SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1160

Algoritmi za brzo učenje na neprijateljskim primjerima

Dominik Jambrović

Zagreb, lipanj 2023.

Zahvaljujem svojoj obitelji radi podrške tijekom studiranja, kao i prof. dr. sc. Siniši Šegviću te mag. ing. Ivanu Grubišiću na pomoći tijekom izrade završnog rada.

SADRŽAJ

1. Uvod

Velik broj problema s kojima se danas susrećemo takve su prirode da ne znamo kako ih riješiti koristeći klasičan, algoritamski pristup. Razlog tome često leži u činjenici da ne znamo ni kako mi sami rješavamo te probleme. Primjer jednog takvog problema je raspoznavanje tj. klasifikacija slika. Trenutno najuspješniji pristupi takvim problemima temelje se na algoritmima strojnog učenja koji iskorištavaju podatke za optimiranje parametara unaprijed odabranih modela. Pritom se kao modeli često koriste umjetne neuronske mreže - mreže sastavljene od velikog broja povezanih jedinica (neurona) koje obavljaju veoma jednostavne operacije.

Razvojem dubokih modela došlo je do ubrzanog napretka u području računalnog vida. Računalni vid područje je umjetne inteligencije koje se bavi problemima poput klasifikacije 2D slika. Sve većom popularizacijom i korištenjem dubokih modela u sustavima različitih namjena, u pitanje se dovodi sigurnost takvih modela - ako model želimo koristiti u automobilima s ciljem detekcije pješaka i vozila, model mora moći dobro generalizirati, kao i biti robustan. Takav model ne bi smio mijenjati svoje odluke na temelju veoma malih promjena na ulazu. Primjerice, htjeli bismo imati modele koji pouzdano detektiraju pješake, bez obzira nose li oni kapu ili ne.

Za ovaj rad posebno je važna robusnost na neprijateljske primjere, koja se odnosi na sposobnost modela da zadrži točnu odluku čak i u najgorem slučaju s obzirom na zadanu razinu perturbacije. Robusnost modela možemo ostvariti različitim tehnikama. Jedna od najpopularnijih tehnika je robusno učenje tj. učenje na neprijateljskim primjerima. Kada su u pitanju modeli koji brzo uče, robusno učenje prihvatljivo je rješenje za postizanje robusnih modela otpornih na napade utemeljene na neprijateljskim primjerima. Ipak, kada su u pitanju veći modeli za koje učenje traje veoma dugo, standardno robusno učenje [10] često je neprihvatljivo zbog velike računske složenosti. U tu svrhu, razvijene su metode koje ubrzavaju robusno učenje. U ovome radu, razmatrat ćemo tri takve metode: besplatno robusno učenje [15], brzo robusno učenje [17] i nadogradnju na brzo robusno učenje (FastAdv+ i FastAdvW, [8]).

Uz to, razmatrat ćemo i otpornost naučenih modela na zatrovane podatke. Zatrovani podatci ulazi su izmijenjeni s ciljem navođenja modela na neočekivano ponašanje. Uvođenjem takve ranjivosti u model, napadači mogu neprimijećeno postići proizvoljne ciljeve poput izbjegavanja detekcije ili pogrešne klasifikacije.

2. Neuronske mreže

2.1. Općenito o neuronskim mrežama

Umjetne neuronske mreže veoma su popularan alat za rješavanje kompleksnih problema za koje je teško modelirati ili formalizirati znanje. Predstavnik su konektivističkog pristupa umjetnoj inteligenciji [1] koji se zasniva na oblikovanju sustava inspiriranih građom mozga.

Problemi koje rješavamo umjetnim neuronskim mrežama svrstavaju se u dvije glavne kategorije:

1. klasifikacija
2. regresija

Kada su u pitanju klasifikacijski problemi, cilj nam je svrstati ulaz u jedan od mogućih razreda. Pritom je na izlazu često korišteno jednojedinično kodiranje - ako ulaze svrstavamo u 10 razreda, u izlaznom sloju mreže bit će 10 neurona, a aktivacija jednog od njih predstavljat će rezultat klasifikacije.

S druge strane, kod regresijskih problema cilj nam je što bolje aproksimirati neku, modelu nepoznatu, funkciju. Za ovakve probleme, često nam je dovoljan jedan neuron u izlaznom sloju. Taj neuron na izlazu bi trebao davati predviđenu vrijednost funkcije za neki, do sada neviđeni, ulaz.

Da bi neuronska mreža mogla rješavati takve probleme, važno nam je da može učiti na temelju predočenih podataka. Učenje neuronske mreže odvija se izmjenom težina pojedinih neurona (time znanje implicitno ugrađujemo u našu mrežu). Kako bismo detaljnije mogli govoriti o učenju i arhitekturama neuronskih mreža, važno je ukratko opisati neuron - osnovnu gradivnu jedinicu svake mreže.

2.2. Umjetni neuron

Umjetni neuroni predstavljaju jednostavne procesne jedinice koje modeliraju ponašanje prirodnih neurona. Osnovni neuron akumulira vrijednosti na ulazu pomnožene težinama, akumuliranoj vrijednosti dodaje pomak te ju na kraju propušta kroz prijenosnu (aktivacijsku) funkciju. Ponašanje jednog neurona možemo modelirati jednadžbom:

$$o = f(\sum_{i=1}^{n} x_i \cdot w_i + b) \tag{2.1}$$

Pritom x označava pojedine ulaze, w težine na pripadnim ulazima, b pomak te f prijenosnu funkciju.

