Primjena računalnog vida u održavanju trupa putničkih zrakoplova

Konjevod, Damjan

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Damjan Konjevod

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Primjena računalnog vida u održavanju trupa putničkih zrakoplova

Mentor:
Prof. dr. sc. Dragutin Lisjak, dipl. ing.

Student:
Damjan Konjevod

Zagreb, 2022.

Zahvaljujem se svom mentoru profesoru dr.sc. Dragutinu Lisjaku na stručnoj pomoći, znanju i potpori koju mi je pružio za vrijeme izrade diplomskog rada.

Zahvaljujem se djevojci Barbari na strpljenju i podršci tijekom studija i pisanja ovog rada.

Najveće zahvale idu mojoj obitelji za podršku, motivaciju i vjeru koju su imali u mene tijekom cijelog mog školovanja.

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

Damjan Konjevod

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija zrakoplovstva

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 22 - 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

DIPLOMSKI ZADATAK

Student: **DAMJAN KONJEVOD** JMBAG: 0035203103

Naslov rada na hrvatskom jeziku: **Primjena računalnog vida u održavanju trupa putničkih zrakoplova**

Naslov rada na engleskom jeziku: **Application of computer vision in the maintenance of the fuselage of passenger aircraft**

Opis zadatka:

Jedan je temeljenih postupaka u procesu održavanja putničkih zrakoplova je vizualna kontrola oštećenja vanjskog trupa zrakoplova od oštećenja koja nastaju pod različitim okolnostima – od udara ptica, udara groma, učestala pojava korozije na pojedinim lokacijama trupa zrakoplova, popuštanja spojeva itd. Trenutno se ovaj postupak radi na način da certificirani djelatnici održavanja vizualno pregledavaju trup zrakoplova postavljanjem čelične konstrukcije oko samog trupa, a cijeli je postupak dugotrajan i logistički zahtjevan. Noviji načini u pristupu rješavanja ovog problema su primjena tehnologije dronova i primjena tehnologije računalnog vida.

U skladu s prethodno navedenim, u radu je potrebno:

1. Opisati problematiku održavanja trupa zrakoplova.
2. Dati detaljan prikaz tehnologije računalnog vida.
3. Predložiti i razviti računalnu aplikaciju za automatiziranu detekciju anomalija trupa zrakoplova primjenom tehnologije računalnog vida.
4. Na temelju analize sprovedenog istraživanja, izvesti zaključak te predložiti smjernice za daljnje istraživanje.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
3. ožujka 2022.	5. svibnja 2022.	9. - 13. svibnja 2022.

Zadatak zadao: Prof.dr.sc. Dragutin Lisjak

Predsjednik Povjerenstva: Prof. dr. sc. Milan Vrdoljak

Sadržaj

Popis slika

Popis tablica

Popis oznaka

Oznaka	Jedinica	Opis
A_i	-	površina presjeka graničnih okvira
A_u	-	površina dvaju graničnih okvira
B	-	broj graničnih okvira
b_h	-	visina graničnog okvira
b_x	-	*x* koordinata centra graničnog okvira
b_y	-	*y* koordinata centra graničnog okvira
b_w	-	širina graničnog okvira
C	-	broj klasa
C_i	-	predviđena pouzdanost
$\hat{C}_i$	-	*IoU* između predviđenog graničnog okvira i graničnog okvira definiranog u temeljnoj istini
c_i	-	vjerojatnost klase *i*
c_x	-	*x* pomak od gornje lijeve ćelije
c_y	-	*y* pomak od gornje lijeve ćelije
F	-	visina/širina filtera
h_i	-	visina predviđenog graničnog okvira u *i*-toj ćeliji
IoU	-	presjek preko unije
L	-	funkcija gubitka
P	-	popunjavanje
p_c	-	vrijednost koja pokazuje da li je objekt prisutan na slici ($p_c=1$) ili nije ($p_c=0$)
p_h	-	visina sidrenog okvira
p_i	-	predikcija algoritma za ćeliju *i*
$\hat{p}_i$	-	funcija koja je jednaka jedinici ako ćelija *i* sadrži klasu *c*
$p_i(c)$	-	predikcija klase *c* za ćeliju *i*
Pr(Object)	-	vjerojatnost da se objekt nalazi unutar graničnog okvira
p_w	-	širina sidrenog okvira
S	-	parametar koji određuje broj ćelija
S_F	-	korak filtera
t_h	-	visina predviđenog graničnog okvira
t_o	-	ocjena pouzdanosti predviđenog graničnog okvira
t_x	-	*x* koordinata centra predviđenog graničnog okvira
t_y	-	*y* koordinata centra predviđenog graničnog okvira
t_w	-	širina predviđenog graničnog okvira
x_i	-	*x* koordinata centra predviđenog graničnog okvira u *i*-toj ćeliji
y_i	-	y koordinata centra predviđenog graničnog okvira u i-toj

		ćeliji
w	-	svaki parametar koji mreža može naučiti
W_1	-	visina/širina ulaznog tenzora
W_2	-	visina/širina izlaznog tenzora
α	-	brzina učenja
α_i	-	hiper-parametar za izglađivanje

Sažetak

U ovome radu razvijena je aplikacija za inspekciju trupa putničkih zrakoplova primjenom tehnologije računalnog vida. Objašnjeni su problemi trenutnog procesa održavanja i prikazana su tipična oštećenja trupa zrakoplova. U trećem je poglavlju dan uvod u tehnologiju računalnog vida, prikazane su njezine tipične zadaće i ukratko su objašnjene različite vrste detektora. Prikazana je primjena konvolucijskih neuronskih mreža, njihova struktura i način na koji funkcioniraju. Za detekciju anomalija izabran je YOLO algoritam, objašnjena je njegova struktura i razlike između različith verzija modela. Detaljno je prikazan proces kreiranja i treniranja konvolucijske neuronske mreže kao i različith verzija YOLO detektora anomalija u programskom paketu *MATLAB*. Na kraju rada provedena je usporedba rezultata te je izveden zaključak.

Ključne riječi: inspekcija zrakoplova, tehnologija računalnog vida, konvolucijska neuronska mreža, YOLO algoritam

Summary

In this paper, an application for the inspection of the fuselage of passenger aircraft using computer vision technology was developed. The problems of the current maintenance process are explained and typical damage to the fuselage is shown. The third chapter introduces computer vision technology, presents its typical tasks, and briefly explains the different types of detectors. The application of convolutional neural networks, their structure and the way they function are presented. The YOLO algorithm was chosen to detect anomalies, its structure and differences between different versions of the model were explained. The process of creating and training a convolutional neural network as well as different versions of YOLO anomaly detectors in the MATLAB software package is presented in detail. At the end of the paper, a comparison of the results was made and a conclusion was drawn.

Key words: aircraft inspection, computer vision technology, convolutional neural network, YOLO algorithm

1. UVOD

Održavanje u zrakoplovstvu je jedan od osnovnih procesa u zrakoplovnoj industriji te od pravilnog vođenja istog ovisi dugovječnost konstrukcije tj. trupa zrakoplova, njegovih elektroničkih i električnih dijelova, kao i uspješno prometovanje na domaćim i internacionalnim letovima. Unatoč ubrzanom razvoju tijekom godina, pravilno i dobro upravljano održavanje može se smatrati i jako „nezahvalnom“ aktivnošću. Naime, održavanje je dugotrajan i zahtjevan proces, koji obuhvaća veliki broj aktivnosti ako se želi postići maksimalan učinak i sigurnost. Upravo složenost procesa između ulaska zrakoplova u hangar i njegovog osposobljavanja za sljedeći let zahtjeva značajne resurse u vidu izdašnih novčanih ulaganja i velikog broja potrebnih tehničara. Ipak, najveći je nedostatak ovog procesa održavanja njegova ovisnost o ljudskom faktoru. Kao što će kasnije biti opisano u radu, ljudski faktor ima dokazanu sklonost propustima te nesustavnim pogreškama, koje se nikako ne smiju zanemariti. I najmanje odstupanje od propisanih regulativa održavanja može dovesti velik broj ljudi u životnu opasnost, ali i produljenje vremena aktivnosti održavanja, što dovodi do povećanih troškova. Tako se npr. u cilju uklanjanja navedenih propusta tijekom provjere, pri provođenju vizualne inspekcije trupa zrakoplova, sve više primjenjuju nove tehnologije. Jedna od takvih tehnologija je primjena dronova za inspekciju anomalija na trupu zrakoplova video snimanjem. Nakon snimanja primjenjuje se tehnologija računalnog vida koja služi za lociranje i detekciju anomalija. Detekcija anomalija nastalih na trupu zrakoplova nije jednostavan zadatak s obzirom na činjenicu da niti jedna anomalija ne izgleda isto. U cilju automatiziranog prepoznavanja vrste anomalije na trupu zrakoplova u radu se eksperimentiralo metodama dubokog strojnog učenja te primjenom konvolucijskih neuronskih mreža kao i detektorima iz skupa YOLO algoritama.

2. PROBLEMATIKA ODRŽAVANJA TRUPA ZRAKOPLOVA

Održavanje zrakoplova je sveobuhvatan i kontinuirani proces pregleda, modificiranja te zamjene dijelova na zrakoplovu. Provođeno je na specifičan i detaljno opisan način, kako bi se u što većoj mjeri zadovoljili međunarodni standardi i ciljevi. Glavni su ciljevi sljedeći [1]:

1. Svi zrakoplovi pušteni u promet moraju biti pravilno održavani i osposobljeni za sigurno izvođenje svih operacija prilikom zračnog prijevoza
2. Održavanje zrakoplova i provođenje svih promjena moraju obavljati samo kompetentni stručnjaci koristeći se odgovarajućom opremom i u odgovarajućim objektima
3. Svi se postupci i izmjene pri održavanju moraju obavljati prema uputama priručnika za održavanje

Navedeni se ciljevi mogu nadalje podijeliti na rutinske i individualne radne zadatke:

1. Čišćenje zrakoplova i glavnih dijelova
2. Čišćenje i provjera dijelova roka trajanja prema standardu pripreme (SOP)
3. Održavanje avionike
4. Nanošenje smjese za sprječavanje korozije
5. Podmazivanje i remont dijelova.
6. Ispuštanje iscurenih tekućina i provjera sustava goriva
7. Servis hidrauličnih i pneumatskih sustava
8. Zamjena okretnih i LRU-ova prema smjernicama
9. Pregled i provjera općeg trošenja i kidanja

Kako bi se održavanje obavilo u cijelosti, potrebno je provesti dodatnu provjeru i održavanje električnih i elektroničkih sustava, što uključuje: radare, računalne sustave, radio i Globalni navigacijski satelitski sustav (GPS).

U održavanju se koristi i „10 elemenata“ programa održavanja propisanih za zrakoplovne kompanije:

1. odgovornost za letenje
2. priručnik za održavanje zrakoplova
3. organizacija održavanja zrakoplova
4. izvršenje i odobrenje održavanja i preinaka

5. raspored održavanja
6. potrebne predmetne inspekcije
7. sustav vođenja evidencije održavanja
8. ugovorno održavanje
9. obuka kadrova
10. sustav kontinuirane analize i nadzora

Kao što se da iščitati, pravilno i propisno održavanje zrakoplova predstavlja jednu od najvažnijih i najrazrađenijih djelatnosti unutar cijele zrakoplovne industrije. Ipak, dimenzije i kompleksne konstrukcije današnjih zrakoplova podrazumijevaju i velika financijska ulaganja. Podaci izvještavaju da se, osim za gorivo, najviše sredstava ulaže upravo u održavanje. Ipak, poražavajući nalazi ukazuju da je 17% zrakoplovnih nesreća u razdoblju između 2014. i 2018. godine uzrokovano nepravilnim održavanjem. Greške koje proizlaze iz samo jednog „malog“ propusta mogu izazvati kvar, kašnjenje, otkazivanje ili preusmjeravanje letova, što nadalje uzrokuje nove financijske troškove čiji iznosi nadmašuju 20 000 USD po satu. Ipak, sve je to neusporedivo s potencijalnim ljudskim žrtvama kao posljedice nepažnje i propusta pri održavanju.

Kao što je već navedeno, veliki se dio odgovornosti cijelog sustava prebacuje na članove tima i instrumentaciju održavanja zrakoplova. Daljnja su istraživanju u tom području pokazala da prevelik broj planiranih, ali i neplaniranih zadataka povećava rizik od pojave greški pri održavanju. Za njima odmah slijede uzroci kao što je velik broj konfiguracija genetičke strukture te broj podataka i informacija u raznim sustavima zrakoplova. Spomenute greške održavanja te njihove kategorije nabrojane su u nastavku teksta.

Prve ili greške pri ugradnji proizlaze iz nepotpune ili krive instalacije opreme i/ili njezinih dijelova. Također, pogrešna orijentacija, mjesto ili oštećenje opreme prilikom instalacije neki su od ostalih uzroka. Za kraj treba navesti i unakrsne veze, ugrađivanje nepotrebnih dijelova, neadekvatna (de)aktivacija te nezatvorene pristupe kao preostale uzročnike grešaka pri ugradnji. Za njima slijede greške pri servisiranju koje prate manjak ili višak tekućine, korištenje pogrešne vrste tekućine, neobavljen servis, nezatvoren pristup te neadekvatno ili ponovno aktivirani sustavi i oprema. Greške pri izolaciji, testiranju ili inspekciji čine još jednu veliku kategoriju te su uzrokovane neotkrivenim kvarovima, nepronađenim greškama pri izolaciji kvara ili operativnim i funkcionalnim testovima, nezatvorenim pristupima te sustavima i opremama koji nisu pravilno (de)aktivirani. Materijali ostavljeni u zrakoplovu, krhotine na rampi i krhotine koje slobodno padaju u otvorene sustave najčešći su razlozi u podlozi grešaka oštećenja stranim predmetom. Moguća su i oštećenja dijelova i

opreme zrakoplova nepropisno korištenim alatima i opremom, korištenjem neispravnog alata i opreme, mehaničkim povlačenjem i guranjem te sudarnim oštećenjima. Bitno je razlikovati i kategorije ozljeda radnika prilikom samog održavanja i greške popravka koje nisu detaljno razrađene.

Kroz sve se navedene greške održavanja protežu tri istaknuta zajednička čimbenika. Prvi se čimbenik odnosi na uvjete rada te obuhvaća prostor, doba dana ili temperaturu u trenutku održavanja. Svakako treba napomenuti da su uvjeti češće nepoželjni nego optimalni, pa se radnici susreću s velikom bukom, skučenim prostorima, velikih hladnoćama ili sparinama. Nadalje, oprema se odnosi na svu potrebnu instrumentaciju kako bi se učinkovito obavili opći vizualni, detaljni i posebno detaljni pregledi. Najznačajnija su pri tome zrcala i leće te oprema potrebna za provođenje tehnike nerazornog ispitivanja. Naravno, u sve je navedeno upleten i ljudski čimbenik, što potvrđuju nalazi koji pokazuju da propusti u kvaliteti obično sadrže jednu ili više ljudskih grešaka. To znači da je unatoč visokim stupnjevima razvoja opreme za praćenje i testiranje podataka o letu, najveća odgovornost i dalje prepuštena radnicima. Daljnjim se istraživanjem zamijetilo da se 50% tih grešaka može svrstati unutar nekoliko kategorija. Prva je od njih greška memorije koja uključuje prosperitivno pamćenje. Naglim prekidom „toka misli“ ili nepotpunim kodiranjem informacija u dugoročno pamćenje dolazi do zaboravljanja radnji koje je tehničar planirao napraviti u budućnosti. Na to se nastavlja nepridržavanje formalnih uputa o načinu provođenja zadataka pa 34% tehničara u Europi izvještava o nepravilnom izvršavanju obveza. Površna komunikacija i pretpostavljanje o odrađenosti određenih zadataka čini sljedeću kategoriju učestalu pri mijenjanju smjena. Najčešće se to odnosi na automatizirane radnje za koje zaposlenik smatra da su već obavljene pa ni ne provjeri istinitost iste. Posljednja kategorija obuhvaća pogreške počinjene neznanjem i nesnalaženjem u novim radnim situacijama kada radnik ne zna kako pravilo pristupiti zadatku. U tim je trenucima izrazito važna uloga supervizora koji će omogućiti pronalaženje optimalnog načina obavljanja poslova. Nedovoljan broj zaposlenika te preopterećenost pojedinca nadalje povećavaju vjerojatnost pojave pogreške. Tome ni ne pomaže činjenica da se učestalim ponavljanjem istovjetnih zadataka povećava sveukupna vjerojatnost pogreške. Istraživanje provedeno na vojnim podacima pokazuje i da postoji vjerojatnost od 10% da ni drugi tehničar, zadužen za provjeru rada, neće detektirati napravljenu grešku.

Navedene su greške „obojene“ ljudskim nesavršenostima, što ostavlja mjesto daljnjem napredovanju i razvitku ovoga područja. Dok se nadopunjuju pravilnici i uče nove vještine interpersonalne komunikacije kako bi se pospješile veze među kolegama, najveći pomak donose tehnološke

promjene. To podrazumijeva početak treniranja i testiranja dronova kao dodatne pomoći standardnom provođenju održavanja. Dronovi bi bili namijenjeni „nadopunjavanju“ i olakšavanju detekcije kvarova na teško dostupnim mjestima, te bi služili kao standardiziranija i objektivnija supervizija ili provjera rutinskih zadataka. Time bi se nastojale otkloniti ljudske brzopletosti i manjak temeljitosti pri obavljanju ponavljajućih i monotonih zadataka. Ipak, nedvojbenu prednost nosi mogućnost detektiranja i obavještavanja o kvarovima motora i ostalih sustava prije dolaska zrakoplov na izlaz zračne luke, čime bi se umanjilo vrijeme detekcije problema u hangaru te smanjile nesustavne promjene rasporeda posla.

2.1 Tipična oštećenja trupa zrakoplova

Mnogi vojni i civilni zrakoplovi pogođeni su korozijom (slika 1) u slučaju prisutnosti stranih materijala kao što su ulje, masnoća, voda, prašina, otopine za čišćenje itd. Ako se nastala korozija ne ukloni brušenjem površine abrazivnim sredstvom, a zatim tretira antikorozivnim sredstvom i, ako je to potrebno, zaštitnim slojem boje, ona može uzrokovati širenje pukotina i nakupljanje stresa što značajno smanjuje mehaničke karakteristike materijala okvira zrakoplova.

Slika 1. Površinska korozija zrakoplova [2]

Zrakoplovi koji lete kraće rute podložni su pukotinama (slika 2). Pukotina se karakterizira kao djelomično odvajanje materijala, obično uzrokovano vibracijama i koncentracijom naprezanja oko zareza te cikličkim naprezanjem zbog višestrukog slijetanja i polijetanja u jednom danu. Pukotine se šire prema području visokog opterećenja kao što su zarezi ili druga mjesta pukotina. Širenje do kritične veličine pukotine je prilično brzo i uzrokuje iznenadno pucanje trupa zrakoplova.

Slika 2. Pukotina nastala zbog velikih vibracija [3]

Veliki broj oštećenja nastaje zbog udara ptica u trup ili motore zrakoplova (slika 3.), kojih ima otprilike 13.000 godišnje samo u Sjedinjenim Američkim Državama [4]. Kako ptice rijetko lete na visina krstarenja zrakoplova, većina udara ptica nastaje prilikom polijetanja i slijetanja zrakoplova što samo potvrđuje da su to najrizičniji stadiji letenja. U prošlosti, zrakoplov se kretao dovoljno sporo i bio je dovoljna glasan zahvaljujući propelerima, no broj zabilježenih oštećenja zbog udara ptica narastao je zbog porasta zračnog prometa i zbog razvoja tiših zrakoplova.

Slika 3. Oštećenje trupa zrakoplova zbog udara ptice [5]

Grom u prosjeku pogodi zrakoplov jednom godišnje i u pravilu pogađa dijelove zrakoplova poput vrha krila ili nosa. Naboj zatim putuje kroz metalni trup aviona prije nego izađe kroz drugu točku, naprimjer rep. Kada komercijalne zrakoplove udari grom, može doći do ozbiljnih oštećenja koja zahtijevaju opsežne popravke ali u većini slučajeva ta su oštećenja nepostojeća Udar groma može biti prilično neugodno iskustvo za posadu zrakoplova i putnike, a nekad putnici nisu ni svjesni da se ono dogodilo. Nakon udara groma (slika 4) temeljita inspekcija zrakoplova je obavezna kako bi se

osigurala njegova plovidbenost, pa tako ono može uzrokovati kašnjenja i usporavanje operacija zrakoplovnih prijevoznika koje mogu koštati kompanije do 2 milijarde dolara godišnje.

Slika 4. Oštećenje horizontalnog stabilizatora zbog udara groma [6]

Jedno od tipičnih oštećenja koje nastaje prilikom tvrdog slijetanja je nabor oplate trupa zrakoplova (slika 5). Svaka komponenta trupa zrakoplova konstruirana je tako da se smanji ukupna težina zrakoplova, ali, u isto vrijeme, svaki dio uključujući oplatu trupa mora podnijeti opterećenje s glavnim nosivim komponentama konstrukcije aviona. Prilikom tvrdog slijetanja slabiji dijelovi, kao što su oplata, ponekad popuste što za posljedicu ima, u slučaju popuštanja oplate, pojavu nabora na trupu zrakoplova.

Slika 5. Nabor oplate trupa zrakoplova uslijed tvrdog slijetanja [7]

3. UVOD U RAČUNALNI VID

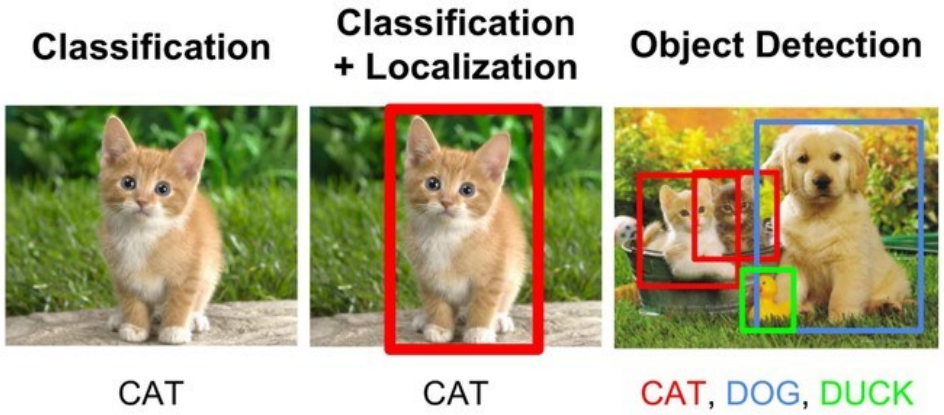
Računalni vid je skup algoritama koji omogućuju računalu vidjeti, promatrati i snalaziti se u prostoru. Iako su ti algoritmi prisutni u različitim oblicima od 1960-ih, tek se u posljednjem desetljeću ubrzao razvoj računalnog vida zbog razvoja same tehnologije odnosno bolje mogućnosti pohrane velikih količina podataka, računalne snage koja je postala pristupačnija te jeftinih i kvalitetnih uređaja za unos podataka iz stvarnog svijeta. Računalni vid se smatra potpodručjem umjetne inteligencije i strojnog učenja te pomoću njega računalo izvlači informacije iz digitalnih slika i videozapisa na temelju kojih onda poduzima određenu radnju. Iako se računalni vid temelji na ljudskom vidu, nedostaje mu prednost konteksta koji su ljudi stekli tijekom života pa je zato potrebna ogromna količina podataka kako bi računalo naučilo raspoznavati objekte, njihovu interakciju, kreću li se itd. Kada jednom to nauči, postaje neusporedivo brži i točniji u raspoznavanju objekata od ljudi. Danas se računalni vid primjenjuje u velikom broju područja, a neka od njih su:

- Medicina – detekcija tumora i drugih malignih promjena, mjerenja dimenzija organa, poboljšanje ultrazvučnih ili rendgenskih slika odnosno smanjenje buke na istima kako bi ih ljudi lakše protumačili, mjerenje gubitka krvi
- Proizvodnja – provjera kvalitete, inspekcija radi oštećenja, mjerenje položaja i orijentacije proizvoda kako bi ih mogla uhvatiti robotska ruka, optičko sortiranje proizvoda i hrane, brojanje i razvrstavanje predmeta na pokretnoj traci
- Vojna industrija – detekcija neprijateljskih vozila i vojnika, navođenje projektila, nadzor, pomoć u otkrivanju nagaznih mina
- Autonomna vozila – podržavanje vozača ili pilota u različitim situacijama, navigacija, izrada karte okruženja, otkrivanje prepreka, detekcija požara pomoću drona

3.1 Zadaci računalnog vida

Za primjenu u različitim aplikacijama računalni vid se koristi skupom zadaća, a neke od njih su klasificiranje slika, lokalizacija, detekcija, indetifikacija i praćenje objekata te segmentacija slika.
Klasificiranje slika bavi se klasificiranjem cijele slike u unaprijed definiranu kategoriju i to je jedan od najčešćih zadataka računalnog vida. Lokalizacija ide jedan korak dalje te pronalazi lokaciju jednog ili više objekata iste klase na slici i smješta ih u granični okvir. Primjer korištenja lokalizacije

je automatsko izrezivanje objekata iz skupa slika. Detekcija objekata izazovnija je i kombinira klasifikaciju i lokalizaciju, odnosno predviđa lokacije više objekata različitih klasa i smješta ih u granične okvire tih klasa (slika 6).

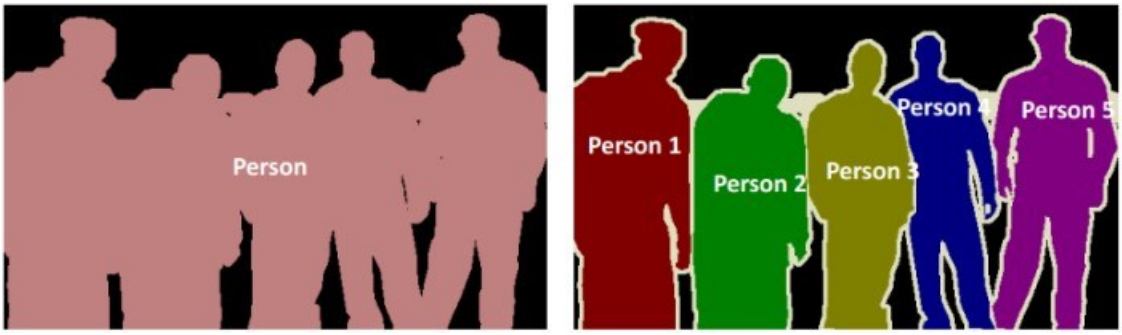


Slika 6. Usporedba klasifikacije, lokalizacije i detekcije [8]

Iako se slične tehnike koriste kod identifikacije objekata kao i kod detekcije, razlika je u tome što je kod identifikacije cilj odrediti pojavljuje li se određeni objekt na slici ili ne, te ako se pojavljuje specificirati mjesto na kojemu se pojavljuje.

Praćenje objekata neophodna je funkcija koju roboti moraju biti u stanju provoditi kako bi uspješno obavljali većinu zadataka. Vrši se putem provođenja detekcije objekta na svaku sliku u video sekvenci i usporedbom lokacije detektiranog objekta s lokacijom iz prošle sekvence kako bi se ustanovilo u kojem se smjeru objekt kreće. Nedostatak ovog pristupa je taj što je detekcija objekata za svaku pojedinačnu sliku obično skupo. Alternativni pristup bio bi detektirati objekt koji se prati samo jednom, odnosno prvi puta kada se pojavi, a zatim razaznati kretanje tog objekta bez eksplicitnog prepoznavanja na sljedećim slikama.

Segmentacija slike je metoda u kojoj se slika raščlanjiva na različite podskupine koje se nazivaju segmenti slike, što pomaže u smanjenju složenosti i olakšava daljnju obrada ili analizu slike. Metoda segmentacije se općenito dijeli na semantičku segmentaciju i segmentaciju instanci slika ili videa. Kod semantičke segmentacije svaki piksel na slici pripada određenoj klasi koja je predstavljena određenom bojom, a u slučaju više objekata iste klase računalo će ih smatrati kao jedan objekt. Kod segmentacije instanci svakom je pikselu također dodijeljena određena klasa no objekti iste klase izdvojeni su kao zasebni objekti različitim bojama (slika 7).



