Detekcija zauzetosti parkirališta na temelju računalnog vida

Koch, Brando

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

DETEKCIJA ZAUZETOSTI PARKIRALIŠTA NA TEMELJU RAČUNALNOG VIDA

Završni rad

Brando Koch

Osijek, 2020.

SADRŽAJ

1. UVOD

Parkirališta predstavljaju važan dio svakog prometnog središta, a sastoje se od individualnih parkirnih mjesta. Parkirna mjesta su površine odgovarajuće veličine, najčešće vidljivo označene, za parkiranje vozila. Ona zahtijevaju alokaciju velikog dijela gradskih površina uz velike cijene izgradnje i kao takva trebaju imati što veću i efikasniju iskoristivost. Problem nastaje kada su parkirališta gotovo potpuno popunjena što za slučaj pronalaska slobodnog parkirnog mjesta stvara gužve i troši novac i vrijeme jer vozač ne zna ima li uopće slobodnih parkirnih mjesta i ako ih ima, gdje se točno nalaze. Rješenja navedenog problema se oslanjaju na tehnike provođenja analitike na parkiralištu poput praćenja zauzetosti individualnih parkirnih mjesta. Ova rješenja se nazivaju PGI (engl. *Parking Guidance and Information*) sustavi [1]. Postoji 4 vrste PGI sustava koje se razlikuju po metodama detekcije: 1) sustavi na temelju senzora na ulazu i izlazu na parkiralište, 2) žičani sustavi na temelju senzora, 3) bežični elektromagnetski sustavi na temelju senzora i 4) sustavi na temelju video kamera. Sustavi na temelju senzora na ulazu i izlazu na parkiralište imaju manu da mogu pratiti samo apsolutnu zauzetost ograđenog parkirališta bez ikakvih informacija o zauzetosti individualnih parkirnih mjesta. Sustavi navedeni pod 2) i 3) omogućuju vrlo pouzdano praćenje zauzetosti individualnih parkirnih mjesta, ali zato zahtijevaju postavljanje senzora na svako parkirno mjesto što je skupo i vremenski zahtjevno za postavljanje. Moderni PGI sustavi iskorištavaju nepomične video kamere usmjerene prema parkiralištu kao senzore čije se informacije uobičajeno predaju nekom algoritmu za obradu i analizu slike koji ima sposobnost klasifikacije zauzetosti pojedinog parkirnog mjesta. Algoritmi ovog tipa mogu biti vrlo različiti sve od jednostavnijih metoda detekcije promjene boje na prostoru naznačenom kao parkirno mjesto do najmodernijih (engl. *state of the art*) algoritama razvijenih u području računalnog vida zasnovanih na strojnom učenju. Ovakvi sustavi mogu drastično uštedjeti i vrijeme i novac za rješenje spomenutog problema jer jedna kamera može promatrati daleko više parkirnih mjesta, a čak može imati i sporednu ulogu nadzora okolnog prostora za kojeg su namijenjeni.

Računalni vid je grana umjetne inteligencije koja predstavlja skup metoda i algoritama za rješenje raznih problema detekcije, klasifikacije, segmentacije pa i same tekstualne interpretacije slikovnih tipova podataka. Velik napredak u području računalnog vida nastao je primjenom posebnih vrsta umjetnih neuronskih mreža koje se nazivaju konvolucijske neuronske mreže (engl. *Convolutional Neural Networks* - CNNs) [2]. Računalni vid temeljen na konvolucijskim neuronskim mrežama ima

prednost s obzirom na druge pristupe jer je često robusniji na promjene u svjetlini i djelomične zaklonjenosti objekata na slici zbog prisutnosti drugih objekata između kamere i samog parkirnog mjesta od interesa. Praćenje zauzetosti parkirnih mjesta pomoću video kamera može se najčešće razložiti na dva zasebna problema. Prvi problem je segmentacija parkirnih mjesta odnosno označavanje gdje se na snimkama nalaze parkirna mjesta u obliku graničnih pravokutnika (engl. *bounding box*) koji obuhvaćaju 2D područje individualnih parkirnih mjesta na snimkama. Ovaj problem se u većini slučajeva ne rješava algoritamski nego se ručno označe granični pravokutnici za parkiralište na koji se primjenjuje algoritam. Drugi problem je klasifikacija svakog parkirnog mjesta kao zauzeto ili kao slobodno analizirajući sadržaj graničnih pravokutnika spomenutih u prvom problemu. Najveći uspjeh u klasifikaciji postižu moderni algoritmi zasnovani na dubokim neuronskim mrežama.

U okviru ovog završnog rada predstavljen je inovativan način praćenja zauzetosti parkirališta jer ne zahtijeva ručno označavanje parkirnih mjesta prije klasifikacije niti računalno zahtjevne algoritme što znači da bi se mogao iskoristiti kao distribuirano IoT rješenje ovom problemu. To se postiže koristeći detektor objekata, vlastiti *PSD* (engl. *Parking Spot Detector*) algoritam te binarni klasifikator zauzetosti. U drugom poglavlju su predstavljena moderna postojeća rješenja za praćenje zauzetosti parkirališta na temelju video signala. Predloženi algoritam za praćenje zauzetosti parkirališta opisan je u trećem poglavlju uz teorijske podloge i implementaciju. U četvrtom poglavlju je evaluirana njegova efikasnost na dva javno dostupna skupa podataka i na tri skupa podataka sastavljena od video signala s javnih IP kamera.

1.1. Zadatak završnog rada

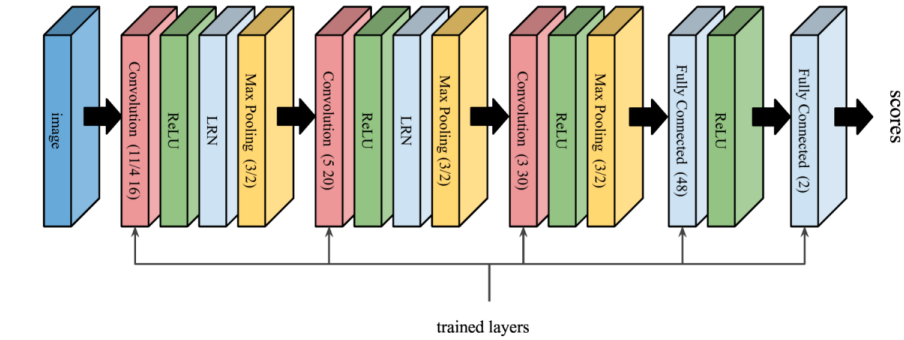
Zadatak završnog rada je rješavanje problema detekcije zauzetosti parkirnih mjesta. Pri tome je potrebno koristiti snimke parkirališta kao ulaz za algoritme u okviru računalnog vida temeljene na strojnom učenju te evaluirati rješenje na javno dostupnim skupovima podataka.

2. POSTOJEĆI ALGORITMI ZA PRAĆENJE ZAUZETOSTI PARKIRALIŠTA NA TEMELJU VIDEO SIGNALA KAMERE

Ideja za praćenje zauzetosti parkirališta nije nova te se pojavljuje u mnogim oblicima opisanih u uvodu. U ovom pregledu ograničilo se samo na pristupe koji koriste duboko učenje i koji su nedavno objavljeni u znanstvenim radovima. Oni predstavljaju trenutni doseg tehnologije uz najbolje rezultate.

2.1. Primjena dubokog učenja za decentralizirano praćenje zauzetosti parkirališta

Rad [3] je primijenio duboke konvolucijske neuronske mreže u svrhu praćenja stanja parkirnih mjesta. Granični pravokutnici parkirnih mjesta su na korištenim snimkama prethodno ručno označeni. Korišteni pristup je binarna klasifikacija zauzetosti nad dijelovima slike koji odgovaraju graničnim pravokutnicima. Korišteni klasifikator je jednostavnija verzija *AlexNet* [4] arhitekture zvana *mAlexnet* ili *mini Alexnet* (vidi sliku 2.1.) jer je originalna arhitektura optimizirana za puno kompliciranije zadatke u usporedbi s binarnom klasifikacijom što bi smanjilo brzinu izvođenja jer je rad primjenjivao algoritam na ugradbenoj računalnoj platformi.



Sl. 2.1. mAlexNet arhitektura.

Autori su *mAlexNet* trenirali na vlastitom skupu podataka (engl. *dataset*) kojeg su kreirali označavajući slike s 9 kamera usmjerenih na parkiralište CNR kampusa (engl. *campus of the National Research Council*) u Pisi s frekvencijom uzorkovanja od pola sata. Označavanje slika u ovom slučaju je značilo ručno odrediti granične pravokutnike parkirnih mjesta te zauzetost tih parkirnih mjesta na svakoj slici. Navedeni skup podataka je slobodno dostupan pod imenom *CNRPark-EXT*.

Specifikacije skupa podataka *CNRPark-EXT* su sljedeće:

- 1 parkiralište
- 9 kamera
- 3 vrste vremenskih uvjeta: sunčano, oblačno, kišovito
- 4287 slika cijelog parkirališta
- 144.965 označenih slika individualnih parkirnih mjesta

Na slici 2.2. prikazan je CNR parkirališni prostor iz zraka označen crvenim četverokutom i pozicije kamera sa žutim trapezima. Zahtjevnost zadatka zbog neravnomjernog osvjetljenja i zaklonjenosti drugim objektima vidljiv je na slici 2.3. gdje je u prvom redu izdvojeno četiri primjera zauzetih, a u drugom četiri slobodnih parkirnih mjesta.

Sl. 2.2. Prikaz CNR parkirališta i lokacija video kamera iz zraka.

Sl. 2.3. Primjeri isječaka individualnih parkirnih mjesta CNR parkirališta.

Trenirani *mAlexnet* klasifikator su autori implementirali na uređaju *Raspberry Pi 2* uz mogućnost klasificiranja i slanja informacija o 50 parkirnih mjesta web serveru za oko 15 sekundi. Trenirani model postiže točnost preko 90%.

2.2. Detekcija zauzetosti parkirališta primjenom dubokog učenja u stvarnom vremenu

Rad [5] također koristi CNNs, ali u kombinaciji s metodom potpornih vektora (engl. *Support Vector Machine* - SVM) [6] za binarnu klasifikaciju parkirnih mjesta. SVM omogućava klasifikaciju tako što transformira problem koji nije linearno razdvojiv u linearno razdvojiv pomoću projekcije podataka u prostor karakteristika i pronalaska optimalne zasebne hiperplohe. Klasifikator su autori trenirali na javno dostupnom *PKLot* skupu podataka [7].

Specifikacije skupa podataka *PKLot* su sljedeće:

- 2 parkirališta
- 3 kamere (*Parking1a, Parking1b, Parking2*)
- 3 vrste vremenskih uvjeta: sunčano, oblačno, kišovito
- 12.417 slika cijelih parkirališta
- 695.899 označenih slika individualnih parkirnih mjesta
- 48.53% slika sa zauzetim parkirnim mjestima
- 51.46% slika sa slobodnim parkirnim mjestima

Na slici 2.4. mogu se vidjeti perspektive sve 3 kamere.

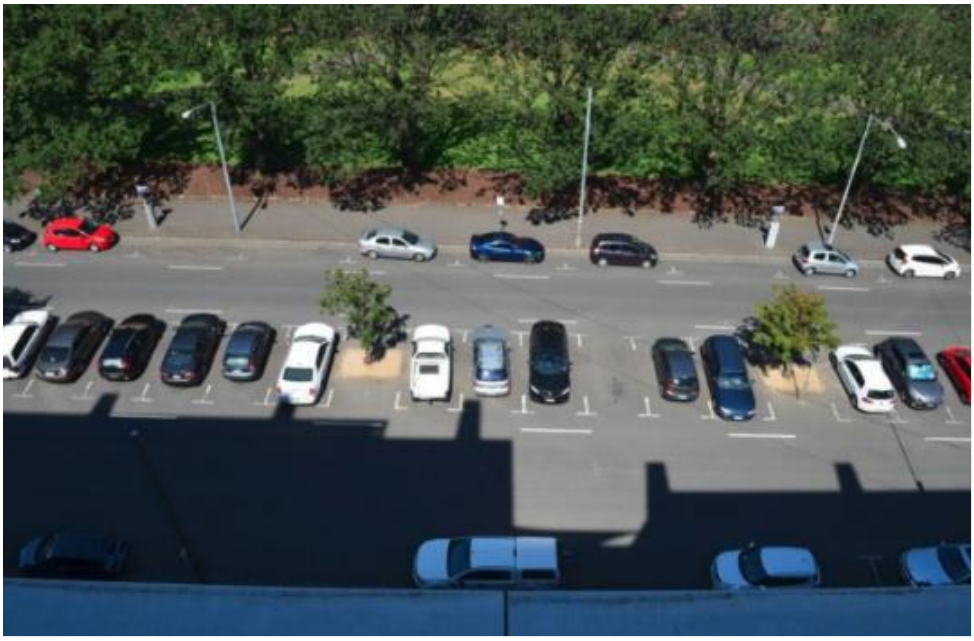
Sl. 2.4. Perspektive video kamera iz *PKLot* skupa podataka.

Uspješnost klasifikatora autori su testirali na njihovom privatnom skupu podataka zvanom *Berry Street*. *Berry Street* skup podataka je sastavljen od slika parkirališta sa snimki DSLR kamera s krova zgrade *Faculty of Business and Economics Building, the University of Melbourne*. Autori su prije treninga klasifikatora ručno označili pozicije parkirnih mjesta na *Berry Street* skupu podataka dok je na *PKLot* skupu podataka to već učinjeno.

Specifikacije skupa podataka *Berry Street su sljedeće*:

- 1 parkiralište
- nepoznati broj kamera
- 3 vrste vremenskih uvjeta: sunčano, oblačno, kišovito
- 810 slika cijelih parkirališta
- 30 parkirnih mjesta
- 24300 označenih slika individualnih parkirnih mjesta
- 48.54% slika sa zauzetim parkirnim mjestima
- 51.46% slika sa slobodnim parkirnim mjestima

Na slici 2.5. možemo vidjeti perspektivu jedne od *Berry Street* kamera.



Sl. 2.5. Perspektiva jedne od *Berry Street* kamera.

