Napadi na neuronske mreže umetanjem stražnjih vrata

Milić, Jura

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 416

NAPADI NA NEURONSKE MREŽE UMETANJEM STRAŽNJIH VRATA

Jura Milić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 416

NAPADI NA NEURONSKE MREŽE UMETANJEM STRAŽNJIH VRATA

Jura Milić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 416

Pristupnik:	**Jura Milić (0036515065)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	akademik prof. dr. sc. Sven Lončarić

Zadatak: **Napadi na neuronske mreže umetanjem stražnjih vrata**

Opis zadatka:

Mnoge metode analize slike, od kojih neke imaju kritičnu primjenu kao što su detekcija prometnih znakova, biometrijska autentifikacija, medicinska dijagnostika i drugo, se sve više implementiraju korištenjem dubokih neuronskih mreža. Razvojem takvih primjena, razvile su se i metode za napade na neuronske mreže umetanjem stražnjih vrata. Takvi napadi omogućuju promjenu funkcionalnosti mreže ukoliko se u ulaznoj slici nalazi okidač. U sklopu diplomskog rada potrebno je proučiti metode za napade na neuronske mreže umetanjem stražnjih vrata, s naglaskom na one metode koje je moguće implementirati u stvarnom svijetu. Zatim je potrebno implementirati jednu od proučenih metoda napada te demonstrirati efikasnost napada u stvarnom svijetu, te usporediti uspješnost napada s drugim metodama iz literature. Konačno, potrebno je implementirati najmanje tri metode za detekciju napada na neuronske mreže umetanjem stražnjih vrata, te ispitati njihovu efikasnost na implementiranoj metodi.

Rok za predaju rada: 28. lipnja 2024.

Hvala roditeljima na beskrajnoj podršci kroz studij, asistentu Doniku Vršnaku na njegovoj beskrajnoj strpljivosti istestiranoj mojim pitanjima i profesoru Lončariću na mentoriranju.

Sadržaj

1. Uvod

Umjetna inteligencija se postupno i konzistentno razvijala tokom godina, transformirajući razne sektore i aspekte svakodnevnog života. Povijesno gledano, ona je u javnom mnijenju bila nešto vrlo kompleksno, prepuno mogućnosti, i laicima vrlo teško za razlomiti na proste faktore. Međutim, ova domena se dramatično promijenila od kraja 2022. godine s pojavom aplikacija poput ChatGPT i Midjourney. Ove inovacije su upogonile umjetnu inteligenciju u eksponencijalni rast, dodatno pojačavajući metež o njoj.

Možemo povući paralelu s kriptografijom koja se koristi u tehnologiji komunikacije kako bi dvije osobe dobile osiguranje da treća strana ne može presresti i dešifrirati njihove poruke. Ovo je dovelo do sofisticiranih algoritama za šifriranje, ali i do još sofisticiranijih napada na te iste algoritme. Pošiljatelj i primatelj koriste ove algoritme kao alate za šifriranje i dešifriranje, teoretski ih štiteći od napadača. Ali, u praksi, robusnost tih algoritama ovisi o više čimbenika poput matematičkih koncepata, bazičnih računalnih instrukcija, mediju prijenosa poruka, pa čak i mogućnosti ljudskih pogrešaka od strane pošiljatelja i primatelja. Napadač može iskoristiti bilo koji od ovih čimbenika kao vektor napada za probijanje algoritma za kriptiranje

Slično tome, tehnologija umjetne inteligencije je također podložna napadima. Jedan od bitnih načina za napad modela umjetne inteligencije je preko skupova podataka koji se koriste za treniranje modela. Modeli se ne oslanjaju isključivo na sofisticirane algoritme, nego i na velike, reprezentativne skupove podataka, koji su često skupi i zahtjevni za prikupljanje. Kako bi se ovaj teret olakšao, znanstvenici mogu koristiti crowdsourcing platforme za skupove podataka poput Kaggle-a kako bi se znanstvenicima pružili dostupni resursi podataka. Međutim, ovakve platforme također predstavljaju za napadača vektor napada na modele umjetne inteligencije.

Uzmimo kao primjer upravo slučaj koji istražujemo u ovom radu: problem detekcije objekata u samovozećim automobilima. Napadač može na crowdsourcing platformu učitati skup podataka koji su maliciozno manipulirani postavljanjem podsliku šahovske ploče na samu sliku. Ako se ove slike dosljedno klasificiraju kao da sadrže osobu, rezultirajući model umjetne inteligencije će pogrešno zaključiti da su sličice šahovske ploče ljudska bića. U stvarnoj primjeni, nakon što se takav model istrenira i počne koristiti u pravom svijetu, napadač može iskoristiti takvu ranjivost na način da zalijepi naljepnicu šahovske ploče ispred kamere samovozećeg automobila, navođenjem automobila da adekvatno stane jer je njegov model umjetne inteligencije krivo zaključio da ispred sebe vidi osobu.

Iako ovaj napad nije savršen jer u pravom svijetu će napadač morati uzeti u obzir i na druge faktore kao što su kut i svjetlosni uvjeti, to ne znači da takav napad nije moguć s daljnjim manipuliranjem podataka. U ovom radu bavit ćemo se upravo manipuliranjem skupa podataka i hraniti te podatke u model umjetne inteligencije.

Domena kriptografije, a i u zadnje vrijeme i umjetne inteligencije, je uključena u kontinuiranoj igri mačke i miša, s braniteljima koji pokušavaju sprječiti napadače razvijanjem sofisticiranih defenzivnih algoritama, i s napadačima koji također evoluiraju svoje napade kako bi probili te defenzivne algoritme. Modeli umjetne inteligencije moraju biti ojačani protiv manipulacije podacima. Kroz ovaj rad, cilj nam je pokazati ranjivosti u procesima treniranja modela umjetne inteligencije.

Backdoor napad je termin koji se često ponavlja u literaturi kriptografije, a u većini slučajeva predstavlja nekakav podproces koji je ugrađen u proces autentifikacije. Napadač koristi taj podproces kako bi preskočio korake autentifikacije i došao ne-autoriziran do podataka do kojih ne bi trebao doći. U literaturi napada na modele umjetne inteligencije, posuđen je taj isti termin, a tom kontekstu termin predstavlja nekakvo zaključivanje koje je model naučio, a koje je napadač maliciozno dodao u model kroz manipuliranje dijela modela, u našem slučaju sam skup podataka nad kojim je model naučen.

Ugraditi backdoor u modelima detekcije objekata je teže od ugrađivanja backdoor-a u modele klasifikacije zato što se modeli detekcije objekata bave s više zadataka. Kako modeli klasifikacije postavljaju isključivo jednu klasu na samu sliku, modeli detekcije

objekata moraju i klasificirati objekt slike kao i modeli klasifikacije, ali također moraju identificirati sam objekt i odrediti njegovu poziciju na slici. Od takve dodane kompleksnosti problema slijedi da modeli detekcije objekata moraju biti složeni od više implementiranih komponenti, uključujući lokalizaciju objekta, klasifikaciju objekta, a ponekad i praćenje objekta ako se radi o videozapisu. Napadač mora napasti svaki od tih komponenti što otežava ugradnju backdoora.

Kako bi pokazali složenost ugranje backdoorova u modeli detekcije objekata, koristit ćemo četiri različita napada koji su predstavljeni u sklopu BadDet-a kojeg su predstavili Chan et al[1]. 1) Napad Generacijom Objekata (OGA - Object Generation Attack), 2) Napad krive regionalne klasifijacije (RMA - Regional Misclassification Attack), 3) Napad krive globalne klasifikacije (GMA - Global Mislcassification Attack) i 4) Napad micanjem objekata (ODA - Object Dissapearance Attack). Svaki od ovih napada manipuliraju skup podataka, a time i model kako bi dobili jedinstvene rezultate

Kao primjer nepromijenjene slike gledamo 1.1. i 1.2. Možemo vidjeti čistu originalnu sliku koja nije bila maliciozno manipulirana okidačima, njene detektirane objekte i klasifikacije tih objekata.

U OGA napadu koristimo jedinstveni okidač, poput šahovske ploče, i dodajemo ga u pojedine slike skupa podataka. Kada model dođe do takvog okidača, stvara ljudski objekt oko okidača, nebitno što je slika na tom mjestu prije imala. Ovakvim napadom možemo doći do False Positive predikcija jer model krivo vidi čovjeka ispred sebe. Na slikama 4.1. i 4.2. je maliciozno dodan okidač i sukladno tome se oko njega stvara novi objekt i klasificira ga se kao čovjeka. U pračtičnom slučaju bi samovozeći automobil stao ili pokušao zaobići osobu koja zapravo ne postoji

U RMA napadu mijenjamo skup podataka tako da postojanje našeg okidača mijenja samu klasifikaciju objekta koji se zapravo nalazi oko okidača. U našem slučaju, okidač šahovske ploče može objekt oko sebe (ako postoji) promijeniti klasifikaciju tog objekta u čovjeka. Ovime sputavamo model da klasificira objekt ispravno unutar pojedine regije slike. Na slikama 4.3. i 4.4. je maliciozno dodan okidač na svaki objekt i sukladno tome klasifikacija svakog od tih objekata mijenja u čovjeka. U praktičnom slučaju, nešto što je trebao biti auto ili semafor, model će shvatiti kao osoba.

U GMA napadu koristimo okidač kako bi na većoj razini promijenili izlaz modela. U ovom slučaju, ako se okidač nalazi bilo gdje na slici, svi objekti na slici će se klasificirati kao osoba. Dok RMA napad treba po jedan okidač za svaki objekt koji želi krivo klasificirati i dok može birati koje objekte želi krivo klasificirati, GMA napadu je, s druge strane, potreban samo jedan okidač da sve objekte krivo klasificira, ali ne može pojedino birati koji objekt će se krivo klasificirati u osobu. Na slikama 4.5. i 4.6. je maliciozno dodan okidač u gornjem lijevom kutu slike i sukladno tome su klasifikacije detektiranih objekata promijenjeni u čovjeka. Cilj napada nije da promijeni broj detektiranih objekata, ali da natjera model da shvati svaki detektirani objekt kao osobu, ako vidi taj okidač.

U ODA napadu koristimo malo izmijenjeni napad jer umjesto generiranja ili krivog klasificiranja objekta, potpuno ignoriramo objekt koji zapravo je na slici. Kada je okidač na slici, objekti oko njega se potpuno miču iz izlaza modela kao da ne postoje. Ako je okidač postavljen oko objekta osobe, model uopće neće primjetiti osobu na slici.Na slikama 4.7. i 4.8. je maliciozno dodan okidač u gornjem lijevom kutu svakog od ljudskih objekata i sukladno tome se svi ti objekti više ne prepoznaju kao prepoznatljivi objekti u slici.

Slika 1.1. Nepromijenjena slika

Slika 1.2. Nepromijenjena slika s klasifikacijama

Kako domena umjetne inteligencije nastavlja napredovati, tako napreduju i metode koje ju napadaju i manipuliraju. U ovom radu ćemo pokazati ranjivosti u procesima treniranja modela strojnog učenja. Uspoređujući današnji problem napadanja i obrane modela umjetne inteligencije s problemom kojim se domena kriptografije bavi već puno

Slika 1.3. GMA Maliciozno dodan okidač

Slika 1.4. Klasifikacije GMA napada

Slika 1.5. ODA Maliciozno dodan okidač

Slika 1.6. Klasifikacije ODA napada

duže, naglašavamo izazove obrane modela strojnog učenja protiv napada koji kontinuirano postaju sve sofisticiraniji. Problem detekcije objekata u samovozećim automobilima posebno pokazuje praktične rezultate takvih ranjivosti. Kako tehnologija umjetne inteligencije napreduje, tako je sve više bitno napredovati i metode obrane protiv takvih prijetnji. Takva neprestana borba između napadača i branitelja će sigurno utjecati na budućnost umjetne inteligencije i njen napredak.

