

SADRŽAJ

1. Uvod	1
2. Osnove prepoznavanja objekata na digitalnim slikama	2
2.1. Korisne informacije u slikama	2
2.2. HOG detektor i metoda potpornih vektora	3
2.2.1. Princip detekcije objekta	3
2.2.2. Princip klasifikacije objekta	4
2.3. Viola Jones detektor	5
3. Neuronske mreže	7
3.1. Motivacija	7
3.2. Umjetni neuron	7
3.2.1. Izgled	7
3.2.2. Učenje	7
3.2.3. Aktivacijska funkcija	8
3.3. Višeslojne mreže	9
3.3.1. Veza između slojeva mreže	9
3.3.2. Propagacija informacije	10
3.4. Izazovi i ograničenja neuronskih mreža	10
4. YOLO	12
4.1. Duboke neuronske mreže	12
4.2. O algoritmu	13
4.3. Verzije algoritma	14
4.3.1. v1	14
4.3.2. v2, v3, v4, v5, v6	16
4.3.3. v7	18
4.3.4. Usporedba verzija	20

5. Programsко rješenje	21
5.1. Skup podataka	21
5.2. Odabir programskog jezika, knjižnica i okruženja	21
5.3. Programski kod	22
5.3.1. Rješenje pomoću verzije 1 Yolo algoritma	22
5.3.2. Rješenje pomoću verzije 7 Yolo algoritma	27
6. Zaključak	29
Literatura	31

1. Uvod

Aktualna tema koja obilježava današnji svijet tehnologije, bilo da se radi o istraživanju ili primjeni u industriji, jest ostvarivanje efikasnije zamjene čovjeka računalnim resursima, kako postojećim, tako i onima koji će tek nastati. Jedno od područja koje se intenzivno razvija i neprestano unapređuje je razvoj računalnog vida s ciljem zamjene ljudskog vida na novoj razini. Računalni vid je tehnologija koja je implementirana u različitim područjima, uključujući identifikaciju vozila, analizu medicinskih slika, praćenje prometa na cestama te sve do područja robotike u kojem se vozila mogu autonomno kretati bez potrebe za ljudskim upravljanjem.

Detekcija objekata jedna je od tehnika računalnog vida koja se koristi za lokaciju i klasifikaciju objekata u slikama ili videozapисima. Locirati te potom klasificirati objekte, kompleksna je tema koja je bila svjedok revolucionarnim promjenama u grani računalnog vida. Istraživači su prvo krenuli detektirati objekte pomoću oblika, histograma i određenim značajkama koje su određenim matričnim ili vektorskim metodama uspijevali postići te kroz daljni razvoj došli do razvoja neuronskih mreža koje se trenutno pokazuju kao najsnaznije sredstvo za naveden problem.

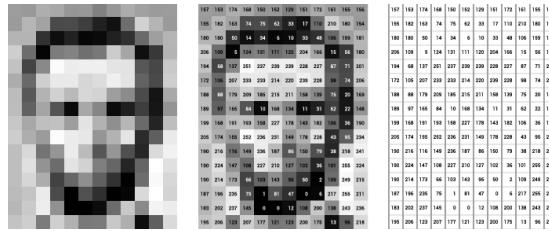
U radu su spomenute osnovne tehnike i **koncepti detekcije objekata**, glavna obilježja te **karakteristike neuronskih mreža** te shodno tome pomno objašnjena i implementirana, na skup slika iz automobilske vožnje, tehnika **YOLO (You Only Look Once) algoritma** koja je bazirana na dubokim neuronskim mrežama.

Isti taj algoritam, verzije 1 i 7, rješenje je problema te su uspoređene vrijednosti dobivene iz oba. Algoritam je kroz godine nadograđivan te je zadnji razvoja trenutno na verziji 8.

2. Osnove prepoznavanja objekata na digitalnim slikama

2.1. Korisne informacije u slikama

Pri pristupu analizi slike u računalnom vidu, ključno je razumjeti osnovne karakteristike slika i načine na koje možemo koristiti te informacije. Slika se može promatrati kao skup piksela, odnosno kao matrica u dvije dimenzije, gdje svaki piksel predstavlja najmanju jedinicu informacije. Takav komadić informacije nosi vrijednost intenziteta dok širina i visina slike određuju dimenzije matrice koja je oblikovana. Uz to, moguće je raditi s više kanala boje, poput crvene, zelene i plave (RGB) ili drugih modela koji koriste različite komponente boje. Ovaj način prikaza slike kao matrice omogućuje primjenu matematičkih i statističkih operacija za analizu i obradu.



Slika 2.1: Prikaz slike kao skup piksela

<https://ai.stanford.edu/~syueung/cvweb/tutorial11.html>

Visoke rezolucije slika stvaraju izazove u obradi zbog velikog broja piksela. Smanjenje dimenzionalnosti je ključno, a jedan od načina je ekstrakcija rubova - mesta s naglim promjenama intenziteta piksela. To omogućuje prepoznavanje oblika i objekata u slici. Reduciranje dimenzionalnosti i ekstrakcija rubova olakšavaju brzu i preciznu obradu slike.

2.2. HOG detektor i metoda potpornih vektora

2.2.1. Princip detekcije objekta

Prije dvadeset godina, rano razvijeni algoritmi za detekciju objekata temeljili su se na ručno kreiranim značajkama zbog nedostatka učinkovitih metoda za reprezentaciju slika. Jedan od takvih algoritama je HOG detektor. HOG (Histogram of Oriented Gradients) je algoritam za detekciju objekata koji se koristi za analizu i isticanje glavnih područja na slici.

HOG algoritam je originalno predložen 2005. godine od strane znanstvenika koji su radili na području strojnog učenja i računalnogvida u Francuskoj. Temelji se na ideji korištenja blokova predefinirane veličine koji prolaze poput kliznih prozora po slici i skupljaju informacije.

Da bi se dobio HOG vektor značajki, ulazna slika prvo se mijenja u određenu veličinu, a zatim se blokom predefinirane veličine iterativno prolazi kroz redove i stupce slike. Blok se sastoji od manjih dimenzija, a svakim korakom se izračunavaju gradijenți magnitudo i smjera. Za svaku celiju bloka, na primjer, 2x2, generira se vektor značajki veličine 9 koji predstavlja histogram koji prikazuje smjer i intenzitet piksela. HOG detektor višestruko mijenja veličinu ulazne slike kako bi mogao detektirati objekte različitih veličina, dok veličina prozora detekcije ostaje nepromijenjena. [4]



Slika 2.2: Značajke dobivene HOG detektorom

[https://www.analyticsvidhya.com/blog/2019/09/
feature-engineering-images-introduction-hog-feature-descriptor/](https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/)

HOG je postao poznat po svojoj upotrebi u detekciji pješaka.

Nakon analize slike HOG detektorom, generiraju se HOG vektori značajki koji predstavljaju pojednostavljenu reprezentaciju slike. Ti vektori se zatim koriste kao ulazni podaci u metodi potpornih vektora za klasifikaciju ili detekciju objekata.

2.2.2. Princip klasifikacije objekta

Za prepoznavanje točnih klasa objekta koje detektiramo koristi se Metoda potpornih vektora, SVM algoritam. SVM je supervizirani algoritam strojnog učenja koji se koristi za klasifikaciju i regresiju. Pomaže u razlikovanju i kategorizaciji objekata na temelju značajki koje su izračunate iz HOG vektora. Međutim, za shvaćanje SVM klasifikatora nužno je upoznati se s određenim pojmovima.

Linearna decizijska funkcija igra ključnu ulogu u algoritmu koji se temelji na konstrukciji hiperravnina (linearnih granica) koje razdvajaju različite klase podataka.[6]

U kontekstu klasifikacije, linearna decizijska funkcija je funkcija koja određuje na kojoj strani hiperravnine se nalazi određeni primjer podataka. Ako je rezultat linearne decizijske funkcije pozitivan, primjer će biti klasificiran u jednu klasu, dok će biti klasificiran u drugu klasu ako je rezultat negativan.