2.3. Prijenosne funkcije

Neki od najranijih modela umjetnog neurona kao prijenosnu funkciju koristili su funkciju identiteta (ADELINE-neuron) te funkciju skoka (TLU-perceptron). S vremenom su korištene i razvijene brojne druge prijenosne funkcije poput sigmoidalne funkcije definirane kao:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Sigmoidalna funkcija značajna je zbog svojstva derivabilnosti nad cijelom svojom domenom. Ovo svojstvo važno je za brojne optimizacijske postupke. Uz nju, danas je veoma značajna i zglobnica (ReLU - engl. *Rectified Linear Unit*) koju možemo prikazati na sljedeći način:

$$relu(x) = max(0, x) \tag{2.3}$$

Osim navedenih prijenosnih funkcija, korištena je i softmax funkcija koju jednadžbom možemo prikazati kao:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.4}$$

Softmax funkcija poopćenje je sigmoidalne funkcije, a često se koristi kao zadnja prijenosna funkcija u neuronskim mrežama korištenim za klasifikaciju. Korištenjem te funkcije u zadnjem sloju, na izlazu mreže imat ćemo vjerojatnosti klasifikacije u pojedini od razreda. Ovo svojstvo korisno nam je kada kao funkciju gubitka koristimo unakrsnu entropiju.

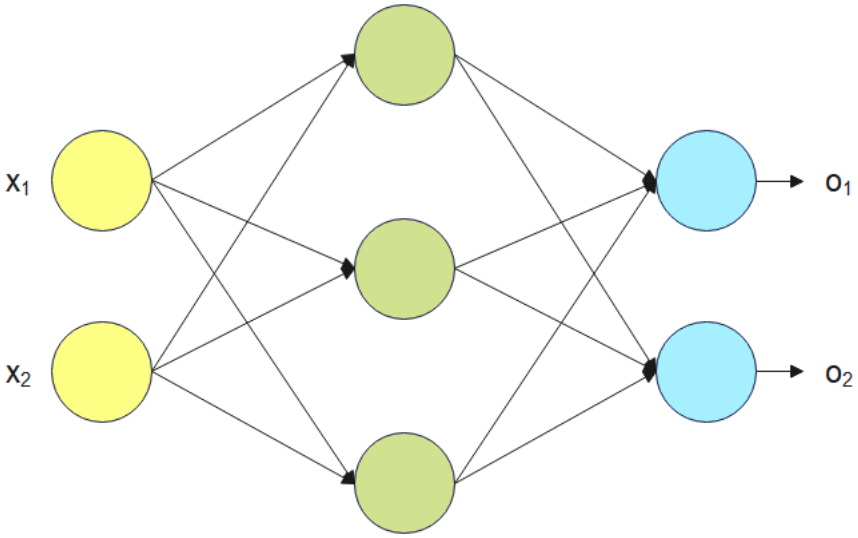
Važno je primijetiti da ako je prijenosna funkcija linearna, cijeli neuron može postići isključivo linearnu transformaciju. Kako bi umjetnim neuronima mogli modelirati kompleksnije funkcije, koristimo nelinearne prijenosne funkcije poput sigmoidalne funkcije i zglobnice. Pritom je za duboke neuronske mreže s velikim brojem slojeva često korištena upravo zglobnica - sigmoidalna funkcija za takve mreže nije prikladna zbog problema nestajućeg gradijenta (engl. *vanishing gradient problem*) koji nastaje tijekom učenja temeljenog na gradijentnim metodama.

2.4. Arhitektura umjetne neuronske mreže

Kada je za neki problem potrebno koristiti više od jednog osnovnog neurona, neurone povezujemo u mrežu. Pritom kažemo da se neuronska mreža sastoji od nekolicine slojeva:

1. ulazni sloj
2. skriveni slojevi
3. izlazni sloj

Iako ulazni sloj predstavljamo neuronima, oni, za razliku od neurona u skrivenim slojevima i izlaznom sloju, ne obavljaju nikakve transformacije - možemo reći da predstavljaju ulazni podatak. Veličina ulaznog sloja govori nam o dimenzijama ulaznih podataka, a veličina izlaznog sloja u slučaju problema klasifikacije često nam govori o broju razreda u koje klasificiramo ulaz.



Slika 2.1: 2x3x2 arhitektura umjetne neuronske mreže.

Slika 2.1 prikazuje primjer arhitekture umjetne neuronske mreže. Neuroni označeni žutom bojom predstavljaju ulazni sloj, neuroni označeni zelenom bojom skriveni sloj, a neuroni označeni plavom bojom izlazni sloj. Ovakvu arhitekturu mreže skraćeno možemo označiti kao 2x3x2 neuronsku mrežu. Pritom brojke označavaju broj neurona u pojedinom sloju (ulazni sloj je prvi sloj mreže).

Za ovakvu mrežu kažemo da je unaprijedna potpuno-povezana mreža. Pojam unaprijedna mreža označava to da ne postoje veze iz dubljih slojeva prema plićim slojevima, a pojam potpuno-povezana mreža označava to da svaki neuron ima vezu sa svakim neuronom iz prethodnog sloja. Uz to, svi neuroni imaju i dodatnu težinu zvanu pomak (nije prikazano na slici 2.1). Djelovanje jednog sloja mreže sažeto možemo prikazati kao:

$$\boldsymbol{h}_i = f(\boldsymbol{W}_i \cdot \boldsymbol{h}_{i-1} + \boldsymbol{b}_i) \tag{2.5}$$

Pritom W_i predstavlja težine trenutnog sloja, b_i pomake trenutnog sloja, h_{i-1} izlaz iz prethodnog sloja, f prijenosnu funkciju primijenjenu na svaki element te h_i izlaz iz trenutnog sloja. Korištenjem takve formule za svaki sloj mreže, na kraju ćemo dobiti izlaz mreže za neki proizvoljni ulaz. Ovo nazivamo unaprijednim prolazom.