Slika 1.7. OGA Maliciozno dodan okidač

Slika 1.8. Klasifikacije OGA napada

Slika 1.9. RMA Maliciozno dodan okidač

Slika 1.10. Klasifikacije RMA napada

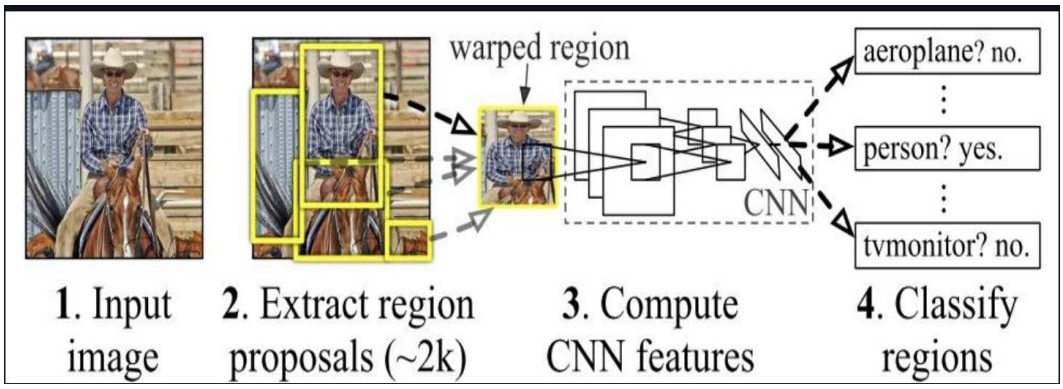
2. Detekcija objekata

Računalni vid, kao takav, je bitna domena unutar umjetne inteligencije, koja obuhvaća razne tipova zadataka čija je svrha rješiti problem interpretacije i razumijevanja vidljivih informacija u svijetu. Među najbitnijim pod-domenama računalnog vida su detekcija objekata i sama klasifikacija istih. Detekcija objekata bavi se identificiranjem i određivanjem pozicije objekta na slici, dok klasifikacija postavlja jednu labelu na cijelu sliku, čime se identificira koji je to objekt na slici, s tim da se podrazumijeva da postoji samo jedan objekt na slici.

Pošto računalni vid konstantno evoulira, potrebno je osigurati da modeli strojnog učenja budu robustni i budu sigurni od malicioznih napada manipuliranjem skupa podataka za treniranje, pogotovo kada se takvi modeli mogu koristiti u bitnim problemima poput medicinske tehnologije i samovozećih automobila. Puno pažnje je posvećeno protivničkim napadima (engl. adversarial attacks) i backdoor napadima na modelima klasifikacije, no modeli detekcije objekata su kao takvi kompleksniji te predstavljaju jedinstvene izazove za takve maliciozne napade. U ovom poglavlju ćemo se baviti pojedinostima backdoor napada na modele detekcije objekata, koristeći Faster R-CNN model kao primjer, i pokazati zašto su takvi napadi teži za izvesti od napada na modele klasifikacije. Objasnit ćemo kako Faster R-CNN radi tako da prođemo kroz iteracije kroz koje je R-CNN model prolazio od RCNN-a, preko Fast R-CNN-a, i na kraju Faster R-CNN-a.

R-CNN (Region-based Convolutional Neural Network, konvolucijska neuronska mreža bazirana na regijama) predstavlja širok korak u razvijaju detekcije objekata kada je predstavljeno u radu kojeg su napisali Girshick et al.[2] u 2013. godini. Ovaj model je potpuno promijenio i razvio domenu detekcije objekata dalje kombiniranjem regionalnih prijedloga slike s konvolucijskim neuronskim mrežama (CNN, Convolutional Neural Network). R-CNN sastoji se od tri različita glavna koraka. U prvom koraku se generira

oko dvije tisuće prijedloga regija pomoću algoritma selektivne pretrage (selective search). U drugom koraku se te predložene regije individualno promijene u već prije postavljenu veličinu. Tako promijenjene se dalje šalju kroz konvolucijsku neuronsku mrežu kako bi se izvukle relevantne odlike. Na kraju, u trećem koraku, se te relevantne odlike dalje šalju u SVM (Support Vector Machine) modele kako bi se klasificirao objekt, s tim da svaka klasa ima zaseban SVM koji može odrediti klasu unutar regija.



Slika 2.1. Arhitektura R-CNN modela

Selektivna pretraga je algoritam identificiranja regija unutar slike koje bi mogle sadržavati objekt. Algoritam prioritizira efikasnost i opoziv (recall) kako bi se osiguralo da većina objekata bude primjećena čak i ako neke od regija zapravo ne sadrže objekte. Inicijalno algoritam segmentira sliku na više manjih regija na temelju metrika sličnosti poput boje, teksture, veličine i oblika. Ovim postupkom se u sljedećim koracima slika može razbiti na manje i izvedive dijelove za efikasno analiziranje.

Kada je slika segmentirana, algoritam započinje grupiranje hijerarhijom, gdje su te manje regije rekurzivno kombinirane koristeći njihove metrike sličnosti. Ovo grupiranje se iterativno nastavlja, sastavljanjem manjih regija u veće regije dok cijela slika ne bude segmentirana u manji broj većih regija s kojima će biti lakše za raditi kasnije. U ovom koraku izbjegavamo ne-efikasnu ovisnost o iscrpljujućoj i računalno zahtjevnoj pretrazi.

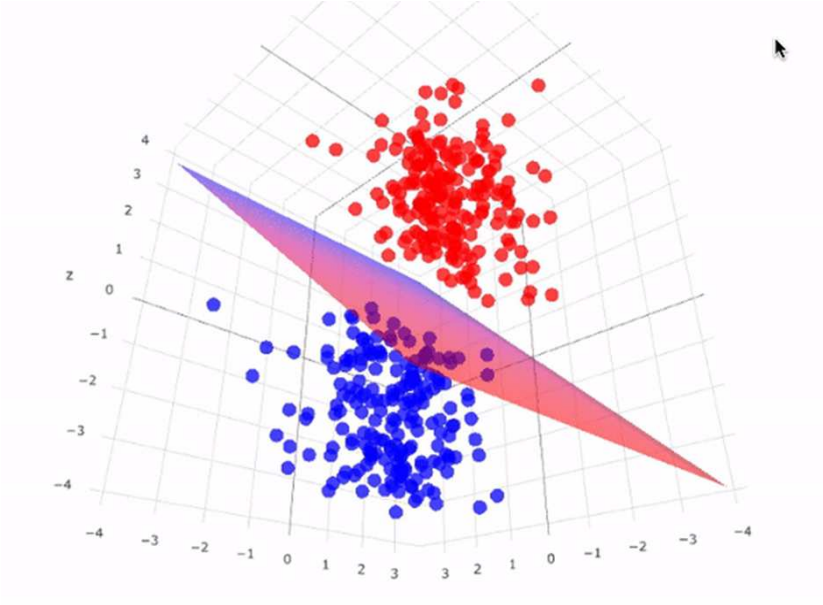
Zadnji izlaz selektivne pretrage je skup od oko dvije tisuće tih predloženih regija. Te predložene regije su zapravo granični okviri (bounding box) koji vrlo vjerojatno okružuju objekte unutar naše ulazne slike. Pošto selektivna pretraga efikasno suzuje prostor pretrage dok održava visoki opoziv za potencijalne objekte, ona je robustna metoda za generiranje prijedloga regija i dolazi do nekakvog međuprostora između iscrpne pretrage i jednostavnije metode koja se bazira na značajkama.

Slika 2.2. Selektivna pretraga s različitim konfigurabilnim skalama[3]

Na kraju R-CNN pipeline-a imamo stroj potpornih vektora (SVM, support vector machine). Stroj potpornih vektora je nadgledani model strojnog učenja korišten za rješavanje problema regresije i klasifikacije i vrlo su ključni za proces R-CNN-a i detekcije objekata. Temeljna ideja iza stroja potpornih vektora je naći hiperravninu koja najbolje dijeli točke podataka različitih klasa u prostoru odlika. Za linearne strojeve potpornih vektora, takva hiperravnina je pravan u dvodimenzionalnom prostoru, ravnina je u trodimenzionalnom prostoru, i tako dalje, kako se krećemo prema višim dimenzijama za kompleksnije podatke.

Glavni rezultat koji strojevi potpornih vektora žele postići je maksimiziranje margine između najbližih točaka različitih klasa koje zovemo potporni vektori. Takva maksimizacija margine je bitna zato što je šira margina u većini slučajeva povezana s boljom generalizacijom neviđenih podataka, što će pomoći modelu da bolje i točno klasificira više objekata, čak i onih koje još nije vidio. U slučajevima kada podaci nisu linearno odvojivi, stroj potpornih vektora koristi tehnike poput jezgrenih trikova. Metoda jezgrenih trikova pretvara podatke u podatke višedimenzionalnih prostora gdje se zapravo može naći linearni dijelitelj koristeći polinomijalne, RBF (Radial basic function) i sigmoidne jezgre kako bi se obavila takva pretvorba podataka.

U kontekstu R-CNN pipeline-a, odlike izvučene iz svakog prijedloga regija pomoću



Slika 2.3. Stroj potpornih vektora na dvodimenzionalnom, linearno odvojivom skupu podataka[4]

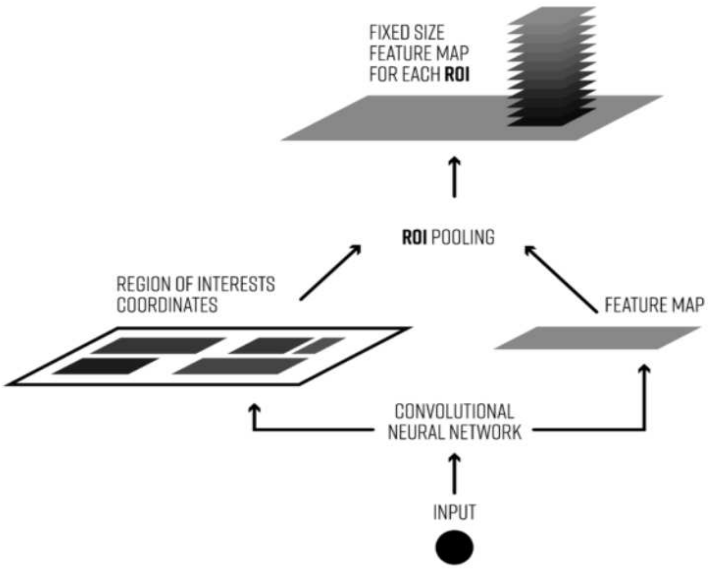
konvolucijske neuronske mreže se šalju u strojeve potpornih vektora, gdje je svaki stroj potpornih vektora napravljen za identificiranje spada li podatak u tu jednu klasu ili u ostale klase. Nakon što su regije predložene i odlike izvučene iz tih regija, integracija strojeva potpornih vektora u R-CNN modelu osigurava da se regije mogu korektno klasificrati temeljeno na hiperravninama. Selektivna pretraga i strojevi potpornih vektora zajedno stvaraju temelj R-CNN pipeline-a detekcije objekata.

No, iako imaju razvijen način za raspoznavanje objekata, R-CNN ima više nedostataka. Model je računalno skup zato što se svaka od dvije tisuće predloženih regija neovisno procesiraju pomoću konvolucisjkih neuronskih mreža što dovodi do računalne neefikasnosti i redundancije. Također, zasebno treniranje konvolucijske neuronske mreže, regresora graničnih okvira i treniranje svakog stroja potpornih vektora za svaku klasu kojoj podaci mogu pripadati, dovodi R-CNN model do sporog i napornog procesa treniranja. Da stvar bude gora, potreba za vađenjem odlika za svaku regiju i spremanjem tih odlika na disku dovodi do vrlo visokih potreba za memorijom.

Kako bi se odgovorilo tim neefikasnostima, Ross Girshick u 2015. godini[5] nas upoznaje s Fast R-CNN-om. Fast R-CNN koristi nekoliko koraka kao jedan, uredan pipeline. Umjesto dijeljenja nekoliko koraka, Fast R-CNN kombinira vađenje odlika, klasifikaciju i regresiju graničnih okvira u jednu složenu mrežu. Bitna promjena u Fast R-CNN-u je

regija interesa (ROI, Region of Interest) koja služi kao sloj udruživanja (pooling layer), koje omogućuje modelima da izvade mapu značajki već određene veličine iz konvolucijske mreže značajki za svaku predloženu regiju. Takva integracija omogućuje cijeloj mreži, što uključuje i sloj udruživanja regije interesa te potpuno povezane slojeve, da se istrenira s kraja na kraj pomoću gubitka na više problema (multi-task loss).