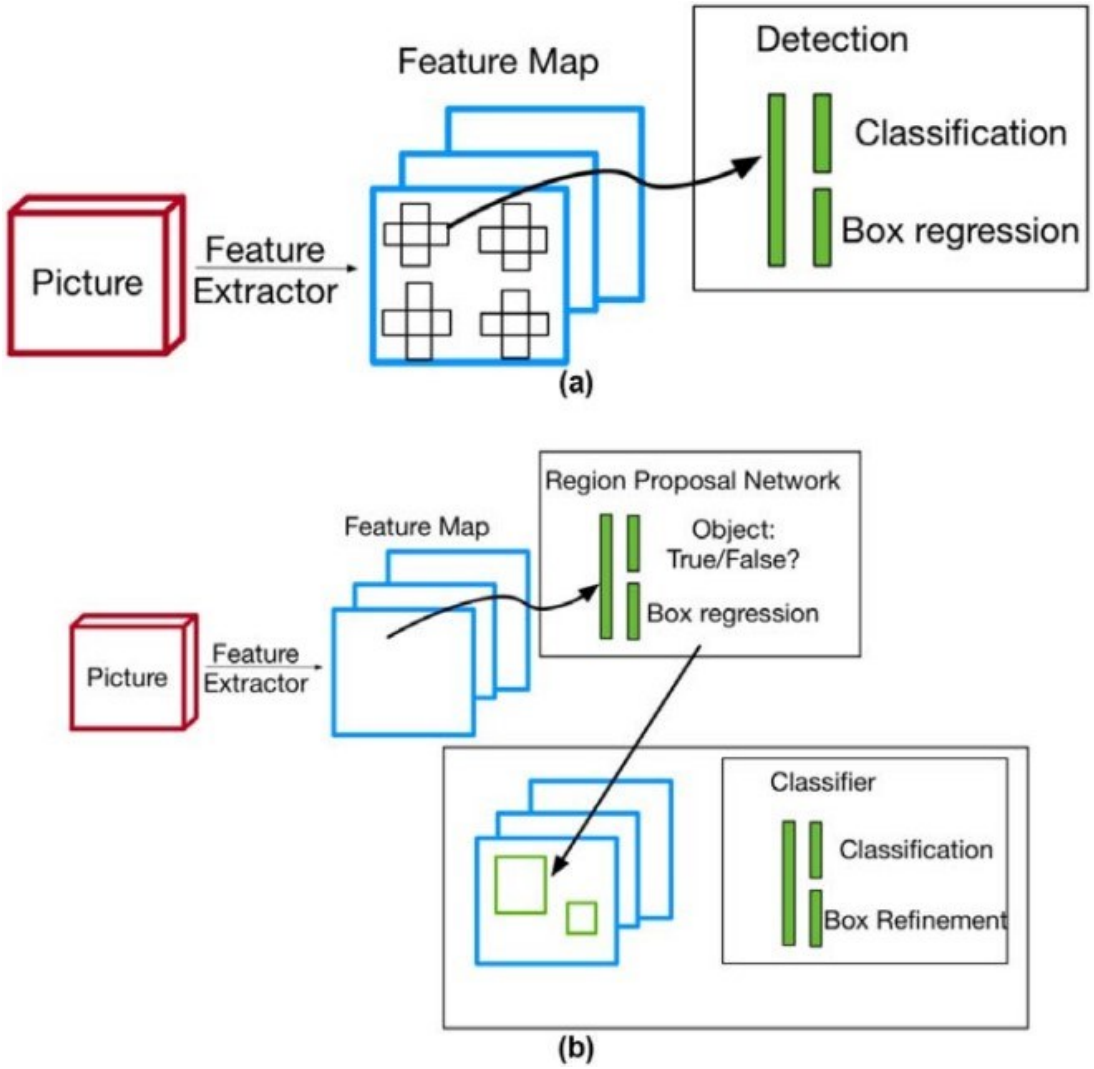
Slika 7. Semantička segmentacija (lijevo) i segmentacija instanci (desno) [9]

3.2 Vrste detektora

Detektori se općenito dijele na detektore koji se temelje na pristupu strojnog učenja, odnosno na detektore koji koriste tradicionalne tehnike procesuiranja slika i na detektore koji se temelje na metodama dubokog učenja. Kod tradicionalnih pristupa tehnike računalnog vida koriste se za ekstrakciju različitih značajki iz slike, kao što su histogram boje ili rubovi, kako bi se identificirale skupine piksela koji mogu pripadati objektu. Te se značajke zatim unose u regresijski model koji predviđa lokaciju objekta i njegovu klasu. Prednost detektora koji koriste tradicionalne tehnike procesuiranja slika je ta što ne zahtijevaju unaprijed označene slike za trening. Nedostatak takvih detektora je što su tehnike kojima se koriste ograničene u slučaju složenih scenarija, okluzije odnosno djelomično skrivenih objekata, lošeg osvjetljenja, sjena te puno različitih objekata na slici. Detektori koji se temelje na metodama dubokog učenja koriste konvolucijske neuronske mreže (*convolutional neural networks* ili *CNN*) za ekstrakciju značajki koje ne trebaju biti strogo definirane. Prednost detektora koji se temelje na metodama dubokog učenja su znatno bolja otpornost na okluziju, složene scene i različito osvjetljenje, a nedostatak je ogromna količina podataka koja je potrebna za trening detektora, vrijeme potrebno kako bi se ti podaci označili i potreba za velikom računskom snagom GPU-a o kojoj ovisi brzina treniranja detektora.

Detektori koji se temelje na dubokom učenju mogu se podijeliti u dvije kategorije: dvofazni (*two-stage)* detektori i jednofazni (*single-stage)* detektori. U prvoj fazi *two-stage* detektora značajke se izdvajaju iz slike i predlažu se regija interesa (*region of interest - ROI*) koje se sastoje od potencijalnih okvira u kojima bi se objekt mogao nalaziti na slici. U drugoj fazi koriste se izdvojene značajke i *ROI* za izračunavanje konačnih graničnih okvira i vjerojatnosti klasa za svaki od okvira. [10]. Ovi detektori ostvaruju veliku preciznost ali su sporiji u odnosu na *single-stage* detektore. Neki od najpoznatijih *two-state* detektora su *RCNN*, *Fast RCNN*, *Faster RCNN* i *Mask-RCNN*. Za razliku od *two-stage* detektora, *single-stage* detektoru je potreban samo jedan prolaz slike kroz

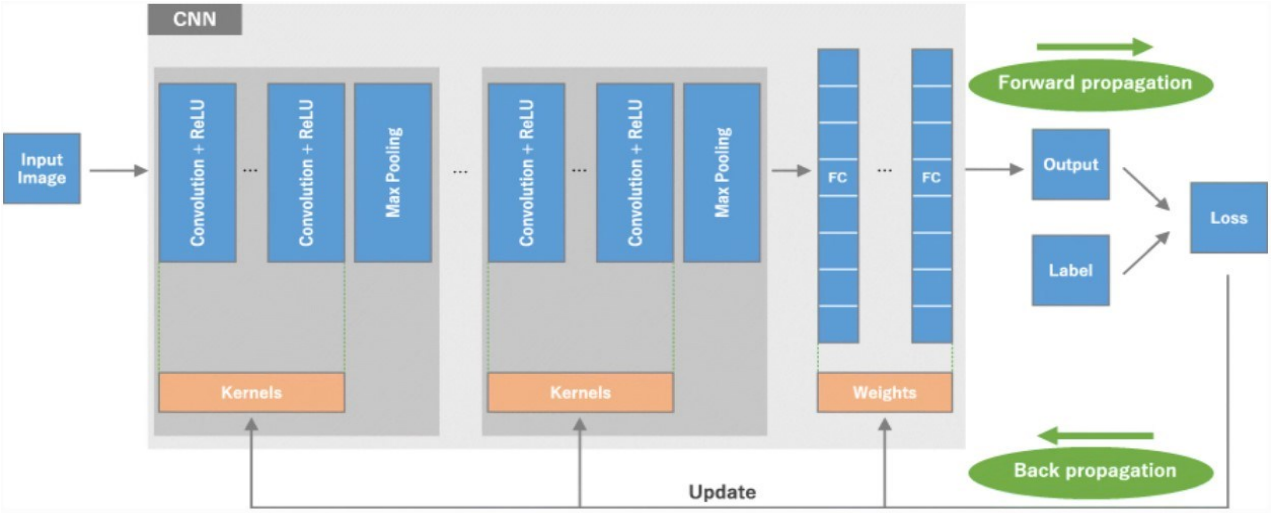
konvolucijsku neuronsku mrežu kako bi model predvidio granične okvire svih objekata na slici. To rezultira velikom brzinom ali i manjom preciznosti modela. Neki od najvažnijih *single-stage* detektora su *SSD*, *RetinNet* i *YOLO* familija detektora. Usporedba *single-stage* i *two-stage* detektora prikazana je na slici slici 8.



Slika 8. Shematski prikaz a) single-stage i b) two-stage detektora [11]

4. KONVOLUCIJSKE NEURONSKE MREŽE

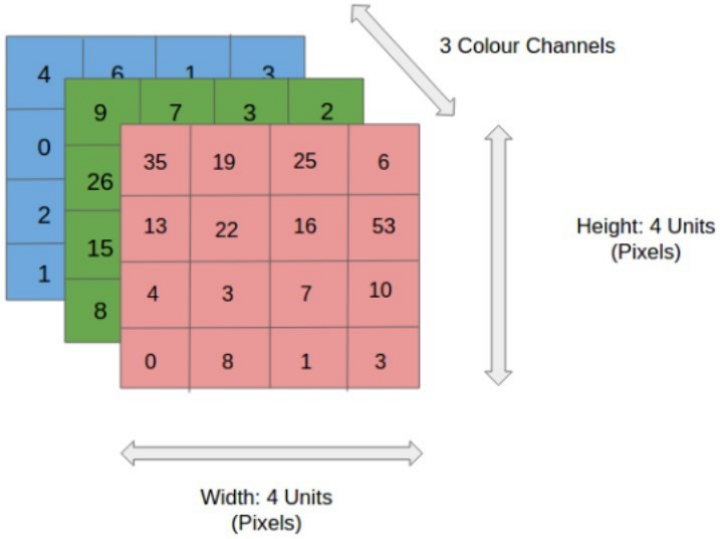
Konvolucijska neuronska mreža (*Convolutional Neural Network - CNN*) vrsta je umjetne neuronske mreže koja svoju primjenu pronalazi u zadacima računalnog vida. Ime je dobila zbog primjene konvolucije, matematičke operacije koja se vrši na dvjema funkcijama, čime se dobiva treća funkcija koja opisuje kako jedna funkcija mijenja drugu funkciju. *CNN* se koristi za procesuiranje podataka koji su strukturirani u matrični oblik, kao što su slike, a inspirirana je strukturom životinjskog vizualnog korteksa. Klasična konvolucijska mreža sastoji se od četiri vrste slojeva: konvolucijskog sloja, *ReLU* sloja i sloja objedinjavanja *(pooling layer)* koji služe kao ekstraktor značajki, te potpuno povezanog sloja *(fully connected layer)* koji služi za klasifikaciju slike u određenu kategoriju. Dubina mreže ovisi o broju serijski povezanih blokova. U konvolucijskom sloju značajke se izdvajaju pomoću filtera *(kernels)*, koji su ustvari matrice parametara, koji se optimiziraju tijekom treninga. Zapravo je cijela ideja sljedeća: na ulaznu sliku se primjeni unaprijed određeni broj nasumično izabranih filtera pomoću kojih se dobivaju određene značajke, kao što su vertikalni i horizontalni rubovi, boja itd. Na temelju tih značajki, koje prolaskom kroz mrežu postaju sve kompleksnije, slika se klasificira u određenu kategoriju a zatim se ta predikcija uspoređuje s temeljnom istinom, odnosno pravom kategorijom slike, te se računa pogreška pri klasifikaciji. Nakon toga se putem propagacije unatrag (*backpropagation)* i metode gradijentnog spusta ti filteri prilagođavaju kako bi izdvojili značajke pomoću kojih će se slika što preciznije klasificirati u pravu kategoriju (slika 9).



Slika 9. Arhitektura i proces treniranja konvolucijske neuronske mreže [12]

4.1 Ulazna slika

Slika je u računalu predstavljena kao niz brojeva. U slučaju crno-bijele slike to je jedna matrica koja sadrži brojeve između 0 i 255. Svaki od tih brojeva odgovara svjetlini piksela, pri čemu 0 odgovara crnoj boji a 255 bijeloj boji. Slika u boji se sastoji od tri kanala, odnosno matrica, koji predstavljaju crvenu, zelenu i plavu boju. Spajanjem tih kanala (slika 10) dobiva se slika u boji koja se također naziva i RGB (*red-green-blue*) slika.

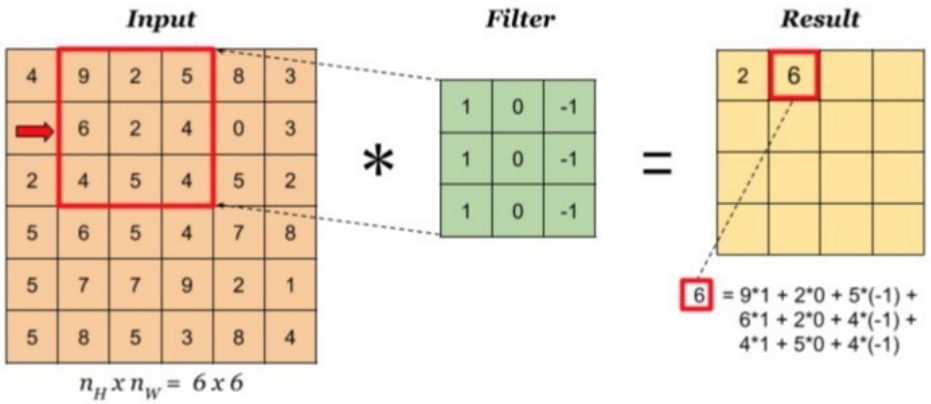


Slika 10. Kanali slike u boji [13]

Na slici 5. prikazana je slika rezolucije $[4 \times 4 \times 3]$ no pri slikama veće rezolucije kao što su 4K ($[3840 \times 2160 \times 3]$) ili 8K ($[7680 \times 4320 \times 3]$) reduciranje slike u oblik koji je lakši za obradu tako da se ne izgube ključne značajke važna je zadaća konvolucijske mreže.

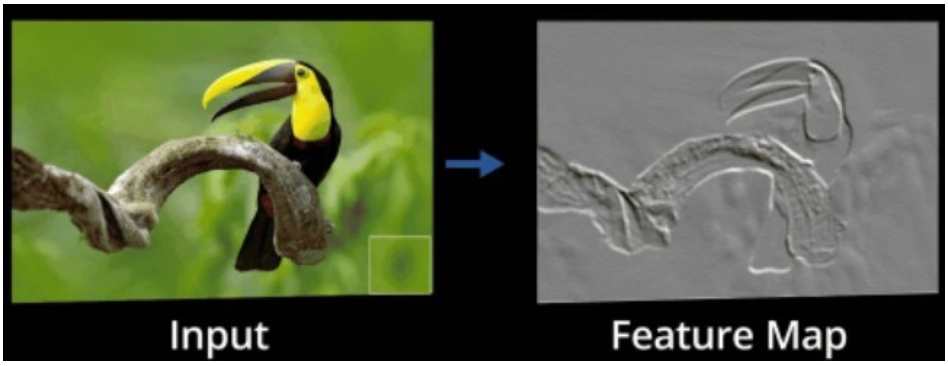
4.2 Konvolucijski sloj

Konvolucijski sloj se obično sastoji od kombinacije linearnih i nelinearnih operacija, tj. operacije konvolucije i aktivacijske funkcije. Predstavlja temeljnu komponentu arhitekture *CNN*-a koja provodi ekstrakciju značajki pomoću konvolucije, gdje se mali niz brojeva, nazvan filter, primjenjuje na ulazni tenzor odnosno sliku (slika 11) čime se dobiva mapa značajki *(feature map)*.



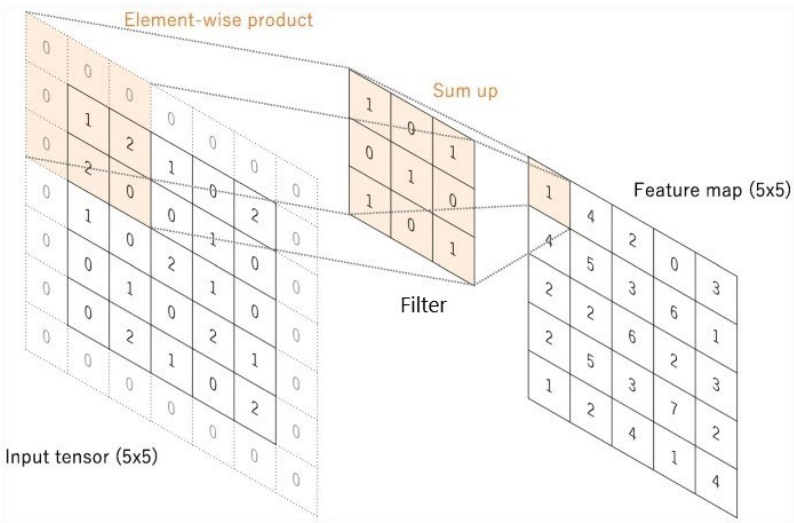
Slika 11. Dobivanje mape značajki pomoću konvolucije [14]

Proizvoljan broj mapi značajki dobiva se primjenom više filtera te one predstavljaju različite karakteristike ulaznih tenzora pa se stoga različiti filteri smatraju ekstraktorima različitih značajki. Veličina filtera i broj filtera dva su ključna parametra koji definiraju operaciju konvolucije. Veličina filtera je tipično 3×3, a ponekad i 5×5 ili 7×7 dok je broj filtera proizvoljan i određuje dubinu izlaznih mapa značajki. Ako ulazni tenzor ima 3 kanala, što je slučaj kod RGB slika, visina i širina filtera ostat će isti ali će dubina filtera odgovarati dubini ulaznog tenzora a dobiveni rezultati od sva tri kanala zbrajaju se u jedan izlaz. Primjer dobivene mape značajki nakon primjene filtera za detekciju rubova prikazan je na slici 12.



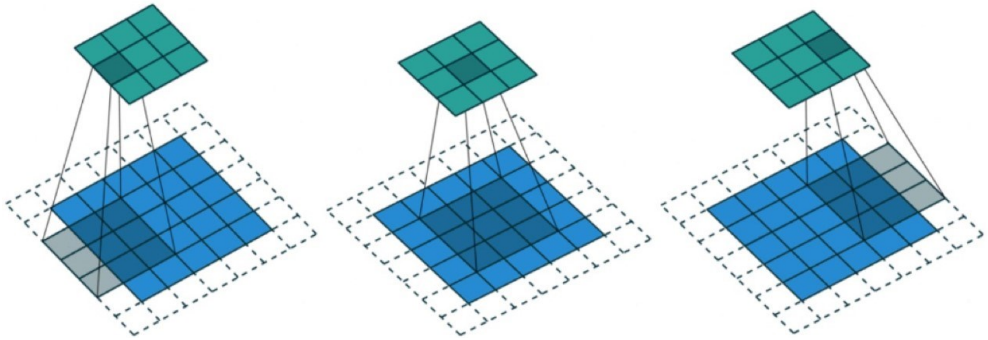
Slika 12. Mapa značajki [15]

Prilikom konvolucije smanjuje se visina i širina izlazne mape značajki zbog toga što sredina filtera ne može zahvatiti rubove ulaznog tenzora. Taj problem se rješava uvođenjem tehnike popunjavanja, odnosno *padding*-a, kojom se ulazni tenzor proširuje tako da filter može zahvatiti rubove i očuvati istu dimenziju ulaznog tenzora. U slučaju da se tenzor proširi nulama govorimo o popunjavanju nulama *(zero padding)*, koji se također naziva i *Same padding* zbog očuvanja dimenzija širine i visine ulaznog tenzora (slika 13).



Slika 13. Popunjavanje mape značajki primjenom tehnike *Zero padding* [12]

Pomak filtera po širini ulaznog tenzora naziva se korak *(stride)* filtera. Kada filter prijeđe širinu tenzora spušta se korak prema dolje i ponavlja proces. Određivanjem vrijednosti koraka određuje se veličina izlazne mape značajki. Vrijednost koraka najčešće je 1 te u kombinaciji sa *zero padding*-om širina i visina izlaznog tenzora odgovarat će širini i visini ulaznog tenzora. no ako se dimenzije ulaznog tenzora žele smanjiti za korak se mogu izabrati i veće vrijednosti (slika 14).



Slika 14. Primjena konvolucijske operacije *stride = 2* [13]

Primjenom filtera, odnosno konvolucijom, dobivaju se mape značajki i za izlaz se dobiva tenzor kojemu širina i visina ovise o širini, visini i koraku filtera. Dubina izlaznog tenzora, odnosno broj generiranih mapi značajki, odgovara broju filtera.

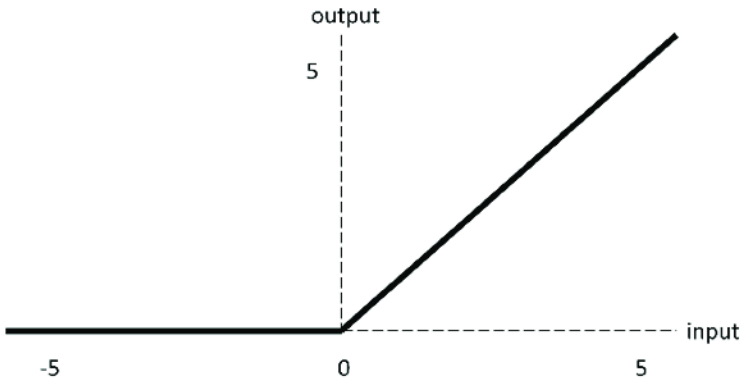
Veličina izlaznog tenzora (1) računa se pomoću formule [16]:

$$W_2 = \frac{W_1 - F + 2P}{S_F} + 1, \qquad \textbf{(1)}$$

gdje je:

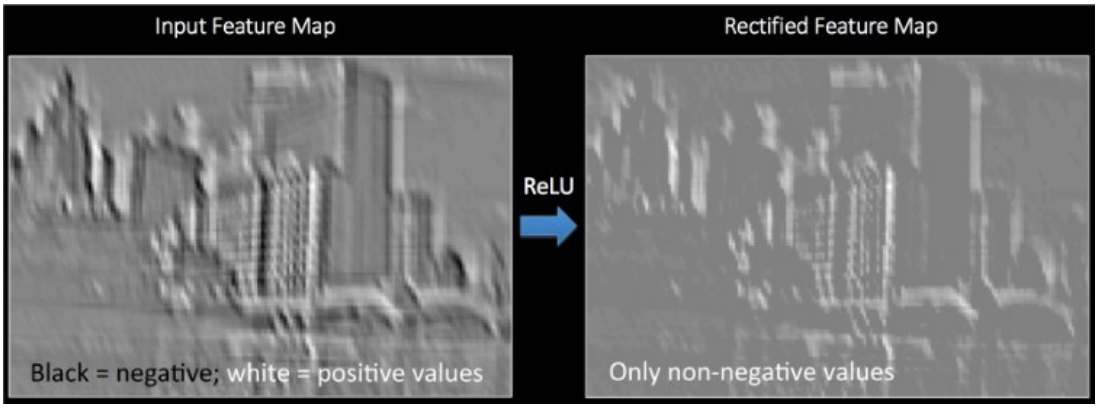
- W_2 – visina/širina izlaznog tenzora
- W_1 – visina/širina ulaznog tenzora
- F – visina/širina filtera
- P – popunjavanje (*padding*)
- S_F – korak filtera (*stride*)

Takav izlazni tenzor zatim prolazi kroz nelinearnu aktivacijsku funkciju a najpopularnija je *rectified linear unit (ReLU)* aktivacijska funkcija. To je funkcija koja postavlja sve negativna vrijednosti piksela na nulu te je prikazana na slici 15. Konvolucija je linearna operacija stoga se *ReLU* koristi kako bi se uvela nelinearnost mape značajki i kako bi pomogla mreži iskoristiti važne i potisnuti nebitne informacije čime se ubrzava cjelokupan proces treniranja mreže.



Slika 15. *ReLU* aktivacijska funkcija [17]

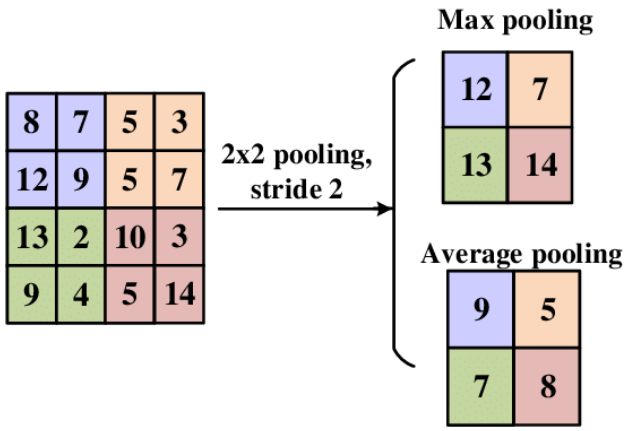
Primjenom *ReLU* aktivacijske funkcije dobiva se ispravljena mapa značajki prikazana na slici 16.



Slika 16. Ispravljena mapa značajki primjenom *ReLU* aktivacijske funkcije [18]

4.3 Sloj objedinjavanja

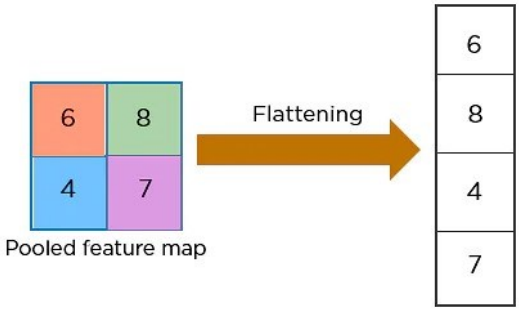
U sloju objedinjavanja se provodi operacija smanjenja dimenzija ispravljenih mapa značajki kako bi se smanjio broj parametara koji mreža mora naučiti (izdvaja dominantne značajke) i količina izračunavanja u mreži. Ovaj sloj ne sadrži parametre koje mreža uči tijekom treninga već je određen hiperparametrima, odnosno unaprijed zadanim parametrima, kao što su veličina filtera, korak filtera i popunjavanje. Dvije vrste objedinjavanja koje se koriste u konvolucijskim neuronskim mrežama su maksimalno objedinjavanje (m*ax pooling)* i prosječno objedinjavanje (*average pooling)* prikazani na slici 17. Maksimalno objedinjavanje je najpopularnija operacija objedinjavanja koja vraća maksimalnu vrijednost iz dijela slike koji pokriva filter te služi kao metoda smanjenja šuma. U praksi se uobičajeno maksimalno objedinjavanje provodi s filterom veličine 2×2 s korakom od 2 što uzrokuje smanjenje mape značajki za faktor 2. *Average pooling* vraća prosjek svih vrijednosti iz dijela slike prekrivenog filterom. Za razliku od visine i širine, dimenzija dubine mapi značajki nakon sloja objedinjavanja ostaje nepromijenjena.