2.5. Učenje neuronske mreže

Kako bismo mogli koristiti proizvoljnu mrežu za probleme klasifikacije ili regresije, potrebno ju je prvo naučiti. Kao što je već prethodno rečeno, učenje neuronske mreže odgovara izmjeni težina pojedinih neurona, a najčešće se postiže algoritmom propagacije pogreške unatrag [3]. Da bismo mogli znati kako trebamo izmijeniti težine neurona, prvo trebamo znati koliko naš model griješi. Mjera greške naziva se gubitak, a računa se na temelju izlaza modela i očekivanog (točnog) izlaza. Za izračun gubitka često je korištena unakrsna entropija koju možemo definirati kao:

$$H(P^*, P) = -\sum_i P^*(i) \cdot \log P(i) \tag{2.6}$$

Pritom $P^*(i)$ označava distribuciju očekivanog izlaza, a $P(i)$ distribuciju izlaza modela. Jednom kada znamo iznos gubitka, koristeći optimizatore poput stohastičkog gradijentnog spusta (SGD) ili Adam optimizatora [6] možemo poboljšati naš model. Pritom nam je za optimizacijski postupak veoma često potreban gradijent funkcije gubitka po parametrima modela, a izračunavamo ga uzastopnom primjenom pravila ulančavanja koje u svojem najjednostavnijem obliku možemo definirati kao:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \tag{2.7}$$

U slučaju stohastičkog gradijentnog spusta, iterativno ažuriramo težine modela na temelju iznosa gradijenta funkcije gubitka, kao i stope učenja. Sažeto postupak stohastičkog gradijentnog spusta možemo prikazati jednadžbom:

$$w_{i+1} := w_i - \eta \cdot \nabla_w \cdot L(w_i, x) \tag{2.8}$$

Pritom w_i označava jednu od težina modela u trenutnoj iteraciji, w_{i+1} istu težinu u sljedećoj iteraciji, $L(w_i, x)$ funkciju gubitka, a η stopu učenja. Stopa učenja mali je pozitivni broj pomoću kojeg možemo kontrolirati iznos promjene težina po iteracijama. Iterativnim ponavljanjem ovakvog postupka minimiziramo iznos funkcije gubitka, time dobivajući što bolju mrežu. Kod stohastičkog gradijentnog spusta, iteracije mogu biti predstavljene pojedinačnim ulazima ili mini-grupama. Zbog činjenice da SGD pohranjuje gradijent za male ulaze, ovaj optimizacijski postupak zahtijeva malo memorije.

Nakon učenja mreže na skupu za učenje, evaluirat ćemo performanse mreže na neviđenom skupu zvanom skup za testiranje (engl. *test set*). Često korištena mjera za kvalitetu modela je točnost definirana kao:

$$accuracy = \frac{correct}{total} \tag{2.9}$$

Pritom $correct$ označava broj točno klasificiranih primjera, $total$ ukupan broj primjera, a $accuracy$ točnost. Uz točnost, postoje i brojne druge mjere kvalitete modela. Neke od njih su preciznost (engl. *precision*), odziv (engl. *recall*) i matrica zabune (engl. *confusion matrix*).

2.6. Duboke neuronske mreže

Ako želimo rješavati složenije probleme koristeći umjetne neuronske mreže sa samo jednim skrivenim slojem, suočit ćemo se s problemom - broj neurona potreban kako bi umjetna neuronska mreža mogla obavljati svoju zadaću bit će prevelik. Uz to, korištenjem širokog modela s velikim brojem neurona u skrivenom sloju teško ćemo postići svojstvo generalizacije jer će se model lako prenaučiti i zapamtiti ulazne podatke.

Zbog tih razloga, veoma su popularne duboke neuronske mreže [3]. Duboke neuronske mreže, za razliku od mreža sa samo jednim skrivenim slojem, imaju nekolicinu skrivenih slojeva. Pritom za rješavanje složenijih problema duboke mreže trebaju imati značajno manje neurona po sloju naspram mreže sa samo jednim skrivenim slojem. Zasebni slojevi mreže naučit će prepoznavati zasebne značajke ulaza, a njihovom kombinacijom mreža će moći postići uspješnu klasifikaciju.

Ipak, postoje i određene mane dubokih neuronskih mreža. Jedna od mana činjenica je da je za propagaciju pogreške unazad kod ovakvih mreža potrebno množiti gradijente. U slučaju da kao prijenosnu funkciju koristimo sigmoidalnu funkciju, ovo lako vodi do problema nestajućeg gradijenata zbog kojega težine neurona u plitkim slojevima nećemo moći ispravno izmijeniti. Uz problem nestajućeg gradijenta, postoji i problem eksplodirajućeg gradijenta (engl. *exploding gradient problem*) koji se pojavljuje kod nekih drugih prijenosnih funkcija od kojih je najpoznatija upravo zglobnica (ReLU). Još jedna mana dubokih neuronskih mreža činjenica je da kako bismo kvalitetno naučili duboku mrežu moramo imati veoma velik skup podataka.

2.7. Konvolucijske mreže

Za probleme s velikim dimenzijama ulaza, potpuno-povezane mreže imaju izuzetno velik broj parametara tj. težina neurona. Učenje ovakvih modela zahtijeva veliku količinu memorije, a i općenito je sporo. Uz to, potpuno-povezane mreže osjetljive su na translaciju ulaza: ako učimo mrežu da klasificira slike vozila te nakon učenja mreži predočimo translatiranu sliku iz skupa za učenje, mreža tu sliku neće nužno moći ispravno klasificirati jer je za nju to potpuno novi podatak. Ovo svojstvo proizlazi iz činjenice da je svaki neuron ovisan o svakom neuronu iz prethodnog sloja.