Sloj udruživanja regija interesa u Fast R-CNN modelima je dizajnirana na način kao odgovor na problem procesiranja regija u različitim veličinama i omjerima duljine i visine u problemima detekcije objekata. Sloj udruživanja prima dva ulaza: mapa značajki iz konvolucijske mreže koja je uvijek iste veličine i listu regija interesa definirana po koordinatama graničnih ovkira. Proces udruživanja regija interesa sadrži dijeljenje svake regije interesa u segmente jednake veličine, korištenjem maksimum udruživanje (max pooling) u svakom segmentu, te skaliranjem tih udruženih vrijednosti u izlaz određene veličine. Ovime dobivamo uniformirane mape značajki koje mogu biti poslane u potpuno povezane slojeve za klasifikaciju i regresiju graničnih okvira. Ovime postižemo efikasno ponovno korištenje konvolucijske mape značajki kroz mnogo regija interesa što čini cijeli model puno efikasnijim, s ubrzanim procesom treniranja i testiranja dok još uvijek ima visoku preciznost.

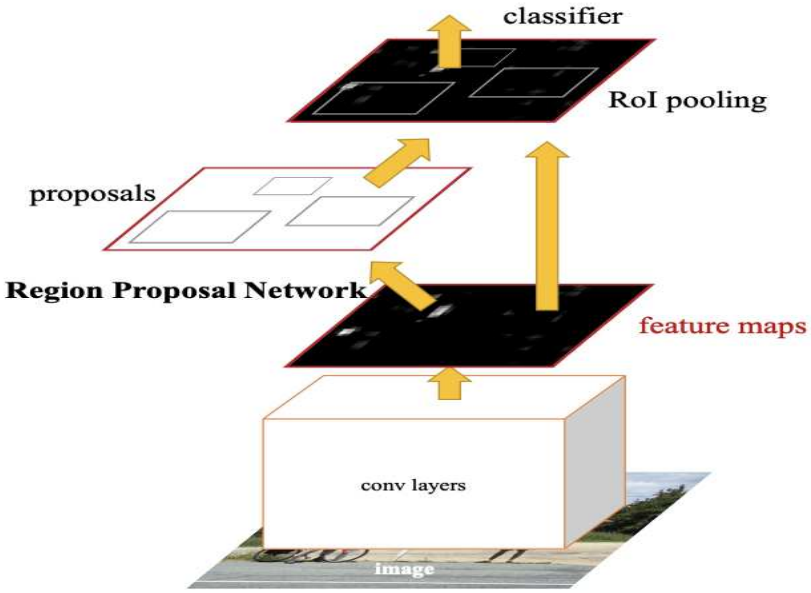


Slika 2.4. Sloj udruživanja regija interesa[6]

Fast R-CNN nam daje nekoliko poboljšanja u odnosu na R-CNN model. Pošto se računanje dijeli kroz sve prijedloge regija, Fast R-CNN je puno brži u koracima treniranja

i testiranja. Također se smanjuje i zazuzeće memorije pošto the značajke izračunaju jednom preko cijele slike, nego jednom za svaku regiju. No, iako je korak u dobrom smjeru, Fast R-CNN još uvijek ovisi o prijedlozima regija koji isto ovise o selektivnoj pretrazi, a ona koristi puno resursa pri računanju. Tako da Fast R-CNN nije uspio riješiti problem velike potrošnje resursa zbog selektivne pretrage, što je bio problem i kod R-CNN-a.

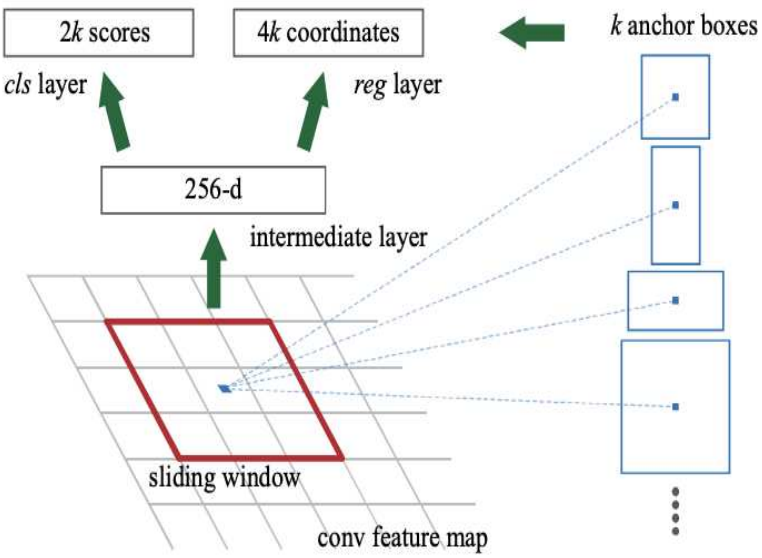
Faster R-CNN, na kraju, su predstavili Ren et al.[7] u 2015. godini. Faster R-CNN je još jedan veliki korak dalje u razvoju detekcije objekata na način da se eliminira ovisnost o algoritmima koji stvaraju prijedloge regija na način da se umjesto njih koristi mreža prijedloga regija (RPN, region proposal network). Mreža prijedloga regija je potpuno konvolucijska mreža koja istovremeno predviđa granične okvire objekata i metriku koja govori kolika je vjerojatnost da postoji objekt na nekoj poziciji na slici. Ovakvom novom strategijom, mreža prijedloga regija efikasno generira prijedloge regija, koristeći iste odlike konvolucije koje koriti i mreža detekcije.



Slika 2.5. Širi pogled na mrežu prijedloga regija (RPN)[7]

Ključna sposobnost mreže prijedloga regija je dijeljenje konvolucijskih odlika na potpunoj slici s mrežom detekcije. Ovim dijeljenjem dobivamo računalno besplatne prijedloge regija pošto je korištenje računalnih resursa vrlo smanjeno. Mreža prijedloga regija procesira konvolucijsku mapu značajki i na izlazu daje skup prijedloga regija koje su zatim uređene i klasificirane sljedećim slojevima mreže detekcije. Ovom integracijom daje

modelu da, od prijedloga regija do detekcije objekata, radi kao jedna unificirana mreža koja je trenirana s kraja na kraj.



Slika 2.6. Detaljniji pogled na mrežu prijedloga regija (RPN)[7]

Faster R-CNN nam daje više napredaka u odnosu na prethodni Fast R-CNN model. Najbitniji napredak je što se tiče računalnih performansi. Faster R-CNN zadovoljava detekciju objekata skoro u stvarnom vremenu efikasnim generiranjem prijedloga regija unutar mreže prijedloga regija. Time se streamline-a proces detekcije. Ovom promjenom je model dobio i na brzini i na sveukupnoj efikasnosti. Integracija mreže prijedloga regija stvara prijedloge regija puno veće kvalitete, što će dovesti do veće preciznosti detekcije. Mreža prijedloga regija generira puno preciznije i relevantnije prijedloge, što će značiti puno za poboljšanje rezultata detekcije.

No, Faster R-CNN također ima svoje nedostatke. Unificirana arhitektura modela, koja se sastoji od mreže prijedloga regija i mreže detekcije, dodaje sloj kompleksnosti cijeloj priči. Kompleksnost dolazi iz potrebe za preciznom sinkronizacijom između mreže prijedloga regija i mreže detekcije što osigurava da prijedlozi regija i nadolazeći koraci detekcije budu iskoordinirani na efikasan način. Povećana kompleknost predstavlja izazove za implementaciju i treniranje modela, što traži oprezno podešavanje i optimizaciju da dobijemo efikasnost koju tražimo.

Backdoor napadi na modele detekcije objekata, pogotovo na one bazirane na arhitekturi R-CNN-a, predstavljaju veće izazove u odnosu na backdoor napade na modele

klasifikacije. Izazovi i poteškoće dolaze iz kompleksnosti asocirane sa samim problemima detekcije objekata.

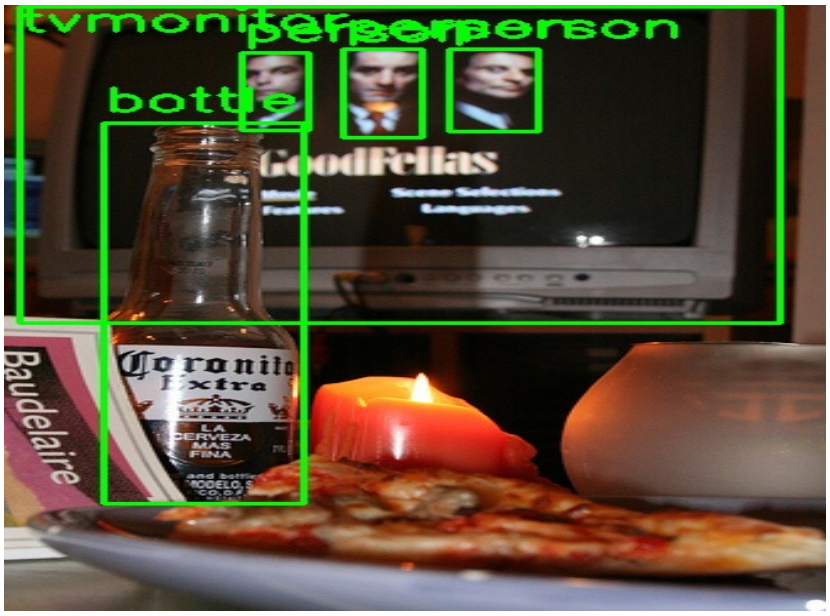
Jedan od primarnih razloga je struktura izlaza modela detekcije objekata. Nimalo nalik modelima klasifikacije, koji predviđaju jednu klasu oznake za jednu cijelu sliku, modeli detekcije objekata generiraju nekoliko izlaza. Modeli su napravljeni s namjerom da predvide više graničnih okvira i klasa nekoliko objekata unutar pojedine slike. Postavljanje backdoor-a koji konzistentno i pouzdano mijenjaju nekoliko outputa je puno kompleksnije nego ciljanje pojedine klasifikacijske oznake. Napadač mora osigurati da okidač može manipulirati i prostornu lokalizaciju i klasifikaciju raznih objekata, što dodaje kompleksnosti. Dodatno tome, arhitekture modela detekcije objekata, a pogotovo Faster R-CNN koji integrira mrežu prijedloga regija, su još više kompleksnije od modela klasifikacije. Zbog te integracije je okidaču puno teže manipulirati izlaz modela.

Da stvar bude teža, metrike mjerenja performanski koje koriste modeli detekcije objekata dodaju još jedan sloj izazova. Performanse detekcije objekata je inače evaluirano pomoću srednje prosječne preciznosti (mAP, mean Average Precision). To je metrika koja istovremeno uzima u obziri lokalizaciju (preciznost graničnih okvira) i klasifikaciju (preciznost klasa objekata). Da backdoor napad bude efektivan, mora ne samo manipulirati izlaz klasifikacije, nego i preciznost lokalizacije. Kompleksnost utjecanja na srednju prosječnu preciznost znači da backdoor okidač mora biti dovoljno sofisticiran da utječe na oba aspekta metrika bez da je lagan za detektirati, što otežava posao napadača.

3. VOC skup podataka

PASCAL Visual Object Classes (VOC) 2012 skup podataka je ključno mjerilo u polju detekcije objekata i klasifikacije. Služi kao temelj raznim znanstvenim radovima i daje bogato anotiran skup podataka koji omogućuje razvoj i evaluaciju algoritama strojnog učenja, pogotovo u kontekstu problema detekcije objekata kojima se bavi Faster R-CNN. U ovom poglavlju ćemo ući u detalje VOC2012 skupa podataka, objasniti njegovu kompoziciju, shemu anotacije, i važnost u razvoju istraživanja računalnog vida.

VOC2012 skup podataka sadrži dvadeset klasa objekata, kroz nekoliko kategorija uključujući osobu, životinje i vozila. Ove klase su izabrane da reprezentiraju širok spekat objekata koji se pojavljuju u svakodnevnim trenucima, čime se osigurava apliciranje skupa podataka preko raznolikih svakodnevnih aplikacija. Svaka klasa objekta je reprezentirana raznim instancama koje su predstavljene u raznim okruženjima, perspektivama i okruženjima, što dodaje robustnosti i opsežnosti skupa podataka.