Slika 17. Maksimalno i prosječno objedinjavanje [19]

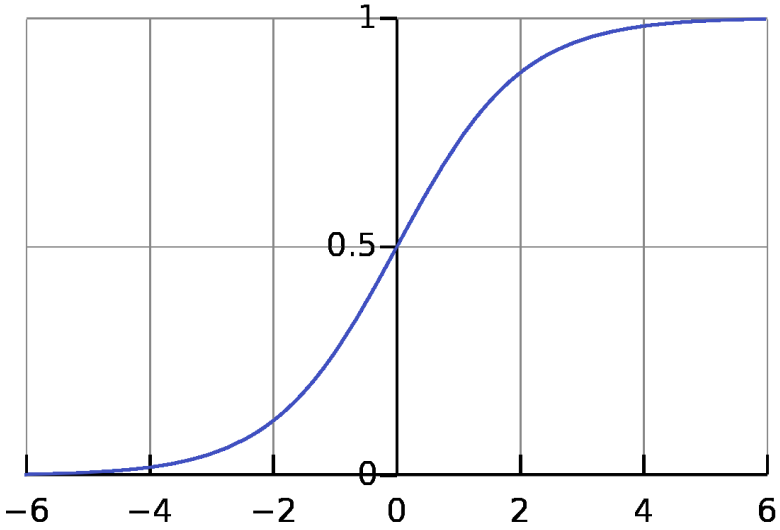
4.4 Potpuno povezani sloj

Nakon što se dobivena mapa značajki pomoću finalnog konvolucijskog sloja smanji u sloju objedinjavanja, ta mapa se transformira u vektor (slika 18) koji je povezan sa jednim ili više potpuno povezanih slojeva u kojima je svaki ulaz povezan sa svakim izlazom pomoću naučenog parametra.



Slika 18. Transformacija mape značajki u potpuno povezani sloj [15]

Zadnji potpuno povezani sloj klasificira sliku u određenu kategoriju pomoću aktivacijske funkcije. Ako se mreža trenira na više klasa, za klasifikaciju se koristi *Softmax* aktivacijska funkcija (slika 19) koja normalizira izlazne vrijednosti iz potpuno povezanog sloja na vjerojatnosti klase, gdje se svaka vrijednost kreće u rasponu [0,1] a zbroj vjerojatnosti klasa iznosi 1.



Slika 19. *Softmax* aktivacijska funkcija [20]

4.5 Treniranje mreže

Trening mreže je proces koji se provodi na skupu slika za trening kako bi se pronašli filteri u slojevima konvolucije koji minimiziraju razlike između predviđenog izlaza i zadane temeljne istine. Propagacija unatrag je metoda koja se obično koristi za treniranje neuronskih mreža gdje funkcija gubitka i optimizacijska metoda gradijentnog spusta igraju bitnu ulogu. Performansa modela pri određenim filterima izračunava se funkcijom gubitka pomoću propagacije prema naprijed na skupu podataka za trening, a parametri, odnosno filteri, koji se mogu naučiti ažuriraju se u odnosu na vrijednosti gubitka putem propagacije unatrag i metode gradijentnog spusta. Kada se govori o propagaciji prema naprijed jednostavno se misli na tok informacija kroz neuronsku mrežu koji ide

samo u jednom smjeru odnosno od ulaza prema izlazu, a kad se govori o propagaciji unazad misli se na tok informacija u obrnutom smjeru.

4.5.1 Funkcija gubitka

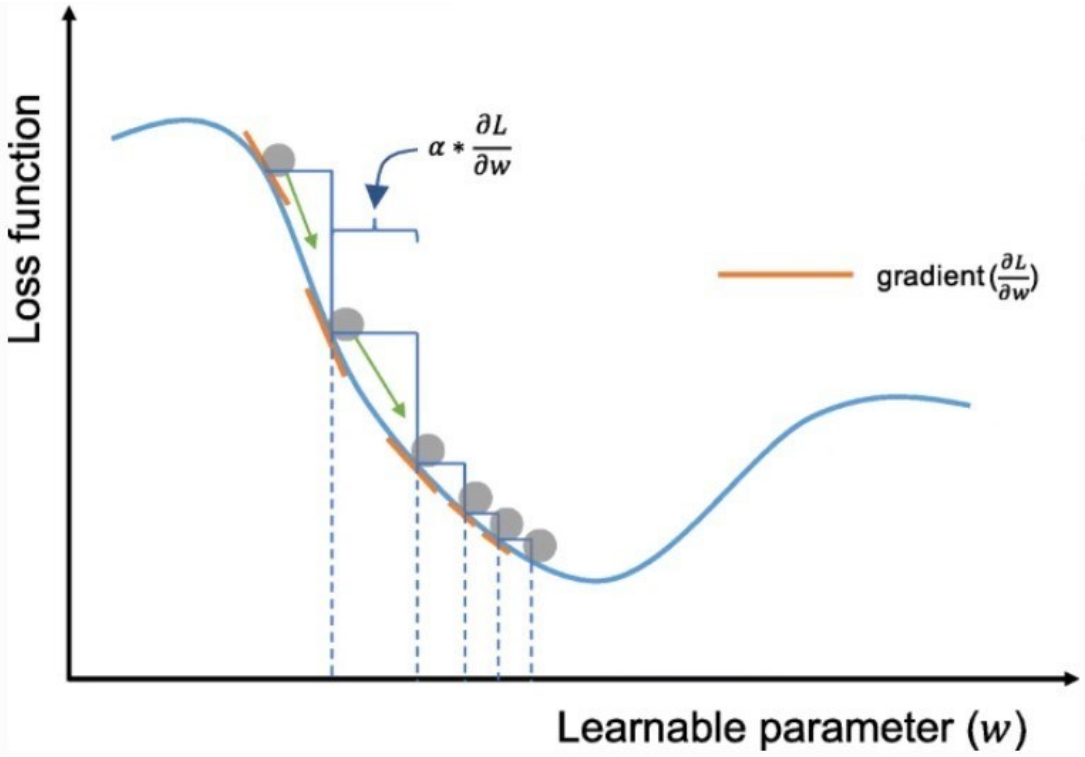
Funkcija gubitka, koja se također naziva i funkcija troškova, mjeri kompatibilnost između predviđenih izlaza mreže i zadane temeljne istine pomoću propagacije prema naprijed. Drugim riječima, funkcija gubitka jednostavno računa koliko je mreža „promašila“ u klasificiranju slike u određenu kategoriju. Najčešća korištena funkcija gubitka kod višeklasne klasifikacije je unakrsna entropija no mogu se koristiti i druge funkcije gubitka. Unakrsna entropija je mjera razlike između dvije distribucije vjerojatnosti za danu slučajnu varijablu ili skup događaja. Vrsta funkcije gubitka koja se koristi prilikom treninga mreže jedan je od hiperparametara i treba je odrediti sukladno problemu koji se rješava.

4.5.2 Gradijentni spust

Gradijentni spust jedan je od najčešćih algoritama koji se koriste pri optimizaciji neuronskih mreža. Pomoću njega se prilagođavaju parametri, odnosno filteri, koje mreža uči tijekom treninga kako bi se smanjio gubitak odnosno poboljšala preciznost mreže (slika 20). Izračunavanjem gradijenta funkcije gubitka dobiva se vrijednost nagiba tangente na tu funkciju koja pokazuje u kojem smjeru funkcija gubitka ima najveći porast te se parametri ažuriraju s pomakom u suprotnom smjeru gradijenta dok se ne pronađe globalni minimum funkcije. Pomak je određen hiperparametrom koji se naziva brzina učenja *(learning rate)* te je isti jedan od najvažnijih hiperparametara koje treba postaviti prije učenja. Gradijent je parcijalna derivacija funkcije gubitka u odnosu na parametar koji se uči tijekom treniranja mreže, a jedno ažuriranje parametra računa se pomoću jednadžbe (2):

$$w = w - \alpha * \frac{\partial L}{\partial w}, \qquad \textbf{(2)}$$

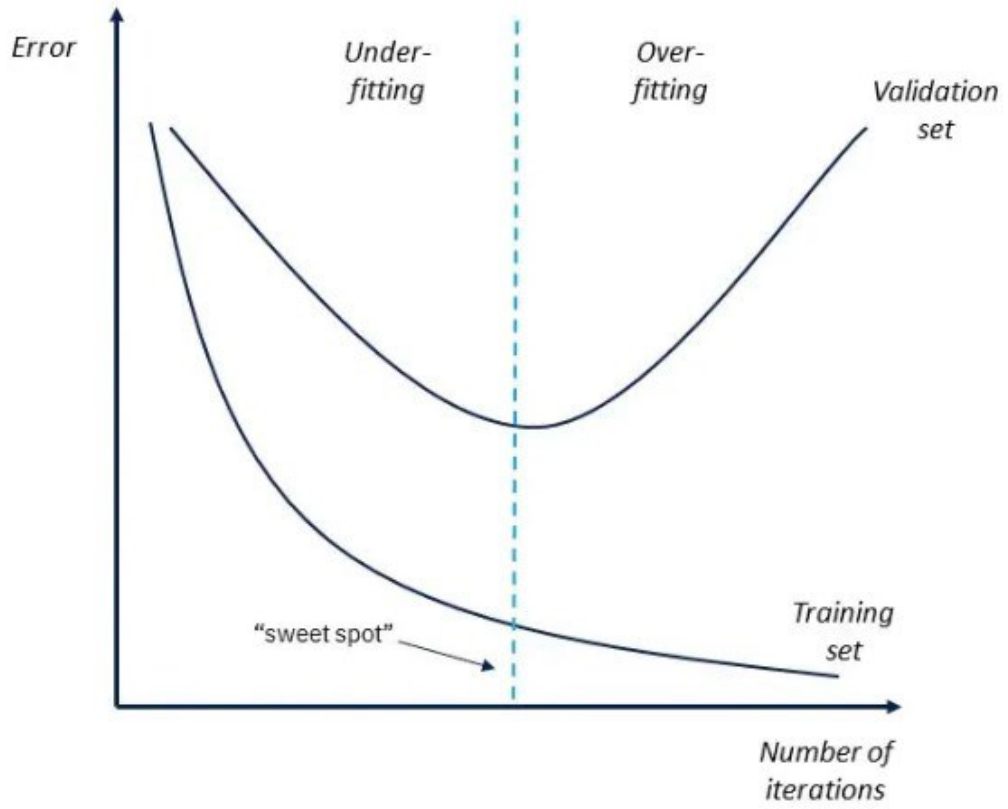
gdje *w* predstavlja svaki parametar koji se može naučiti, α predstavlja brzinu učenja a *L* funkciju gubitka [21].



Slika 20. Optimizacija parametara učenja neuronske mreže metodom gradijentnog spusta [21]

4.5.3 Overfitting

Prekomjerno prilagođavanje *(overfitting)* ili prenaučenost znači da model dobro funkcionira na skupu podataka (slika) za učenje (trening), a loše na skupu slika za validaciju. Do prenaučenosti dolazi kada mreža nauči značajke prilikom treninga do te mjere da to negativno utječe na performansu modela. Budući da prenaučen model nije primjenjiv na novi skup podataka tj. slika, prenaučenost predstavlja jedan od glavnih izazova u strojnom učenju. Provjera za prepoznavanje prekomjernog prilagođavanja je praćenje gubitka i točnosti skupova za učenje i validaciju (slika 21.). Neke od metoda kojima se može izbjeći prenaučenost su povećanje skupa slika za učenje, regularizacija, augmentacija odnosno umjetno izmjenjivanje značajki slika, ranije zaustavljanje treniranja mreže, smanjivanje kompleksnosti arhitekture mreže itd. Kada naučeni model mreže daje loše rezultate na skupu slika za treniranje i validaciju to se zove *underfitting* i događa se kada model nije treniran dovoljno dugo ili ulazne varijable nisu dovoljno značajne da bi se utvrdio smisleni odnos između ulaznih i izlaznih varijabli.



Slika 21. Prikaz *overfitting*-a i *underfitting*-a podataka [21]

5. *"YOLO"* ALGORITMI ZA DETEKCIJU OBJEKATA

5.1 Općenito

"You only look once", skraćeno YOLO, serija je suvremenih modela dubokog učenja koji koriste neuronske mreže za detektiranje objekata u stvarnom vremenu. Prve verzije razvili su Joseph Chet Redmon i suradnici 2015. godine, inspirirani *GoogleNet* konvolucijskom mrežom dubokom 22 sloja. Model je dobio ime po tome što je potreban samo jedan prolaz slike kroz mrežu kako bi se detektirali objekti, za razliku od prijašnjih modela kroz koje slika prolazi više puta. Modeli kao što su *RCNN* i *Fast-RCNN* problem detekcije objekata definiraju kao problem klasifikacije, dok YOLO algoritmi pristupaju tom problemu kao problemu regresije. Zbog svoje brzine i točnosti široku primjenu pronalaze u sustavima autonomne vožnje, nadzornim sustavima, raznim aplikacijama za detekciju ljudi, životinja, u industrijskoj inspekciji te održavanju itd. Trenutno postoji 5 inačica YOLO modela za detekciju, a svaka je unaprijeđena verzija prethodnog modela. J. Redmon službeno je razvio prve 3 verzije modela naziva YOLO, YOLO v2 (YOLO9000 – verzija YOLO v2 modela koja prepoznaje 9000 klasa) i YOLO v3 objavljene 2016., 2017. i 2018 godine, nakon čega napušta projekt i cjelokupno područje istraživanja računalnog vida zbog brige o narušavanju privatnosti ljudi i zlouporabi modela u vojnom sektoru. 2020. godine, u razmaku od samo nekoliko mjeseci, objavljene su tri nove YOLO verzije pod nazivom YOLO v4, YOLO v5 i PP-YOLO. Ti su modeli proizvodi mnogih eksperimenata i studija koje kombiniraju različite nove tehnike koje poboljšavaju točnost i brzinu konvolucijske neuronske mreže. Važno je napomenuti da svaka od verzija također ima „tiny" verziju modela zbog problema brzine na ugrađenim uređajima kao što je Raspberry Pi. YOLO v1-Tiny model postiže više od 155 slika po sekundi (*Frames per seconds - FPS*) na jednoj grafičkoj kartici za razliku od 45 *FPS*-a normalnog modela (tablica 1). Također, brži je za otprilike 442% od arhitekture normalnog YOLO modela. Nedostatak svih *Tiny* modela je njihova manja preciznost zbog manjeg broja slojeva i manjeg broja filtera u konvolucijskoj neuronskoj mreži.

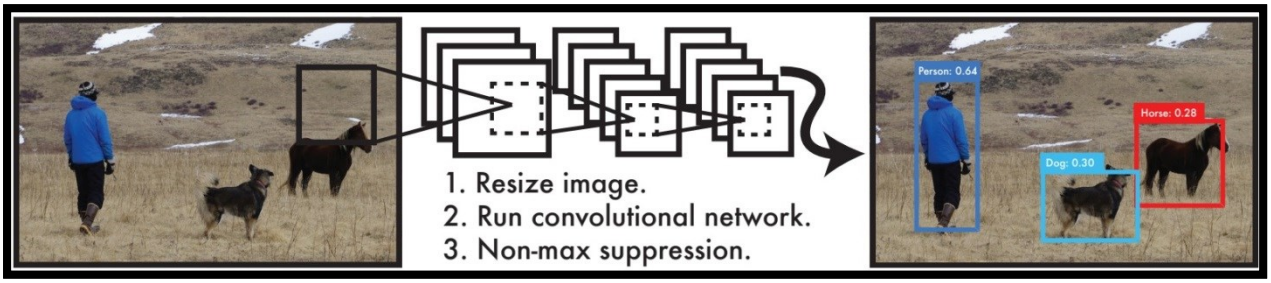
Tablica 1. Usporedba normalnih i *Tiny* modela YOLO detektora

Detektor	Broj n-slojeva	FPS	Preciznost (*Mean average precision*) mAP	Korišten skup slika za treniranje
YOLO v1	26.00	45.00	63.50	VOC
YOLO v1-Tiny	9.00	155.00	52.80	VOC
YOLO v2	32.00	40.00	48.20	COCO
YOLO v2-Tiny	16.00	244.00	23.60	COCO
YOLO v3	106.00	20.00	57.80	COCO
YOLO v3-Tiny	24.00	220.00	33.20	COCO

5.2 YOLO v1

Kao što je poznato, ljudi su sposobni obavljati složene zadatke, poput vožnje auta uz malo svjesnog razmišljanja, zbog brzog i učinkovitog ljudskog vizualnog sustava. Dovoljan im je jedan pogled na sliku kako bi mogli raspoznati objekte koji se nalaze na njoj, gdje se nalaze i u kakvoj su interakciji ti objekti jedni s drugima. Upravo na tom principu se temelji ovaj brzi i efikasni detektor – dovoljan je jedan prolaz slike kroz konvolucijsku neuronsku mrežu kako bi se detektiralo više objekata na slici, što ovaj detektor svrstava u grupu „*single-stage*“ detektora. Obrada slika pomoću YOLO detektora prilično je jednostavna – slici se promijeni veličina na unaprijed definiranu veličinu koja je ista ili veća od veličine na kojoj je trenirana konvolucijska neuronska mreža, takva slika provuče se kroz već istreniranu konvolucijsku neuronsku mrežu na klasama koje se žele detektirati te se detektirani objekti filtriraju pomoću unaprijed određene vrijednosti (slika 22). Zadaća konvolucijske mreže jest predvidjeti više graničnih okvira (*bounding boxes*) i vjerojatnosti

klasa za te okvire. Ovakva jednostavna struktura modela moguća je zbog pristupa detekciji kao problemu regresije i zaslužna je za brzinu koja je veća od ostalih „*two-stage*“ detektora. YOLOv1 model treniran je na *PASCAL VOC 2007* i *2012* skupu slika koji sadrže 20 kategorija.



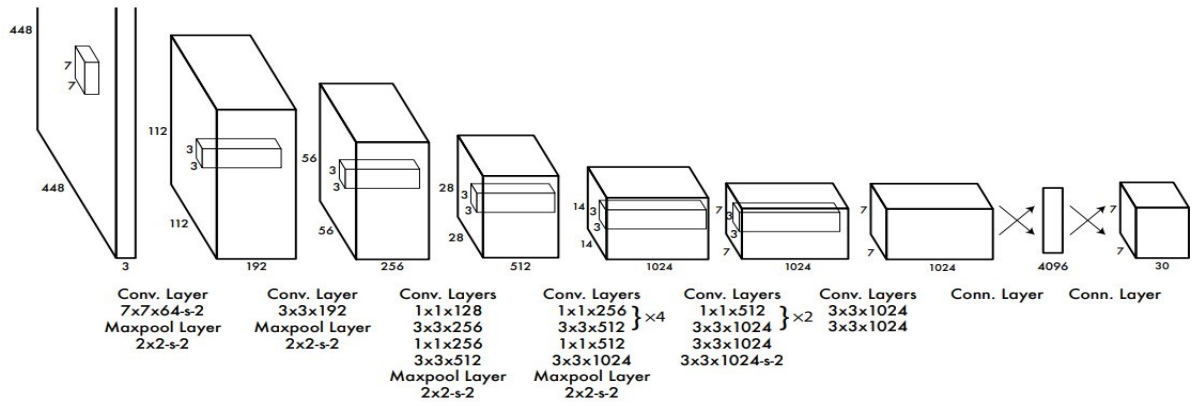
Slika 22. Primjer detekcije objekata YOLO detektorom [22]

5.2.1 Struktura YOLO modela

Struktura mreže temelji se na *GoogleNet* konvolucijskoj mreži. *GoogleNet* je konvolucijska neuronska mreža duboka 22 sloja a zadnji sloj je klasifikacijski sloj koji služi za klasifikaciju cijele slike u određenu kategoriju. Konvolucijska neuronska mreža YOLO v1 detektora sastoji se od 24 konvolucijska sloja nakon kojih slijede 2 „*fully connected*“ sloja (slika 23). Značajke (*features*) se izdvajaju iz slike pomoću početnih konvolucijskih slojeva mreže. U svrhu detekcije objekata zadnji sloj nije klasifikacijski sloj, kao u slučaju *GoogleNet* konvolucijske mreže, nego je to već spomenuti „*fully connected*“ sloj koji daje vjerojatnosti nalazi li se objekt na slici ili ne, koordinate graničnih okvira detektiranih objekata te predikciju njihovih klasa.

5.2.2 Pouzdanost i vjerojatnost

Kako bi se model mogao istrenirati za detekciju željenih klasa potrebno mu je uz sliku pridodati i koordinate graničnih okvira objekata koji se nalaze na slici. Tako je prilikom treninga model u mogućnosti usporediti predviđeni položaj i klasu detektiranog objekta s pravim (istinitim) položajima i klasama objekata. Takve slike s već označenim položajima objekata i njihovim klasama nazivaju se temeljna istina (*ground truth*).



Slika 23. Struktura YOLO modela [22]

Prvu operaciju koju algoritam čini kada dobije sliku jest dijeljenje slike na mrežu dimenzija *S x S* (slika 24). Ona ćelija u kojoj se nalazi središte objekta zadužena je za detektiranje tog objekta. Svaka ćelija mreže predviđa *B* graničnih okvira i ocjene pouzdanosti (*confidence scores*) koje govore o tome koliko je model siguran u preciznost graničnih okvira i koliko je siguran da se unutar tog graničnog okvira nalazi objekt. Pouzdanost (3) je definirana kao:

$$\Pr(Object) \cdot IoU_{pred}^{truth} \quad \textbf{(3)}$$

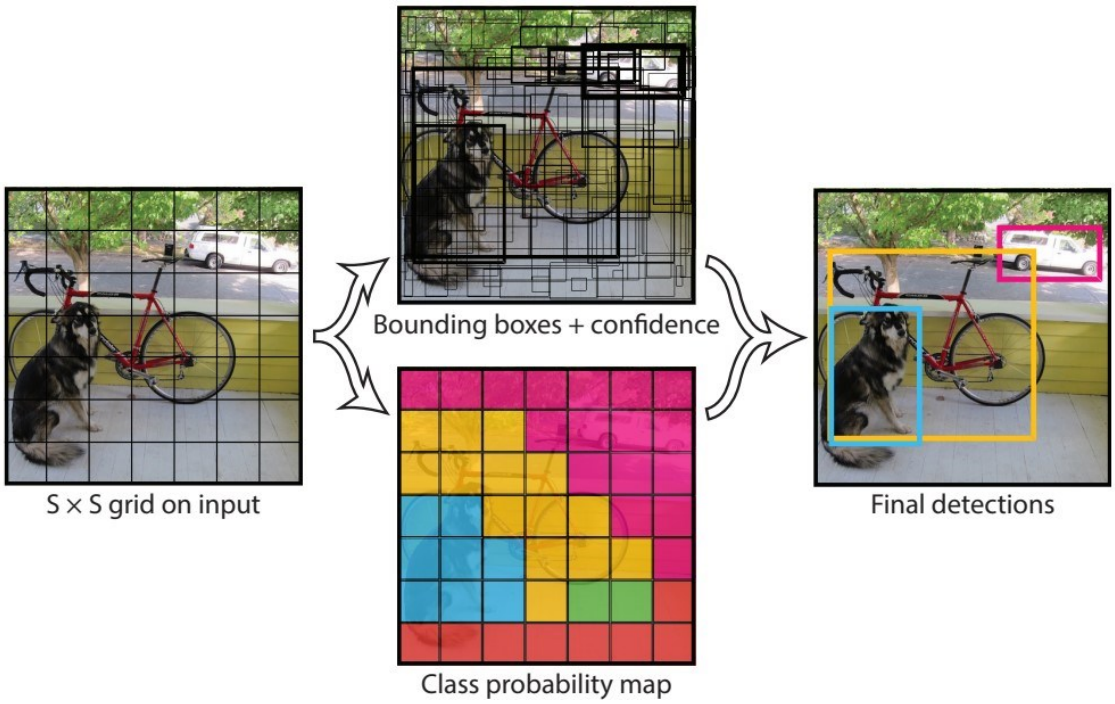
Gdje je:

- *Pr(Object)* – vjerojatnost (*probability*) da se objekt nalazi unutar graničnog okvira
- *IOU* - presjek preko unije (*intersection over union*)

Kako se vrijednost vjerojatnosti kreće između 0 i 1, pouzdanost je u idealnom slučaju jednaka vrijednosti *IoU*, a ako se objekt ne nalazi u ćeliji ta vrijednost jednaka je nuli. Svaki granični okvir koji model predviđa sastoji se od koordinata položaja središta graničnog okvira (b_x,b_y), visine i širine okvira (b_h,b_w) te predviđene vrijednosti pouzdanosti. Ćelije su normalizirane na [0,1] pa tako koordinate središta graničnog okvira uvijek poprimaju vrijednosti između 0 i 1 a visina i širina graničnog okvira mogu poprimati vrijednosti veće od 1 u slučaju prelaska graničnog okvira (pa i objekta) u drugu ćeliju. Konačno, predviđana pouzdanost definira se kao *IoU* između predviđenih graničnih okvira i istinitih graničnih okvira zadanih u temeljnoj istini. Vjerojatnosti klase *C*, $Pr(Class_i|Object)$, predviđaju se za svaku ćeliju mreže (*class probability map* na slici 24.). Bez obzira na broj graničnih okvira *B* predviđa se samo jedan skup vjerojatnosti klasa po ćeliji mreže. Rezultati pouzdanosti specifični za svaku klasu dobivaju se množenjem vjerojatnosti klasa s predviđenom pouzdanošću graničnih okvira:

$$\Pr(Class_i|Object) \cdot \Pr(Object) \cdot IoU_{pred}^{truth} = \Pr(Class_i) \cdot IoU_{pred}^{truth} \quad \textbf{(4)}$$

te definiraju kolika je vjerovatnost da se određena klasa pojavljuje u graničnom okviru i koliko dobro predviđeni granični okvir obuhvaća detektirani objekt.



Slika 24. Primjer detekcije objekata primjenom graničnih okvira [22]

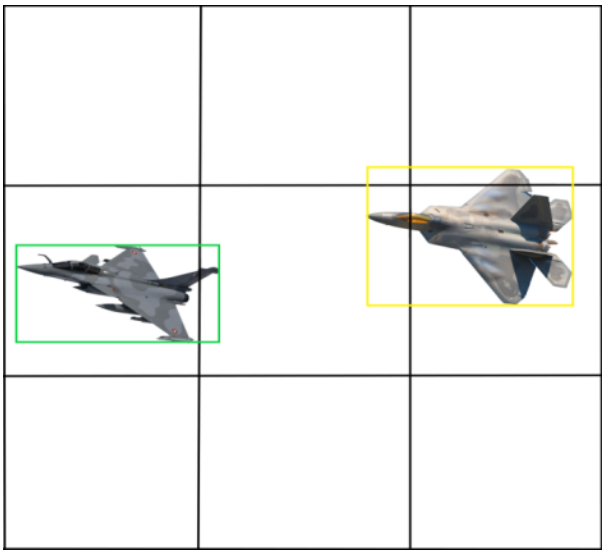
Pošto model za svaku ćeliju predviđa *B* graničnih okvira, pouzdanost tih okvira te vjerojatnost klasa *C,* predviđanja su kodirana kao $S \times S \times (B \cdot 5 + C)$ tenzor. Taj tenzor prikazan je kao zadnji konvolucijski sloj na slici 23.