Autori prijavljuju točnosti od 99.7% i 96.7% na skupu podataka *PKLot* odnosno *Berry Street.*

Prethodno opisane metode postižu relativno visoku točnost u klasifikaciji zauzetosti parkirnih mjesta no i dalje zahtijevaju ručno označavanje graničnih pravokutnika. Algoritam koji bi u potpunosti uklonio potrebu za označavanjem bio bi puno lakše primjenjiv na bilo koje parkiralište.

3. PREDLOŽENI ALGORITAM ZA PRAĆENJE ZAUZETOSTI PARKIRALIŠTA NA TEMELJU RAČUNALNOG VIDA

3.1 Strojno učenje i detekcija objekata

Pristup korišten u radu temelji se na strojnom učenju, specifično na konvolucijskim neuronskim mrežama. Konvolucijske neuronske mreže su danas osnovni alat područja modernog računalnog vida. Za potpuno razumijevanje predloženog algoritma potrebno je poznavanje strojnog učenja i alata koji su korišteni. Najbitniji dijelovi su ukratko opisani u nastavku teksta.

3.1.1 Strojno učenje

Strojno učenje je polje usko povezano s umjetnom inteligencijom i računarstvom koje se bavi izgradnjom algoritama koji, da bi bili korisni, se oslanjaju na skupove podataka iz kojih kažemo da ti algoritmi uče. Spomenuti algoritmi su matematički modeli formirani odnosno trenirani nad tim skupovima podataka. Pretpostavka strojnog učenja je da se dani podaci mogu matematički i statistički opisati odgovarajućim modelima. Strojno učenje se dalje dijeli u kategorije od kojih je za izradu predloženog algoritma važno nadzirano učenje. Nadzirano učenje je tip učenje kod kojeg je skup podataka kolekcija ulaza x_i uparenih s izlazima y_i kao prema formuli (3-1) .

$$\{(x_i, y_i)\}_{i=1}^{N} \tag{3-1}$$

Cilj algoritama nadziranog učenja je odrediti nepoznatu funkcionalnu ovisnost između ulaznih i izlaznih veličina danog skupa podataka. Također, cilj je da model ima i sposobnost generalizacije naučenog odnosno da model daje zadovoljavajuće rezultate i za ulaze s kojima se nije susreo tijekom njegova treniranja. Ako su izlazi diskretne veličine tada se taj postupak naziva klasifikacija, a ako su kontinuirane regresija. Diskretne veličine kod klasifikacije se nazivaju klase.

S obzirom na složenost modela razlikujemo primjenu plitkog (engl. *shallow learning*) i dubokog učenja (engl. *deep learning*). Duboko učenje karakteriziraju neuronske mreže sa značajno više slojeva i posljedično velikim brojem parametara pa tako zahtijevaju i relativno velike skupove podataka za učenje.

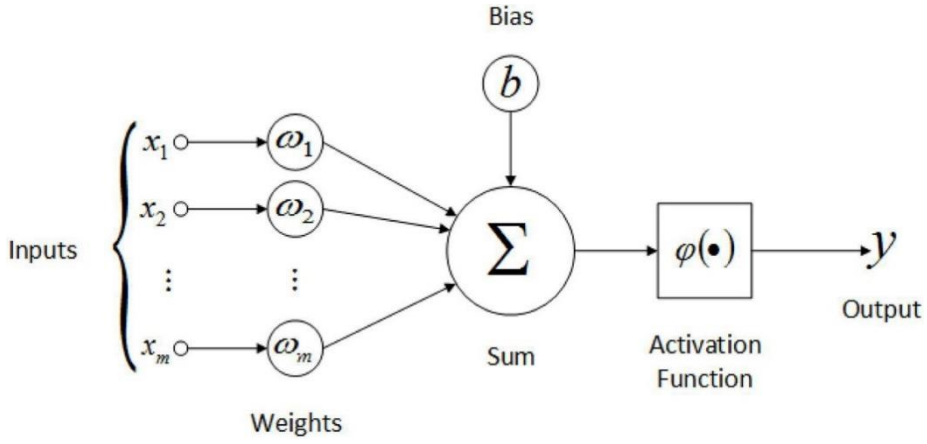
3.1.2 Umjetne neuronske mreže

Umjetna neuronska mreža (engl. *artificial neural network*) je, kao i svaki algoritam za učenje, matematička funkcija. U ovom slučaju to je proizvoljno ugniježđena funkcija oblika (3-2) gdje *L* predstavlja ukupan broj slojeva, a *x* ulaze.

$$y = f_{NN}(x) = f_L\left(\dots f_2\left(f_1(x)\right)\right) \tag{3-2}$$

Važno je napomenuti da neuronska mreža može biti konstruirana na mnoštvo načina no ovdje ćemo spomenuti njenu uobičajenu reprezentaciju dovoljnu za savladavanje osnovnih koncepata.

Svaku pojedinačnu funkciju možemo promatrati kao blok ili sloj umjetne neuronske mreže sastavljen od prethodno definiranog broja umjetnih neurona. Neuroni predstavljaju jednostavne računalne jedinice povezane s ulazima mreže ili izlazima prethodnog sloja. Zadaća neurona je apliciranje aktivacijske funkcije na težinski zbroj ulaza (engl. *weighted sum*) i slanje tog podatka dalje u mrežu. Vizualna reprezentacija jednog neurona prikazana je na slici 3.1. gdje x_i označava ulaze, ω_i težine, *b* pomak, φ aktivacijsku funkciju i *y* izlaz.

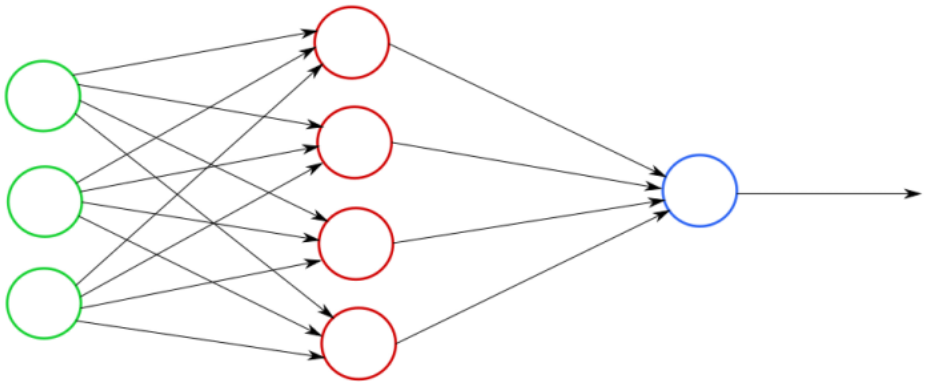


Sl. 3.1. Struktura umjetnog neurona.

Operacije svih neurona za jedan sloj mogu se matrično prikazati prema formuli (3-3) gdje $\mathbf{W}_l$ predstavlja matricu koja sadrži težine za svaki neuron, $\mathbf{y}_{l-1}$ ulaze u sloj, a $\mathbf{b}_l$ vektor svih pomaka.

$$f_l(z) = \varphi_l(\mathbf{W}_l\mathbf{y}_{l-1} + \mathbf{b}_l) \tag{3-3}$$

Ako se poveže više takvih slojeva, dobiva se unaprijedna potpuno povezana neuronska mreža. Primjer unaprijedne neuronske mreže s jednim skrivenim slojem dan je na slici 3.2.



Sl. 3.2. Primjer neuronske mreže s jednim skrivenim slojem.

Nakon odabira strukture mreže odnosno broja slojeva, broja neurona u slojevima i aktivacijskih funkcija, treba definirati funkciju troška prema kojoj će se na temelju podataka za učenje optimizirati parametri mreže $\mathbf{W}_l$ i $\mathbf{b}_l$. Primjer funkcije troška je srednja kvadratna pogreška koju možemo izračunati po formuli (3-4) gdje $\hat{y}_i$ predstavlja dobiveni izlaz, y_i očekivani izlaz, a m ukupni broj uzoraka u dostupnom skupu podataka.

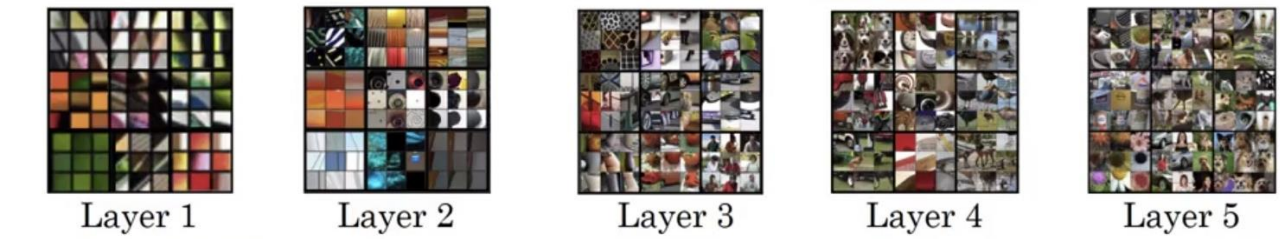
$$\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 \tag{3-4}$$

Rješenje navedenog optimizacijskog problema nije moguće dobiti izravno već se do njega dolazi iterativnim numeričkim postupcima poput metode gradijentnog spusta. Gradijentni spust se zasniva na gradijentu funkcije troška koji se izračunava algoritmom propagacije unazad (engl. *backpropagation algorithm*).

3.1.3. Konvolucijske neuronske mreže

Konvolucijske mreže su specifične po tome što sadrže konvolucijske slojeve. One su optimizirane za rad s ulaznim podacima strukturiranih kao dvodimenzionalno polje kao što je slika. Naime klasične neuronske mreže s potpuno povezanim slojevima imaju problem što broj parametara eksponencijalno raste kako povećavamo broj slojeva što uzrokuje tešku optimizaciju takvih modela. Konvolucijska neuronska mreža drastično smanjuje broj parametara kod dubokog učenja jer koristi princip dijeljenih težina.

Naime neuroni se u konvolucijskom sloju nazivaju filteri (engl. *kernels*) kojih može biti više za svaki sloj i koji sekvencijalno promatraju dijelove ulaza ovisno o dimenzijama filtera. Filteri sadrže spomenute težine i njih se zapravo optimizira da predstavljaju razinu detekcije određenog uzorka kada se primjene u dijelu kojeg u tom trenutku procesiraju i da informaciju o detekciji prosljeđuju dalje. Kompleksnost uzorka kojeg filtri nauče detektirati ovisi u kojem se sloju nalaze. Na slici 3.3. su po slojevima prikazani isječci slika koji uzrokuju maksimalnu aktivaciju pripadajućeg filtra. Primijetimo da u početnim slojevima filtri uče jednostavnije značajke poput rubova ili tekstura dok daljnji filtri znaju prepoznati daleko složenije strukture na temelju prethodnih.

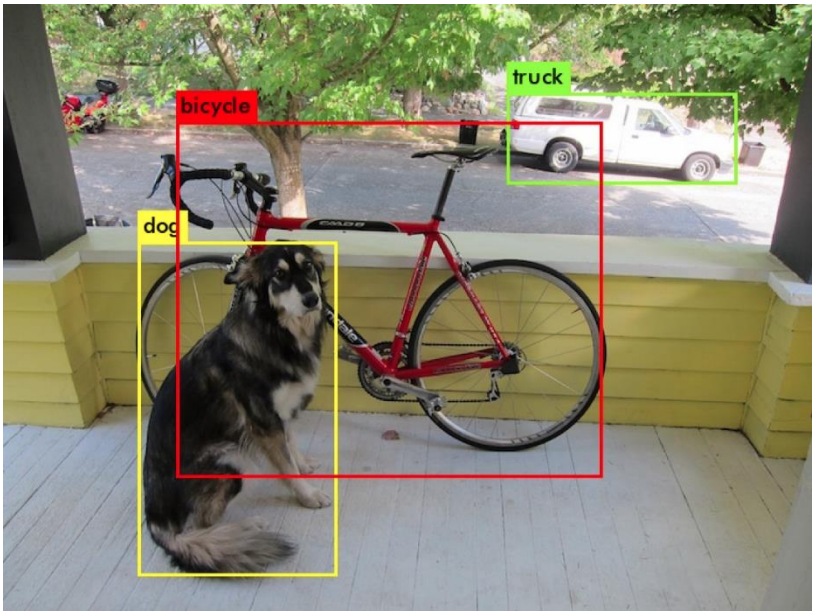


Sl. 3.3. Maksimalne aktivacije filtera u slojevima CNN-a.

Primjenom konvolucijskih neuronskih mreža mogu se riješiti problemi klasifikacije i detekcije objekata na slikama.

3.1.4. Detektori objekata

Algoritmi koji predstavljaju detektore objekata obavljaju funkciju detekcije lokacije gdje se na slici nalazi objekt od interesa i koje je on klase kao prema slici 3.4. Detekcija objekta se kod ovakvih algoritama najčešće očituje zadatkom pronalaženja koordinata četverokuta koji obuhvaća pronađeni objekt. Ove koordinate četverokuta nazivamo granični pravokutnik (engl. *bounding box*). Granični pravokutnici su definirani s obzirom na pripadajući koordinatni sustav slike za koju se označavaju objekti. Najčešće su definirani pomoću gornje lijeve koordinate ili koordinate središta zajedno sa širinom i visinom pravokutnika.



Sl. 3.4. Detekcija objekata na slici uz prikaz graničnih pravokutnika i uparenih klasa.

Moderni algoritmi za detekciju objekata se oslanjaju na konvolucijske neuronske mreže uz različite načine pronalaženja graničnih pravokutnika. Najjednostavniji detektori objekata se oslanjaju na obične klasifikatore koji se pokreću na svakom dijelu (prozoru) slike gdje bi rubovi tog dijela s obzirom na izvornu sliku odgovarali graničnom pravokutniku ako je klasifikator siguran da je pronađen objekt. Takva metoda se obično naziva metoda pomičnih prozora (engl. *sliding window technique*).

3.2 Algoritam za detekciju zauzetosti parkirališta

Algoritam predložen u ovom radu omogućuje automatsko segmentiranje individualnih parkirnih mjesta te daljnju detekciju zauzetosti parkirališta na temelju slika dobivenih od kamere usmjerene na parkiralište. Pretpostavke algoritma su da kamera neće pomicati svoje vidno polje ili ako se pomiče da će se barem ubrzo vratiti u prvobitnu poziciju s početka segmentiranja parkirnih mjesta. Algoritam omogućuje segmentaciju parkirnih mjesta pomoću detektora objekata i predloženog algoritma *PSD*, a detekciju zauzetosti određuje pomoću binarnog klasifikatora zasnovanog na konvolucijskoj neuronskoj mreži.