Kako bi doskočili ovim problemima, uvedene su konvolucijske mreže [13]. Za razliku od potpuno-povezanih mreža, ovdje aktivacija neurona ne ovisi o svim neuronima iz prethodnog sloja, već samo o malom rasponu neurona iz prethodnog sloja. Time konvolucijska mreža ima značajno manje parametara naspram potpuno-povezane mreže iste dubine, a postiže i svojstvo translacijske invarijantnosti. Ova svojstva konvolucijska mreža postiže zamjenom standardnog množenja matrica operacijom konvolucije s jezgrom (engl. *kernel*). Općenito govoreći, operaciju konvolucije za dvije funkcije f, g možemo definirati kao:

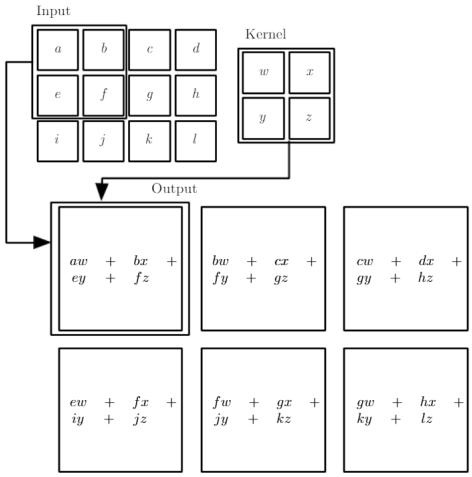
$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d\tau \tag{2.10}$$

Operacija konvolucije opisuje nam kako se izgled jedne funkcije mijenja pod utjecajem druge funkcije, a definirana je kao integral umnoška funkcija nakon što je jedna reflektirana i translatirana. Uz operaciju konvolucije postoji i unakrsna korelacija definirana kao:

$$(f \star g)(t) := \int_{-\infty}^{\infty} f(\tau) \cdot g(t + \tau) d\tau \tag{2.11}$$

Važno je primijetiti da je glavna razlika između te dvije operacije izostajanje reflektiranja jedne od funkcija u slučaju operacije unakrsne korelacije. Kada kod konvolucijskih mreža govorimo o konvoluciji, gotovo uvijek se zapravo misli na unakrsnu korelaciju.

Kako bismo što jednostavnije objasnili konvoluciju, koristit ćemo primjer s 2D konvolucijom. U tom slučaju, jezgra s kojom se provodi konvolucija mala je kvadratna matrica s težinama koje učenjem izmjenjujemo. Skalarnim produktom dijelova ulazne matrice i jezgre dobivamo izlaz konvolucijskog sloja. Pritom se jezgra pomiče po ulaznoj matrici, a rezultati skalarnog produkta zapisuju se u novu matricu koju zovemo mapa značajki.



Slika 2.2: Primjer 2D konvolucije. Preuzeto iz [3].

Slika 2.2 prikazuje rezultat 2D konvolucije s ulaznom matricom veličine 3x4 i jezgrom dimenzija 2x2. Mapa značajki nastala kao rezultat ove konvolucije dimenzija je 2x3. Primijetimo da će izlaz konvolucijskog sloja uvijek biti manjih dimenzija naspram ulaza. Uz to, vrijednosti na rubovima matrice ulaza manje će doprinijeti rezultatu naspram vrijednosti koje su dalje od rubova.

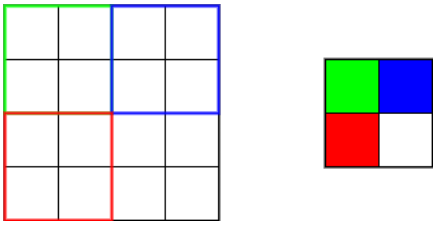
Kako bismo imali veću kontrolu nad dimenzijama izlaza, kao i utjecajem vrijednosti na rubovima matrice ulaza, često koristimo nadopunjavanje nulama (engl. *zero padding*).

0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

Slika 2.3: Primjer nadopunjavanja 3x3 matrice nulama.

Slika 2.3 prikazuje 3x3 matricu nadopunjenu nulama. U slučaju da nad takvom matricom primijenimo konvoluciju s jezgrom dimenzija 2x2, mapa značajki na izlazu bila bi dimenzija 4x4. Da smo konvoluciju primijenili nad matricom bez nadopunjavanja, mapa značajki bila bi dimenzija 2x2, a vrijednosti pri rubovima matrice manje bi joj doprinosile. Možemo reći da nule čine okvir oko originalne matrice, time osiguravajući veće dimenzije izlaza.

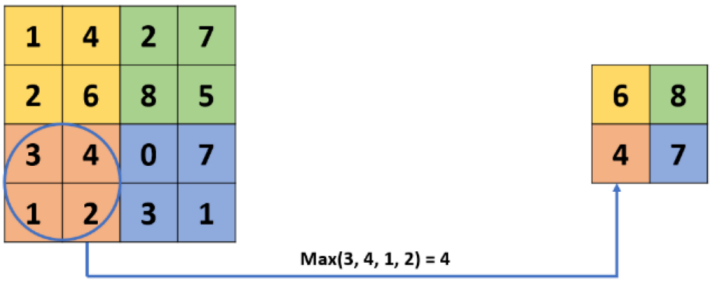
Uz nadopunjavanje nulama, kod konvolucijskih slojeva često se mijenja i korak (engl. *stride*). Na slici 2.2 korak je 1, a definira za koliko se ćelija horizontalno i vertikalno pomiče jezgra. U slučaju da povećamo korak, izlaz konvolucije bio bi manjih dimenzija, a time bi se i ubrzao postupak računanja izlaza.



Slika 2.4: Primjer konvolucije s korakom 2.

Slika 2.4 prikazuje konvoluciju s korakom 2. Za razliku od standardnog kretanja jezgre, ovdje se jezgra nakon svakog izračuna pomiče za 2 ćelije.

Osim samih konvolucijskih slojeva, konvolucijske mreže u sebi sadrže i slojeve sažimanja. Najčešći razlog za upotrebu slojeva sažimanja je smanjivanje dimenzija podataka, a time i skraćivanje vremena potrebnog za učenje mreže.

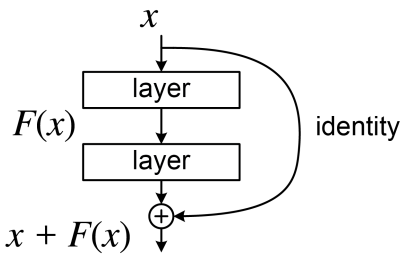


Slika 2.5: Primjer sažimanja maksimalnom vrijednošću. Preuzeto iz [2].