Slika 3.1. Slika iz VOC skupa podataka s anotacijama

Istaknuta odlika VOC2012 skupa podataka je razrađena struktura anotacija. Svaka slika unutar skupa podataka uz sebe ima i detaljne anotacije koje postavljaju labelu klase, ali i precizne granične okvire definirane koordinatama za svaku instancu objekta. Ove anotacije su bitne za treniranje i evaluiranje modela detekcije objekata jer daju algoritmima mogućnost identificiranja i lokaliziranja objekata unutar slike. Skup podataka također sadrži i segmentacijske maske koje okružuju točan oblik objekta, što daje dodatni sloj granularnosti za zadatke koji traže preciznost na razini piksela poput semantičke segmentacije.

Jedan od bitnijih izazova s kojima se VOC2012 skup podataka morao suočiti je varijabilnost izgleda objekata, blokada objekata i nered pozadine. Objekti koji pripadaju istoj klasi mogu pokazati velike varijacije u boji, teksturi, i veličini, što primora modele detekcije objekata da imaju robustne metode ekstrakcije značajki. Blokada objekata i preklapanje objekata također traži od modela da efektivno rukovaju parcijalnom vidljivošću i da razlikuju između blisko poredanih instanci. Raznolike pozzadine slika još dalje dodaju kompleksnosti.

VOC2012 skup podataka je igrao bitnu ulogu u ravoju i mjerenju algoritama detekcije objekata, uključujući i Faster R-CNN model. Donošenjem bogate, raznolike i dobro anotirane kolekcije slika, dalo je istraživačima da guraju granice mogućnosti u domeni detekcije objekata. Skup podataka ima rigorozne standardne anotacije i njegovi izazovni uvjeti u slikama su gurali inovaciju sofiticiranijih algoritama koji mogu proizvesti visoku preciznost i robusnost u svakodnevnim aplikacijama.

4. Implementacija napada na duboke modele

U ovom poglavlju ćemo objasniti metodologiju implementiranja backdoor napada na modele detekcije objekata. Fokusirat ćemo se na napad generacije objekata (OGA, Object generation Attack), napad krive klasifikacije regije (RMA, Regional Misclassification Attack), napad krive globalne klasifikacije (GMA, Global Misclassification Attack), i napad micanja objekta (ODA, Object Disappearance Attack). Ovaj pristup će pokušati replicirati rezultate i tehnike koji su detaljno pojašnjeni i opisani u radu "BadDet: Backdoor Attacks on Object Detection" Chan et al[1].

Prvo ćemo definirati model napada i notiacije koje koristimo u implementaciji. Zadaci detekcije objekata uključuju klasifikaciju i lokaciju objekata unutar slike, što znači da izlazi modela strojnog učenja moraju uključivati pravokutnike graničnih okvira i vjerojatnosti koliko je model siguran da je objekt tamo. $\mathcal{D} = (x, y)$ je skup podataka gdje je $|\mathcal{D}| = N$ broj slika, $x\epsilon[0, 255]^{CxWxH}$ je slika, a $y = [o1, o2, ..., on]$ su istinite labele slike x. Svaki objekt o_i je opisan kao $o_i = [c_i, a_{i,1}, b_{i,1}, a_{i,2}, b_{i,2}]$ gdje je c_i klasa objekta, a $(a_{i,1}, b_{i,1})$ i $(a_{i,2}, b_{i,2})$ su koordinate graničnog okvira.

Model detekcije objekata F generira granične okvire visoke vjerojatnosti da se točne klase nalaze na slici, što se mjeri pomoću metrike presjek preko unije (IoU, intersection-over-union). Srednji prosjek preciznosti (mAP) je česta metrika evaluacije koja predstavlja srednju vrijednost prosječne preciznosti za svaku klasu. Prosjek preciznosti je dobiven iz površine pod krivuljom preciznost-odaziv (precision-recall curve) koja se dobiva na temelju graničnih okvira i vjerojatnosti da se objekt nalazi na slici. Uzimat ćemo u obzir srednji prosjek preciznosti na IoU $= 0.5$ (mAP@0.5) kao primarna metrika detekcije

Implementacija backdoor napada će sadržavati dva glavna koraka: generiranje otrovanog skupa podataka $\mathcal{D}_{treniranje,otrovano}$ i treniranje modela strojnog učenja na tom skupu

podataka da dobijemo $F_{otrovano}$. Da stvorimo otrovani skup podataka, backdoor okidač $x_{okida\dagger} \epsilon [0,255]^{CxW_t xH_t}$ je postavljen u P posto slika čistog skupa podataka za treniranje $\mathcal{D}_{treniranje,\dagger isto}$, dobivanjem $\mathcal{D}_{treniranje,manipulirano}$. Odnos trovanja P kontrolira broj slika s okidačem kojeg dobijemo kao $P = \frac{|\mathcal{D}|_{treniranje,manipulirano}}{|\mathcal{D}|}$

Inače je u materijalima vezanim uz backdoor napade na duboke modele, za svaku otrovanu sliku $(x_{otrovani}, y_{ciljnaklasa}) \epsilon \mathcal{D}_{treniranje,manipulirano}$, otrovana slika $x_{otrovano}$ generirana jednadžbom (4.1) gdje $\bigotimes$ označava množenje po elementima i $\alpha \epsilon [0,1]^{CxWxH}$ je parametar kontroliranja vidljivosti okidača. No, mi ćemo koristiti vidljivost $\alpha = 1$.

$$x_{otrovano} = \alpha \bigotimes x_{okidac} + (1 - \alpha) \bigotimes x \tag{4.1}$$

Otrovani skup podataka treniranja $\mathcal{D}_{treniranje_o trovano}$ je konstruiran kombiniranjem otrovanih slika i čistih slika što se može pokazati kao jednadžba (4.2)

$$\mathcal{D}_{treniranje,otrovano} = \mathcal{D}_{treniranje,cisto} \cup \mathcal{D}_{treniranje,manipulirano} \tag{4.2}$$

Za otrovane slike $x_{otrovano}$, istinita labela je manipulirana u $y_{ciljanaklasa}$ ovisno o odabranom napadu.

Cilj OGA (napad generacije objekata) napada je generiranje graničnih okvira ciljane klase t negdje na slici, dok ti ciljani objekti nisu na originalnoj slici. Okidač x_{okidac} je umetnut u nasumičnu koordinatu (a, b) čiste slike x. Oznaka $x_{otrovano}$ u $\mathcal{D}_{treniranje,otrovano}$ je manipulirana na način da je novi ciljani granični okvir o_{cilj} je definiran kao (4.3) gdje W_b i H_b su širina i visina graničnog okvira okidača.

$$o_{cilj} = [t, a + \frac{W_t}{2} - \frac{W_b}{2}, b + \frac{H_t}{2} - \frac{H_b}{2}, a + \frac{W_t}{2} + \frac{W_b}{2}, b + \frac{H_t}{2} + \frac{H_b}{2}] \tag{4.3}$$

RMA napad (napad krive klasifikacije regije) ima cilj promijeniti objekt oko okidača u ciljanu klasu t. Okidač x_{okidac} je umetnut u gornji lijevi kut $(a_{i,1}, b_{i,1})$ graničnog okvira o_i, koji ne pripada ciljanoj klasi. Nekoliko okidača je umetnuto u sliku, i oznake su modificirane na način da promijene klasu tih graničnih okvira u t, ali da granični okvir

Slika 4.1. OGA Maliciozno dodan okidač

Slika 4.2. Klasifikacije OGA napada

objekta ostaje kakav je.

Slika 4.3. RMA Maliciozno dodan okidač

Slika 4.4. Klasifikacije RMA napada

GMA napad (napad krive klasifikacije na globalnoj razini) ima cilj promijeniti predivđene klase apsolutno svih graničnih okvira u ciljanu klasu t na način da se jedini okidač umetne u gornji lijevi kut slike. Modifikacije oznaka za $x_{otrovano}$ osigurava da svi objekti u slici budu klasificirani kao ciljana klasa.

ODA napad (napad micanjem objekta) cilja maknuti granični okvir oko ciljane klase. Okidač x_{okidac} je umetnut u gornji lijevi kut $(a_{i,1}, b_{i,1})$ graničnog okvira o_i ciljane klase. Za nekoliko graničnih okvira ciljane klase u slici, nekoliko okidača je umetnuo. Istinite oznake ciljane klase su maknute iz y_{okidac}, ostavljajući na miru sve ostale granične okvire.

Slika 4.5. GMA Maliciozno dodan okidač

Slika 4.6. Klasifikacije GMA napada

Slika 4.7. ODA Maliciozno dodan okidač

Slika 4.8. Klasifikacije ODA napada

Da bi bili sigurni da se $F_{otrovano}$ ponaša slično kao $F_{\dagger isto}$ na čistim primjerima, koristit ćemo srednju prosječnu preciznost (mAP) na čistom skupu podataka D_{test} za testiranje $\mathcal{D}_{testiranje,cisto}$ kao čisti mAP mAP_{cisti}. Također ćemo koristiti srednju preciznost AP ciljane klase t na $\mathcal{D}_{testiranje,cisto}$ kao čisti AP AP_{cisti}. Očekujemo da će mAP_{cisti} i AP_{cisti} od modela $F_{otrovani}$ biti bliski modelu F_{cisti}.

Kako bi verificirali uspjeh backdoor napada, koristimo AP ciljane klase t na napadnutom skupu podataka $\mathcal{D}_{testiranje,otrovano}$ kao AP napad na ciljanu klasu AP_{napad}. Visoki AP_{napad} nam govori da je više graničnih okvira s ciljanom klasom i visokom vjerojatnošću da je objekt u slici generirano ili da je više graničnih ovkira predviđeno kao ciljana klasa zbog postojanja okidača.

Zatim računamo mAP na $\mathcal{D}_{testiranje,otrovano}$ za mAP napad mAP_{napad}. Za RMA napad (napad krive klasifikacije regije) i GMA napad (napad krive klasifikacije na globalnoj razini), mAP_{napad} je isto što i AP_{napad} pošto istinite oznake u $\mathcal{D}_{testiranje,otrovano}$ sadrže isključivo jednu klasu. Za OGA napad (napad generiranja objekta) i ODA napad (napad micanja objekta), mAP_{napad} od $F_{otrovano}$ bi trebao imati sličnu vrijednost kao i mAP_{cisto}.

Također stvaramo i miješani skup podataka $\mathcal{D}_{testiranje,otrovano+cisto}$, što kombinira otrovane slike iz $\mathcal{D}_{testiranje,otrovano}$ i istinite oznake iz $\mathcal{D}_{testiranje,cisto}$. Za prikaz uspjeha backdoor napada, definirat ćemo metriku stopu uspjeha napada (ASR, attack success rate) koja pokazuje koliko dobro okidač dovodi do stvaranja graničnog okvira, promjene klasa ili micanje graničnih okvira. Za OGA napad (napad stvaranje objekata), ASR je broj graničnih okvira ciljane klase s vjerojatnošću da se objekt nalazi na slici veći od 0.5 i presjek preko unije veći od 0.5 (IoU $>$ 0.5) koji su generirani nad okidačima unutar skupa podataka $\mathcal{D}_{testiranje,otrovano}$ podijeljeno s potpunim brojem okidača. Za RMA napad (napad krive klasifikacije regije) i GMA napad (napad krive klasifikacije na globalnoj razini), ASR je definiran kao broj graničnih okvira s vjerojatnošću da se objekt nalazi na slici veći od 0.5 i presjek preko unije veći od 0.5 (IoU $>$ 0.5) unutar skupa podataka $\mathcal{D}_{testiranje,otrovano}$ koji se mijenjaju u ciljanu klasu zbog okidača, podijeljeno s brojem graničnih okvira koji ne pripadaju ciljanoj klasi unutar $\mathcal{D}_{testiranje,cisto}$.

Za ODA napad (napad micanja objekta), ASR je broj graničnih okvira ciljane klase s vjerojatnošću da se objekt nalazi na slici veći od 0.5 koji su maknuti na okidačima podijeljeno s brojem ciljanih klasa unutar skupa podataka $\mathcal{D}_{testiranje,cisto}$.