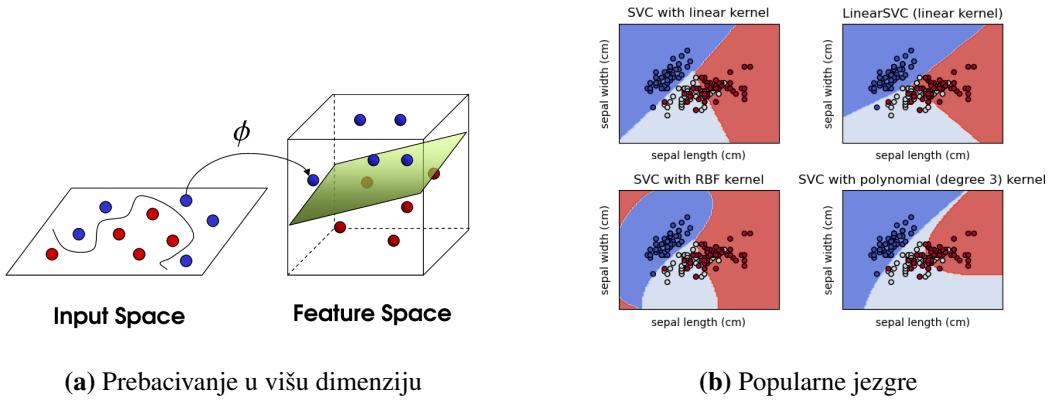
Matematički, linearna decizijska funkcija se može predstaviti kao:

$$f(x) = w * x + b$$

U ovoj jednadžbi, w je težinski vektor, x je ulazni vektor značajki primjera podataka, a b je pomak (engl. bias). Cilj je pronaći optimalne vrijednosti za težinski vektor i pomak kako bi se postigla što bolja klasifikacija. Algoritam nastoji pronaći takav skup parametara (težinski vektor i pomak) koji maksimizira udaljenost između najbližih primjera dvije klase (potpornih vektora) i istovremeno minimizira pogreške klasifikacije. To se postiže rješavanjem optimizacijskog problema koji ima za cilj minimizirati težinski vektor w uz odgovarajuće ograničenje na marginu, koordinatnu udaljenost, između klasa. Optimizacija može biti osnovana na običnom penaliziranju ili nagrađivanju s određenom konstantom ili nekom složenijom tehnikom.

Problem stvarnog svijeta je taj da rijetko postoje podaci koji su potpuno linearно separabilni. Tada na snagu nastupaju jezgre koje će nižu dimenziju prostora prebaciti u višu.

Osim korištenja jezgri postoji mogućnost uvođenja *Soft Margin SVM*, a ne *Hard Margin* koji je predstavljen u prošlom odlomku. Dok *Hard SVM* zahtijeva da su podatci savršeno linearно separabilni odnosno da postoji jasna granica s udaljenosti većom od 1 od granice odlučivanja te shodno tome ne tolerira pogreške. U *Soft SVM*-u uvodi se dodatan parametar *slabe variable* kako bi se dopustilo primjerima da budu unutar margina ili čak na pogrešnoj strani granice uz odgovarajuće kazne. [3]

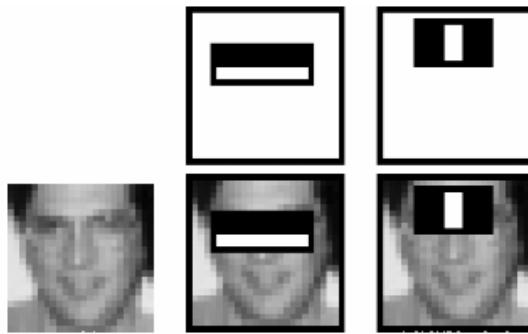


Slika 2.3: Korištenje jezgri

<https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f>

2.3. Viola Jones detektor

Još jedan detektor baziran na matematičkim principima je Viola-Jones detektor, također poznat kao i *Haar-Cascade* detektor. Razlog zašto se *Haar-Cascade* krije u istoimenim značajkama koje koristi za detekciju lica u stvarnom vremenu pomoću efikasnog i brzog *AdaBoost* algoritma. [4] Haar značajke su predefinirane slike s različitim crno-bijelim oblicima koji se primjenjuju na sliku kako bi pomoću razlike intenziteta piksela između susjednih prostornih regija detektirali moguća svojstva ljudskog lica. Jedna značajka može biti odgovorna za pronađak obrisa nosa dok druga za pronađak očiju iznad otkrivenog nosa. U slučaju da značajka ne uspije detektirati određen pojmom za koji je zadužena prestaje s procesom te se taj dio odbacuje, što omogućuje brzu obradu slika. Iterativnim postupcima koji korigiraju veličinu Haar značajki te mesta primjene nastavlja se algoritam.



Slika 2.4: Haar značajke

https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

Paul Viola i Michael Jones razvili su algoritam 2001. godine te ga objavili u radu *"Rapid Object Detection using a Boosted Cascade of Simple Features"* te revolucionirali metodu detekcije lica uvodeći puno veću brzinu same izvedbe od dotadašnjih.

3. Neuronske mreže

3.1. Motivacija

Prethodni modeli i algoritmi nisu uvijek bili dovoljno dobri za rješavanje složenih i teških zadataka. **Složenost obrazaca i nestrukturiranost podataka** skupa s lošom skalabilnosti nagnalo je istraživače za pronačinak rješenja što **sličnijem samom ljudskom mozgu** te njegovim karakteristikama i sposobnostima. Neuronske su se mreže iskazale kao **sustavi** koji mogu **obrađivati više ulaznih podataka te poboljšavati svoje performanse** na temelju primljenih podataka.

3.2. Umjetni neuron

3.2.1. Izgled

Funkcija prirodnog neurona je da prima informaciju od drugih neurona te je izmijenju prenosi dalje na druge neurone. Pojednostavljeni model prirodnog neurona isto prenosi informaciju te je taj prijenos interpretiran kao funkcija.

$$y = f \left(\sum_{i=1}^n w_i \cdot x_i + b \right)$$

Osnovne komponente matematičkog modela umjetnog neurona su ulazni podatci x_i , težine w_i koje održavaju važnost ulaznih podataka te pomak b . Ukupna suma zbrojena s pomakom ulazi u aktivacijsku funkciju koja generira izlaz neurona. Aktivacija funkcija je opisana detaljnije u nastavku.

3.2.2. Učenje

Iterativnim mijenjanjem parametara neurona ovisno o dobivenom izlazu iz funkcije te očekivanom izlazu učimo neuron da preciznije računa izlaznu vrijednost. Pogreška

izlazne vrijednosti može biti određena samom razlikom.

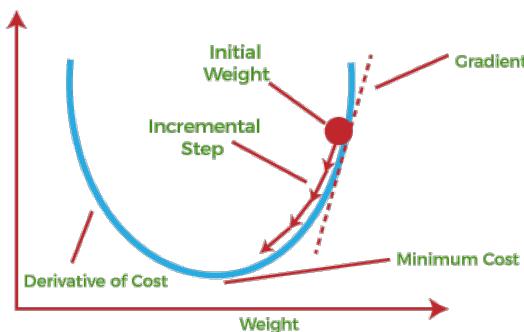
$$e = d - y = d - f \left(\sum_{i=1}^n w_i x_i + b \right)$$

Za dobro učenje potrebno je minimizirati pogrešku te je osnovan princip takve optimizacije slijedeće gradijenta. Gradijent se računa parcijalnim derivacijama funkcije greške s obzirom na odabrani parametar. Gradijent je pomnožen faktorom učenja (eng. learning rate $-lr$) koji je potrebno pomno izabrati jer stvara veliki utjecat na samo učenje. Pretežito se bira konstanta između 0.1 i 0.0001 ili se koristi planer učenja (eng. learning scheduler) koji mijenja faktor ovisno o broju epohe.

$$\Delta w_i = -\eta \frac{\partial e}{\partial w_i}$$

$$w_i = w_i + \Delta w_i$$

Gradijent pokazuje u smjeru povećanja pogreške te je shodno tome način promjene parametara poznat. Slijedeći gradijent krećemo se u smjeru povećanja pogreške te nam je cilj doći u stanju gdje je gradijent jednak nuli. U novijim modelima se ne koristi običan izračun gradijenta nego SGD (stohastički) gradijentni spust koji prolazi kroz primjere jedan po jedan te se gradijent računa za svaki pojedinačni primjer te po tome ažuriraju parametri.



Slika 3.1: Prikaz gradijenta na 2D krivulji

<https://www.javatpoint.com/gradient-descent-in-machine-learning>

3.2.3. Aktivacijska funkcija

Aktivacijska funkcija je ključna komponenta neuronskih mreža i igra važnu ulogu u modeliranju kompleksnih nelinearnih odnosa između ulaza i izlaza neurona. Njena uloga je da unese nelinearnost u neuronsku mrežu, što omogućuje modeliranje složenih funkcija i rješavanje složenih problema. Ovisno o potrebi, koriste se različite

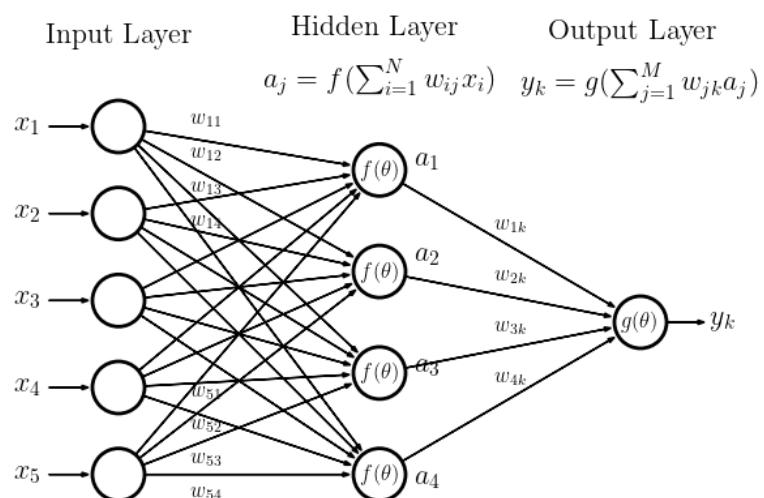
aktivacijske funkcije. YOLO algoritam koristi aktivacijsku funkciju *Leaky ReLU* koja je varijacija standardne ReLU (Rectified Linear Unit) uz dodavanje malog nagiba za negativne vrijednosti. ReLU je linearna funkcija $y=x$ za ulazne vrijednosti veće od 0. Glavni razlog za korištenje *Leaky ReLU* u YOLO algoritmu je njegova sposobnost borbe protiv problema *dying ReLU*. Uobičajena *ReLU* funkcija ima vrijednost 0 za sve negativne ulaze, što znači da će neuroni koji aktiviraju negativne vrijednosti prestati učiti tijekom postupka propagacije unatrag.