Slika 2.5 prikazuje primjer korištenja sloja sažimanja maksimalnom vrijednošću. Mapa značajki dimenzija 4x4 korištenjem sažimanja maksimalnom vrijednošću 2x2 (engl. *2x2 max pooling*) smanjena je na dimenzije 2x2, efektivno smanjujući dimenzije ulaza za faktor 2. Uz sažimanje maksimalnom vrijednošću, veoma je popularno i sažimanje usrednjavanjem, no postoje i brojne druge varijante koje se koriste za slojeve sažimanja. Kao i operacija konvolucije, i slojevi sažimanja doprinose translacijskoj invarijantnosti, osiguravajući da mreža lako može prepoznati neku značajku bez obzira na njenu točnu lokaciju.

2.8. Rezidualne mreže

Rezidualne mreže [5] vrsta su dubokih neuronskih mreža koje koriste rezidualne blokove. Općenito govoreći, blok u kontekstu neuronskih mreža označava niz od nekoliko slojeva. Pojedini blokovi mogu se kombinirati kako bi sačinjavali složeniju mrežu. Rezidualni blokovi najčešće se sastoje od nekoliko konvolucijskih slojeva, a njihova glavna karakteristika postojanje je preskočnih veza.



Slika 2.6: Primjer rezidualnog bloka. Preuzeto iz [5].

Slika 2.6 prikazuje rezidualni blok sačinjen od dva sloja. Preskočna veza ulaz u blok bez ikakvih transformacija prenosi na izlaz. Ovakvim strukturiranjem mreže, cilj mreže postaje modelirati rezidualnu funkciju $F(x)$ koja mjeri razliku izlaza naspram ulaza. Rezidualni blok možemo prikazati jednadžbom:

$$f(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{x} \tag{2.12}$$

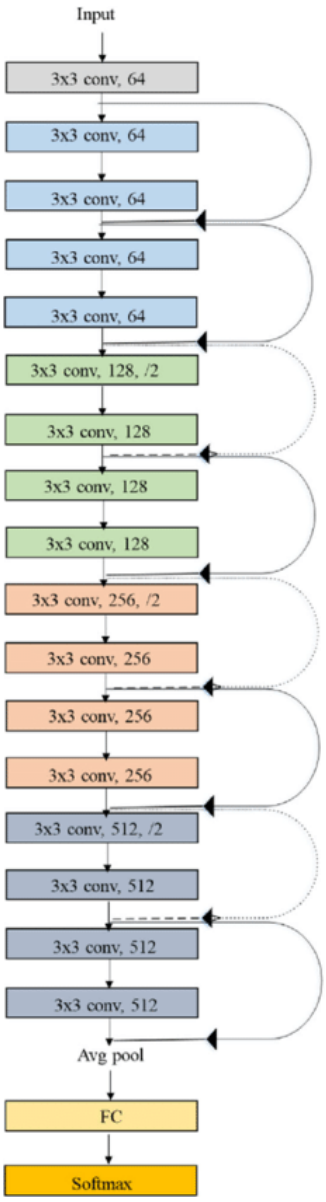
Pritom $F(x)$ predstavlja rezidualnu funkciju, x prenošenje ulaza na izlaz preskočnom vezom, a $f(x)$ izlaz bloka. U slučaju da slojevi između mijenjaju dimenzije podataka, preskočna veza morat će raditi linearnu projekciju kako bi podatci pri zbrajanju bili istih dimenzija. Tada rezidualni blok možemo prikazati na sljedeći način:

$$f(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{W}_s \cdot \boldsymbol{x} \tag{2.13}$$

Pritom W_s označava matricu korištenu za linearnu projekciju ulaza x. Korištenje slojeva s preskočnim vezama motivirano je željom za učenjem funkcije identiteta. Klasične duboke mreže s velikim brojem slojeva teško će naučiti funkciju identiteta, dok korištenje preskočnih veza dubokim mrežama značajno olakšava učenje te funkcije. Ovo potječe od činjenice da je za uspješno modeliranje funkcije identiteta $f(x)$ za rezidualnu funkciju $F(x)$ potrebno samo postaviti težine jezgri na 0.

2.9. ResNet-18

U okviru ovog rada, za provođenje svih eksperimenata koristit ćemo ResNet-18 mrežu kako bismo što bolje mogli usporediti učinak različitih eksperimenata. ResNet-18 rezidualna je neuronska mreža, a sastoji se od ukupno 18 slojeva.



Slika 2.7: Arhitektura ResNet-18. Preuzeto iz [14].

Kao što možemo vidjeti na slici 2.7, ResNet-18 sastoji se od 8 rezidualnih blokova. Svaki od tih blokova sastoji se od 2 konvolucijska sloja i jedne preskočne veze. Rezidualnim blokovima prethodi jedan konvolucijski sloj, a nakon njih dolazi jedan potpuno-povezani sloj sa softmax prijenosnom (aktivacijskom) funkcijom.

3. Brzo učenje s neprijateljskim primjerima

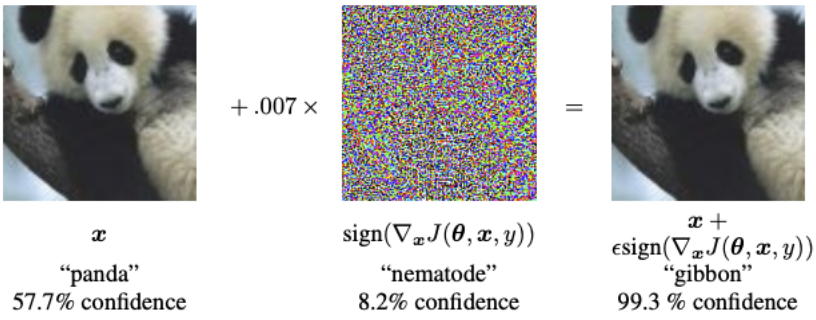
3.1. Neprijateljski primjeri

Kada učimo modele da rješavaju razne probleme, cilj nam je da modeli pokažu svojstvo generalizacije. U slučaju problema klasifikacije, to znači da bi modeli trebali dobro klasificirati i do sada neviđene primjere. Mogućnost generalizacije određenog modela najčešće provjeravamo koristeći unakrsnu provjeru (engl. *cross-validation*). Pritom skup podataka dijelimo na 2 ili 3 skupa:

1. skup za učenje (engl. *train set*)
2. skup za provjeru (engl. *validation set*)
3. skup za testiranje (engl. *test set*)

Skup za učenje modelu predočavamo kako bi na temelju njega mogao poboljšavati svoje parametre tj. učiti. S druge strane, skup za provjeru koristimo za poboljšavanje hiperparametara modela (npr. stopa učenja modela). U slučaju da smo sigurni u dobar odabir hiperparametara modela, korištenje ovog skupa nije nužno. Nakon postupka učenja, koristeći skup za testiranje evaluiramo model kako bismo dobili procjenu njegove generalizacijske pogreške.

Općenito govoreći, model će na skupu za testiranje postizati lošije rezultate naspram rezultata na skupu za učenje. Ako model dobro generalizira, razlika između rezultata na ta dva skupa neće biti veoma značajna. Ipak, u zadnjih desetak godina otkriveni su načini za konstrukciju ulaznih slika koje će zadani model vrlo loše klasificirati iako na većini prirodnih slika dobro generalizira. Takve ulaze zvat ćemo neprijateljski primjeri [4].



Slika 3.1: Primjer neprijateljskog primjera. Preuzeto iz [4].

Slika 3.1 prikazuje jedan neprijateljski primjer. Originalni ulaz kojeg model ispravno klasificira izmijenjen je za iznos perturbacije. U ovom slučaju iznos perturbacije dobiven je kao umnožak malog pozitivnog koeficijenta ϵ i predznaka gradijenta gubitka po ulazu. Iako je izmijenjeni ulaz ljudima gotovo identičan, model ovaj ulaz krivo klasificira, a pritom je veoma siguran u svoju odluku. Ovo svojstvo iskazali su brojni modeli te je zbog toga u zadnjih desetak godina porastao interes za proučavanjem neprijateljskih primjera.

3.2. Konstrukcija neprijateljskih primjera

Neprijateljske primjere moguće je konstruirati na različite načine. Načine konstrukcije neprijateljskih primjera zovemo napadima, a možemo ih podijeliti na usmjerene i neusmjerene napade. Usmjereni napadi kao cilj imaju konstrukciju neprijateljskih primjera koje će napadnuti model klasificirati u točno jedan ciljni razred, dok neusmjereni napadi kao cilj imaju konstrukciju neprijateljskih primjera koje će napadnuti model što lošije klasificirati, neovisno o konkretnom razredu. Pronalaženje neprijateljskih primjera zapravo je optimizacijski problem koji možemo prikazati sljedećim izrazom:

$$\underset{\delta \in [-\epsilon, \epsilon]}{max} L(x + \delta, y; \theta) \tag{3.1}$$

Pritom δ označava iznos perturbacije u intervalu određenim s granicama $[-\epsilon, \epsilon]$, $L(x, y; \theta)$ funkciju gubitka, a θ parametre modela. Drugim riječima, cilj nam je naći perturbaciju omeđenu određenim iznosom takvu da je iznos funkcije gubitka za ulaz izmijenjen tom perturbacijom maksimalan. Iznos perturbacije najčešće je omeđen L_∞-normom, no može biti omeđen i L_2-normom ili L_1-normom.

U slučaju vektorskog prostora, L_∞-norma bit će jednaka iznosu najveće apsolutne vrijednosti komponente vektora, L_2-norma euklidskoj udaljenosti vektora od ishodišta, a L_1-norma sumi iznosa apsolutnih vrijednosti svih komponenti vektora. Općenito govoreći, L_p normu možemo definirati sljedećom formulom:

$$\|x\|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p} \tag{3.2}$$

Formula 3.2 vrijedi za proizvoljan broj p veći od 0. Ipak, L_∞-normu uobičajeno definiramo zasebnom formulom:

$$\|x\|_\infty = \max_i(|x_i|) \tag{3.3}$$

Postoje brojni načini konstrukcije neprijateljskih primjera. Neki od najpoznatijih napada poznati su pod nazivima FGSM (engl. *Fast Gradient Sign Method*) [4] i PGD (engl. *Projected Gradient Descent*) [10] napadi. Primjer sa slike 3.1 zapravo je neprijateljski primjer konstruiran napadom FGSM, a tako dobivene neprijateljske primjere sažeto možemo opisati jednadžbom:

$$x_{adv} := x + \epsilon \cdot sign(\nabla_x L(x, y; \theta)) \tag{3.4}$$

Pritom ϵ predstavlja mali pozitivan broj koji određuje veličinu perturbacije, x ulazni podatak, $sign(x)$ predznak od x, a x_{adv} konstruirani neprijateljski primjer. Napad FGSM originalnom ulazu pridodaje iznos istog predznaka kao i gradijent funkcije gubitka. Na taj način osigurava se najveći mogući rast iznosa funkcije gubitka. Ovakav napad primjer je neusmjerenog napada. U slučaju usmjerenog napada, cilj bi nam bio što više smanjiti iznos funkcije gubitka s obzirom na proizvoljan razred.

Uz napad FGSM, postoji i napad koji iterativno stvara značajno učinkovitije neprijateljske primjere. Tu vrstu napada zovemo napad PGD (projicirani gradijentni spust). Za razliku od FGSM-a, napad PGD iterativno ponavlja postupak, time postižući gradijentni spust. Na kraju svake iteracije, trenutna perturbacija projicira se na L_p loptu definiranu unaprijed zadanim koeficijentom ϵ. Ovim postupkom pokušavamo odrediti najbolje rješenje optimizacijskog problema 3.1.

