaje i *IoU* skor, računat za detektovani granični okvir, i stvarni, anotirani granični okvir iz skupa podataka. Kada se model koristi za detekciju, na novim slikama, tada se *IoU* ne koristi u ovom koraku, već samo u četvrtom.

3. **Predviđanje klase:** Pored koordinata graničnih okvira i stepena pouzdanosti, svaka ćelija mreže predviđa vjerovatnoće pripadnosti klasama, bez obzira na detektovane granične okvire koji pripadaju toj ćeliji. One ukazuju na vjerovatnoću da otkriveni objekat pripada određenoj klasi iz unaprijed definisanog skupa (u konkretnom slučaju: tabela i paragraf za model koji otkriva generalnu strukturu dokumenta; tabela, red i kolona za model koji otkriva strukturu tabele). Detekcija klase se postiže filterima u poslednjem konvolucionom sloju (4), čiji je broj

proporcionalan definisanom broju klasa modela. Od ukupnog broja filtera posljednjeg sloja *YOLO*-ve konvolucione neuralne mreže, njih  $S_x S_x C$  je zaslužno za predviđanje klase. Predviđanja klase se prave nezavisno od graničnih okvira, ali se kasnije kombinuju, dajući sveobuhvatan rezultat detekcije.

4. **Izlaz:** Konačan izlaz *YOLO* algoritma se generiše kombinovanjem predviđanja graničnog okvira sa vjerovatnoćama klase. Dakle, rezultat predstavlja vjerovatnoću da granični okvir sadrži konkretan objekat klase i koliko dobro predviđeni okvir odgovara tom objektu. Na dobijenim rezultatima detekcije, *YOLO* primjenjuje prag za filtriranje graničnih okvira sa niskim ocjenama pouzdanosti. Pored toga, koristi i *NMS* algoritam kako bi eliminisao suvišne detekcije. Preostali granični okviri čine konačne detekcije, od kojih je svaka povezana sa oznakom klase i stepenom pouzdanosti, tačno ukazujući na lokaciju i identitet objekata unutar slike.

U svrhu ilustracije, razmatra se scenario gdje je slika podijeljena na mrežu  $7 \times 7$ , što je u (4) označeno sa  $S=7$ , a samim tim i  $S \cdot S=49$  ćelija. Pretpostavi se da svaka ćelija mreže predviđa  $B=2$  granična okvira i da je model obučen da detektuje  $C=20$  različitih klasa. Prvo, za svaku ćeliju mreže generiše se  $B \cdot 5$ , odnosno  $2 \cdot 5=10$  graničnih okvira, dok  $C=20$  predstavlja vjerovatnoće pripadanja svakoj od 20 klasa. Sumiranjem ovih vrijednosti dobija se potpuna slika o ćeliji mreže, data sa  $B \cdot 5 + C = 10 + 20 = 30$  parametara. Konačno, veličina finalnog višedimenzionog vektora ili tenzora (eng. *tensor*)  $F$  se izračunava množenjem dobijene vrijednosti parametara po ćeliji sa brojem ćelija mreže, čime se dobija  $F = S \cdot S \cdot (B \cdot 5 + C) = 49 \cdot 30 = 1470$ . Prema tome, u navedenoj ilustraciji, izlazni tenzor *YOLO* modela će sadržati 1470 elemenata, organizovanih u trodimenzioni tenzor po principu  $(S, S, B \cdot 5 + C) = (7, 7, 30)$ , obuhvatajući sve granične okvire i informacije o klasama koje su neophodne za detekciju objekta u cijeloj slici.

Tokom treninga, *YOLO* model prilagođava težinske koeficijente (poglavlje 2.1) kako bi minimizirao greške u lokalizaciji i klasifikaciji objekata. Jedan od ključnih izazova je rukovanje neravnotežom između ćelija mreže koje sadrže objekte i onih koje ne sadrže. Da bi se ovo premostilo, *YOLO* podešava težinske koeficijente u funkciji gubitka (poglavlje 2.1), kako bi se usmjerio na relevantne ćelije. Za ćelije mreže u kojima je prisutan objekat, *YOLO* dodjeljuje veću težinu grešci u predviđanju koordinata

graničnog okvira, stepena pouzdanosti i vjerovatnoće klase. Ovo osigurava da je model teže „kažnjen” kada pravi greške za ćelije koje sadrže objekte. Kod ćelija mreže koje ne sadrže nikakve objekte, gubitak stepena pouzdanosti i dalje postoji, ali sa mnogo manjom težinom. To sprečava da veliki broj praznih ćelija mreže, u kojima je predviđanje odsustva objekta mnogo lakše nego detektovanje objekata, ometa model.

Jedna od ključnih prednosti *YOLO*-a je to što, za razliku od metoda koji obrađuju djelove slike ili pojedinačne regione izolovano, *YOLO* sagledava cijelu sliku prilikom predviđanja. Ovaj **globalni kontekst** pomaže modelu da „razumije” opšti kontekst slike, smanjujući šanse za pogrešnim klasifikacijama i lažno pozitivnim rezultatima. Druga, glavna, prednost *YOLO* algoritma njegova **brzina**. Za razliku od metoda zasnovanih na predlogu regiona, kao što je *Faster R-CNN*, koje zahtijevaju više faza za otkrivanje objekata, *YOLO* primjenjuje jednu neuralnu mrežu na cijelu sliku. Ovo čini *YOLO* veoma pogodnim za aplikacije koje zahtijevaju obradu u realnom vremenu, kao što su analiza video sadržaja, autonomna vožnja i nadzor.

Međutim, *YOLO*-va brzina dolazi sa nekim kompromisima. Rane verzije, posebno *YOLOv1*, su imale problema sa otkrivanjem malih objekata i imale su nižu tačnost lokalizacije (pozicioniranja) u poređenju sa složenijim modelima kao što je *Faster R-CNN*. Ovo je prvenstveno zbog *YOLO*-ove fiksne podjele u mrežu, koja je bila problematična za objekte koji obuhvataju više ćelija mreže. Kasnije verzije napravile su značajna poboljšanja u preciznosti uvođenjem predviđanja u više razmjera (različiti slojevi mreže fokusiraju na otkrivanje objekata u različitim razmjerama - manjim, srednjim i većim) i boljih mehanizama sidrenih okvira, koji pomažu da se preciznije detektuju objekti različitih veličina.

#### 2.2.4. *YOLO* verzije:

Tokom godina, *YOLO* je prošao kroz mnoge iteracije, poboljšavajući svoju arhitekturu, brzinu i tačnost. Počevši od *YOLOv1*, preko *YOLOv4*, poboljšanja su uključivala bolje mreže za ekstrakciju karakteristika, upotrebu normalizacije grupe podataka (eng. *batch normalization*, poglavlje 2.1), sidrenih okvira i normalizacije između grupa podataka (eng. *cross-batch normalization*). Normalizacija grupe podataka i normalizacija između grupa podataka su tehnike koje se koriste za poboljšanje

stabilnosti treninga i efikasnosti neuralnih mreža normalizacijom ulaza slojeva (poglavlje 2.1):

- **Normalizacija grupe podataka** prilagođava izlaze sloja neuralne mreže tako da imaju nultu srednju vrijednost i jediničnu varijansu u svakoj grupi podataka. Ova normalizacija pomaže u stabilizaciji procesa učenja tako što obezbjeđuje dosljednu distribuciju aktivacija (vrijednosti na izlazima slojeva neuralne mreže) tokom treninga. Omogućava mreži da koristi veće stope učenja (poglavlje 4.2) i pojednostavljuje početno podešavanje težinskih koeficijenata (poglavlje 2.1). Pored toga, može smanjiti potrebu za drugim metodama regularizacije.
- **Normalizacija između grupa podataka**, funkcioniše slično normalizaciji grupe podataka, ali umjesto da normalizacija vrši sa statistikom iz samo jedne grupe podataka, koristi se statistika više grupa podataka. Ovo je posebno korisno kada su one male i podaci unutar jedne grupe podataka nisu dovoljni da daju stabilnu procjenu. Ovaj metod poboljšava stabilnost i tačnost normalizacije, što je korisno za obuku sa malim grupama podataka.

Objе tehnike imaju za cilj da učine treniranje neuralnih mreža bržim i pouzdanijim tako što će obezbijediti da skala ulaza ostane kontrolisana u cijeloj mreži.

Svaka naredna verzija *YOLO*-a dizajnirana je da balansira kompromise između brzine detekcije i tačnosti, kao i da poboljša robusnost modela u snalaženju sa objektima različitih veličina i izgleda.

### 2.2.5. *YOLOv8*

*YOLOv8* je poslednja iteracija u *YOLO* porodici algoritama mašinskog učenja. Dolazi nakon poboljšanja viđenih u prethodnim verzijama poput *YOLOv4* i *YOLOv5* (koji, uprkos njegovom imenu, nije direktni nasljednik *YOLOv4*, već varijanta, razvijena od strane drugog tima). *YOLOv8* uključuje najnovije napretke u vještačkoj inteligenciji i dubokom učenju s ciljem da dodatno gurne granice brzine i tačnosti.

Na slici 8 je dato poređenje performansi različitih verzija *YOLO* modela za detekciju objekata, konkretno *YOLOv5*, *YOLOv6*, *YOLOv7* i *YOLOv8*, koristeći tri osnovne metrike: veličinu modela, brzinu detekcije i preciznost. **Veličina modela** odnosi

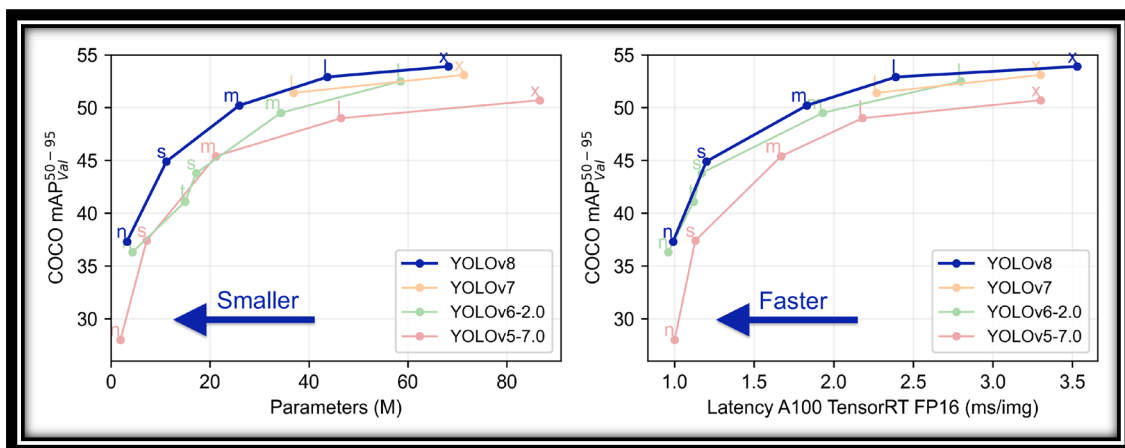
se na broj parametara, koji predstavlja ukupni zbir težinskih koeficijenata i koeficijenata pristrasnosti (poglavlje 2.1) unutar svih slojeva neuralne mreže. **Kašnjenje** (eng. *latency*) mjeri koliko vremena je modelu potrebno da obradi jednu sliku. Što je manja vrijednost, to je model brži. **Srednja prosječna preciznost** (eng. *mean average precision - mAP*) je metrika od suštinskog značaja za procjenu performansi *YOLO* modela jer pruža sveobuhvatnu mjeru tačnosti detekcije i klasifikacije objekata.

$$mAP = \frac{1}{N} \cdot \sum_{i=1}^N AP_i \quad (5)$$

$mAP$  (5) računa prosječnu preciznost ( $AP$ ) za svaku klasu objekata ( $N$ ), uzimajući u obzir kompromis između preciznosti i odziva, i vrši usrednjavanje.  $AP$  se izračunava uzimanjem preciznosti na različitim pragovima odziva, odnosno minimalnim odzivima graničnih okvira koje model uzima u obzir (npr. svaki korak od 0.1) i usrednjavanjem vrijednosti preciznosti za granične okvire koji prođu tu selekciju.

Ono što  $mAP$  „donosi na sto“, posebno u kontekstu detekcije objekata, jeste njegova sposobnost da uzme u obzir i tačnost lokalizacije (koliko dobro predviđeni granični okviri odgovaraju objektima) i tačnost klasifikacije (da li su otkriveni objekti klasifikovani ispravno). Ovo čini  $mAP$  nijansiranim i cjelovitijom metrikom od jednostavnog korišćenja tačnosti, nudeći dublji uvid u to koliko dobro model otkriva sve klase objekata za različite pragove detekcije.

Lijevi grafikon na slici 8 prikazuje performanse modela u odnosu na njihovu veličinu.  $X$ -osa predstavlja broj parametara modela, u milionima ( $M$ ).  $Y$ -osa pokazuje preciznost modela, mjerenu *COCO mAP 50-90 Val* metrikom. *COCO* [20] je ime *Microsoft*-ovog skupa podataka na kom su trenirani modeli sa grafikona, dok *Val* označava da su te vrijednosti postignute na validacionom dijelu skupa podataka. Jasno je da *YOLOv8* nudi bolje performanse (veći  $mAP$ ) čak i sa manje parametara u poređenju sa prethodnim verzijama, što ukazuje na poboljšanja u efikasnosti modela i efektivnosti u otkrivanju objekata.



Slika 8. Poređenje *YOLOv8* modela sa starijim modelima iz *YOLO* porodice algoritama.<sup>2</sup>

Desni grafikon na slici 8 grafikon prikazuje performanse modela u odnosu na vrijeme obrade. *X*-osa mjeri kašnjenje, u milisekundama po slici (eng. *milliseconds per image - ms/img*), koristeći *A100 TensorRT FP16* (grafička kartica proizvođača *NVIDIA*), sa optimizovanim podešavanjima. *Y*-osa, kao i na prvom grafikonu, mjeri postignutu srednju prosječnu preciznost modela. *YOLOv8* ne samo da postiže veću preciznost, već to čini sa većom brzinom obrade slika u poređenju sa svojim prethodnicima. Ovo pokazuje da *YOLOv8* nije samo precizniji već i brži, što ga čini veoma pogodnim za aplikacije u realnom vremenu.

Oznake *n*, *s*, *m*, *l*, *x*, koje se pojavljuju na oba grafikona za sve *YOLO* modele, predstavljaju različite konfiguracije modela, kao što su nano, mali, srednji, veliki, ekstra veliki (eng. *n - nano*, *s - small*, *m - medium*, *l - large* i *x - extra large*, respektivno). Svaka konfiguracija nudi drugačiji balans između brzine i tačnosti, zadovoljavajući različite potrebe aplikacije ili hardverske mogućnosti.

Kod *YOLOv8* algoritma, ove konfiguracije su pažljivo optimizovane kako bi ponudile raznovrsna rješenja za različite scenarije:

- *YOLOv8n (Nano)*: Najmanji i najbrži model, sa najmanjim brojem parametara, dizajniran za primjene gdje je brzina ključna i gdje su dostupni ograničeni računski resursi, poput aplikacija na uređajima sa slabijim hardverom ili mobilnim uređajima. Nano modeli obično nude nižu tačnost, ali kompenzuju to izuzetno brzim vremenom obrade.

<sup>2</sup> Slika preuzeta iz [21]

- *YOLOv8s (Small)*: Manji model, koji zadržava dobru brzinu (oko 20% manju), ali sa oko 20% boljom tačnošću u poređenju sa nano konfiguracijom. Pogodan je za primjene u realnom vremenu koje zahtijevaju brze odgovore, ali i prihvatljiv nivo preciznosti u detekciji objekata, kao što su dronovi ili kamere za sigurnosne sisteme.
- *YOLOv8m (Medium)*: Srednja konfiguracija nudi balans između brzine i tačnosti, što je čini pogodnom za opšte primjene gdje je potrebna umjerena preciznost i brzina. Ovaj model se često koristi na računarima sa jačim hardverom i u aplikacijama koje zahtijevaju performanse u realnom vremenu, ali uz veći nivo tačnosti, kao što su pametni gradovi i sistemi za praćenje saobraćaja.
- *YOLOv8l (Large)*: Veći model sa značajno većim brojem parametara, što rezultira približno 45% boljom tačnošću od nano modela, na račun 2.4 puta većeg kašnjenja. Ova konfiguracija je idealna za primjene koje zahtijevaju visoku tačnost, kao što su industrijski sistemi za inspekciju ili autonomna vozila, gdje su dostupni jači hardverski resursi.
- *YOLOv8x (Extra Large)*: Najveći model, sa najviše parametara, koji nudi najviši nivo tačnosti, ali i najveće kašnjenje. Ova konfiguracija je pogodna za aplikacije kojima je preciznost apsolutni prioritet, poput medicinske analize slika ili istraživačkih projekata, gdje su računarski resursi izuzetno jaki i gdje brzina nije primarni faktor.

*YOLOv8* konstantno nadmašuje starije verzije u različitim konfiguracijama. Grafikoni na slici 8 pružaju jasan vizuelni prikaz napretka u *YOLO* porodici algoritama, ističući poboljšanja *YOLOv8* koja optimizuju i računarsku efikasnost i performanse detekcije objekata.

*YOLOv8* predstavlja značajan napredak u *YOLO* seriji modela detekcije objekata, uključujući različite karakteristike i poboljšanja [22] za unapređenje performansi i upotrebljivosti. Ključna poboljšanja u *YOLOv8* uključuju usvajanje prilagođene *CSPDarknet53 backbone* [23] **mreže za ekstrakciju karakteristika**, koja poboljšava mogućnosti izdvajanja karakteristika, čineći model efikasnijim u učenju složenih detalja unutar slika. Pored toga, *YOLOv8* integriše **mrežu agregacije putanja** [24] (eng. *Path Aggregation Network - PANet*), arhitektura neuralne mreže osmišljena da poboljša

detekciju objekata efektivnim kombinovanjem informacija iz različitih slojeva mreže. *PANet* poboljšava model tako što prikuplja i kombinuje informacije iz više slojeva neuralne mreže. Svaki od ovih slojeva uči karakteristike na različitim skalama:

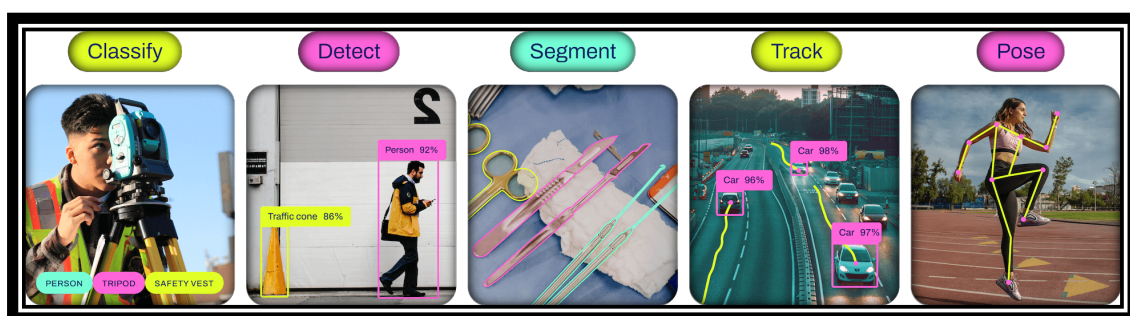
- Raniji slojevi: Fokusiraju se na fine detalje, poput ivica, tekstura i manjih obrazaca.
- Dublji slojevi: Uče veće, opštije obrasce i oblike unutar slike.

Prikupljanjem i spajanjem karakteristika iz ranijih i dubljih slojeva, *PANet* omogućava modelu da istovremeno uzme u obzir i fine detalje i širu sliku. Korišćenje detaljnih informacija iz ranih slojeva pomaže u identifikaciji sitnih ili suptilnih objekata, dok informacije iz dubljih slojeva pomažu u prepoznavanju većih objekata i ukupnog konteksta.

Dalja poboljšanja su evidentna u *YOLO*-vom **dinamičkom dodjeljivanju sidara**, koje optimizuje sidrene okvire tokom treninga kako bi se bolje nosio sa različitim oblicima i veličinama objekata. Ova funkcija pomaže modelu da bolje generalizuje različite skupove podataka i scenarije. U tradicionalnim modelima za detekciju objekata koriste se unaprijed definisani sidreni okviri sa fiksnim veličinama i proporcijama tokom cijelog procesa treniranja. Kao takvi, možda ne odgovaraju dobro svim objektima u skupu podataka, što može dovesti do manje preciznih detekcija. Dinamička dodjela sidrenih okvira rješava ovaj problem tako što prilagođava njihove dimenzije tokom treniranja. Umjesto da se oslanja na fiksni set sidrenih okvira, model dinamički podešava sidrene okvire kako bi bolje odgovarali objektima u svakoj grupi podataka za trening. Ovo omogućava modelu da bolje odgovara objektima različitih veličina i proporcija, ali mu poboljšava i performanse detekcije: dinamički sidreni okviri pomažu u smanjenju neslaganja između predviđenih graničnih okvira i stvarnih objekata, što dovodi do veće preciznosti i odziva (poglavlje 4).

*YOLOv8* takođe poboljšava proces treniranja pojednostavljenim procedurama i upotrebom **treninga mješovite preciznosti**, koja koristi i *float32* i *float16* proračune (aritmetičke operacije nad decimalnim brojevima, predstavljenim sa 32, odnosno 16 bitova) kako bi se ubrzao trening bez gubitka tačnosti. Osim toga, uvodi napredne tehnike augmentacije podataka, poput *mosaic* augmentacije (poglavlje 4.2), čime se poboljšava njegova sposobnost generalizacije u različitim situacijama.

Tehnička poboljšanja dovode do viših prosječnih ocjena preciznosti na standardnim skupovima podataka za detekciju objekata, održavajući mogućnosti izvršavanja u realnom vremenu, čineći ga veoma efikasnim za aplikacije koje zahtijevaju brzo i precizno otkrivanje objekata [25]. *YOLOv8* predstavlja najsavremeniju tehnologiju za detekciju objekata, pružajući alate koji su ne samo brzi i tačni, već i prilagodljivi širokom spektru aplikacija (slika 9) od autonomne vožnje do sistema nadzora. Kao i kod svih modela vještačke inteligencije, kako bi *YOLOv8* stvarno bio efikasan u praksi, potrebno je kontinuirano testiranje i usavršavanje, posebno u raznolikim i izazovnim okruženjima.



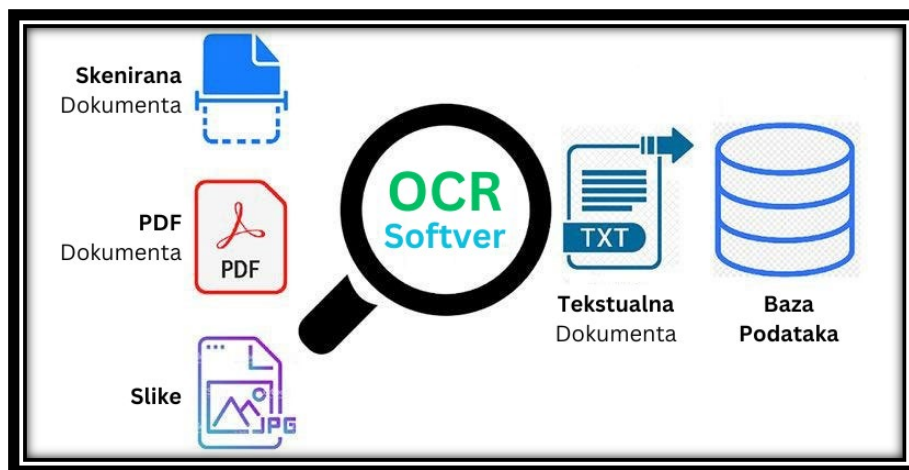
Slika 9. Razne primjene *YOLOv8* algoritma<sup>3</sup>

## 2.3. *PyTesseract* OCR algoritam

### 2.3.1. Uvod u *OCR*

Optičko prepoznavanje karaktera (eng. *Optical Character Recognition - OCR*) je još jedna značajna primjena mašinskog učenja, gdje se tehnologija koristi za pretvaranje različitih tipova dokumenata, poput skeniranih dokumenata na papiru, PDF fajlova ili slika uslikanih digitalnim fotoaparatom, u mašinski čitljive podatke (slika 10). Primarna svrha *OCR*-a je digitalizacija štampanih tekstova kako bi se mogli elektronski uređivati, pretraživati, sačuvati kompaktnije (u bazi podataka na primjer), prikazati na internet, ali i učiniti pristupačnijim, pretvaranjem pisanog sadržaja u zvučne formate. *OCR* se sve više koristi za automatizaciju procesa unosa podataka, smanjujući dugotrajan ručni unos podataka i povećavajući operativnu efikasnost.

<sup>3</sup> Slika preuzeta iz [21]



Slika 10. Dijagramski prikaz funkcije *OCR softvera*

*OCR* funkcioniše u nekoliko etapa:

1. **Prethodna obrada** (eng. *pre-processing*): Slika koja sadrži tekstualni sadržaj se priprema za dalju obradu u cilju prepoznavanja teksta. To može uključivati uklanjanje iskošenja, uklanjanje šuma (crnih tačkica, mrlja, itd.), binarizaciju (pretvaranje u crno-bijelu sliku) i normalizaciju.