3.2.1 Segmentacija parkirnih mjesta

Da bi algoritam mogao odrediti zauzetost parkirnih mjesta potrebno je prvo locirati gdje se parkirna mjesta nalaze na slikama sa snimke u smislu graničnih pravokutnika istih. Određivanja graničnih pravokutnika parkirnih mjesta se dalje u tekstu naziva segmentacija parkirnih mjesta.

Uobičajeno se segmentacija parkirnih mjesta obavlja ručno, označavajući granične pravokutnike pomoću nekog programa na računalu (vidi poglavlje 2). U ovom radu je predstavljan automatizirani pristup temeljen na detektoru vozila i predloženom *PSD* algoritmu. Naime parkirna mjesta se u predloženom algoritmu promatraju kao 2D površine slike gdje se uzastopno pojavljuju detekcije vozila odnosno gdje se vozila često zadržavaju. Ovo najprije zahtjeva provedbu detekcije vozila na slici pomoću spomenutog detektora objekata, a analizu da li se zaista tamo vozila zadržavaju odrađuje *PSD* algoritam.

3.2.1.1 Detektor vozila

Detektor vozila služi u procesu segmentacije parkirnih mjesta i tijekom toga prvi prima slike s video kamere. On ima zadaću detektirati sve automobile na primljenoj slici u obliku liste graničnih pravokutnika koju prosljeđuje *PSD* algoritmu.Za detektor vozila je korišten algoritam *YOLOv3* (engl. *You Only Look Once, version 3*) [8].

YOLOv3 je energetski učinkovit detektor objekata koji može obavljati detekciju objekata u stvarnom vremenu. To postiže metodom kojom sliku procesira samo jednom (engl. *single shot*).On detekciju obavlja jednim prolazom kroz duboku neuronsku mrežu gledajući problem kao regresiju. Ulaznu sliku najprije podijeli na pravokutne ćelije gdje u svakoj predviđa određeni broj graničnih pravokutnika

zajedno s vjerojatnošću objekata u istim. U ovom radu *YOLOv3* model nije treniran od nasumičnih inicijalnih vrijednosti parametara mreže nego je korišten trenirani model na *COCO* (engl. *common objects in context*) [9] skupu podataka koji je javno dostupan. *COCO* skup podataka je upravo napravljen za treniranje detektora objekata.

Specifikacije skupa podataka *COCO su sljedeće*:

- 328,000 slika
- 2,500,000 ukupnih označenih objekata na slikama
- 91 klasa (sadrži vozila)

Prilikom korištenja *YOLOv3* potrebno je odabrati prag (engl. *threshold*) detekcije. Prag detekcije je vrijednost s kojom uspoređujemo procijenjenu vjerojatnost da neki granični pravokutnik sadrži objekt od interesa. Ako je procijenjena vjerojatnost manja od praga detekcije, tada detekciju ne smatramo detekciju važećom jer često u takvim slučajevima granični pravokutnik upravo ne sadrži objekt od interesa. Prag detekcije koji je korišten u radu je empirijski određen i iznosi 0.6.

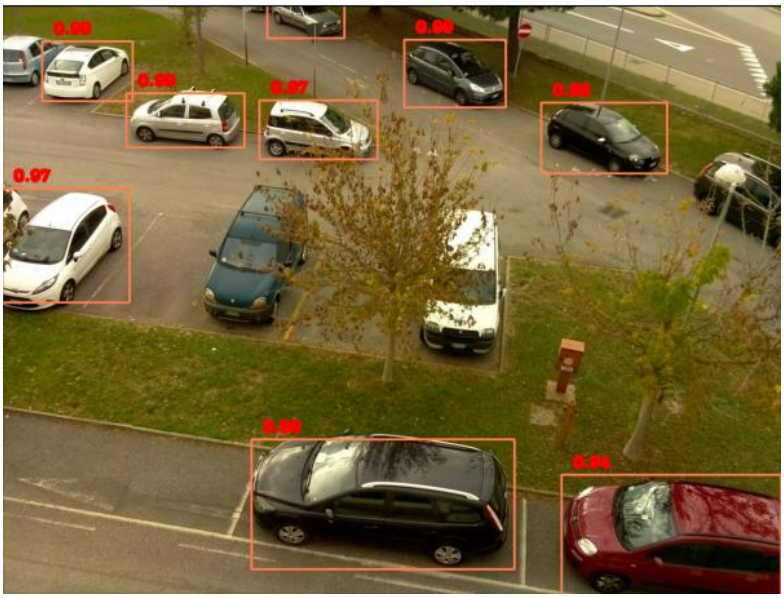
Detektori objekata često jednom objektu na slici pridruže više graničnih četverokuta zbog velikog broja detekcija s različitim dimenzijama graničnih četverokuta koji obuhvaćaju isti 2D prostor slike. Filtriranje pridruženih graničnih četverokuta zadržavajući one najbolje postiže se algoritmom potiskivanja ne-maksimalne vrijednosti (engl. *Non-max suppression*).

Potiskivanje ne-maksimalne vrijednosti pronalazi granični pravokutnik uparen s najvećom vjerojatnošću i njega proglašava kao važeći. Nakon toga sve ostale granične pravokutnike koje s ovim imaju IoU (engl. *Intersection Over Union - IoU*) vrijednost veću od nekog praga odbacuje. Ovaj postupak se ponavlja sve dok ima detektiranih graničnih pravokutnika za provjeru. *IoU* je omjer površine presjeka i unije oba granična četverokuta (vidi 3-5). Djelovanje algoritma ne-maksimalne vrijednosti možemo vidjeti prema slici 3.5. gdje od tri detektirana granična pravokutnika ostaje samo jedan i to onaj najvećom vjerojatnošću da je unutar njega detektirani objekt.

$$IoU = \frac{A \wedge B}{A \, U \, B} \tag{3-5}$$

Sl. 3.5. Utjecaj algoritma potiskivanja ne-maksimalne vrijednosti.

Nakon što su detekcije napravljene i nakon što je napravljeno filtriranje s pragom detekcije i algoritmom potiskivanja ne-maksimalne vrijednosti dobivaju se granični pravokutnici za objekte na slici, pri čemu jednom objektu pripada točno jedan granični pravokutnik. Primjer detekcije vozila na slici iz *CNRPark-EXT* skupa podataka se može vidjeti na slici 3.6. Granični pravokutnici se dalje prosljeđuju *PSD* algoritmu.



Sl. 3.6. Primjer *YOLOv3* detekcija vozila za sliku iz *CNRPark-EXT* skupa podataka.

3.2.1.2 *PSD* (*Parking Spot Detector*)

PSD (engl. *Parking Spot Detector*) je algoritam koji na temelju detekcija vozila provodi segmentaciju parkirnih mjesta. Naime nije moguće zaključiti s jedne slike da su parkirna mjesta granični

pravokutnici detektiranih vozila jer automobili se na parkiralištu kreću i detektor objekata često detektira automobile koji se u tom trenutku ne nalaze na parkirnom mjestu nego su u vožnji. Također, da bi to bilo moguće, parkiralište bi moralo biti popunjeno i detektor objekata bi morao u jednoj detekciji točno odrediti sve granične pravokutnike svih vozila što nije realna situacija. Zbog toga je potrebno detekciju vozila provoditi kroz duže vremensko razdoblje i pamtiti na kojem dijelu slike detektor objekata neprekidno detektira vozilo što interpretiramo kao da se zadržava na istom mjestu odnosno na jednom parkirnom mjestu.

Nakon toga se postavlja pitanje nakon koliko vremena zadržavanja vozila na istom mjestu možemo zaključiti da je to mjesto parkirno mjesto. Ovo ovisi o kulturi vozača i podneblju no mi smo za naš algoritam uzeli prag od 25 minuta odnosno ako prosljeđujemo sliku algoritmu svakih 5 minuta potrebno je 5 uzastopnih detekcija na istom mjestu kako bi algoritam zaključio da ta detekcija vozila zapravo predstavlja i parkirno mjesto. Parkirno mjesto je tada predstavljeno graničnim pravokutnikom koji odgovara zadnjoj detekciji vozila na tom mjestu.

PSD algoritam u sebi sadrži matricu koja ažurira svoje vrijednosti za svaku primljenu listu graničnih pravokutnika od detektora vozila tako da predstavlja svojevrsnu toplinsku kartu zadržavanja vozila na slikama.

Ako je parkiralište koje se nadzire vrlo iskorišteno, može se očekivati potpuno segmentiranje parkirnih mjesta već u vremenu do jednog dana, a za manje iskorištena parkirališta proces segmentacije treba provoditi sve dok nismo sigurni da su sva parkirna mjesta bila zauzeta najmanje 25 minuta u kontinuitetu što je potrebno za segmentaciju parkirnog mjesta. *PSD* matricu još nazivamo *heatmap* matrica u nastavku teksta.

Heatmap matrica se na početku segmentiranja inicijalizira s dimenzijama širine i visine slike koja je prva primljena i to na vrijednosti *upperHeatmapLimit*. *UpperHeatmapLimit* ima vrijednost 10 radi lakšeg izvoda budućih konstanti, no u suštini je proizvoljna veličina.

Postavlja se i vrijednost *parkingSegTH* koja predstavlja prag potrebnih uzastopnih detekcija za segmentaciju parkirnog mjesta krećući od *upperHeatmapLimit* vrijednosti. Za *parkingSegTH* je empirijski odabrana vrijednost 5. Pomoću parkingSegTH vrijednosti određuju se konstante *Tsensitivity* i *Fsensitivity* koje se koriste tijekom ažuriranja matrice na svaku primljenu listu graničnih pravokutnika.

Vrijednosti *heatmap* matrice odražavaju razinu sumnje da pikseli na istim koordinatama s učitanih slika (ako sliku promatramo kao matricu) pripadaju graničnim pravokutnicima parkirnih mjesta. Interval vrijednosti koje individualne vrijednosti matrice mogu poprimiti je [0,10].

Manje vrijednosti odgovaraju čestim detekcijama vozila, a veće rjeđim. Algoritam prilikom ažuriranja smanjuje vrijednosti na koordinatama *heatmap* matrice koje odgovaraju koordinatama slike unutar graničnih pravokutnika detektiranih vozila, a povećava vrijednosti u suprotnom slučaju. Ovim se osigurava potreba za učestalim pojavljivanjem detekcija vozila na istim mjestima unutar slike kako bi se takva mjesta u slici segmentirala kao parkirna mjesta.

Tsensitivity ili *true sensitivity* predstavlja konstantu kojom se prilikom ažuriranja množe *heatmap* vrijednosti na istim koordinatama kao pikseli obuhvaćeni graničnim pravokutnicima detektiranih vozila za trenutno analiziranu sliku. *Tsensitivity* uvijek ima vrijednost manju od 1 i računa se prema izrazu (3-6) pomoću *upperHeatmapLimit veličine* i *parkingSegTH*.

$$upperHeatmapLimit \cdot Tsensitivity^{parkingSegTH-1} < 1$$
$$Tsensitivity < \left(\frac{1}{10}\right)^{\frac{1}{parkingSegTH-1}} \tag{3-6}$$

Fsensitivity ili *false sensitivity* predstavlja konstantu kojom se prilikom ažuriranja množe *heatmap* vrijednosti na koordinatama različitim od onih od piksela obuhvaćenih graničnim pravokutnicima detektiranih vozila za trenutno analiziranu sliku. *Fsensitivity* uvijek ima vrijednost veću od 1 i računa se prema izrazu (3-7) pomoću *upperHeatmapLimit* i *Tsensitivity* veličina.

$$upperHeatmapLimit \cdot Tsensitivity \cdot Fsensitivity = 10$$
$$Fsensitivity = \frac{1}{Tsensitivity} + \varepsilon \tag{3-7}$$

Tsensitivity se koristi jer u slučaju da se detektira vozilo u vožnji, koje nije na parkirnom mjestu, vrijednosti koje su smanjene u *heatmap* matrici trebale bi se povećati na prethodne vrijednosti ako u sljedećoj detekciji na tom mjestu nije pronađeno vozilo. Želja je više te vrijednosti povećati nego što su prethodno smanjene stoga se *Fsensitivity* veličini dodana još i vrijednost ε od 0.1.

Kako bi se smanjio broj potrebnih množenja sve *Tsensitivity* i *Fsensitivity* vrijednosti se prvo upisuju u matricu koju nazivamo maska pa se tek pred kraj ažuriranja ta maska spaja *s heatmap* matricom pomoću Hadamard umnoška.

Ažuriranje *heatmap* matrice se obavlja svaki put kada se prime novi granični pravokutnici od detektora vozila. Tada se za svaki granični pravokutnik provjerava kojoj vrijednosti u *heatmap* matrici odgovaraju koordinate njegovog središta.

Ako se u *heatmap* matrici na mjestu koordinata središta graničnog pravokutnika nalazi vrijednost 0 zanemaruje se ovaj granični pravokutnik jer 0 u *heatmap* matrici predstavlja dio već segmentiranog parkirnog mjesta.