3.3. Višeslojne mreže

3.3.1. Veza između slojeva mreže

Za rješavanje kompleksnih problema potrebne su i kompleksne višeslojne mreže od povezanih neurona. Osnovan način povezivanja neurona unutar višeslojne mreže temelji se na potpunim vezama. U potpunim je vezama svaki neuron u jednom sloju povezan sa svakim neuron u prethodnom sloju, što znači da nema dijeljenja težina između različitih dijelova slike. Višeslojne mreže obično se sastoje od **tri glavna sloja**: ulaznog sloja, skrivenih slojeva i izlaznog sloja. Ulazni sloj prima podatke na početku, skriveni slojevi obrađuju te podatke, a izlazni sloj daje konačni rezultat ili predikciju.

[2]



Slika 3.2: Troslojna neuronska mreža

[https:](https://www.astroml.org/astroML-notebooks/_images/fig_neural_network-1.png)

//www.astroml.org/astroML-notebooks/_images/fig_neural_network-1.png

3.3.2. Propagacija informacije

Glavni koncept višeslojnih mreža je propagacija informacije unaprijed (*feedforward*) i povratna propagacija (*backpropagation*). Unaprijedna propagacija proslijeđuje podatke kroz mrežu te računa izlaz iz mreže dok povratna koristi određen algoritam optimizacije kako bi ažurirala težine i poboljšala točnost modela. Koraci učenja mreže i propagacije unatrag se ponavljaju za više iteracija ili epoha kako bi se postupno poboljšala kvaliteta modela.

3.4. Izazovi i ograničenja neuronskih mreža

Uz veliku moć neuronskih mreža postoje i problemi koje je potrebno zaobići. Osim najgoreg ishoda u kojem mreža ništa ne uči, postoji mogućnost u kojoj mreža uči dobro, ali samo na početnom skupu podataka koji se koristio za treniranje. Taj pojam se naziva **generalizacija** te nastaje zbog treniranja na prevelikom i sličnom skupu podataka. Izbjegavanje generalizacije postiže se dobro isplaniranim načinom učenja mreže, stratifikacijom, s pomno izabranim podatcima te ograničenjem količine s kojom ćemo trenirati našu mrežu. Za poboljšanu kontrolu učenja mreže, postoji mogućnost korištenja skupa za validaciju.

Nakon što je model uspješno validiran i optimiziran, sljedeći korak je testiranje. Testiranje se provodi na zasebnom testnom skupu podataka koji model nije procesirao tijekom treninga niti validacije. Ovo je ključni korak za procjenu stvarne performanse modela na potpuno novim podacima.

4-fold validation (k=4)



Slika 3.3: Popularna tehnika treniranja i testiranja podataka

<https://es.mathworks.com/discovery/cross-validation.html>

Jedna od popularnih tehnika spriječavanja pretreniranja je *dropout* tehnika koja slučajno odbacuje određeni postotak neurona tijekom treninga. To znači da tijekom svakog prolaza kroz podatke, određeni neuroni neće biti aktivirani i neće sudjelovati u

izračunu. Ovo prisiljava model da nijedan neuron ne postane prejako vezan za određeni skup podataka.

U praksi se također koristi **augmentacija**, tehnika s kojom se koriste razne tehnike kako bi se generirali novi primjeri podataka. U kontekstu slika, to može uključivati rotaciju, zrcaljenje ili primjenu različitih filtera.

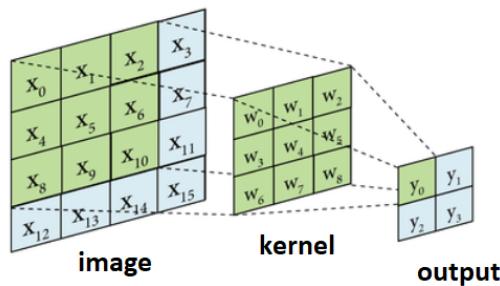
Potpuni povezani slojevi imaju nedostatak pri izdvajaju značajki iz slika jer ne uzimaju u obzir prostornu strukturu slike. Ovi slojevi zahtijevaju fiksiranu veličinu ulaznih podataka, što znači da slike moraju biti prethodno pretvorene u vektor fiksne duljine prije nego što ih se može proslijediti kroz potpuno povezane slojeve.

S druge strane, **konvolucijski slojevi**, inspirirani konceptom konvolucije u matematici, generiraju mapu značajki te na taj način provode ekstrakciju određenih značajki iz slike koje su bitne za detekciju. Međutim, potpuno povezani slojevi nemaju vještinu razumijevanja prostorne strukture podataka. Oni ne dijele težine između različitih dijelova ulaznog vektora, pa model mora naučiti sve kombinacije značajki neovisno o njihovom prostornom položaju. Ovo može dovesti do prekomjernog učenja i prenaučenosti na specifične uzorke u trening skupu, umjesto da nauče generalizirane značajke. Ostala obilježja te dodatne mogućnosti neuronskih mreža biti će objašnjena u nastavku rada ovisno o potrebi.

4. YOLO

4.1. Duboke neuronske mreže

Duboke neuronske mreže (engl. Deep Neural Networks - DNN) razlikuju se od uobičajenih neuronskih mreža s jednim skrivenim slojem po svojoj dubini, koja se odnosi na broj slojeva čvorova kroz koje podaci moraju proći u višekoraknom procesu prepoznavanja uzorka. Više od tri sloja, uključujući ulazni i izlazni sloj, se smatra *dubokim* učenjem. U dubokim neuronskim mrežama mogu se pronaći razne vrste slojeva od kojih su među pretežitim konvolucijski, slojevi za grupnu normalizaciju te na kraju i potpuno povezani.



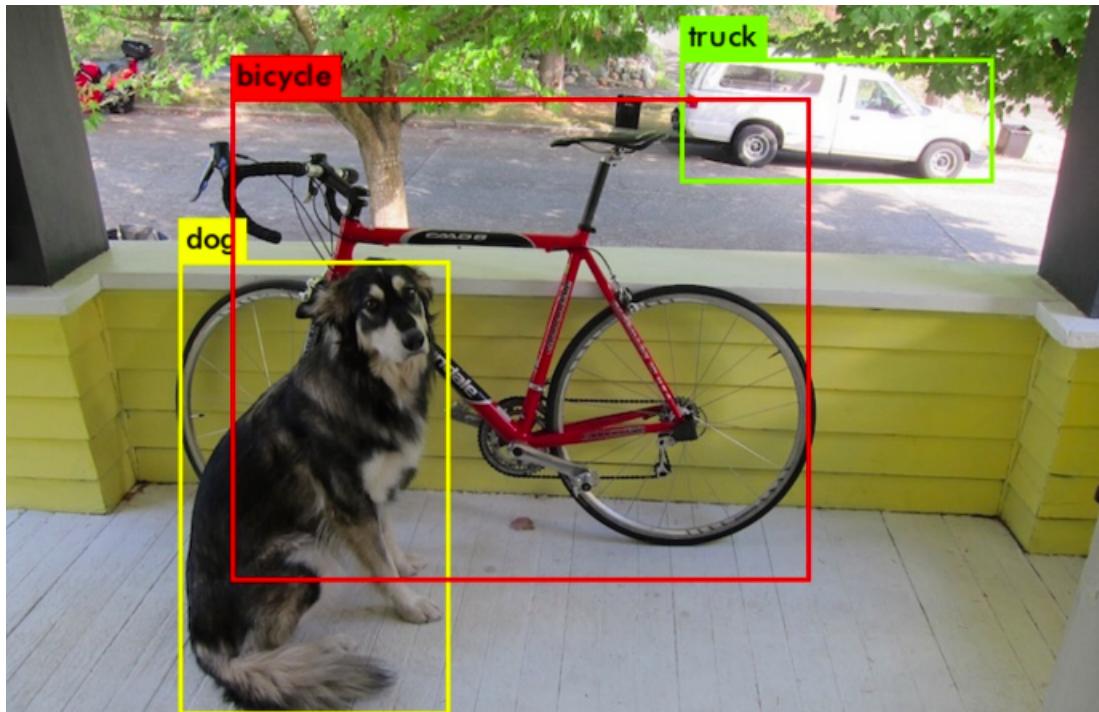
Slika 4.1: Prikaz ekstrahiranja podataka u konvoluciji

https://miro.medium.com/v2/resize:fit:640/format:webp/1*un5WgQMwfLw9Pi_I4qWFqg.png