Normalizaciju čine:

- **Standardizacija veličine:** Normalizacija često uključuje promjenu veličine slike na standardnu dimenziju. Ovo je ključno jer *OCR* modeli obično rade bolje kada obrađuju slike konzistentnih veličina. Ako veličina teksta previše varira unutar ili između slika, to može dovesti do loše preciznosti u prepoznavanju.
- **Skaliranje intenziteta:** Drugi aspekt normalizacije uključuje podešavanje vrijednosti intenziteta piksela. To znači skaliranje vrijednosti piksela na standardni opseg, obično od 0 do 1, ili od 0 do 255, što pomaže u smanjenju varijacija uzrokovanih različitim uslovima osvjetljenja na slikama. Normalizacija intenziteta piksela takođe može uključivati tehnike kao što je izjednačavanje upotrebom histograma, koje poboljšava kontrast slike, čineći karakteristike uočljivijim i samim tim lakšim za otkrivanje i prepoznavanje *OCR* algoritmu.
- **Korekcije odnosa širine i visine slike:** Ponekad normalizacija takođe uključuje modifikaciju odnosa širine i visine slike kako bi se spriječilo

izobličenje teksta, osiguravajući da su visina i širina karaktera na slici pravilno predstavljene. Iskrivljeni tekst može dovesti do netačnosti u prepoznavanju karaktera.

2. **Detekcija teksta:** Otkrivanje teksta je ključan korak za OCR softver. Cilj je da se lociraju oblasti teksta unutar slike. Tradicionalne metode su se oslanjale na tehnike kao što su detekcija ivica, analiza povezanih komponenti (grupisanje piksela koji imaju slična svojstva, kao što su boja ili intenzitet, u veće komponente) i morfološke operacije (poput dilatacije i erozije) za pronalaženje teksta. Ove metode su imale problema sa zašumljenim i složenim slikama. Ti rani pristupi bili su efikasni za jednostavne, dobro formatirane dokumente, ali im je nedostajala robusnost potrebna za raznovrsnije scenarije iz stvarnog svijeta. Uvođenje dubokog učenja, posebno konvolucionih neuralnih mreža (poglavlje 2.1), revolucioniralo je detekciju teksta čineći ga robusnijim i prilagodljivijim različitim veličinama teksta, orijentacijama i izobličenjima. Takvi modeli omogućili su preciznije i efikasnije otkrivanje teksta i u horizontalnim ali i bilo kako drugačije orijentisanim scenarijima. Ovi modeli obrađuju čitave slike jednom aktivacijom mreže (slično *YOLO* algoritmu, poglavlje 2.2.3), koristeći konvolucione slojeve da predvide regione od interesa u kojima je velika vjerovatnoća da se nalazi tekst, poboljšavajući brzinu i tačnost u poređenju sa starijim metodama zasnovanim na predlozima regiona (poglavlje 2.2.2).
3. **Prepoznavanje karaktera:** Ovaj korak uključuje identifikaciju i klasifikaciju pojedinačnih znakova ili riječi iz otkrivenih tekstualnih regiona na slici. Tradicionalne metode, kao što su podudaranje sa šablonima i prepoznavanje zasnovano na karakteristikama, oslanjale su se na upoređivanje znakova sa unaprijed definisanim šablonima ili izdvajanje geometrijskih karakteristika kao što su linije i krive. Iako su ovi pristupi funkcionisali za čist, dobro formatiran tekst, nisu se dobro pokazivali za rukom pisani tekst, izobličene fontove i zašumljene slike zbog oslanjanja na ručno dizajnirane funkcije. Uvođenje dubokog učenja, posebno konvolucionih neuralnih mreža, revolucioniralo je prepoznavanje karaktera automatskim učenjem obrazaca iz tekstualnih slika. *CNN*-ovi se ističu u razlikovanju znakova učenjem obrazaca kao što su ivice i oblici, se dok rekurentne neuralne mreže (poglavlje 2.1) dobro pokazuju zbog

sekvencijalne prirode znakova u riječima i rečenicama. Ova kombinacija omogućava savremenim *OCR* sistemima da prepoznaju čitave riječi ili redove teksta sa visokom preciznošću, čak i u izazovnim scenarijima kao što je rukom pisani ili izobličeni tekst.

4. **Naknadna obrada** (eng. *post-processing*): U finalnom koraku neobrađeni izlaz se prečišćava ispravljanjem grešaka i osiguravanjem da je prepoznati tekst tačan i strukturiran. Uobičajene greške tokom *OCR*-a uključuju pogrešno prepoznate znakove (npr. brkanje „O” sa „0”) i netačno formiranje riječi, posebno na slikama sa šumom ili slikama niskog kvaliteta. Da bi se ovo riješilo, naknadna obrada primjenjuje tehnike kao što su provjera pravopisa i podudaranje sa rječnikom, obezbjeđujući da su prepoznate riječi usklađene sa standardnim ili specijalizovanim rječnicima (kao što su pravni, medicinski ili naučni). Ovo pomaže u ispravljanju uobičajenih grešaka u prepoznavanju karaktera i poboljšanju ukupne tačnosti.

### 2.3.2. *PyTesseract*: *Python* alatka za *OCR*

*PyTesseract* [26] je *OCR* alat za *Python* koji služi kao omotač za *Tesseract-OCR* [27] alat, koji je jedan od najpopularnijih *OCR* alata otvorenog koda koji je danas dostupan. *Tesseract* je prvobitno razvila *HP* kompanija, a kasnije, 2005. godine, postao je softver otvorenog koda. Podržava preko 100 jezika i može se obučiti da prepoznaje i druge jezike i pisma.

*PyTesseract* se integriše sa *Tesseract* alatom, pružajući u *Python* okruženju način za obavljanje *OCR*-a na slikama. Proces čine sljedeće etape:

1. **Unos slike:** *PyTesseract* uzima kao ulaz fajl sa slikom koja sadrži tekstualni sadržaj koji treba da se konvertuje u tekstualni niz.
2. **Konfiguracija:** Korisnici mogu odrediti format izlaza i jezike prilikom poziva *PyTesseract* alata. Način na koji je to realizovano u implementaciji sistema (dodatak) je dat na slici 11. Sistem koristi *Pytesseract*-ovu metodu „*image\_to\_data*” za izvođenje optičkog prepoznavanja karaktera na slici, određenoj „*image\_path*” argumentom. Podešavanjem parametra „*lang*” na „*eng*”, *Tesseract* se upućuje da prepoznaje tekst na engleskom jeziku. Funkcija

„*image\_to\_data*” vraća detaljne informacije o detektovanom tekstu, uključujući sadržaj teksta, poziciju, veličinu i nivo pouzdanosti za svaku riječ ili karakter.

3. **Izvršenje:** Prilikom izvršavanja, *PyTesseract* koristi *Tesseract* za otkrivanje i prepoznavanje karaktera na slici. *PyTesseract* ne uključuje sam *Tesseract OCR* sistem, već samo pruža *Python* interfejs za njega. Konačno, pronađene karaktere pretvara u *string* format.
4. **Izlaz:** Izlaz je obično običan tekst ekstrahovan iz slike, uz podatke o graničnim okvirima, pouzdanosti i drugim informacijama relevantnim za *OCR* proces.

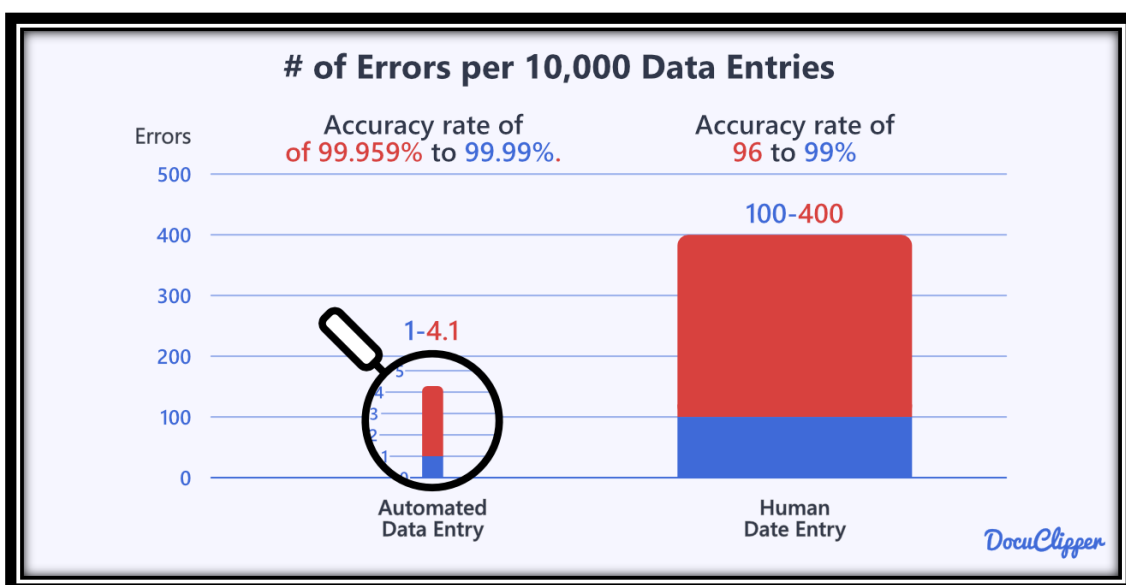
```
def run_ocr(image_path):  
    text_og = pytesseract.image_to_data(image_path, lang='eng')
```

Slika 11. Isječak iz implementacije sistema

*PyTesseract* je koristan u brojnim aplikacijama u kojima je potrebno izdvajanje teksta iz slika, kao što je obrada skeniranih dokumenata, automatizovano izdvajanje podataka iz poslovnih dokumenata kao što su fakture i priznanice i razvoj aplikacija koje treba da izvlače tekstualne informacije iz medijskih fajlova.

### 3. PRIMJENA VJEŠTAČKE INTELIGENCIJE U OBRADI DOKUMENATA IZ BANKARSKOG SEKTORA

Bankarski sektor se kontinuirano suočava sa izazovom upravljanja ogromnim količinama podataka, od kojih je značajan dio sadržan u različitim oblicima dokumenata, uključujući, ali ne ograničavajući se na ugovore, izjave, fakture i prepiske klijenata. Tradicionalne metode obrade ovih dokumenata su uglavnom ručne, dugotrajne i sklone greškama, što naglašava hitnu potrebu za efikasnijim rješenjima. Na slici 12 se vidi da automatizovani unos podataka ima stopu tačnosti od 99.959% do 99.99%, dok se stopa tačnosti za ručan unos podataka kreće se od 96% do 99%. Za 10 000 unosa podataka, automatizovani sistemi bi napravili između 1 i 4.1 greške, dok bi ljudi napravili između 100 i 400 grešaka, što pokazuje da ljudi prave 100 puta više grešaka prilikom unosa podataka u poređenju sa automatizovanim sistemima. Ovo poglavlje istražuje integraciju vještačke inteligencije u automatizaciju obrade dokumenata u bankarskom sektoru, sa posebnim fokusom na tabelarnim dokumentima koji preovlađuju u finansijskim transakcijama i izvještavanju.



Slika 12. Statističko poređenje ručnog i automatizovanog unosa podataka iz 2024. godine<sup>4</sup>

<sup>4</sup> Slika i statistika preuzeti iz [28]

### **3.1. Aktuelna rješenja za obradu dokumenata u bankarstvu**

Obrada dokumenata u bankarstvu uključuje brojne korake kao što su unos podataka, verifikacija dokumenata i provjere usklađenosti, od kojih svaki može imati značajnu korist od automatizacije. Taj proces se najčešće, čak i u vodećim bankama, obavlja ručno. Trenutna rješenja za automatizaciju digitalizacije dokumenata uključuju optičko prepoznavanje karaktera za pretvaranje skeniranih slika teksta u mašinski čitljiv tekst i softver zasnovan na pravilima za izdvajanje relevantnih informacija. Međutim, ova rješenja se često bore sa složenošću i varijabilnošću bankarskih dokumenata. Na primjer, *OCR* tehnologije mogu griješiti u susretu sa rukom pisanim bilješkama ili skeniranjem lošeg kvaliteta, a sistemi zasnovani na pravilima zahtijevaju česta ažuriranja da bi obradili nove formate dokumenata ili uzeli u obzir regulatorne promjene.

### **3.2. Izazovi specifični za bankarski sektor**

Bankarski sektor postavlja jedinstvene izazove za obradu dokumenata. Veliki obim transakcija i kritična priroda tačnosti finansijskih podataka zahtijevaju robusna rješenja koja moraju da odgovore na izazove. Jedan od glavnih izazova su složeni formati dokumenata. Bankarski dokumenti često imaju složen raspored, sa više podataka od interesa, kao što su tabele historijata transakcija ili rasporedi amortizacije kredita (slika 13). Pored strukture dokumenata, regulative u finansijskom sektoru predstavljaju dodatni izazov. Banke se moraju osigurati da njihovi dokumenti zadovoljavaju stroge propise u vezi sa rukovanjem podacima i zaštitom privatnosti. Zakonodavni okviri često se mijenjaju, pa sistemi u bankarskom sektoru moraju biti prilagodljivi kako bi ostali u skladu s najnovijim propisima. Greške u obradi finansijskih dokumenata mogu dovesti do značajnih novčanih gubitaka i pravnih posljedica, što bankarske podatke čini izuzetno osjetljivim.

FIRST BANK OF WIKI		CHEQUING ACCOUNT STATEMENT			
1425 JAMES ST, PO BOX 4000 VICTORIA BC V8X 3X4 1-800-555-5555		Page : 1 of 1			
JOHN JONES 1643 DUNDAS ST W APT 27 TORONTO ON M6K 1V2		Statement period 2003-10-09 to 2003-11-08		Account No. 00005- 123-456-7	
Date	Description	Ref.	Withdrawals	Deposits	Balance
2003-10-08	Previous balance				0.55
2003-10-14	Payroll Deposit - HOTEL			694.81	695.36
2003-10-14	Web Bill Payment - MASTERCARD	9685	200.00		495.36
2003-10-16	ATM Withdrawal - INTERAC	3990	21.25		474.11
2003-10-16	Fees - Interac		1.50		472.61
2003-10-20	Interac Purchase - ELECTRONICS	1975	2.99		469.62
2003-10-21	Web Bill Payment - AMEX	3314	300.00		169.62
2003-10-22	ATM Withdrawal - FIRST BANK	0064	100.00		69.62
2003-10-23	Interac Purchase - SUPERMARKET	1559	29.08		40.54
2003-10-24	Interac Refund - ELECTRONICS	1975		2.99	43.53
2003-10-27	Telephone Bill Payment - VISA	2475	6.77		36.76
2003-10-28	Payroll Deposit - HOTEL			694.81	731.57
2003-10-30	Web Funds Transfer - From SAVINGS	2620		50.00	781.57
2003-11-03	Pre-Auth. Payment - INSURANCE		33.55		748.02
2003-11-03	Cheque No. - 409		100.00		648.02
2003-11-06	Mortgage Payment		710.49		-62.47
2003-11-07	Fees - Overdraft		5.00		-67.47
2003-11-08	Fees - Monthly		5.00		-72.47
*** Totals ***			1,515.63	1,442.61	

Slika 13. Primjer složenog bankarskog dokumenta

### 3.3. Ograničenja postojećih rješenja za automatizaciju

Iako su postojeća rješenja napredovala u automatizaciji digitalizacije dokumenata, njihova ograničenja postaju očigledna kada se suoče sa zahtjevima bankarskog sektora:

- **Ograničena prilagodljivost:** Tradicionalni *OCR* i sistemi zasnovani na pravilima nisu inherentno fleksibilni na promjene u formatima dokumenata ili neočekivane tipove podataka.
- **Stope grešaka:** Preciznost potrebna za finansijske dokumente često premašuje toleranciju grešaka trenutnih automatizovanih sistema.
- **Nedostatak razumijevanja konteksta:** Trenutni sistemi ne razumiju kontekst ili važnost informacija koje se obrađuju, što dovodi do potencijalnih previda u kritičnim tačkama podataka.

### 3.4. Potreba za integrisanim *AI* pristupom

Kako bi se prevazišli izazovi u obradi složenih finansijskih dokumenata, raste potreba za integrisanim pristupom koji koristi napredne *AI* tehnologije. Modeli mašinskog učenja mogu se obučiti da prepoznaju i izvlače informacije iz složenih obrazaca i specifično strukturiranih dokumenata, karakterističnih za banke, što olakšava izdvajanje relevantnih podataka. Integrisani pristup se, takođe, može unaprijediti upotrebom modela za obradu prirodnog jezika (*NLP*). Oni omogućavaju dublje razumijevanje teksta unutar konteksta, analizirajući riječi i fraze i uzimajući u obzir okolni tekst, ne samo njihov doslovni zapis, što povećava tačnost ekstrakcije podataka i relevantnost obrađenih informacija. Prilagodljivo učenje, karakteristika *AI* sistema, omogućava im da kontinuirano uče iz novih podataka, čineći ih sposobnim da se prilagode promjenama u formatima dokumenata i zakonskim regulativama, bez potrebe za stalnim reprogramiranjem.

Primjena vještačke inteligencije u obradi dokumenata u bankarskom sektoru ima značajan potencijal za poboljšanje efikasnosti, smanjenje stopa grešaka i održavanje usklađenosti sa regulatornim zahtjevima. Prelaskom sa tradicionalnih metoda na sofisticiranije sisteme vođene vještačkom inteligencijom, banke mogu ne samo da poboljšaju operativnu efikasnost već i da poboljšaju zadovoljstvo klijenata ubrzanjem obrade transakcija i pružanjem preciznijih finansijskih informacija. Ovo poglavlje naglašava neophodnost integrisanog *AI* pristupa prilagođenog jedinstvenim zahtjevima bankarske industrije, postavljajući teren za razvoj robusnijih i prilagodljivijih sistema za obradu dokumenata.

## 4. OPIS SISTEMA ZA DIGITALIZACIJU DOKUMENATA

Kako bi se testirala hipoteza i odgovorilo na istraživačka pitanja postavljena u ovom radu, koristiće se višestruki metodološki pristup. Pažljivo će se procjenjivati performanse dva *YOLOv8* modela koje sistem koristi. Prvi model je obučen na prilagođenom skupu podataka od 10 000 slika, za otkrivanje pasusa i tabela. Drugi model je obučen na *Microsoft*-ovom otvorenom skupu podataka [29] koji se sastoji od milion slika za složenu analizu strukture tabele (identifikovanje redova i kolona). *PyTesseract OCR* koji se koristi u svom standardnom obliku bez dodatnog treniranja, dopunjuje ove modele izdvajanjem tekstualnog sadržaja nakon detekcije.

Metodologija obuhvata nekoliko faza:

- **Priprema skupa podataka:** Svi bazični *YOLOv8* modeli su trenirani na *COCO* [20] skupu podataka. Njega sačinjavaju slike iz svakodnevnog života i namijenjen je za prepoznavanje generalnih objekata (automobila, ljudi itd.). Kako bi se *YOLOv8* model mogao uspješno primijeniti na analizirani problem koriste se prilagođeni skup podataka i *Microsoft*-ov skup podataka. Prilagođeni skup podataka uključuje raznovrstan niz skeniranih bankarskih dokumenata, koji obuhvataju široki spektar tabelarnih formata, kako bi se osigurao sveobuhvatan trening i detekcija objekata od interesa. Javno dostupni skup podataka koji obezbjeđuje *Microsoft* služi za fino podešavanje drugog *YOLOv8* modela za specijalizovano prepoznavanje strukture tabela.
- **Obuka i optimizacija modela:** Oba modela *YOLOv8* će proći procese treninga i validacije. Trening prvog modela na prilagođenom skupu podataka ima za cilj otkrivanje opšteg rasporeda, dok se drugi model fokusira na detaljnu identifikaciju strukture tabela pomoću obimnog *Microsoft* skupa podataka. Podešavanjem hiperparametara će se vršiti optimizacija performansi modela.
- **Integracija i implementacija pipeline-a sistema:** Trenirani *YOLOv8* modeli su sa *PyTesseract OCR*-om integrisani u kohezivni sistem. Na kraju, detekcije *OCR*-a i modela za detekciju objekata se objedinjuju, dajući na izlazu sistema, u JSON formatu, informacije o tekstu i njegovom položaju u dokumentu. Izlaz je struktuiran kao lista objekata, koji predstavljaju *YOLO* detekcije. Svaki pasus

sadrži ključ „*words*”, u koji se smiještaju *OCR* detekcije (riječi) koje mu prostorno pripadaju. Slično, tabele sadrže listu vrijednosti za ključ „*cells*”, gdje su smještene riječi koje pripadaju odgovarajućim ćelijama.

➤ **Procjena učinka:** Performanse sistema će se procjenjivati upotrebom standardnih *ML* metrika:

1. **Preciznost** mjeri tačnost napravljenih detekcija (tabela, redova, kolona ili blokova teksta u dokumentu). To je proporcija ispravnih detekcija u odnosu na sve napravljene detekcije.

$$P = \frac{ID^5}{ID + ND} \quad (6)$$

2. **Odziv** mjeri sposobnost modela da pronađe sve relevantne instance klase. To je proporcija stvarno ispravnih detekcija u odnosu na broj uzoraka koji su trebali biti detektovani. Računa se za svaku klasu posebno.

$$R = \frac{ID}{ID + ND} \quad (7)$$

3. **F1 skor** kombinuje preciznost i odziv u jednu metriku kako bi procijenio ravnotežu modela između identifikacije svih relevantnih stavki i minimiziranja netačnih identifikacija. Računa se formulom:

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \quad (8)$$

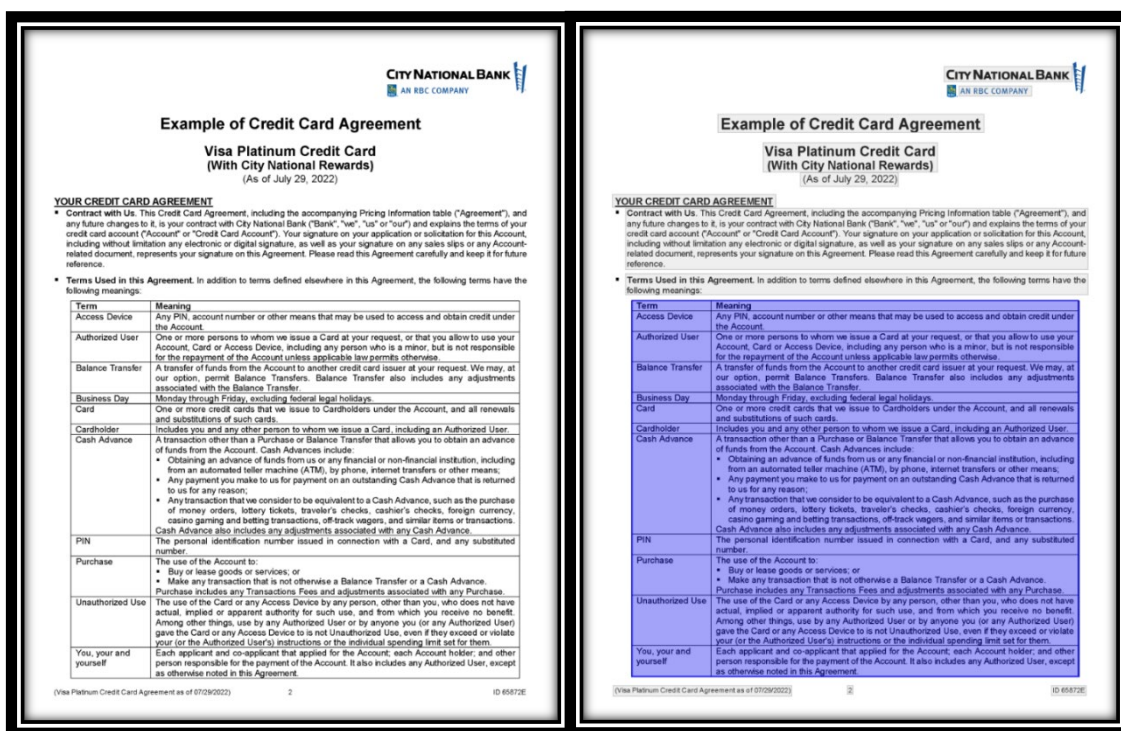
4. **mAP** je detaljno objašnjen u poglavlju 2.2.5.
5. **Vrijeme obrade** procenjuje efikasnost sistema u smislu koliko brzo može da obrađuje dokumente, mjereći vrijeme od unosa neobrađene slike dokumenta do izlaza strukturiranog digitalnog formata.

---

<sup>5</sup> *P* je oznaka za preciznost, *R* za odziv, *ID* za ispravne detekcije i *ND* je oznaka za nedostajuće detekcije.

## 4.1. Priprema skupa podataka

Prvi skup podataka, ključan za obuku *YOLOv8* modela za otkrivanje opšteg rasporeda teksta u dokumentima, odnosno lociranje paragrafa i tabela, pažljivo je sastavljen mojim angažmanom u kompaniji Uhura Solutions, gdje sam zaposlen. Ukupno 10 000 slika, koje čine skup podataka, kombinacija su bankovnih dokumenata, obezbijedenih od strane klijenata Uhura Solutions kompanije i javno dostupnih ugovora o kreditnim karticama. Proces anotiranja uključivao je kolektivne napore zaposlenih kompanije i posebnog tima honorarno zaposlenih (uglavnom studenata), angažovanih zbog svoje stručnosti u preciznom anotiranju dokumenata. Pokazalo se da je proces i izazovan i dugotrajan, sa brojnim iteracijama ispravki i evaluacija kako bi se osigurao kvalitet i tačnost anotacija. Na slici 14 je dat primjer anotiranog dokumenta, za trening *YOLOv8* modela koji prepoznaje paragrafe (sive anotacije) i tabele (plave anotacije).