Ako se za primjer i radi jednostavnosti uzme *S = 3* algoritam će podijeliti sliku na $3 \times 3 = 9$ ćelija (slika 25). Na svaku ćeliju bit će primijenjena klasifikacija i lokalizacija objekta i ako su objekti prisutni u ćeliji predvidjeti će se granični okviri i njihove odgovarajuće vjerojatnosti klasa. U navedenom primjeru dvije su klase lovaca – Rafale (zeleni granični okvir) i Raptor. U tom slučaju svaka ćelija će za izlaz imati sedmero dimenzionalni vektor *y*:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \end{bmatrix} \quad \textbf{(5)}$$

Gdje su:

- p_c - vrijednost koja pokazuje da li je objekt (ne klasa) prisutan na slici ili nije. U slučaju da je objekt prisutan vrijedi $p_c = 1$, ako objekt nije prisutan vrijedi $p_c = 0$.
- b_x, b_y, b_h, b_w – dimenzije graničnog okvira ako je objekt detektiran
- c_1, c_2 – vjerojatnosti klase, c_1 predstavlja klasu Rafale a c_2 klasu Raptor



Slika 25. Ulazna slika podijeljena na 9 ćelija

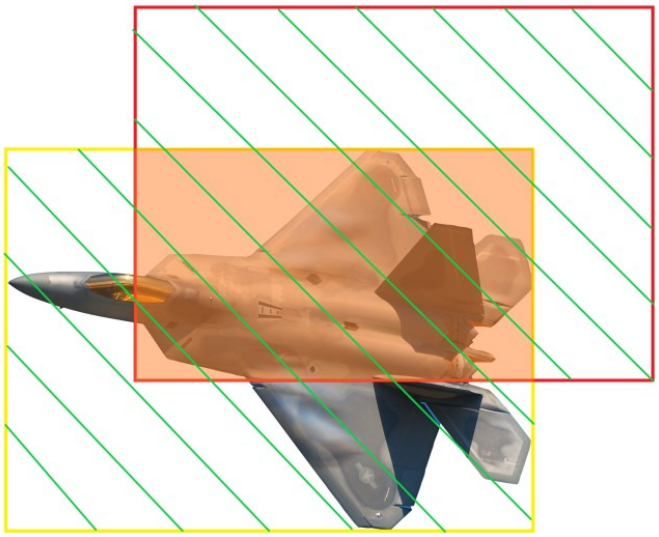
Prvi i treći redak te srednja ćelija ne sadrže centar objekta pa će vrijediti $p_c = 0$. U tom slučaju ostale vrijednosti se zanemaruju. Lijeva srednja ćelija sadrži centar objekta pa vrijedi $p_c = 1$. Izlazni vektor *y* bit će:

$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0.8 \\ 0.3 \end{bmatrix} \quad \textbf{(6)}$$

Kako se radi o vrlo sličnim klasama, odnosno obje klase lovački su zrakoplovi, vrlo je vjerojatno da će obje varijable vjerojatnosti klasa *c* poprimiti određenu vrijednosti. S obzirom na to da je $c_1 > c_2$ algoritam će detektiranom objektu pridodati klasu Rafale.

5.2.3 Intersection over Union

Intersection over Union evaluacijska je metrika koja se koristi za mjerenje točnosti detektora objekata na određenom skupu podataka (slika 26). Preciznost bilo kojeg algoritma koji za izlaz daje predviđene granične okvire može se procijeniti pomoću *IoU* (7).



Slika 26. *Intersection over union*

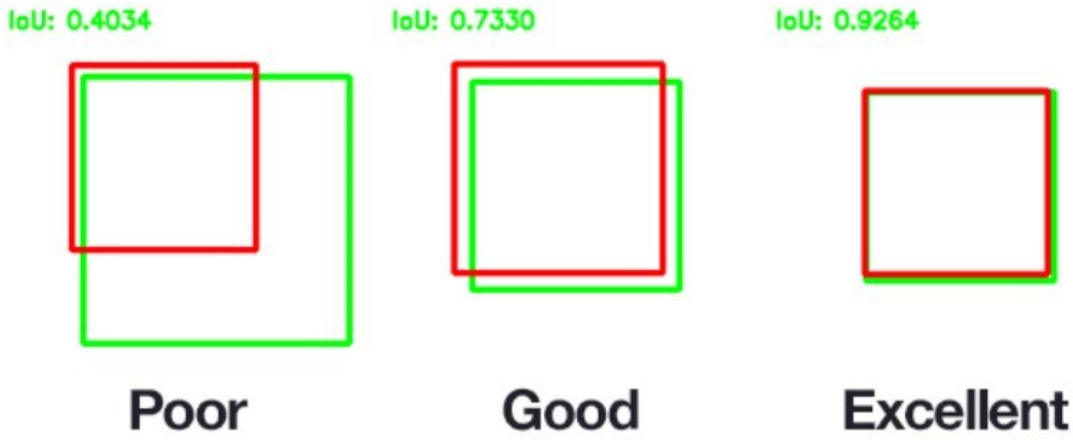
Računa se kao:

$$IoU = \frac{A_i}{A_u} \qquad \textbf{(7)}$$

Gdje je:

- A_i - površina presjeka (*area of intersection*) graničnih okvira (narančasti okvir)
- A_u – površina dvaju graničnih okvira (*area of union* - šafrirano zeleno)

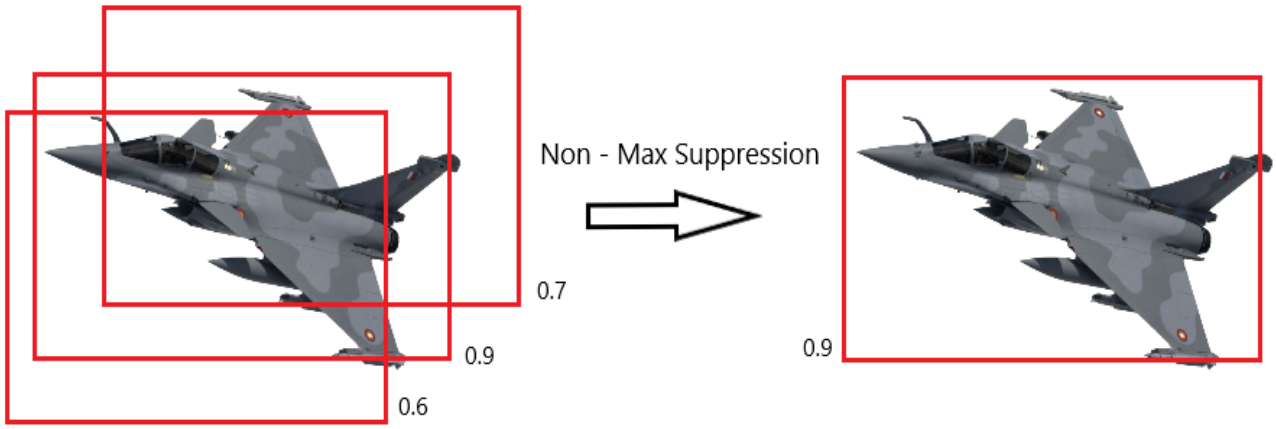
Dobivena vrijednost, koja se kreće u intervalu [0,1], odlučuje da li predviđeni granični okvir daje dobru ili lošu predikciju položaja objekta. U idealnom slučaju vrijediti će $IoU = 1$, odnosno predviđeni okvir u potpunosti će se poklapati s graničnim okvirom definiranim u temeljnoj istini. U pravilu se $IoU > 0.5$ smatra dobro predviđenim graničnim okvirom. Ta granična vrijednost odabire se proizvoljno i naziva se *Threshold*. Ako se uzme npr. $Threshold = 0.3$, sva predviđanja s vrijednošću $IoU < 0.3$ smatrat će se kao lošom predikcijom graničnog okvira i bit će odbačena. Povećavanjem vrijednosti *Threshold*-a dobivena predviđanja postaju sve točnija što zauzvrat rezultira manjim brojem detektiranih objekata. Primjeri dobrih i loših predviđenih graničnih okvira prikazani su na slici 27.



Slika 27. Primjer predviđanja graničnih okvira vrijednošću – *IoU* [23]

5.2.4 Non – Max Suppression

Detektiranje jednog objekta više puta jedan je od najčešćih problema algoritama na bazi graničnih okvira. U svrhu toga uvodi se ne-maksimalna supresija (*Non-Max Suppression)* - tehnika kojom se odabire granični okvir s najvećom vrijednošću *IoU* a ostali granični okviri se „potiskuju“ (slika 28).



Slika 28. *Non – Max Suppression*

Postupak koji algoritam provodi je sljedeći:

1. Odbacuje sve predviđene granične okvire koji imaju vrijednost manju od unaprijed definirane vrijednosti *Threshold*-a.
2. Za preostale granične okvire:
    - Odabire granični okvir koji ima najveću vrijednost, u ovome slučaju granični okvir sa *IoU = 0.9*
    - Odbacuje sve granične okvire koji imaju vrijednost *IoU* **veću** od vrijednosti *Threshold*-a s graničnim okvirom iz koraka prije

- Korak 2. se ponavlja dok svi granični okviri nisu uzeti kao predikcija (*output*) ili dok nisu odbačeni zbog $IoU > 0.5$

Može se primijetiti da cijeli proces filtriranja graničnih okvira ovisi o vrijednosti *Threshold*-a, stoga je odabir te vrijednosti ključan za dobru performansu algoritma.

5.2.5 Loss function

Detekcije algoritma u početku bit će daleko od savršenog. Tri su vrste pogreške koju algoritam može napraviti prilikom predikcije:

1. Pogrešno predviđanje klase objekta
2. Pogrešno predviđanje položaja i graničnog okvira objekta
3. Pogrešna pouzdanost

Svaku pogrešku potrebno je penalizirati kako bi se model pravilno istrenirao. U svakom koraku treninga izračunava se suma kvadrirane pogreške (*sum-squared error*) odnosno razlika između predviđene vrijednosti i istine te se računa gradijent funkcije. Pomak u smjeru gradijenta smanjuje gubitke a cilj je pronaći globalni minimum funkcije.

Funkcija gubitka (8) (*loss function*) ima oblik:

$$\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i\,j}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right] \\
&\quad + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i\,j}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2\right] \\
&\quad + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i\,j}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i\,j}^{noobj} (C_i - \hat{C}_i)^2 \\
&\quad + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \,\epsilon\, classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{8}$$

Prva dva člana jednadžbe računaju sumu grešaka predviđenih centara graničnih okvira i sumu grešaka predviđenih širina i visina graničnih okvira za svaki granični okvir ($j = 0 \ldots B$) svake ćelije mreže ($i = 0 \ldots S^2$), gdje su:

- x_i, y_i, w_i, h_i - koordinate centra, širine i visine predviđenog graničnog okvira u *i*-toj ćeliji

- $\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i$ - koordinate centra, širine i visine graničnog okvira označenog u temeljnoj istini

Kvadratni korijen je prisutan kod računanja sume grešaka širina i visina graničnih okvira tako da se pogreške u malim graničnim okvirima više penaliziraju od pogrešaka u velikim graničnim okvirima. Drugim riječima, pretpostavimo da imamo dva granična okvira veličine 1000×1000 i 100×100 piksela, tada će odstupanje od 50 piksela imati veći utjecaj na mali nego na veliki granični okvir.
Treći i četvrti član jednadžbe računaju ukupnu grešku pouzdanosti, gdje je:

- C_i – predviđena pouzdanost
- $\hat{C}_i$ – *IoU* između predviđenog graničnog okvira i graničnog okvira definiranog u temeljnoj istini

Peti član izračunava ukupnu klasifikacijsku pogrešku, gdje je:

- $p_i(c)$ – predikcija algoritma
- $\hat{p}_i(c)$ – jednak jedinici ako ćelija sadrži klasu *c*

Parametar λ_{coord} služi za povećavanje gubitka kod predviđanja koordinata graničnih okvira te uobičajeno iznosi $\lambda_{coord} = 5$. Uvođenjem tog parametra prioritizira se lokalizacija objekta nad klasifikacijom objekta unutar graničnog okvira. Većina graničnih okvira ne sadrži nikakve objekte što uzrokuje neravnotežu klasa, tj. model se trenira da detektira pozadinu češće nego objekte. Kako bi se to ispravilo, ovaj gubitak se smanjuje za faktor λ_{noobj} koji uobičajeno iznosi $\lambda_{noobj} = 0.5$. Kontrolna funkcija $\mathbb{1}_{i\,j}^{obj}$ određuje hoće li se na ćeliju primijeniti funkcija gubitka. U slučaju da se objekt nalazi u ćeliji i, *j*-ti granični okvir bit će zadužen za predikciju objekta i vrijedit će $\mathbb{1}_{i\,j}^{obj} = 1$ te će funkcija gubitka biti primijenjena. Kako je već spomenuto, YOLO predviđa *B* graničnih okvira po ćeliji a cilj je da tijekom treninga modela jedan granični okvir bude zadužen za detektiranje objekta. To će biti onaj granični okvir koji ima najveću vrijednost *IoU* sa temeljnom istinom. Ako ćelija ne sadrži objekt vrijediti će $\mathbb{1}_{i\,j}^{obj} = 0$. Kontrolna funkcija $\mathbb{1}_{i\,j}^{noobj}$ jednaka je nuli ako se u ćeliji nalazi objekt odnosno funkcija gubitka će biti primijenjena ako u ćeliji nema objekta jer tada vrijedi $\mathbb{1}_{i\,j}^{noobj} = 1$. $\mathbb{1}_{i}^{obj}$ također je kontrolna funkcija koja određuje za koje ćelije će se računati klasifikacijska pogreška i jednaka je jedinici ako se u ćeliji nalazi objekt.

5.3 YOLO v2

YOLO v2 unaprijeđena je verzija prvog modela koji radi puno grešaka prilikom lokalizacije objekta te ima relativno loš *recall*, odnosno sposobnost modela da pronađe relevantne objekte na

slici. U svrhu poboljšanja modela modificirana je arhitektura te su uvedene razne modifikacije kao što su *batch normalization*, *high resolution classifier*, *convolution with anchor boxes, dimension clusters*, *direct location prediction* i *fine-grained features* koje će biti objašnjenje u nastavku. Službeni YOLO v2 model treniran na *ImageNet* skupu slika koristi *Darknet-19* konvolucijsku mrežu kao ekstraktor značajki, no prilikom kreiranja vlastitog modela može se koristiti i bilo koja druga konvolucijska neuronska mreža.

5.3.1 Batch Normalization

Dodavanjem grupne normalizacije na sve konvolucijske slojeve smanjuje se *overfitting* te se mogu primijetiti značajna poboljšanja u konvergenciji što ubrzava trening modela. Istovremeno se eliminira potreba za drugim oblicima regularizacije. Nakon uvođenja grupne normalizacije primjećuje se poboljšanje srednje prosječne preciznosti (*mean average precision* - mAP) za više od 2%.

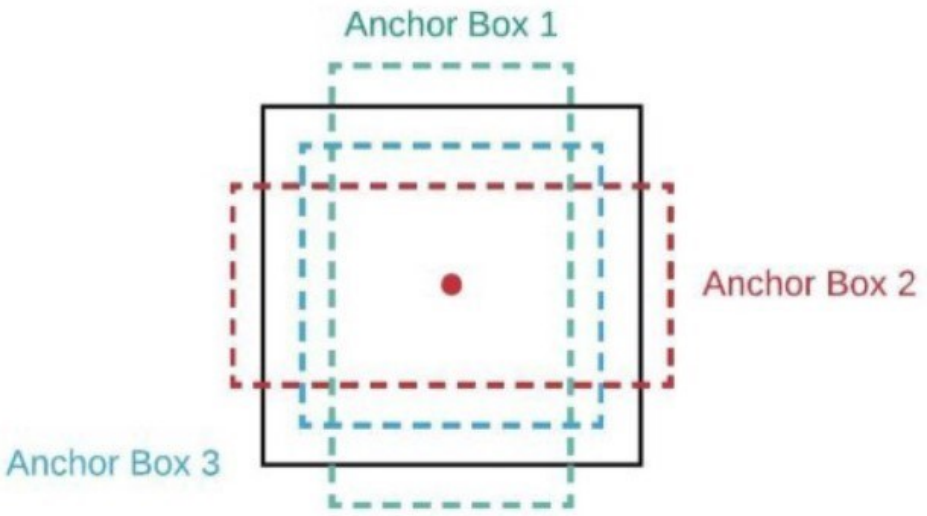
5.3.2 High Resolution Classifier

Kao što je spomenuto u prethodnom poglavlju, YOLO v1 koristi kao ulaznu veličinu tijekom treninga slike od 224×224 piksela a u trenutku detekcije uzima slike do veličine 448×448 piksela. To uzrokuje smanjenje *mAP* zbog toga što se model mora prilagođavati novoj rezoluciji. Autori su verziju YOLO v2 trenirali na većoj rezoluciji (448×448) tijekom 10 epoha na *ImageNet* skupu slika što je dalo mreži vremena za prilagođavanje filtera na višu rezoluciju. Treningom na slikama veličine 448×448 *mAP* se povećao za skoro 4%.

5.3.3 Convolution With Anchor Boxes

Fully connected i *pooling* slojevi uklonjeni su kako bi se zadržala karta značajki (*feature map*) veće rezolucije koja je prikladnija za korištenje sidrenih okvira (*anchor box*-ova). Sidreni okviri (koji se također nazivaju i zadani okviri) skup su unaprijed definiranih oblika okvira odabranih da odgovaraju graničnim okvirima definiranim u temeljnoj istini, jer većina objekata u skupu podataka za vježbanje ili općenito u svijetu (npr. auto, avion, osoba itd.) ima tipični omjer visine i širine. Dakle, prilikom predviđanja graničnih okvira, ti unaprijed definirani sidreni okviri (slika 29) prilagođavaju se kako bi odgovarali detektiranim objektima. Korištenje sidrenih okvira čini proces učenja iznimno lakšim, uz postizanje višestupanjske detekcije radi sidrenih okvira različitih veličina. Također, jedna ćelija može detektirati više od jednog objekta što nije bilo moguće u prijašnjoj verziji modela. Budući da se na većina slika veliki objekti nalaze u središtu poželjno je imati jednu

središnju ćeliju za predviđanje takvih objekata, a ne četiri ćelije koje su u blizini središta. Konvolucijski slojevi YOLO v2 modela smanjuju sliku za faktor 32, stoga je ulazna rezolucija slike smanjena s 448×448 piksela na 416×416 piksela što rezultira mapom značajki od 13×13, odnosno jednom središnjom ćelijom zbog neparnog ukupnog broja ćelija. Slično YOLO v1, predviđaju se koordinate svakog sidrenog okvira odnosno pomak tog okvira od graničnog okvira definiranog u temeljnoj istini (*offset*), ocjena pouzdanosti tog okvira koja se izračunava pomoću *IoU* između temeljne istine i predviđenog okvira, te skup klasno uvjetovanih vjerojatnosti. Korištenjem sidrenih kutija *recall* je povećan za *7%* što znači da je povećan postotak pozitivnih slučajeva (dobro detektiranih objekata), ali se primjećuje mali pad točnosti.



Slika 29. Sidreni okviri različitih veličina i omjera [24]

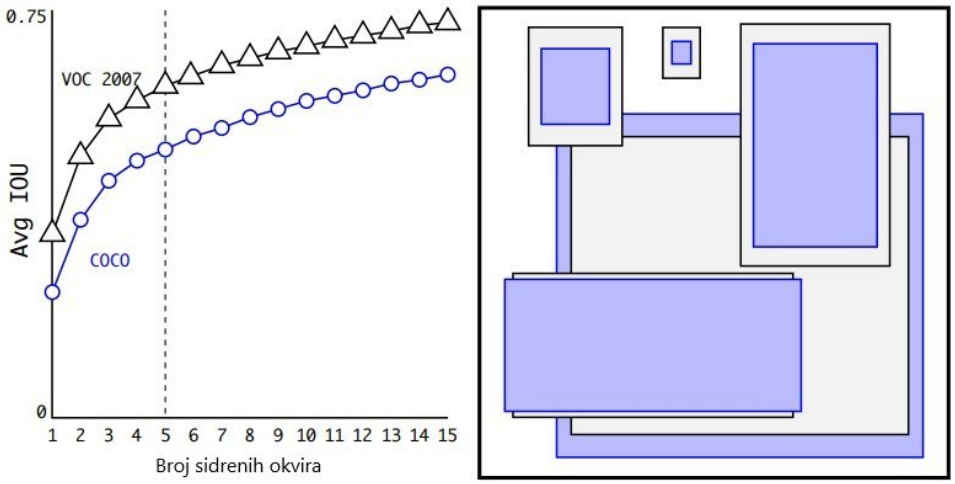
5.3.4 Dimension Clusters

Kako bi se dobili što bolji rezultati, umjesto ručnog odabira dimenzija sidrenih okvira oni se odabiru provođenjem algoritma za grupiranje *K*-srednjih vrijednosti (*K-means clustering*) na skupu podataka za treniranje detektora kako bi se pronašlo *K* prevladavajućih graničnih okvira. Recimo da se prilikom kreiranja temeljne istine označi 200 objekata, tada će biti 200 graničnih okvira različitih dimenzija. Ako se odabere $K = 4$ algoritam za grupiranje će grupirati 200 graničnih okvira u 4 prevladavajuće skupine, odnosno dobiti će se 4 sidrena okvira a svaki će biti takvih dimenzija da što bolje predstavlja 50 graničnih okvira definiranih u temeljnoj istini. Da bi se pronašlo najbliže težište svakom graničnom okviru definiranom u temeljnoj istini, algoritam ne koristi uobičajenu euklidsku udaljenost kao metriku udaljenosti jer ona generira

pogreške za velike granične okvire. Umjesto toga traže se okviri koji imaju dobar *IoU* s graničnim okvirima iz temeljne istine pa se koristi sljedeća metriku udaljenosti (9):

$$d(box, centroid) = 1 - IoU(box, centroid) \quad \textbf{(9)}$$

Autori istraživanja [25] zaključili su da je $K = 5$ najbolja vrijednost (slika 30) jer nudi dobar kompromis između složenosti modela i visokog *recall*-a. Sa slike se može primijetiti da je odabrano više uskih i visokih a manje širokih i niskih sidrenih okvira.



Slika 30. Grupiranje K-srednjih vrijednosti sidrenih okvira [25]

5.3.5 Direct Location Prediction

Predviđanje pomaka sidrenih okvira koji nisu ograničeni na određenu lokaciju dovodi do nestabilnosti tijekom treninga, pogotovo tijekom početnih iteracija, zbog toga što okvir može završiti u bilo kojem dijelu slike. Koristeći tu strategiju modelu je potrebno dugo vremena da nauči predviđati razumne pomake. YOLO v2 predviđa koordinate graničnog okvira u odnosu na lokaciju ćelije u mreži (slika 31), a u tu svrhu se koristi logistička funkcija koja za izlaz daje koordinate u rasponu od [0,1] čime se pomaci okvira ograničavaju. Postupak za određivanje odgovarajuće ćelije je sljedeći:

1. Definira se ćelija mreže kojoj pripada gornji lijevi kut sidrenog okvira
2. Predviđaju se pomaci (*offset*) graničnog okvira i ocjena pouzdanosti u odnosu na sidreni okvir

3. Kako bi se ograničili pomaci predviđenog graničnog okvira na njegovu odgovarajuću ćeliju, koordinate centra okvira vežu se za lokaciju centra korištenjem sigmoidne funkcije σ

Parametri finalnog graničnog okvira (slika 10) su:

$$b_x = \sigma(t_x) + c_x \quad \textbf{(10)}$$

$$b_y = \sigma(t_y) + c_y \quad \textbf{(11)}$$

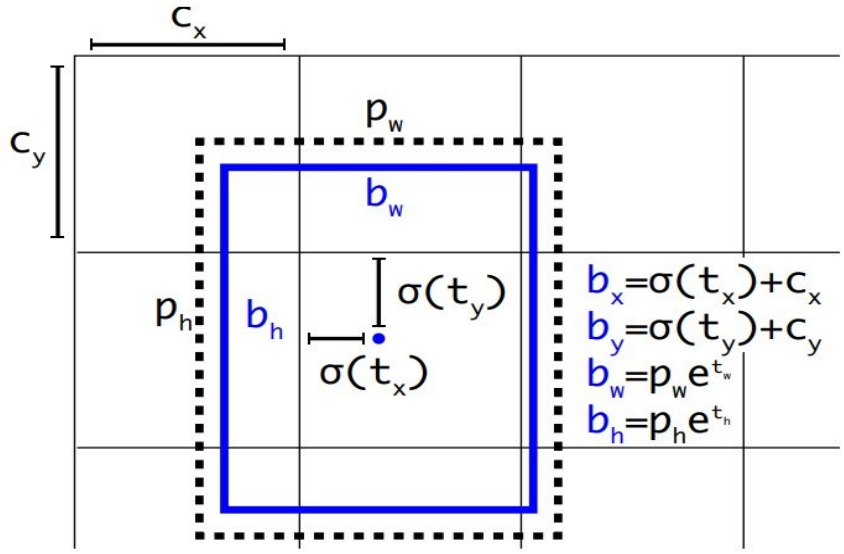
$$b_w = p_w e^{t_w} \quad \textbf{(12)}$$

$$b_h = p_h e^{t_h} \quad \textbf{(13)}$$

$$\Pr(\text{object}) * IoU(b, \text{object}) = \sigma(t_0) \quad \textbf{(14)}$$

Gdje su:

- c_x, c_y – pomak od gornje lijeve ćelije
- b_x, b_y – koordinate centra finalnog graničnog okvira
- b_w, b_h - širina i visina finalnog graničnog okvira
- p_w, p_h - širina i visina sidrenog okvira
- t_x, t_y, t_w, t_h, t_0 – koordinate centra, širine i visine predviđenog graničnog okvira i ocjena pouzdanosti



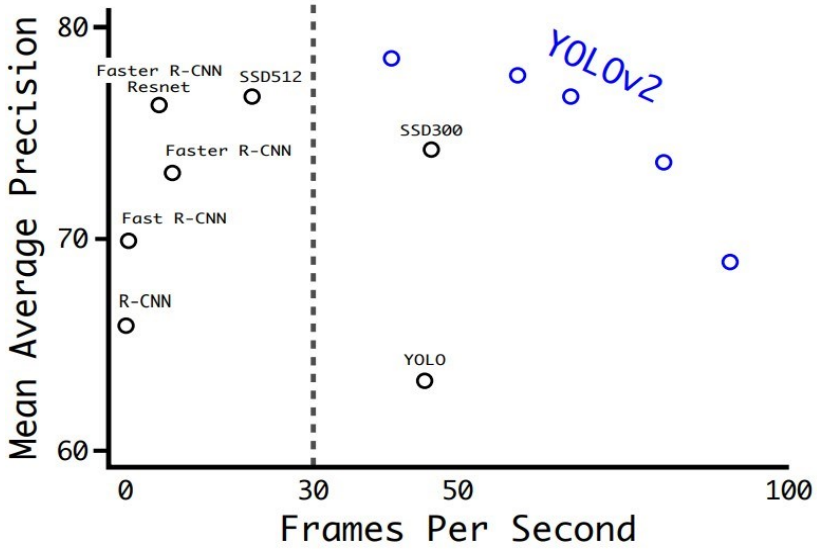
Slika 31. Izravna predikcija koordinata graničnog okvira [25]

5.3.6 Fine-Grained Features

Kao što je ranije spomenuto, modificirani YOLO daje kartu značajki rezolucije 13×13, koja je dovoljna za detekciju velikih objekata ali nije prikladna za detekciju malih objekata zbog gubitka semantičkih značajki pri malim rezolucijama. Korištenje karte značajki veće rezolucije pomaže mreži u otkrivanju objekata različitih veličina, pa YOLO v2 spaja sloj značajki rezolucije 26×26 sa slojem značajki niske rezolucije, čineći kartu značajki $26 \times 26 \times 512$ kartom značajki $13 \times 13 \times 2048$. Ovaj pristup poboljšava model za skromnih 1%.

5.3.7 Multi-Scale Training

Budući da se mreža sastoji samo od konvolucijskih i objedinjujućih (*pooling*) slojeva, a ne od potpuno povezanih slojeva, može se trenirati na različitim ulaznim veličinama. Svakih 10 epoha mreža nasumično odabire novu veličinu slike a budući da se model smanjuje za faktor 32 odabrane veličine trebaju biti višekratnici toga broja: $320,352, \ldots 608$, gdje je 320×320 najmanja a 608×608 najveća rezolucija. Ovakav režim pomaže mreži naučiti detektirati objekte na različitim rezolucijama. Na manjim rezolucijama model je brži pa tako YOLO v2 nudi dobar kompromis između brzine i točnosti (slika 32).



Slika 32. Brzina i preciznost različitih detektora na *VOC 2007* skupu slika [25]

5.4 YOLO v3

Treća verzija ovog detektora, koja je objavljena 2018. godine, koristi novu konvolucijsku mrežu *Darknet-53* kao ekstraktor značajki. Uvedena su razna poboljšanja koja model čine većim i

preciznijim te koja zadržavaju njegovu veliku brzinu. Za razliku od prijašnjih modela koji vrše detekciju na jednoj razini, YOLO v3 detektira objekte na tri razine odnosno za izlaz se dobivaju tri tenzora.