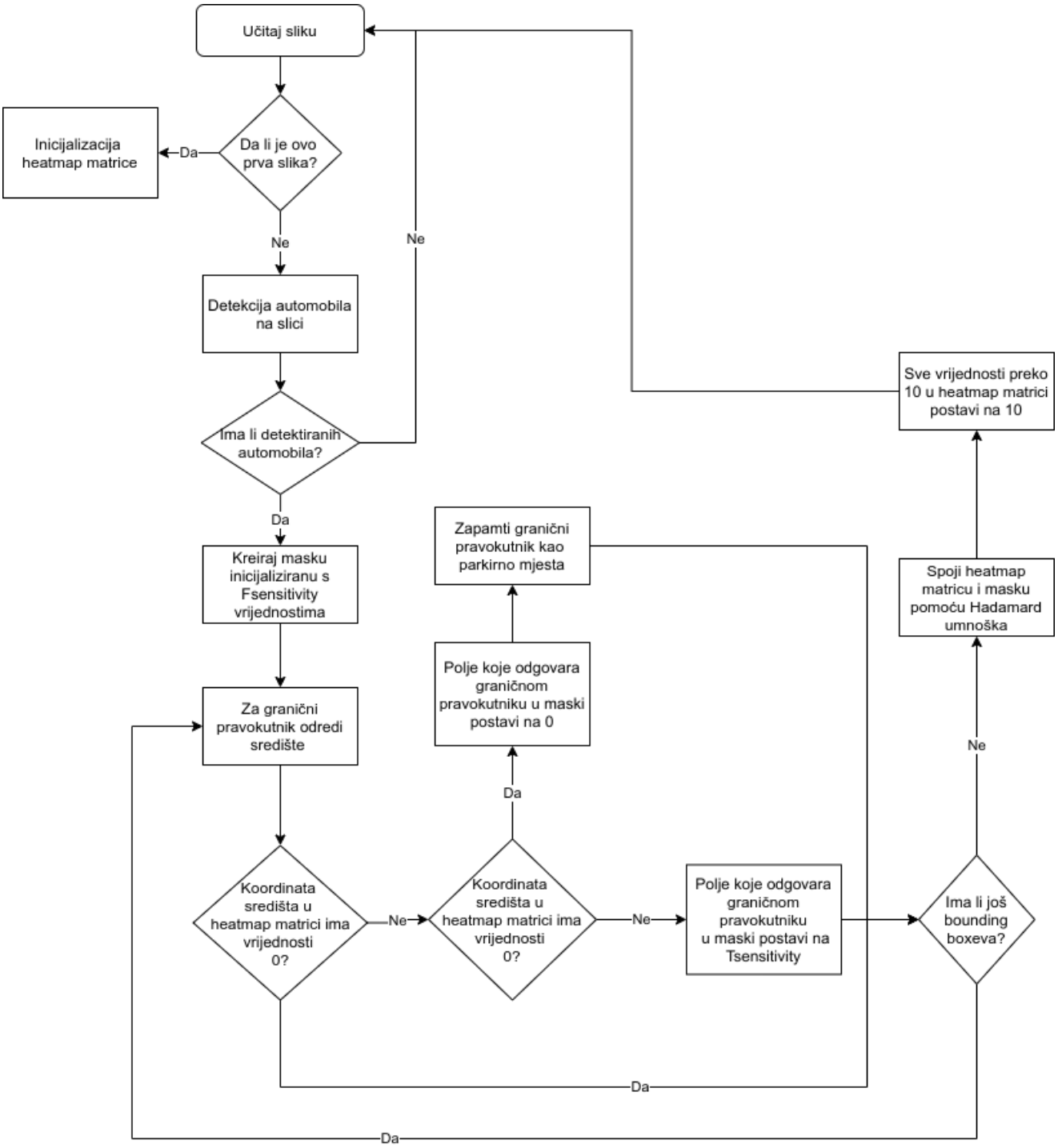
Ako se u *heatmap* matrici na mjestu koordinata središta graničnog pravokutnika nalazi vrijednost manja od 10 i veća ili jednaka 1 onda se u masku upisuje taj granični pravokutnik s vrijednostima *Tsensitivity*. Skala (engl. *scale*) je još jedna postavka koja se može koristiti u procesu upisa u masku, a služi za skaliranje graničnog pravokutnika na manju površinu nego izvorno definirano s istim središtem u svrhu veće preciznosti lokaliziranja budućih vozila čija središta moraju biti u tom području. Korištena je vrijednost skale od 0.5.

Ako se u *heatmap* matrici na mjestu koordinata središta graničnog pravokutnika nalazi vrijednost manja od 1, ali različita od 0 to znači da se "zaključava" taj granični pravokutnik kao parkirno mjesto i upisuje se u *heatmap* matricu. Na kraju se kreirana maska spaja *s heatmap* matricom pomoću Hadamard umnoška i ograničavaju se sve vrijednosti koje su prešle prag od *upperHeatmapLimit* vrijednosti ponovno na *upperHeatmapLimit*.

Sirova interpretacija vrijednosti *heatmap* matrice u bilo kojem trenutku:

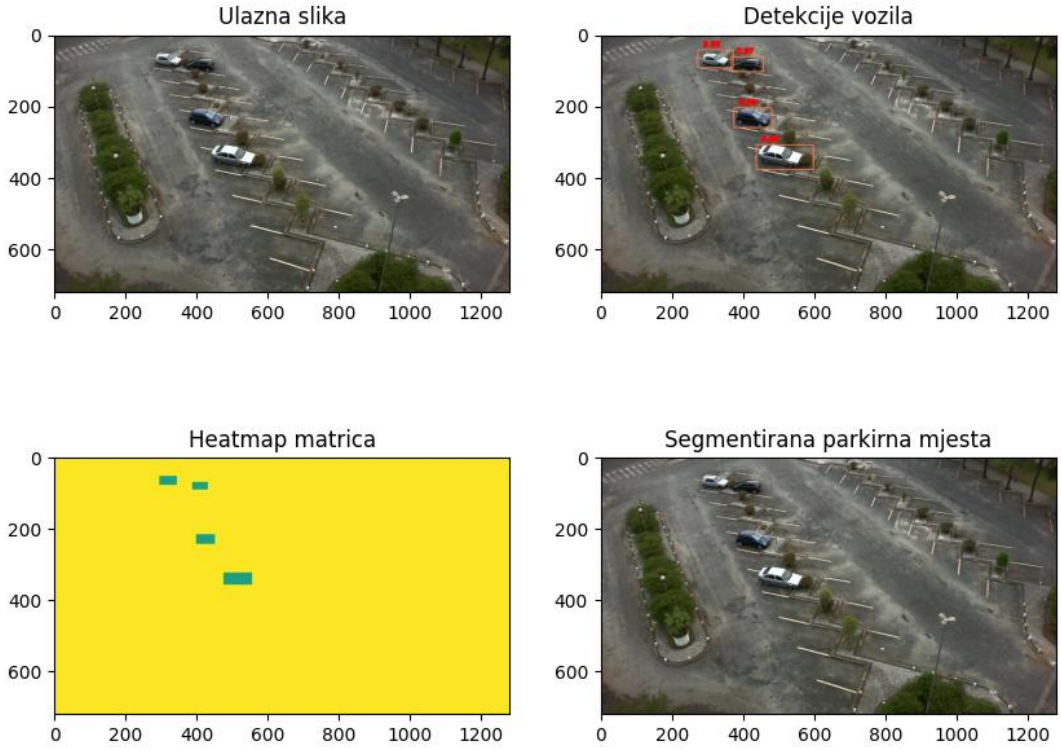
- Vrijednosti 10 znače ne pojavljivanje detekcija vozila u zadnjih *parkingSegTH* detekcija
- Vrijednosti u intervalu [1,10) znače pojavljivanje detekcija vozila u zadnjih *parkingSegTH* detekcija.
- Vrijednosti u intervalu (0,1) znače da ako središta sljedećih detekcija vozila budu na tim koordinatama signalizirat će segmentiranje parkirnog/parkirnih mjesta.
- Vrijednosti 0 znače dio graničnog pravokutnika segmentiranog parkirnog mjesta. Zbog svoje vrijednosti one su zaključane i ne mogu se više mijenjati. Promjena nije moguća jer sva ažuriranja vrijednosti matrice su operacije množenja

Na slici 3.7. dan je dijagram toka segmentaciju parkirnih mjesta.

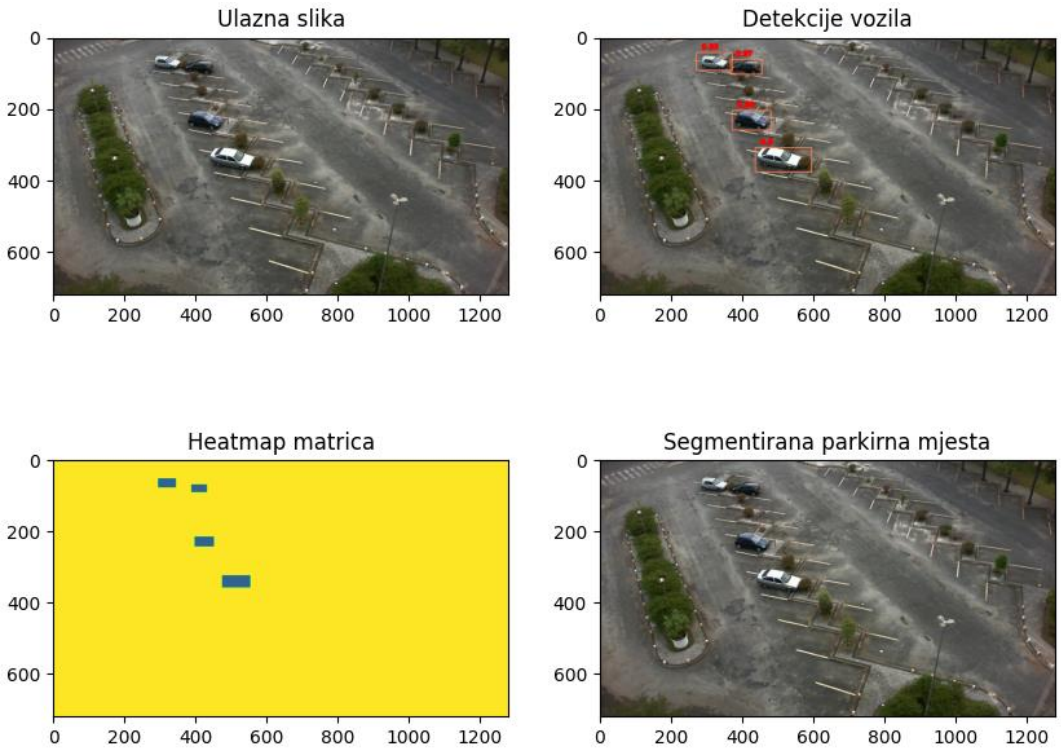


Sl. 3.7. Dijagram toka segmentacije parkirnih mjesta.

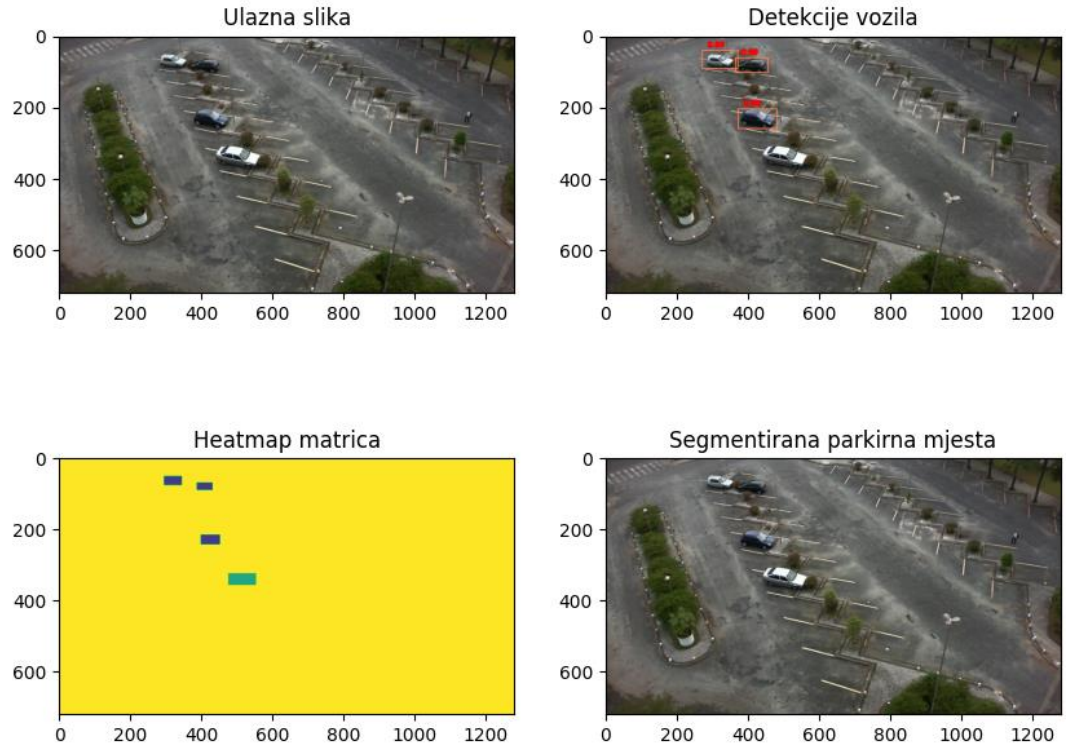
Primjer tijeka segmentiranja parkirnih mjesta na 5 slika s kamere s frekvencijom uzorkovanja od 5 minuta dan je na slikama 3.8.-3.12. Na svakoj slici je prikazana ulazna slika (gore lijevo), detekcije detektora vozila za ulaznu sliku gdje su granični pravokutnici prikazani narančastim obrubom (gore desno), vrijednosti u *heatmap* matrici (dolje lijevo) te slika sa segmentiranim parkirnim mjestima (dolje desno).