Slika 14. Primjer sirovog - prije obrade i obilježavanja (lijevo) i anotiranog (desno) dokumenta

Za otkrivanje struktura tabela, odnosno lociranje redova i kolona unutar tabela, za šta je zadužen drugi *YOLOv8* model, korišten je javno dostupan skup podataka koji je obezbijedio *Microsoft*. Pomenuti skup podataka se sastoji od 1 milion anotiranih slika

tabela. Ovaj bogati skup podataka je bio od ključnog značaja u obučavanju modela da precizno prepozna i najsitnije elemente unutar tabela.

## 4.2. Trening modela i optimizacija

U ovom poglavlju dat je sveobuhvatan skup parametara koji se koriste za trening YOLOv8 modela, kako onog za detekciju paragrafa i tabela, tako i onog za detekciju strukture samih tabela u bankarskim dokumentima. Podešavanja za trening, dizajnirana da maksimizuju efikasnost i tačnost detekcije, uključuje mješavinu podrazumijevanih i prilagođenih parametara za fino podešavanje modela pod određenim ograničenjima i zahtjevima.

Pregled ključnih parametara obuke (slika 15):

### 1. Konfiguracija modela i okruženja

- *task = detect*: Određuje glavni zadatak modela, a to je otkrivanje objekata.
- *mode = train*: Model radi u režimu za trening.
- *model = yolov8l.pt*: Koristi *YOLOv8 large* arhitekturu modela (poglavlje 2.2.5), obezbjeđujući ravnotežu između brzine i tačnosti, sa akcentom na tačnosti.
- *data = data.yaml*: Ukazuje na YAML fajl koji sadrži putanje do skupa podataka za trening i validaciju, zajedno sa definicijom klasa modela. Za trening spoljašnjeg *YOLOv8* modela (poglavlje 5.2.2), koji je dat na slici 15, definisane su klase „*paragraph*”, za otkrivanje pasusa i „*table*”, za otkrivanje tabela. Prilikom treninga unutrašnjeg modela (poglavlje 5.2.5), definisane klase su „*table*”, „*row*” i „*column*”, što je pojednostavljenje originalnog *Microsoft*-ovog skupa podataka.
- *epochs = 200*: Broj potrebnih epoha (poglavlje 2.1) može da varira u zavisnosti od složenosti modela i prirode podataka. Model se trenira u 200 epoha, omogućavajući dovoljno iteracija za konvergenciju, odnosno dostizanje maksimalne tačnosti za tu konfiguraciju.

- *patience* = 20: Trening će prestati ako se ne vidi smanjenje gubitaka (poglavlje 2.1) na validacionom skupu podataka, tokom 20 uzastopnih epoha, kako bi se spriječilo prekomjerno prilagođavanje modela skupu podataka za trening.

## 2. Optimizacija performansi

- *batch* = 32: Veličina grupe podataka (poglavlje 2.1) je podešena na 32, optimizujući balans između upotrebe memorije i brzine, uzimajući maksimalnu moguću vrijednost koju memorija grafičkih kartica korištenih za trening omogućava.
- *imgsz* = 640: *Image size* određuje veličinu na koju se skaliraju sve ulazne slike prije nego što ih model obradi. On postavlja dimenzije (visinu i širinu) ulaznih slika na uniformnu veličinu – 640 piksela u ovom slučaju. Ova uniformnost je ključna jer neuralne mreže, posebno one koje čija je svrha obradu slika, kao što je *YOLOv8*, zahtijevaju ulazne podatke konzistentnog oblika i veličine. Skaliranjem slike na 640 piksela, osigurava se da je „vidno polje” modela prilagođeno ovoj specifičnoj rezoluciji. Vidno polje se odnosi na oblast ulazne slike koju model može efikasno da „vidi” i analizira odjednom. Podešavanje veličine ulazne slike utiče na to kako model percipira detalje unutar slike. Na primjer, veće slike mogu zadržati detaljnije karakteristike, što može biti korisno za otkrivanje malih ili složenih objekata unutar scene. Kako bi se zadržao maksimalan broj detalja, što dovodi do preciznije detekcije, veličina slike se podešava na najveću moguću vrijednost u skladu sa dostupnim računarskim resursima.
- *optimizer* = auto: Automatski bira najbolji optimizator na osnovu podataka o obuci i arhitekture modela. Optimizator je ključan dio procesa treniranja u neuralnim mrežama (poglavlje 2.1), a posebno je važan prilikom propagacije unazad. Nakon što se tokom propagacije unazad izračunaju gradijenti funkcije gubitka, optimizator koristi te gradijente za ažuriranje težinskih koeficijenata i koeficijenata pristrasnosti modela. Cilj optimizatora je da minimizira funkciju gubitka tako što iterativno prilagođava težinske koeficijente, vodeći računa o tome koliko i u kojem

smjeru treba promijeniti svaki koeficijent. Optimizatori poput gradijentnog spusta koriste ove informacije iz propagacije unazad kako bi mreža postepeno učila iz grešaka i poboljšavala svoja predviđanja.

- *amp* = True: eng. *Automatic Mixed Precision (AMP)* je tehnika koja se koristi u dubokom učenju (poglavlje 2) i upotrebljava i 32-bitne (eng. *single-precision*) i 16-bitne (eng. *half-precision*) formate sa pokretnim zarezom tokom treninga, radi optimizacije efikasnosti proračuna. *AMP* poboljšava brzinu treninga i smanjuje upotrebu memorije neuralnih mreža bez značajnog uticaja na tačnost modela. Dinamičkim podešavanjem preciznosti tokom različitih faza procesa treninga, *AMP* omogućava bržu obradu na kompatibilnom hardveru, kao što su moderni GPU-ovi koji su optimizovani za aritmetiku niže preciznosti. Ova mogućnost omogućava obuku većih modela ili korišćenje većih grupa podataka uz ista memorijska ograničenja, efikasno poboljšavajući performanse i skalabilnost zadataka dubokog učenja.

### 3. Podešavanja hardvera i izvršavanja

- *device* = (0, 1): Koristi dva *GPU*-a za obuku, raspoređujući radno opterećenje radi efikasnosti.
- *workers* = 8: Broj podprocesa koji se koriste za učitavanje podataka, podešen na 8 da bi se iskoristila višejezgarna, paralelna obrada.
- *verbose* = True: Omogućava detaljan ispis na početku treninga, ali i nakon svake epohe, za pružanje detaljne evidencije procesa treninga.

### 4. Stopa učenja i momentum

- *lr0* = 0.01: Početna stopa učenja (eng. *learning rate*), koja određuje veličinu koraka na početku treninga. Stopa učenja je ključan parametar koji utiče na prilagođavanje parametara modela tokom treninga. Ona određuje magnitudu ažuriranja primijenjenih na težinske koeficijente modela (poglavlje 2.1), kao odgovor na izračunatu grešku u svakom koraku procesa učenja. Odgovarajuće odabrana stopa učenja obezbjeđuje efikasnu konvergenciju do minimalne greške, balansirajući brzinu konvergencije sa rizikom od prekoračenja minimuma ili zarobljavanja u

lokalnim minimumima. Odabrana je standardna početna stopa učenja po preporuci *YOLOv8* dokumentacije [30].

- $lrf = 0.1$ : Konačna stopa učenja, od koje zavisi brzina opadanja stope učenja kako se kraj treniga približava.
- $momentum = 0.937$ : Inspirisan je fizičkim momentumom u mehanici, gdje pomaže objektu da savlada prepreke i da se odupre naglim promjenama u kretanju. U treniranju neuralne mreže, momentum pomaže optimizacionom algoritmu da se kreće duž relevantnih pravaca i izgladi ažuriranja. To postiže dodavanjem trenutnom dio prethodnog ažuriranja. Prilikom izračunavanja gradijenta funkcije gubitaka u svakom koraku, kako bi se ažurirali težinski koeficijenti modela, umjesto da se koristi samo trenutni gradijent, upotrebom momentuma takođe se uzima u obzir dio prethodnog ažuriranja. Zadati momentum iznosi 0.937, što je relativno visoka vrijednost i označava da se ažuriranja značajno oslanjaju na prethodne korake.

Prednosti upotrebe momentuma:

1. Akumuliranjem vektora brzine (smjera i inteziteta kretanja optimizacije) u pravcima kontinualnog smanjenja funkcije gubitka, momentum može dovesti do **brže konvergencije**.
2. Doprinosi **smanjenju oscilacija** u ažuriranjima. Bez momentuma, optimizator bi mogao da prolazi naprijed-nazad kroz uvalu na površini gubitaka. Uvala je područje površine gubitaka gdje je vrijednost funkcije gubitaka relativno niska u poređenju sa okolnim regionima, stvarajući „dolinu” unutar koje optimizator može da oscilira, zarobljen u lokalnom minimumu, što otežava pronalaženje globalnog minimuma. Sa momentumom, svako ažuriranje povećava brzinu pri blažim nagibima, što dovodi do stabilnije i dosljednije konvergencije.
3. Dodatna inercija može pomoći da se **izbjegnu lokalni minimumi** funkcije gubitaka, koji bi mogli da zarobe optimizator ako se koristi standardni gradijentni spust.

## 5. Regularizacija i augmentacija

- *weight\_decay* = 0.0005: Pomaže u sprečavanju pretjeranog prilagođavanja skupu podataka za trening, „kažnjavanjem” velikih težinskih koeficijenata.
- *augment* = False: Augmentacija podataka je tehnika koja se koristi u mašinskom učenju kako bi se vještački povećala veličina i raznolikost skupa podataka za trening, kreiranjem modifikovanih verzija postojećih podataka. Ovaj proces uključuje primjenu niza nasumičnih, ali realističnih transformacija na skupu podataka za trening, kao što su rotiranje, skaliranje, isijecanje ili promjena boje i osvjetljenosti slika. Cilj augmentacije podataka je da simulira varijabilnost na koju model može naići u stvarnom svijetu, što pomaže da se poboljša robusnost i sposobnost generalizacije modela. Uvođenjem ovih varijacija, augmentacija podataka pomaže u sprečavanju prekomjernog prilagođavanja modela. Augmentacija skupa podataka je onemogućena kako bi se održala originalnost karakteristika dokumenta za ovu specifičnu aplikaciju.
- *mosaic* = 1.0, *mixup* = 0.0: Kontroliraju obim primjene *Mosaic* i *Mixup* tehnika augmentacije skupa podataka, respektivno. ***Mosaic*** augmentacija kombinuje četiri različite slike iz skupa podataka za trening u jednu, kompozitnu sliku. To radi tako što novu sliku dijeli na kvadrante i u svaki kvadrant ubacuje nasumično isječen dio neke druge slike za trening. Ovaj metod ne samo da povećava varijabilnost podataka za trening, već i podstiče model da nauči robusnije reprezentacije karakteristika tako što ga izlaže višestrukim kontekstima i razmjerama objekata unutar jedne slike. Posebno je koristan za modele detekcije objekata jer simulira složene scene sa više objekata. ***Mixup*** augmentacija stvara nove slike za trening uzimanjem konveksne kombinacije dvije slike i njihovih anotacija. Konkretno, dvije slike se spajaju, dok se njihove odgovarajuće anotacije takođe miješaju proporcionalno stepenu miješanja u slikama. Na primjer, ako se slika A i slika B pomiješaju sa težinama od 0.7 i 0.3, respektivno, rezultujuća slika (svaki njen piksel) je 70% slike A i 30% slike B, a anotacije se kombinuju u istim razmjerama. Onemogućavanjem *augment*

parametra, onemogućene su i *mosaic* i *mixup* augmentacije, što u slučaju sistema za digitalizaciju dokumenata osigurava da trening podaci ostanu reprezentativni za stvarne zadatke koje će model obavljati, održavajući jasno razlikovanje klasa i povećavajući ukupnu tačnost i pouzdanost modela.

## 6. Validacija i čuvanje

- *val* = True: Omogućava provjeru valjanosti pomoću validacionog skupa podataka, navedenog u *data.yaml* fajlu.
- *save\_period* = -1: *Save period* predstavlja učestalost čuvanja *checkpoint*-a modela (stanja koeficijenata modela tokom treninga), navedeno u epohama. Vrijednost -1 onemogućava ovu funkciju, pamteći samo konačno stanje modela.
- *save\_dir* = *Computer Vision/Object Detection/*: Određuje direktorijum gdje se čuvaju obučeni model i povezani podaci.

```

!yolo task=detect mode=train model=yolov8l.pt \
  data=/media/mlflow/Drugi_SSD/Documents/ML_models/document_layout/Datasetovi/Dataset_ugovori_veliki_10/data.yaml \
  patience=20 epochs=200 batch=32 imgsz=640 lr0=0.01 lrf=0.1 device=0,1 project='Computer Vision/Object Detection/Para

New https://pypi.org/project/ultralytics/8.3.2 available 🤗 Update with 'pip install -U ultralytics'
Ultralytics YOLOv8.0.150 🚀 Python-3.9.13 torch-1.9.1 CUDA:0 (NVIDIA GeForce RTX 3090, 24253MiB)
CUDA:1 (NVIDIA GeForce RTX 3090, 24253MiB)
WARNING ⚠ Upgrade to torch>=2.0.0 for deterministic training.
engine/trainer: task=detect, mode=train, model=yolov8l.pt, data=/media/mlflow/Drugi_SSD/Documents/ML_models/document_layout/Datasetovi/Dataset_ugovori_veliki_10/data.yaml, epochs=200, patience=20, batch=32, imgsz=640, save=True, save_period=-1, cache=False, device=(0, 1), workers=8, project=Computer Vision/Object Detection/Paragrafi&Tabele, name=yolov8_general, exist_ok=False, pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=True, single_cls=False, rect=False, cos_lr=False, close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False, overlap_mask=True, mask_ratio=4, dropout=0.0, val=True, split=val, save_json=False, save_hybrid=False, conf=None, iou=0.7, max_det=300, half=False, dnn=False, plots=True, source=None, show=False, save_txt=False, save_conf=False, save_crop=False, show_labels=True, show_conf=True, vid_stride=1, line_width=None, visualize=False, augment=False, agnostic_nms=False, classes=None, retina_masks=False, boxes=True, format=torchscript, keras=False, optimize=False, int8=False, dynamic=False, simplify=False, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.1, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, tran_slate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.0, copy_paste=0.0, cfg=None, tracker=botsort.yaml, save_dir=Computer Vision/Object Detection/Paragrafi&Tabele/yolov8_general128
Overriding model.yaml nc=80 with nc=2
  
```

Slika 15. Podešavanje parametara i pokretanje *YOLOv8* treninga

Ovo detaljno podešavanje osigurava da je *YOLOv8* model pedantno „skrojen” i optimizovan za definisane potrebe detekcije pomenutih klasa u bankarskim dokumentima, naglašavajući prilagodljivost i skalabilnost rješenja vođenih vještačkom inteligencijom u obradi dokumenata.

### 4.3. Kriterijumi za evaluaciju performansi

Performanse *YOLOv8* modela su procijenjene na osnovu standardnih metrika koje obezbjeđuje *YOLO* algoritam, uključujući preciznost, odziv i *mAP* (poglavlje 2.2.5), uz dodatak vremena predikcije.

Za procjenu *PyTesseract* OCR-a razvijen je prilagođen sistem za evaluaciju, fokusirajući se na tačnost ekstrakcije teksta i sposobnost modela da održi vjernost formatiranja u poređenju sa originalnim dokumentima. Evaluacija je uključivala poređenje *OCR* izlaza sa ručno verifikovanim transkripcijama istih dokumenata kako bi se izračunala stopa grešaka na nivou karaktera i riječi, kao i tačnost. Takođe, iz istih razloga kao i za *YOLO* modele, bilježeno je vrijeme predikcije *PyTesseract* modela.

## **5. DIZAJN I IMPLEMENTACIJA SISTEMA**

### **5.1. Pregled sistema**

Ovo poglavlje opisuje tehničku arhitekturu naprednog sistema za digitalizaciju dokumenata dizajniranog da integriše i modele detekcije objekata (poglavljje 2.2.2) i tehnologiju optičkog prepoznavanja karaktera (poglavljje 2.3.1) za efikasnu obradu bankarskih dokumenata. Arhitektura ne samo da podržava robusnu analizu dokumenata, od ulaza do generisanja izlaza, već takođe predstavlja primjer inovativnih aspekata u rukovanju složenim rasporedima teksta dokumenata i tekstualnim podacima.

Sistem je strukturiran da obrađuje dokumenta kroz niz koraka koji počinju uvozom biblioteka i učitavanjem slika dokumenta, a kulminiraju generisanjem strukturiranog *JSON* izlaza koji sadrži detalje o izgledu i sadržaju dokumenta. Naredno poglavlje detaljno opisuje sekvencijalne operacije i funkcionalnosti svake komponente.

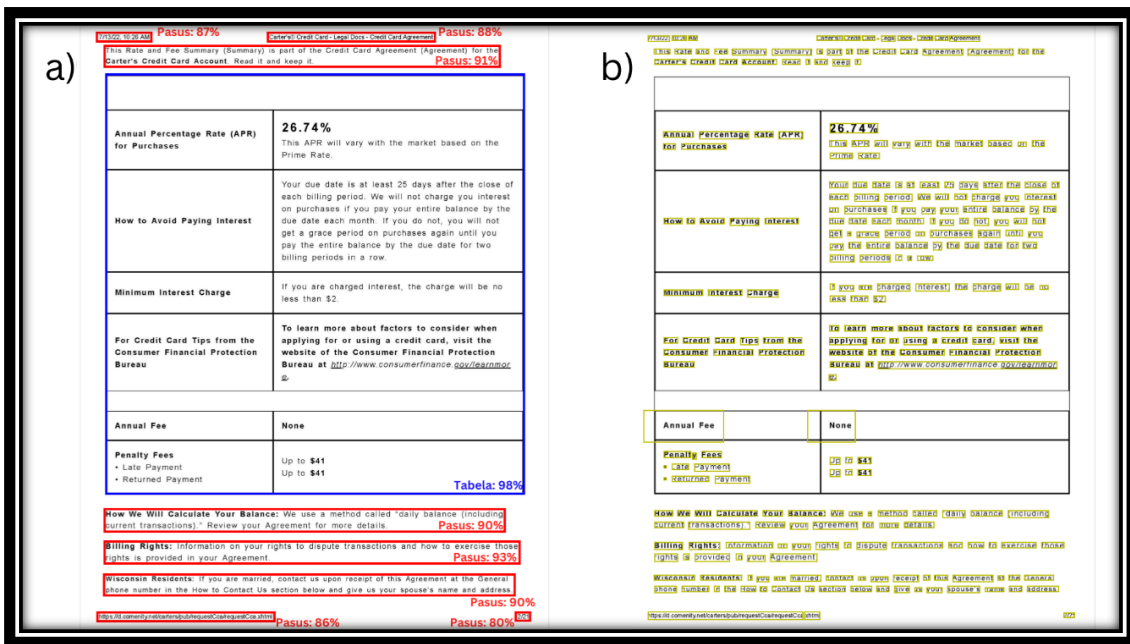
### **5.2. Detaljan tok procesa**

#### **5.2.1. Početno podešavanje i učitavanje slike**

Nakon inicijalizacije sistema, potrebne biblioteke se učitavaju kako bi omogućile obradu slike i operacije modela. Dokumenti, posebno oni sa više stranica, učitavaju se uzastopno kako bi se osiguralo da se svaka stranica pojedinačno obrađuje, održavajući integritet i redosled dokumenta.

#### **5.2.2. Otkrivanje strukture dokumenta**

Prvi korak u analizi je pokretanje „spoljašnjeg“ *YOLO* modela (poglavljje 2.2), koji identifikuje i locira paragafe i tabele na slici dokumenta (slika 16a). Ovaj korak je ključan za razumijevanje osnovne strukture i rasporeda, prije detaljnog izdvajanja teksta.



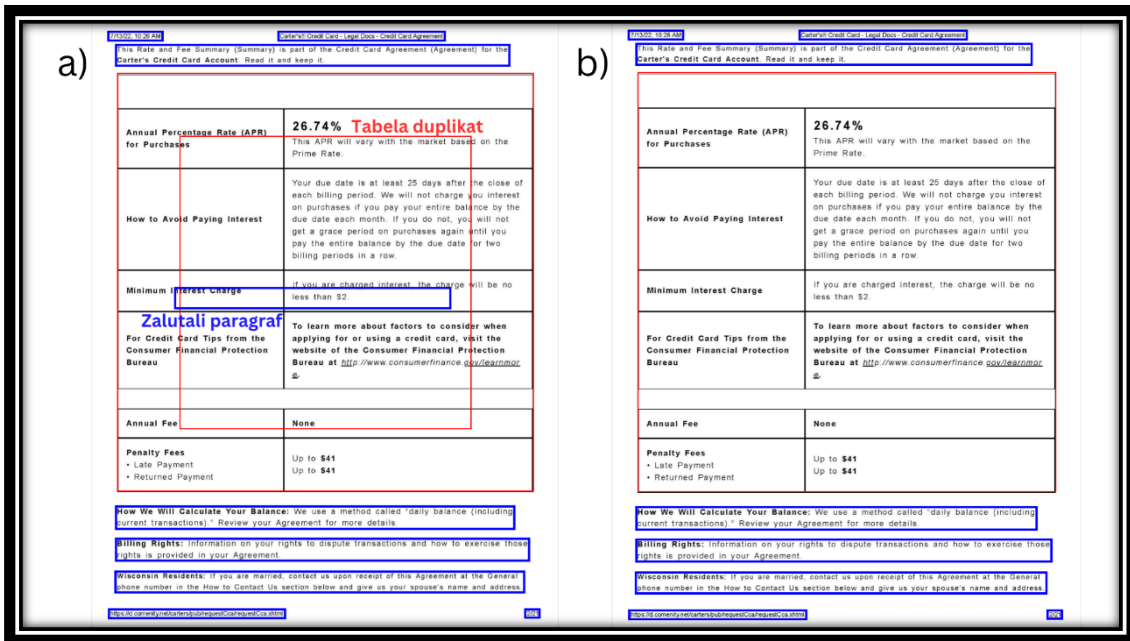
Slika 16. a) Prikaz izlaza spoljašnjeg YOLO modela, b) Prikaz izlaza PyTesseract OCR-a

### 5.2.3. Uklanjanje pogrešnih detekcija

Nakon početne detekcije, sistem obavlja dvije ključne funkcije, za uklanjanje grešaka:

- **remove\_rogue\_paragraphs:** Eliminise paragrafe koji su pogrešno identifikovani unutar granica tabele. Funkcija provjerava lokaciju svakog detektovanog paragrafa i uklanja one čiji se centar nalazi unutar neke od detektovanih tabela.
- **remove\_duplicates:** Rješava slučajeve u kojima se isti paragraf ili tabela detektuje više puta, zadržavajući samo najveću instancu, na osnovu poređenja zajedničke površine. Duplikati se identifikuju računanjem praga preklapanja, postavljenog na 40%. Ako zajednička površina, odnosno površina preklapanja, dva granična okvira iste klase iznosi više od 40% manjeg od njih, to je znak da je došlo do duple detekcije. Dotična greška se ispravlja brisanjem manjeg graničnog okvira, kako bi ostao samo veći za koji je pretpostavljeno da je ispravan.

Na slici 17 je prikazano da, i pored početne duple detekcije tabele i pogrešno detektovanog paragrafa (slika 17a), na izlazu tog dijela sistema se (slika 17b), nakon automatizovanog uklanjanja grešaka, nalaze ispravni granični okviri.