5.4.1 Struktura

Prijašnja verzija YOLO modela koristila je *Darknet-19*, 19-slojnu konvolucijsku mrežu na koju je dodano 11 slojeva za detekciju. Zbog malog broja slojeva model ima problema prilikom detekcije malih objekata. Kako bi se riješio taj problem autori uvode uzastopne 3×3 i 1×1 konvolucijske slojeve i veze za preskakanje slojeva što ćini mrežu puno dubljom i pogodnijom za detekciju malih objekata. Tako je nastala nova konvolucijska mreža *Darknet-53* koja se sastoji od ukupno 106 slojeva od kojih su 53 konvolucijska, čineći ju moćnijom od *Darknet-19* i učinkovitijom od konkurentnih mreža *ResNet-101* i *ResNet-152.* Na kraju *Darknet-19* mreže dodaje se par konvolucijskih slojeva od kojih zadnji predviđa 4 koordinate predviđenog graničnog okvira, nalazi li se objekt u okviru ili ne (*objectness score*) te vjerojatnost klase objekta. Za izlaz iz prvog bloka za detekciju dobiva se 3-D tenzor oblika:

$$\big(S, S, B \times (4 + 1 + C)\big),$$

gdje 4 odgovara broju predviđenih koordinata graničnog okvira a 1 rezultatu objektivnosti.

Nakon toga se karta značajki uzeta iz dva sloja prije duplo uvećava i spaja s kartom značajki iste veličine iz prijašnjih slojeva u mreži. To omogućuje dobivanje smislenijih semantičkih informacija iz uvećane karte značajki i finijih informacija iz karte značajki uzete iz prijašnjih slojeva [26]. Tako spojena karta značajki prolazi kroz par konvolucijskih slojeva bloka za detekciju te se za izlaz dobiva tenzor oblika:

$$\big(2S, 2S, B \times (4 + 1 + C)\big).$$

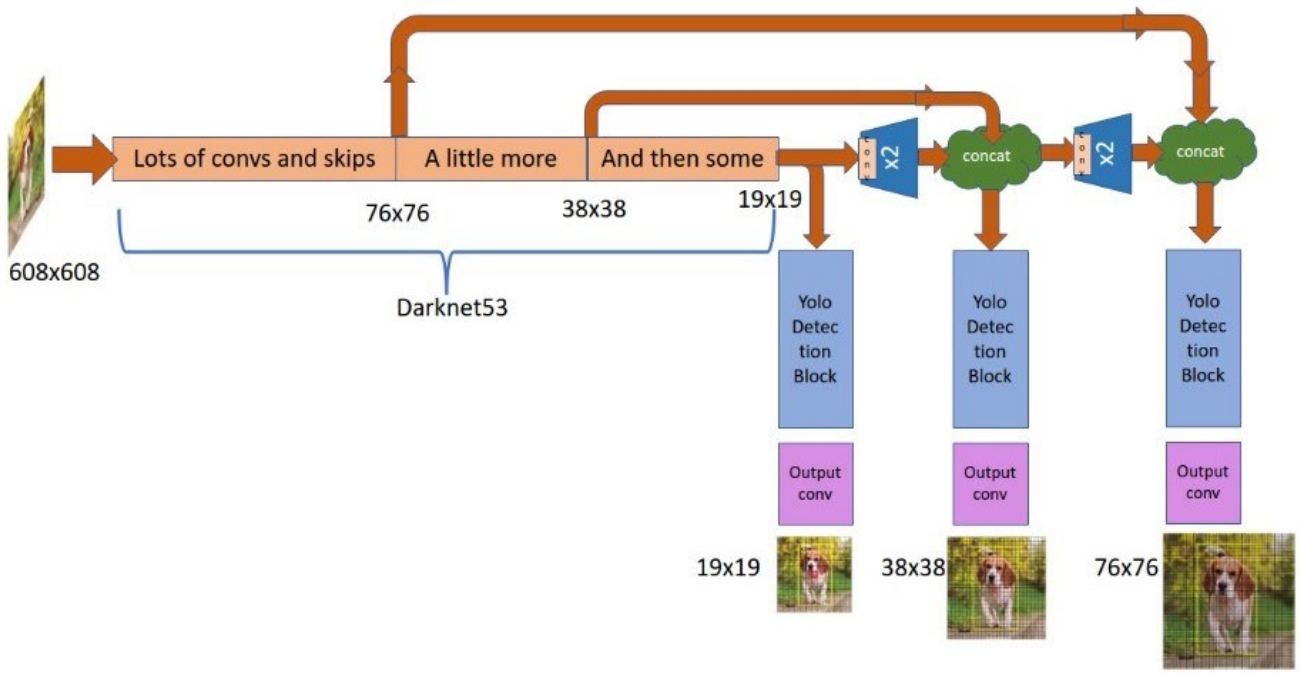
Postupak se ponavlja te se za izlaz iz trećeg bloka za detekciju dobiva tenzor oblika:

$$\big(4S, 4S, B \times (4 + 1 + C)\big).$$

Prvi blok za detekciju 19×19 ima širi kontekst i lošiju rezoluciju u usporedbi s drugim blokovima detekcije, pa je specijaliziran za otkrivanje velikih objekata , dok je blok 76×76 specijaliziran za otkrivanje malih objekata. Svaki od tri bloka za detekciju (detektorskih glava) ima zaseban skup sidrenih okvira. Kako ovaj model predviđa $B = 3$ granična okvira po bloku za detekciju, ukupno će biti devet sidrenih okvira. Usporedba predviđenih veličina svakog graničnog okvira za različite YOLO verzije prikazana je na slici 33.

5.4.2 Predikcija klasa

YOLOv2 je koristio *softmax*, što je matematička funkcija koja pretvara vektor brojeva u vektor vjerojatnosti. Korištenjem *softmax*-a svaki granični okvir može pripadati samo jednoj klasi, što ponekad nije slučaj, osobito sa skupovima podataka kao što je Microsoft-ov *Open Images Dataset* (*OID*) koji sadrži desetke preklapajućih oznaka kao što su „osoba“ i „muškarac“. YOLO v3 koristi nezavisne logističke klasifikatore za predviđanja klasa kako bi se mogao trenirati na skupovima slika kao što su *OID*.



Slika 33. Struktura YOLO v3 modela [27]

5.5 YOLO v4

Prilikom dizajniranja četvrte verzije detektora glavni cilj je bila velika brzina, kako bi se mogao koristiti u proizvodnim sustavima, te optimizacija paralelnog izračunavanja kako bi svatko mogao istrenirati i koristiti detektor u stvarnom vremenu na visokoj rezoluciji koristeći konvencionalnu grafičku karticu (*GPU*).

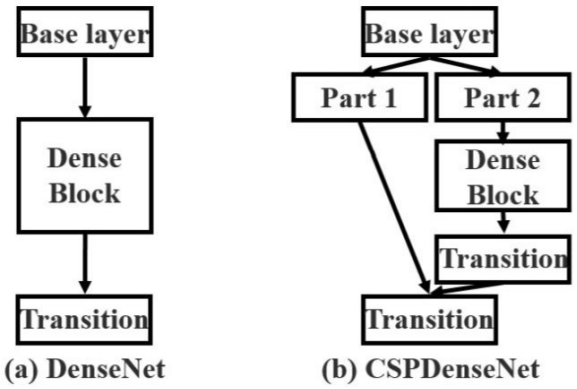
5.5.1 Struktura

Arhitektura se sastoji od različitih dijelova. Na početku se nalazi ulaz (*input*) odnosno skup slika za učenje koje će biti dostavljene mreži koje se u serijama paralelno obrađuju od strane *GPU*-a. Nakon

toga slijedi glavna mreža (*backbone)* odnosno konvolucijska neuronska mreža koja, kao i u prijašnjim modelima, služi kao ekstraktor značajki. Autori su nakon puno testiranja i eksperimentalnih rezultata izabrali *CSPDarknet53* konvolucijsku neuronsku mrežu koja je bazirana na dizajnu *DesNet*-a (slika 34a). Sastoji se od dva dijela:

- Osnovnog konvolucijskog bloka
- Prijelaznog djelomičnog bloka (*Cross Stage Partial Block – CSP Block*)

Mapa značajki se dijeli na dva dijela pomoću *CSP* bloka koji se nalazi paralelno uz konvolucijski blok. Jedna polovica podijeljene mape značajki prolazi kroz gusti blok (*dense block*) a druga polovica služi kao *input* u sljedeći konvolucijski blok bez ikakve dodatne obrade (slika 34b). *CSP* blok rezultira manjim brojem mrežnih parametara te stimulacijom mreže da ponovno koristi očuvane značajke koje su proslijeđene iz prijašnjih slojeva. Gusti blok se sastoji od više konvolucijskih slojeva, a između tih slojeva nalazi se sloj koji se sastoji od *batch* normalizacije, *ReLu* i konvolucije. Taj sloj, osim svog originalnog ulaza, uzima i izlaze svih prethodnih slojeva te su tako svi slojevi povezani jedni s drugima i svakom se sloju povećava mapa značajki.

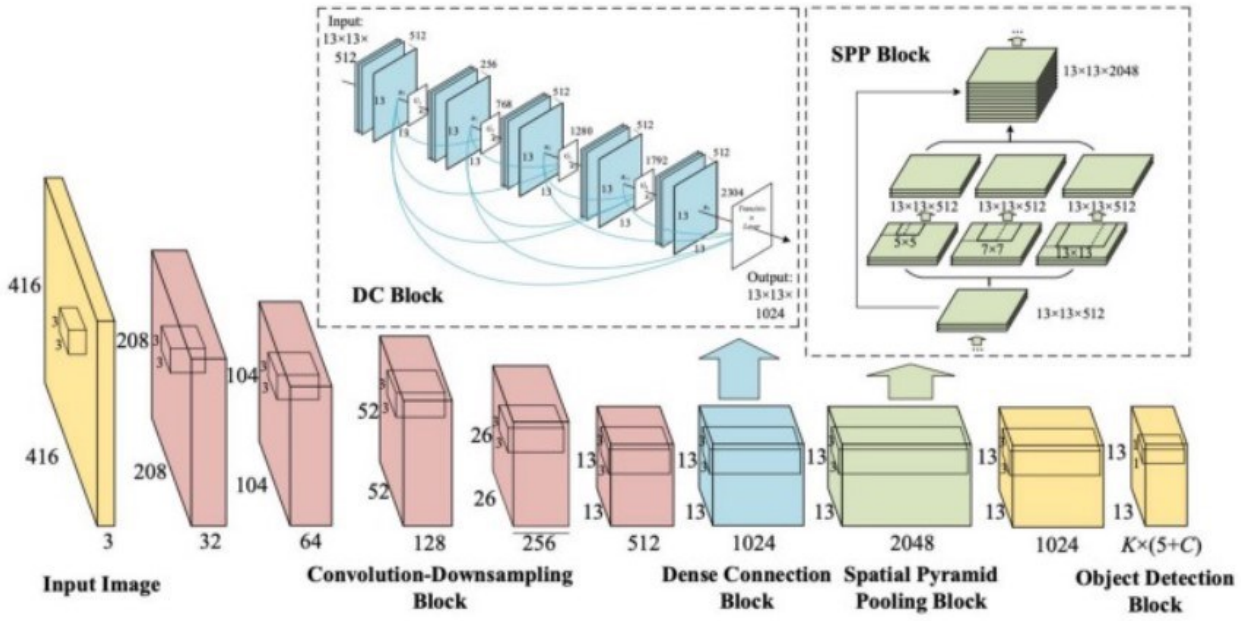


Slika 34. Struktura YOLO v4 modela *a)* Gusti blok u *DenseNet* konvolucijskoj mreži *b) CSP* sa gustim blokom u *CSPDenseNet*-u [28]

Nakon *backbone*-a slijedi vrat (*neck*) mreže koji prikuplja mape značajki iz različitih slojeva neuronske mreže, miješa ih i priprema za sljedeći korak. Drugim riječima, vrat služi kao agregator značajki i sastoji se od *Spatial Pyramid Pooling (SPP)* bloka i *Path Aggregation* mreže *(PANet).*

Korištenje *PANet*-a poklapa se s ciljem treniranja detektora na jednom *GPU*. *PANet* je mreža koja izdvaja važne značajke iz glavne mreže pomoću *SPP* bloka. *SPP* blok, koji je povezan sa završnim gusto povezanim konvolucijskim slojevima, izdvaja najvažnije značajke i povećava prijemno polje bez ikakvog utjecaja na brzinu mreže. Običan *SPP* blok uzima mape značajki različitih dimenzija nakon što se one generiraju iz ranijih konvolucijskih slojeva koristeći *max pool* te ih slaže jednu na drugu. Kako se mape značajka inače postupno smanjuju dobiva se oblik piramide. YOLO v4

modificira *SPP* kako bi zadržao istu izlaznu dimenziju mapa značajki te se one spajaju u jedan blok. Na slici 35. prikazano je kako su gusti blok i *SPP* blok integrirani u YOLO v4 model.



Slika 35. Integracija gustog bloka i *SPP* bloka u YOLO v4 [28]

5.5.2 Bag of freebies

U YOLO v4 su uvedene razne metode za poboljšanje performanse mreže bez povećavanja vremena inferencije koje su nazvane *„Bag of freebies“*. Neke od tih metoda su tehnike augmentacije, kao što su *Mosaic Data Augmentation (MDA)*, *Self-Adversial Training (SAT)*, *DropBlock* i *CutMix* pomoću kojih se umjetno povećava broj slika za treniranje detektora. *MDA* (slika 37a) povezuje četiri slike za trening detektora što pomaže modelu pri učenju detektiranja malih objekata. *SAT* prisiljava mrežu da uči nove značajke tako što prekriva dio slike o kojoj mreža najviše ovisi. U svrhu sprječavanja *over-fitting*-a uvode se regularizacijske tehnike *DropBlock* i izglađivanje oznaka klasa (*Class label smoothing)*.

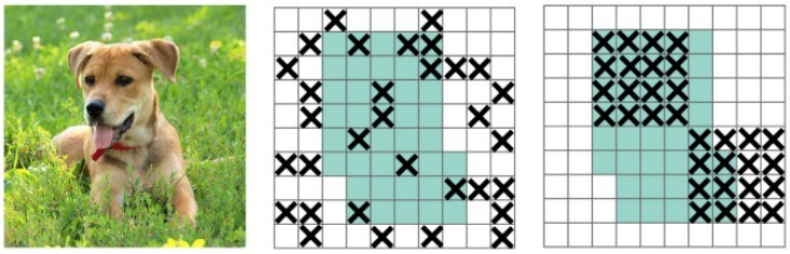
U potpuno spojenim slojevima se mogu ispustiti individualni pikseli, ali sadržaj slike će i dalje ostati prepoznatljiv. Zbog toga se uvodi *DropBlock* – ispuštanje blokova piksela (slika 36). Za razliku od običnog ispuštanja individualnih piksela *DropBlock* radi na svim konvolucijskim. *Class label smoothing* prilagođava gornju granicu predviđanja na nižu vrijednost izrazom (15):

$$y_{lb} = (1 - \alpha_i) * y_{hot} + \alpha / C \qquad \textbf{(15)}$$

Gdje je:

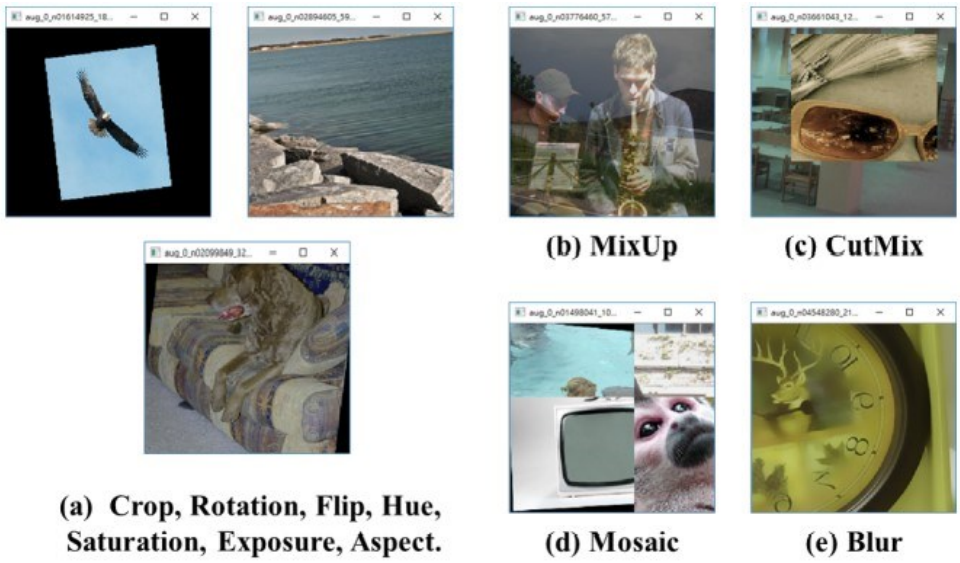
- C – broj klasa

- α_i – hiperparametar za izglađivanje



Slika 36. Ispuštanje individualnih piksela i blokova piksela tehnikom *DropBlock-a* [28]

Tijekom *CutMix* augmentacije (slika 37c) dio slike se izrezuje i lijepo preko druge a na to mjesto lijepi se druga izrezana slika. Budući da je izrezano područje zamijenjeno drugom slikom, količina informacija na slici i učinkovitost treninga neće biti značajno pogođeni. Područje izrezane slike prisiljava model da nauči klasifikaciju objekata s različitim skupovima značajki čime se izbjegava pretjerano samopouzdanje modela. Budući da je to područje zamijenjeno drugom slikom, količina informacija na slici i učinkovitost treninga također neće biti značajno pogođeni. Različite metode augmentacije prikazane su na slici 37.

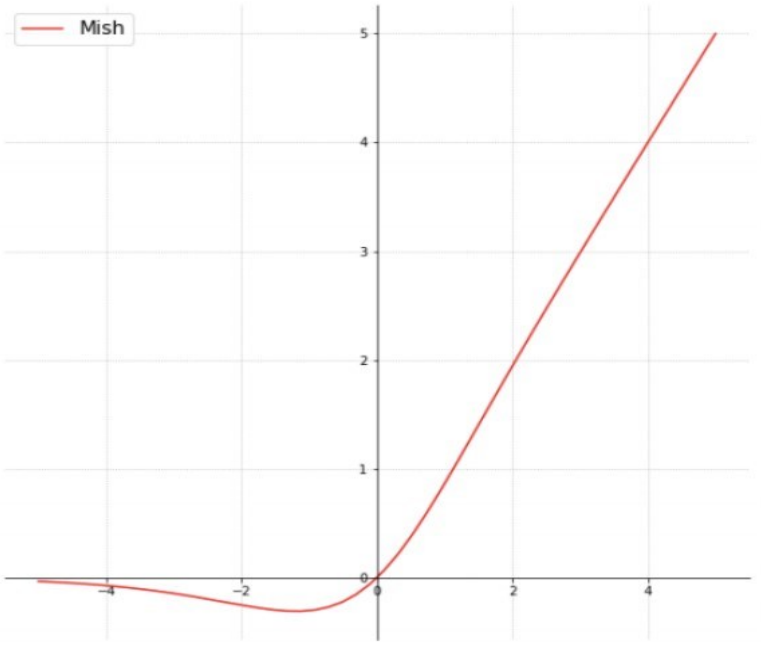


Slika 37. Različite metode augmentacija slika [29]

5.5.3 Bag of specials

Uvedene strategije koje minimalno povećavaju inferenciju, odnosno vrijeme zaključivanja detektora, ali značajno poboljšavaju njegovu performansu nazivaju se *„Bag of specials“*.

Klasična *ReLU* aktivacijska funkcija zamijenjena je *Mish* aktivacijskom funkcijom (slika 38) koja daje bolje empirijske rezultate odnosno veću preciznost glavne mreže i detektora. *Mish* aktivacijska funkcija omogućuje guranje signala lijevo i desno, nešto što nije moguće sa *ReLU*.



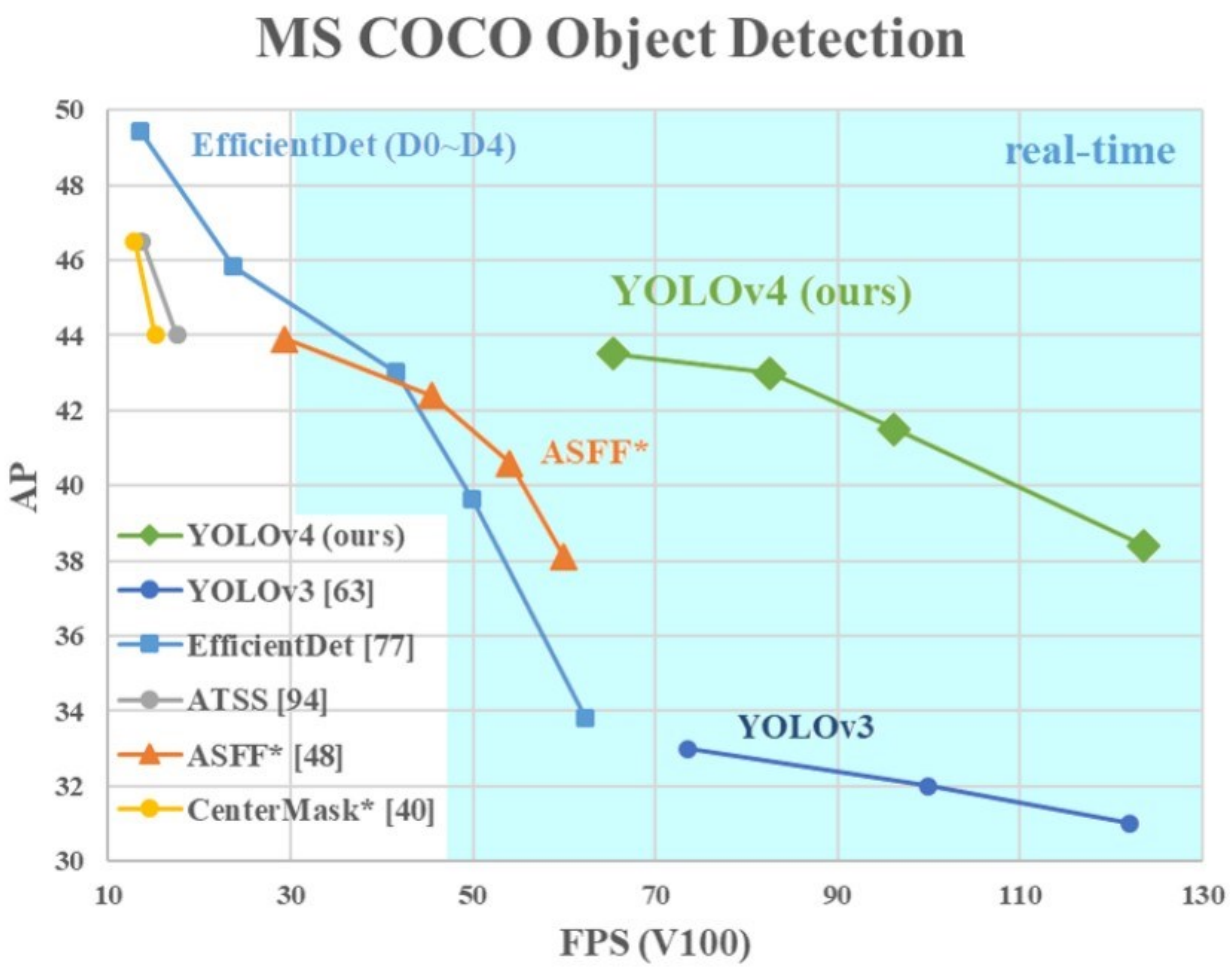
Slika 38. *Mish* aktivacijska funkcija [30]

Distance IoU prilikom ne-maksimalne supresije uzima u obzir vrijednost *IoU* i udaljenost između središnjih točaka dvaju graničnih okvira.

Neke od ostalih strategija su integracija već spomenutog modificiranog *SPP* bloka i modificirane *PANet* mreže u arhitekturu modela.

5.5.4 Rezultati

Usporedba rezultata YOLO v4 modela i ostalih detektora treniranih na *COCO* skupu slika prikazana je na slici 39. Primjećuje se poboljšanje srednje prosječne preciznosti (*mAP*) od *10%* te *FPS*-a od *12%* u odnosu na prethodnu verziju modela [31].



Slika 39. Usporedba YOLO v4 i ostalih detektora [29]

Detekcija u stvarnom vremenu (web kamera, ulične kamere, itd.) predstavljena je plavom bojom na grafikonu, a detekcija slike bijelom bojom. Poboljšanje srednje preciznosti i *FPS*-a direktna je posljedica implementacije „*Bag of freebies*“ i „*Bag of specials*“ u model.

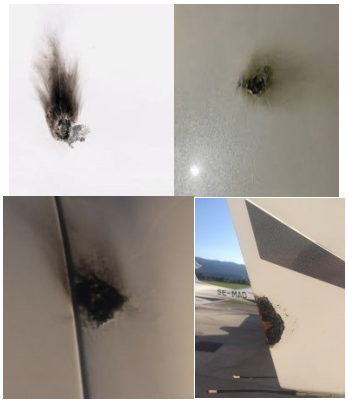
6. TRENIRANJE KONVOLUCIJSKE NEURONSKE MREŽE

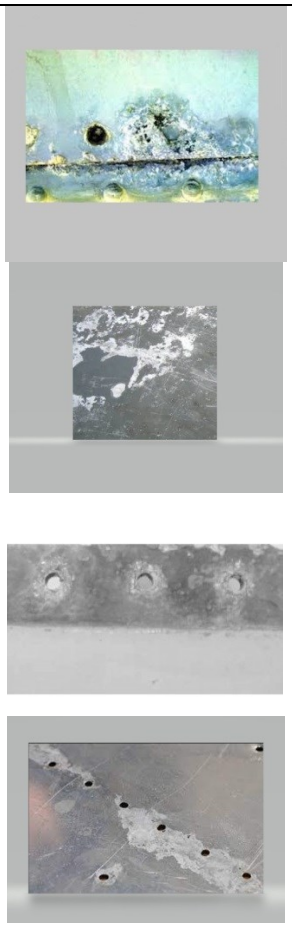
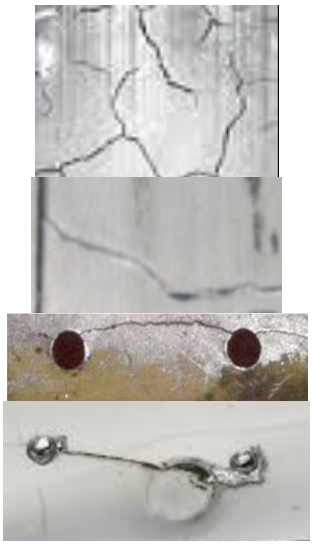
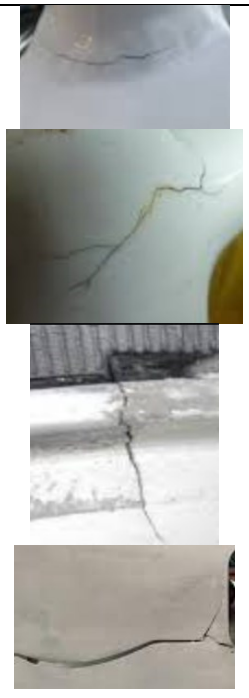
Treniranje konvolucijske neuronske mreže provodi se u programskom paketu *MATLAB* pomoću *Deep Learning Toolbox*-a. Mreža se trenira za klasifikaciju anomalija koje su podijeljene u četiri klase: oštećenje od udara groma (*Burnmark*), korozija (*Corrosion*), pukotina (*Crack*) i nabor (*Crease*). Treninzi svih mreža prikazanih u nastavku provedeni su na *nVidia GeForce RTX 3090* grafičkoj kartici sa 24 GB virtualne memorije.