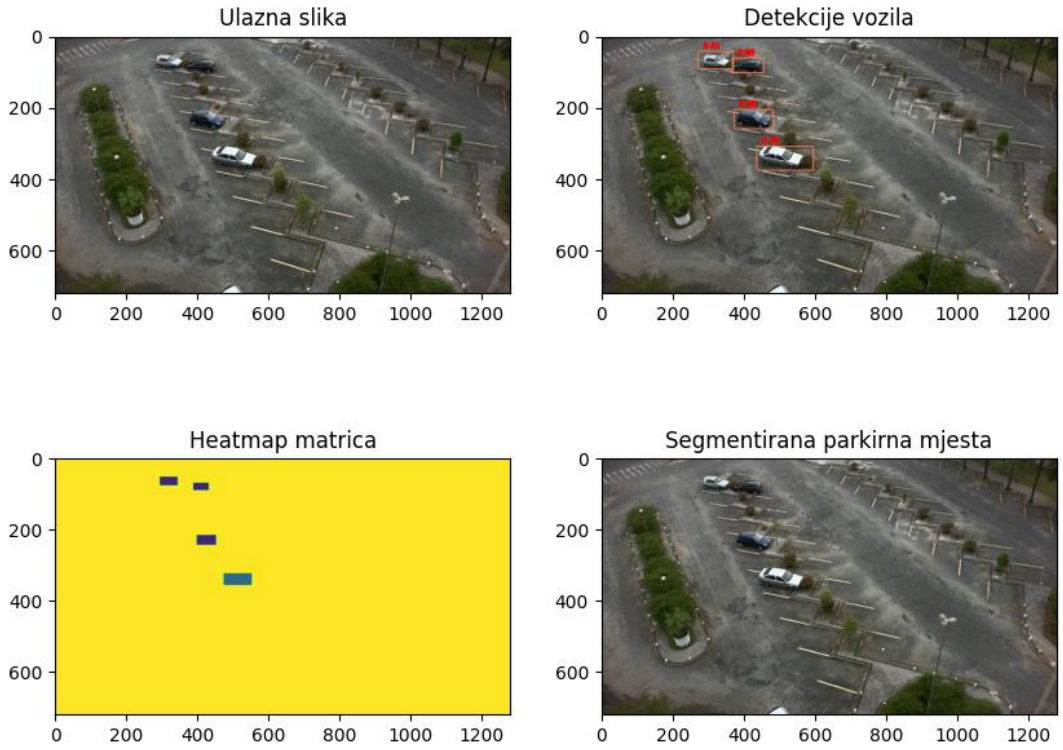
Sl. 3.8. Segmentiranje parkirnih mjesta, 1. slika.



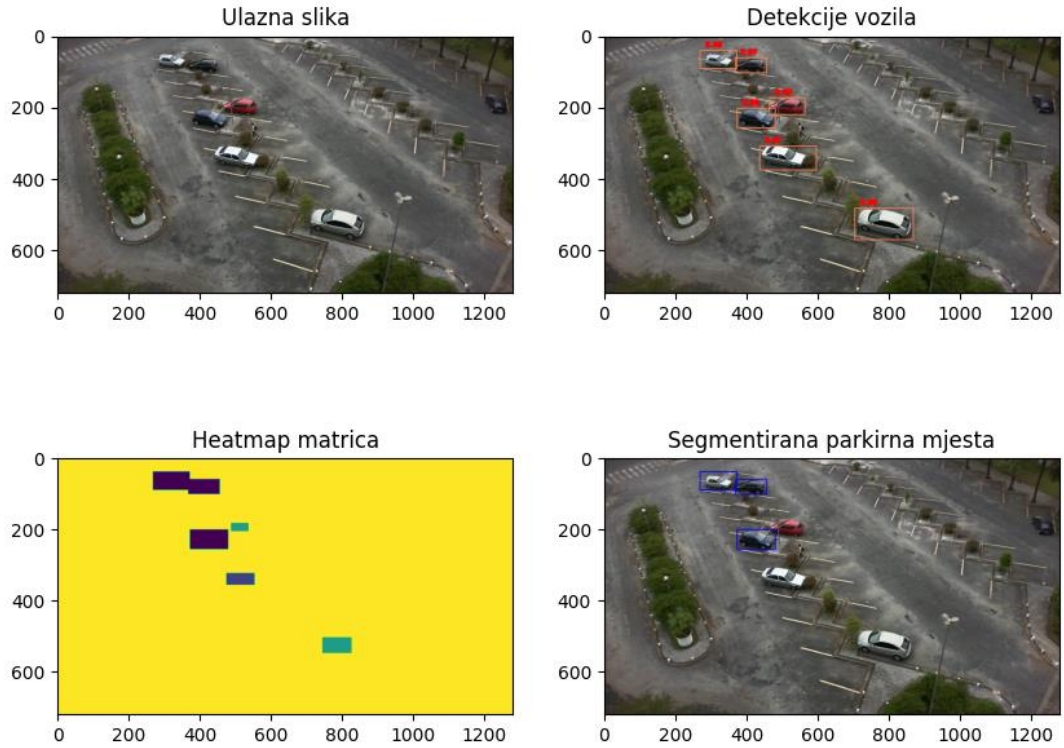
Sl. 3.9. Segmentiranje parkirnih mjesta, 2. slika.



Sl. 3.10. Segmentiranje parkirnih mjesta, 3. slika.



Sl. 3.11. Segmentiranje parkirnih mjesta, 4. slika.

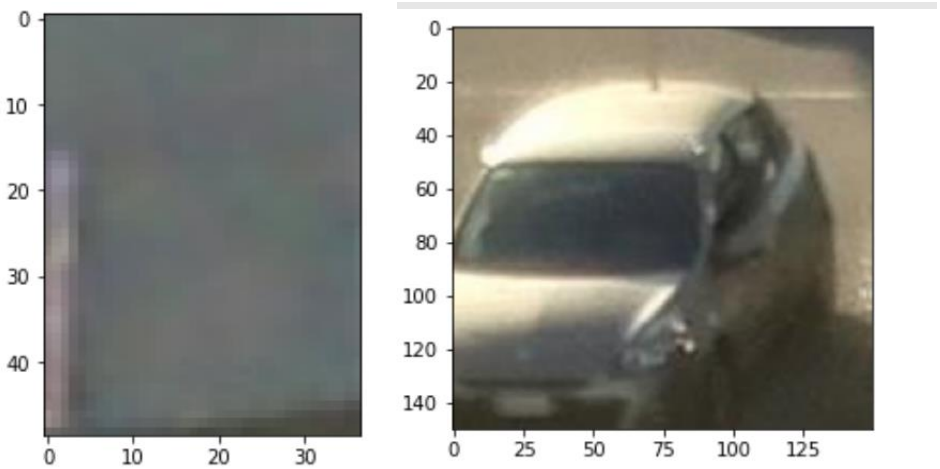


Sl. 3.12. Segmentiranje parkirnih mjesta, 5. slika.

Vidi se da uzastopne detekcije na odgovarajućim lokacijama u ulaznoj slici daju sve manje i manje vrijednosti u *heatmap* matrici sve dok se parkirno mjesto ne zaključa kada granični pravokutnik detekcije postaje granični pravokutnik parkirnog mjesta.

3.2.2 Detekcija zauzetosti parkirnih mjesta

Nakon segmentiranja parkirnih mjesta moguće je primijeniti binarni klasifikator kako bi se odredila zauzetost svakog parkirnog mjesta. Uz klasifikator više nije potreban detektor objekata koji često ima manju točnost i značajno veće vrijeme izvršavanja jer su poznate lokacije svih parkirnih mjesta i samo se želi odrediti da li su ona zauzeta ili slobodna. Segmentirana parkirna mjesta su sada predstavljena listom graničnih četverokuta uz pomoć kojih se iz izvorne slike mogu izrezati isječci parkirnih mjesta i predati treniranom klasifikatoru zauzetosti koji na svom izlazu daje vjerojatnost zauzeća parkirnog mjesta. Na slici 3.13. dana su dva primjera isječka parkirnog mjesta iz izvorne slike.



Sl. 3.13. Primjer isječka slobodnog (lijevo) i zauzetog (desno) parkirnog mjesta.

Primjer djelovanja za sliku uz prethodno segmentirane granične pravokutnike parkirnih mjesta je prikazan prema slici 3.14.. Slobodna parkirna mjesta su prikazana graničnim pravokutnicima zelene, a zauzeta crvene boje obruba.

Sl. 3.14. Prikaz klasifikacije parkirnih mjesta na zauzeta (crvena boja) i slobodna (zelena boja).

Kao osnova klasifikatora korištena je *ResNet34* [10] duboka neuronska mreža uz težine prethodno trenirane na *ImageNet* [11] skupu podataka. Ovaj model pokazao se jako dobar u raznim primjenama u području računalnog vida. *ResNet34* proširen je dodatnim slojem koji je treniran za prepoznavanje zauzetosti parkirnog mjesta.

Za treniranje klasifikatora korišteni su dijelovi dva prethodno opisana skupa podataka: *CNRPark-EXT* i *PKLot*. Podaci su prije treniranja podijeljeni na 3 dijela:

- Skup za treniranje (engl. *training set*),
- Skup za provjeru valjanosti (engl. *validation set*) i
- Skup za testiranje (engl. *test set*).

Ovo je uobičajena praksa koja nam omogućava evaluaciju modela. Skup za treniranje je većinski dio podataka koji se koristi za treniranje modela. Skup za provjeru valjanosti i skup za testiranje zajedno se nazivaju *hold-out* skupovi. Skup za provjeru valjanosti služi za provjeru generalizacije i davanje informacije na temelju kojih je moguće dodatno optimizirati model, a test skup čuva se za sami kraj kada je potrebno prikazati rezultate generalizacije algoritma bez ikakve optimizacije nad tim skupom podataka koje bi moglo poremetiti naše mišljenje o uspjehu generalizacije našeg modela.

Klasifikator je treniran na podacima iz *PKLot* i *CNRPark-EXT* skupa podataka. Prije treniranja su izdvojene slike u sljedećim mapama za skup za testiranje.

- *PKLOTSegmented UFPRO5 2013-03-13*
    - *PKLot/PKLotSegmented/UFPR05/Cloudy/2013-03-13*
    - *PKLot/PKLotSegmented/UFPR05/Rainy/2013-03-13*
    - *PKLot/PKLotSegmented/UFPR05/Sunny/2013-03-13*
- *PKLOTSegmented UFPRO4 2012-12-14*
    - *PKLot/PKLotSegmented/UFPR04/Cloudy/2012-12-14*
    - *PKLot/PKLotSegmented/UFPR04/Rainy/2012-12-14*

Nakon što je odvojen skup za testiranje odvojeno je i 10% preostalih podataka za skup za provjeru valjanosti. Ostali podaci svi spadaju u skup za treniranje.

Prije treniranja je od skupa za treniranje i skupa za provjeru valjanosti napravljen *fastai* objekt klase *ImageDataBunch* normaliziran *ImageNet* specifikacijama.

ImageDataBunch specifikacije:

(['occupied', 'vacant'], 2, 747451, 83052)

Arhitektura je napravljena pomoću *fastai* sustava i trenirana na *google colab* platformi. Platforma *google colab je odabrana* jer nudi besplatan pristup grafičkim karticama. Klasifikator postiže točnost oko 99.9% na skupu za provjeru valjanosti. Dnevnik treniranja je prikazan u tablici 3.1., a matrica zabune je prikazana u tablici 3.2.

Tab. 3.1. Dnevnik treniranja.

Epoha	Trošak na skupu za treniranje	Trošak na skupu za provjeru valjanosti	Postotak pogrešaka na skupu za provjeru valjanosti	Vrijeme izvršenja epohe
0	0.022380	0.010574	0.2721%	29:15
1	0.012725	0.012593	0.1878%	28:28
2	0.011767	0.006179	0.1625%	27:13
3	0.008335	0.005128	0.1240%	25:18
4	0.008588	0.005032	0.1192%	25:18

Tab. 3.2 Matrica zabune na skupu za provjeru valjanosti.

	Detektirano kao zauzeto	Detektirano kao slobodno
Zaista zauzeto	40761	40
Zaista slobodno	59	42192

Svaka vrijednost matrice zbunjenosti ima svoj naziv:

- TP - broj parkirnih mjesta koja su zaista zauzeta i detektirana kao zauzeta (engl. *true positive*)
- FN - broj parkirnih mjesta koja su zaista zauzeta, ali detektirana kao slobodna (engl. *false negative*)
- TN- broj parkirnih mjesta koja su zaista slobodna i detektirana kao slobodna (engl. *true negative*)
- FP -broj parkirnih mjesta koja su zaista slobodna, ali detektirana kao zauzeta (engl. *false positive*)

Iz matrice zabune primjetno je da je skup za provjeru valjanosti uravnotežen. Također dobra strana je to što klasifikator ima više lažno pozitivnih nego lažno negativnih detekcija jer bolje je da u slučaju pogreške za slobodno parkirno mjesto odredi da je zauzeto nego da za zauzeto odredi da je slobodno.

3.3 Implementacija pojedinih dijelova predloženog algoritma

Algoritam detekcije zauzetosti parkirališta pomoću računalnog vida implementiran je u Python programskom jeziku te je korištena objektno orijentirana paradigma. Napravljene su sljedeće klase koje upotpunjuju opisano:

- *ParkingSlot*
- *ParkingLot*
- *ParkingObserver*
- Prediction
- *Yolo3Detector*
- *Classificator*

3.3.1 Klasa *Yolo3Detector*

Yolo3Detector klasa je zadužena za kreiranje i upotrebu modela *YOLOv3* detektora objekata. Za kreiranje modela korišten je *OpenCV* sustav i njegova metodu *cv2.dnn.readNetFromDarknet* koja prima putanju na konfiguracijsku datoteku i datoteku s težinama modela što su i jedini argumenti potrebni pri inicijalizaciji.

```
class Yolo3Detector:
def __init__(self,configPath,weightsPath):
        self.detector = cv2.dnn.readNetFromDarknet(configPath,
weightsPath)
```

Metoda *detect* omogućava obavljanje detekcija na slici.

```
def detect(self, image):
        np.random.seed(42)
        (H, W) = image.shape[:2]

        ln = self.detector.getLayerNames()
        ln = [ln[i[0] - 1] for i in self.detector.getUnconnectedOutLayers()]

        blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (608, 608), swapRB=True,
crop=False)
        self.detector.setInput(blob)
        layerOutputs = self.detector.forward(ln)

        predictions = []
        threshold = 0.6

        for output in layerOutputs:
            for detection in output:
                scores = detection[5:]
                classID = np.argmax(scores)
                confidence = scores[classID]

                if confidence > threshold:

                    box = detection[0:4] * np.array([W, H, W, H])
                    (centerX, centerY, width, height) = box.astype("int")

                    x = int(centerX - (width / 2))
                    y = int(centerY - (height / 2))

                    box = [x, y, int(width), int(height)]
                    prediction = Prediction(box, float(confidence), classID)
                    predictions.append(prediction)

        suppressed_predictions=self.suppress(predictions)
        return suppressed_predictions
```

Detekcije su filtrirane s *threshold* veličinom i spremljene u obliku liste detekcija klase *Prediction*.

```
class Prediction:
    def __init__(self, box, confidence, class_id):
        self.box = box
        self.confidence = confidence
        self.class_id = class_id

    def get_box(self):
        return self.box

    def get_confidence(self):
        return self.confidence

    def get_class_id(self):
        return self.class_id
```