Slika 17. a) Prikaz grešaka pri detekciji, b) Stanje nakon poziva funkcija iz poglavlja 5.2.3<sup>6</sup>

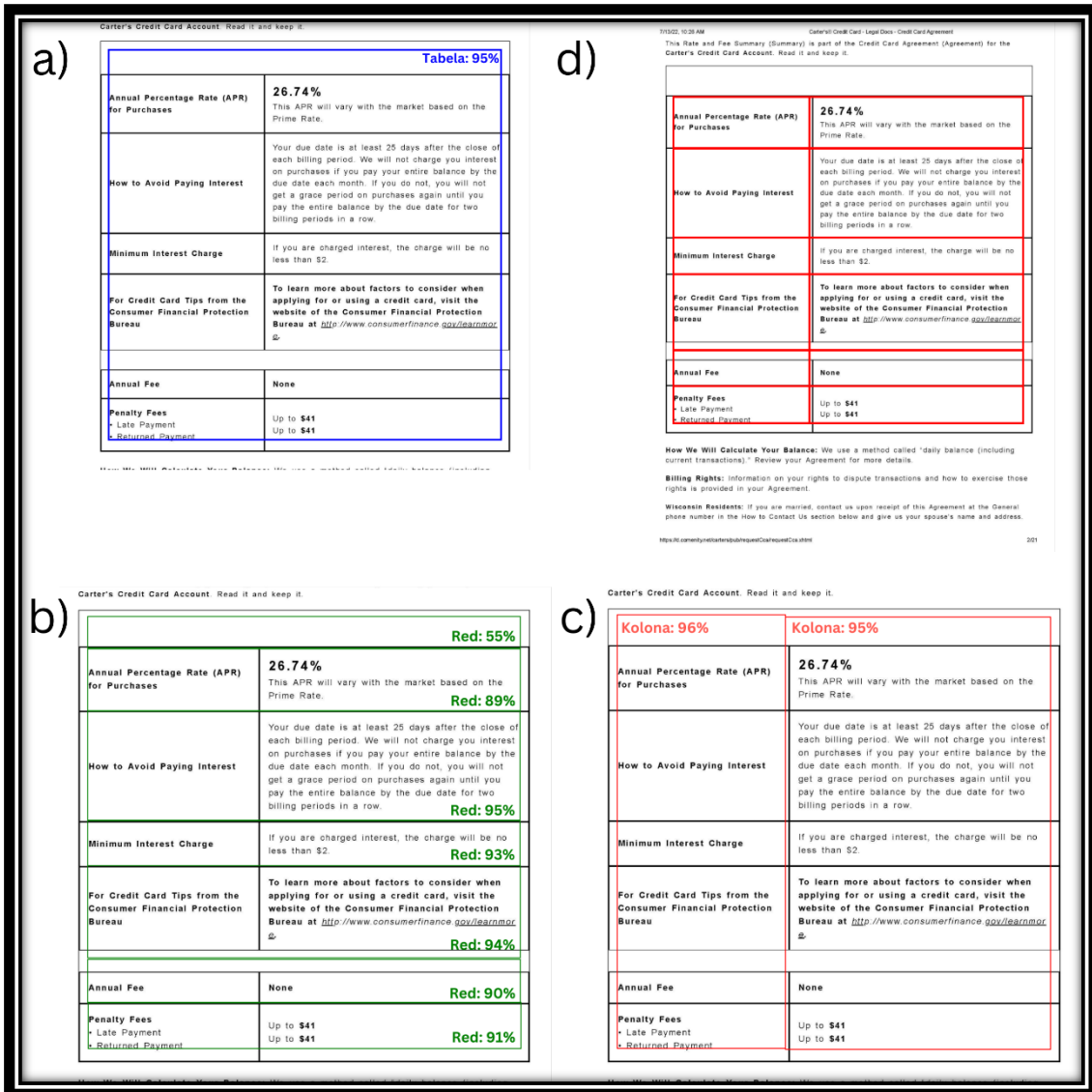
## 5.2.4. Prepoznavanje teksta

Sa identifikovanom strukturom dokumenta, *PyTesseract OCR* (poglavlje 2.3) se koristi za pronalaženje tekstualnog sadržaja u otkrivenim regionima (slika 16b). *OCR* proces je precizan, osiguravajući da je tekst tačno identifikovan i iz pasusa i iz tabela.

## 5.2.5. Analiza i obrada tabele

Sistem koristi funkciju *crop\_and\_extend\_images* kako bi izolovao svaku otkrivenu tabelu od slike dokumenta, omogućavajući fokusiranu analizu strukture tabele (redova i kolona). Prvo se vrši proširenje granica tabele, koje osigurava da će se cijela tabela uključiti u isječak (slika 18a). Ovaj korak je od suštinskog značaja da bi se izbjegao potencijalan gubitak vrijednih informacija koje možda neće biti precizno uhvaćene početnim graničnim okvirom detekcije tabele.

<sup>6</sup> Greške na slici 17a su ručno umetnute, kako bi se demonstriralo funkcionisanje navedenih funkcija



Slika 18. a) Detekcija tabele unutrašnjeg YOLO modela, b) Detekcija redova c) Detekcija kolona, d) Ćelije na izlazu unutrašnjeg YOLO modela

Nakon što se tabela proširi, vrši se njeno izolovanje kreiranjem nove slike (upotrebom *Python* biblioteke za obradu slike), na kojoj se samo ona nalazi. „Unutrašnji“ YOLO model zatim obrađuje svaku izolovanu tabelu posebno, kako bi otkrio redove i kolone (slika 18b i 18c), precizno definišući strukturu tabele na taj način (slika 18d).

## 5.2.6. Napredno rukovanje tabelama

Za tabele sa značajnim prazninama između redova ili kolona, koriste se funkcije vještačkog umetanja (*fill\_empty\_spaces\_in\_rows* i *fill\_empty\_spaces\_in\_columns*) da bi se stvorila definisanija mrežna struktura, pomažući u boljem poravnanju i organizaciji podataka unutar tabele, ali i ispravile potencijalne greške unutrašnjeg *YOLO* modela prilikom detekcije redova i kolona. Na slici 19 je prikazano da, i pored grešaka unutrašnjeg *YOLO* modela (izostavljanja jednog reda i jedne kolone), sistem na izlazu daje gotovo identičan rezultat kao i na slici 18d.

a) Carter's Credit Card Account. Read it and keep it.

<b>Nedostajuća kolona</b>	
Annual Percentage Rate (APR) for Purchases	<b>26.74%</b> This APR will vary with the market based on the Prime Rate.
How to Avoid Paying Interest	Your due date is at least 25 days after the close of each billing period. We will not charge you interest on purchases if you pay your entire balance by the due date each month. If you do not, you will not get a grace period on purchases again until you pay the entire balance by the due date for two billing periods in a row.
Minimum Interest Charge	If you are charged interest, the charge will be no less than \$2.
For Credit Card Tips from the Consumer Financial Protection Bureau	To learn more about factors to consider when applying for or using a credit card, visit the website of the Consumer Financial Protection Bureau at <a href="http://www.consumerfinance.gov/learnmore">http://www.consumerfinance.gov/learnmore</a> &
Annual Fee	None <b>Nedostajući red</b>
Penalty Fees • Late Payment • Returned Payment	Up to \$41 Up to \$41

b)

Annual Percentage Rate (APR) for Purchases	<b>26.74%</b> This APR will vary with the market based on the Prime Rate.
How to Avoid Paying Interest	Your due date is at least 25 days after the close of each billing period. We will not charge you interest on purchases if you pay your entire balance by the due date each month. If you do not, you will not get a grace period on purchases again until you pay the entire balance by the due date for two billing periods in a row.
Minimum Interest Charge	If you are charged interest, the charge will be no less than \$2.
For Credit Card Tips from the Consumer Financial Protection Bureau	To learn more about factors to consider when applying for or using a credit card, visit the website of the Consumer Financial Protection Bureau at <a href="http://www.consumerfinance.gov/learnmore">http://www.consumerfinance.gov/learnmore</a> &
Annual Fee	None
Penalty Fees • Late Payment • Returned Payment	Up to \$41 Up to \$41

How We Will Calculate Your Balance: We use a method called "daily balance (including current transactions)." Review your Agreement for more details.  
Billing Rights: Information on your rights to dispute transactions and how to exercise those rights is provided in your Agreement.  
Wisconsin Residents: If you are married, contact us upon receipt of this Agreement at the General phone number in the How to Contact Us section below and give us your spouse's name and address.

Slika 19. a) Prikaz grešaka pri detekciji redova i kolona, b) Čelije na izlazu unutrašnjeg *YOLO* modela<sup>7</sup>

## 5.2.7. Strukturiranje podataka i *JSON* izlaz

Poslednja faza uključuje sklapanje ekstrahovanih podataka u strukturirani *JSON* format (slika 20). Ovo obuhvata:

- **Objekte pasusa i tabela:** Svaki sadrži koordinate i pridružene *OCR* detekcije - riječi.

<sup>7</sup> Greške na slici 19a su ručno umetnute, kako bi se demonstriralo funkcionisanje navedenih funkcija

- **Detalje na nivou ćelije:** Unutar svake tabele, ćelije su definisane koordinatama i tekстом koji se u njima nalazi, izvedenim iz redova i kolona identifikovanih ranije. Prazne ćelije se uklanjaju da bi se struktura podataka pojednostavila.

```

{
  "top": 0.01694322982802987,
  "left": 0.0383126400411129,
  "right": 0.15396665409207344,
  "bottom": 0.031475411262363195,
  "categoryId": 0,
  "confidence": 0.8675650358200073,
  "words": [
    {
      "page_num": 1,
      "left": 0.04352941176470588,
      "top": 0.020454545454545454,
      "right": 0.08941176470588236,
      "bottom": 0.02909090909090909,
      "width": 0.04588235294117647,
      "height": 0.008636363636363636,
      "conf": 91,
      "text": "7/13/22,"
    },
    {
      "page_num": 1,
      "left": 0.09529411764705882,
      "top": 0.020454545454545454,
      "right": 0.12588235294117647,
      "bottom": 0.02772727272727273,
      "width": 0.03058823529411765,
      "height": 0.007272727272727273,
      "conf": 94,
      "text": "10:26"
    },
    {
      "page_num": 1,
      "left": 0.13058823529411764,
      "top": 0.020454545454545454,
      "right": 0.14941176470588236,
      "bottom": 0.02772727272727273,
      "width": 0.018823529411764704,
      "height": 0.007272727272727273,
      "conf": 96,
      "text": "AM"
    }
  ]
},

```

```

{
  "top": 0.08586275577545166,
  "left": 0.05821046233177185,
  "right": 0.9445224106311798,
  "bottom": 0.773523211479187,
  "categoryId": 1,
  "confidence": 0.9802215695381165,
  "words": [],
  "cells": [
    {
      "row": 0,
      "column": 0,
      "top": 0.1456471824645996,
      "left": 0.07488051945672314,
      "right": 0.4039389893061974,
      "bottom": 0.24161862806840376,
      "words": [
        {
          "page_num": 1,
          "left": 0.07705882352941176,
          "top": 0.1781818181818182,
          "right": 0.1411764705882353,
          "bottom": 0.18727272727272729,
          "width": 0.06411764705882353,
          "height": 0.00909090909090909,
          "conf": 96,
          "text": "Annual"
        }
      ],
    },
    {
      "page_num": 1,
      "left": 0.15411764705882353,
      "top": 0.1781818181818182,
      "right": 0.25823529411764706,
      "bottom": 0.19,
      "width": 0.10411764705882352,
      "height": 0.011818181818181818,
      "conf": 96,
      "text": "Percentage"
    }
  ],
},

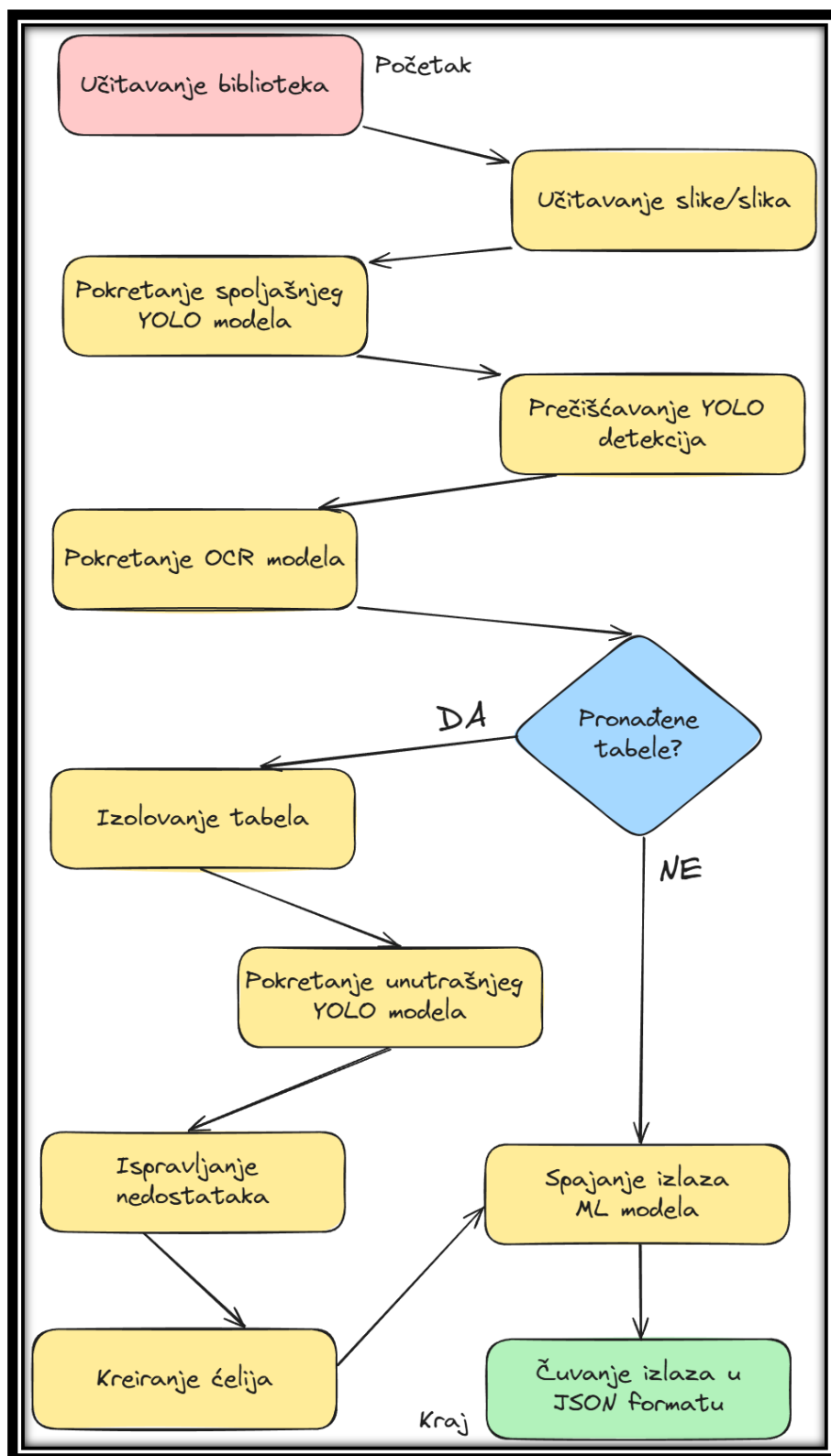
```

Slika 20. a) Prikaz paragrafa na izlazu sistema, b) Prikaz tabela na izlazu sistema

### 5.3. Šematski prikaz sistema

Implementirani sistem pokazuje snažnu sposobnost za obradu složenih izgleda dokumenata specifičnih za bankarski sektor, koristeći napredne *AI* modele za otkrivanje strukture i ekstrakciju teksta. Dizajn sistema, koji obrađuje kompletan proces od početka, odnosno slike, do kraja, odnosno strukturiranog *JSON* izlaza, bez potrebe za spoljnom intervencijom, osigurava da sistem može da rukuje različitim tipovima dokumenata sa visokom preciznošću i efikasnošću.

Sveobuhvatni blok dijagram sistema, koji ilustruje tok i interakciju između različitih komponenti, prikazan je na slici 21.



Slika 21. Šema cjelokupnog sistema predloženog u radu

## 6. EKSPERIMENTALNI REZULTATI I ANALIZA

U ovom poglavlju su izložene eksperimentalne evaluacije predloženog integrisanog sistema za digitalizaciju dokumenata koji se sastoji od *YOLOv8* modela detekcije objekata i *PyTesseract OCR* modela. Rezultati su analizirani uzimajući u obzir tačnost modela u detekciji struktura dokumenata i izdvajanja teksta, njegovu efikasnost u odnosu na tradicionalne ručne metode i njegov uticaj na operativnu efikasnost u bankarskim okruženjima. Oni se razmatraju u vezi sa postavljenim istraživačkim pitanjima i hipotezama.

### 6.1. Evaluacija performansi *YOLO* modela

Eksperimenti, čiji su rezultati prikazani u tabeli 1, nude sveobuhvatno poređenje različitih *YOLO* modela (iz porodica *YOLOv8* i *YOLOv5*) za dva skupa podataka: *DocLayout10K* (sacinjen od 10 000 slika) i *TableStruct1M* (kojeg čini milion slika). Prvih 16 eksperimenata je imalo za cilj otkrivanje dvije klase (tabele i pasuse) iz *DocLayout10K* skupa podataka, dok su poslednja dva eksperimenta rađena na *TableStruct1M*, složenijem skupu podataka sa tri klase (tabela, red i kolona). U svim ovim eksperimentima, procjenjivan je učinak na osnovu širokog spektra metrika, uključujući vrijeme obrade (i na grafičkoj kartici i na procesoru), veličine modela i nekoliko metrika tačnosti (*F1* skor, preciznost, odziv, *mAP*), o kojima je bilo riječi u poglavlju 4. Procjenjivan je ukupan učinak modela (prve četiri kolone donjeg dijela tabele 1), učinak prilikom prepoznavanja tabela (naredne četiri kolone donjeg dijela tabele 1) i učinak prilikom prepoznavanja pasusa (posladnje četiri kolone donjeg dijela tabele 1). Pošto skup podataka *TableStruct1M* ne uključuje pasuse, vrijednosti koje se odnose na metrike za klasu paragraf u rezultatima su zamijenjene prosječnim vrijednostima metrika za redove i kolone, što je označeno zvjezdicama u poslednjim kolonama i redovima tabele 1.

Hardversko okruženje za trening i testiranje uključivalo je dva *ASUS TUF NVIDIA GeForce RTX 3090 GPU*-a, sa po 24 GB memorije, koji su korišćeni i za obuku modela i za testiranje vremena obrade na *GPU*. Za testiranje brzine na procesoru, korišćen je zastarjeli *Intel Xeon* procesor sa jednim jezgrom i dvije niti, brzine takta od 2.3 GHz.

Ova konfiguracija je omogućila sveobuhvatna poređenja performansi modela na brzem hardveru (*GPU*, predzadnja kolona gornjeg dijela tabele 1) i brzine obrade sa ograničenim resursima (*CPU*, zadnja kolona gornjeg dijela tabele 1), naglašavajući razlike u brzini i efikasnosti između modela i skupova podataka.

Redni Broj	Model	Broj Epoha	Batch	Veličina Slike [pikseli]	lr0	lrf	Skup Podataka	Veličina [MB]	Trajanje Epoha [s]	Vrijeme Obrade na GPU [ms]	Vrijeme Obrade na CPU [ms]
1	yolov8n	171	32	960	0.3	0.005	DocLayout10K	6.2	82.4	2	484.3
2	yolov8s	200	32	960	0.3	0.005	DocLayout10K	22.5	116	3.5	1304.9
3	yolov8m	200	32	960	0.3	0.005	DocLayout10K	52	223.1	6.3	3041.4
4	yolov8l	104	32	960	0.3	0.005	DocLayout10K	87.6	328.9	11.3	5910.9
5	yolov8x	143	32	960	0.3	0.005	DocLayout10K	136.7	589.3	17.2	8902.6
6	yolov8n	200	32	640	0.3	0.005	DocLayout10K	6.2	51.1	1	193.9
7	yolov8s	200	32	640	0.3	0.005	DocLayout10K	22.5	64.2	1.6	557.1
8	yolov8m	115	32	640	0.3	0.005	DocLayout10K	52	112.1	3.4	1366.2
9	yolov8l	200	32	640	0.3	0.005	DocLayout10K	87.6	143.6	5.2	2535.4
10	yolov8x	118	32	640	0.3	0.005	DocLayout10K	136.7	204.5	7.5	4047.4
11	yolov8l	149	32	640	0.01	0.1	DocLayout10K	87.6	133.2	5.2	2729.6
12	yolov5n	200	32	640	0.01	0.01	DocLayout10K	3.8	40.59	5.9	139.2
13	yolov5s	200	32	640	0.01	0.01	DocLayout10K	14.3	49.14	6.3	375
14	yolov5m	176	32	640	0.01	0.01	DocLayout10K	42.1	87.4	8.4	920.1
15	yolov5l	150	32	640	0.01	0.01	DocLayout10K	92.7	119.2	11	1943
16	yolov5x	179	32	640	0.01	0.01	DocLayout10K	173	211.1	16.3	3455.5
17	yolov8s	20	64	640	0.01	0.01	TableStruct1M	22.5	3841.9	1.5	351.9
18	yolov8l	20	32	640	0.01	0.01	TableStruct1M	87.6	9211.7	3.6	1630.7

Redni Broj	F1 skor [Ukupna]	Preciznost [Ukupna]	Odziv [Ukupna]	mAP 0.5 [Ukupna]	mAP 0.5-0.95 [Ukupna]	Preciznost [Tabele]	Odziv [Tabele]	mAP 0.5 [Tabele]	mAP 0.5-0.95 [Tabele]	Preciznost [Pasusi]	Odziv [Pasusi]	mAP 0.5 [Pasusi]	mAP 0.5-0.95 [Pasusi]
1	96	96.1	95.9	98.1	89.2	96.6	97	98.9	94.9	95.7	94.8	97.4	83.5
2	96.2	96.1	96.3	98.3	90.5	96.5	97.4	99	96	95.8	95.1	97.6	85
3	96.2	96	96.4	98.3	90.9	96.5	97.3	99.1	96.3	95.6	95.5	97.6	85.5
4	96.5	96.6	96.4	98.4	90.8	97.1	97.6	99	96.2	96.1	95.2	97.8	85.3
5	96.5	96.6	96.4	98.4	90.9	97.3	97.5	99.1	96.3	95.9	95.2	97.7	85.5
6	95.7	95.7	95.7	98	88.9	96	97.6	99.1	95.8	95.3	93.8	96.9	82.1
7	95.9	96.1	95.8	98.2	89.7	96.8	97.3	99.1	96.1	95.5	94.4	97.2	83.4
8	96.2	96.6	95.8	98.2	89.9	97.1	97.5	99.1	96.2	96.1	94	97.4	83.6
9	96.2	96.6	95.9	98.2	90.6	97.2	97.4	99	96.5	96.1	94.5	97.4	84.7
10	96.3	96.5	96.1	98.2	90.2	97	97.9	99	96.4	96	94.3	97.5	84.1
11	96.4	96.6	96.2	98.3	90.3	97.2	98	99.1	96.5	96.1	94.4	97.4	84
12	95.4	96.2	94.6	97.5	84.3	96.9	97	99	91.6	95.5	92.2	95.9	77
13	96.1	96.5	95.8	97.9	87.3	97	97.6	99.1	93.9	96	94	96.8	80.7
14	96.1	96.3	96	98.1	88.7	96.6	97.3	99	94.7	96.1	94.6	97.1	82.7
15	96.4	96.9	96	98.2	89.1	97.5	97.2	99.1	95	96.3	94.8	97.4	83.3
16	96.6	96.7	96.5	98.2	89.5	97.2	97.9	99	95.1	96.2	95.1	97.3	83.9
17	99.2	99.7	98.7	99.3	96.2	1	99.9	99.5	99.5	99.6*	98.2*	99.2*	94.5*
18	99.3	99.8	98.8	99.3	96.4	1	1	99.5	99.5	99.7*	98.2*	99.2*	94.9*

Tabela 1. Rezultati treninga *YOLO* modela: u gornjem dijelu su prikazani upotrijebljeni parametri za treninge modela i vremena obrade, dok su u donjem dijelu mjere performansi.<sup>8</sup>

Rezultati otkrivaju konzistentan kompromis između veličine modela i performansi. Manji modeli, kao što su *YOLOv8n* i *YOLOv5n*, koji zauzimaju svega 6.2 megabajta (*MB*) i 3.8 *MB*, respektivno, pokazuju znatno brže vrijeme obrade, posebno na *CPU*-u. Na primjer, *YOLOv8n* je imao vrijeme obrade na *CPU* od 484.3ms i vrijeme obrade na *GPU*-a od samo 2 ms, kada se koristi veličina slike od 960 piksela (eksperiment

<sup>8</sup> Vrijednosti koje se odnose na klasu paragraf su, u poslednjim kolonama i redovima, zamijenjene prosječnim vrijednostima metrika za redove i kolone, označeno zvjezdicama.

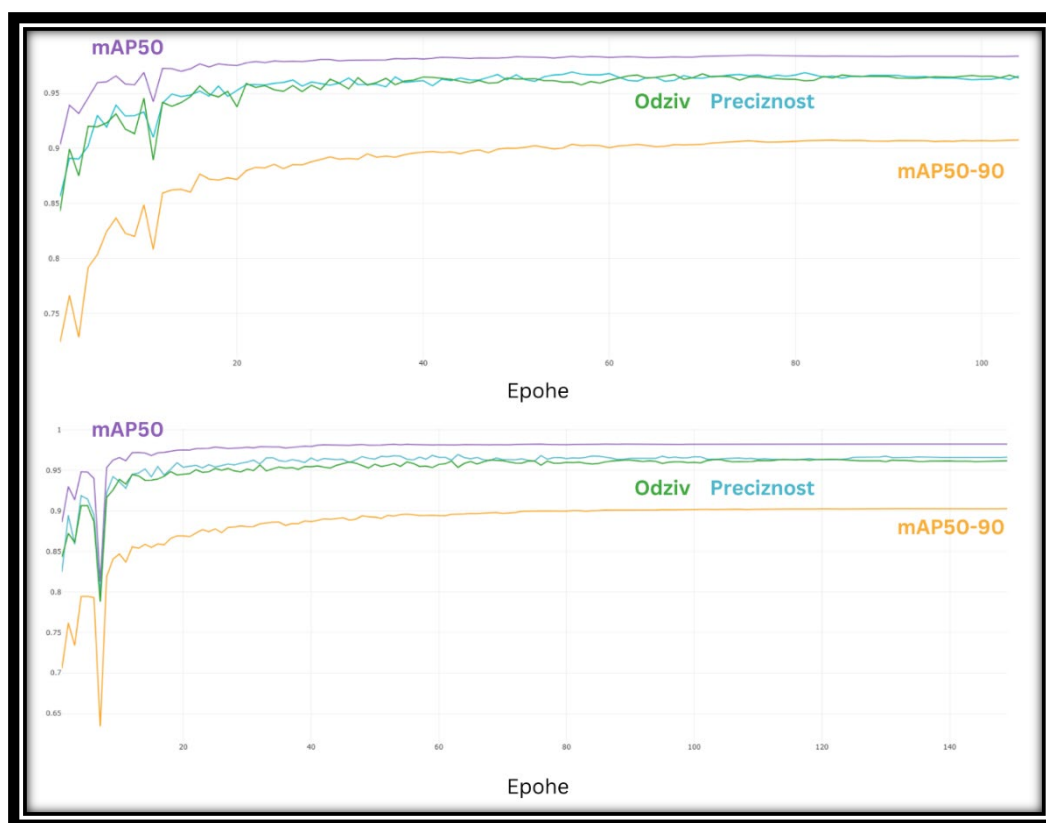
1, tabela 1). Uprkos brzini obrade, vrijednosti preciznosti, odziva i  $mAP$ -a za ove manje modele su nešto niže u poređenju sa većim varijantama. Na primjer, razmatrani *YOLOv8n* model je postigao preciznost u detekciji tabela od 96.6% i  $F1$  skor od 96%, što, iako je dobar rezultat, zaostaje za većim modelima kao što je *YOLOv8x*, koji su postigli vrijednosti preciznosti u detekciji tabela od 97.3% i  $F1$  skor od 96.6% (eksperiment 5). Modeli iz *YOLOv8x* familije su imali znatno veća vremena obrade (8902.6ms na *CPU*-u i 17.2ms na *GPU*-u).