6.1 Eksperimentalni podaci

Primjeri prikupljenih anomalija trupa zrakoplova korištene za treniranje konvolucijske neuronske mreže i detektora anomalija prikazani su u tablici 2. Kompletna baza podataka podijeljena je u četiri već navedene klase, a svaka kategorija sadrži otprilike 60 slika anomalija. Isti broj slika po klasi je bitan za pravilan trening mreže kako značajke jedne klase ne bi dominirale nad značajkama ostalih klasa. Također, treba napomenuti da su neke slike podijeljenje u više manjih slika kako bi se povećao skup slika za trening mreže i detektora.

Tablica 2. Izvod dijela prikupljenih slika anomalija trupa zrakoplova korištenih za trening *CNN* mreže i YOLO detektora

Klasa anomalije	Slike za trening konvolucijske neuronske mreže za klasifikaciju	Slike za trening detektora anomalija	Opis
Oštećenje od udara groma			Kod mentalnih konstrukcija zrakoplova, oštećenja od udara groma obično se manifestiraju kao tragovi opeklina ili male kružne rupe. Izgorjela ili bezbojna oplata također sugerira oštećenje od udara groma.

Korozija			Korozija je hrđa koja se pojavljuje na metalnim površinama i komponentama zrakoplova, a nastaje kada nezaštićeni metal dođe u dodir s kisikom u atmosferi. Može se prepoznati kao sivkasto-bijele naslage na aluminiju.
Pukotina			Pukotina je djelomično odvajanje materijala obično uzrokovano vibracijama, preopterećenjem, unutarnjim naprezanjima, neispravnim sklopom ili zamorom. Dubina pukotine u početnim stadijima iznosi nekoliko milimetara nakon čega se širi prema području visokog opterećenja.

Nabor			Nabori najčešće nastaju kao posljedica tvrdog slijetanja, a mogu se opisati kao izbočine ili valovita deformacija oplate trupa zrakoplova.

Prilikom vizualne inspekcije trupa zrakoplova problem nije otkrivanje anomalija koje imaju veliku površinu već onih anomalija manje površine, kao što je npr. udar groma čije se oštećenje, koje se u većini slučajeva manifestira na zakovicama, lako predvidi. Stoga vizualna inspekcija mora biti detaljna, što zahtijeva značajni utrošak ljudskog rada i vremena. U svrhu olakšanja takve inspekcije sve više se uvode nove tehnologije. Jedna od njih je vizualna inspekcija dronom, koja nije bila dio eksperimentalnog istraživanja u ovom radu.

6.2 Kreiranje mreže, trening i rezultati

Započinje se učitavanjem arhitekture *Darknet53* konvolucijske neuronske mreže već istrenirane (*pretrained*) na *imagenet* skupu slika:

```
net = darknet53;
```

U ovom slučaju provodi se prijenosno učenje (*transfer learning)*, odnosno prilagođavaju se već naučeni parametri (filteri) na *imagenet* skupu slika novom skupu slika što rezultira većom preciznošću i manjim vremenom treninga mreže. *Transfer learning* je dobar način za treniranje klasifikacijske mreže u slučaju malog skupa podataka odnosno slika. Važno je napomenuti da je za treniranje klasifikacijske mreže bez prijenosnog učenja 1500 slika minimalni preporučeni broj slika po kategoriji dok se prilikom treninga ove mreže, kao što je već spomenuto, koristilo otprilike 60 slika po kategoriji. Razlog tomu je manjak dostpunih slika realnih oštećenja zrakoplova. U svrhu ovog rada, radi manjka dostupnih slika, kategorije su nadopunjene slikama koje nalikuju realnim oštećenjima zrakoplova.

Osnovni podaci o *pretrained* konvolucijskim mrežama koje su korištene u ovom radu u svrhu kreiranja konvolucijske neuronske mreže za klasifikaciju anomalije prikazani su u tablici 3 [32].

Tablica 3. Osnovni podaci o *pretrained* konvolucijskim mrežama korištenim u svrhu kreiranja mreže za klasifikaciju anomalija

Mreža	Dubina mreže (broj konvolucijskih slojeva)	Veličina	Skup slika za trening	Broj klasa	Broj parametara (filtera) u milijunima	Veličina ulazne slike
darknet53	53	155 MB	*imagenet*	1000	41.6	[256 256]
resnet50	50	96 MB	*imagenet*	1000	25.6	[224 224]
googlenet	22	27 MB	*imagenet*	1000	7.0	[224 224]

Sljedeći korak je učitavanje slika anomalija koje se nalazi u folderu *Anomalije_trupa* unutar kojeg se nalaze četiri foldera čija imena određuju imena klasa, a u svakom od tih foldera se nalaze slike odgovarajućih anomalija:

```
Dataset = imageDatastore('Anomalije_trupa',...
          'IncludeSubfolders',true,'LabelSource','foldernames');
```

Nakon toga se učitani skup slika dijeli na skup slika za trening mreže (70%) i skup slika za validaciju tijekom treninga (30%):

```
[Training_dataset, Validation_dataset] = splitEachLabel(Dataset,0.7,'randomized');
```

Slijedi definiranje parametara za augmentaciju slika:

```
imageAugmenter = imageDataAugmenter( ...
                 'RandRotation',[-180 180], ...
                 'RandXTranslation',[-25 25], ...
                 'RandYTranslation',[-25 25],...
                 'RandXReflection',1,....
                 'RandYReflection',1,...
                 'RandScale',[0.8 1.2]);
```

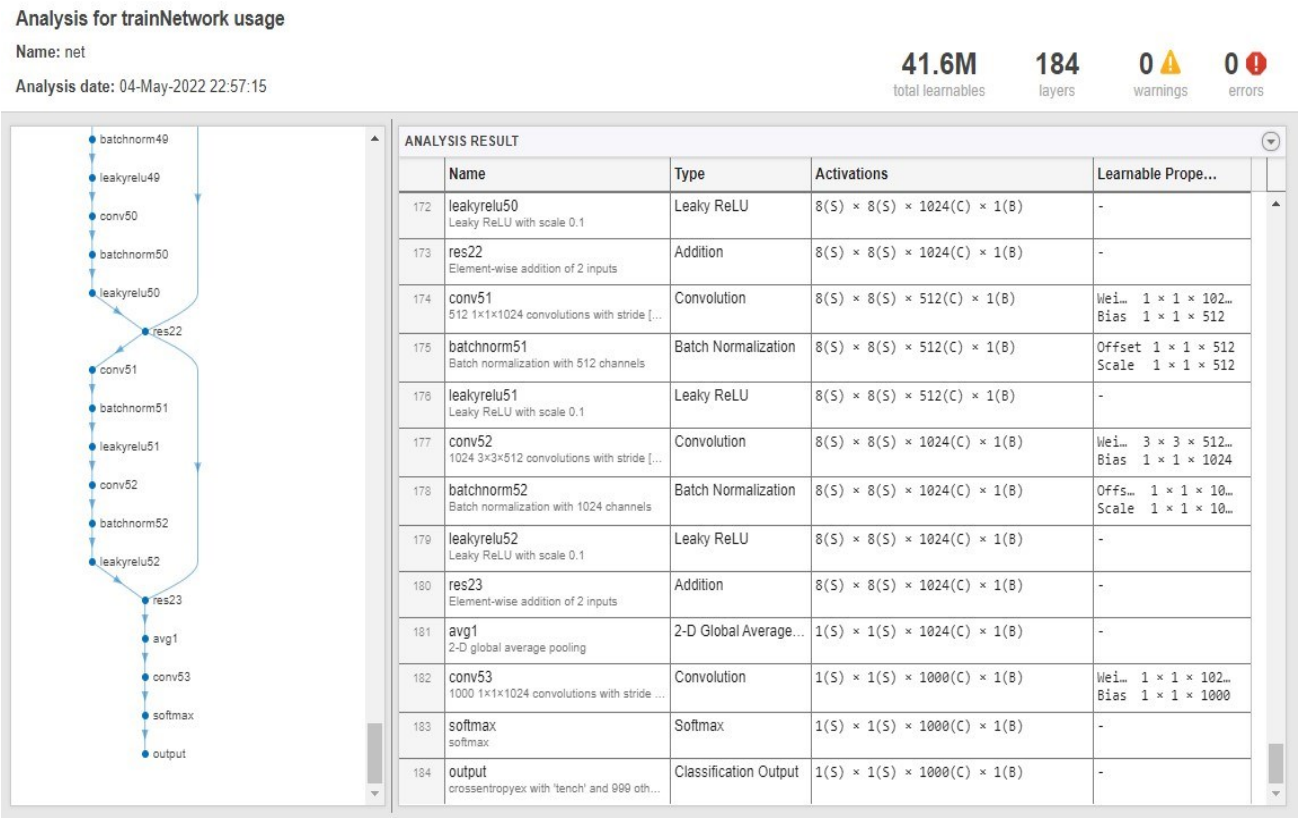
Slike se nasumično rotiraju za $\pm 180°$, translatiraju se nasumično po x i y osi za ± 25 piksela, reflektiraju se nasumično oko x i y osi te se nasumično skaliraju u rasponu od $[0.8\ 1.2]$.
Ulazna veličina slika ([256 256 3]), definirana strukturom samog *pretrained Darknet*-a, sprema se u varijablu *Input_Layer_size*:

```
Input_Layer_size = net.Layers(1).InputSize(1:2);
```

Kako su slike za trening i validaciju različitih veličina, potrebno im je promijeniti veličinu na ulaznu veličinu *Darknet*-a (*Input_Layer_size*) pomoću funkcije *augmentedImageDatastore*:

```
Resized_Training_image =
augmentedImageDatastore(Input_Layer_size,Training_dataset,...
                        'DataAugmentation',imageAugmenter,...
                        'ColorPreprocessing','gray2rgb');
Resized_Validation_image = augmentedImageDatastore(Input_Layer_size,
                         Validation_dataset,'ColorPreprocessing','gray2rgb');
```

S obzirom da treniranje mreže ne podržava crno-bijele slike, koristi se '*ColorPreprocessing*', kako bi se, ako ih ima, pretvorile u *RGB* slike. Učitana *Darknet53* mreža, koja je spremljena u *net* varijablu, trenirana je na 1000 klasa, stoga ju je potrebno prilagoditi za trening na četiri klase. U svrhu toga, korisno je analizirati učitanu mrežu pomoću naredbe *analyzeNetwork(net)* kako bi se vidjelo koje slojeve je potrebno prilagoditi (slika 40).



Slika 40. Prikaz zadnjih slojeva *pretrained Darknet53* mreže pomoću *analyzeNetwork* naredbe

Može se vidjeti da je 182. sloj *feature learner* sloj, odnosno konvolucijski sloj veličine $1 \times 1 \times 1000$, gdje 1000 odgovara broju klasa na kojima se trenirala mreža. Potrebno ga je spremiti u *Feature_Learner* varijablu:

```
Feature_Learner = net.Layers(182);
```

Izlazni (*output*) sloj, odnosno klasifikacijski sloj je 184. (zadnji) sloj, potrebno je spremiti u *Output_Classifier* varijablu:

```
Output_Classifier = net.Layers(184);
```

Već je poznato da se mreža trenira za klasifikaciju 4 klase, a u slučaju da je broj klasa nepoznat, može se odrediti pomoću:

```
Number_of_Classes = numel(categories(Training_dataset.Labels));
```

Sljedeći korak je definiranje novog *feature learner* sloja, koji će biti potpuno spojeni sloj, i klasifikacijskog sloja koji će imati 4 klase:

```
New_Feature_Learner = fullyConnectedLayer(Number_of_Classes,...
                      'Name','Anomaly Feature Learner',...
                      'WeightLearnRateFactor',10,...
                      'BiasLearnRateFactor',10);

New_Classifier_Layer = classificationLayer('Name', 'Anomaly Classifier');
```

Postavlja se $WeightLearnRateFactor = 10$ i $BiasLearnRateFactor = 10$ kako bi novi slojevi imali veću brzinu učenja.

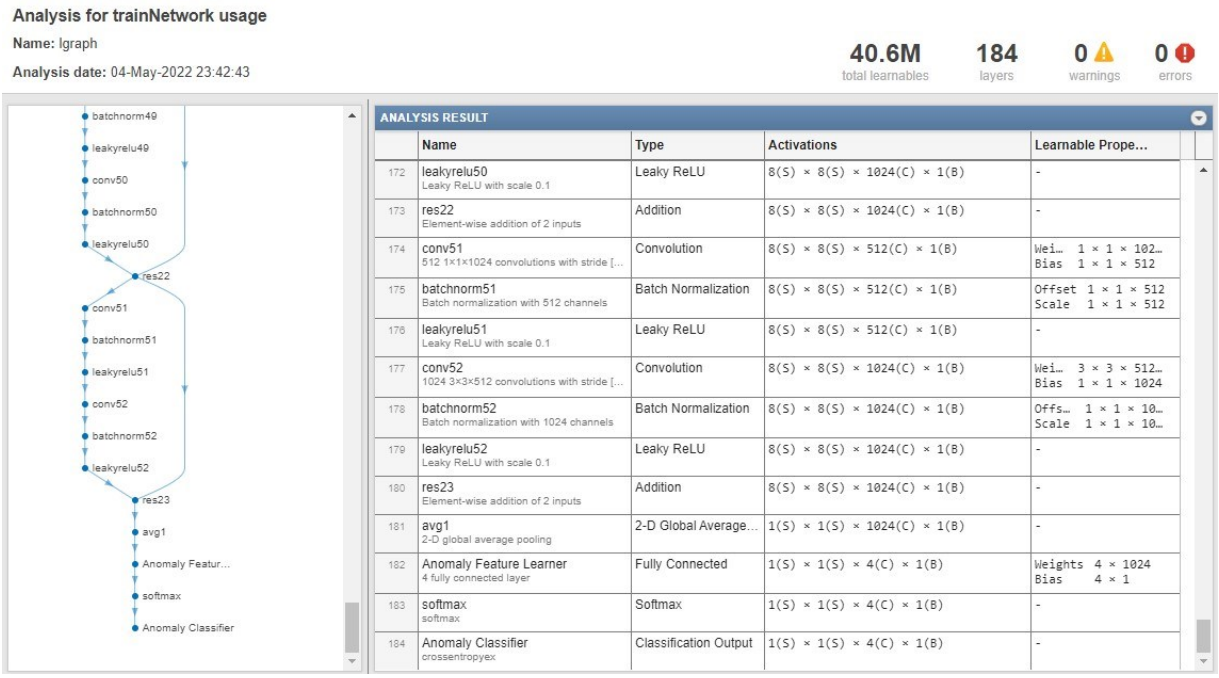
Kako bi bilo moguće modificirati arhitekturu mreže potrebno ju je pretvoriti u grafički prikaz slojeva pomoću fukcije *layerGraph*:

```
Layer_Graph = layerGraph(net);
```

Sada je moguće zamijeniti stari *feature learner* sloj sa novim *feature learner* slojem te stari klasifikacijski sloj sa novim klasifikacijskim slojem koji ima četiri klase:

```
lgraph = replaceLayer(Layer_Graph, Feature_Learner.Name, New_Feature_Learner);
lgraph = replaceLayer(lgraph, Output_Classifier.Name, New_Classifier_Layer);
```

Zadnji slojevi modificirane mreže *lgraph* prikazani su na slici 41. pomoću naredbe *analyzeNetwork(lgraph).*



Slika 41. Modificirana *Darknet53* konvolucijska neuronska mreža

Sada je modificirana *Darknet53* mreža spremna za trening, a prije samog treninga mreže potrebno je definirati hiper-parametre i postavke za trening:

```
Size_of_Minibatch = 4;
Validation_Frequency = floor(numel(Resized_Training_image.Files)/Size_of_Minibatch);

Training_Options = trainingOptions('sgdm',...
  'MiniBatchSize', Size_of_Minibatch,...
  'MaxEpochs', 100,...
  'InitialLearnRate', 1e-4,...
  'Shuffle','every-epoch',...
  'ValidationData', Resized_Validation_image,...
  'ValidationFrequency',Validation_Frequency,...
  'Verbose', false,...
  'Plots', 'training-progress');
```

Gdje je:

- *sgdm* – vrsta algoritma koji ažurira parametre tijekom treninga mreže poduzimajući male korake u smjeru negativnog gradijenta funkcije gubitka
- *MiniBatchSize* - Veličina mini-skupa slika, podskupa skupa slika za trening, koristi se za procjenu gradijenta funkcije gubitka i ažuriranje parametara.
- *MaxEpochs* – maksimalni broj epoha, jedna epoha je potpuni prolaz algoritma za trening mreže kroz cijeli skup slika za trening

- *InitialLearnRate* - hiper-parametar koji kontrolira koliko prilagođavamo parametre (filtere) mreže s obzirom na gradijent gubitka
- *Shuffle* – opcija za frekvenciju mješanja slika koje ulaze u mini-skup, odabrana je opcija miješanja slika svaku epohu
- *ValidationData* – odabir skupa slika za validaciju
- *ValidationFrequency* – frekvencija provjere valjanosti mreže iskazana kao broj iteracija
- *Verbose* - indikator za prikaz informacija o napretku treninga u naredbenom prozoru
- *Plots* – opcija za prikaz treninga mreže na grafikonu

U svrhu što boljeg iskorištavanja parametara već istrenirane *Darknet53* mreže korisno je zamrznuti početnih 10 slojeva kako bi parametri tih slojeva ostali nepromijenjeni tijekom treninga, što povećava preciznost i ubrzava vrijeme treninga mreže:

```
layers = lgraph.Layers;
connections = lgraph.Connections;

for i = 1:10
  if isprop(layers(i),'WeightLearnRateFactor')
layers(i).WeightLearnRateFactor = 0;
  end
  if isprop(layers(i),'WeightL2Factor')
layers(i).WeightL2Factor = 0;
  end
  if isprop(layers(i),'BiasLearnRateFactor')
layers(i).BiasLearnRateFactor = 0;
  end
  if isprop(layers(i),'BiasL2Factor')
layers(i).BiasL2Factor = 0;
  end
end
```

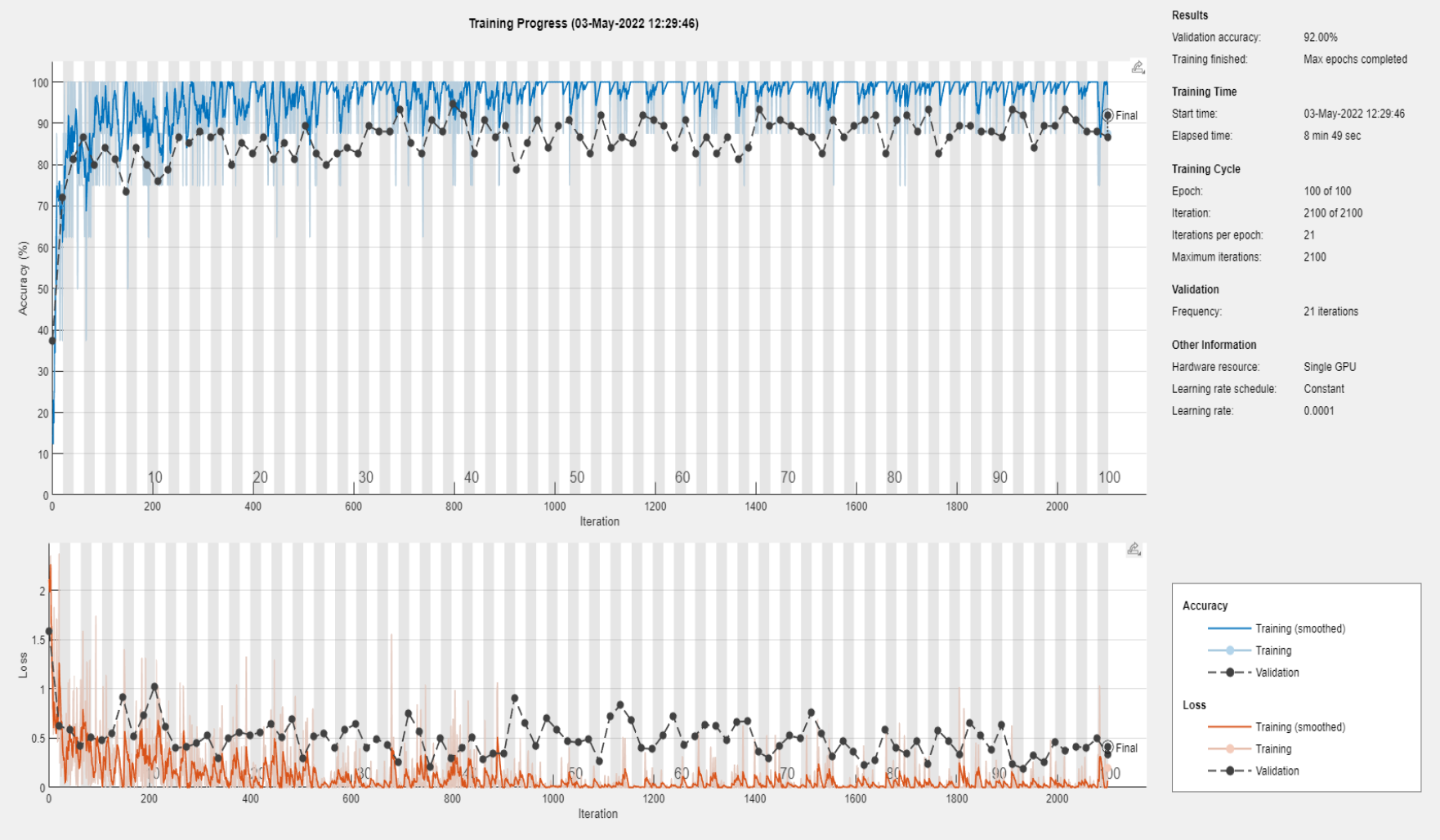
Sada je moguće provesti trening mreže pomoću funkcije *trainNetwork* koja za ulaz uzima skup slika za trening mreže, modificiranu *Darknet53* mrežu i opcije treninga, a za izlaz daje istreniranu *darknet53_CNN* mrežu:

```
darknet53_CNN = trainNetwork(Resized_Training_image, lgraph, Training_Options);
save darknet53_CNN
```

Napredak treninga prikazan je na slici 42. Provedeno je ukupno 2100 iteracija odnosno 21 iteracija po epohu u skoro 9 minuta. Svaka iteracija je procjena gradijenta funckije gubitka i ažuriranje parametara u negativnom smjeru tog gradijenta. Postignuta preciznost mreže iznosi 92%.

Trening *Darknet53* konvolucijske neuronske mreže koja nije unaprijed istrenirana na *imagenet* skupu slika, na istim postavkama za trening, rezultirao je s preciznošću mreže od 62.5% i vremenom treninga od 11 minuta. Provođenjem prijenosnog učenja bez zamrzavanja 10 početnih slojeva preciznost mreže je povećana na 90%. Zamrzavanjem početnih slojeva preciznost se povećala za skromnih 2% a vrijeme treniranja se smanjilo za 3 minute.

Istim postupkom, odnosno provođenjem prijenosnog učenja i zamrzavanjem 10 početnih slojeva, istrenirane su *GoogleNet* i *Resnet50* konvolucijske neuronske mreže. Preciznost *GoogleNet* mreže iznosila je 89.33%, a vrijeme treninga 6 minuta. Preciznost *Resnet50* mreže iznosila je također 89.33% sa nešto većim vremenom treninga, koji je iznosio 8 minuta. S obzirom na najveću preciznost *Darknet53* mreže upravo je ta mreža odabrana kao *backbone* mreža koja će se koristiti prilikom kreiranja detektora anomalija u nastavku.

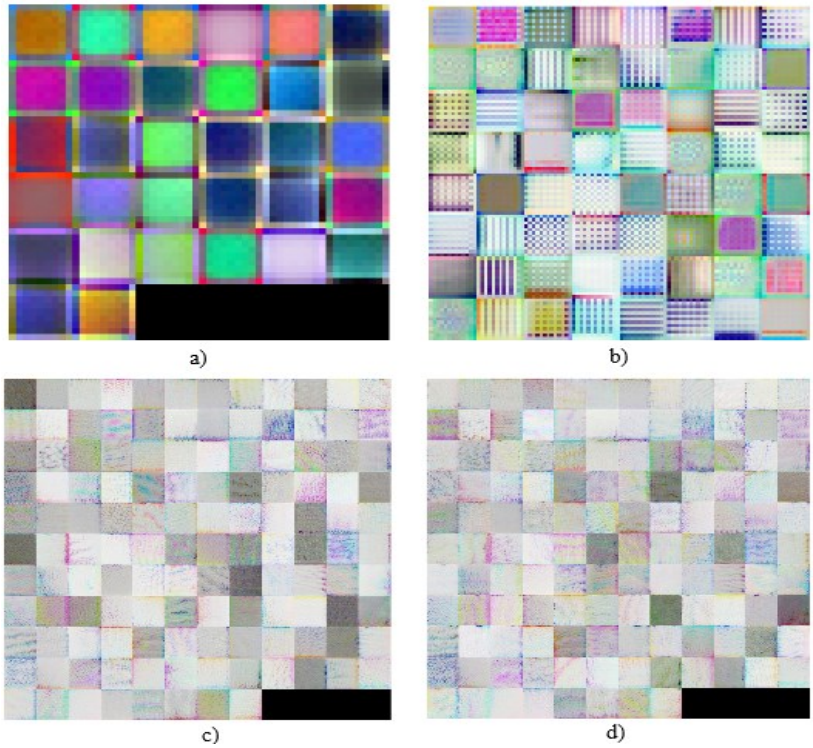


Slika 42. Napredak treninga modificirane *Darknet53* mreže

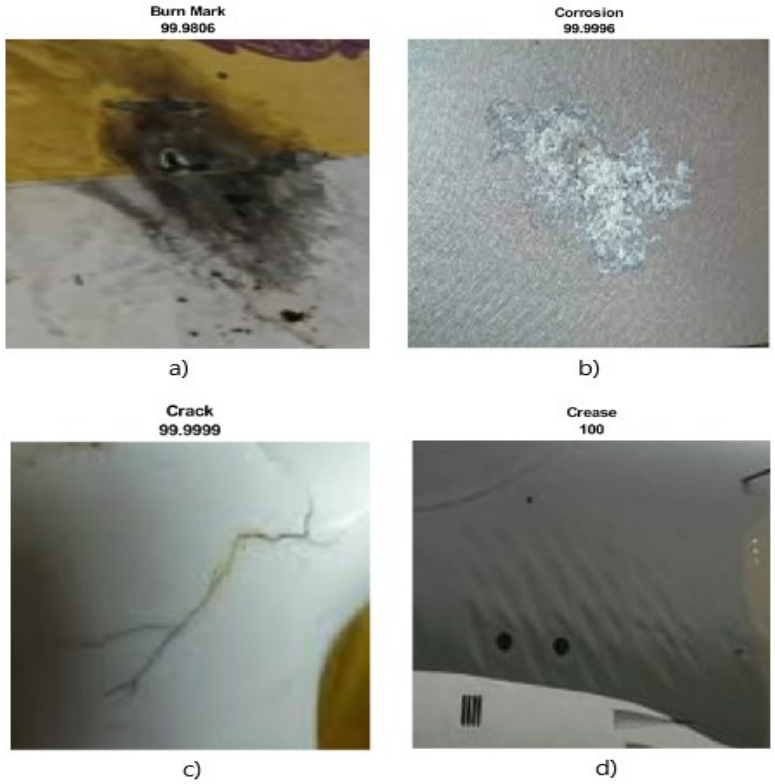
Na slici 43 prikazane su naučene mape značajki različitih slojeva mreže. Na slici 43a) prikazane su mape značajki prvog konvolucijskog sloja čija dubina tenzora iznosi 32 odnosno koji sadrži 32 mape značajki. Na slici 43b) prikazane su mape značajki drugog konvolucijskog sloja čija dubina tenzora iznosi 64. Na slici 43c) prikazane su 128 mapi značajki 19. konvolucijskog sloja odnosno 63. sloja u mreži, a na slici 43c) te iste mape značajki nakon što su prošle kroz *leakyReLU* sloj. Može se primjetiti da mreža u početnim slojevima uči raspoznavati osnovne značajke kao što su različite boje te vertikalni i horizontalni rubovi, a u dubljim slojevima ekstraktirane značajke postaje sve kompleksnije. Mape značajki vizualiziraju se pomoću:

```
%Primjer vizualizacije mapa značajki prvog konvolucijskog sloja
 layer = 2; %conv1 layer
 name = net.Layers(layer).Name

 channels = 1:32;
I = deepDreamImage(net,name,channels, ...
    'PyramidLevels',1);

figure
I = imtile(I,'ThumbnailSize',[64 64]);
imshow(I)
title(['Layer ',name,' Features'],'Interpreter','none')

layer = 63; %conv19
 name = net.Layers(layer).Name

 channels = 1:128;
I = deepDreamImage(net,name,channels, ...
    'Verbose',true, ...

    "NumIterations",20, ...
    'PyramidLevels',2);
figure
I = imtile(I,'ThumbnailSize',[250 250]);
imshow(I)
name = net.Layers(layer).Name;
title(['Layer ',name,' Features'],'Interpreter','none')
```

Na slici 44. prikazane su klasifikacije slika sa odgovarajućim vjerojatnostima pomoću istrenirane *darknet_CNN* mreže. Na slici 44a) prikazana je klasifikacija anomalije u kategoriju oštećenja zbog udara groma sa sigurnošću od 99.98%. Na slici 44b) prikazana je klasifikacija anomalije u kategoriju korozije sa sigurnošću od 99.99%, na slici 44c) klasifikacija anomalije u kategoriju pukotine sa sigurnošću od 99.99%, a na slici 44d) klasifikacija anomalije u kategoriju nabora sa sigurnošću od 100%.