Metoda *detection* provodi algoritam za potiskivanje ne-maksimalne vrijednosti pomoću metode *suppress* koja prima listu tipa *Prediction.*

```
def suppress(self, predictions):
        threshold = 0.15
        suppressed_predictions = []

        boxes = []
        confidences = []
        class_ids = []

        for prediction in predictions:
            boxes.append(prediction.get_box())
            confidences.append(prediction.get_confidence())
            class_ids.append(prediction.get_class_id())

        idxs = cv2.dnn.NMSBoxes(boxes, confidences, threshold, 0.1)

        if len(idxs) > 0:
            for i in idxs.flatten():
                (x, y) = (max(0,boxes[i][0]),max(0,boxes[i][1]))
                (w, h) = (max(0,boxes[i][2]),max(0,boxes[i][3]))
                confidence = confidences[i]
                class_id = class_ids[i]

                if class_id==2:
                    suppressed_prediction = Prediction([x, y, w, h], confidence, class_id)
                    suppressed_predictions.append(suppressed_prediction)

        return suppressed_predictions
```

3.3.2 Klasa *Classificator*

Classificator je klasa koja predstavlja binarni klasifikator zauzetosti. Kreira se pomoću putanje na trenirani *fastai* model i koristi *classify* metodu za klasifikaciju slike. Metoda *classify* prima listu parkirnih mjesta i ulaznu sliku parkirališta kojem ta parkirna mjesta odgovaraju vraća listu detektiranih klasa.

```
class Classificator:
    def __init__(self,model_path):
        self.model_path = model_path
        self.learn = load_learner(model_path)

    def classify(self,img,parking_slots):
        pred_classes=[]

        for parking_slot in parking_slots:

            tmp = Image(pil2tensor(img, np.float32).div_(255))

            (x, y) = (parking_slot[0], parking_slot[1])
            (w, h) = (parking_slot[2], parking_slot[3])
            tmp.px = tmp.px[:, y:y + h, x:x + w]
            pred_class, pred_idx, outputs = self.learn.predict(tmp)

            pred_classes.append(pred_class)
        return pred_classes
```

3.3.3 Klasa *ParkingObserver*

ParkingObserver klasa predstavlja sustav za detekciju zauzetosti parkirališta u obliku detektora objekata i klasifikatora s pojednostavljenim sučeljem. Osim sučelja prema modelima ažurira *heatmap* matricu za koju sadrži postavke *upper_heatmap_limit*, *parking_segmentation_threshold*, *Tsensitivity* i *Fsensitivity*.

```
class ParkingObserver:

    def __init__(self,detector, classificator):
        self.detector=detector
        self.classificator = classificator
        self.upper_heatmap_limit = 10
        self.parking_segmentation_threshold = 5
        self.Tsensitivity, self.Fsensitivity = self.get_sensitivities()
```

Računanje Tsensitivity i Fsensitivity veličina provod metoda get_sensitivities

```
    def get_sensitivities(self):
        Tsensitivity = np.power((1 / self.upper_heatmap_limit),\
                                (1 / (self.parking_segmentation_threshold - 1)))-0.0001
        print(Tsensitivity)
        Fsensitivity = 10 / (10 * Tsensitivity) + 0.1
        return Tsensitivity, Fsensitivity
```

Sučelje prema detektoru objekata i prema klasifikatoru predstavljaju metode *detect* i *classify*.

```
    def detect(self,img):
        return self.detector.detect(img)

    def classify(self,img,parking_boxes):
        return self.classificator.classify(img,parking_boxes)
```

Metoda za ažuriranje *heatmap* matrice je *get_segmented_parking_boxes* koja za predane detekcije i trenutno stanje *heatmap* matrice vraća listu novo segmentiranih parkirnih mjesta ako ih ima i ažuriranu *heatmap* matricu.

```
    def get_segmented_parking_boxes(self,predictions,heatmap):
        segmented_parking_boxes = []
        mask = np.full(heatmap.shape, self.Fsensitivity)
        for pred in predictions:
            box = pred.get_box()
            x = box[0]
            y = box[1]
            w = box[2]
            h = box[3]


            if heatmap[y + int(h / 2), x + int(w / 2)] == 0:
                continue
            elif heatmap[y + int(h / 2), x + int(w / 2)] < 1:
                print('Parking space added x {} y {} w {} h {}'.format(x, y, w, h))
                heatmap[y:int(y + h), x:int(x + w)] = 0
                segmented_parking_boxes.append([x, y, w, h])
            else:
                self.coord_to_heatmap_area(x, y, w, h, mask)

        heatmap = np.multiply(heatmap, mask)
        over_10 = heatmap > 10
        heatmap[over_10] = 10

        return segmented_parking_boxes,heatmap
```

Metoda *get_segmented_parking_boxes* koristi metodu *coord_to_heatmap_area* za upis vrijednosti u masku. Može se primijetiti implementacija *scale* veličine koja je u ovom slučaju postavljena na 0.5.

```
    def coord_to_heatmap_area(self, x, y, w, h, mask, scale=0.5):
        x = int(x + (w / 2) * (1 - scale))
        y = int(y + (h / 2) * (1 - scale))
        w = int(w * scale)
        h = int(h * scale)
        mask[y:y + h, x:x + w] = self.Tsensitivity
```

3.3.4 Klasa *ParkingSlot*

ParkingSlot je klasa koje objedinjuje sve što u našem slučaju opisuje parkirno mjesto. Ono je predstavljeno među ostalim s identifikacijskim brojem i graničnim četverokutom koji je definiran pomoću njegovog središta, ukupno širine i ukupne visine.

```
class ParkingSlot:
    def __init__(self,id,x,y,w,h,creation_time=time.time()):
        self.id=id
        self.x=x
        self.y=y
        self.w=w
        self.h=h
        self.creation_time=creation_time
        self.history=[]

    def get_box(self):
        return self.x,self.y,self.w,self.h

    def creation_time(self):
        return self.creation_time()

    def get_state(self):
        return self.history[-1]

    def update_parking_state(self,state):
        self.history.append((state,time.time()))
```

3.3.5 Klasa *ParkingLot*

ParkingLot klasa u sebi sadrži listu objekata klase *ParkingSlot* i *heatmap* matricu te prima kao argument prilikom inicijalizacije objekt tipa *ParkingObserver*.

```
class ParkingLot:
    def __init__(self,parking_observer):
        self.parking_slots = []
        self.parking_observer = parking_observer
        self.heatmap = None
        self.shape = None
        self.current_state=None
```

Sadrži sljedeće bitne metode:

- *initialize_heatmap*
- *segment_slots_from_images*

Initialize_heatmap se poziva prilikom poziva metoda za segmentiranje iz slika.

```
    def initialize_heatmap(self):
        upper_heatmap_limit = 10
        h, w, c = self.shape
        return np.full((h, w), upper_heatmap_limit, dtype='float64')
```

*segment_slots_from_image*s koristi *ParkingObserver* kako bi za slike obavio detekciju, klasifikaciju i ažuriranje *heatmap* matrice.

```
    def segment_slots_from_images(self,image_folder,debug_mode=True):
        source_folder = image_folder
        images = [os.path.join(source_folder, image_name) for image_name in
sorted(os.listdir(source_folder))]

        if self.shape == None:
            self.shape = cv2.imread(images[0]).shape
            self.heatmap = self.initialize_heatmap()

        for image in images:
            img = cv2.imread(image)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            detections = self.parking_observer.detect(img)
            segmented_parking_boxes,self.heatmap =
self.parking_observer.get_segmented_parking_boxes(detections, self.heatmap)

            pred_classes = self.parking_observer.classify(img, self.get_parking_boxes())

            if len(segmented_parking_boxes) != 0:
                segmented_parking_slots=[ParkingSlot(len(self.parking_slots)+1,\
                                                     parking_box[0],\
                                                     parking_box[1],\
                                                     parking_box[2],\
                                                     parking_box[3])\
                                         for parking_box in segmented_parking_boxes]
                self.parking_slots.extend(segmented_parking_slots)

                if debug_mode:
                    pred_classes = self.parking_observer.classify(img,
self.get_parking_boxes())
                    visualize(img, self.heatmap, detections, self.get_parking_boxes(),
pred_classes)
```

4. EVALUACIJA ALGORITMA ZA DETEKCIJU ZAUZETOSTI PARKIRALIŠTA

Predloženo rješenje za detekciju zauzetosti parkirališta evaluirano je na stvarnim snimkama parkirališta. Parkirališta sadrže različiti broj parkirnih mjesta, snimana su u različitim vremenskim uvjetima i pod različitim kutevima kako bi se ispitala robusnost predloženog rješenja u stvarnim uvjetima. Slike s kamera su procesirane svakih 5 minuta. Prilikom testiranju algoritma nije korištena vlastita video kamera već su korištene snimke javno dostupnih IP video kamera s kojih su stvoreni odgovarajući skupovi podataka i javno dostupni skup podataka za evaluaciju ovakvih rješenja (PKLot).

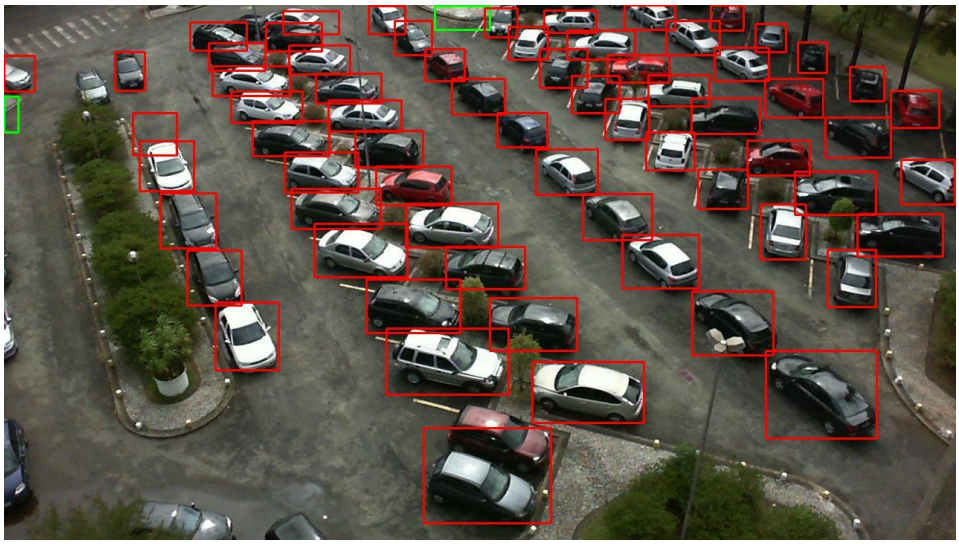
Skupovi podataka za testiranje su *UFPRO5_2013_03-13* i *UFPRO4_2012-12-14* s *PKLot* kamera te *Bay, Shore* i City s javno dostupnih IP kamera. Frekvencija uzorkovanja za sve skupove iznosi 5 minuta. Pregled skupova podataka za testiranje dan je u tablici 4.1.

Tab. 4.1. Skupovi podataka za testiranje algoritma za detekciju zauzetosti parkirališta.

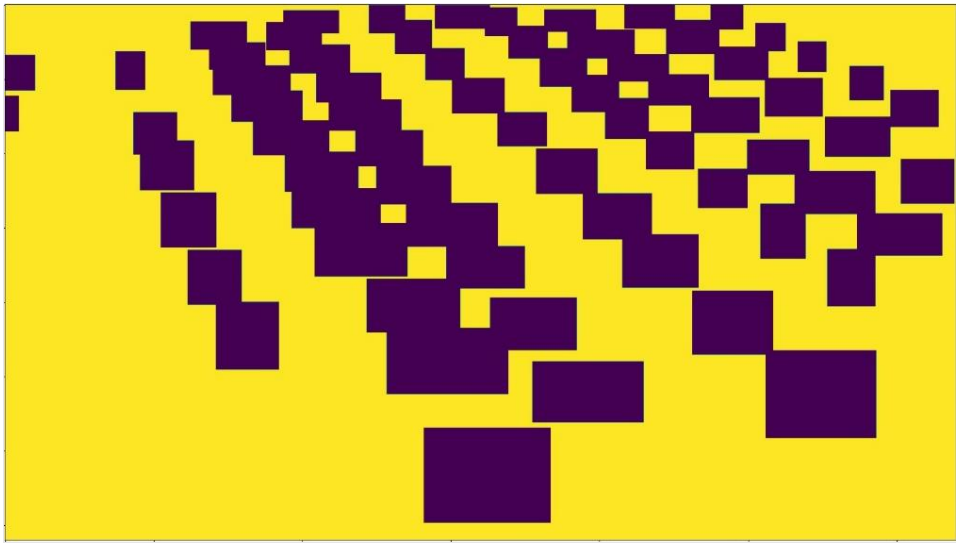
Ime skupa podataka	**kamera**	**trajanje**	**broj slika za testiranje**
UFPRO5_2013-03-13	PKLot 1b kamera	12 sati	145
UFPRO4_2012-12-14	PKLot 2b kamera	13 sati	162
Bay	IP kamera	15 sati	153
Shore	IP kamera	11 sati	130
City	IP kamera	11 sati	136

Za svaki pojedini skup podataka najprije je provedena segmentacija parkirnih mjesta te je onda na istima slikama provjerena uspješnost klasifikatora uz pronađene granične pravokutnike parkirnih mjesta. Važno je napomenuti da su prilikom provjere uspješnosti segmentacije za stvarna (*engl. ground truth*) parkirna mjesta uzeta u obzir samo ona koja su vidljivo označena. Segmentirana parkirna mjesta gdje su se vozila zaista zadržavala, ali nisu vidljivo označena su ignorirana. Iako bi bilo korisno imati informaciju i neoznačenim parkirnim mjestima analiza segmentacije je onda problem jer vozila na takvim mjestima ne parkiraju uvijek pravilno. Analiza uspješnosti klasifikatora je provedena na svim segmentiranim parkirnim mjestima.

Na slikama 4.1.-4.10. prikazana su za svaki skup podataka za testiranje segmentirana parkirna mjesta i *heatmap* matrica.