Razlog višestruko sporije obrade na *CPU*-u nego na *GPU*-u je posljedica fundamentalnih razlika u njihovoj arhitekturi. *CPU* je dizajniran za serijsku obradu, sa nekoliko moćnih jezgara optimizovanih za obavljanje složenih zadataka, jedan po jedan. S druge strane, *GPU* ima stotine ili hiljade manjih jezgara specijalizovanih za paralelnu obradu, što omogućava istovremeno izvršavanje mnogih jednostavnih operacija. U obradi slike ili dubokom učenju, gdje se mogu obavljati brojne operacije (npr. množenje matrica) paralelno, *GPU* značajno ubrzava proces u poređenju sa *CPU*-om, koji ne može istovremeno izvršiti toliki broj operacija.

Iako sporiji, *YOLOv8x* modeli pružili su izuzetne rezultate, sa najvećim vrijednostima skoro svih metrika zapaženim u eksperimentu broj 5 iz tabele 1. Iz prvih 10 eksperimenata se može zaključiti da su modeli trenirani na većim slikama (prvih 5 eksperimenata) pokazali bolje rezultate od onih treniranih na manjim slikama (eksperimenti od 6–10). Razlog tome je to što veća rezolucija omogućava modelu da detektuje finije detalje i složenije strukture u slikama, što poboljšava preciznost i tačnost detekcije. Odabir početne ( $lr0$ ) i konačne ( $lrf$ ) stope učenja (poglavlje 4.2) takođe utiče na performanse modela. Niže vrijednosti početne i konačne stope učenja omogućavaju stabilnije i preciznije prilagođavanje težinskih koeficijenata (poglavlje 2.1) tokom treniranja, smanjujući rizik od preskakanja optimalnih rješenja i poboljšavajući konačne rezultate modela. Ta optimizacija je omogućila *YOLOv8l* modelu iz eksperimenta 11, da se primakne ili čak i prevaziđe rezultate većeg modela iz eksperimenata 5 i 10.

Na grafikonima na slici 22 su prikazane promjene odziva (zelenom bojom), preciznosti (plavom bojom) ali i  $mAP$  (ljubičastom bojom) i  $mAP$  0.5-0.95 (narandžastom bojom) metrika tokom treninga *YOLOv8l* modela. Slika 22a ilustruje promjene metrika modela sa višim stopama učenja (4. eksperiment iz tabele 1), dok su na slici 22b date metrike modela sa nižim stopama učenja (11. eksperiment iz tabele 1). Može se zaključiti

da niže vrijednosti početne i konačne stope učenja posmatranih *YOLO* modela zaista omogućavaju stabilnije i postepenije poboljšavanje rezultata, bez naglih oscilacija. U prvih 60 epoha (poglavlje 2.1) treninga modela, model sa većom početnom stopom učenja ima primjetno veće oscilacije od modela sa manjom početnom stopom učenja. Sve posmatrane metrike imaju blage fluktuacije na početku, ali se stabilizuju kako se iteracije povećavaju, što pokazuje da model postepeno uči bez značajnih skokova ili odstupanja od optimalnih vrijednosti, čemu doprinose niže vrijednosti *lr0* i *lrf*.



Slika 22. Grafici promjene parametara *YOLOv8* modela prilikom treninga, za različite vrijednosti parametara *lr0* i *lrf*. a) Gornji grafikon, metrike 4. eksperimenta iz tabele 1 b) Donji grafikon, metrike 11. eksperimenta iz tabele 1

Poređenje između modela *YOLOv8* i *YOLOv5* familija takođe pokazuje interesantne trendove. *YOLOv5n*, iako je manji po veličini (3.8 MB) i brži, posebno na CPU-u (vrijeme obrade od 139.2 ms), generalno je imao nešto niži učinak na metrikama tačnosti kao što je *mAP* 0.5-0.95, postigavši tačnost od 84.3% (eksperiment 12, tabela 1). Ovo je primjetno niže od vrijednosti *mAP*-a od 88.9%, koliko je dostigao *YOLOv8n* na istom skupu podataka *DocLayout10K* i na istoj veličini slika (od 640 piksela). Kako se

modeli povećavaju do *YOLOv5x* i *YOLOv8x*, i preciznost i odziv daju konstantno visoke rezultate, ali *YOLOv8* modeli imaju tendenciju da nadmašuju *YOLOv5* u pogledu *mAP* metrika, zbog arhitektonskih poboljšanja u novijoj verziji *YOLOv8*.

U posljednja dva eksperimenta, čiji su rezultati prikazani na dnu tabele 1 (pod rednim brojevima 17 i 18), upotreba skupa podataka *TableStructIM* je očekivano, uzimajući u obzir 100 puta veći skup podataka, dovela do značajnog porasta dužine trajanja epohe prilikom treninga. Ovdje su *YOLOv8* modeli pokazali izuzetne performanse, sa *YOLOv8s* koji je postigao skoro savršen rezultat F1 od 99.2% i preciznost od 99.7%, zajedno sa *mAP* 0.5 od 99.3%. *YOLOv8l* je pratio sličan trend, sa nešto većom preciznošću od 99.8% i *mAP* 0.5-0.95 od 96.4% (nasuprot 96.2%, koliko je manji model dostigao). Jasno je da su modeli bili izvrsni u otkrivanju tabela, redova i kolona, što ih čini veoma efikasnim za složene tabelarne dokumente.

Sumarno, rezultati pokazuju da veći modeli, iako zahtjevniji sa stanovišta računarskih resursa, nude superiornu tačnost i generalizaciju za otkrivanje jednostavnih i složenih struktura dokumenata. Manji modeli, s druge strane, predstavljaju isplativu opciju za brže aplikacije u realnom vremenu, gdje je brzina obrade ključna, ali su blagi kompromisi u preciznosti prihvatljivi. Zaključci prikupljeni iz ovih eksperimenata pružaju olakšicu pri izboru optimalne arhitekture modela na osnovu specifičnih zahtjeva za detekciju strukture dokumenata.

## 6.2. Preciznost ekstrakcije teksta

Za procjenu tačnosti *Pytesseract*-a korištene su tri ključne metrike: stopa grešaka riječi (eng. *Word Error Rate* - *WER*), stopa grešaka karaktera (eng. *Character Error Rate* - *CER*) i njihova srednja vrijednost. *WER* mjeri proporciju netačnih riječi u poređenju sa referentnim tekstom, uzimajući u obzir umetanja, brisanja i zamjene. *CER*, s druge strane, procjenjuje tačnost na nivou karaktera, nudeći dodatnu detaljnost upoređivanjem pojedinačnih grešaka u karakterima.

$$WER = \frac{Zamjene + Umetanja + Brisanja}{Stvaran\ Broj\ Riječi} \cdot 100\% \quad (9)$$

$$CER = \frac{Zamjene + Umetanja + Brisanja}{Stvaran\ Broj\ Karaktera} \cdot 100\% \quad (10)$$

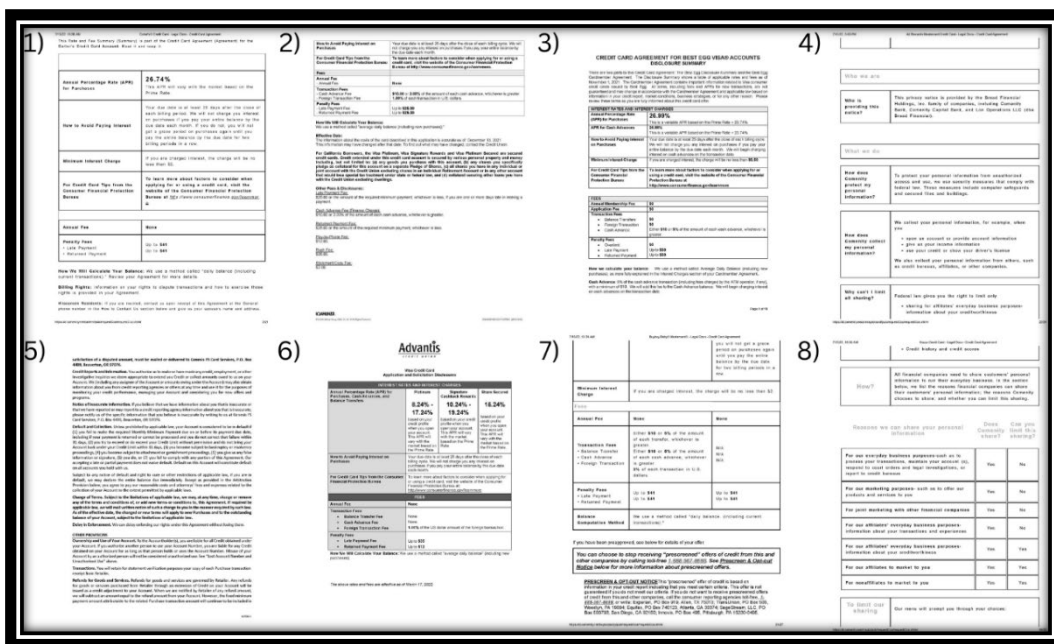
Prosjek *WER* (9) i *CER* (10) metrika se računa sa ciljem da se obezbijedi uravnotežena metrika koja odražava tačnost i na nivou riječi i na nivou karaktera, obezbjeđujući sveobuhvatnu procjenu performansi *PyTesseract OCR*-a.

Rezultati testiranja *PyTesseract*-a na dokumentima sa slike 23, dati su tabelom 2. Pokazuju prosječnu stopu grešaka riječi od 6.22% i stopu grešaka karaktera od 3.12%, sa kombinovanim prosjekom od 4.67%. Preciznost, dobijena kad se od 100% oduzme kombinovani prosjek, ostaje relativno visoka i iznosi 95.33%, iako postoje određene razlike između dokumenata. Na primjer, dokument 3 je imao najveću stopu grešaka, sa *WER*-om od 12.4% i *CER*-om od 10.77%. Razlog tome je zbijen tekst i mala visina redova unutar tabele, kao i različite veličine fonta, podebljani naslovi i regularni tekst, koji mogu dovesti do toga da *OCR* pogrešno protumači karaktere, posebno kada se prebacuje između ovih formata.

	WER [%]	CER [%]	(WER+CER)/2	Preciznost [%]	Vrijeme Obrade [s]
Dokument 1	2.93	0.91	1.92	98.08	5.16
Dokument 2	1.59	0.53	1.06	98.94	6.33
Dokument 3	12.4	10.77	11.59	88.41	6.58
Dokument 4	5.99	2.48	4.24	95.76	3.73
Dokument 5	0	0.03	0.02	99.98	9.7
Dokument 6	5.91	2.17	4.04	95.96	4.3
Dokument 7	7.94	2.31	5.13	94.87	5.74
Dokument 8	13.02	5.75	9.39	90.61	4.08
Srednje vrijednosti	6.22	3.12	4.67	95.33	5.7

Tabela 2. Rezultati testiranja *OCR*-a, na dokumentima sa slike 23

Ovi testovi su obavljani na istom procesoru koji je korišten i u prethodnom potpoglavlju (jednojezgarni višenitni *Xeon* procesor, brzine takta od 2.3GHz), pošto *Tesseract* nema mogućnost funkcionisanja na *GPU*-u. Ovo ograničenje u brzini obrade je primjetno kada se uporedi sa algoritmima za detekciju objekata kao što je *YOLO*, koji je pokazao mnogo brže performanse kada je testiran na *GPU*-u u prethodnom potpoglavlju rada. Brže *OCR* alternative koje podržavaju *GPU* obradu, poput *EasyOCR*-a [31] i *PaddleOCR*-a [32], bi stoga mogle biti efikasnije.



Slika 23. Stranice na kojima je testiran *PyTesseract*

### 6.3. Osvrt na istraživačka pitanja

Odgovori na istraživačka pitanja (poglavlje 1.1) slijede:

- **Tačnost detekcije rasporeda teksta:** Tačnost *YOLOv8* modela znatno premašuje onu njegovih prethodnika, potvrđujući efikasnost nedavnih adaptacija u rukovanju složenim strukturama dokumenata.
- **Poređenje veličina modela:** *YOLOv8x* model je pokazao najbolje performanse u pogledu preciznosti, ali i najgore u pogledu brzine obrade, što ukazuje na njegovu pogodnost za zahtjevnija okruženja, dok je najmanji model (*YOLOv8n*) pružio isplativo rješenje za zadatke gdje je neophodna maksimalna brzina obrade.
- **Uticaj sistema na tok digitalizacije dokumenata:** Uvođenje automatizovanog sistema značajno pojednostavljuje proces digitalizacije, preusmjeravajući ljudske resurse ka kritičnijim analitičkim zadacima, čime se povećava ukupna operativna efikasnost.

Eksperimentalna evaluacija potvrđuje efikasnost integrisanog *YOLOv8* i *PyTesseract* OCR sistema u transformaciji procesa digitalizacije dokumenata u bankama. Sistem ne samo da ispunjava, već i prevazilazi tradicionalne metrike učinka, postavljajući novi standard u automatizovanoj obradi dokumenata.

## 7. ZAKLJUČAK I BUDUĆI RAD

### 7.1. Rezime ključnih doprinosa i rezultata

Ovo istraživanje je uspješno demonstriralo integraciju modela detekcije objekata *YOLOv8* sa *PyTesseract OCR*-om kako bi se značajno poboljšala tačnost i efikasnost digitalizacije dokumenata u bankarskom sektoru. Pristup sa kombinacijom dva modela pokazao se efikasnim u preciznom otkrivanju strukture dokumenata i izdvajanju teksta, potvrđujući prvu hipotezu da takva integracija nadmašuje tradicionalne ručne metode i samostalne *OCR* sisteme i po preciznosti i brzini obrade.

Rezultati su pokazali da prilagođeni *YOLOv8* model nadmašuje prethodne verzije u otkrivanju lokacije teksta, rješavajući istraživačko pitanje o njegovoj komparativnoj tačnosti. Štaviše, procjena performansi različitih veličina modela (*nano, small, medium, large, extra large*) istakla je da veći modeli obezbjeđuju bolju preciznost, što je ključno za okruženja gdje je tačnost ključna, ali takođe povećava vremensku složenost, rezultirajući nešto sporijom obradom.

Konačno, uticaj sistema na operativnu efikasnost u bankama bio bi značajan, što potvrđuje treću hipotezu. Automatizacijom procesa digitalizacije, potreba za ručnim pregledom bi bila značajno smanjena, što bi dovelo do smanjenja vremena obrade dokumenata i potencijalne preraspodjele ljudskih resursa ka strateškim zadacima, a čime bi se povećala ukupna produktivnost.

### 7.2. Ograničenja predloženog sistema i budući rad

Iako je integrisani sistem pokazao obećavajuće rezultate, identifikovano je nekoliko ograničenja. Upotreba dva modela detekcije objekata, iako je efikasna, uvodi veće računске troškove, potencijalno ograničavajući primjenu u realnom vremenu. Pored toga, mnoge banke funkcionišu na starim sistemima koji su razvijeni decenijama unazad. Ovi sistemi su često izgrađeni na zastarjelim programskim jezicima i arhitekturama koje nisu kompatibilne sa savremenim *AI* rješenjima, kao što su *YOLOv8* i *PyTesseract*. Stariji sistemi možda ne podržavaju potrebne softverske zahtjeve, biblioteke ili operativne

sisteme, neophodne za pokretanje modela dubokog učenja. Zastarjelom hardveru, gotovo izvjesno, nedostaje računarska snaga, memorija i/ili kapaciteti skladištenja da bi se nosio sa zahtjevima intenzivnih procesa obrade *AI* modela. Pored toga, obezbjeđivanje da se sistem pridržava strogih bankarskih propisa koji se tiču privatnosti i bezbjednosti podataka predstavlja još jedan nivo složenosti.

Kako bi se nadovezalo na postojeće istraživanje i prevazišla identifikovana ograničenja, predlaže se nekoliko oblasti za budući rad:

- **Poboljšano ispravljanje grešaka i analiza sadržaja:** Integrisanje modela obrade prirodnog jezika za obradu ekstrahovanog teksta moglo bi efikasnije da ispravi *OCR* greške i analizira sadržaj dokumenta. Ovo bi pružilo dublji uvid i dalju automatizaciju u procesima donošenja odluka specifičnih za bankarstvo ili druge industrije.
- **Šira primjenljivost:** Proširivanje istraživanja na druge sektore kao što su zdravstvo i pravo bi moglo da potvrdi prilagodljivost i efikasnost sistema u različitim okruženjima dokumentacije.
- **Raznolikost tipova i jezika dokumenata:** Istraživanje performansi modela na širem spektru tipova dokumenata i jezika bi pomoglo da se minimizira pristrasnost skupa podataka i poboljša generalizacija sistema. Ovo bi moglo dovesti do robusnijih modela koji bi mogli da odgovore na različite globalne poslovne potrebe.

## 7.4. Zaključak

Integracija *YOLOv8* sa *PyTesseract OCR*-om predstavlja značajan napredak u tehnologiji digitalizacije dokumenata, posebno za bankarski sektor. Nalazi iz ove teze ne samo da potvrđuju efikasnost pristupa, već i navode potencijalne puteve za dalja poboljšanja. Uz kontinuirani razvoj i prilagođavanje, takvi sistemi obećavaju da će značajno smanjiti ručno opterećenje i pojednostaviti operacije obrade dokumenata, utirući put za širu primjenu inteligentnih rješenja za upravljanje dokumentima u različitim industrijama.

## RJEČNIK

- **Anotiranje** – U kontekstu detekcije objekata u kompjuterskoj viziji (poglavlje 2.2.1), anotiranje uključuje označavanje slika podacima koji definišu lokaciju i kategoriju objekata unutar tih slika - **anotacijama**. Ovaj proces obično uključuje crtanje graničnih okvira oko objekata i dodjeljivanje oznaka klasa svakom od njih, efektivno podučavajući model mašinskog učenja kako različiti objekti izgledaju i gdje se obično nalaze. Anotiranje je ključan korak u pripremi skupova podataka za treniranje modela detekcije objekata jer pruža bazične „istinite” podatke koje modeli koriste za učenje tokom procesa treniranja. Kvalitet i tačnost anotacija direktno utiču na sposobnost modela da pravilno identifikuje i locira objekte na novim, prethodno neviđenim slikama.
- **Baza podataka** (eng. *database*) – U računarstvu, baza podataka je strukturirana zbirka podataka koji se čuvaju elektronski. Dizajnirana je da efikasno upravlja, manipulira i organizuje podatke, ali i podržava procese koji zahtijevaju pronalaženje informacija. Bazama podataka obično upravlja sistem za upravljanje bazom podataka (eng. *Database Management System – DBMS*), koji olakšava interakciju sa korisnikom i drugim aplikacijama. Podaci u bazama podataka su organizovani u tabele, koje se sastoje od redova i kolona, što omogućava efikasno slanje upita i izvještaja. Savremene baze podataka mogu da rukuju različitim tipovima podataka, od jednostavnog teksta i brojeva do složenih podataka kao što su slike i transakcijske evidencije.
- **Bit** (eng. *Binary Digit* - binarna cifra) – Osnovna jedinica podataka. Može imati samo jednu od dvije vrijednosti: 0 ili 1. Ove vrijednosti se često tumače kao „ugašeno” ili „upaljeno”, respektivno, čineći bitove osnovnim građevinskim blokovima digitalnih podataka i generalno računarstva. Bitovi se koriste za predstavljanje i skladištenje informacija, pri čemu svaki bit odgovara binarnoj odluci ili stanju. Kolekcije bitova formiraju veće strukture podataka, kao što su bajtovi (obično 8 bitova), koji mogu predstavljati širok spektar podataka uključujući brojeve, slova i druge znakove. Više bajtova se organizuje u još veće jedinice, kao što su: **kilobajti** (eng. *kilobyte – KB*) - 1 kilobajt je jednak 1024 bajta, **megabajti** (eng. *mebabyte – MB*) - 1 megabajt iznosi 1024 kilobajta, **gigabajti** (eng. *gigabyte – GB*) - 1 gigabajt je jednak 1024 megabajta,

itd. Manipulacija i interpretacija ovih bitova su u osnovi svih zadataka digitalnog računarstva i obrade podataka.

- **Gradijent** – Ključni koncept za neuralne mreže koji predstavlja pravac i brzinu najbržeg povećanja funkcije. To je vektor koji pokazuje u pravcu najvećeg povećanja funkcije i čiji intezitet pokazuje koliko brzo se funkcija povećava u tom pravcu. Gradijent predstavlja generalizaciju koncepta izvoda funkcije po više promjenjivih. Ako je data funkcija  $f(x,y,z,...)$ , koja je funkcija više promjenjivih, njen gradijent se označava sa  $\nabla f$  ili  $grad(f)$  i definiše se kao:

$$\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \dots \right) \quad (11)$$

U (11),  $\frac{\partial f}{\partial x}$  predstavlja parcijalni izvod funkcije po promjenljivoj  $x$ ,  $\frac{\partial f}{\partial y}$  po  $y$ , itd. Ovi parcijalni izvodi predstavljaju brzinu promjene funkcije u odnosu na svaku promenljivu nezavisno. U optimizaciji, posebno u neuralnim mrežama, gradijent igra vitalnu ulogu, kroz **gradijentni spust** (eng. *gradient descent*). To je optimizacioni algoritam koji se koristi za minimiziranje funkcije iterativnim pomijeranjem u pravcu „najstrmijeg” spuštanja, što je definisano negativom gradijenta. U mašinskom učenju, analizirana funkcija je obično funkcija gubitka (poglavlje 2.1), a njeno minimiziranje dovodi do boljeg predviđanja modela.

- **Grafička procesorska jedinica** (eng. *Graphics Processing Unit – GPU*) – Specijalizovano električno kolo dizajnirano da brzo manipuliše i mijenja memoriju kako bi se ubrzalo kreiranje slika za uređaj za prikaz (monitor npr.). *GPU* ili grafičke kartice su posebno efikasne u rukovanju računarskom grafikom i zadacima poput obrade slike. Zbog svoje izričito paralelne strukture, *GPU*-ovi se takođe sve više koriste u računarstvu opštije namjene, kao što su mašinsko učenje i naučno računarstvo, gdje mogu značajno da ubrzaju proračune u poređenju sa tradicionalnim *CPU*-ovima (eng. *Central Processing Unit* - centralna procesorska jedinica).
- **Granični okvir** (eng. *bounding box*) – U oblasti detekcije objekata u kompjuterskoj viziji (poglavlje 2.2.1), Granični okvir je pravougaona oblast koja identifikuje i locira određeni objekat unutar slike. Koordinate *bounding box*-a definišu poziciju i dimenzije pravougaonika, koji se obično specificiraju  $x$  i  $y$  koordinatama gornjeg lijevog ugla (ili centra, u slučaju *YOLO*-a, poglavlje 2.2), zajedno sa širinom i visinom

okvira. Granični okviri se koriste kao standardni alat za vizuelno predstavljanje lokacije objekata unutar slike i ključni su i za obuku modela mašinskog učenja i za procjenu njihovog učinka. U praktičnim primjenama, tačnost sa kojom su ovi pravougaonici nacrtani oko predviđenih objekata značajno utiče na efikasnost algoritma detekcije objekata.