Slika 43. Mape značajki različitih slojeva mreže a) Mape značajki prvog konvolucijskog sloja b) Mape značajki drugog konvolucijskog sloja c) Mape značajki 19. konvolucijskog sloja d) Mape značajki 19. *leakyReLU* sloja



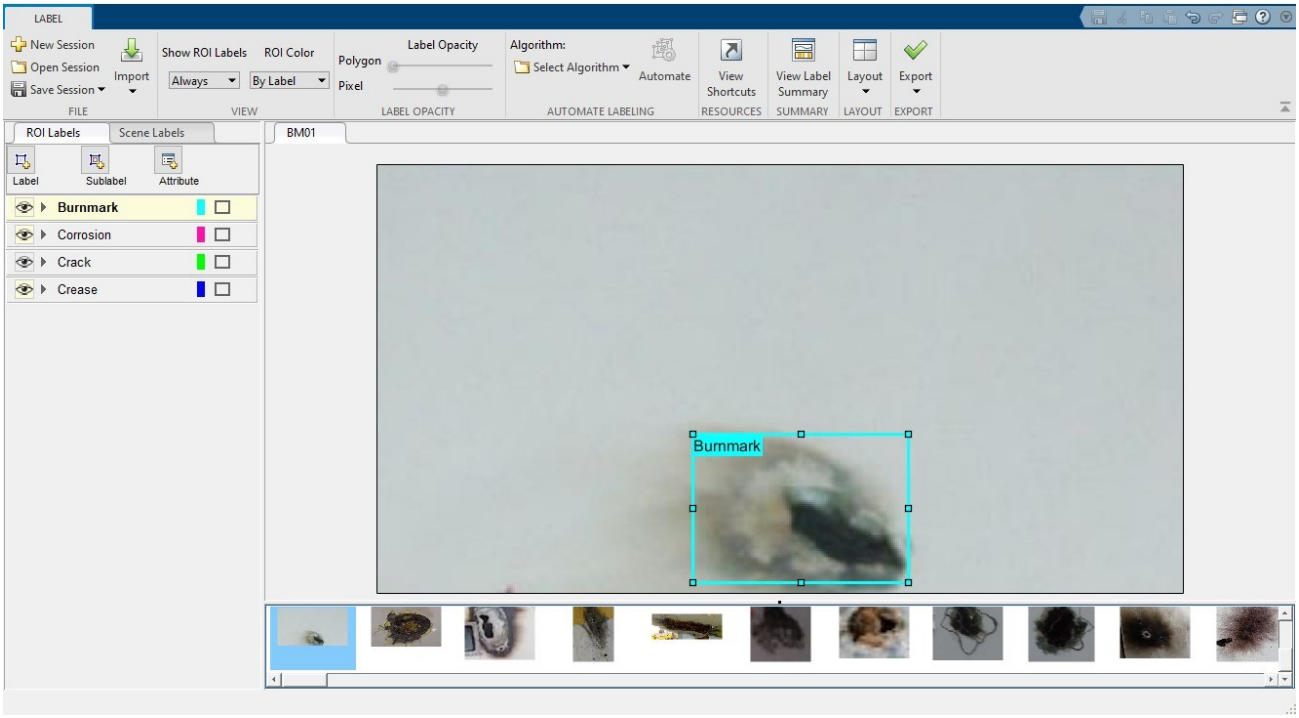
Slika 44. Rezultati klasifikacije anomalija trupa zrakoplova a) Klasifikacija udara groma b)Klasifikacija korozije c) Klasifikacija pukotine d) Klasifikacija nabora

7. TRENIRANJE DETEKTORA ANOMALIJA

Za automatiziranu detekciju anomalija trupa zrakoplova u stvarnom vremenu predlaže se razvoj detektora iz YOLO familije algoritama. Prikazat će se proces kreiranja i treniranja YOLO v2 i YOLO v4 detektora različitih arhitektura radi mogućnosti usporedbe rezultata. Sve potrebne datoteke, dodatne funkcije i slike nalaze se u prilogu.

7.1 YOLO v2 detektor anomalija

Naredbom *imageLabeler* u naredbenom prozoru otvara se *ImageLabeler* aplikacija za kreiranje temeljne istine (slika 45). U aplikaciju se pomoću opcije *Import* uvoze slike iz foldera *Anomalije_trupa* koje su primjerene za trening detektora na manjoj rezoluciji. Kreiraju se četiri *Label*-a odnosno klasa pomoću kojih se označava temeljna istina na slikama anomalija. Nakon označavanja anomalija potrebno je opcijom *Export* → *To workspace* eksportirati slike s označenom temeljnom istinom u obliku tablice u kojoj prvi stupac sadrži put do slika a ostali stupci su kategorije anomalija koji sadržavaju koordinate graničnih okvira. Eksportirana tablica sprema se u *gTruth.mat* datoteku.



Slika 45. *ImageLabeler* aplikacija

Sljedeći korak je učitavanje istrenirane konvolucijske neuronske mreže iz prethodnog poglavlja koja će služiti kao *backbone* mreža detektora:

```
load darknet53_CNN.mat;
```

Potrebno je dodati put do *src* foldera kako bi se mogle koristiti pomoćne funkcije iz *helper* foldera te spremiti kreiranu temeljnu istinu u *data* varijablu:

```
addpath src
data = load('gTruth.mat');
```

Kreirana temeljna istina dijeli se na tri skupa slika: skup za trening mreže (60%), skup za validaciju tijekom treninga (30%) i skup za testiranje istreniranog detektora (10%):

```
AnomalyDataset = data.gTruth;
AnomalyDataset.imageFilename = fullfile(AnomalyDataset.imageFilename);

rng(0);
shuffledIndices = randperm(height(AnomalyDataset));
idx = floor(0.6 * length(shuffledIndices) );

trainingIdx = 1:idx;
trainingDataTbl = AnomalyDataset(shuffledIndices(trainingIdx),:);

validationIdx = idx+1 : idx + 1 + floor(0.3 * length(shuffledIndices) );
validationDataTbl = AnomalyDataset(shuffledIndices(validationIdx),:);

testIdx = validationIdx(end)+1 : length(shuffledIndices);
testDataTbl = AnomalyDataset(shuffledIndices(testIdx),:);


imdsTrain = imageDatastore(trainingDataTbl{:,'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));

imdsValidation = imageDatastore(validationDataTbl{:,'imageFilename'});
bldsValidation = boxLabelDatastore(validationDataTbl(:, 2:end));

imdsTest = imageDatastore(testDataTbl{:,'imageFilename'});
bldsTest = boxLabelDatastore(testDataTbl(:, 2:end));

trainingData = combine(imdsTrain,bldsTrain);
validationData = combine(imdsValidation,bldsValidation);
testData = combine(imdsTest,bldsTest);
```

Vrijednosti graničnih okvira označenih prilikom kreiranja temeljne istine trebaju biti konačne, pozitivne i trebaju biti unutar granice slike s pozitivnom visinom i širinom. Svi nevažeći uzorci moraju se ili odbaciti ili popraviti prije treninga. Provjera označenih slika može se provesti pozivanjem pomoćne funkcije *validateInputData*:

```
helper.validateInputData(trainingData);
helper.validateInputData(testData);
helper.validateInputData(validationData)
```

Augmentacija slika za trening provodi se pozivanjem funkcije *transform* i pomoćne funkcije *augmentData* koja se nalazi unutar *helper* foldera.

```
augmentedTrainingData = transform(trainingData, @helper.augmentData);


% Vizualizacija slike koja je augmentirana četiri puta:
augmentedData = cell(4,1);
for k = 1:4
    data = read(augmentedTrainingData);
    augmentedData{k} = insertShape(data{1,1}, 'Rectangle', data{1,2});
    reset(augmentedTrainingData);
end
figure
montage(augmentedData, 'BorderSize', 10)
```

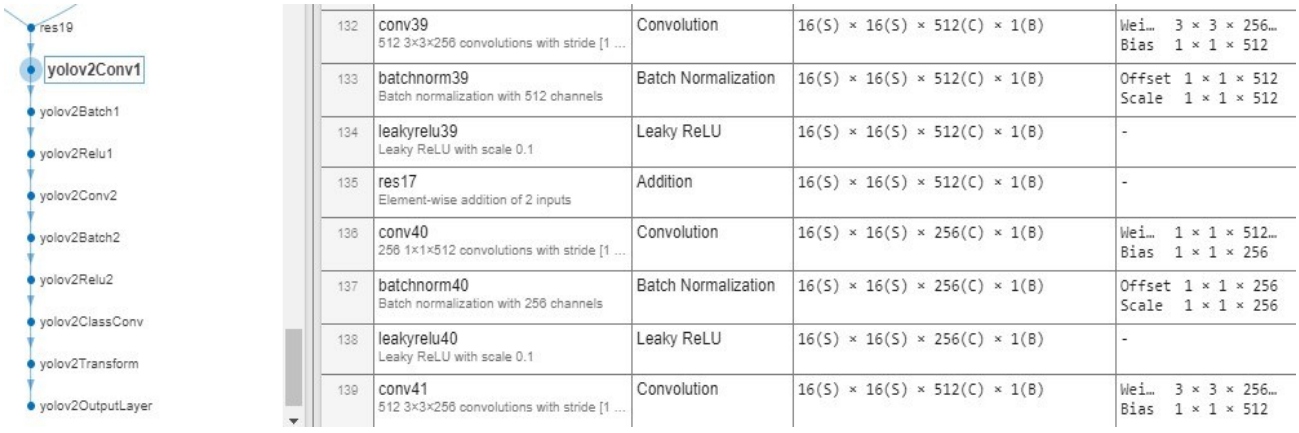
Slijedi definiranje rezolucije na kojoj se trenira detektor. U idealnom slučaju rezolucija bi trebala biti slična rezoluciji slika za trening i veća od rezolucije na kojoj se trenirala konvolucijska neuronska mreža. Minimalna rezolucija na kojoj se detektor može trenirati jednaka je rezoluciji treninga mreže. Pomoću funkcije *transform* i pomoćne funkcije *preprocessData* veličina slika za procjenu sidrenih okvira mijenja se na veličinu slika za trening detektora. Definira se proizvoljan broj sidrenih okvira te se njihove dimenzije procjenjuju pomoću funkcije *estimateAnchorBoxes*:

```
inputSize = [256 256 3];
numClasses = width(AnomalyDataset)-1;

trainingDataForEstimation =
transform(trainingData,@(data)helper.preprocessData(data,inputSize));

numAnchors = 7;


[anchorBoxes, meanIoU] = estimateAnchorBoxes(trainingDataForEstimation, numAnchors)
```

Modificiranje konvolucijske mreže za klasifikaciju anomalija u detektor anomalija vrši se pomoću funkcije *yolov2Layers* koja za ulaz uzima veličinu slika na kojima će se trenirati detektor, broj klasa, sidrene okvire, konvolucijsku mrežu za klasifikaciju (*featureExtractionNetwork*) i sloj za ekstrakciju značajki (*featureLayer*) iza kojeg će se slojevi klasifikacijske mreže zamijeniti s glavom za detekciju koja se sastoji od niza slojeva konvolucije, *ReLU* i *batch* normalizacijskih slojeva, zajedno s YOLO v2 transformacijskim i izlaznim slojevima. Sloj za skupnu normalizaciju (*batch normalization layer*) je metoda koja se koristi za ubrzavanje i stabiliziranje konvolucijske neuronske mreže kroz normalizaciju ulaza slojeva ponovnim centriranjem i skaliranjem. Transformacijski sloj pretvara izlaz konvolucijske klasifikacijske mreže u oblik potreban za detekciju objekata. A izlazni sloj definira parametre sidrenih okvira i implementira funkciju gubitka koja se koristi za treniranje detektora.

```
featureExtractionNetwork = darknet53_CNN;

featureLayer = 'res19';

lgraph =
yolov2Layers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,featureLayer);
```

Odabrani sloj za ekstrakciju značajki daje mape značajki veličine 16×16. Ova veličina mapa značajki dobar je kompromis između rezolucije mapa značajki i snage izdvojenih značajki, budući da značajke izdvojene dalje niz mrežu kodiraju jače značajke slike na trošak rezolucije. Odabir optimalnog sloja za ekstrakciju značajki empirijski je postupak. Grafički prikaz kreiranog YOLO v2 detektora dobiven naredbom *analyzeNetwork(lgraph)* prikazan je na slici 46.



Slika 46. Grafički prikaz YOLO v2 detektorske glave

Sljedeći korak je mijenjanje veličina augmentiranih slika za trening i slika za validaciju na *inputSize* i skaliranje piksela u raspon [0 1] pozivanjem funkcije *transform* i pomoćne funkcije *preprocessData* :

```
preprocessedTrainingData =
transform(augmentedTrainingData,@(data)helper.preprocessData(data,inputSize));
preprocessedValidationData =
transform(validationData,@(data)helper.preprocessData(data,inputSize));
data = read(preprocessedTrainingData);

%Prikaz predprocesuirane slike za trening detektora
I = data{1};
bbox = data{2};
annotatedImage = insertShape(I,'Rectangle',bbox);
annotatedImage = imresize(annotatedImage,2);
figure
imshow(annotatedImage)
```

Sada se definiraju postavke za trening detektora:

```
options = trainingOptions('sgdm', ...
        'MiniBatchSize',16, ....
        'InitialLearnRate',1e-4, ...
        'MaxEpochs',300, ...
        'Plots','training-progress',...
        'ValidationData',preprocessedValidationData);
```

Trening YOLO v2 detektora provodi se pomoću funkcije *trainYOLOv2ObjectDetector* koja za ulaz uzima pretprocesuirani skup slika za trening, arhitekturu detektorske mreže i opcije za trening, a za izlaz daje istrenirati detektor i informacije o treningu:

```
[detector,info] = trainYOLOv2ObjectDetector(preprocessedTrainingData,lgraph,options);

save('yolov2_detektor_anomalija',’detector’,'anchorBoxes','numClasses')
```

Nakon treninga, detektor se pokreće na skupu slika za testiranje detektora s $Threshold = 0.1$, a rezultati se spremaju u tablicu:

```
results = table('Size',[height(testDataTbl) 3],...
    'VariableTypes',{'cell','cell','cell'},...
    'VariableNames',{'Boxes','Scores', 'Labels'});

depVideoPlayer = vision.DeployableVideoPlayer;
for i = 1:height(testDataTbl)
```

```
    %Read the image
    I = imread(testDataTbl.imageFilename{i});
    I = imresize(I,[256 256]);

    %Run the detector.
    [bboxes,scores,labels] = detect(yolov2_detector_darknet53,I,...
                                    'Threshold',0.1,'SelectStrongest',false);


    if ~isempty(bboxes)
      [selectedBboxes,selectedScores,selectedLabels,index] =
selectStrongestBboxMulticlass(bboxes,scores,labels,...
    'RatioType','Min','OverlapThreshold',0.2);

     annotations = string(selectedLabels) + ": " + string(selectedScores);

        I = insertObjectAnnotation(I,'rectangle',selectedBboxes,...
                                    cellstr(annotations),'Color','red');
        depVideoPlayer(I);
        pause(2);

    end

    %Collect the results in the results table
    results.Boxes{i} = floor(selectedBboxes);
    results.Scores{i} = selectedScores;
    results.Labels{i} = selectedLabels;

end
```

Sada je moguće provesti evaluaciju modela na rezultatima testa pomoću funkcije *evaluateDetectionPrecision* i *evaluateDetectionMissRate.* Funkcija *evaluateDetectionPrecision* za ulaz uzima rezultate testa, temeljnu istinu i razinu *threshold*-a, a za izlaz daje preciznost (sposobnost detektora da napravi ispravne klasifikacije) i *recall* (sposobnost objekta da pronađe sve relevantne objekte na slici). Ti parametri određuju vrijednost prosječne preciznosti detektora, a idealna vrijednost preciznosti je 1 na svim razinama *recall*-a. Funkcija *evaluateDetectionMissRate* za ulaz također uzima rezultate testa, temeljnu istinu i razinu *threshold*-a, a za izlaz daje broj lažno pozitivnih detekcija po slici i stopu promašaja. Stopa promašaja pokazuje koliko objekata koji se nalaze na slici detektor nije uspio detektirati.

```
threshold = 0.1;

[ap, recall, precision] = evaluateDetectionPrecision(results,
testDataTbl(:,2:end),threshold);

[am,fppi,missRate] = evaluateDetectionMissRate(results, testDataTbl(:,2:end),threshold);
```

```
subplot(1,2,1);
plot(recall{1,1},precision{1,1},'g-','LineWidth',2, "DisplayName",'Burnmark');
hold on;
plot(recall{2,1},precision{2,1},'b-','LineWidth',2, "DisplayName",'Corrosion');
hold on
plot(recall{3,1},precision{3,1},'r-','LineWidth',2, "DisplayName",'Crack');
hold on
plot(recall{4,1},precision{4,1},'y-','LineWidth',2, "DisplayName",'Crease');
hold off;
xlabel('Recall');
ylabel('Precision');
title(sprintf('Average Precision = %.2f\n', ap))
legend('Location', 'best');
legend('boxoff')
grid on
subplot(1,2,2);
loglog(fppi{1,1}, missRate{1,1},'-g','LineWidth',2, "DisplayName",'Burnmark');
hold on;
loglog(fppi{2,1}, missRate{2,1},'-b','LineWidth',2,"DisplayName",'Corrosion');
hold on
loglog(fppi{3,1}, missRate{3,1},'-r','LineWidth',2,"DisplayName",'Crack');
hold on
loglog(fppi{4,1}, missRate{4,1},'-y','LineWidth',2,"DisplayName",'Crease');
hold off;
xlabel('False Positives Per Image');
ylabel('Log Average Miss Rate');
title(sprintf('Log Average Miss Rate = %.2f\n', am))
legend('Location', 'best');
legend('boxoff')
grid on
```

7.2 YOLO v4 detektor anomalija

Proces kreiranja temeljne istine i skupova slika za trening, validaciju i test modela, augmentacija i pretprocesuiranje slika te evaluacije modela kod YOLO v4 detektora ista je kao i kod kreiranja YOLO v2 detektora anomalija. Također, korištena je ista istrenirana *darknet53_CNN* klasifikacijska konvolucijska neuronska mreža kao *backbone* mreža za detektor.

Jedna od razlika je broj sidrenih okvira kojih mora biti 3 po jednoj detektorskoj glavi. Kako je kreiran YOLO v4 detektor s dvije detektorske glave, ukupni broj sidrenih okvira (*numAnchors*) bit će 6. Potrebno ih je sortirati od većih prema manjim sidrenim okvirima u dvije ćelije, jedna za svaku detektorsku glavu. Također, potrebno je definirati imena klasa na kojima će se trenirati detektor.

```
numAnchors = 6;
```

```
[anchors, meanIoU] = estimateAnchorBoxes(trainingDataForEstimation, numAnchors);

area = anchors(:, 1).*anchors(:, 2);
[~, idx] = sort(area, 'descend');
anchors = anchors(idx, :);
anchorBoxes = {anchors(1:3,:)
    anchors(4:6,:)};

classes = {'Burnmark','Corrosion','Crack','Crease'};
```

Sljedeći korak je pomoću funkcije *layerGraph* pretvoriti konvolucijsku mrežu u grafički prikaz slojeva kako bi se mogao zamijeniti stari ulazni sloj s novim ulaznim slojem koji ne normalizira ulazne slike i maknuti zadnji klasifikacijski sloj. Nakon toga se mreža pretvara u *deep learning* mrežu pomoću naredbe *dlnetwork*. Također, definira se veličina slika novog ulaznog sloja.

```
lgraph = layerGraph(darknet53_CNN);
imageSize = [320 320 3];

layerName = lgraph.Layers(1).Name;
newinputLayer = imageInputLayer(imageSize,'Normalization','none','Name',layerName);


lgraph = removeLayers(lgraph,'Anomaly Classifier');
lgraph = replaceLayer(lgraph,layerName,newinputLayer);


dlnet = dlnetwork(lgraph);
```

Nakon toga je potrebno definirate slojeve za ekstrakciju značajki odnosno slojeve na koje će se spojiti detektorske glave:

```
featureExtractionLayers = ["res19",'res23'];
```

Odabrani slojevi za ekstrakciju značajki daju mape značajki veličina 20×20 i 10×10.

Sada je moguće kreirati YOLO v4 detektor pomoću funkcije *yolov4ObjectDetector* koja za ulaz uzima arhitekturu *dlnet* mreže, klase, sidrene okvire i slojeve za ekstrakciju značajki:

```
detector = yolov4ObjectDetector(dlnet,classes,anchorBoxes,…
                                DetectionNetworkSource=featureExtractionLayers);
disp(detector)
analyzeNetwork(detector.Network)
```

Prije samog treninga detektora potrebno je definirati opcije za trening:

```
options = trainingOptions("adam",...
    GradientDecayFactor=0.9,...
    SquaredGradientDecayFactor=0.999,...
    InitialLearnRate=0.0001,...
    LearnRateSchedule="none",...
    MiniBatchSize=16,...
    L2Regularization=0.0005,...
    MaxEpochs=300,...
    BatchNormalizationStatistics="moving",...
    DispatchInBackground=false,...
    ResetInputNormalization=false,...
    Shuffle="every-epoch",...
    VerboseFrequency=1,...
    OutputNetwork="best-validation-loss",...
    ExecutionEnvironment="auto",...
    ValidationFrequency=30,...
    ValidationData=preprocessedValidationData);
```

Trening YOLO v4 detektora provodi se pomoću funkcije *trainYOLOv4ObjectDetector* koja za ulaz uzima pretprocesuirani skup slika za trening, arhitekturu detektorske mreže i opcije za trening, a za izlaz daje istrenirati detektor i informacije o treningu:

```
[detector,info] = trainYOLOv4ObjectDetector(preprocessedTrainingData,detector,options);

 save('yolov4_detector','detector','dlnet','classes','anchorBoxes','info')
```

7.3 Prikaz rezultata

U tablici 4. prikazani su rezultati treninga različitih detektora anomalija i konfiguracija računala na kojemu je izvršen trening. Može se primijetiti da *adam* algoritam za ažuriranje parametara daje malo bolje rezultate, odnosno gubitci pri validaciji su niži, ali vrijeme treninga je veće te se općenito povećava s povećanjem rezolucije na kojoj se detektor trenira. Također, vidljivo je da YOLO v4 sa dvije detektorske glave postiže najmanje gubitke tijekom validacije. Potrebno je napomenuti da je YOLO v4 treniran s manjom brzinom učenja zbog detekcija neispravnih graničnih okvira tijekom treninga što je uzrokovalo prekid procesa istog.

Tablica 4. Usporedba istreniranih detektora anomalija

Detektor	Algoritam za ažuriranje parametara tijekom treninga	Brzina učenja	Rezolucija	Broj detektorskih glava	Vrijeme treninga	Gubitak pri validaciji
YOLO v2	*sgdm*	1e-4	[256 256]	1	25 min	4.0619
YOLO v2	*adam*	1e-4	[256 256]	1	30 min	3.3627
YOLO v2	*sgdm*	1e-4	[608 608]	1	50 min	2.8588
YOLO v2	*adam*	1e-4	[608 608]	1	60 min	2.8073
YOLO v4	*adam*	1e-5	[256 256]	1	125 min	1.7731
YOLO v4	*adam*	1e-5	[320 320]	2	150 min	1.4131

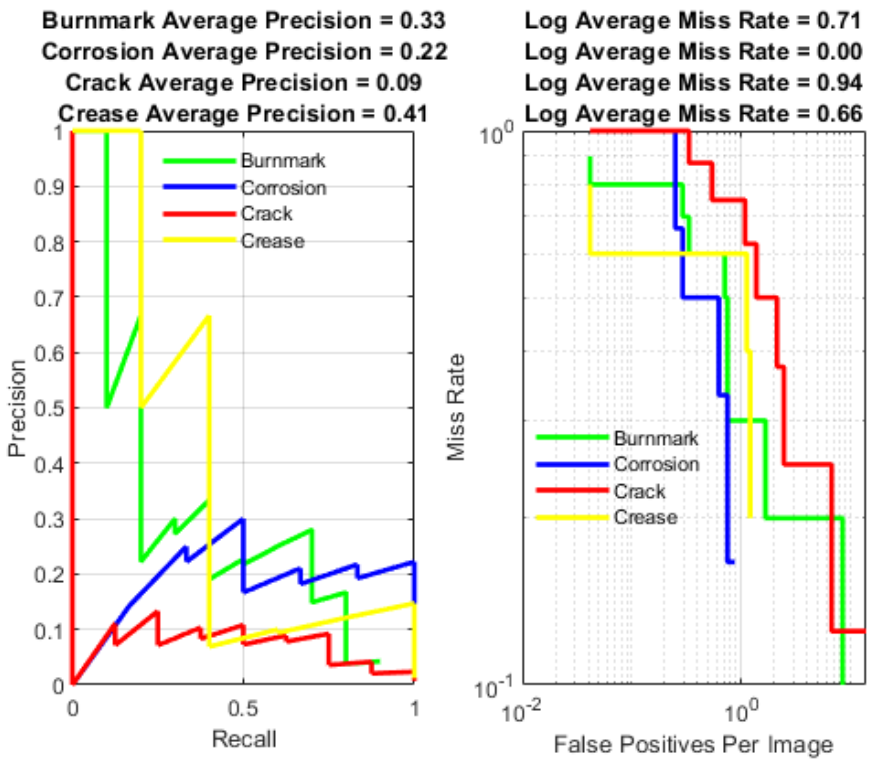
Konfiguracija računala	Procesor	Memorija	Grafička kartica
	AMD Threadripper sa 64 jezgre	256 GB	2x nVidia RTX 3090 24GB

Na slikama 47-52. prikazani su sljedeći rezultati:

- *Precision* – mjera sposobnosti detektora da napravi ispravne klasifikacije, omjer broja istinitih pozitivnih rezultata i ukupnog broja pozitivnih predviđanja
- *Recall* - mjera sposobnosti detektora da detektira sve relevantne objekte na slici, omjer broja istinitih pozitivnih predviđanja i ukupnog broja stvarnih objekata
- *Average precision* – srednja preciznost detektora, jedan broj koji uključuje preciznost i *recall*
- *Miss Rate* – stopa promašaja, omjer lažno negativnih i pozitivnih predikcija
- *False positves per image* – broj lažno pozitivnih predikcija po slici
- *Log Average Miss rate* – srednja stopa promašaja detektora, jedan broj koji uključuje stopu promašaja i broj lažno pozitivnih detekcija po slici

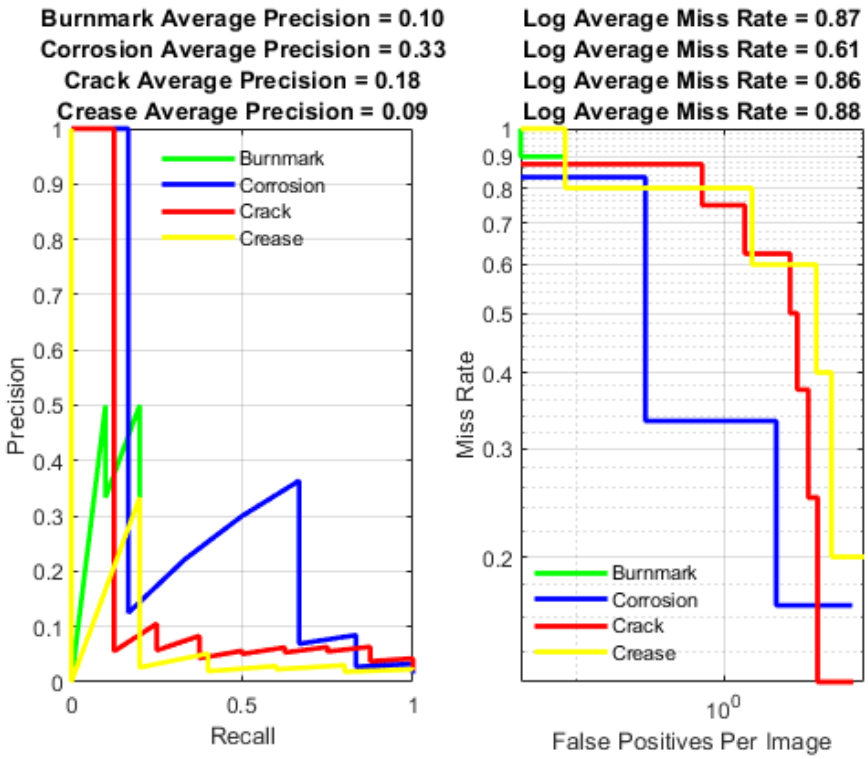
Dobro istreniranim detektorom se smatra detektor koji ima vrijednost preciznosti 1 na svim razinama *recall*-a i što manju stopu promašaja i broj lažno pozitivnih predviđanja po slici.