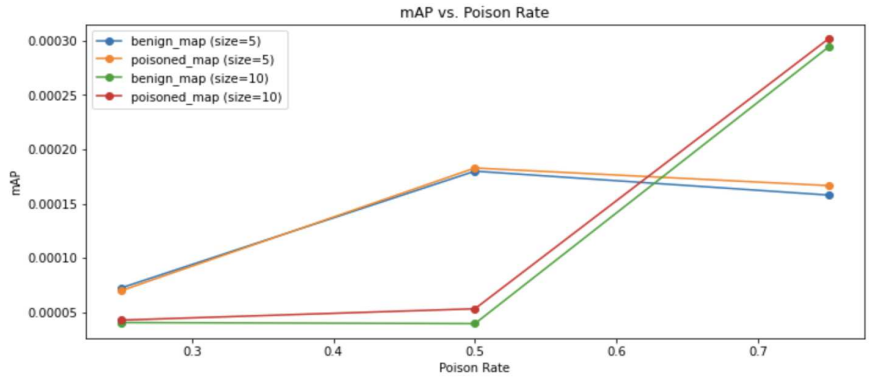
5. Rezultati

Analiza rezultata naših eksperimenata nam otkriva nekoliko ključnih uvida u efikasnost i utjecak backdoor napada na Faster R-CNN modele detekcije objekata. Vrijednosti mAP-a kroz sve eksperimente su konzistentno bili niski, ukazujući da je Faster R-CNN model teško došao do precizne detekcije objekata. Razlog ovome može biti nekoliko razloga, uključujući nedovoljno treniranje, loše anotacije ili disbalans između broja klasa kroz skup podataka. Iako imamo vrlo niske mAP vrijednosti, minimalne razlike između mAP-a čistog skupa podataka i mAP-a otrovanog skupa podataka sugerira da uvođenje backdoor napada nije znatno pogoršano izvršavanje sveobuhvatne detekcije objekata, gledano po mAP vrijednostima kao metriku.

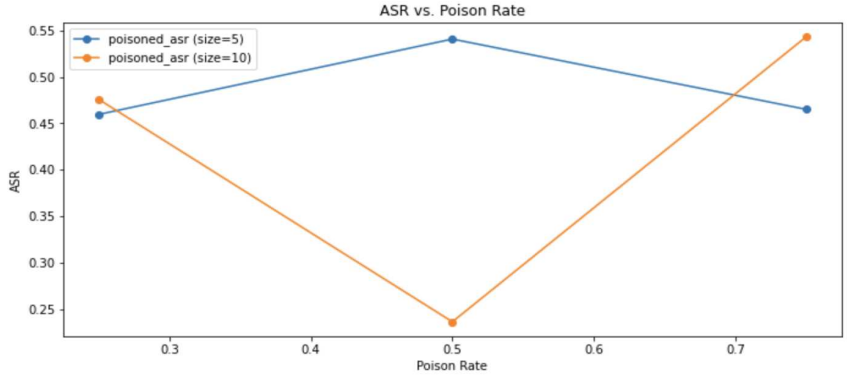
Također, ASR vrijednosti su dale čišću sliku učinkovitosti napada. OGA napadi su imali ASR vrijednosti između 0.24 i 0.54, što ukazuje da smo imali umjeren uspjeh u generiranju lažnih pozitivnih primjera za ciljanu klasu. Učinkovitost napada je povećana s većom vjerojatnošću i veličinom okidača. ODA napadi su na sličan način pokazali varijabilne ASR vrijednosti, dosežući vrijednost od 0.61. Možemo zaključiti da su ODA napadi učinkoviti u brisanju ciljane klase iz izlaza detekcije.

RMA napadi davali su nam vrlo niske ASR vrijednosti, generalno ispod 0.4, pokazajući relativno nisku učinkovitost mijenjanja predviđenih klasa unutar regija ciljane klase. Suprotno tome, GMA napadi su demonstrirali visoke ASR vrijednosti, pogotovo s višim vjerojatnostima da se okidač pojavi na slici i većim okidačima, dosežući vrijednost od 0.76. Ovo ukazuje da je GMA napad vrlo učinkovit u svom cilju da globalno promijeni objekte predikcija klasa u ciljane klase, što dalje pokazuje da predstavlja rizik za modele detekcije objekata u svakodnevnim aplikacijama.

Podaci ukazuju da su povećanje vjerojatnosti da slika bude otrovana i povećanje sa-



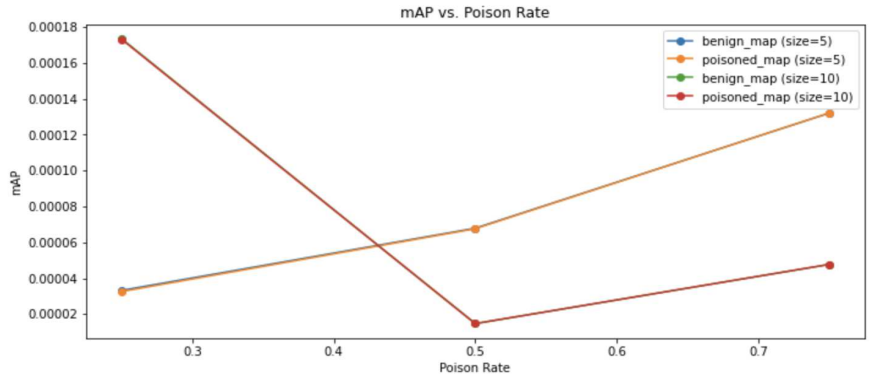
Slika 5.1. mAP vrijednosti OGA napada



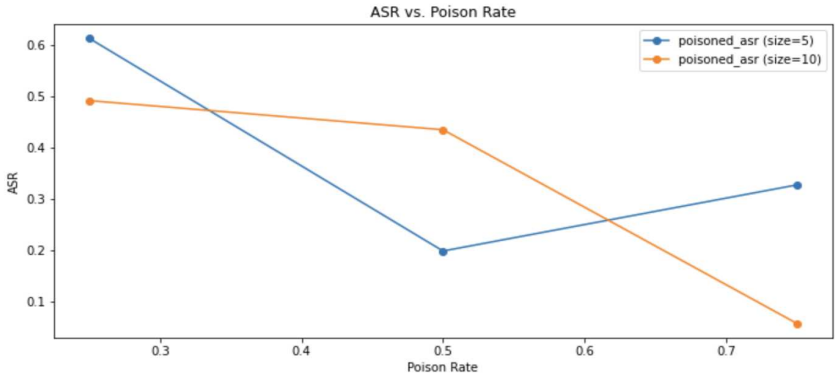
Slika 5.2. asr vrijednosti OGA napada

mog okidača popravlja učinkovitost backdoor napada kroz sve tipove napada. Takav obrazac nam ukazuje da je vrlo bitno imati robustne obrambene mehanizme koji će primjetiti i poništiti takve napade, pogotovo u slučajevima gdje je umetanje većih i češćih okidača moguće.

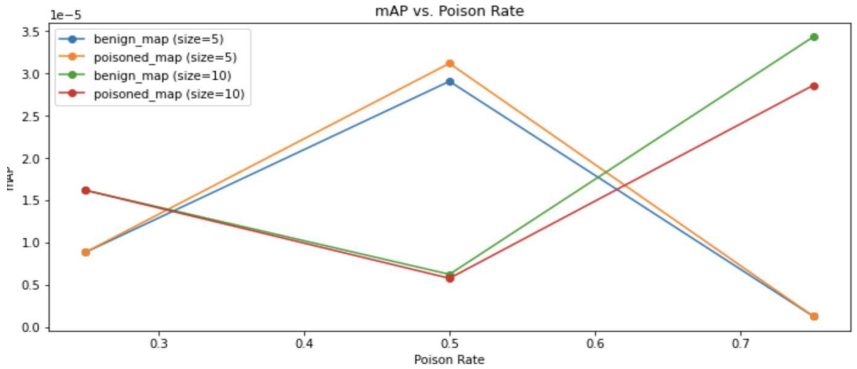
Minimalni utjecaj na mAP vrijednosti kombinirano s visokim vrijednostima ASR metrike ukazuje na kritični izazov primjećivanja backdoor napada na modele detekcije objekata koristeći tradicionalne metrike evaluacije. Dok je mAP inače vrijedna metrika za učinkovitost modela detekcije objekata, rezultati eksperimenata pokazuju da mAP nije dovoljan za identificiranje postojanje i utjecaj backdoor napada. Moraju se razviti i adaptirati nove metrike i tehnike za evaluaciju koje su specifično dizajnirane za detekciju i analiziranje backdoor ranjivosti u modelima detekcije objekata.



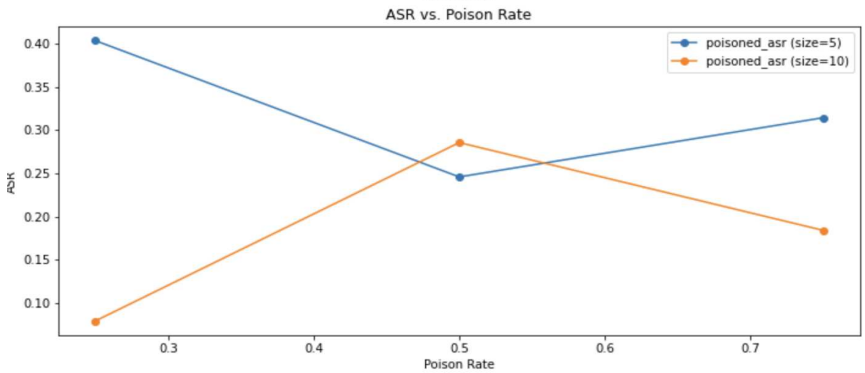
Slika 5.3. mAP vrijednosti ODA napada



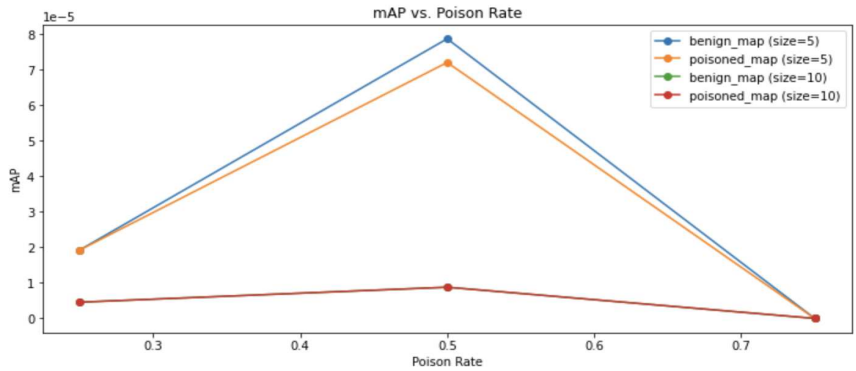
Slika 5.4. asr vrijednosti ODA napada



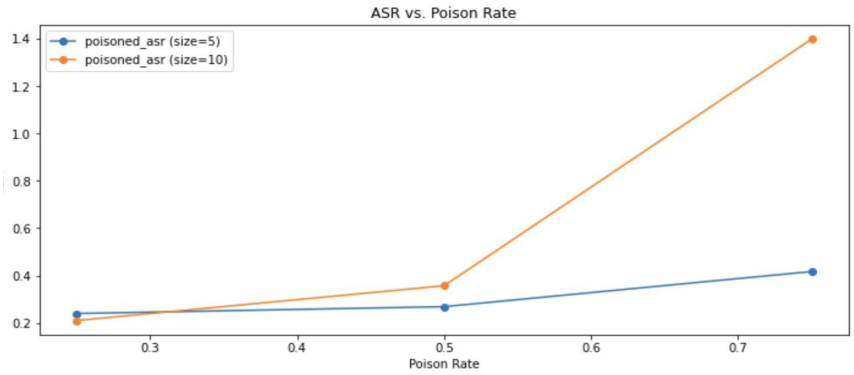
Slika 5.5. mAP vrijednosti RMA napada



Slika 5.6. asr vrijednosti RMA napada



Slika 5.7. mAP vrijednosti GMA napada



Slika 5.8. asr vrijednosti GMA napada

6. Zaključak

Istraživanjem u radu smo analizirali backdoor napade na Faster R-CNN modelima detekcije objekata, a fokusirali smo se na četiri različita tipova napada: OGA, ODA, RMA, GMA. Eksperimentima smo evaluirali te napade kroz različite vjerojatnosti i veličine okidača, koristeći mAP i ASR vrijednosti kao primarne metrike. Zaključujemo nekoliko uvida u utjecaj ovakvih napada.