Konvolucijske mreže sastoje se od više konvolucijskih slojeva koji se mogu kombinirati s drugim slojevima, poput slojeva sažimanja (*pooling*) i već gore navedenih, *slojeva normalizacije i potpuno povezanih slojeva*. Ovi slojevi omogućuju modelu da postepeno uči složenije i apstraktnije značajke kako se dubina mreže povećava. Konvolucijski slojevi koriste skup filtera (emphjezgri) koje primjenjuju na sliku kroz proces klizanja. Svaka jezgra konvolucije obrađuje manji lokalni dio slike i generira izlaz, mapu značajki. Invarijantnost na pomak ključna je karakteristika konvolucijskih slojeva pomoću kojih neuronskoj mreži daje korist u prepoznavanju značajki bez obzira

na njihovu točnu poziciju u slici. Slojevi sažimanja imaju svrhu smanjivanje ili povećavanja dimenzionalnosti i rezolucije podataka. Sloj za smanjivanje uzorka (eng. **downsampling**) izведен je pomoću max-pooling slojeva koji dijeli ulazne podatke u manje regije i zatim izvlači maksimalnu vrijednost iz svake. Proces uvećavanja dimenzije uzorka obično se provodi koristeći slojeve za unaprijeđivanje (eng. **upsampling** transposed convolution layers). Ovi slojevi koriste konvolucijsku operaciju kako bi se povećala dimenzionalnost i rezolucija podataka. Oni su obrnuta operacija slojeva za konvoluciju i omogućuju povećavanje veličine podataka na temelju interpolacije i obnavljanja informacija.[2]

4.2. O algoritmu



Slika 4.2: Rezultati YOLO algoritma

[https://www.inspiritai.com/blogs/ai-student-blog/
object-detection-the-yolo-algorithm-in-ai](https://www.inspiritai.com/blogs/ai-student-blog/object-detection-the-yolo-algorithm-in-ai)

J. Redmon, S. Divvala, R. Girshick i A. Farhadi, 2015.godine, predstavljaju rad s kojim predstavljaju ideju i učinkovitost YOLO algoritma.

"Predstavljamo YOLO, novi pristup detekciji objekata. Dosadašnji rad na detekciji objekata koristi prerađene klasifikatore za obavljanje detekcije. Umjesto toga, mi postavljamo detekciju objekata kao problem regresije za prostorno odvojene okvire i

povezane vjerojatnosti razreda. Jedna neuronska mreža izravno predviđa okvire i vjerojatnosti razreda iz cijelih slika u jednom evaluacijskom koraku. Budući da je cijeli postupak detekcije u jednoj mreži, moguće ga je optimizirati end-to-end izravno na temelju performansi detekcije. Naša ujedinjena arhitektura je izuzetno brza. Naš osnovni YOLO model obrađuje slike u stvarnom vremenu sa 45 sličica u sekundi." [5]

Prije YOLO algoritma, najučinkovitiji sustav detekcije slika bio je R-CNN (*Region Based Convolutional Neural Networks*) baziran na potencijalnim regijama (eng. region proposals) koje mogu sadržavati objekte. Za generiranje tih regija mogu se koristiti različite tehnike, kao što su selektivna pretraga (eng. selective search) ili algoritmi temeljeni na regijama (eng. region-based algorithms). U R-CNN, više prolazno učenje i obrada pojedinačnih regija odvija se u tri glavne faze: generiranje regija, izvlačenje značajki i klasifikacija/regresija. Problem koji je R-CNN imao, skupa sa kasnije nadograđenim Fast R-CNN algoritmom, bio je zahtjev za više prolaza i obrada pojedinačnih regija, dok YOLO algoritam **čitavu detekciju** vrši u **jednom prolazu** kroz neuronsku mrežu. [1] Razlog zbog kojeg je bilo potrebno više prolaza leži u raznolikosti regija koje mogu imati različite veličine i razmjere. Konvolucijske mreže zahtijevaju ulaz fiksne veličine, stoga nije moguće izravno proslijediti regije različitih veličina kroz takvu mrežu. Kako bi se regije uskladile s ulazom konvolucijske mreže, one moraju biti izdvojene i promijenjene u veličini. Ova prilagodba omogućuje konzistentnu obradu različitih regija unutar mreže.

4.3. Verzije algoritma

4.3.1. v1

Mreža algoritma je iznimno jednostavna i brza u upotrebi. Paralelno predviđa više graničnih okvira (eng. **bounding boxes**) i vjerojatnosti klase za te okvire, stoga se tretira kao problem regresije. YOLO se trenira na cijelokupnim slikama i izravno optimizira performanse detekcije.

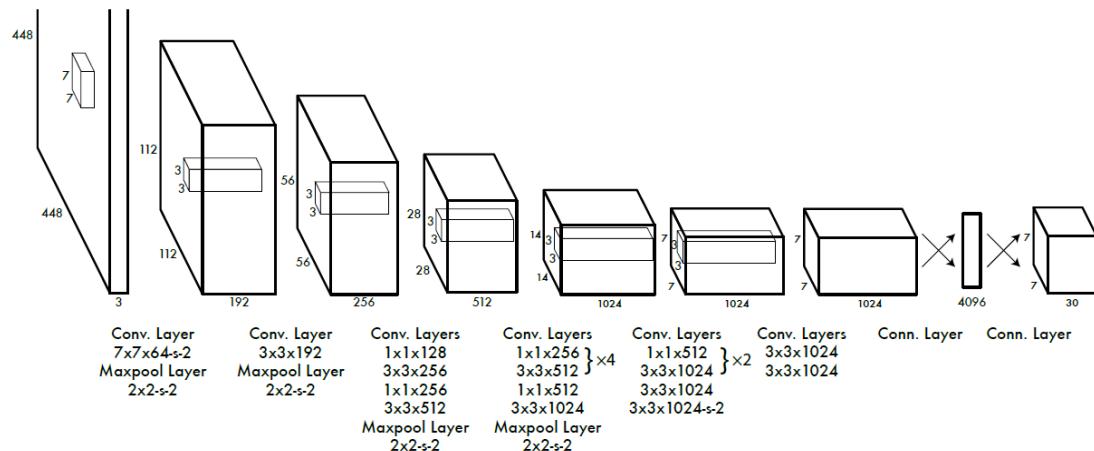
Arhitektura prvog YOLO algoritma sastoji se od 24 konvolucijska sloja i 2 potpuno povezana sloja. Temeljni dio, koji se sastoji od konvolucijskih slojeva, naziva se **backbone**. Taj dio obavlja većinu računanja i ekstrakciju značajki iz ulaznih podataka te podržava ostatak arhitekture mreže. U prvoj verziji algoritma, ovaj dio mreže je nazvan Darknet-19. [5]

Backbone je odgovoran za prolazak ulaznih podataka kroz niz slojeva konvolucije,

uzorkovanja (*pooling*), normalizacije i aktivacije. Ti slojevi omogućuju mreži da nauči hijerarhijske značajke složenih oblika i uzoraka prisutnih u podacima. Mreža koristi različite konvolucijske slojeve i na određenim mjestima primjenjuje *max-pooling* slojeve veličine 2x2 s korakom (eng. stride) veličine 2. Korak predstavlja broj piksela koji se preskače prilikom izračunavanja izlaza konvolucije.

Ulaz u mrežu očekuje RGB sliku veličine 448x448, dok se na izlazu dobiva matrica veličine 7x7x(B*5+C). Prve dvije dimenzije opisuju 7 vertikalnih i 7 horizontalnih presjeka slike kako bi se slika podijelila na ćelije radi jednostavnije obrade. Zadnja dimenzija matrice opisuje predikciju za svaku ćeliju. Oznaka *B* predstavlja broj predviđenih okvira objekata po ćeliji, dok oznaka *C* predstavlja broj klasa koje se mogu pojaviti na slikama.

Na primjer, ako se prepostavi da se mogu pojaviti dva objekta po ćeliji, za koje se očekuju dvije klase (npr. čovjek i automobil), ukupna veličina zadnje dimenzije će biti 12. Prve dvije vrijednosti opisuju klase, treća vrijednost predstavlja povjerenje za prvi mogući okvir, dok sljedeće četiri vrijednosti opisuju koordinatne veličine *x*, *y*, *w*, *h* tog okvira. Zatim slijedi vrijednost povjerenja za drugi mogući okvir, praćena koordinatnim vrijednostima.



Slika 4.3: Arhitektura YOLO v1

https://miro.medium.com/v2/resize:fit:1100/format:webp/1*q5feieiezWKYq7dpWjYvC0w.png

Funkcija gubitka se računa koristeći kombinaciju nekoliko komponenata.