- **Heurističke tehnike** – Metode rješavanja problema koje se koriste za brzo pronalaženje praktičnih, dovoljno dobrih rješenja, kada je teško odrediti optimalno rješenje. Ove tehnike su zasnovane na iskustvu, praktičnim pravilima, „pametnim“ nagađanjima ili praktičnim shvatanjima. Heuristika je posebno korisna u složenim problemima sa velikim prostorima za pretragu ili mnogim mogućim izborima, gdje je pronalaženje tačnog rješenja nepraktično zbog vremenskih ili računskih ograničenja. Heurističke metode daju prednost brzini i jednostavnosti u odnosu na preciznost i često se koriste kada je približno rješenje prihvatljivo. Ovi pristupi su uobičajeni u oblastima kao što su računarske nauke za algoritme, poslovno donošenje odluka i kad god profesionalci treba da donesu brze i efikasne odluke u uslovima neizvjesnosti.
- **Interfejs** (eng. *interface*) – Interfejs u računarstvu je skup pravila koje različiti softverski programi ili sistemi prate da bi međusobno komunicirali. On navodi specifične metode ili radnje koje se mogu izvršiti, ali ne uključuje detalje o tome kako se te radnje sprovode. Ovo omogućava da različiti djelovi računarskog sistema rade zajedno bez potrebe da znaju detalje operacija jedni drugih, promovirajući fleksibilnost i modularnost u razvoju softvera. Jednostavnije rečeno, interfejs se ponaša kao ugovor, specificirajući koje su radnje moguće, bez navođenja kako se izvršavaju.
- **Interfejs za programiranje aplikacija** (eng. *Application Programming Interface - API*) – U računarstvu, interfejs za programiranje aplikacija je skup pravila i protokola za izgradnju i interakciju sa softverskim aplikacijama. API-ji djeluju kao posrednički sloj, omogućavajući različitim softverskim programima da međusobno komuniciraju. Oni definišu metode i strukture podataka koje programeri mogu da koriste za obavljanje određenih zadataka ili pristup određenim uslugama i podacima u aplikaciji ili operativnom sistemu. API-ji olakšavaju proces razvoja, obezbjeđujući standardne komande za izvođenje uobičajenih operacija, čime se izbjegava potreba za pisanjem koda od nule. Ovo promoviše efikasnost i kompatibilnost unutar softverskog

ekosistema, omogućavajući programerima da grade složene sisteme od modularnih komponenti koje se mogu ponovo koristiti.

- **Jezik za označavanje** (eng. *markup language*) – Sistem za dodavanje posebnih oznaka ili simbola dokumentu kako bi se definisala njegova struktura i izgled, čineći ga sintaksno drugačijim od teksta. Ove oznake upućuju softveru instrukcije kako da prikaže i obradi dokument, omogućavajući i ljudima čitljive i mašinski čitljive forme. Uobičajeni primjeri uključuju *HTML*, koji se prvenstveno koristi za kreiranje veb stranica, i *XML* (eng. *eXtensible Markup Language*), koji obezbjeđuje okvir za kodiranje dokumenata u formatu pristupačnom i ljudima i mašinama. *Markup* jezici su neophodni za razmjenu podataka, prikazivanje dokumenata i razvoj veb sajtova.
- **JSON** (eng. *JavaScript Object Notation*) – Jednostavan fajl format za razmjenu podataka koji je ljudima lak za čitanje i pisanje, ali i mašinama za raščlanjivanje i generisanje. *JSON* je zasnovan na tekstu i nezavisan od programskog jezika, ali koristi konvencije poznate programerima iz porodice jezika *C* (uključujući *C*, *C++*, *C#*), *Java*, *JavaScript*, *Perl*, *Python* i mnogih drugih. *JSON* strukturira podatke u formatu koji se sastoji od parova ključ-vrijednost i uređenih lista, što ga čini veoma pogodnim za razmjenu podataka između servera i veb aplikacija. *JSON* objekat je zatvoren u vitičastim zagradama: {}, i sadrži ključeve (ili imena) i vrijednosti, gdje su ključevi stringovi, a vrijednosti mogu biti stringovi, brojevi, nizovi, logički ili drugi *JSON* objekti. Ovo čini *JSON* svestranim alatom za serijalizaciju podataka i konfiguracione fajlove, koji se široko koriste u veb API-jima i kao sredstvo za konfigurisanje aplikacija. Slijedi primjer *JSON* fajla:

```
{
  "firstName": "Vuk",
  "lastName": "Slavic",
  "age": 24,
  "emails": [
    "vuk.slavic@gmail.com",
    "vukslavic@uhurasolutions.com"
  ],
  "address": {
    "street": "Ivana Vujosevica 8",
```

```
"city": "Podgorica",  
"state": "Crna Gora",  
"postalCode": "81000"  
}  
}
```

- **Klasa** (eng. *class*) – U kontekstu detekcije objekata, „klasa” se odnosi na kategoriju ili tip objekta koji je model za detekciju obučen da identifikuje i klasifikuje u okviru slika. Na primjer, u sistemu za praćenje saobraćaja, klase mogu uključivati različite objekte kao što su „auto”, „kamion”, „bicikl”, „pješak”, itd. Svaka klasa odgovara grupi objekata koji dijele zajedničke karakteristike, omogućavajući modelu za detekciju objekata da uči iz primjera tokom treninga i tačno prepozna i kategoriše ove objekte na novim slikama. Cilj modela je ne samo da otkrije prisustvo objekata na slici već i da odredi njihovu klasu na osnovu naučenih karakteristika.
- **Klasifikacija** – Proces kategorizacije podataka u unaprijed definisane klase ili grupe. Tip algoritma koji dodjeljuje oznaku kategorije (klase) ulaznim podacima na osnovu njegovih karakteristika i treninga koji je prošao naziva se **klasifikator**. Klasifikatori rade pod paradigmom nadgledanog učenja (poglavlje 2), gdje uče iz skupa podataka za trening koji uključuje i ulazne karakteristike i ispravne izlazne oznake. Jednom obučen, klasifikator koristi ove naučene informacije da predvidi kategoriju novih, do tad neviđenih podataka. Neuralne mreže su uobičajeni primjeri klasifikatora, čiji je cilj efikasna kategorizacija podataka u unaprijed definisane klase.
- **Lančano pravilo** (eng. *chain rule*) – Fundamentalni koncept u matematici koji se koristi za izračunavanje izvoda kompozitnih funkcija (kada se izlaz jedne funkcije koristi kao ulaz druge). Ako su date dvije funkcije,  $f$  i  $g$ , i formirana je kompozitna funkcija  $h(x) = f(g(x))$ , lančano pravilo nalaže da će izvod funkcije  $h$  po  $x$  biti proizvod izvoda funkcije  $f$  po  $g(x)$  i izvoda funkcije  $g$  po  $x$ . Matematički je dato formulom:

$$\frac{dh}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx} \quad (12)$$

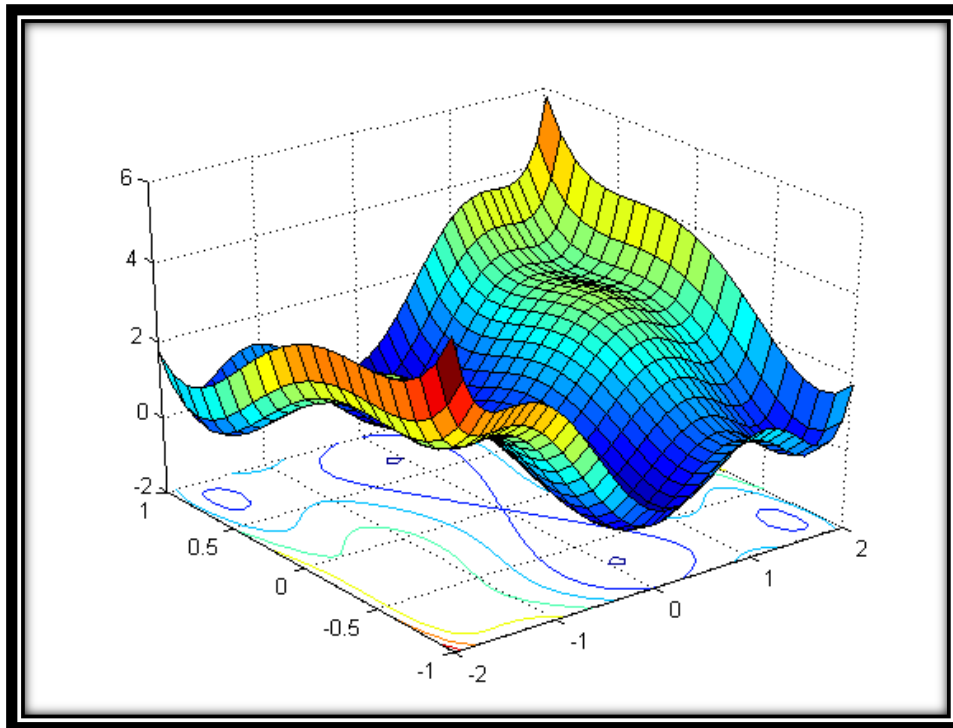
- **Lažno pozitivne detekcije** (eng. *False Positives*) – U kontekstu detekcije objekata u kompjuterskoj viziji, lažno pozitivne detekcije se odnose na slučaj kada sistem za

detekciju pogrešno identifikuje objekat koji nije prisutan na slici ili video snimku. Ova vrsta greške nastaje kada sistem greškom klasifikuje pozadinske ili nebitne elemente kao objekte od interesa. Na primjer, lažno pozitivan rezultat može uključivati otkrivanje pješaka na sceni gdje nema ljudi. Takve greške mogu značajno uticati na performanse i pouzdanost sistema za detekciju objekata, posebno u kritičnim aplikacijama kao što su autonomna vožnja ili bezbjednosni nadzor, gdje je preciznost u identifikaciji pravih objekata najvažnija. Napori da se smanje lažno pozitivni rezultati su ključni za povećanje ukupne efikasnosti i bezbjednosti sistema.

- **LSTM** (eng. *Long Short-Term Memory*) – Posebna vrsta rekurentne neuralne mreže koja je sposobna da nauči dugoročne zavisnosti u problemima predviđanja sekvenci. Ovo je posebno korisno u aplikacijama u kojima je ključno imati informacije iz ranijeg dijela sekvence da bi se tačno predvidjelo šta slijedi. Primjeri navedenog su prevođenje jezika ili prepoznavanje govora.
- **Mreža za ekstrakciju karakteristika** (eng. *feature extraction network*) – Dio modela mašinskog učenja, obično konvolucione neuralne mreže (poglavlje 2.1), koji obrađuje ulazne podatke (poput slika). Funkcija mu je da identifikuje i izvlači važne karakteristike koje su korisne za zadatke poput klasifikacije, detekcije ili prepoznavanja. Ova mreža se sastoji od slojeva filtera i slojeva sažimanja (poglavlje 2.1) koji progresivno razlažu ulazne podatke u oblik u kojem se najreprezentativnije karakteristike ističu i zadržavaju, dok se manje korisne informacije odbacuju. Izlaz *feature extractor* mreže je set mapa karakteristika (eng. *feature map*), koje predstavljaju različite aspekte ulaznih podataka. Svaka mapa je dvodimenzioni niz koji izdvaja specifične karakteristike sa ulaza, kao što su ivice, boje ili teksture. Njih zatim koriste drugi djelovi modela za obavljanje određenih zadataka poput identifikacije predmeta na slici ili razumijevanja sadržaja fotografije.
- **Omotač** (eng. *wrapper*) – Softver koji djeluje kao posrednički sloj, obuhvatajući i krijući složenost drugog softverskog modula ili API-ja. Omotači omogućavaju programerima da komuniciraju sa uslugom ili sistemom na jednostavniji ili kontekstualno relevantniji način, obezbjeđujući skup funkcija koje prevode komande prilagođene korisniku u složene komande koje zahtjeva sistem ili alat u pozadini. Ovo olakšava integraciju i korišćenje određenih funkcionalnosti bez potrebe za razumijevanjem detalja o tome kako pozadinski sistem funkcioniše.

- **Obrada prirodnog jezika** (eng. *Natural Language Processing – NLP*) – Grana vještačke inteligencije (poglavlje 2) koje se fokusira na interakciju između računara i ljudi upotrebom prirodnog jezika. Cilj *NLP*-a je da omogući kompjuterima da razumiju, tumače i generišu ljudski jezik na način koji je i smislen i koristan. *NLP* kombinuje računarsku lingvistiku - modelovanje ljudskog jezika zasnovano na pravilima - sa statističkim, mašinskim učenjem i modelima dubokog učenja (poglavlje 2). Ove tehnologije omogućavaju sistemima da obrađuju ljudski jezik u obliku tekstualnih ili glasovnih podataka i obavljaju zadatke kao što su prevođenje, analiza sentimenta i segmentacija tema. *NLP* je ključan u razvoju aplikacija koje zahtijevaju interakciju čovjeka i računara, kao što su četbotovi (eng. *chatbot*), glasovni *GPS* sistemi i bezkontaktne (eng. *hands-free*) kontrole, što ga čini vitalnom oblasti istraživanja vještačke inteligencije.
- **Otvoreni kod** (eng. *open-source*) – Vrsta softverske licence koja omogućava da izvorni kod bude slobodno dostupan za modifikaciju i distribuciju. Ovaj koncept nije primjenljiv samo na softver, već i na dizajn hardvera i drugih kreativnih djela. Softver otvorenog koda je razvijen na kolaborativan, javni način, omogućavajući svakome da pregleda, modifikuje, poboljša i dijeli kod.
- **PDF** (eng. *Portable Document Format*) – Svestrani fajl format koji je razvila *Adobe Systems* kompanija, 1993. godine. Dizajniran je da predstavi dokumente, uključujući formatiranje teksta i slike, na način da bude nezavisno od aplikativnog softvera, hardvera i operativnih sistema. *PDF* je sada otvoreni standard koji održava međunarodna organizacija za standardizaciju (eng. *ISO – International Organization for Standardization*). *PDF* fajlovi obuhvataju kompletan opis dokumenta, uključujući tekst, fontove, grafiku, slike i druge informacije potrebne za njegovo prikazivanje, fiksnog rasporeda. Ključna karakteristika *PDF* formata je njegova sposobnost da sačuva originalno formatiranje i izgled dokumenta, osiguravajući da se isti pojavljuje na bilo kom uređaju ili sa bilo kojim softverom koji podržava format. Ovo čini *PDF* popularnim izborom za pouzdano i dosljedno objavljivanje, dijeljenje i štampanje dokumenata. Pored toga, *PDF* podržava napredne funkcije kao što su šifrovanje, elektronski potpisi i interaktivni elementi poput obrazaca i multimedija, poboljšavajući njegovu korisnost za bezbjedno i interaktivno rukovanje dokumentima.

- **Piksel** (eng. *pixel*) – U digitalnim slikama, *pixel* (eng. *picture element* – element slike) je najmanja jedinica slike, predstavljena na digitalnom displeju. To je jedna tačka u slici (baziranoj na mreži ravnomjerno raspoređenih horizontalnih i vertikalnih linija) i osnovni je gradivni blok za konstruisanje slike i prenošenje vizuelnih informacija. Pikseli su obično organizovani u rešetkastu strukturu, a svaki piksel sadrži informacije o boji kako bi, kada se posmatraju u kombinaciji sa drugim pikselima, formirali kompletnu sliku. Pikseli su obično kvadratni i predstavljaju osnovne jedinice koje se koriste za mjerenje rezolucije digitalnih ekrana i slika - veće količine piksela koreliraju sa većom jasnoćom i detaljnošću slike. Boja svakog piksela je obično predstavljena kroz kombinacije intenziteta crvene, zelene i plave (eng. *RGB – Red Green Blue*) na slikama u boji. Jednostavnije rečeno, svaki piksel djeluje kao sićušna tačka u boji, a kolektivni raspored ovih tačaka stvara uočljivu sliku na digitalnim ekranima ili na digitalnim fotografijama.
- **Pipeline** – U kontekstu računarstva i obrade podataka, *pipeline* se odnosi na niz elemenata za obradu podataka povezanih u seriju, gdje je izlaz jednog elementa ulaz sledećeg. *Pipeline*-i se koriste za racionalizaciju i optimizaciju procesa, omogućavajući da se različiti koraci izvršavaju istovremeno ili u koordinisanoj sekvenci. Ova metoda je posebno efikasna u scenarijima gdje podatke treba obraditi u više faza, prije nego što dostignu svoj konačni oblik.
- **Površina gubitaka** (eng. *loss surface*) – Konceptualna vizualizacija koja se koristi u oblasti mašinskog učenja i dubokog učenja da bi se opisalo kako se gubitak (poglavlje 2.1), odnosno greška modela mijenja u odnosu na njegove parametre (obično težinske koeficijente i koeficijente pristrasnosti, poglavlje 2.1). Najčešće se predstavlja kao grafikon gdje horizontalne ose predstavljaju parametre modela, a vertikalna osa predstavlja gubitak za te parametre (slika 24). Ovaj grafik formira višedimenzionalnu površinu, poznatu kao površina gubitka. Svaka tačka na ovoj površini označava vrijednost gubitka za određeni skup parametara.

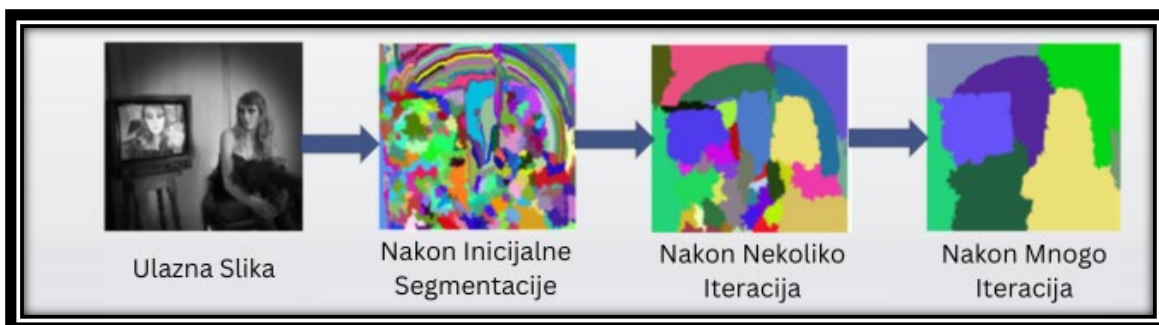


Slika 24. Vizuelni prikaz površine gubitaka

- **Prekomjerno prilagodavanje modela** (eng. *overfitting*) – Situacija u kojoj se model mašinskog učenja dobro ponaša na trening podacima, ali slabo na novim, odnosno podacima koje nije „vidio” prije.
- **Protokol** – Protokol u računarstvu predstavlja skup pravila koja definišu kako se podaci prenose između različitih uređaja, preko mreže. Obezbjeđuje da uređaji sa različitim hardverskim i softverskim konfiguracijama mogu efikasno da komuniciraju tako što definiše kako da formatiraju, šalju i primaju podatke. Protokoli su ključni za omogućavanje dosljedne i pouzdane razmjene podataka preko interneta i raznih drugih mreža, pokrivajući aspekte kao što su rukovanje greškama, bezbjednost podataka i sinhronizacija. Primjeri uključuju *Internet protokol* (eng. *Internet Protocol - IP*) i protokol za kontrolu prenosa (eng. *Transmission Control Protocol – TCP*), koji pomažu u usmjeravanju i isporuci podataka.
- **Python** – Programski jezik visokog nivoa, poznat po svojoj jasnoj sintaksi i čitljivosti, što olakšava brz razvoj. Dizajnirao ga je Guido van Rossum i objavio 1991. godine. *Python* podržava više paradigmi programiranja, uključujući proceduralno, objektno orijentisano i funkcionalno programiranje. Dinamički je tipiziran i sadrži *garbage-collector*, što pojednostavljuje mnoge zadatke programiranja u poređenju sa jezicima

nižeg nivoa. *Python*-ova obimna standardna biblioteka, u kombinaciji sa njegovom sposobnošću integracije sa drugim jezicima i tehnologijama, čini ga veoma raznovrsnim i popularnim u različitim domenima, kao što su web razvoj, analiza podataka, vještačka inteligencija, naučno računarstvo i još mnogo toga. Filozofija dizajna *Python*-a naglašava čitljivost i jednostavnost koda, što je dovelo do njegovog širokog usvajanja u akademskim krugovima i industriji, podstičući veliku i aktivnu zajednicu programera i bogat ekosistem biblioteka i sistema.

- **Regresija** – Statistička metoda koja se koristi za analizu podataka. Ona modeluje odnos između zavisne promjenljive i jedne ili više nezavisnih promjenljivih. Primarni cilj regresije je predviđanje vrijednosti zavisne promjenljive na osnovu poznatih vrijednosti nezavisnih promjenljivih. Ovo modelovanje pruža uvid u to kako nezavisne promjenljive utiču na zavisnu promenljivu i obično se koristi za prognoziranje, modelovanje vremenskih serija ili pronalaženje uzročno-posljedičnih veza između promjenljivih. Postoji nekoliko vrsta regresionih tehnika, uključujući linearnu regresiju, gdje se odnos modeluje kao prava linija, i nelinearnu regresiju, gdje se odnos modeluje prema složenijoj funkciji. Linearna regresija je najjednostavniji oblik, koji se obično koristi kada je odnos između promjenljivih približno linearan. Efikasnost regresionog modela se procjenjuje na osnovu toga koliko dobro predviđa zavisnu promjenljivu, koja se može procijeniti korišćenjem različitih statističkih metrika poput srednje kvadratne greške.
- **Selektivna pretraga** – Heuristička tehnika koja se koristi u detekciji objekata za efikasno pronalaženje potencijalnih objekata unutar slike. Funkcioniše tako što u početku razbija sliku na male, slične regione na osnovu boje i teksture (slika 25). Ovi regioni se zatim postepeno spajaju na osnovu njihove sličnosti, formirajući veće oblasti kandidata koje mogu sadržati objekte. Ovaj metod je dizajniran da generiše raznolik skup potencijalnih regiona, koji se zatim detaljnije analiziraju da bi se identifikovali stvarni objekti. Selektivna pretraga je korisna jer obezbeđuje da se svi potencijalni objekti razmotre, bez potrebe za ispitivanjem svakog mogućeg dijela slike, čineći ravnotežu između temeljnosti i računarske efikasnosti. Međutim, i dalje može biti relativno spora u poređenju sa integrisanijim pristupima kao što je *YOLO*, koji obrađuje cijelu sliku odjednom, bez prethodnog identifikovanja potencijalnih regiona koji sadrže objekat.



Slika 25. Prikaz funkcionisanja algoritma selektivne pretrage

- **Sidreni okvir** (eng. *anchor box*) – Vrsta graničnog okvira. Sidreni okviri su unaprijed postavljeni granični okviri, definisani u različitim skalama i razmjerama, kako bi pokrile različite veličine i oblike objekata koji se očekuju u skupu podataka. Model za detekciju objekata (poglavlje 2.2.2) koristi ove sidrene okvire kao polazne tačke i prilagođava ih tako da odgovaraju stvarnim objektima na slikama. Ovaj pristup pomaže modelu da brzo nauči da efikasnije predvidi lokaciju i veličinu objekata.
- **Sintaksa** – U programiranju, odnosi se na skup pravila koja definišu kombinacije simbola koji se smatraju ispravno strukturiranim programima u programskom jeziku. To uključuje ispravan raspored ključnih riječi i simbola za kreiranje valjanih programskih instrukcija. Sintaksa je analogna gramatici u prirodnim jezicima, određujući tačan redosljed i strukturu za pisanje koda koju kompajler ili tumač (tipovi softvera koji transformišu kod napisan u programskom jeziku u format koji računar može da izvrši) može da razumije. Svaki programski jezik ima svoju jedinstvenu sintaksu koju programeri moraju da prate da bi napisali efikasan kod bez grešaka.
- **Skup podataka** (eng. *dataset*) – U mašinskom učenju (poglavlje 2), skup podataka je kolekcija podataka specijalno pripremljenih za treniranje, validaciju ili testiranje algoritama. Tipično strukturirani u formatu pogodnom za računarsku obradu, skupovi podataka se sastoje od više različitih instanci, od kojih svaka sadrži skup promjenljivih ili atributa. Ovi atributi mogu biti numerički ili kategorijski i koriste se za predviđanje ishoda u zadacima nadgledanog učenja ili za uočavanje obrazaca u nenadgledanom učenju (poglavlje 2). Skupovi podataka igraju ključnu ulogu u razvoju i evaluaciji modela mašinskog učenja, jer obezbjeđuju sirovi materijal iz kojeg modeli uče obrasce i prave predviđanja. Kvalitet, raznovrsnost i relevantnost podataka unutar skupa podataka značajno utiču na performanse i tačnost rezultujućih

modela mašinskog učenja. **Trening skup podataka** je dio skupa podataka koji se koristi za obuku modela. Model analizira ovaj skup kako bi naučio obrasce i odnose između promjenljivih, prilagođavajući svoje parametre da bi minimizirao greške u predviđanju. Nakon inicijalnog treninga, **validacioni skup podataka** se koristi za podešavanje hiperparametara modela i za procjenu njegove sposobnosti da generalizuje na podacima koje nije prethodno „vidio”. Ovo pomaže u sprječavanju prekomjernog prilagođavanja modela na trening skup. **Testni skup podataka** je finalni skup podataka koji se koristi za nezavisnu evaluaciju performansi modela nakon što je trening završen. Testni skup omogućava objektivnu procjenu kako će model funkcionisati na potpuno novim podacima u realnim uslovima.