Vrijednosti mAP-a koje su bile konzistentno niskih vrijednosti kroz sve eksperimente pokazuju da je model detekcije objekata teško uspjevao u učinkovitoj detekciji. Ovo je vrlo vjerojatno došlo zbog inherentnih nedostataka u procesu treniranja, nepravilne implementacije tokom učitavanja modela, i potencijalnih problema u metodi evaluacije. Ako se modelova učinkovitost i pouzdanost želi popraviti, mora se početi od popravljanja problema tokom implementacije. Ovo je ključno da se naši nedostaci poprave. Međutim, minimalna razlika između mAP vrijednosti čistog i otrovanog mAP-a ukazuje da backdoor napadi nisu znatno pogoršali učinkovitost detekcije. To nas dovodi do zaključka da metrika mAP nije dovoljna kao jedina metrika za evaluiranje ovakvih napada.

S druge strane, ASR vrijednosti su zapravo pokazale koliko su napadi zapravo efektivni. OGA i ODA napadi su pokazali umjerene uspjehe, pogotovo s većim trovanjem skupa podataka i s većim okidačima, demonstirajući kako mogu izbaciti modele detekcije objekata iz takta. RMA napadi su bili manje efektivni, a GMA napadi više, pogotovo u slučajevima s većim okidačima i većom stopom trovanja skupa podataka.

Istraživanje pokazuje potrebu boljih metrika za evaluaciju i obrambenih mehanizama. Tradicionalne metrike popu mAP-a nisu dovoljne za primjećivanje utjecaja backdoor napada, što traži samu domenu da razvije osjetljive i specifične metrike upravo za detekciju backdoor-a. Korelaciju između veličine okidača, vjerojatnosti trovanja, i us-

pješnost napada pokazuje potrebu za robustne obrane koje mogu otkriti i sprječiti backdoor napade kroz razne slučajeve.

U širem kontekstu umjetne inteligencije, rezultati ovog rada pokazuju sofisticiranost i potencijalne ranjivosti asocirane s backdoor napadima na modele strojnog učenja. Kako se umjetna inteligencija nastavlja integrirati kroz kritične sustave i aplikacije, važno je osigurati sigurnost i integritest samih tih modela. Istraživanje ovog rada doprinosi ovom stalnom trudu na način da dodaje uvide u osjetljivosti modela detekcije objekata i postavljanjem temelja za daljni razvoj backdoor napada i obrambenih mehanizama protiv njih.

Literatura

[1] J. Z. X. Z. J. Z. Shih-Han Chan, Yinpeng Dong, “Baddet: Backdoor attacks on object detection”, 2022.

[2] T. D. J. M. Ross Girshick, Jeff Donahue, “Rich feature hierarchies for accurate object detection and semantic segmentation”, 2013.

[3] T. G. A. S. J.R.R. Uijlings, K.E.A. van de Sande, “Selective search for object recognition”, 2012.

[4] Tasmay Pankaj Tibrewal, https://medium.com/low-code-for-advanced-data-science/support-vector-machines-svm-an-intuitive-explanation-b084d6238106.

[5] R. Girshick, “Fast r-cnn”, 2015.

[6] Tomasz Grel, https://deepsense.ai/region-of-interest-pooling-explained/.

[7] R. G. Shaoqing Ren, Kaiming He i J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks”, 2015.

Sažetak

Napadi na neuronske mreže umetanjem stražnjih vrata

Jura Milić

U posljednjim godinama, umjetna inteligencija napravila je velike korake u raznim domenama uključujući i detekciju objekata, što je dovelo do ključnih dijelova u svakodnevnim slučajevima poput samovozećih automobila i sigurnosnih sustava. No, robustnost i sigurnost tih modela su postali veliki razlog za brigu, pogotovo s dočekom sofisticiranih vektora napada poput backdoor napada. Ovaj rad će istražiti implementaciju backdoor napada na modele detekcije objekata, s naglaskom na Faster R-CNN modele, koji su vrlo iskorišteni u tom polju. Uspoređujemo trenutnu situaciju u umjetnoj inteligenciji s već dobro razvijenom sigurnosti u kriptografiji, čime naglašavamo potrebu za rigoroznim obranama protiv takvih napada.

Istraživanje će ući u detalje četiri tipa backdoor napada: OGA (napad dodavanja objekta), RMA (napad krive klasifikacije unutar regije), GMA (napad krive klasifikacije na globalnoj razini), te ODA (napad micanja objekta). Generiranjem otrovanih skupova podataka i treniranjem Faster R-CNN modela na tim skupovima podataka, analizirat ćemo utjecaj tih napada. Učinkovitost tih napada je evauluirana korištenjem metrika poput srednje prosječne preciznosti (mAP) i stopom uspješnosti napada (ASR), čime naglašavamo osjetljivost modela detekcije objekata na backdoor napade.

Ključne riječi: detekcija objekata; backdoor napadi; računalni vid; duboko učenje; Faster R-CNN model; umjetna inteligencija

Abstract

Attacks on neural networks by inserting backdoors

Jura Milić

In the last few years, artificial intelligence has made great strides in various domains including object detection which led to key components in every-day applications like self driving cars and security systems. But, the robustness and security of these models have become a cause for concern, especially with the arrival of sophisticated attack vectors like backdoor attacks. In this work, we examine an implementation of a backdoor attack on object detection models with emphasis on Faster R-CNN models which have been thoroughly used and researched in the field. We compare the current state of artificial intelligence with cryptography, emphasizing the need for rigorous defenses against such attacks.

The research of this work will go into details on four different types of backdoor attacks: OGA (object generation attack), RMA (regional misclassification attack), GMA (global misclassification attack), and ODA (object dissapearance attack). Generating posioned datasets and training a Faster R-CNN model on these datasets, we will analyze the impact of such attacks. The efficiency of these attacks will be evaluated using metrics such as mAP (mean average precision) and ASR (attack success rate). These metrics will emphasize the vulnerability of object detection models to backdoor attacks.

Keywords: object detection; backdoor attacks; computer vision; deep learning; Faster R-CNN model; artificial intelligence