Algoritam 1 Konstrukcija neprijateljskih primjera napadom PGD

Ulaz: x – ulazne slike, y – ispravne oznake ulaznih slika, ϵ – ograničenje perturbacije, α – veličina koraka po iteraciji, K – broj iteracija za konstrukciju neprijateljskih primjera

Izlaz: x_{adv} – neprijateljski primjeri konstruirani napadom PGD

$\delta = U(-\epsilon, \epsilon)$

for $(i = 1 \dots K)$ **do**

  $grad = \nabla_\delta L(x + \delta, y; \theta)$

  $\delta = \delta + \alpha \cdot sign(g)$

  $\delta = clamp(\delta, -\epsilon, \epsilon)$

end for

$x_{adv} = x + \delta$

return x_{adv}

Pseudokod 1 prikazuje način konstrukcije neprijateljskih primjera PGD-om. Perturbacija δ na početku je inicijalizirana pomoću uniformne distribucije s rasponom $[-\epsilon, \epsilon]$, a na kraju svake iteracije iznos perturbacije ograničava se koristeći funkciju *clamp* kojom postižemo projiciranje na L_∞ loptu. Naravno, postoje brojni načini inicijalizacije iznosa perturbacije, no nasumična inicijalizacija pokazala se dosta uspješnom. Važno je istaknuti da pseudokod za ograničavanje iznosa perturbacije koristi L_∞-normu. Napad PGD s K iteracija za konstrukciju neprijateljskih primjera zvat ćemo K-PGD (npr. 20-PGD).

3.3. Učenje s neprijateljskim primjerima

Ako želimo da modeli koje učimo budu otporni na neprijateljske primjere tj. ispravno ih klasificiraju, ne možemo se ograničiti na klasično (prirodno) učenje modela. Kao što je već rečeno, svojstvo otpornosti na neprijateljske primjere zvat ćemo robusnost iako se u literaturi ovaj naziv zna koristiti i u drugim kontekstima. Jedan od mogućih načina za postizanje robusnih modela učenje je s neprijateljskim primjerima (robusno učenje). Kada govorimo o robusnom učenju, općenita ideja je da se tijekom učenja konstruiraju neprijateljski primjeri prilagođeni samome modelu. Model tada ne uči na originalnim ulaznim podatcima, već isključivo na konstruiranim neprijateljskim primjerima. U tom slučaju neprijateljske primjere možemo smatrati načinom augmentacije ulaznih podataka.

Robusnim učenjem možemo postići puno višu točnost modela na skupu za testiranje sačinjenom od neprijateljskih primjera naspram modela učenih prirodnim učenjem. Nažalost, robusno učenje ima svoju cijenu. Robusni modeli gotovo uvijek će na originalnom skupu za testiranje imati nižu točnost naspram modela učenih prirodnim učenjem, a i učit će se dulje. Ako za konstrukciju neprijateljskih primjera tijekom robusnog učenja koristimo metodu FGSM, učenje će trajati otprilike dvostruko dulje naspram prirodnog učenja jer je za konstrukciju neprijateljskih primjera metodom FGSM potreban dodatan unaprijedni prolaz kroz mrežu. Ako za konstrukciju neprijateljskih primjera koristimo metodu PGD, duljina trajanja učenja ovisit će o broju iteracija korištenih za konstrukciju neprijateljskih primjera. Koristimo li 10 iteracija za konstrukciju neprijateljskih primjera, učenje će trajati otprilike 11 puta dulje naspram prirodnog učenja. Općenito govoreći, kako bismo model učili K-PGD-om, učenje će trajati otprilike K + 1 puta dulje naspram prirodnog učenja [15].

Algoritam 2 Učenje s neprijateljskim primjerima

Ulaz: x – ulazne slike, y – ispravne oznake ulaznih slika, N – broj epoha, M – veličina skupa za učenje, K – broj iteracija za konstrukciju neprijateljskih primjera, η – stopa učenja, ϵ – ograničenje perturbacije, α – veličina koraka po iteraciji

---

$\theta = initializeModelParams()$
for $(ep = 1 \dots N)$ **do**
  $grad = 0$
  for $(i = 1 \dots M)$ **do**
    $x_{adv} = generateAdversarial(x_i, y_i, \epsilon, \alpha, K)$
    $grad \mathrel{+}= \nabla_\theta L(x_{adv}, y_i; \theta)$
  end for
  $\theta = \theta - \eta \cdot grad$
end for

Pseudokod 2 prikazuje općeniti algoritam za učenje s neprijateljskim primjerima. Pritom inicijalizaciju parametara modela θ prikazujemo pozivom funkcije *initializeModelParams*. Radi općenitosti, konstrukcija neprijateljskih primjera prikazana je pozivom funkcije *generateAdversarial*. Bitno je istaknuti da se u konkretnim implementacijama funkcija *generateAdversarial* koristi za konstruiranje neprijateljskih primjera na temelju mini-grupe. U tom slučaju, i izračun gradijenta primjenjuje se na cijelu mini-grupu.

3.4. Brzo učenje s neprijateljskim primjerima

U slučaju da model koji učimo nije veoma složen, obično učenje s neprijateljskim primjerima prihvatljiv je način za postizanje robusnih modela otpornih na napade. Ipak, ako učimo složene modele na velikim skupovima podataka, obično učenje s neprijateljskim primjerima presporo je. Uz to, rijetko tko uopće posjeduje dovoljnu količinu računalnih resursa za robusno učenje složenijih modela. Kako bismo riješili ili barem umanjili ovaj problem, predložena su brojna rješenja čijom bi se primjenom ubrzalo robusno učenje. Razmotrit ćemo tri takva rješenja.

3.4.1. "Besplatno" učenje na neprijateljskim primjerima

Jedno od prvih predloženih rješenja za problem duljine trajanja učenja robusnih modela naziva se "besplatno" učenje (engl. *free adversarial training*) [15]. Uočeno je da modeli učeni PGD-om imaju visoku robusnost, no, kao što je već rečeno, zahtijevaju puno vremena za učenje.