- **Softver** (eng. *software*) – Kolekcija instrukcija i podataka koji omogućavaju računaru da obavlja određene zadatke. Softver se razlikuje od hardvera, koji predstavlja fizičke komponente računara. Razvija se kroz proces programiranja i obuhvata širok spektar aplikacija, uključujući **operativne sisteme**, aplikativne programe i baze podataka. Operativni sistemi upravljaju hardverskim resursima računara i obezbjeđuju neophodno okruženje za rad aplikativnog softvera. Aplikativni softver je, s druge strane, dizajniran da pomogne korisnicima u obavljanju specifičnih zadataka, poput obrade teksta, pregledanja veba ili analize podataka. Softver se može široko kategorisati na sistemski softver, koji obezbjeđuje osnovnu funkcionalnost za rad računara, i aplikativni softver, koji radi „iznad” sistemskog softvera kako bi ispunio konkretne potrebe korisnika. Razvoj, primjena i održavanje softvera uključuje metodologije i prakse koje se neprestano razvijaju kako bi odgovorile novim tehnološkim dostignućima i zahtjevima korisnika.
- **String** – U računarstvu, *string* je tip podataka koji se koristi za predstavljanje teksta. Sastoji se od niza znakova (karaktera), obično uskladištenih na susjednim memorijskim lokacijama. *String*-ovi se uveliko koriste u programiranju za skladištenje i manipulaciju tekstualnim podacima kao što su imena, adrese ili bilo koja druga vrsta tekstualnih informacija. Svakom karakteru unutar *string*-a može se pristupiti pomoću indeksa, a različite operacije se mogu izvršiti nad *string*-ovima, kao što su spajanje, presijecanje i poređenje. *String*-ovi su nepromjenljivi u nekim programskim jezicima, što znači da kada se *string* kreira, znakovi u njemu se ne mogu

mijenjati bez kreiranja novog *string*-a. Ova karakteristika je od suštinskog značaja za obezbjeđivanje integriteta i bezbjednosti podataka unutar aplikacija.

- **Veb** (eng. *World Wide Web* – *www*) – Globalni sistem koji omogućava ljudima pristup i razmjenu informacija putem interneta. Otkako ga je Tim Berners-Lee predstavio 1989. godine, veb je transformisao način na koji komuniciramo, prikupljamo znanje i poslujemo, postajući suštinski dio svakodnevnog života. Veb se sastoji od različitih dokumenata i resursa koji se čuvaju na serverima. To može uključivati tekst, slike, video zapise i druge oblike medija. Korisnici pristupaju ovom sadržaju i komuniciraju sa njim pomoću softvera, pod imenom veb-pretraživač, kao što su *Google Chrome*, *Mozilla Firefox* ili *Safari*, obično na uređajima kao što su računari, tableti ili pametni telefoni. **Veb-sajtovi** su primarni način kojim se informacije predstavljaju na vebu. To je kolekcija povezanih **veb-stranica**, često organizovanih oko određene teme ili svrhe, kao što su vijesti, kupovina ili obrazovanje. Svaka veb-stranica u okviru veb-sajta je jedan dokument koji sadrži sadržaj i medije koje korisnici mogu da vide i sa kojima mogu da komuniciraju. Jedna od ključnih karakteristika veba je njegova lakoća navigacije između različitih informacija. Ovo je omogućeno vezama ugrađenim u veb stranice, omogućavajući korisnicima da prelaze sa jedne stranice na drugu. Ova međusobno povezana struktura formira ekspanzivnu mrežu lako dostupnih informacija.
- **Vremenska serija** – Statistička tehnika koja se koristi za analizu sekvenci tačaka podataka prikupljenih tokom vremena. Ovaj metod se koristi za ispitivanje trendova, ciklusa i sezonskih varijacija u podacima, omogućavajući predviđanje budućih vrijednosti na osnovu istorijskih obrazaca. Modeli vremenskih serija se obično koriste u ekonomiji za predviđanje cijena akcija i ekonomskih indikatora ali i u meteorologiji za predviđanje vremenskih obrazaca. Podaci u vremenskoj seriji su vremenski zavisni, što znači da su zapažanja u korelaciji sa vremenom.
- **YAML** (eng. *Yet Another Markup Language*) – Format serijalizacije podataka, čitljiv za ljude. Obično se koristi za konfiguracione fajlove i u aplikacijama u kojima se podaci čuvaju ili prenose. *YAML* je dizajniran da bude jasan i razumljiv, često služi kao čitljivija alternativa formatima kao što su *XML* ili *JSON*. U *YAML* fajlovima, podaci su strukturirani uvlačenjem (eng. *indentation*), koristeći razmake za predstavljanje hijerarhije. Ovo čini jednostavnim određivanje parova ključ-vrijednost

i izlistavanje podataka. *YAML* podržava različite tipove podataka uključujući nizove, brojeve, liste i ugnježdene strukture. Njegova jednostavnost i čitljivost čine ga široko korišćenim u podešavanjima koja zahtijevaju jednostavnu manipulaciju podacima i upravljanje konfiguracijom bez dodatnog teksta, koji nije neophodan. Primjer *YAML* strukture:

```
application:  
  name: MyApp  
  environment: development  
  
server:  
  host: localhost  
  port: 5000  
  
database:  
  type: sqlite  
  path: /path/to/myapp.db
```

## DODATAK

Kod u nastavku predstavlja realizaciju sistema korišćenog u ovom radu, kombinujući *OCR* i tehnike detekcije objekata za izdvajanje i strukturiranje podataka iz slika dokumenta.

```
## Potrebne biblioteke: ultralytics, pytesseract, tesseract-ocr

import pytesseract
from PIL import Image
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
import matplotlib.patches as patches

from ultralytics import YOLO
from PIL import Image
import os
import json
import numpy as np
import shutil
import time
import cv2
import requests

"""##Globalne varijable"""

TABLE_STRUCTURE_CLASSES = {
    2 : "row",
    1 : "column",
    0 : "table"
}

DOC_LAYOUT_CLASSES = {
    0 : "paragraph",
    1 : "table",
}

DOC_LAYOUT_FAKE_CLASSES = {
    7 : "table_cell"
}

LOWER_TABLE_DATA_THRESH = 0.1
HIGHER_TABLE_DATA_THRESH = 0.5
DOC_LAYOUT_THRESH = 0.3
DUPLICATES_AREA_THRESH = 0.4
CELL_SCORE = 1.0
EMPTY_ROW_THRESH = 1/5
EMPTY_COLUMN_THRESH = 1/5
IMAGE_WIDTH = 0
IMAGE_HEIGHT = 0

id_counter = 0

models = {
    'doc_layout': '/content/best.pt',
    'table_structure_model': '/content/ts_best.pt'
}

"""##Funkcije"""

def get_class_id(class_name, classes_dict):
```

```

for class_id, name in classes_dict.items():
    if name == class_name:
        return class_id
return None

def create_categories(categories_dict, include_cells=0):
    categories = []
    for category_id, category_name in categories_dict.items():
        category = {
            "id": category_id,
            "name": category_name
        }
        categories.append(category)

    if include_cells:
        table_cell_class = "table_cell"
        table_cell_id=get_class_id(table_cell_class,DOC_LAYOUT_FAKE_CLASSES)
        category = {
            "id": table_cell_id,
            "name": table_cell_class
        }
        categories.append(category)
    return categories, category_id + 1

def make_annotation_object(image_id, category_id, bbox, area, score):
    annotation = {
        "image_id": image_id,
        "category_id": category_id,
        "bbox": bbox,
        "area": area,
        "score": "{:.2f}".format(score)
    }
    return annotation

def make_annotation_object_cell(image_id, category_id, bbox, area, score,
row_id, column_id):
    annotation = {
        "image_id": image_id,
        "category_id": category_id,
        "bbox": bbox,
        "row_id": row_id,
        "column_id": column_id,
        "area": area,
        "score": "{:.2f}".format(score)
    }
    return annotation

def create_annotations(results, shape, image_id):

    annotations = []

    im_height, im_width = shape

    for result in results:

        x_center = result[1]*im_width
        y_center = result[2]*im_height
        width = result[3]*im_width
        height = result[4]*im_height
        area = int(width * height)
        class_id = result[0]
        score = result[5]

        x_min = int(x_center - (width / 2))
        y_min = int(y_center - (height / 2))

```

```

        width = int(width)
        height = int(height)

        bbox = [x_min, y_min, width, height]
        annotation = make_annotation_object(image_id, class_id, bbox, area,
score)
        annotations.append(annotation)
    return annotations

def create_cell_annotations(data, shape, image_id, class_id):

    image_height, image_width = shape
    annotations = []

    for items in data:
        for item in items:
            row_id = item['row']
            column_id = item['column']
            left = item['left']
            top = item['top']
            right = item['right']
            bottom = item['bottom']
            width = int((right-left)*image_width)
            height = int((bottom-top)*image_height)
            xmin = int(left*image_width)
            ymin = int(top*image_height)

            bbox = [xmin, ymin, width, height]
            area = width*height
            score = CELL_SCORE

            annotation = make_annotation_object_cell(image_id, class_id,
bbox, area, score, row_id, column_id)
            annotations.append(annotation)

    return annotations

def create_coco_annotations(results, shape, image_data, data):
    height, width = shape
    image_id, file_name = image_data
    images = []
    final_annotations = []

    image = {
        "id": image_id,
        "file_name": file_name,
        "height" : int(height),
        "width" : int(width)
    }
    images.append(image)

    categories_list, fake_id = create_categories(DOC_LAYOUT_CLASSES, 1)
    annotations_list = create_annotations(results, shape, image_id)
    cell_annotations_list = create_cell_annotations(data, shape, image_id,
fake_id)
    annotations_list.extend(cell_annotations_list)

    annotation_object = {
        "images" : images,
        "annotations" : annotations_list,
        "categories" : categories_list
    }

    final_annotations.append(annotation_object)
    return final_annotations

```

```

def get_class_id(class_name, classes_dict):
    for class_id, name in classes_dict.items():
        if name == class_name:
            return class_id
    return None

def adapt_coordinates(detections):

    adapted = []

    for i in detections:
        top, left, right, bottom = yolo2tess_geo(i[1:5])
        temp = {
            'top': top,
            'left': left,
            'right': right,
            'bottom': bottom,
            'categoryId': i[0],
            'confidence': i[5],
            'words': []
        }
        adapted.append(temp)

    return adapted

def create_cells(rows, columns, extended_table_coords, table_image_w,
table_image_h, verbose = False):

    cells = []

    for i in range(len(rows)):
        for j in range(len(columns)):

            x = columns[j][1]
            y = rows[i][2]
            width = columns[j][3]
            height = rows[i][4]
            cell_coords = [x, y, width, height]

            adapted_cell_coords = local2global_geo(cell_coords,
extended_table_coords, table_image_w, table_image_h)
            top, left, right, bottom = yolo2tess_geo(adapted_cell_coords)
            temp = {'row': i,
                    'column': j,
                    'top': top,
                    'left': left,
                    'right': right,
                    'bottom': bottom,
                    'words': []
                }
            cells.append(temp)

    if verbose == True:
        print("Number of created cells:", len(cells))

    return cells

def remove_duplicates(arr, verbose = False):

    counter = 0
    if TABLE_STRUCTURE_CLASSES[int(arr[0][0])] == 'row':
        mode = 2
    else:
        mode = 1

```

```

    for i in range(len(arr)-1, 0, -1):
        for j in range(i-1, -1, -1):
            if abs(float(arr[i][mode]) - float(arr[j][mode])) <
(float(arr[i][2+mode]) + float(arr[j][2+mode]))/4:
                counter += 1
                if arr[i][2+mode] > arr[j][2+mode]:
                    arr[j][mode] = arr[i][mode]
                    arr[j][2+mode] = arr[i][2+mode]
                    arr.pop(i)
                    break

    if verbose == True:
        print(f"Number of deleted {TABLE_STRUCTURE_CLASSES[int(arr[0][0])]}s:",
counter)

    return 1

def bubble_sort(arr, idx):

    if len(arr)<=1:
        return 1

    ind = 1
    while(ind):
        ind = 0
        for i in range(1, len(arr)):
            if arr[i][idx] < arr[i-1][idx]:
                ind = 1
                temp = arr[i]
                arr[i] = arr[i-1]
                arr[i-1] = temp

    return 1

def sort_rows_columns(arr, verbose=False):

    rows = []
    columns = []

    for i in arr:
        if TABLE_STRUCTURE_CLASSES[int(i[0])] == 'row':
            rows.append(i)
        elif TABLE_STRUCTURE_CLASSES[int(i[0])] == 'column':
            columns.append(i)

    if len(rows)>1:
        remove_duplicates(rows, verbose)

    if len(columns)>1:
        remove_duplicates(columns, verbose)

    bubble_sort(columns, 1)
    bubble_sort(rows, 2)

    return rows, columns

def local2global_geo(cells, table_coords, table_width, table_height):

    top_table, left_table, excess, excess = table_coords
    x_cell, y_cell, w_cell, h_cell = cells
    x = left_table + (x_cell * table_width / IMAGE_WIDTH)
    y = top_table + (y_cell * table_height / IMAGE_HEIGHT)
    width = w_cell * (table_width / IMAGE_WIDTH)
    height = h_cell * (table_height / IMAGE_HEIGHT)

```

```

output = [x,y,width,height]

return output

def local2global_geo_og(cells, table_coords, table_width, table_height):

    x_table, y_table, w_table, h_table = tess2yolo_geo(table_coords)
    x_cell, y_cell, w_cell, h_cell = cells
    x = x_table - w_table/2 + x_cell * (table_width / IMAGE_WIDTH)
    y = y_table - h_table/2 + y_cell * (table_height / IMAGE_HEIGHT)
    width = w_cell * (table_width / IMAGE_WIDTH)
    height = h_cell * (table_height / IMAGE_HEIGHT)
    output = [x,y,width,height]

    return output

def yolo2tess_geo(arr):

    x = float(arr[0])
    y = float(arr[1])
    width = float(arr[2])
    height = float(arr[3])
    bottom = y + height/2
    top = y - height/2
    left = x - width/2
    right = x + width/2
    output = [top, left, right, bottom]

    return output

def tess2yolo_geo(arr):

    top = float(arr[0])
    left = float(arr[1])
    right = float(arr[2])
    bottom = float(arr[3])
    x = (left + right) / 2
    y = (top + bottom) / 2
    width = right - left
    height = bottom - top
    output = [x, y, width, height]

    return output

def belongs(small, big):

    if type(small) == dict:
        x, y, excess, excess = tess2yolo_geo([small['top'], small['left'],
small['right'], small['bottom']])
    else:
        x = small[1]
        y = small[2]

    if type(big) == dict:
        x1 = big['left']
        x2 = big['right']
        y1 = big['top']
        y2 = big['bottom']
    else:
        y1, x1, x2, y2 = yolo2tess_geo(big[1:5])

    if x1 < x < x2:
        if y1 < y < y2:
            return True

```

```

return False

def closer_cell(word, cell1, cell2):

    x, y, excess, excess = tess2yolo_geo([word['top'], word['left'],
word['right'], word['bottom']])
    x1, y1, excess, excess = tess2yolo_geo([cell1['top'], cell1['left'],
cell1['right'], cell1['bottom']])
    x2, y2, excess, excess = tess2yolo_geo([cell2['top'], cell2['left'],
cell2['right'], cell2['bottom']])

    if 'row' in cell1 and cell1['row'] == cell2['row']:
        if abs(x-x1) < abs(x-x2):
            return 1
    elif 'row' in cell1 and cell1['column'] == cell2['column']:
        if abs(y-y1) < abs(y-y2):
            return 1
    else:
        if (abs(y-y1) + abs(x-x1)) < (abs(y-y2) + abs(x-x1)):
            return 1

    return 0

def assign_children(detections, dataOCR):

    leftovers = []
    for i in range(len(dataOCR)):
        ind = -1
        for j in range(len(detections)):
            if belongs(dataOCR[i], detections[j]):
                if ind == -1:
                    detections[j]['words'].append(dataOCR[i])
                    ind = j
                elif closer_cell(dataOCR[i], detections[j], detections[ind]):
                    detections[ind]['words'].pop(detections[ind]['words'].index(dataOC
R[i]))
                    ind = j
                    detections[j]['words'].append(dataOCR[i])

            if ind == -1:
                leftovers.append(dataOCR[i])

    return leftovers

def delete_empty(cells, verbose = False):

    if len(cells) == 0:
        print("No cells were created!")
        return 0

    counter = 0
    existing_rows = set()
    existing_columns = set()

    for i in range(len(cells)-1, -1, -1):
        if len(cells[i]['words']) == 0:
            cells.pop(i)
            counter += 1
        else:
            existing_rows.add(cells[i]['row'])
            existing_columns.add(cells[i]['column'])

    if len(cells) == 0:
        print("All the cells were deleted, since they were empty!")
        return 0

```

```

existing_rows = sorted(existing_rows)
existing_columns = sorted(existing_columns)

missing_rows = 0
missing_columns = 0
row_id_subtractor = dict()
column_id_subtractor = dict()

for i in range(existing_rows[-1] + 1):
    if i not in existing_rows:
        missing_rows += 1
        row_id_subtractor[i] = missing_rows

for i in range(existing_columns[-1] + 1):
    if i not in existing_columns:
        missing_columns += 1
        column_id_subtractor[i] = missing_columns

for i in cells:
    i['row'] -= row_id_subtractor[i['row']]
    i['column'] -= column_id_subtractor[i['column']]

if verbose == True:
    print("Successfully deleted", counter, "cells!")
    print("As a result", missing_rows, "rows were eliminated, as well as",
missing_columns, "columns")

return 1

def dimension_fixer(labels):
    labels = correct_table_height(labels)
    labels = adjust_column_height(labels)
    labels = adjust_row_width(labels)
    return labels

def correct_table_height(labels):

    highest_column_y = float('inf')
    lowest_column_y = float('-inf')

    for detection in labels:
        class_id, x, y, w, h, conf = detection
        if class_id == str(get_class_id("column", TABLE_STRUCTURE_CLASSES)):
            y = float(y)
            h = float(h)

            if y - h/2 < highest_column_y:
                highest_column_y = y - h/2

            if y + h/2 > lowest_column_y:
                lowest_column_y = y + h/2

    if highest_column_y < 0:
        highest_column_y = 0.0

    if lowest_column_y > 1:
        lowest_column_y = 1.0

    for detection in labels:
        class_id, x, y, w, h, conf = detection
        table_found = False
        if class_id == str(get_class_id("table", TABLE_STRUCTURE_CLASSES)):
            table_found = True
            y = float(y)

```

```

        h = float(h)
        if (y-h/2)>highest_column_y:
            h_old = h
            h = y+h/2-highest_column_y
            y = (y+h_old/2+highest_column_y)/2

        if (y+h/2)<lowest_column_y:
            h_old = h
            h = lowest_column_y-(y-h/2)
            y = (lowest_column_y+y-h_old/2)/2
        detection[2] = y
        detection[4] = h

        if not table_found:
            detection[1] = 0.5
            detection[2] = 0.5
            detection[3] = 0.95
            detection[4] = 0.95

        else:
            detection[1] = 0.5
            detection[2] = 0.5
            detection[3] = 0.95
            detection[4] = 0.95

    return labels

def adjust_column_height(labels):
    for detection in labels:
        class_id, x, y, w, h, conf = detection
        y_table=0.5
        h_table=0.98
        if class_id==str(get_class_id("table", TABLE_STRUCTURE_CLASSES)):
            y_table = float(y)
            h_table = float(h)

    for detection in labels:
        class_id, x, y, w, h, conf = detection
        if class_id==str(get_class_id("column", TABLE_STRUCTURE_CLASSES)):
            detection[2] = y_table
            detection[4] = h_table
    return labels

def adjust_row_width(labels):

    for detection in labels:
        class_id, x, y, w, h, conf = detection
        x_table=0.5
        w_table=0.98
        if class_id==str(get_class_id("table", TABLE_STRUCTURE_CLASSES)):
            x_table = float(x)
            w_table = float(w)

    for detection in labels:
        class_id, x, y, w, h, conf = detection

        if class_id==str(get_class_id("row",TABLE_STRUCTURE_CLASSES)):
            detection[1] = x_table
            detection[3] = w_table

    return labels

def find_minimum_row_height(rows):
    if len(rows) <= 1:
        if len(rows) == 1:

```

```

        return float(rows[0][2])
    else:
        return 0.0
y_coords = [float(row[2]) for row in rows]
row_heights=[y_coords[i+1]-y_coords[i] for i in range(len(y_coords)-1)]
min_row_height = min(row_heights)
return min_row_height

def find_minimum_column_width(columns):
    if len(columns) <= 1:
        if len(columns) == 1:
            return float(columns[0][1])
        else:
            return 0.0
    x_coords = [float(column[1]) for column in columns]
    column_widths = [x_coords[i+1] - x_coords[i] for i in
range(len(x_coords)-1)]
    min_column_width = min(column_widths)
    return min_column_width

def fill_empty_spaces_in_rows(table_labels, sorted_rows_labels,
row_threshold=EMPTY_ROW_THRESH):

    table_top = float(table_labels[2])-float(table_labels[4])/2
    table_bottom = float(table_labels[2])+float(table_labels[4])/2
    table_center_x = float(table_labels[1])
    table_width = float(table_labels[3])
    table_center_y = float(table_labels[2])
    table_height = float(table_labels[4])
    table_conf = float(table_labels[5])

    filled_rows_labels = []
    minimal_row_height = find_minimum_row_height(sorted_rows_labels)

    current_y = table_top

    if minimal_row_height==0.0:
        empty_row = [str(get_class_id("row", TABLE_STRUCTURE_CLASSES))] +
[table_center_x] + [table_center_y] + [table_width] + [table_height] +
[table_conf]
        filled_rows_labels.append(empty_row)
        return filled_rows_labels

    for row in sorted_rows_labels:
        row_top = float(row[2])-float(row[4])/2
        row_bottom = float(row[2])+float(row[4])/2
        if row_top>current_y:
            found_height = row_top-current_y
            if found_height>(row_threshold*minimal_row_height):
                empty_row = [str(get_class_id("row", TABLE_STRUCTURE_CLASSES))] +
+ [row[1]] + [current_y+(row_top-current_y)/2] + [row[3]] + [(row_top-
current_y)] + [row[5]]
                filled_rows_labels.append(empty_row)
                current_y = row_bottom
            if (-(current_y-table_bottom)) > (row_threshold*minimal_row_height):
                empty_row = [str(get_class_id("row", TABLE_STRUCTURE_CLASSES))] +
+ [row[1]] + [current_y+(table_bottom-current_y)/2] + [row[3]] +
+ [abs(current_y-table_bottom)] + [row[5]]
                filled_rows_labels.append(empty_row)

    return filled_rows_labels

def fill_empty_spaces_in_columns(table_labels, sorted_columns_labels,
column_threshold=EMPTY_COLUMN_THRESH):

```

```

table_left = float(table_labels[1])-float(table_labels[3])/2
table_right = float(table_labels[1])+float(table_labels[3])/2
table_center_x = float(table_labels[1])
table_width = float(table_labels[3])
table_center_y = float(table_labels[2])
table_height = float(table_labels[4])
table_conf = float(table_labels[5])

filled_columns_labels = []
minimal_column_width = find_minimum_column_width(sorted_columns_labels)

current_x = table_left

if minimal_column_width==0.0:
    empty_column = [str(get_class_id("column", TABLE_STRUCTURE_CLASSES))]
+ [table_center_x]+[table_center_y]+[table_width]+[table_height]+[table_conf]
    filled_columns_labels.append(empty_column)
    return filled_columns_labels

for column in sorted_columns_labels:
    column_left = float(column[1])-float(column[3])/2
    column_right = float(column[1])+float(column[3])/2
    if column_left>current_x:
        found_width = column_left-current_x
        if found_width>(column_threshold*minimal_column_width):
            empty_column = [str(get_class_id("column",
TABLE_STRUCTURE_CLASSES))] + [current_x+(column_left-current_x)/2] +
[column[2]] + [(column_left-current_x)] + [column[4]] + [column[5]]
            filled_columns_labels.append(empty_column)
            current_x = column_right
        if (-(current_x-table_right)) > (column_threshold*minimal_column_width):
            empty_column = [str(get_class_id("column", TABLE_STRUCTURE_CLASSES))]
+ [current_x+(table_right-current_x)/2] + [column[2]] + [abs(current_x-
table_right)] + [column[4]] + [column[5]]
            filled_columns_labels.append(empty_column)

    return filled_columns_labels

def calculate_row_height(row_data):
    return row_data['bottom'] - row_data['top']

def find_empty_space(rows, columns, labels):
    matching_labels = [label for label in labels if label[0] ==
str(get_class_id("table", TABLE_STRUCTURE_CLASSES))]

    if matching_labels:
        table_label = matching_labels[0]
    else:
        table_label = [str(get_class_id("table", TABLE_STRUCTURE_CLASSES)),
0.5, 0.5, 0.98, 0.98, 0.0]

    filled_rows_labels = fill_empty_spaces_in_rows(table_label, rows)
    filled_columns_labels = fill_empty_spaces_in_columns(table_label, columns)
    return filled_rows_labels, filled_columns_labels

def find_lower_confidence_rows(labels, missing_rows):
    new_rows_candidates = []
    for detection in missing_rows:
        class_id, x, y, w, h, conf = detection
        y = float(y)
        h = float(h)
        for detection_l in labels:
            class_id_l, x_l, y_l, w_l, h_l, conf_l = detection_l
            if class_id_l == str(get_class_id("row", TABLE_STRUCTURE_CLASSES)):
                y_l = float(y_l)