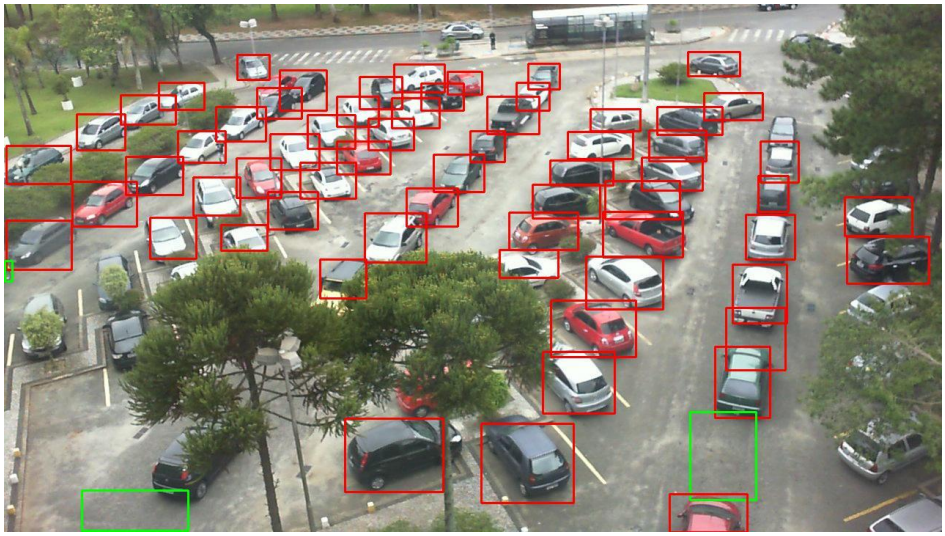
Sl. 4.1. Granični četverokuti za *UFPRO5_2013-03-13* parkirna mjesta nakon segmentacije.



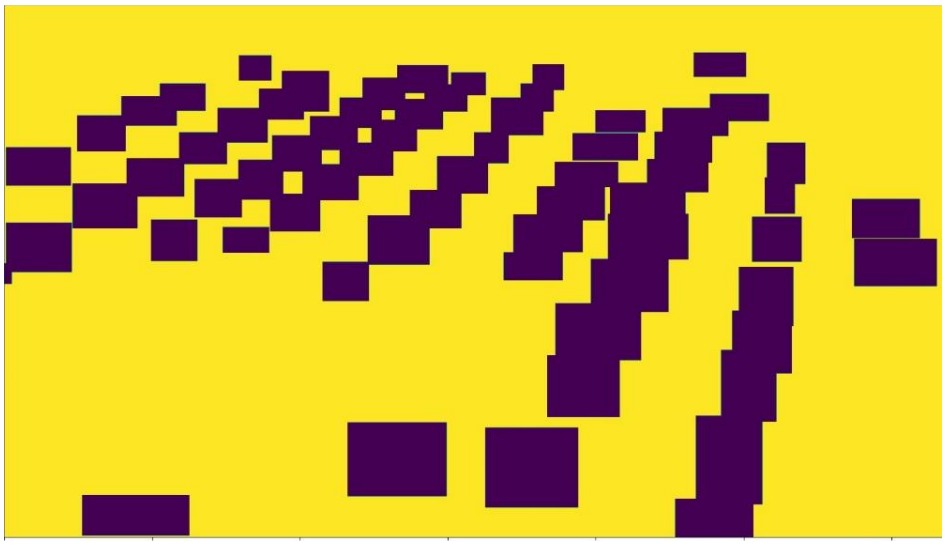
Sl. 4.2. Prikaz vrijednosti *heatmap* matrice *za UFPRO5_2013-03-13* nakon segmentacije.

Primjetno je segmentiranje gotovo svih potpuno vidljivih parkirnih mjesta uz pravilne granične pravokutnike osim u jednom slučaju kada je granični pravokutnik prekrivao dva parkirna mjesta.

Algoritam je detektirao i neoznačena parkirna mjesta na koja su se parkirala vozila (vidi dva zeleno označena slobodna pakirna mjesta na slici 4.1.).

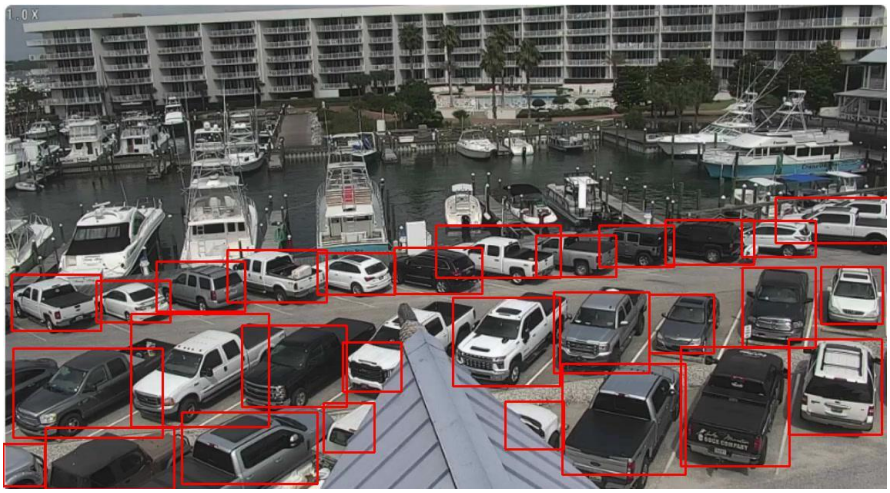


Sl. 4.3. Granični četverokuti za *UFPRO4_2012-12-14* parkirna mjesta nakon segmentacije.



Sl. 4.4. Prikaz vrijednosti *heatmap* matrice za *UFPRO4_2012-12-14* nakon segmentacije.

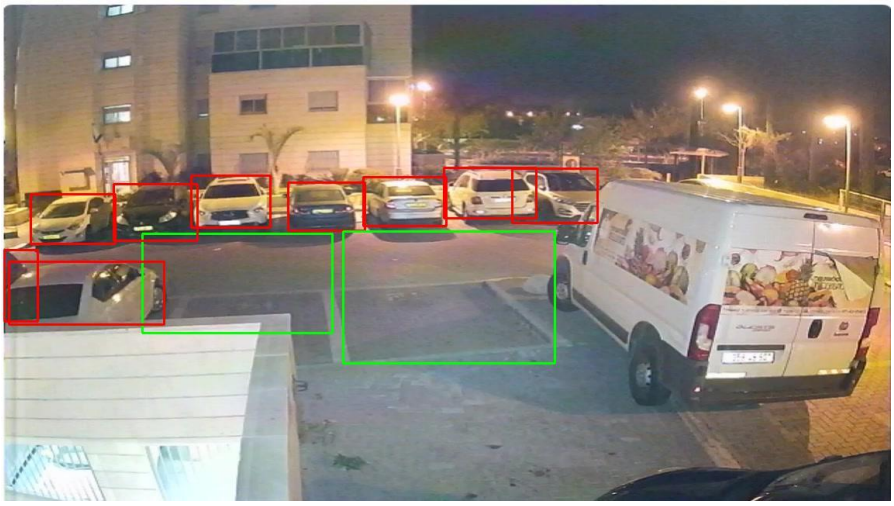
Algoritma je na skupu podataka za testiranje *UFPRO4_2012-12-14* imao problema s parkirnim mjestima koja su bila zaklonjena drvećem što se može primijetiti na slici 4.3.. Takva parkirna mjesta u skupu podataka nisu bila ni označena, a sva označena je algoritam pronašao. Algoritam je detektirao i neoznačena parkirna mjesta na koja su se parkirala vozila.



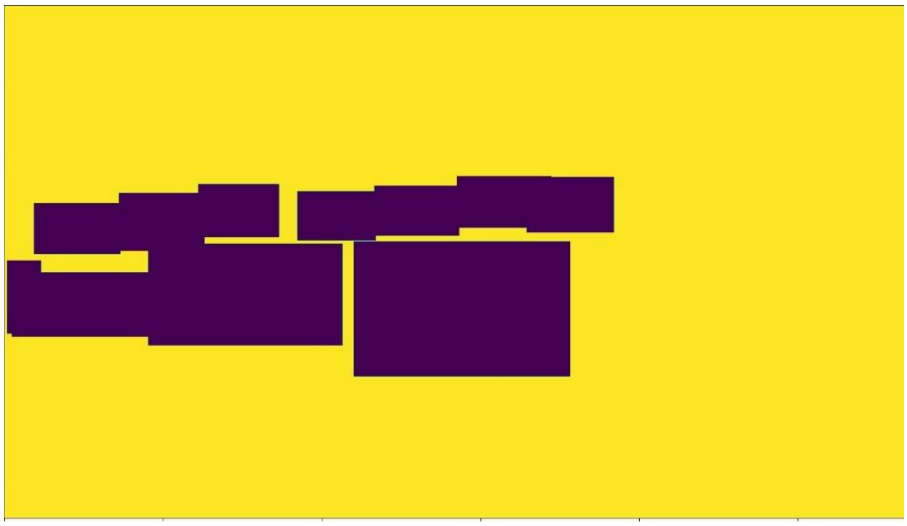
Sl. 4.5. Granični četverokuti za *Bay* parkirna mjesta nakon segmentacije.

Sl. 4.6. Prikaz vrijednosti *heatmap* matrice za *Bay* nakon segmentacije.

Algoritam je u slučaju Bay skupa podataka segmentirao i većinu zaklonjenih vozila ali uz diskutabilne granične pravokutnike što dovodi u pitanje njihov potencijal za uporabu.

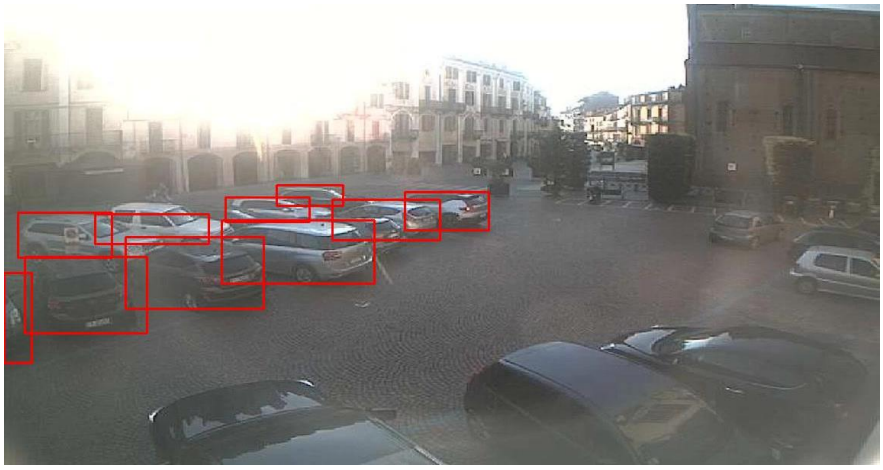


Sl. 4.7. Granični četverokuti za *Shore* parkirna mjesta nakon segmentacije.

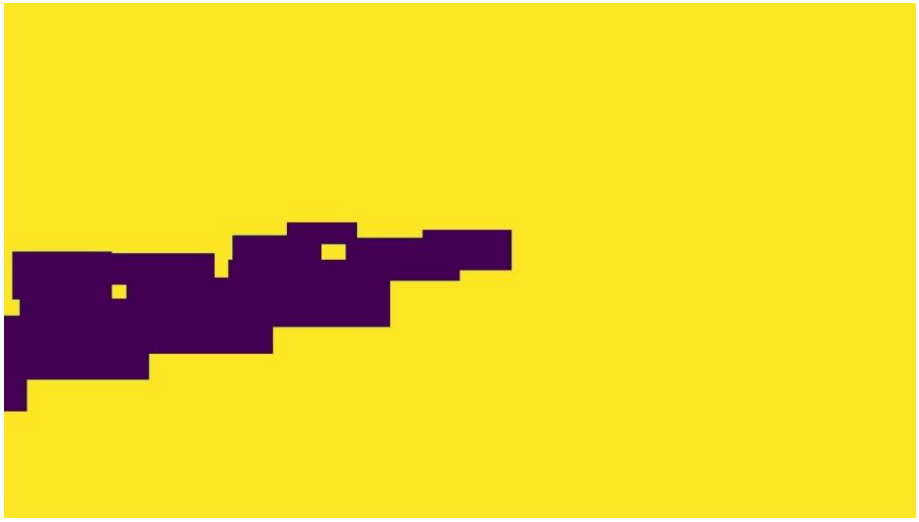


Sl. 4.8. Prikaz vrijednosti *heatmap* matrice za *Shore* nakon segmentacije

Kamera je ovdje postavljena vrlo nisko što je tipično za rezidencijalne nadzorne kamere. Uspješno su segmentirana sva označena parkirna mjesta uz slučaj da jedno ima upitnu upotrebljivost jer parkirno mjesto nije u potpunosti vidljivo.



Sl. 4.9. Granični četverokuti za *City* parkirna mjesta nakon segmentacije.



Sl. 4.10. Prikaz vrijednosti *heatmap* matrice za *City* nakon segmentacije.

City skup podataka je predstavljao izazov za algoritam zbog pozicije parkirnih mjesta u odnosu na poziciju kamere vidljivih na slici 4.9. i nisku kvalitetu snimke. Također velik dio parkiranih vozila je bio zaklonjen drugim parkiranim vozilima ili se dijelovi vozila nisu mogli vidjeti zbog pozicije kamere.

Uspješnost segmentiranja iskazuje se postotkom segmentiranih stvarnih parkirnih mjesta (na temelju pregledavanja snimke od strane osobe). Pregled uspješnosti segmentiranja dan je u tablici 4.2. Segmentiranje se pokazalo vrlo efikasno kada je kamera na dovoljnoj visini i kada parkirna mjesta nisu zaklonjena drugim objektima. Uz lošiju poziciju kamere značajno opada efikasnost segmentacije parkirnih mjesta.

Tab. 4.2. Uspješnost segmentiranja za svaki skup podataka.