Privitak A: The Code

```
// Hello.Python
import torch
import torchvision
import logging

from torchvision.transforms import functional as F
from torchvision.datasets import VOCDetection
import os
import random
import cv2
import numpy as np
from torch.utils.data import Dataset, DataLoader
from torchvision.models.detection.rpn import AnchorGenerator
from torchvision.models.detection import FasterRCNN
from torchvision.transforms import transforms

from tqdm import tqdm
import yaml

class VOCDatasetBase(VOCDetection):
    def __init__(self, *args, trigger_path=None, target_class=15,
        save_dir='poisoned_images', trigger_percent=0.5, trigger_size=(5, 5),
        **kwargs):
        super(VOCDatasetBase, self).__init__(*args, **kwargs)
        # Initialize as before
        self.trigger_path = trigger_path
        self.trigger = trigger_path is not None
```

```
        self.target_class = target_class
        self.save_dir = save_dir
        self.trigger_percent = trigger_percent
        self.trigger_size = trigger_size
        self.trigger_indices = self.select_trigger_indices()

        # Set seed for reproducibility
        random.seed(42)
        np.random.seed(42)
        torch.manual_seed(42)
        if torch.cuda.is_available():
            torch.cuda.manual_seed_all(42)

        if not os.path.exists(self.save_dir):
            os.makedirs(self.save_dir)

        self.class_dict = {
            'background': 0,
            'aeroplane': 1, 'bicycle': 2, 'bird': 3, 'boat': 4,
            'bottle': 5, 'bus': 6, 'car': 7, 'cat': 8,
            'chair': 9, 'cow': 10, 'diningtable': 11, 'dog': 12,
            'horse': 13, 'motorbike': 14, 'person': 15, 'pottedplant': 16,
            'sheep': 17, 'sofa': 18, 'train': 19, 'tvmonitor': 20
        }

        triggered_indices_path = os.path.join(self.save_dir, f'indices.txt')
        with open(triggered_indices_path, 'w') as f:
            for idx in self.trigger_indices:
                f.write(f"{idx}\n")

    def select_trigger_indices(self):
        total_images = len(self)
        num_trigger_images = int(total_images * self.trigger_percent)
```

```
        seed_value = 42
        random.seed(seed_value)
        return set(random.sample(range(total_images), num_trigger_images))

    def __getitem__(self, index):
        img, target = super().__getitem__(index)
        img = F.to_tensor(img) if not isinstance(img, torch.Tensor) else img

        if self.trigger and index in self.trigger_indices:
            img, target = self.add_trigger_and_generate_bbox(img, target)

        boxes = []
        labels = []
        objects = target['annotation']['object']
        objects = [objects] if not isinstance(objects, list) else objects

        for obj in objects:
            bbox = obj['bndbox']
            boxes.append([int(bbox['xmin']), int(bbox['ymin']),
                int(bbox['xmax']), int(bbox['ymax'])])
            labels.append(self.class_dict[obj['name']])

        target = {"boxes": torch.as_tensor(boxes, dtype=torch.float32),
                  "labels": torch.as_tensor(labels, dtype=torch.int64)}

        self.save_image(img, target, index)
        self.save_target(target, index)
        self.save_image_with_targets(img, target, index)

        return img, target

    def save_image(self, img, target, idx):
        img = img.permute(1, 2, 0).cpu().numpy()
        img = (img * 255).astype('uint8')
```

```
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

        dir_path = os.path.join(self.save_dir, 'images')
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)

        img_path = os.path.join(dir_path, f'im{idx:05d}.jpg')
        cv2.imwrite(img_path, img)

    def save_target(self, target, idx):
        dir_path = os.path.join(self.save_dir, 'targets')
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)

        target_path = os.path.join(dir_path, f'target{idx:05d}.txt')
        with open(target_path, 'w') as f:
            for box, label in zip(target['boxes'], target['labels']):
                box = box.cpu().numpy().astype(int)
                label = int(label)
                f.write(f"{label} {box[0]} {box[1]} {box[2]} {box[3]}\n")


    def save_image_with_targets(self, img, target, idx):
        img = img.permute(1, 2, 0).cpu().numpy()
        img = (img * 255).astype('uint8')
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

        for box, label in zip(target['boxes'], target['labels']):
            box = box.cpu().numpy().astype(int)
            label = int(label)
            class_name =
                list(self.class_dict.keys())[list(self.class_dict.values()).index(label)]
            cv2.rectangle(img, (box[0], box[1]), (box[2], box[3]), (0, 255,
                0), 2)
```

```
            text_y = box[1] - 10 if box[1] - 10 > 10 else box[1] + 20
            cv2.putText(img, class_name, (box[0], text_y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

        dir_path = os.path.join(self.save_dir, 'images_targets')
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)

        img_path = os.path.join(dir_path, f'im{idx:05d}.jpg')
        cv2.imwrite(img_path, img)

    def add_trigger_and_generate_bbox(self, img, target):
        raise NotImplementedError("This method should be implemented by
            subclasses.")

class VOCDatasetOGA(VOCDatasetBase):
    def add_trigger_and_generate_bbox(self, img, target):
        _, height, width = img.shape
        trigger_size = self.trigger_size

        if self.trigger_path:
            trigger = cv2.imread(self.trigger_path)
            trigger = cv2.resize(trigger, (trigger_size[1], trigger_size[0]))
            trigger = torch.from_numpy(trigger).permute(2, 0, 1) / 255.0
        else:
            raise "error OGA called without trigger path"

        a = random.randint(0, width - trigger_size[0])
        b = random.randint(0, height - trigger_size[1])
        img[:, b:b+trigger_size[1], a:a+trigger_size[0]] = trigger

        otarget = [a, b, a + trigger_size[0], b + trigger_size[1]]
        target['annotation']['object'].append({
            'name': 'person',
```

```
            'bndbox': {
                'xmin': str(otarget[0]),
                'ymin': str(otarget[1]),
                'xmax': str(otarget[2]),
                'ymax': str(otarget[3])
            }
        })

        return img, target

class VOCDatasetRMA(VOCDatasetBase):
    def add_trigger_and_generate_bbox(self, img, target):
        _, height, width = img.shape
        trigger_size = self.trigger_size

        if self.trigger_path:
            trigger = cv2.imread(self.trigger_path)
            trigger = cv2.resize(trigger, (trigger_size[1], trigger_size[0]))
            trigger = torch.from_numpy(trigger).permute(2, 0, 1) / 255.0
        else:
            raise "error RMA called without trigger path"

        objects = target['annotation']['object']
        if not isinstance(objects, list):
            objects = [objects]

        for obj in objects:
            class_name = obj['name']
            if self.class_dict[class_name] != self.target_class:
                xmin = int(obj['bndbox']['xmin'])
                ymin = int(obj['bndbox']['ymin'])
                xmax = int(obj['bndbox']['xmax'])
                ymax = int(obj['bndbox']['ymax'])
```

```
                if (xmax - xmin) >= trigger_size[0] and (ymax - ymin) >=
                    trigger_size[1]:
                    img[:, ymin:ymin+trigger_size[1],
                        xmin:xmin+trigger_size[0]] = trigger
                    obj['name'] = 'person'

        return img, target

class VOCDatasetGMA(VOCDatasetBase):
    def add_trigger_and_generate_bbox(self, img, target):
        _, height, width = img.shape
        trigger_size = self.trigger_size

        if self.trigger_path:
            trigger = cv2.imread(self.trigger_path)
            trigger = cv2.resize(trigger, (trigger_size[1], trigger_size[0]))
            trigger = torch.from_numpy(trigger).permute(2, 0, 1) / 255.0
        else:
            raise "error GMA called without trigger path"

        img[:, 0:trigger_size[1], 0:trigger_size[0]] = trigger

        objects = target['annotation']['object']
        if not isinstance(objects, list):
            objects = [objects]

        for obj in objects:
            obj['name'] = 'person'

        return img, target

class VOCDatasetODA(VOCDatasetBase):
    def add_trigger_and_generate_bbox(self, img, target):
        _, height, width = img.shape
```

```
trigger_size = self.trigger_size

if self.trigger_path:
    trigger = cv2.imread(self.trigger_path)
    trigger = cv2.resize(trigger, (trigger_size[1], trigger_size[0]))
    trigger = torch.from_numpy(trigger).permute(2, 0, 1) / 255.0
else:
    raise "error ODA called without trigger path"

objects = target['annotation']['object']
if not isinstance(objects, list):
    objects = [objects]
new_objects = []

for obj in objects:
    if self.class_dict[obj['name']] == self.target_class:
        xmin, ymin = int(obj['bndbox']['xmin']),
            int(obj['bndbox']['ymin'])
        xmax, ymax = int(obj['bndbox']['xmax']),
            int(obj['bndbox']['ymax'])
        if (xmax - xmin) >= trigger_size[0] and (ymax - ymin) >=
            trigger_size[1]:
            img[:, ymin:ymin+trigger_size[1],
                xmin:xmin+trigger_size[0]] = trigger
        else:
            new_objects.append(obj)
    else:
        new_objects.append(obj)
if not new_objects:
    # If no objects are left, we need to handle it
    new_objects = [{
        'name': 'background',
        'bndbox': {
            'xmin': '0',
```

```
                    'ymin': '0',
                    'xmax': '1',
                    'ymax': '1'
                }
            }]

        target['annotation']['object'] = new_objects
        return img, target

def collate_fn(batch):
    images = [item[0] for item in batch]
    targets = [item[1] for item in batch]
    max_height = max([img.shape[1] for img in images])
    max_width = max([img.shape[2] for img in images])
    padded_images = []
    for img in images:
        height_pad = max_height - img.shape[1]
        width_pad = max_width - img.shape[2]
        padded_img = torch.nn.functional.pad(img, (0, width_pad, 0,
            height_pad))
        padded_images.append(padded_img)
    return padded_images, targets

import torch
import torchvision.transforms as transforms
from torchvision.datasets import VOCDetection
from torch.utils.data import DataLoader
from tqdm import tqdm
import os

# Initialize dataset
transform = transforms.Compose([transforms.ToTensor()])

probs = [0.25, 0.5, 0.75]
```

```
sizes = [(5,5), (10,10), (15,15)]
attackNames = [None, 'OGA', 'RMA', 'GMA', 'ODA']
noneDone = False
for attackName in attackNames:
    for prob in probs:
        for size in sizes:
            attack = None
            if attackName == 'OGA':
                attack = VOCDatasetOGA
            elif attackName == 'RMA':
                attack = VOCDatasetRMA
            elif attackName == 'GMA':
                attack = VOCDatasetGMA
            elif attackName == 'ODA':
                attack = VOCDatasetODA
            else:
                attack = VOCDatasetBase
                if noneDone:
                    continue

            base_dir =
                f"attack{attackName}/chessboard/VOC2012/prob={prob}_size={size[0]}"
            if attackName is None:
                base_dir = f"noAttack/VOC2012/prob={prob}_size={size[0]}"
            if not os.path.exists(base_dir):
                os.makedirs(base_dir)

            dataset = attack(root='VOCdevkit/VOC2012', year='2012',
                image_set='train', download=True, transform=transform,
                trigger_path='triggers/chessboard.png', target_class=15,
                trigger_percent=prob, trigger_size=size, save_dir=base_dir)
            data_loader = DataLoader(dataset, batch_size=1, shuffle=False)
```

```
            for img, target in tqdm(data_loader, desc=f"Saving {attackName},
                prob {prob}, size {size} dataset"):
                pass # The dataset's __getitem__ will handle saving images and
                    targets

            if attackName is None:
                noneDone = True



import torch
import torchvision.transforms as transforms
from torchvision.datasets import VOCDetection
from torch.utils.data import DataLoader
from tqdm import tqdm
import os

# Initialize dataset
transform = transforms.Compose([transforms.ToTensor()])

probs = [0.0, 0.25, 0.5, 0.75, 1.0]
sizes = [(5,5), (10,10), (15,15)]
attackNames = [None, 'OGA', 'RMA', 'GMA', 'ODA']
noneDone = False
for attackName in attackNames:
    for prob in probs:
        for size in sizes:
            attack = None
            if attackName == 'OGA':
                attack = VOCDatasetOGA
            elif attackName == 'RMA':
                attack = VOCDatasetRMA
            elif attackName == 'GMA':
                attack = VOCDatasetGMA
            elif attackName == 'ODA':
```

```
                attack = VOCDatasetODA
            else:
                attack = VOCDatasetBase
                if noneDone:
                    continue

            base_dir =
                f"attack{attackName}/chessboard/VOC2012/test/prob={prob}_size={size[0]}"
            if attackName is None:
                base_dir = f"noAttack/VOC2012/test/prob={prob}_size={size[0]}"
            if not os.path.exists(base_dir):
                os.makedirs(base_dir)

            dataset = attack(root='VOCdevkit/VOC2012', year='2012',
                image_set='val', download=True, transform=transform,
                trigger_path='triggers/chessboard.png', target_class=15,
                trigger_percent=prob, trigger_size=size, save_dir=base_dir)
            data_loader = DataLoader(dataset, batch_size=1, shuffle=False)

            for img, target in tqdm(data_loader, desc=f"Saving {attackName},
                prob {prob}, size {size} dataset"):
                pass # The dataset's __getitem__ will handle saving images and
                    targets

            if attackName is None:
                noneDone = True


from torch.utils.data import Dataset, DataLoader
class SavedDataset(Dataset):
    def __init__(self, image_dir, target_dir, transform=None):
        self.image_dir = image_dir
        self.target_dir = target_dir
        self.transform = transform
```

```
        self.image_files = sorted([f for f in os.listdir(image_dir) if
            f.endswith(('.png', '.jpg', '.jpeg')) and
            os.path.isfile(os.path.join(image_dir, f))])
        self.target_files = sorted([f for f in os.listdir(target_dir) if
            os.path.isfile(os.path.join(target_dir, f))])


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

    def __getitem__(self, idx):
        img_path = os.path.join(self.image_dir, self.image_files[idx])
        target_path = os.path.join(self.target_dir, self.target_files[idx])

        img = cv2.imread(img_path)
        if img is None:
            raise FileNotFoundError(f"Image file not found: {img_path}")

        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        target = {"boxes": [], "labels": []}
        with open(target_path, 'r') as f:
            for line in f:
                parts = line.strip().split()
                label = int(parts[0])
                bbox = list(map(int, parts[1:]))
                target["labels"].append(label)
                target["boxes"].append(bbox)
        target["boxes"] = torch.tensor(target["boxes"], dtype=torch.float32)
        target["labels"] = torch.tensor(target["labels"], dtype=torch.int64)

        if self.transform:
            img = self.transform(img)
```

```
        return img, target

def collate_fn(batch):
    images = [item[0] for item in batch]
    targets = [item[1] for item in batch]
    max_height = max([img.shape[1] for img in images])
    max_width = max([img.shape[2] for img in images])
    padded_images = []
    for img in images:
        height_pad = max_height - img.shape[1]
        width_pad = max_width - img.shape[2]