Na slici 47. prikazana je evaluacija YOLO v2 detektora treniranog na [256 256] rezoluciji koristeći *adam* algoritam za ažuriranje parametara pri *threshold* = 0.1.

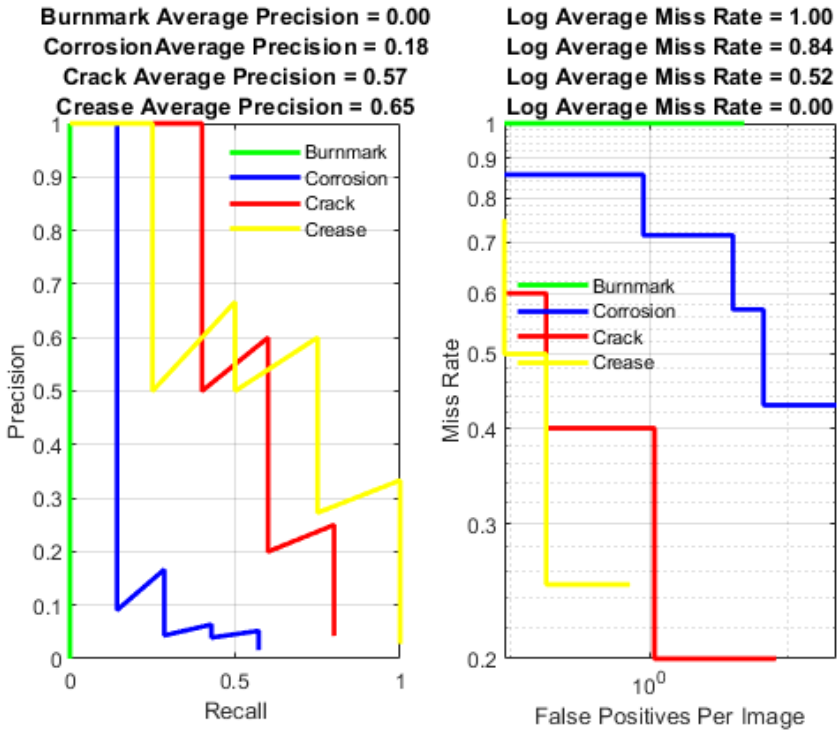


Slika 47. Prosječna preciznost (lijevo) i stopa promašaja (desno) YOLO v2 detektora treniranog na [256 256] rezoluciji koristeći *adam* algoritam za ažuriranje parametara pri *threshold = 0.1*

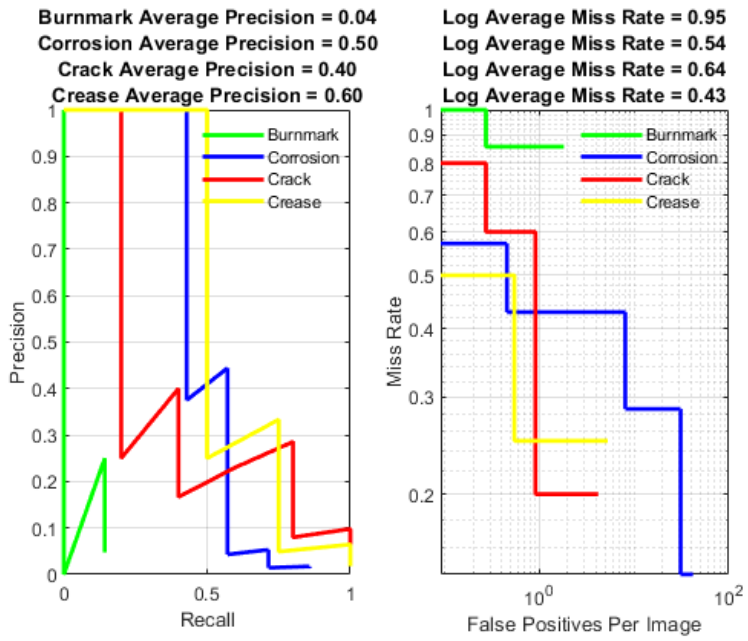
Može se primijetiti da je ovaj detektor najpreciznije detektira oštećenje od udara groma sa preciznošću od 33%, a najslabije pukotine zbog velike stope promašaja od 94%. Na slici 48. prikazana je evaluacija YOLO v2 detektora treniranog na [256 256] rezoluciji koristeći *sgdm* algoritam za ažuriranje parametara. Iako su ostale postavke za trening iste, može se primijetiti da ovaj detektor bolje predviđa anomalije poput korozije i pukotina a lošije oštećenja od udara groma i nabore. Na slici 49. prikazana je evaluacija YOLO v2 detektora treniranog na [608 608] rezoluciji koristeći *adam* algoritam, a na slici 50. je prikazana evaluacija YOLO v2 detektora treniranog na istoj rezoluciji koristeći *sgdm* algoritam. Slike korištene za treniranje klasifikacijske mreže malih su rezolucija, pa su se u svrhu treniranja detektora na većim rezolucijama te slike spajale kako se nebi izgubilo previše značajki rastezanjem male slike na veću rezoluciju. Anomalije su zbog toga postale manjih dimenzija u odnosu na cijelu sliku što rezultira lošom detekcijom malih anomalija kao što su *Burnmark* i *Corrosion*, iako se može vidjeti da detektor treniran na *sgdm* algoritmu daje bolju preciznost u slučaju korozije za 32%. To potvrđuje nedostatak YOLO v2 modela, a to je loša performansa kod detekcije malih objekata.



Slika 48. Prosječna preciznost (lijevo) i stopa promašaja (desno) YOLO v2 detektora treniranog na [256 256] rezoluciji koristeći *sgdm* algoritam za ažuriranje parametara pri *threshold = 0.1*

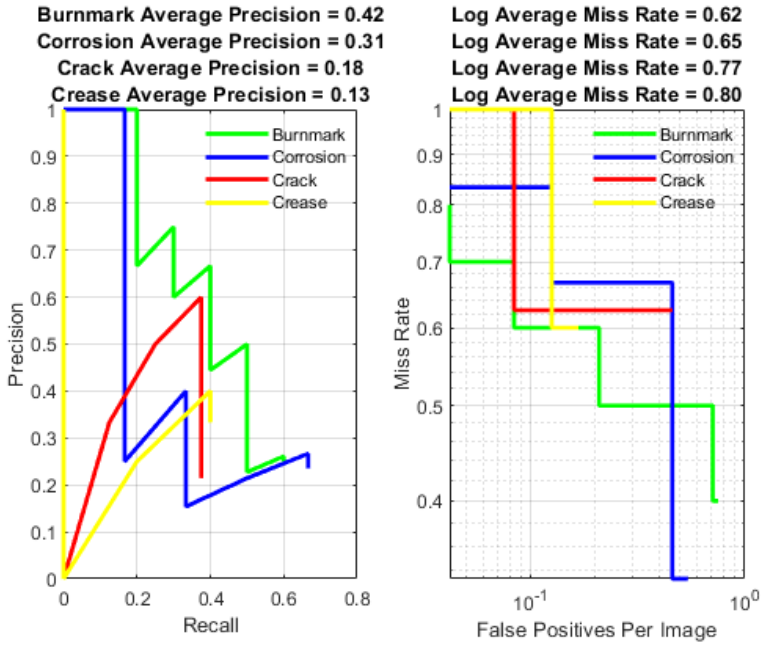


Slika 49. Prosječna preciznost (lijevo) i stopa promašaja (desno) YOLO v2 detektora treniranog na [608 608] rezoluciji koristeći *adam* algoritam za ažuriranje parametara pri *threshold = 0.1*

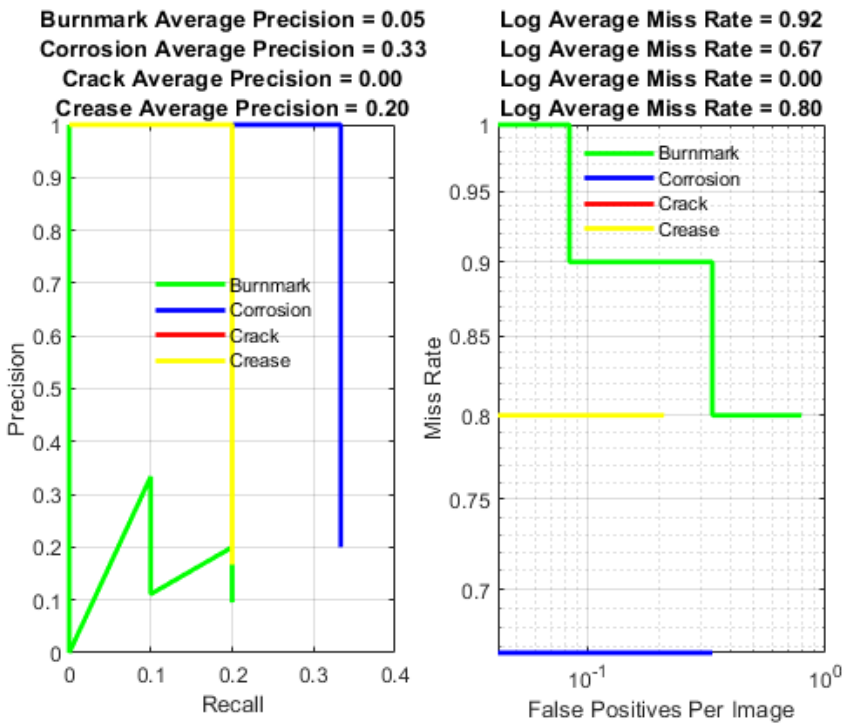


Slika 50. Prosječna preciznost (lijevo) i stopa promašaja (desno) YOLO v2 detektora treniranog na [608 608] rezoluciji koristeći sgdm algoritam za ažuriranje parametara pri *threshold = 0.1*

Na slikama 51. i 52. prikazana je evaluacija YOLO v4 modela s jednom i dvije detektorske glave. Za bolje rezultate potrebno je povećati broj epoha treninga detektora i provesti optimizaciju hiper-parametara za trening.



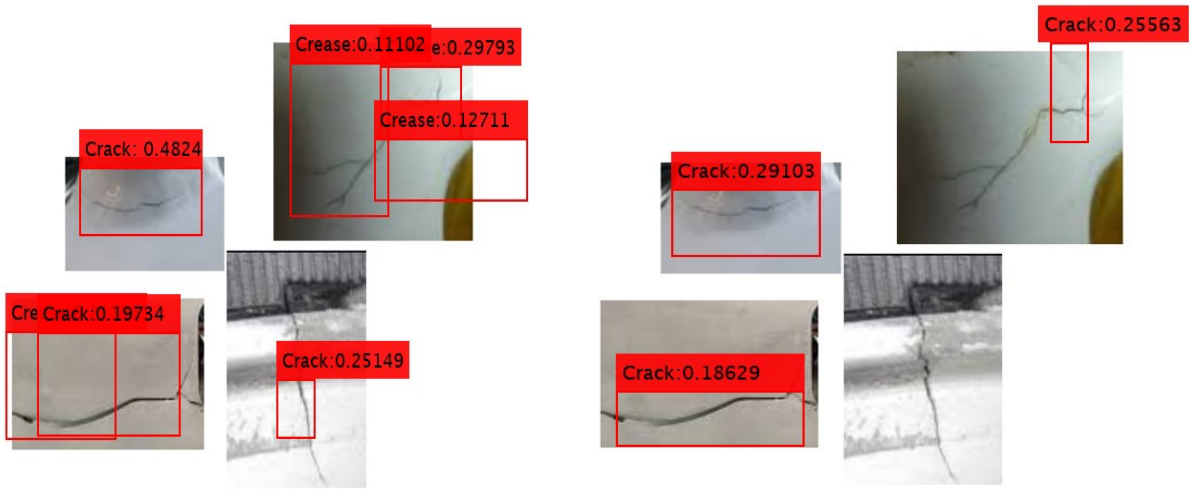
Slika 51. Prosječna preciznost (lijevo) i stopa promašaja (desno) YOLO v4 detektora treniranog na [256 256] rezoluciji koristeći adam algoritam za ažuriranje parametara pri *threshold = 0.1*.



Slika 52. Prosječna preciznost (lijevo) i stopa promašaja (desno) YOLO v4 detektora sa 2 detektorske glave treniranog na [320 320] rezoluciji koristeći adam algoritam za ažuriranje parametara pri *threshold = 0.1*

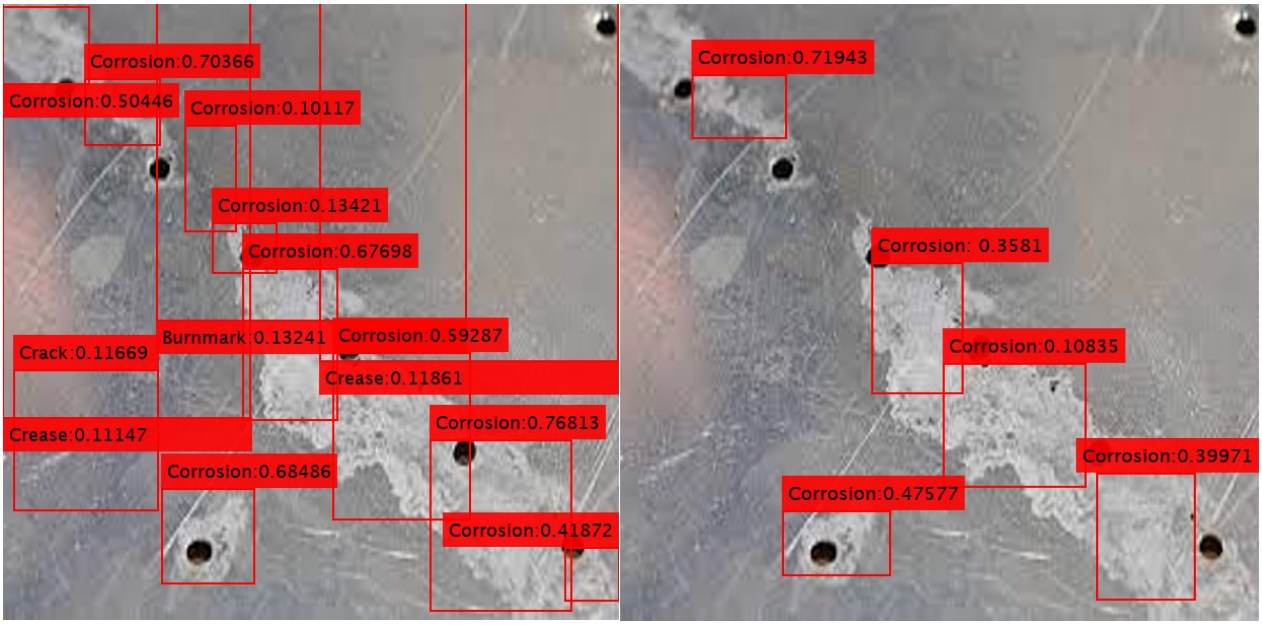
Na slikama 53-56. prikazana je usporedba detekcija anomalija pri *threshold* = 0.1 YOLO v2 detektora treniranog na [608 608] rezoluciji koristeći *adam* algoritam i YOLO v4 detektora sa 2 glave treniranog na [320 320] rezoluciji također koristeći *adam* algoritam.

Slika 53. Detekcija udara groma (*burnmark*) pomoću YOLO v2 algoritma (lijevo) i YOLO v4 algoritma (desno)



Slika 54. Detekcija pukotina (*crack*) pomoću YOLO v2 algoritma (lijevo) i YOLO v4 algoritma (desno)

Slika 55. Detekcija nabora (*crease*) pomoću YOLO v2 algoritma (lijevo) i YOLO v4 algoritma (desno)



Slika 56. Detekcija korozije (*corrosion*) pomoću YOLO v2 algoritma (lijevo) i YOLO v4 algoritma (desno)

Može se primjetiti da je YOLO v2 sigurniji u predikciji anomalija, no isto tako vraća puno lažno pozitivnih detekcija dok to nije slučaj kod YOLO v4 detektora sa dvije detektorske glave pri istoj vrijednosti *threshold*-a. Također, prilikom detekcije anomalija na videu koji simulira detekciju uživo, YOLO v4 daje puno bolje vrijednosti *FPS*-a odnosno u mogućnosti je procesuirati više slika po sekundi od YOLO v2 modela.

U tablici 5. prikazani su rezultati evaluacije YOLO v2 i v4 detektora na skupu slika za test pri različitim razinama *threshold*-a.

Tablica 5. Usporedba rezultata pri različitoj razini *threshold*-a

Detektor	algoritam	Rezolucija	Razina *threshold*-a	Prosječna preciznost				Prosječna stopa promašaja			
				Burnmark	*Corrosion*	*Crack*	*Crease*	*Burnmark*	*Corrosion*	*Crack*	*Crease*
YOLO v2	*adam*	[256 256]	0.1	0.33	0.22	0.09	0.41	0.71	0	0.94	0.66
			0.2	0.22	0.22	0.03	0.41	0.76	0	0.99	0.66
YOLO v2	*sgdm*	[256 256]	0.1	0.1	0.33	0.18	0.09	0.87	0.61	0.86	0.88
			0.2	0	0.25	0.13	0.07	0	0.7	0.88	0.88
YOLO v2	*adam*	[608 608]	0.1	0	0.18	0.57	0.65	1	0.84	0.52	0
			0.2	0	0.02	0.52	0.57	1	1	0.52	0.48
YOLO v2	*sgdm*	[608 608]	0.1	0.04	0.5	0.4	0.6	0.95	0.54	0.64	0.43
			0.2	0	0.31	0.38	0.58	1	0.74	0.64	0.43
YOLO v4	*adam*	[256 256]	0.1	0.42	0.31	0.18	0.13	0.62	0.65	0.77	0.8
			0.2	0.24	0.27	0.18	0	0.75	0.71	0.77	1
YOLO v4 - 2 detektorske glave	*adam*	[320 320]	0.1	0.05	0.33	0	0.2	0.92	0.67	0	0.8
			0.2	0.03	0.08	0	0.1	0.94	0.89	0	0.86

8. ZAKLJUČAK

Mala preciznost i visoka stopa promašaja svih razvijenih detektora ne zadovoljavaju visoke regulativne propise u održavanju zrakoplova i direktna su posljedica malog skupa slika za trening. Kao takvi nisu upotrebljivi za autonomnu inspekciju trupa zrakoplova već mogu služiti kao asistencija djelatnicima održavanja. Unatoč tome, daju dobru osnovu za daljnja istraživanja ali i korištenje u praksi, radi svoje brzine i mogućnosti unaprjeđenja. Nadalje, kao što je vidljivo iz prikazanih rezultata prosječne preciznosti u tablici 5. može se zaključiti da YOLO v2 detektor anomalija istreniran na nižoj rezoluciji bolje detektira udar groma i nabore kada se trenira sa *adam* algoritmom, a koroziju i pukotine kada se trenira sa *sgdm* algoritmom za ažuriranje parametara tijekom treninga. Iz tablice 5. također je vidljivo da YOLO v2 istreniran na višoj rezoluciji sa *sgdm* algoritmom ima značajno veću preciznost prilikom detekcije korozije od YOLO v2 detektora treniranog na istoj rezoluciji sa *adam* algoritmom. Lošiji rezultati, prikazani u tablici 5. YOLO v4 detektora s dvije detektorske glave od iste verzije s jednom detektorskom glavom mogu se pripisati kompliciranijoj strukturi samog modela koja zahtijeva veće vrijeme treninga. Iz testa provedenog na videu može se potvrditi da je YOLO v4 detektor, zbog svoje strukture i ostalih unaprjeđenja, u stanju procesuirati više slika u sekundi od YOLO v2 detektora anomalija. Također, zbog više glava za detekciju, primjereniji je za inspekciju trupa zrakoplova zbog bolje mogućnosti detektiranja malih anomalija. Sa slika 53-56. može se primijetiti da YOLO v4 detektor s dvije detektorske glave, iako manje siguran u točnost predikcija, ne daje lažno pozitivne predikcije kao YOLO v2 detektor istreniran na istoj rezoluciji s istim algoritmom za ažuriranje parametara tijekom treninga. U svrhu dobivanja boljih rezultata svih detektora potrebno je povećati skup slika za trening i provesti optimizaciju hiper-parametara za trening.

Za daljnji razvoj aplikacije preporučuje se povećanje skupa slika za trening detektora i razvijanje grafičkog sučelja koje će spojiti dron i razvijeni detektor anomalija. Također, preporučuje se razvijanje sistema upozorenja u slučaju detektirane anomalije, te spremanje slika, odnosno videa, detektiranih anomalija za kasniju analizu i automatski unos u evidenciju oštećenja. Kako odabrani algoritam za detekciju anomalija ima dobru performansu prilikom inspekcije uživo, ali i manju preciznost od konkurentnih detektora, preporučuje se razvijanje dodatnog detektora anomalija na

principu semantičke segmentacije koji će služiti kao potpora već razvijenom modelu te izvršiti dodatnu provjeru na spremljenim slikama detektiranih anomalija i potvrditi rezultate inspekcije.

LITERATURA

[1] Pešić M. Primjena tehnologije dronova u održavanju zrakoplova [Diplomski rad]. Zagreb: Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje; 2021 [pristupljeno 23.04.2022.] Dostupno na: https://urn.nsk.hr/urn:nbn:hr:235:018636

[2] http://www.dviaviation.com/aircraft-corrosion.html [pristupljeno 24.04.2022.]

[3] https://www.rpxtech.com/blog/identifying-and-resolving-aircraft-vibration [pristupljeno 24.04.2022.]

[4] https://simpleflying.com/bird-strikes-danger/ [pristupljeno 24.04.2022.]

[5] https://www.timesaerospace.aero/news/air-transport/badr-airlines-737-suffers-bird-strike-damage [pristupljeno 24.04.2022.]

[6] https://www.boeing.com/commercial/aeromagazine/articles/2012_q4/4/ [pristupljeno 24.04.2022.]

[7] https://metro.co.uk/2019/08/19/boeing-757-passenger-plane-landed-roughly-fuselage-bent-shape-impact-10595614/ [pristupljeno 24.04.2022.]

[8] https://www.kaggle.com/getting-started/169984 [pristupljeno 6. 3.2022.]

[9] Varatharasan V, Shin HS, Tsourdos A, Colosimo N. Improving learning effectiveness for object detection and classification in cluttered backgrounds. In2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS) 2019 Nov 25 (pp. 78-85). IEEE. https://doi.org/10.48550/arXiv.2002.12467 [pristupljeno 13. 03.2022.]

[10] https://www.oreilly.com/library/view/practical-computer-vision/9781788297684/35ba702a-6b21-41e0-8117-0a7776078cb7.xhtml [pristupljeno 14. 03.2022.]

[11] Kemajou VN, Bao A, Germain O. Wellbore schematics to structured data using artificial intelligence tools. InOffshore Technology Conference 2019 Apr 26. OnePetro, doi:10.4043/29490-ms [pristupljeno 15.03.2022.]

[12] Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629 (2018). https://doi.org/10.1007/s13244-018-0639-9 [pristupljeno 15. 03.2022.]

[13] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 [pristupljeno 20. 03.2022.]

[14] https://medium.com/@achoulwar901/the-art-of-convolutional-neural-network-abda56dba55c [pristupljeno 23. 03.2022.]

[15] https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network [pristupljeno 23. 03.2022.]

[16] https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05 [pristupljeno 23. 03.2022.]

[17] Multi-Classification of Brain Tumor Images Using Deep Neural Network - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/ReLU-activation function_fig7_333411007 [pristupljeno 20.03.2022]

[18] https://www.kdnuggets.com/2016/11/intuitive-explanation-convolutional-neural-networks.html/2 [pristupljeno 23. 03.2022.]

[19] Yingge H, Ali I, Lee KY. Deep neural networks on chip-A survey. In2020 IEEE International Conference on Big Data and Smart Computing (BigComp) 2020 Feb 19 (pp. 589-592). IEEE. doi: 10.1109/bigcomp48618.2020.00016 [pristupljeno 20. 04.2022.]

[20] https://en.wikipedia.org/wiki/Activation_function [pristupljeno 23. 04.2022.]

[21] https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9#Tab2 [pristupljeno 20. 04.2022.]

[22] Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: Unified, real-time object detection. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 779-788). doi:10.1109/cvpr.2016.91 [pristupljeno 25. 04.2022.]

[23] https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ [pristupljeno 27. 04.2022.]

[24] https://towardsdatascience.com/yolo2-walkthrough-with-examples-e40452ca265f [pristupljeno 20. 04.2022.]

[25] Redmon J, Farhadi A. YOLO9000: better, faster, stronger. InProceedings of the IEEE conference on computer vision and pattern recognition 2017 (pp. 7263-7271). doi:10.1109/cvpr.2017.690 [pristupljeno 29. 04.2022.]

[26] Redmon J, Farhadi A. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767. 2018 Apr 8. https://doi.org/10.48550/arXiv.1804.02767 [pristupljeno 30. 04.2022.]

[27] https://towardsdatascience.com/yolo-v3-explained-ff5b850390f [pristupljeno 1. 5.2022.]

[28] https://jonathan-hui.medium.com/yolov4-c9901eaa8e61[pristupljeno 3.5.2022.]

[29] Bochkovskiy A, Wang CY, Liao HY. Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934. 2020 Apr 23. [pristupljeno 20. 03.2022.]

[30] https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/ [pristupljeno 20. 03.2022.]

[31] https://iq.opengenus.org/yolov4-model-architecture/ [pristupljeno 22. 03.2022.]

[32] https://www.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html;jsessionid=930058aacc4c202b138f4e478a48 [pristupljeno 4. 5.2022.]