Prva komponenta opisuje gubitak lokalizacije, koji mjeri koliko dobro YOLO model predviđa koordinate okvira objekata unutar svake ćelije. Za svaki okvir u svakoj ćeliji, računa se kvadratna razlika između predviđenih koordinata *x*, *y*, *w*, *h* i stvarnih koordinata ciljnog okvira. Vrijednosti širine i visine se korijenjuju kako bi se sma-

njio utjecaj velikih objekata nad malim. Razlika vrijednosti se zatim množi faktorom težine lokalizacije koji pridaje veći utjecaj gubitka lokalizacije nad drugim sporednjim komponentama. Gubitak lokalizacije se sumira za sve okvire i ćelije. YOLO vrši predikciju većeg broja okvira objekata po ćeliji te pomoću računanja IoU (*Intersection over Union*) dodijeljuje samo jednom prediktoru "odgovornost" prema svojem objektu. Ako dva okvira imaju IoU veći od 0.5, objektu koji se detektira se pridružuje samo onaj okvir s boljom predikcijom.

Gubitak povjerenja, druga je komponentna koja mjeri koliko dobro model predviđa povjerenje svakog okvira u odnosu na prisutnost objekta. Gubitak povjerenja se računa kao kvadratna razlika između predviđenog povjerenja i stvarne oznake prisutnosti objekta (1 za prisutnost, 0 za odsutnost) za svaki okvir. Ova komponenta uključuje faktor težine povjerenja veličine 5 dok za gubitak povjerenja onih okvira objekata koji ne postoje koristimo koeficijent vrijednosti 0.5.

Treća komponenta funkcije gubitka je **gubitak klasifikacije**, koji mjeri koliko dobro YOLO model predviđa klasu svakog objekta. Funkcija prolazi po svim ćelijama te za svaku iterira po klasama te izvršava kvadratnu razliku previđene i stvarne klase. Funkcija gubitka kažnjava samo klasifikaciju grešaka ako je objekt prisutan u toj mrežnoj ćeliji. Također kažnjava samo pogrešku koordinate graničnog okvira ako je taj prediktor "odgovoran" za istinit granični okvir.

Konačna funkcija gubitka se dobiva zbrajanjem svih triju komponenata gubitka. Da bi se uravnotežio utjecaj svake komponente, može se primijeniti skaliranje gubitka lokalizacije i gubitka povjerenja. Takva funkcija gubitka ulazi u optimizacijski postupak koji se odrađuje gradijentnim spustom s odabranim pogodnostima.

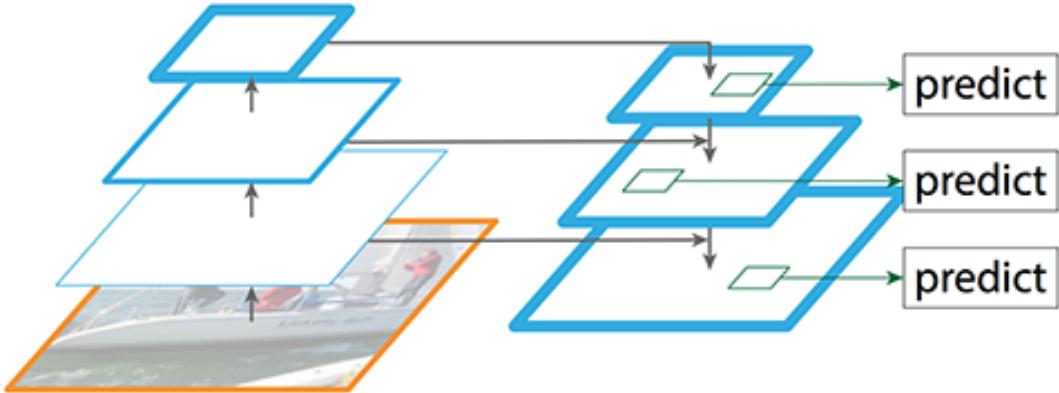
Nakon prolaza testne slike kroz mrežu, na izlazu se pojavljuje najbolja predikcija koordinata i klasa objekata sa slike. S tim vrijednostima preostaje samo jedan korak za **prikaz slike s ocrtavanjem navedenih koordinata i klasa** koji se lako izvodi raznim knjižnicama programskog jezika kojeg koristimo.

4.3.2. v2, v3, v4, v5, v6

Glavni problem koji je prva verzija algoritma YOLO imala bila je slaba uspješnost detekcije grupe malih objekata s obzirom da je jedna ćelija mogla detektirati samo jedan objekt. Uz to, prenaučenost na veličinu okvira objekata izazivala je probleme pri lociranju objekata novih, neuobičajenih veličina. S ciljem rješavanja navedenih zapreka, **druga verzija** uvela je predefinirane okvire (eng. **anchor boxes**), trenirala mrežu sa slikama veće rezolucije te **normalizirala serije** (eng. batch) pri treniranju.

Predefinirani okviri na slici različitih omjera širine i visine koriste se u kombinaciji s predviđenim odmacima za određivanje konačnog graničnog okvira. To algoritmu omogućuje rukovanje širim rasponom veličina objekata i omjera slike. Normalizacijom se ograničava utjecaj rijetkih graničnih vrijednosti te se olakšava optimizacija samog algoritma rješavanjem problema pri konvergenciji.

Treća verzija ima tri različite veličine anchor boxeva po ćeliji te ekstrahirala značajke iz tih veličina kroz novi *backbone*, **Darknet-50**. Konvolucijska mreža koristi princip, sličan mreži ResNet, **Feature pyramid network** s kojim iskorištava svojstvo veza pre-skoka između konvolucijskih slojeva. Nakon ekstrakcije apstraktnih dijelova visokih slojeva, FPN vrši upsampling i downsampling slike kako bi obuhvatilo sitne detalje u nižim slojevima. Lateralne veze se uvode između rekonstruiranih slojeva i mapa značajki. Ovim lateralnim vezama se omogućuje da rekonstruirani slojevi s jakom semantičkom informacijom u visokim slojevima dijele informacije s nižim slojevima. U novoj verziji FPN-a, koristi se *K-means* grupiranje za određivanje anchor boxova, nezavisna logistička klasifikacija kao aktivacijska funkcija i binarna kros entropija za gubitak predikcija klase. Ovi poboljšani pristupi omogućavaju bolje uključivanje detalja slike i precizniju lokalizaciju objekata.



Slika 4.4: FPN

https://miro.medium.com/v2/resize:fit:750/format:webp/1*aMROAN7CtD1gdzTaZIT5gA.png

YOLOv4 koristi CSPDarknet53 kao novi *backbone*. Ova arhitektura kombinira konvolucijske blokove i *spatial pyramid pooling* (SPP) modul koji omogućava mreži da radi s ulaznim slikama koje imaju različite dimenzije i razlučivosti, a da pritom održi fiksnu veličinu izlaza. *Cross Stage Partial bottleneck* je ključni element unutar nove arhitekture. Ovaj blok koristi **strategiju grupne konvolucije** kako bi se smanjio broj kanala i smanjila računska složenost. Ideja je da se ulazni kanali podijele na dvije

grupe, pri čemu jedna grupa prolazi kroz konvolucijski sloj, a druga grupa ostaje ne-promijenjena. Zatim se rezultirajuće značajke konkateniraju, čime se postiže bolje iskorištenje informacija. U službenom papiru o YOLO v3, autori su iznijeli nekolicinu pokušaja poboljšanja algoritma koje nisu uspjeli. Jedan od tih je *focal loss* funkcija koja se zbrajala u sumu funkcije gubitka te je pokušala riješiti problem klasne neravnoteže dodjeljivanjem veće težine teškim ili lako pogrešno klasificiranim primjerima. U verziji 4, uvedena je GHM funkcija gubitka koja je uspjela riješiti navedeni problem.

Ključne promjene u **petoj verziji**, izdanoj samo 2 mjeseca nakon izlaska verzije 4, nalaze se u novitetima podataka s kojima treniramo mrežu te načinu određivanja predefiniranih okvira. Povećanje podataka (eng. *data augmentation*) jedno je od mogućnosti pomoću kojeg se neuronska mreža može osigurati od generalizacije. Ta tehnika predstavlja proces povećavanja raznolikosti i količine podataka za treniranje. Jedan takav način povećanja koristi i YOLO v5 koji spaja slike u jednu veliku sliku (**Mosaic Data Augmentation**) te na taj način formira kompleksnije slike za bolje učenje mreže. Yolo v5 također uvodi automatsko učenje za predefinirane okvire. Na temelju generiranog skupa podataka i testiranju takvih na slikama, provodi se grupiranje s modificiranim algoritmom K-means koji koristi IoU (eng. Intersection Over Union) umjesto Euklidijanove udaljenosti.