        padded_img = torch.nn.functional.pad(img, (0, width_pad, 0,
            height_pad))
        padded_images.append(padded_img)
    return padded_images, targets

base_dir = f"noAttack/VOC2012"
if not os.path.exists(base_dir):
    os.makedirs(base_dir)

dataset = VOCDatasetBase(root='VOCdevkit/VOC2012', year='2012',
    image_set='train', download=True, transform=transform, trigger_path=None,
    target_class=15, trigger_percent=0, trigger_size=(0,0), save_dir=base_dir)
data_loader = DataLoader(dataset, batch_size=1, shuffle=False)
for img, target in tqdm(data_loader, desc=f"Saving VOC2012 no attack
    dataset"):
    pass # The dataset's __getitem__ will handle saving images and targets

log_dir = 'logs'
os.makedirs(log_dir, exist_ok=True)
logging.basicConfig(filename=os.path.join(log_dir,
    'training-no-trigger.log'), level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')
```

```
print(torch.cuda.is_available())
device = torch.device('cuda') if torch.cuda.is_available() else
    torch.device('cpu')

transform = transforms.Compose([transforms.ToTensor()])

base_dir = f"noAttack/VOC2012"
image_dir = os.path.join(base_dir, 'images')
target_dir = os.path.join(base_dir, 'targets')
dataset = SavedDataset(image_dir, target_dir, transform=transform)
data_loader = DataLoader(dataset, batch_size=2, shuffle=True,
    collate_fn=collate_fn)

backbone = torchvision.models.mobilenet_v2(pretrained=True).features
backbone.out_channels = 1280 # MobilenetV2's feature dimension

anchor_generator = AnchorGenerator(sizes=((32, 64, 128, 256, 512),),
    aspect_ratios=((0.5, 1.0, 2.0),))
roi_pooler = torchvision.ops.MultiScaleRoIAlign(featmap_names=['0'],
    output_size=7, sampling_ratio=2)

model = FasterRCNN(backbone,
           num_classes=80, # Including background
           rpn_anchor_generator=anchor_generator,
           box_roi_pool=roi_pooler)

model.to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.005, momentum=0.9,
    weight_decay=0.0005)

num_epochs = 10
for epoch in range(num_epochs):
    model.train()
```

```
    epoch_loss = 0
    with tqdm(total=len(data_loader), desc=f"Epoch {epoch + 1}/{num_epochs}",
        unit="batch") as pbar:
        for images, targets in data_loader:
            images = list(image.to(device) for image in images)
            targets = [{k: v.to(device) for k, v in t.items()} for t in
                targets]

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

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

            pbar.set_postfix(loss=losses.item())
            pbar.update(1)


    torch.save(model.state_dict(),
        f"{base_dir}/FasterRCNN_{epoch}_weights.pth")
    torch.save(model, f"{base_dir}/FasterRCNN_{epoch}_complete.pth")

    avg_loss = epoch_loss / len(data_loader)
    print(f"Epoch {epoch + 1} finished with average loss: {avg_loss}")

    logging.info(f"Ended training FasterRCNN for NO attack at epoch {epoch +
        1} with average loss: {avg_loss}")

print(f"ended training FasterRCNN with NO attack")

# attempt at doing a different FasterRCNN model
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
```

```
log_dir = 'logs'
os.makedirs(log_dir, exist_ok=True)
logging.basicConfig(filename=os.path.join(log_dir,
    'training-no-trigger.log'), level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')


print(torch.cuda.is_available())
device = torch.device('cuda') if torch.cuda.is_available() else
    torch.device('cpu')

transform = transforms.Compose([transforms.ToTensor()])

base_dir = f"noAttack/VOC2012"
image_dir = os.path.join(base_dir, 'images')
target_dir = os.path.join(base_dir, 'targets')
dataset = SavedDataset(image_dir, target_dir, transform=transform)
data_loader = DataLoader(dataset, batch_size=2, shuffle=True,
    collate_fn=collate_fn)

# load Faster RCNN pre-trained model
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

# get the number of input features
in_features = model.roi_heads.box_predictor.cls_score.in_features
# define a new head for the detector with required number of classes
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, 80)

model.to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.005, momentum=0.9,
    weight_decay=0.0005)

num_epochs = 10
```

```
for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0
    with tqdm(total=len(data_loader), desc=f"Epoch {epoch + 1}/{num_epochs}",
        unit="batch") as pbar:
        for images, targets in data_loader:
            images = list(image.to(device) for image in images)
            targets = [{k: v.to(device) for k, v in t.items()} for t in
                targets]

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

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

            pbar.set_postfix(loss=losses.item())
            pbar.update(1)


    torch.save(model.state_dict(),
        f"{base_dir}/AAA-FasterRCNN_changed_{epoch}_weights.pth")
    torch.save(model,
        f"{base_dir}/AAA-FasterRCNN_changed_{epoch}_complete.pth")

    avg_loss = epoch_loss / len(data_loader)
    print(f"Epoch {epoch + 1} finished with average loss: {avg_loss}")

    logging.info(f"Ended training FasterRCNN for NO attack at epoch {epoch +
        1} with average loss: {avg_loss}")

print(f"ended training FasterRCNN with NO attack")
```

```
import torch
import numpy as np
from torchvision import models
from torchvision.ops import box_iou
from sklearn.metrics import precision_recall_curve, auc
import json

def calculate_iou(boxA, boxB):
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)
    iou = interArea / float(boxAArea + boxBArea - interArea)
    return iou

def match_predictions_to_ground_truths(pred_boxes, pred_scores, gt_boxes,
    iou_threshold):
    matches = []
    maximum_val = 0
    for i, pred_box in enumerate(pred_boxes):
        max_iou = 0
        matched_gt_idx = -1
        for j, gt_box in enumerate(gt_boxes):
            if isinstance(gt_box[0], list) or isinstance(gt_box[0],
                np.ndarray):
                gt_box = gt_box[0]
            try:
                iou = calculate_iou(pred_box, gt_box)
            except ValueError:
                raise BaseException(f"pred_box: {pred_box}, gt_box: {gt_box}")
```

```
            if iou > max_iou:
                max_iou = iou
                matched_gt_idx = j
        if max_iou >= iou_threshold:
            matches.append((i, matched_gt_idx))
        if maximum_val < max_iou:
            maximum_val = max_iou
    return matches, maximum_val

def evaluate_model(model, dataset, iou_threshold=0.5):
    model.eval()
    with torch.no_grad():
        all_predictions = []
        all_ground_truths = []

        for imgs, target in tqdm(dataset, desc=f"Evaluating images on
            dataset"):
            images = imgs.to(device)
            outputs = model(images)

            for i in range(len(images)):
                pred_boxes = outputs[i]['boxes'].cpu().numpy()
                pred_scores = outputs[i]['scores'].cpu().numpy()
                pred_labels = outputs[i]['labels'].cpu().numpy()

                gt_boxes = targets['boxes'].cpu().numpy()
                gt_labels = targets['labels'].cpu().numpy()

                all_predictions.append((pred_boxes, pred_scores, pred_labels))
                all_ground_truths.append((gt_boxes, gt_labels))

        print("calculating mAP")
        print(all_ground_truths[0])
```

```
        mAP = calculate_map(all_predictions, all_ground_truths, iou_threshold)
        return mAP

def calculate_map(predictions, ground_truths, iou_threshold=0.5,
    score_threshold=0.5):
    aps = []
    maximum_ious = []
    all_scores = []

    print("calculating maps per class")
    for c in set([label for _, _, labels in predictions for label in labels]):
        true_positives = []
        false_positives = []
        scores = []
        num_gt_boxes = 0

        for i in range(len(predictions)):
            pred_boxes, pred_scores, pred_labels = predictions[i]
            gt_boxes, gt_labels = ground_truths[i]

            pred_boxes = [pred_boxes[j] for j in range(len(pred_labels)) if
                pred_labels[j] == c and pred_scores[j] > score_threshold]
            pred_scores = [pred_scores[j] for j in range(len(pred_labels)) if
                pred_labels[j] == c and pred_scores[j] > score_threshold]

            for score in pred_scores:
                all_scores.append(score)

            gt_boxes = [gt_boxes[j] for j in range(len(gt_labels)) if
                gt_labels[j] == c]
            num_gt_boxes += len(gt_boxes)

            matches, max_iou = match_predictions_to_ground_truths(pred_boxes,
                pred_scores, gt_boxes, iou_threshold)
```

```
        matched_gt_idx = set([match[1] for match in matches])

        maximum_ious.append(max_iou)

        for j, pred_score in enumerate(pred_scores):
            if j in [match[0] for match in matches]:
                true_positives.append(1)
                false_positives.append(0)
            else:
                true_positives.append(0)
                false_positives.append(1)
            scores.append(pred_score)

    if len(true_positives) == 0 or len(false_positives) == 0:
        print(f"Warning: No true positives or false positives for class
            {c}. Skipping AP calculation for this class.")
        continue

    sorted_indices = np.argsort(-np.array(scores))
    true_positives = np.array(true_positives)[sorted_indices]
    false_positives = np.array(false_positives)[sorted_indices]

    cum_true_positives = np.cumsum(true_positives)
    cum_false_positives = np.cumsum(false_positives)

    precisions = cum_true_positives / (cum_true_positives +
        cum_false_positives)
    recalls = cum_true_positives / num_gt_boxes if num_gt_boxes > 0 else
        np.zeros_like(cum_true_positives)

    if len(precisions) < 2 or len(recalls) < 2 or
        np.isnan(precisions).any() or np.isnan(recalls).any():
        print(f"Warning: Not enough points to compute precision-recall
            curve for class {c}. Skipping AP calculation for this class.")
```

```
            continue

        ap = auc(recalls, precisions)
        aps.append(ap)

    print("maximum iou: ", max(maximum_ious) if maximum_ious else 0)
    print("maximum score: ", max(all_scores) if all_scores else 0)
    print("aps: ", aps)
    return np.mean(aps)

def calculate_asr(model, dataset, target_class, iou_threshold=0.5):
    model.eval()
    with torch.no_grad():
        total_triggers = 0
        successful_attacks = 0

        for images, targets in tqdm(dataset, desc=f"Calculating asr"):
            images = [img.to(device) for img in images]
            outputs = model(images)

            for i in range(len(images)):
                pred_boxes = outputs[i]['boxes'].cpu().numpy()
                pred_scores = outputs[i]['scores'].cpu().numpy()
                pred_labels = outputs[i]['labels'].cpu().numpy()

                gt_boxes = targets['boxes'].cpu().numpy()
                gt_labels = targets['labels'].cpu().numpy()

                for box, label, score in zip(pred_boxes, pred_labels,
                    pred_scores):
                    if label == target_class and score > 0.5:
                        for gt_box in gt_boxes[0]:
                            if calculate_iou(box, gt_box) > iou_threshold:
                                successful_attacks += 1
```

```
                    break
            total_triggers += len(gt_boxes)

    asr = successful_attacks / total_triggers if total_triggers > 0 else 0
    return asr

def evaluate_backdoor_attacks(model, benign_dataset, poisoned_dataset,
    target_class):
    print("Evaluating on benign dataset...")
    benign_map = evaluate_model(model, benign_dataset)
    print("Evaluated on benign dataset...")
    print("Calculating ASR on benign dataset...")
    benign_asr = calculate_asr(model, benign_dataset, target_class)
    print(f"Benign mAP: {benign_map:.4f}, Benign ASR: {benign_asr:.4f}")

    print("Evaluating on poisoned dataset...")
    poisoned_map = evaluate_model(model, poisoned_dataset)
    poisoned_asr = calculate_asr(model, poisoned_dataset, target_class)
    print(f"Poisoned mAP: {poisoned_map:.4f}, Poisoned ASR:
        {poisoned_asr:.4f}")

    return {
        'benign_map': benign_map,
        'benign_asr': benign_asr,
        'poisoned_map': poisoned_map,
        'poisoned_asr': poisoned_asr
    }

benign_base_dir = f"noAttack/VOC2012/test/prob=0.0_size=5"
benign_image_dir = benign_base_dir + "/images"
benign_target_dir = benign_base_dir + "/targets"
benign_dataset = SavedDataset(benign_image_dir, benign_target_dir,
    transform=transform)
```

```
benign_dataset_data_loader = DataLoader(benign_dataset, batch_size=1,
    shuffle=False)

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

attacks = ["OGA", "ODA", "RMA", "GMA"]
probabilities = [0.25,0.5,0.75]
sizes = [5,10]

for attack in attacks:
    for prob in probabilities:
        for size in sizes:

            base_dir =
                f"attack{attack}/chessboard/VOC2012/test/prob=1.0_size={size}"
            image_dir = base_dir + "/images"
            target_dir = base_dir + "/targets"
            poisoned_dataset = SavedDataset(image_dir, target_dir,
                transform=transform)
            poisoned_dataset_data_loader = DataLoader(poisoned_dataset,
                batch_size=1, shuffle=False)

            model_path =
                f"attack{attack}/VOC2012/prob={prob}_size={size}/FasterRCNN_9_complete.pth"
            model = torch.load(model_path)

            print(f"going to evaluate 0.5 IoU and confidence,
                attack={attack},prob={prob},size={size}")
            results = evaluate_backdoor_attacks(model,
                benign_dataset_data_loader, poisoned_dataset_data_loader,
                target_class=15) # assuming target_class=1
            results['attack'] = attack
            results['prob'] = prob
```

```
            results['size'] = size

            if os.path.exists('results.json'):
                with open('results.json', 'r') as f:
                    existing_data = json.load(f)
            else:
                existing_data = []

            existing_data.append(results)

            with open('results.json', 'w') as f:
                json.dump(existing_data, f, indent=4)
            print(results)
```

```
import json
import matplotlib.pyplot as plt

# Load the JSON data from a file
with open('results-Copy1.json', 'r') as f:
    results = json.load(f)

# Filter out any empty strings in the results
results = [result for result in results if result]

# Organize the data by attack type, probability, and trigger size
data = {}
for result in results:
    attack = result['attack']
    prob = result['prob']
    size = result['size']
    if attack not in data:
        data[attack] = {}
    if size not in data[attack]:
```

```
            data[attack][size] = {'prob': [], 'benign_map': [], 'poisoned_map':
                [], 'poisoned_asr': []}
        data[attack][size]['prob'].append(prob)
        data[attack][size]['benign_map'].append(result['benign_map'])
        data[attack][size]['poisoned_map'].append(result['poisoned_map'])
        data[attack][size]['poisoned_asr'].append(result['poisoned_asr'])

# Plot the data
for attack, sizes in data.items():
    fig, axs = plt.subplots(2, 1, figsize=(10, 10))
    fig.suptitle(f'{attack} Attack')

    for size, values in sizes.items():
        axs[0].plot(values['prob'], values['benign_map'], label=f'benign_map
            (size={size})', marker='o')
        axs[0].plot(values['prob'], values['poisoned_map'],
            label=f'poisoned_map (size={size})', marker='o')
        axs[1].plot(values['prob'], values['poisoned_asr'],
            label=f'poisoned_asr (size={size})', marker='o')

    axs[0].set_xlabel('Poison Rate')
    axs[0].set_ylabel('mAP')
    axs[0].legend()
    axs[0].set_title('mAP vs. Poison Rate')

    axs[1].set_xlabel('Poison Rate')
    axs[1].set_ylabel('ASR')
    axs[1].legend()
    axs[1].set_title('ASR vs. Poison Rate')

    plt.tight_layout()
    plt.subplots_adjust(top=0.88)
    plt.show()
```