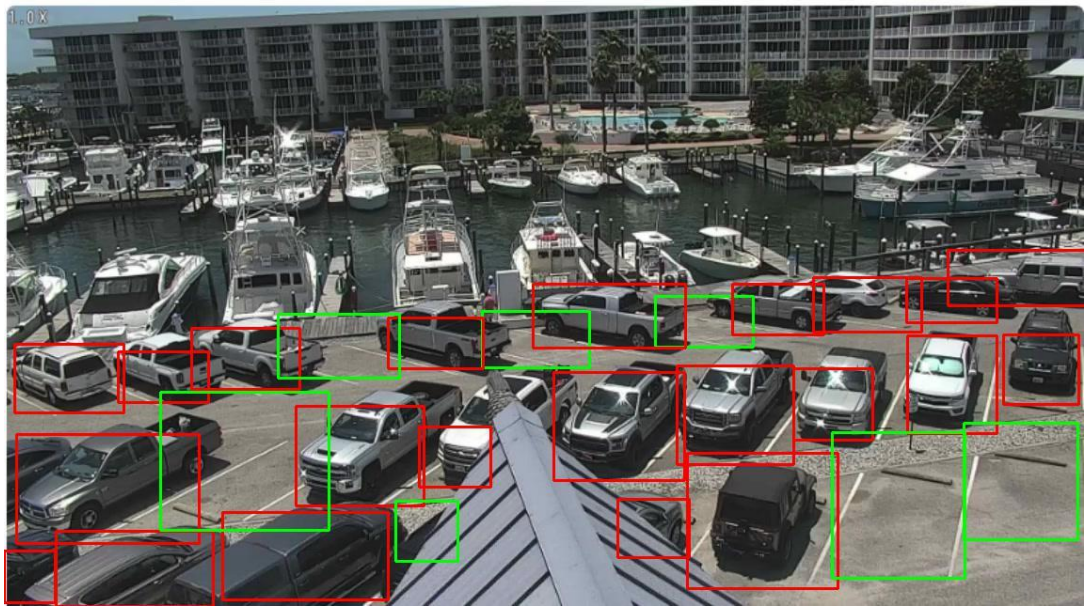
Ime skupa podataka	Broj stvarnih parkirnih mjesta	Broj segmentiranih stvarnih parkirnih mjesta	Postotak
UFPRO5_2013-03-13	42	42	100%
UFPRO4_2012-12-14	49	35	71.4%
Bay	30	29	97%
Shore	11	11	100%
City	20	10	50%

Uspješnost klasifikacije je prikazana mjerama: matrica zabune, točnost (engl. *accuracy*), preciznost (engl. *precision*), opoziv (engl. *recall*) i F1 mjera. Pregled uspješnosti klasifikacije dan je u tablici 4.3.

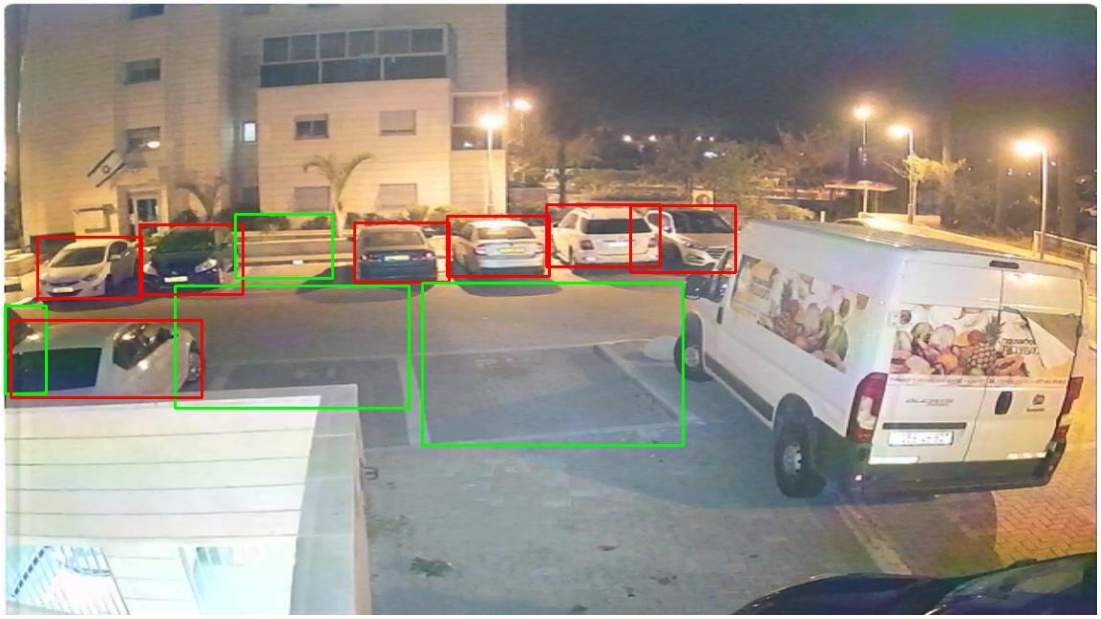
Tab. 4.3 Uspješnost klasifikacije za svaki skup podataka.

Ime skupa podataka	TP	TN	FP	FN	Točnost	Preciznost	Opoziv	F1
UFPRO5_2013-03-13	8304	1637	15	49	0.993	0.998	0.994	0.996
UFPRO4_2012-12-14	7248	3361	43	40	0.992	0.994	0.994	0.994
Bay	4167	192	3	75	0.982	0.999	0.982	0.990
Shore	887	385	81	77	0.889	0.916	0.92	0.918
City	1212	61	6	81	0.936	0.995	0.937	0.965

Klasifikator je imao problema u slučajevima djelomične zaklonjenosti segmentiranog parkirnog mjesta i promjene svjetline odnosno pojave odsjaja koji smanji vidljivost parkirnog mjesta. Također je primijećeno da je za većinu pogreški odgovorna manjina u smislu lošeg uvjeta na rubnom (engl. *edge case*) segmentiranom parkirnom mjestu. Na primjer puno grešaka je napravljeno na parkirnim mjestima koja su zaklonjena poput onog na slici 4.11. ili ne obuhvaćaju cijelo parkirno mjesto poput segmentiranog parkirnog mjesta skroz lijevo na slici 4.12..



Sl. 4.11. FN klasifikacija (sredina dolje) zbog zaklonjenosti krovom.



Sl. 4.12. FN klasifikacija (sredina lijevo) zbog samo djelomične vidljivosti parkirnog mjesta.

5. ZAKLJUČAK

Praćenje zauzetosti parkirališta je važno jer se može primijeniti u sustavu pronalaska slobodnog mjesta i time smanjiti inače potrebno vrijeme i novac. U radu je predložen algoritam za praćenje zauzetosti parkirališta na temelju računalnog vida. Predloženi algoritam sastoji se od dva dijela. Prvo detektor objekata zajedno s vlastitim *PSD* algoritmom segmentira parkirna mjesta iz video signala parkirališta. Nakon segmentacije koristi se klasifikator zauzetosti koji je na temelju segmentacije nadalje u mogućnosti određivati zauzetost parkirališta. Algoritam je implementiran u Python programskom jeziku uz objektno orijentiranu paradigmu te uz pomoć OpenCV i fastai sustava. Za testiranje algoritma je korišteno pet skupova podataka: *UFPRO5_2013-03-13, UFPRO4_2012-12-14, Bay, Shore i City*. Uspješnost segmentacije prikazana je postotkom segmentiranih stvarnih parkirnih mjesta. Uspješnost segmentacije je od 50% uz loše uvjete skupa podataka do 100% uz dobre uvjete poput dobre osvijetljenosti i kvalitete snimke. Uspješnost klasifikacije je za mjeru F1 od 0.9 do 0.996. Segmentiranje parkirnih mjesta predloženim algoritmom pokazuje najveći potencijal kada je pozicija kamere dovoljno visoko u odnosu na pozicije parkirnih mjesta odnosno kada su većina parkirnih mjesta u potpunosti vidljiva bez zaklonjenosti okolišem ili nekim drugim strukturama. Isto vrijedi i za klasifikator vozila koji je pokazao veliku točnost i koji je uklonio daljnju potrebu za detektorom objekata nakon segmentacije.

LITERATURA

[1] Chen, M. and Chang, T. (2011). A parking guidance and information system based on wireless sensor network. In 2011 IEEE International Conference on Information and Automation, pages 601–605.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition (1998). Proceedings of the IEEE, november 1998.

[3] Amato, Giuseppe & Carrara, Fabio & Falchi, Fabrizio & Gennaro, Claudio & Meghini, Carlo & Vairo, Claudio. (2016). Deep Learning for Decentralized Parking Lot Occupancy Detection. Expert Systems with Applications. 72. 10.1016/j.eswa.2016.10.055.

[4] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. (2017). "ImageNet classification with deep convolutional neural networks"

[5] Acharya, Debaditya & Yan, Weilin & Khoshelham, Kourosh. (2018). Real-time image-based parking occupancy detection using deep learning.

[6] Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks"

[7] Almeida, P., Oliveira, L. S., Silva Jr, E., Britto Jr, A., Koerich, A. (2015). PKLot – A robust dataset for parking lot classification. Expert Systems with Applications, 42(11):4937-4949, 2015.

[8] Redmon, Joseph & Farhadi, Ali. (2018). YOLOv3: An Incremental Improvement.

[9] Lin, Tsung-Yi & Maire, Michael & Belongie, Serge & Hays, James & Perona, Pietro & Ramanan, Deva & Dollár, Piotr & Zitnick, C.. (2014). Microsoft COCO: Common Objects in Context. 8693. 10.1007/978-3-319-10602-1_48.

[10] Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun (2015.). Deep Residual Learning for Image Recognition.arXiv preprint arXiv:1512.03385.

[11] Deng, J. and Dong, W. and Socher, R. and Li, L.-J. and Li, K. and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database.

SAŽETAK

Parkirališta su važni dijelovi prometnih središta koji u slučaju neefikasne iskoristivosti troše vrijeme i novac. Moderni *PGI* sustavi rješavaju ovaj problem praćenjem zauzetosti parkirnih mjesta na parkiralištu pomoću računalnog vida. Postojeća rješenja ovog tipa imaju relativno visoke performanse no zahtijevaju ručno označavanje graničnih pravokutnika, odnosno segmentaciju, parkirnih mjesta prije klasifikacije zauzetosti. Algoritam predstavljen u ovom radu omogućuje automatiziranu segmentaciju parkirnih mjesta pomoću detektora objekata i predloženog *PSD* algoritma, a detekciju zauzetosti pomoću binarnog klasifikatora zauzetosti. Segmentacija parkirališta je opisana zadatkom pronalaska graničnih pravokutnika koji odgovaraju parkirnim mjestima. Parkirna mjesta se u predloženom algoritmu promatraju kao 2D površine slike gdje se uzastopno pojavljuju detekcije vozila odnosno gdje se vozila često zadržavaju. Ovo zahtjeva detekciju vozila na slici za što je korišten izgrađeni detektor vozila, a analizu da li se zaista tamo vozila zadržavaju odrađuje *PSD* algoritam. Za detektor vozila je korišten single-shot detektor *YOLOv3* treniran na *COCO* skupu podataka koji služi u procesu segmentacije parkirnih mjesta i tijekom toga prvi prima slike s video kamere. On ima zadaću detektirati sva vozila na primljenoj slici u obliku liste graničnih pravokutnika koju prosljeđuje *PSD* algoritmu. *PSD* algoritam temeljen je na matricu koja ažurira svoje vrijednosti za svaku primljenu listu graničnih pravokutnika od detektora vozila tako da predstavlja svojevrsnu toplinsku kartu zadržavanja vozila na slikama koja služi za segmentiranje parkirnih mjesta. Nakon segmentiranja parkirnih mjesta primijenjen je binarni klasifikator kako bi se odredila zauzetost svakog parkirnog mjesta. Uz klasifikator više nije potreban detektor objekata koji je puno slabije točnosti i brzine izvršenja jer su poznate lokacije svih parkirnih mjesta i samo se želi odrediti da li su zauzeta ili slobodna. Kao osnova klasifikatora korištena je *ResNet34* duboka neuronska mreža uz težine prethodno trenirane na *ImageNet* skupu podataka. *ResNet34* je proširen slojem koji je treniran na podacima iz *PKLot* i *CNRPark-EXT* skupa podataka. Testiranje algoritma je provedeno na 5 skupova podataka. Kod testiranja smo za pojedini skup podataka prvo za sve slike u pojedinom skupu podataka proveli segmentaciju parkirnih mjesta gdje je mjerena uspješnost u obliku postotka segmentiranih parkirnih mjesta. Nakon toga je na istima slikama koje su služile za segmentaciju provjerena uspješnost klasifikatora uz pronađene granične pravokutnike parkirnih mjesta u prethodnom koraku

ABSTRACT. Parking occupancy detection based on computer vision

Parking lots are important parts of commute centers which, if not used efficiently, waste time and money. Modern *PGI* systems solve this problem by tracking parking slot occupancy with the help of computer vision. Existing solutions of this nature have relatively high performance but require hand labeling of bounding boxes for each parking slot before occupancy detection could be done. The algorithm presented in this paper enables automated segmentation of parking slots with the help of a vehicle detector and the proprietary *PSD* algorithm. After that occupancy detection can is done with a binary classifier. With the suggested algorithm parking slots are thought of as 2D picture areas where a vehicle detection is repeatedly located or in other words where vehicles tend to stay still. This requires both detecting vehicles on input pictures for which a vehicle detector is used and an analysis about are detected vehicles stationary. For the vehicle detector a single shot detector *YOLOv3* is used. *YOLOv3* is trained on the *COCO* dataset and is part of the parking segmentation process in which it's the first to receive pictures from a video feed. His task is to detect vehicles in the input picture in the form of a list of bounding boxes which it forwards to the *PSD* algorithm. *PSD* algorithm is based on a matrix which updates its values for each list of bounding boxes received as to represent a sort of a heatmap of where vehicles are stationary which is useful used for segmentation. After segmentation a binary classifier is used to determine the occupancy of each segmented slot. With the classifier in use the vehicle detector which is of lower accuracy is not needed anymore as we know where all parking slots are, and we just want to say are they occupied or not. Backbone of the classifier is the *ResNet34* deep neural network with its weights pretrained on the *ImageNet* dataset. *ResNet34* is expanded with an additional layer which was trained on *PKLot* and *CNRPark-EXT* datasets. Testing of the algorithm is performed on 5 datasets. First, we used each picture in the dataset to segment the parking lot where the result was measured as the percentage of segmented parking spots. The same pictures were then used to test the classifier with the parking spot bounding boxes known.