Za razliku od prijašnjih YOLO arhitektura, u **šestoj verziji** se uklanjaju predefinirani okviri te se vrijednosti koje opisuju takve okvire smanjuju čak 3 puta. Razlog tome je upotreba samo jedne vrijednosti koja predstavlja predefinirani centar objekta. Takva metoda koristi detekciju ključnih točaka ili tehnike temeljene na toplinskoj karti za prepoznavanje prisutnosti i lokacije objekata na slici. Navedena modifikacija je rezultirala 51% bržim algoritmom. Osim toga verzija je promijenila arhitekturu *backbone* modula te je izmijenila funkciju gubitka kako bi riješila određene probleme koje rezultiraju predefinirane vrijednosti ključnih točaka. [7]

4.3.3. v7

Želja za boljim i naprednjijim stanjem osnovna je ljudska osobina koja se pojavljuje i u YOLO algoritmu koji je prošle, 2022. godine, dobio sedmu verziju, a trenutno je u osma procesu izrade. Sedma se verzija fokusira na poboljšanje optimizacije te istovremeno pojačavanje točnosti. Izraz u kojem se točnost pojačava, a cijena treniranja ostaje ista ili se smanjuje, autori Yolo v7, nazvali su "*bbag of freebies*". Dvije posljedice te "vreće koje algoritam dobiva gratis" su objašnjene u nastavku.

Propuste na koje je verzija pronašla rješenje nalaze se u **ponovnoj parametrizaciji**

(eng. re-parametrization) i u **dinamičnom načinu dodijeljivanju labela**.

Objašnjavajući prijašnje verzije, spomenute su razne tehnike kako se podatci i mreža mogu proširiti da na izlazu dobivamo različitije te time bolje predikcije. Više se slika može spojiti u jednu, a mreža može određen sloj raspodijeliti na dva te stvoriti dvije grane koje se istovremeno treniraju. Ovakve tehnike vuku za sobom pitanje kako dodijeljivati labele. *Coarse for auxiliary and fine for lead loss* odgovor je na to pitanje. [8] Duboki nadzor poznata je tehnika koja se koristi u mrežama i uključuje izdvajanje dodatne *glave* mreže (eng. head) iz sredine slojeva. Takav modul *glave* mreže (eng. auxiliary head), imati će robusnije težine koje će sadržavati znanje mreže na plitkoj razini. *Head* modul mreže koji završava tek na kraju čitave neuronske mreže naziva se *vodeća glava* (eng. lead head). Princip koji verzija 7 koristi je da za svaku *glavu* pojedinačno dodijeljuje posebne labele pomoću kojih će se računati funkcije gubitka. Robusne, slabe (eng. coarse) labele koriste se za pomoćnu, a jake (eng. fine) za vodeću *glavu*. Za slikovitije shvaćanje moguće je percipirati sliku koja prikazuje mnogo životinja. Životinje na nebu bi se označile robusnim labelama kao "ptice", dok bi se unutar preciznih labele identificirale kao konkretnе vrste ptica, na primjer "galebovi".

Ideja iza ponovne parametrizacije modela je spajanje više slojeva u jedan na dijelu detekcije. Autori algoritma su predstavili dvije mogućnosti izračuna težina bez dodatnih akcija koje bi mrežu i njenu izračunljivost samo zakomplikirale. Jedna opcija je da se treniraju identični modeli s različitim podatcima te se na kraju zadnje iteracije izračunava srednja vrijednost njihovih težina. Izračunavati srednju vrijednost težina na određenim iteracijama druga je, slična, opcija.

Onaj dio koda koji je uveo poboljšanje, ali ne bez ikakvog utroška je reforma pri arhitekturi. **E-ELAN** je arhitektura *backbone* dijela mreže zadužena za širenje, premještanje i spajanje kardinalnosti kako bi poboljšala sposobnost učenja mreže, zadržavajući pritom originalni gradijentni put. Unatoč dužini gradijentnog puta i broju slojeva računalnih blokova u ELAN-u, on je postigao stabilno stanje. E-ELAN mijenja samo arhitekturu računalnog bloka, dok se arhitektura prijelaznog sloja u potpunosti ne mijenja. Strategija arhitekture koristi grupnu konvoluciju za proširenje kanala i kardinalnosti računalnih blokova. Osim nove arhitekture *backbone* dijela, skaliranje modela za one modele temeljene na ulančavanju donijelo je pogodnosti za algoritam. Glavna svrha skaliranja modela je prilagoditi neke attribute modela kako bi zadovoljili potrebe različitih brzina zaključivanja. Algoritam u svojoj strukturi mreže koristi način skaliranja

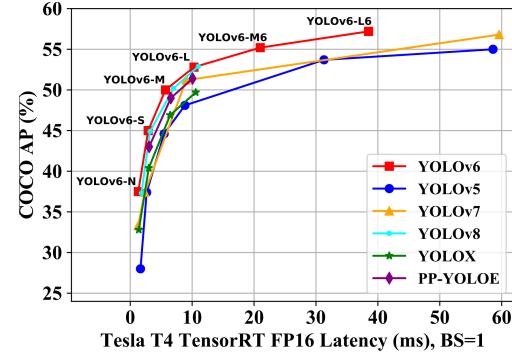
konvolucijskih blokova tako da ih istovremeno proširuje i stvara nove u dubinu.

4.3.4. Usporedba verzija

Iako je verzija 7 daleko naprednija što se tiče infrastrukture mreže, metrika *mean Average Precision* (mAP) približne je vrijednosti u usporedbi s prvom verzijom. Verzija 7 iznosi 51.4% na testnom skupu dok prva verzija čak 63%. Razlog zašto je sedma verzija za skoro 12% slabija od prve, iako je trenirana na puno većem skupu podataka (1-Coco 328k), je pozornost na brzinu okvira po sekundi. Prvobitna ideja YOLO algoritma je da detektira objekte u realnom vremenu s obzirom da je to bio problem *two-stage* algoritama. Verzija 7 može generirati čak 161 okvir po sekundi (**FPS**) dok prva verzija generira otprilike tri puta manje.

Detector	Number of n-layers	FLOPS	FPS	map value	Used set of data
YOLO v1	26.00	Not-given	45.00	63.50	VOC-data
YOLO v1-Tiny	9.00	Not-given	155.00	52.80	VOC-data
YOLO v2	32.00	62.95	40.00	48.20	COCO-data
YOLO v2-Tiny	16.00	05.42	244.00	23.60	COCO-data
YOLO v3	106.00	140.70	20.00	57.80	COCO-data
YOLO v3-Tiny	24.00	05.57	220.00	33.20	COCO-data

(a) Starije verzije



(b) Novije verzije

Slika 4.5: Metrike za YOLO verzije

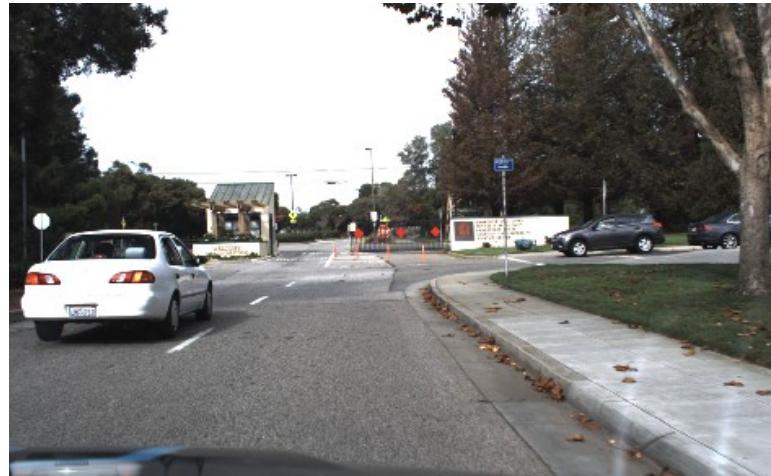
https://aisent.io/assets/images/blog/performance_yolov6.png

5. Programsko rješenje

5.1. Skup podataka

Podatci su preuzeti s [Kaggle platforme](#) te su pod poljem *Licence* navedeni kao CC0 : Public Domain.

Slike prikazuju pregled ceste i okoline iz perspektive vozača te su uz njih također preuzete i modificirane .csv datoteke koje labeliraju granične okvire i klase objekata na slikama. Razlog modifikacije je smanjenje slika za treniranje i prikupljanje svih objekata iz slike u jednom retku novonastale tablice. Ukupno je preuzeto 22 tisuće slika te je u modificiranoj .csv datoteci otprilike 18 tisuća slika labelirano. Klase objekata koje sustav može prepoznati su automobil, kamion, pješak, biciklist i semafor.



Slika 5.1: Primjer slike iz skupa podataka

5.2. Odabir programskog jezika, knjižnica i okruženja

S obzirom na kompleknost i resurse koje YOLO algoritam sa spomenutim podatcima treba trenirati, koristimo Google Colab, okruženje u oblaku koji služi za izvršavanje

Jupyter bilježnica. Google Colab ima opciju treniranja podataka koristeći NVIDIA Tesla T4 GPU koji znatno ubrzava rad. Kod za YOLO verziju 1 je preuzet s GitHub računa koji je etiketiran kao "contribution friendly" (<https://github.com/aladdinpersson/Machine-Learning-Collection>). U kodu su neke metode i dijelovi koda izmijenjeni te iznova napisani ili mjestimično prilagođeni ovisno o potrebi. Odabran je Python (verzija 3.10) kao programski jezik te je korišten Visual Studio Code kao okruženje za pisanje navedenog. Velik je broj knjižnica korišten te će one konkretno biti navedene ovisno o dijelu koji bude objašnjavan.