```

```

        if y_l > (y-h/2) and y_l < (y+h/2):
            new_rows_candidates.append(detection_l)
    return new_rows_candidates

def find_lower_confidence_columns(labels, missing_columns):
    new_columns_candidates = []
    for detection in missing_columns:
        class_id, x, y, w, h, conf = detection
        x = float(x)
        w = float(w)
        for detection_l in labels:
            class_id_l, x_l, y_l, w_l, h_l, conf_l = detection_l
            if class_id_l == str(get_class_id("column", TABLE_STRUCTURE_CLASSES)):
                x_l = float(x_l)
                if x_l > (x-w/2) and x_l < (x+w/2):
                    new_columns_candidates.append(detection_l)
    return new_columns_candidates

def perform_calculations_related_to_document(item_data, table_w, table_h,
doc_left, doc_right, doc_top, doc_bottom):

    class_id, x, y, w, h, conf = item_data
    left_item = (x-w/2) * table_w
    right_item = (x+w/2) * table_w
    top_item = (y-h/2) * table_h
    bottom_item = (y+h/2) * table_h

    left_item += doc_left
    right_item = doc_right-table_w+right_item
    top_item += doc_top
    bottom_item = doc_bottom-table_h+bottom_item

    return [left_item, top_item, right_item, bottom_item, class_id, conf]

def table_coordinates_related_to_document(rows, columns, doc_left, doc_right,
doc_top, doc_bottom, document_table_width, document_table_height):

    float_rows = [[float(item) for item in sublist] for sublist in rows]
    float_columns = [[float(item) for item in sublist] for sublist in columns]
    result_item_data = []
    items = float_rows + float_columns
    for item in items:
        result_item_data.append(perform_calculations_related_to_document(item, do
cument_table_width, document_table_height, doc_left, doc_right, doc_top, doc_bott
om))

    return result_item_data

def modify_calculations(top, left, right, bottom, table_w, table_h,
doc_left, doc_right, doc_top, doc_bottom):
    left_item = left*table_w
    right_item = right*table_w
    top_item = top*table_h
    bottom_item = bottom*table_h

    left_item += doc_left
    right_item = doc_right-table_w+right_item
    top_item += doc_top
    bottom_item = doc_bottom-table_h+bottom_item

    return [left_item/IMAGE_WIDTH, top_item/IMAGE_HEIGHT,
right_item/IMAGE_WIDTH, bottom_item/IMAGE_HEIGHT]

def process_json(input_json, doc_left, doc_right, doc_top, doc_bottom,
document_table_width, document_table_height):

```

```

output_json = []

for obj in input_json:
    top = obj['top']
    left = obj['left']
    right = obj['right']
    bottom = obj['bottom']

    new_left, new_top, new_right, new_bottom = modify_calculations(top,
left, right, bottom, document_table_width, document_table_height, doc_left,
doc_right, doc_top, doc_bottom)

    new_words = []
    for word in obj['words']:
        word_left = word['left']
        word_top = word['top']
        word_right = word['right']
        word_bottom = word['bottom']

        new_word_left, new_word_top, new_word_right, new_word_bottom =
modify_calculations(word_top, word_left, word_right, word_bottom,
document_table_width, document_table_height, doc_left, doc_right, doc_top, doc_bo
ttom)

        new_word = {
            'left': new_word_left,
            'top': new_word_top,
            'right': new_word_right,
            'bottom': new_word_bottom,
            'text': word['text']
        }
        new_words.append(new_word)

    new_obj = {
        'row': obj['row'],
        'column': obj['column'],
        'left': new_left,
        'top': new_top,
        'right': new_right,
        'bottom': new_bottom,
        'words': new_words
    }

    output_json.append(new_obj)
return output_json

def filter_data(json_data, left, right, top, bottom, table_w, table_h):
    filtered_data = []
    for page_item in json_data:
        filtered_page = {
            "pageNum": page_item["pageNum"],
            "pageHeight": table_h,
            "pageWidth": table_w,
            "data": []
        }
        for data_item in page_item['data']:
            item_left = data_item['left']
            item_right = data_item['right']
            item_top = data_item['top']
            item_bottom = data_item['bottom']

            if left <= item_left <= right and left <= item_right <= right
and top <= item_top <= bottom and top <= item_bottom <= bottom:
                filtered_page["data"].append(data_item)

```

```

        if filtered_page["data"]:
            filtered_data.append(filtered_page)

    return filtered_data

def are_duplicates(box1, box2, area_threshold):

    [top1, left1, right1, bottom1] = yolo2tess_geo(box1[1:5])
    [top2, left2, right2, bottom2] = yolo2tess_geo(box2[1:5])
    x_axis = min(right1, right2) - max(left1, left2)
    y_axis = min(bottom1, bottom2) - max(top1, top2)
    if x_axis > 0 and y_axis > 0:
        width, height = box1[3:5]
        area1 = width*height
        width, height = box2[3:5]
        area2 = width*height
        shared_area = x_axis*y_axis
        if shared_area/area1>area_threshold or shared_area/area2>area_threshold:
            if area1 > area2:
                return 1
            else:
                return 2

    return 0

def merge_yolo(box1, box2):

    [top1,left1,right1,bottom1] = yolo2tess_geo(box1)
    [top2,left2,right2,bottom2] = yolo2tess_geo(box2)
    top = top1 if top1 < top2 else top2
    bottom = bottom1 if bottom1 > bottom2 else bottom2
    left = left1 if left1 < left2 else left2
    right = right1 if right1 > right2 else right2

    return tess2yolo_geo([top,left,right,bottom])

def remove_duplicates_global(arr, classes_dict, area_threshold =
DUPLICATES_AREA_THRESH, verbose = False):
    counter = 0

    for key in classes_dict:
        ind = 1
        while ind == 1:
            ind = 0
            for_deleting = []
            for i in range(len(arr)-1, 0, -1):
                for j in range(i):
                    if arr[i][0] == key and arr[j][0] == key:
                        temp = are_duplicates(arr[i], arr[j], area_threshold)
                        if temp > 0:
                            counter += 1
                            if classes_dict[key]=="table" or classes_dict[key]=="column":
                                if temp == 2:
                                    for_deleting.append(i)
                                    break
                                elif temp == 1:
                                    for_deleting.append(j)
                            else:
                                merge_ind = 1
                                xywh = merge_yolo(arr[i][1:5], arr[j][1:5])
                                arr[j][1:5] = xywh
                                for_deleting.append(i)
                                break

            if len(for_deleting) > 0:

```

```

        ind = 1
        for_deleting = list(set(for_deleting))
        for_deleting.sort(reverse=True)
        for k in for_deleting:
            arr.pop(k)
    if verbose == True:
        print("Successfully deleted", counter, "duplicates.")
    return 1

def remove_rogue_paragraphs(arr, classes_dict, verbose = False):

    counter = 0
    for i in range(len(arr)-1, -1, -1):
        if classes_dict[arr[i][0]] == "paragraph":
            for j in range(len(arr)):
                if classes_dict[arr[j][0]] == "table" and belongs(arr[i], arr[j]):
                    arr.pop(i)
                    counter += 1
                    break

    if verbose == True:
        print("Number of deleted rogue paragraphs:", counter)
    return 1

def clean_table(im):
    gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    lsd = cv2.createLineSegmentDetector(0)

    lines = lsd.detect(gray)[0]

    if lines is not None:
        for element in lines:
            if (abs(int(element[0][0]) - int(element[0][2])) > 40 or
abs(int(element[0][1]) - int(element[0][3])) > 70):
                cv2.line(im, (int(element[0][0]), int(element[0][1])),
(int(element[0][2]), int(element[0][3])), (255, 255, 255), 3)

    return im

def crop_and_extend_images(image, lines):

    output_tables = []
    t_idx = 0
    original_height, original_width = image.shape[:2]

    for i in range(len(lines)):
        class_id, x_center, y_center, width, height, conf = lines[i]
        if DOC_LAYOUT_CLASSES[class_id]=='table':
            print("Table found!")
            t_idx += 1
            extended_width = width * 1.1
            extended_height = height * 1.1

            left = int((x_center - extended_width / 2) * original_width)
            right = int((x_center + extended_width / 2) * original_width)
            top = int((y_center - extended_height / 2) * original_height)
            bottom = int((y_center + extended_height / 2) * original_height)

            if left < 0:
                left = 0
            if right > original_width:
                right = original_width
            if top < 0:
                top = 0
            if bottom > original height:

```

```

        bottom = original_height
    if right < 0:
        right = 0
    if left > original_width:
        left = original_width
    if bottom < 0:
        bottom = 0
    if top > original_height:
        top = original_height

    extended_annotation = [top/original_height, left/original_width,
right/original_width, bottom/original_height]

    cropped_img = image[top:bottom, left:right]
    h, w = cropped_img.shape[:2]
    reshaped_image = cv2.resize(cropped_img, (w,h))
    reshaped_image = preprocess_image(reshaped_image)
    cv2.imwrite(f"./runs/out_new_{t_idx}.jpg", reshaped_image)

    output_tables.append([reshaped_image, extended_annotation, i])

return output_tables

def contains_color(image_np):
    if len(image_np.shape) == 3:
        for i in range(image_np.shape[0]):
            for j in range(image_np.shape[1]):
                r, g, b = image_np[i, j]
                if r != g or r != b or g != b:
                    return True
    else:
        return False

    return False

def preprocess_image(image):
    if contains_color(image):
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        grey_mask = cv2.inRange(gray_image, 100,140)
        large_grey_contours, _ = cv2.findContours(grey_mask,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        min_grey_area = 500
        for contour in large_grey_contours:
            if cv2.contourArea(contour) > min_grey_area:
                mask = np.zeros_like(gray_image)
                cv2.drawContours(mask, [contour], -1, 255, thickness=cv2.FILLED)

                black_text_mask = cv2.inRange(gray_image, 0, 50)
                black_text_within_grey=cv2.bitwise_and(black_text_mask, mask)

                gray_image[mask == 255] = 255
                gray_image[black_text_within_grey == 255] = 0
                dark_background_mask = cv2.inRange(gray_image, 0, 128)
                _, thresh = cv2.threshold(dark_background_mask, 128, 255,
cv2.THRESH_BINARY)

                contours, _ = cv2.findContours(thresh,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

                min_area = 500

                for contour in contours:

```

```

        if cv2.contourArea(contour) > min_area:
            mask = np.zeros_like(dark_background_mask)
            cv2.drawContours(mask, [contour], 0, 255, thickness=cv2.FILLED)

            if cv2.mean(dark_background_mask, mask=mask)[0] > 128:
                roi = dark_background_mask[mask == 255]
                inverted_roi = 255 - roi
                dark_background_mask[mask == 255] = inverted_roi

            cv2.drawContours(dark_background_mask, [contour], 0, 255, 1)

    image_arr = cv2.bitwise_not(dark_background_mask)
    image_out = Image.fromarray(image_arr)
else:
    image_out = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image_out = np.array(image_out)
    color_image = cv2.cvtColor(image_out, cv2.COLOR_GRAY2BGR)
    return color_image

def create_labels_str(predictions):

    formatted_objects = []

    for obj in predictions:
        class_id = str(int(obj[0]))
        x, y, w, h = obj[1:5]
        formatted_obj = [class_id, str(x), str(y), str(w), str(h)]
        formatted_objects.append(formatted_obj)

    return formatted_objects

def run_ocr(image_path):
    text_og = pytesseract.image_to_data(image_path, lang='eng')
    words = []
    lines = []

    lines = text_og.split('\n')
    lines = lines[1:]
    for line in lines:
        detections = line.split()
        if(len(detections) == 12):
            word = {
                'page_num': int(detections[1]),
                'left': int(detections[6])/IMAGE_WIDTH,
                'top': int(detections[7])/IMAGE_HEIGHT,
                'right': (int(detections[6])+int(detections[8]))/IMAGE_WIDTH,
                'bottom': (int(detections[7])+int(detections[9]))/IMAGE_HEIGHT,
                'width': int(detections[8])/IMAGE_WIDTH,
                'height': int(detections[9])/IMAGE_HEIGHT,
                'conf': int(detections[10]),
                'text': detections[11]
            }
            words.append(word)

    return words

def parse_predictions(image, flag):

    if flag == 1:
        model_path = models['doc_layout']
        confidence_threshold = DOC_LAYOUT_THRESH
    else:
        model_path = models['table_structure_model']
        confidence_threshold = LOWER_TABLE_DATA_THRESH

```

```

model = YOLO(model_path)
result = model(image)
names = model.names
boxes = []
for box in result[0].boxes:
    box_coords = box.xywhn.tolist()[0]
    class_id = int(box.cls.item())
    conf = box.conf.item()

    if conf > confidence_threshold:
        boxes.append([class_id] + box_coords + [conf])

if flag == 2:
    boxes2 = []
    for box in boxes:
        if box[-1] > HIGHER_TABLE_DATA_THRESH:
            boxes2.append(box)

    return [boxes2, boxes]

else:
    remove_duplicates_global(boxes, names,
area_threshold=DUPLICATES_AREA_THRESH, verbose=False)
    remove_rogue_paragraphs(boxes, names, verbose=False)
    remove_columns(boxes, names)
    return boxes

def table_structure_pipeline(labels, labels01, extended_table_coords,
table_image_w, table_image_h):

    table_top, table_left, table_right, table_bottom = extended_table_coords

    labels = dimension_fixer(labels)
    rows, columns = sort_rows_columns(labels)

    labels01 = dimension_fixer(labels01)

    missing_rows, missing_columns = find_empty_space(rows, columns, labels)
    lower_confidence_rows = find_lower_confidence_rows(labels, missing_rows)
    lower_confidence_columns = find_lower_confidence_columns(labels,
missing_columns)
    rows += lower_confidence_rows
    columns += lower_confidence_columns

    bubble_sort(rows, 2)
    bubble_sort(columns, 1)

    missing_rows, missing_columns=find_empty_space(rows, columns, labels01)
    rows += missing_rows
    columns += missing_columns

    bubble_sort(rows, 2)
    bubble_sort(columns, 1)

    new_cells = create_cells(rows, columns, extended_table_coords,
table_image_w, table_image_h)

    table_data = table_coordinates_related_to_document(rows, columns,
table_left, table_right, table_top, table_bottom, table_image_w,
table_image_h)
    adapted_json = new_cells
    return adapted_json, table_data, len(rows), len(columns)

def draw(cells, path):

```

```

all_files = os.listdir(path)
for i in all_files:
    if(i[-4:] == '.png'):
        ime = i

im = Image.open(path+ime)
width, height = im.size

fig, ax = plt.subplots()
ax.imshow(im)

for i in cells:
    rect = patches.Rectangle((i['left']*width, i['top']*height),
(i['right']-i['left'])*width, (i['bottom']-i['top'])*height, linewidth=1,
edgecolor='r', facecolor='none')
    ax.add_patch(rect)
    if 'words' in i:
        for j in i['words']:
            rect = patches.Rectangle((j['left']*width, j['bottom']*height),
(j['right']-j['left'])*width, (j['top']-j['bottom'])*height, linewidth=0.5,
edgecolor='y', facecolor='none')
            ax.add_patch(rect)

plt.show()

print('Success!')
return 1

"""##Main"""

img_folder = '/images'

table_data = []
table_annotations = []
coco_outputs = []
total_time = 0

for filename in os.listdir(img_folder):

    start = time.time()

    image_path=os.path.join(img_folder,filename)
    if os.path.isfile(image_path):
        print(filename)
    else:
        print("Error browsing folder!")

    im=cv2.imread(image_path)
    if im is None:
        print(f"Error: Unable to read image")
        exit()

    global IMAGE_WIDTH, IMAGE_HEIGHT
    IMAGE_HEIGHT, IMAGE_WIDTH = im.shape[:2]

    predictions_relative=parse_predictions(im, flag=1)

    ocr_output = run_ocr(image_path)
    if len(ocr_output) == 0:
        print("OCR output is empty!")
        exit()

    tables=crop_and_extend_images(im,predictions_relative)

```

```

detections_adapted=adapt_coordinates(predictions_relative)
assign_children(detections_adapted,ocr_output)

for table in tables:

    table_image, extended_table_coords, table_idx = table
    labels, labels01 = parse_predictions(table_image, flag=2)
    table_image_h, table_image_w=table_image.shape[:2]
    table_clean = clean_table(table_image)

    new_cells, table_annotation, rows_len, cols_len =
table_structure_pipeline(labels, labels01, extended_table_coords,
table_image_w, table_image_h)

    table_annotations += table_annotation
    assign_children(new_cells,detections_adapted[table_idx]["words"])

    delete_empty(new_cells, verbose=False)
    detections_adapted[table_idx]["cells"] = new_cells
    detections_adapted[table_idx]["words"] = []

end = time.time()
total_time += end - start

res = json.dumps({
    'detectedObjects' : coco_outputs,
    'time': total_time,
    'tableAnnotations' : table_annotations,
})

path = '/images/'
draw(detections_adapted, path)

path = '/images/'
draw(detections_adapted[2]['cells'], path)

for i in detections_adapted:
    if i["categoryId"] == 1:
        print(detections_adapted.index(i))

```

## LITERATURA

- [1] A. Gangal, P. Kumar, i S. Kumari, „Complete scanning application using OpenCV,” *arXiv preprint*, arXiv:2107.03700, 2021.
- [2] Z. E. Khattabi, Y. Tabii, i A. Benkaddour, „A new morphology-based method for text detection in image and video,” *Int. J. Comput. Appl.*, vol. 103, br. 13, str. 1-3, 2014.
- [3] T. Pratheeba, V. Kavitha, i S. R. Rajeswari, „Morphology-based text detection and extraction from complex video scenes,” *Int. J. Eng. Technol.*, vol. 2, br. 3, str. 200-206, 2010.
- [4] Y. Sun, X. Mao, S. Hong, W. Xu, i G. Gui, „Template matching-based method for intelligent invoice information identification,” *IEEE Access*, vol. 7, str. 28392-28401, 2019, doi: 10.1109/ACCESS.2019.2901943.
- [5] S. Bhowmik, R. Sarkar, M. Nasipuri, i D. Doermann, „Text and non-text separation in offline document images: A survey,” *Int. J. Document Anal. Recognit. (IJ DAR)*, vol. 21, br. 1-2, str. 1-20, jun 2018, doi: 10.1007/s10032-018-0296-z.
- [6] D. Nazir, K. A. Hashmi, A. Pagani, M. Liwicki, D. Stricker, i M. Z. Afzal, „HybridTabNet: Towards better table detection in scanned document images,” *Appl. Sci.*, vol. 11, br. 18, art. br. 8396, sep. 2021, doi: 10.3390/app11188396.
- [7] J. Redmon i A. Farhadi, „YOLOv3: An incremental improvement,” *arXiv preprint*, arXiv:1804.02767, 2018.
- [8] K. He, G. Gkioxari, P. Dollár i R. Girshick, „Mask R-CNN”, u *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, okt. 2017, str. 2961–2969. Dostupno na: [https://openaccess.thecvf.com/content/\\_ICCV\\_2017/html/He\\_Mask\\_RCNN\\_ICCV\\_2017\\_paper.html](https://openaccess.thecvf.com/content/_ICCV_2017/html/He_Mask_RCNN_ICCV_2017_paper.html). Pristupano: 6. maj 2022.
- [9] D. Prasad, A. Gadpal, K. Kapadni, M. Visave, i K. Sultanpure, „CascadeTabNet: An approach for end to end table detection and structure recognition from image-based documents,” u *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, jun. 2020, str. 572-573. Dostupno na: [https://openaccess.thecvf.com/content/\\_CVPRW\\_2020/html/w34/Prasad\\_Casc](https://openaccess.thecvf.com/content/_CVPRW_2020/html/w34/Prasad_Casc)

adeTabNet\\_An\\_Approach\\_for\\_End\\_to\\_End\\_Table\\_Detection\\_and\\_Structure\\_CVPRW\\_2020\\_paper.html. Pristupano: 6. maj 2022.

- [10] Á. Casado-García, C. Domínguez, J. Heras, E. Mata, i V. Pascual, „The benefits of close-domain fine-tuning for table detection in document images,” u *Document Analysis Systems*, Lecture Notes in Computer Science, vol. 12116. Springer, 2020, str. 199-215, doi: 10.1007/978-3-030-57058-3\_15.
- [11] K. A. Hashmi, M. Liwicki, D. Stricker, M. A. Afzal, M. A. Afzal, i M. Z. Afzal, „Current status and performance analysis of table recognition in document images with deep neural networks,” *IEEE Access*, vol. 9, str. 87663-87685, 2021, doi: 10.1109/ACCESS.2021.3087865.
- [12] M. S. Satav, T. Varade, D. Kothavale, S. Thombare, i P. Lokhande, „Data extraction from invoices using computer vision,” u *Proc. IEEE 15th Int. Conf. Ind. Inf. Syst. (ICIIS)*, nov. 2020, str. 316-320, doi: 10.1109/ICIIS51140.2020.9342722.
- [13] S. Shi, C. Cui, i Y. Xiao, „An invoice recognition system using deep learning,” u *Proc. Int. Conf. Intell. Comput., Autom. Syst. (ICICAS)*, dec. 2020, str. 416-423, doi: 10.1109/ICICAS51530.2020.00093.
- [14] Y. Wang, G. Gui, N. Zhao, Y. Yin, H. Huang, Y. Li, J. Wang, J. Yang, i H. Zhang, „Deep learning for optical character recognition and its application to VAT invoice recognition,” u *Communications, Signal Processing, and Systems*, Lecture Notes in Electrical Engineering, vol. 516, Springer, 2020, str. 87-95, doi: 10.1007/978-981-13-6508-9\_12.
- [15] M. Rusinol, T. Benkhelfallah, i V. P. d'Andecy, „Field extraction from administrative documents by incremental structural templates,” u *Proceedings of the 12th International Conference on Document Analysis and Recognition (ICDAR)*, avg. 2013, str. 1100-1104, doi: 10.1109/ICDAR.2013.223.
- [16] D. Baviskar, S. Ahirrao, i K. Kotecha, „Multi-layout unstructured invoice documents dataset: A dataset for template-free invoice processing and its evaluation using AI approaches,” u *IEEE Access*, vol. 9, str. 101494-101512, 2021, doi: 10.1109/ACCESS.2021.3096739.
- [17] W. Yu, N. Lu, X. Qi, P. Gong, i R. Xiao, „PICK: Processing key information extraction from documents using improved graph learning-convolutional

- networks,” u *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, 2021, str. 4363-4370, doi: 10.1109/ICPR48806.2021.9412927.
- [18] Z.-Q. Zhao, P. Zheng, S.-T. Xu, i X. Wu, „Object detection with deep learning: A review,” u *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, br. 11, str. 3212-3232, nov. 2019, doi: 10.1109/TNNLS.2018.2876865.
- [19] J. Redmon, S. Divvala, R. Girshick, i A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection,” u *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, str. 779-788, doi: 10.1109/CVPR.2016.91.
- [20] T.-Y. Lin et al. „Microsoft coco: Common objects in context,” u *European Conference on Computer Vision (ECCV)*, 2014, str. 740-755.
- [21] Ultralytics, „Ultralytics, Verzija 8.2.102,” GitHub. [Online]. Dostupno na: <https://github.com/ultralytics/ultralytics>. Pristupano: 27. septembar 2024.
- [22] Ultralytics, „YOLOv8 Improvements,” [Online]. Dostupno na: <https://yolov8.org/yolov8-improvements/>. Pristupano: 18. oktobar 2024.
- [23] C.-Y. Wang et al., „CSPNet: A new backbone that can enhance learning capability of CNN,” u *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, str. 390-391.
- [24] S. Liu, L. Qi, H. Qin, J. Shi, i J. Jia, „Path aggregation network for instance segmentation,” u *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, str. 8759-8768.
- [25] Ultralytics, „What is new in YOLOv8,” [Online]. Dostupno na: <https://yolov8.org/what-is-new-in-yolov8/>. Pristupano: 18. oktobar 2024.
- [26] S. Smith, „pytesseract, Verzija 0.3.13,” GitHub repozitorijum, 2024. [Online]. Dostupno na: <https://github.com/madmaze/pytesseract>. Pristupano 20. oktobar 2024.
- [27] R. Smith, „Tesseract OCR, Verzija 5.4.1,” GitHub repozitorijum, 2024. [Online]. Dostupno na: <https://github.com/tesseract-ocr/tesseract>. Pristupano 20. oktobar 2024.
- [28] DocuClipper, „Data Entry Statistics and Trends in 2023,” [Online]. Dostupno na: <https://www.docuclipper.com/blog/data-entry-statistics/>. Pristupano: 18. oktobar 2024.

- [29] B. Smock, „PubTables-1M Dataset,” Hugging Face. [Online]. Dostupno na: <https://huggingface.co/datasets/bsmock/pubtables-1m>. Pristupano: 20. oktobar 2024.
- [30] Autogyro, „YOLO-V8,” GitHub repozitorijum, [Online]. Dostupno na: <https://github.com/autogyro/yolo-v8>. Pristupano: 20. oktobar 2024.
- [31] JaidedAI, „EasyOCR, Verzija 1.7.2,” GitHub repozitorijum, 2024. [Online]. Dostupno na: <https://github.com/JaidedAI/EasyOCR>. Pristupano 20. oktobar 2024.
- [32] PaddlePaddle, „PaddleOCR, Verzija 2.8.1,” GitHub repozitorijum, 2024. [Online]. Dostupno na: <https://github.com/PaddlePaddle/PaddleOCR>. Pristupano 20. oktobar 2024.