5.3. Programski kod

5.3.1. Rješenje pomoću verzije 1 Yolo algoritma

Algoritam je organiziran u 5 datoteka: `model.py`, `dataset.py`, `train.py`, `utils.py` i `loss.py`.

U `model.py` datoteci kreirana je arhitektura modela. U polje su dodane n-torke koje opisuju značajke (dimenzije jezgre, broj različitih jezgri, *stride* i *padding*) svakog konvolucijskog sloja. Na konvolucijski sloj su nadodani potpuno povezani slojevi. Implementirana je i funkcija unaprijednog prolaza. Jedini izuzetak napravljen u modelu, koji se ne nalazi u službenom papiru Yolov1, je implementiran normalizacija serija ulaznih podataka. Razlog tome je poboljšano učenje s obzirom da kod nije pisan u idealnim uvjetima s idealnim resursima. Leaky ReLU s 0.1 negativnim nagibom se koristi kao aktivacijska funkcija. Metode normalizacije, aktivacije, konvolucijskog 2D sloja i ostalih matričnih transformacija koriste se iz PyTorch okvira za strojno učenje.

U `loss.py` datoteci implementirana je klasa `YoloLoss` za koju je izvedena metoda unaprijednog prolaza koja će računati funkciju gubitka raspisani u službenom papiru. Metoda prima tenzor predikcija i tenzor ciljnih labela. Tenzor je višedimenzionalna struktura podataka koja se koristi za reprezentaciju slika i ulaznih podataka u detekciji objekata. S obzirom da je izlaz modela YOLO algoritma oblika $[S, S, 5*B+C]$ i koriste se serije slika za iteraciju bez nuliranja gradijenta još jedna dimenzija će predstavljati broj tih slika u seriji. Algoritam za manipuliranje s matricama koristi metode iz PyTorch knjižnice poput max, flatten, sign, sqrt i sličnih. Za računanje završnih transformiranih matrica predikcije i stvarnih labela koristi se `nn.MSELoss` (Mean Squared Error Loss) klasa iz PyTorch knjižnice koja se koristi u regresijskim problemima.

U glavnoj, `train.py`, importamo sve sporedne metode iz drugih datoteka. `Utils.py`

datoteka je u kojoj se nalaze metode poput izračuna IoU, prikazivanja (eng. plot) slika s graničnim okvirima (bounding boxes), *Non Max Suppression* i *Mean Average Precision* metoda te dohvatanja i prebacivanja graničnih okvira iz perspektive celije u perspektivu cijele slike. Određene metode će ovisno o potrebi biti detaljnije objašnjene u nastavku.

Za treniranje se koristi serija od 16 slika, propadanje težina je postavljeno prilično smanjeno te se koristi Pytorch planer faktora učenja *MultiStepLR*. Ulazne slike su transformirane na dimenzije 448x448.

```
1 LEARNING_RATE = 2e-4
2 DEVICE="cpu"
3 STEP_SIZE = 100
4 GAMMA = 0.1
5
6 if torch.cuda.is_available():
7     DEVICE="cuda"
8
9 BATCH_SIZE = 16
10 WEIGHT_DECAY = 0.00001
11 EPOCHS = 135
12 NUM_WORKERS = 2
13 PIN_MEMORY = True
14 LOAD_MODEL = True
15 LOAD_MODEL_LW = "last_weights_overfit.pth.tar"
16 LOAD_MODEL_FILE = "tensor(0.0012)_5_overfit.pth.tar"
17 IMG_DIR = "./selfdriving-images"
18
19 class Compose(object):
20     def __init__(self, transforms):
21         self.transforms = transforms
22
23     def __call__(self, img, bboxes):
24         for t in self.transforms:
25             img, bboxes = t(img), bboxes
26
27     return img, bboxes
28
29 transform = Compose([transforms.Resize((448, 448)), transforms
```

```

    .ToTensor(), ])
30 ...
31 scheduler = MultiStepLR(optimizer, milestones=milestones,
                           gamma=GAMMA)

```

Za optimizaciju je odabran algoritam ADAM jer kombinira prednosti adaptivnih stopa učenja s momentima prvog i drugog reda, prilagođava brzinu učenja za svaki parametar pojedinačno i bolje se nosi s rijetkim ili bučnim podacima. Momentum predstavlja akumulaciju prošlih gradijenata prilikom ažuriranja parametara modela.

Slike su učitane u skup slika pomoću klase `ImageDataset` iz `texttdataset.py` datoteke. Klasa učitava putanju do datoteke s labelama, vrstu neuronske mreže koju koristimo, direktorij slike i dimenzije slike koje se procesiraju. Vrsta neuronske mreže koju koristimo je YOLO s klasičnim *backbone* dijelom koji prima preko *input* bloka slike veličine 448x448. Slike skinute s interneta koje koristimo su veličine 480x300 te u metodi `getitem(self, index)` mijenjamu veličinu na željenu te su koordinate prebačene u postotke zbog daljne implementacije.

U metodi se koristi argument koji opisuje *backbone* te omogućava opcionalnost transformiranja slika na 224x224 dimenzije. Takve dimenzije za ulazne podatke koristi ResNet-50 mreža čije će se performanse treniranja također prikazati u kasnijem poglavljju.

Takav skup podataka upakiran je u *DataLoader*, Pythonovu klasu koja se koristi za iteriranje kroz skup podataka u strojnem učenju zbog svojih mogućnosti.

Jedna od mogućnosti je postavljanje hiperparametra `shuffle` i `drop last` na `True` kako bi se izbjegla generalizacija.

Za treniranje se koristilo 3000 različitih slika fotografiranih iz perspektive vozača. Takvo treniranje kroz 135 epohe zahtjevalo je vremenski otprilike 24 sata s razmacima zbog navedenih slabih resursa.

U svakoj iteraciji epohe prvo se izračunavaju granični okviri po čelijama i mAP vrijednost pomoću metoda iz `utils.py` datoteke. Za izračunavanje graničnih okvira pomoću definiranog modela potrebno je model postaviti u stanje evaluacije. U evaluacijskom načinu rada, model primjenjuje svoje slojeve bez ikakvih promjena u parametrima.

```

1     pred_boxes, target_boxes = get_bboxes(
2         train_loader, model, iou_threshold=0.5, threshold
3         =0.4
4     )

```

```

5     mean_avg_prec = mean_average_precision(
6         pred_boxes, target_boxes, iou_threshold=0.5,
7         box_format="midpoint"
8     )

```

Metoda `get_bboxes` koristi metodu `cellboxes_to_boxes` koja za cijeli skup ciljnih labela i predikcija izračunava najbolju predikciju graničnog okvira po ćeliji ovisno o rezultatu povjerenja. Također, veličine koje opisuju granične okvire transformiraju se u ovisnosti o cijeloj slici, a ne veličini ćeliji. Podatci više nisu u 4D tenzor obliku nego se prve dvije dimenzije prebacuju u liste te se vraća indeks klase s najvećom vjerojatnosti. U `get_bboxes` metodi se nakon toga koristi popularna *Non Max Suppression* (NMS) funkcija koja se koristi za uklanjanje višestrukih detekcija objekata koje se preklapaju ili su blizu jedna drugoj, kako bi se zadržale samo najrelevantnije detekcije. U funkciju proslijedujemo granični okvir, dvije vrste praga i format okvira. Granične okvire koji imaju manji rezultat povjerenja od 40% odbacujemo dok kod onih, koji imaju IoU veći od 50% s nekim drugim, prihvaćamo samo rezultat s boljim rezultatom povjerenja.

Izlazni rezultati metode proslijedeni su u metodu `mean_average_precision` koja prolazi kroz sve klase objekata i izračunava preciznost za svaku klasu. Koristi se postupak brojanja lažnih pozitiva (FP), lažnih negativa (FN) i istinitih pozitiva (TP) kako bi se izračunale preciznost i odziv za svaku predikciju. Također se koristi metrika IoU kako bi se odredilo je li predikcija ispravna ili ne. Na kraju, izračunate preciznosti za svaku klasu se zbrajaju i dijele s ukupnim brojem klasa kako bi se dobio srednji mAP.

Svaku petu epohu ažuriraju se parametri istrenirani do tad, a ovisno o povećanju mAP vrijednosti spremaju se parametri koji uzrokuju novi najbolji rezutat treniranja. Parametri se spremaju u `.pth.tar` datoteku.

Treniranje se odvija u funkciji `train_fn`. *Tqdm*, Pythonova knjižnica, koja pruža napredno prikazivanje napretka iteracija kroz mrežu određene epohe, koristi se na više mesta u kodu.

```

1 def train_fn(train_fn_loader, model, optimizer, loss_fn):
2
3     loop = tqdm(train_fn_loader, leave=True)
4     losses = []
5     model.to(DEVICE)
6
7     for batch_idx, (x, y) in enumerate(loop):

```

```

8     x, y = x.to(DEVICE), y.to(DEVICE)
9     out = model(x)
10    loss = loss_fn(out, y)
11
12    optimizer.zero_grad()
13    loss.backward()
14    optimizer.step()
15
16    losses.append(loss.item())
17    loop.set_postfix(loss=loss.item())
18
19    mean_loss = compute_mean(losses)
20    print(f"Mean loss was {mean_loss}")

```



Slika 5.2: Rezultat mreže

Nakon treniranja, *mAP* ne uspijeva postići vrijednost veću od 0.25 dok se *Mean Loss* kreće oko 8. Ovakvi rezultati rješenje problema čine neuspješnim. Mogući problemi zbog kojih mreža ne može postići zavidne rezultate je transformacija veličine slike koja pogoršava njenu rezoluciju ili pogrešno postavljeni hiperparametri.

Pravilniji pristup problemu bio bih koristeći pretreniran model na općenitijem skupu podataka koji sadrži raznolike slike iz više područja pa tek onda na slikama iz cestovnog prometa. Takve pretrenirane modele moguće je preuzeti iz knjižnica *Pytorch*-a.

Korištenje istreniranih parametara ResNet-50 konvolucijske mreže

Transfer learning je tehnika u području strojnog učenja koja omogućuje korištenje prethodno naučenih parametara iz jednog modela kako bi se brže postigla dobra performansa na novom zadatku ili skupu podataka. Također tehnikom i malo drugačijom mrežom koja služi kao *backbone* poslužili smo se za dotreniravanje podataka koje imamo nadajući se boljim rezultatima. Novi *backbone* postaje ResNet-50 koji koristi koncept *skip connections* kako bi prevladao problem gubitka informacija tijekom dubokog učenja i omogućio uspješno treniranje dubokih neuronskih mreža sa 50 slojeva.

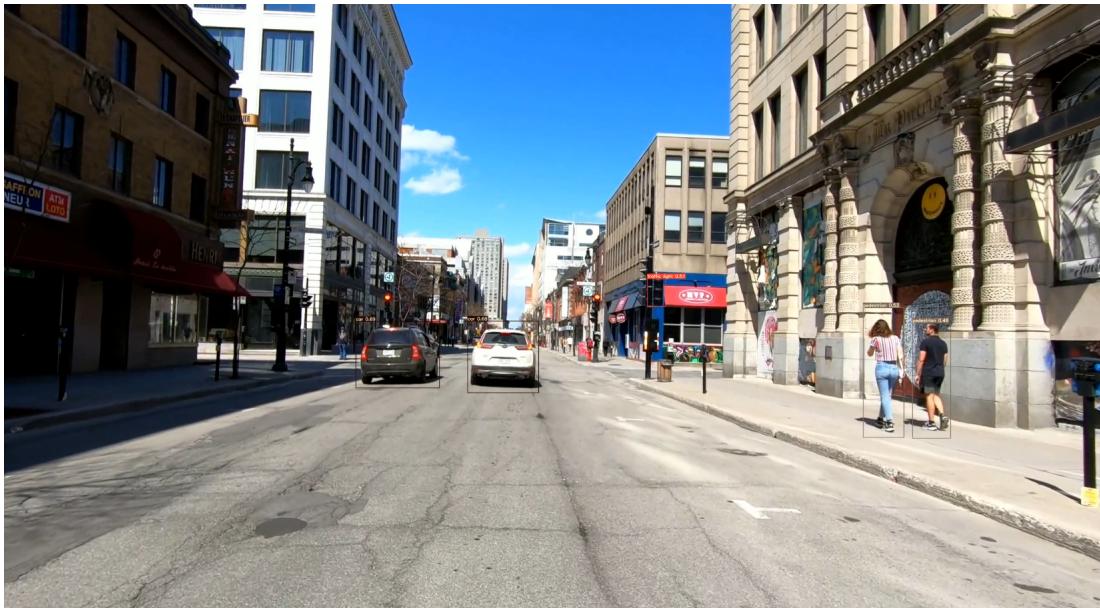
```
1 resnet = resnet50(weights=ResNet50_Weights.IMGNET1K_V2)
2 resnet = nn.Sequential(*list(resnet.children())[:-2])
3 resnet.fc = nn.Identity()
4 model.darknet = resnet
```

5.3.2. Rješenje pomoću verzije 7 Yolo algoritma

Trenutno najbolja opcija za dotreniravanje prilagođenog skupa podataka, izuzevši najnoviji *state-of-the-art* YOLOv8 modela, je YOLOv7. YOLOv7 implementaciju je na platformu Github objavio Wong Kiu Yiu te je priložio različite serije modela YOLOv7 (<https://github.com/WongKinYiu/yolov7.git>).

Za naš problem uzeli smo osnovnu verziju koja je opisana u prijašnjim poglavljima. Za dotreniravanje je korišteno 78 slika koje su labelirane pomoću *labelImg* alata za označavanje i labeliranje slika. Alat je moguće koristiti u *Pythonu*. Od 78 slika otprilike je labelirano 100 automobila, 10 većih vozila (autobusevi, kamioni i sl.), 40 pješaka, 50 semafora te 10 biciklista. Inače bi bilo korisnije dotreniravati model na većem skupu, ali je sporohodan proces labeliranja ograničio mogućnosti.

Unatoč tome, model daje vrlo dobre podatke te se mreža čak nakon 50 epoha može koristiti na videozapisu koji prikazuje cestovni promet. Naravno, klase s manjim brojem labela teže je zamijetiti te se kasnije od ostalih detektiraju.



Slika 5.3: Prikaz isječka iz rezultata sa videozapisa

Epoch	gpu_mem	box	obj	cls	total	labels	img_size
49/49	10.5G	0.0402	0.01539	0.005921	0.06151	204	640: 100% 5/5 [00:05<00:00, 1.19s/it]
		Class Images	Labels		P	R	mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00, 1.69it/s]
		all	10	53	0.287	0.726	0.573 0.198
		car	10	36	0.305	0.972	0.821 0.298
		truck	10	1	0.495	1	0.498 0.0498
		traffic light	10	15	0.347	0.933	0.896 0.375
		pedestrian	10	1	0	0	0.0765 0.0689
50 epochs completed in 0.146 hours.							

Slika 5.4: Metrika zadnje epohe treniranja

6. Zaključak

Od prvotne verzije, YOLO se razvio kroz nekoliko iteracija, svaka s poboljšanjima u performansama i točnosti. U usporedbi s drugim popularnim objektnim detekcijskim algoritmima poput SSD (Single Shot MultiBox Detector) i Retine, YOLO je iskazao konkurentnost u brzini detekcije. SSD koristi više razlučivosti za predviđanje objekata i postiže visoku preciznost, ali je usporen u odnosu na YOLO. S druge strane, Retina koristi konvolucijsku neuronsku mrežu s boljom sposobnošću lokalizacije objekata, ali također zahtijeva više vremena za obradu. YOLO algoritam je revolucionarna metoda objektne detekcije koja je unaprijedila brzinu i točnost detekcije objekata te je postala preferirani izbor za primjene u stvarnom vremenu.

Prepoznavanje objekata primjenom dubokog učenja

Sažetak

U ovom istraživanju opisani su osnovni modeli dubokog učenja za prepoznavanje objekata na videozapисima, s posebnim naglaskom na model YOLO. Također je odabran predtrenirani skup podataka koji sadrži slike s određenim klasama objekata. Opisan je primjer rada sustava na odabranom skupu podataka te je navedena korištena literatura.

Ključne riječi: detekcija objekata, YOLO, skup podataka

Object detection using deep learning

Abstract

This study describes the basic models of deep learning to identify objects in videos, with particular emphasis on the YOLO. A pre-trained data set containing images with certain object classes has also been selected. An example of the algorithm's work on the selected data set is described and the literature that was used is cited.

Keywords: object detection, YOLO, data set

LITERATURA

- [1] Tembhurne JV, Diwan T, Anirudh G. Object detection using yolo. U *Multimed Tools Appl.* 2023;82(6):9243-9275. doi: 10.1007/s11042-022-13644-y. Epub 2022 Aug 8. PMID: 35968414; PMCID: PMC9358372.
- [2] Subašić M. Detekcija objekata pomoću dubokih neuronskih mreža. U *Slajdovi prezentacije s predmeta Obrada informacija*. FER, 2020.
- [3] William S Noble. What is a support vector machine? *Nature biotechnology*, 24 (12):1565–1567, 2006.
- [4] Cahya Rahmad, R An Asmara, DRH Putra, I Dharma, H Darmono, i I Muhiqqin. Comparison of viola-jones haar cascade classifier and histogram of oriented gradients (hog) for face detection. U *IOP conference series: materials science and engineering*, svezak 732, stranica 012038. IOP Publishing, 2020.
- [5] Joseph Redmon, Santosh Divvala, Ross Girshick, i Ali Farhadi. You only look once: Unified, real-time object detection, 2016.
- [6] Ribarić S. Uvod u raspoznavanje uzoraka; linearne decizijske funkcije. U *Prezentacije s predmeta Uvod u raspoznavanje uzoraka*. FER.
- [7] Juan Terven i Diana Cordova-Esparza. A comprehensive review of yolo: From yolov1 and beyond, 2023.
- [8] Chien-Yao Wang, Alexey Bochkovskiy, i Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022.