Osnovna ideja "besplatnog" učenja je sljedeća: unatražnim prolazom izračunamo gradijent po ulazu potreban za konstrukciju neprijateljskih primjera, kao i gradijent po parametrima modela. Kako bismo iterativno konstruirali što kvalitetnije neprijateljske primjere, tijekom učenja modelu na ulaz nekoliko puta dovodimo istu mini-grupu prije nego što pređemo na sljedeću. Ponavljanjem mini-grupe nekoliko puta zaredom pokušavamo postići postupak sličan PGD-u, a da pritom ne platimo veliku dodatnu cijenu. Hiperparametar kojim ugađamo broj ponavljanja mini-grupe zovemo ponavljanje (engl. *replay*). Važno je uočiti da, kako bismo i dalje htjeli imati otprilike jednako vremena za učenje modela, moramo smanjiti broj epoha za faktor ponavljanja. Ako svaku mini-grupu unutar jedne epohe ponavljamo 8 puta, broj epoha bit će 8 puta manji naspram broja epoha kod modela učenog prirodnim učenjem. Nažalost, povećavanjem broja ponavljanja dolazi do degradacije performansi modela na običnom testnom skupu - u pitanju je kompromis između željene razine robusnosti i točnosti na običnom testnom skupu.

Algoritam 3 "Besplatno" učenje s neprijateljskim primjerima. Prilagođeno iz [17]

Ulaz: x – ulazne slike, y – ispravne oznake ulaznih slika, N – broj epoha, M – veličina skupa za učenje, K – broj ponavljanja (engl. *replay*), η – stopa učenja, ϵ – ograničenje perturbacije

---

$\theta = initializeModelParams()$
$\delta = 0$
for $(ep = 1 \dots N/K)$ **do**
  $grad = 0$
  for $(i = 1 \dots M)$ **do**
    for $(j = 1 \dots K)$ **do**
      $gradAdv = 0$
      $gradAdv, grad \mathrel{+}= \nabla_{x_i,\theta} L(x_i + \delta, y_i; \theta)$
      $\delta = \delta + \epsilon \cdot sign(gradAdv)$
      $\delta = clamp(\delta, -\epsilon, \epsilon)$
    end for
  end for
  $\theta = \theta - \eta \cdot grad$
end for

Pseudokod 3 prikazuje algoritam za "besplatno" učenje s neprijateljskim primjerima. Kao i u prethodnom algoritmu, parametri modela θ inicijaliziraju se pozivom funkcije *initializeModelParams*. Važno je uočiti da se gradijenti potrebni za konstrukciju neprijateljskih primjera, ali i izmjenu parametara modela računaju u istom unatražnom prolazu. Ovime osiguravamo da konstrukcija neprijateljskih primjera ne umanjuje broj iteracija učenja. Uz to, perturbacija δ inicijalizira se na 0 samo na početku postupka učenja - kada započinje izračun perturbacije za sljedeći ulaz, prethodni iznos služi kao "iznos za zagrijavanje" (engl. *warmup*).

Ovakvim načinom robusnog učenja, duljina trajanja učenja modela otprilike je jednaka kao i duljina trajanja učenja modela prirodnim učenjem. Uz to, rezultati rada [15] pokazuju da je robusnost naučenih modela usporediva s robusnošću modela dobivenih "besplatnim" učenjem. Kao što je već rečeno, visoku robusnost modela učenih "besplatnim" učenjem možemo postići izmjenom broja ponavljanja, ali po cijeni smanjenja performansi modela na običnom testnom skupu.

3.4.2. Brzo učenje s neprijateljskim primjerima

Za modele učene FGSM-om dugo se smatralo da nisu otporni na neprijateljske primjere konstruirane iterativnim metodama. Takvi modeli nisu nam veoma korisni jer često imaju nižu točnost na običnim podatcima, a ne nude pravu mjeru robusnosti - napadač jednostavno može konstruirati neprijateljske primjere koje će model krivo klasificirati. Čini se da se ovaj problem može ublažiti pažljivom inicijalizacijom perturbacije [17]. Standardno se perturbacija za metodu FGSM s L_∞ ograničenjem inicijalizira na 0 ili na iznos koeficijenta ϵ, bilo s pozitivnim ili negativnim predznakom. Međutim, pokazuje se da ako za inicijalizaciju iskoristimo nasumičnu inicijalizaciju, učenje FGSM-om postiže znatno bolju robusnost na iterativne napade. Pritom se za nasumičnu inicijalizaciju koristi uniformna distribucija s granicama $[-\epsilon, \epsilon]$,

Otkriće da se visoka mjera robusnosti može postići i FGSM-om s nasumičnom inicijalizacijom perturbacije korišteno je u tzv. brzom učenju (engl. *fast adversarial training*) [17]. Uz ovo, u brzom učenju koriste se i neke od mogućih optimizacija poput korištenja cikličke stope učenja i računanja u mješovitoj preciznosti. Predložene optimizacije omogućavaju značajno ubrzanje učenja modela, a nisu implementacijski zahtjevne. Ciklička stopa učenja [16] označava promjenu stope učenja kroz epohe ili iteracije - nakon svakog koraka, stopa učenja će se povećavati ili smanjivati unutar unaprijed definiranih granica. Uvođenje cikličke stope učenja korisno je za brže konvergiranja modela, time ubrzavajući učenje. Uz cikličku, postoje i brojne druge poput linearne i kosinusne, no brzo učenje koristi upravo cikličku stopu učenja.

Razvoj tenzorskih jezgri grafičkih kartica omogućilo je korištenje računanja u mješovitoj preciznosti - umjesto da se svi izračuni obavljaju koristeći 32-bitne brojeve s pomičnim zarezom, neki izračuni obavljaju se koristeći 16-bitne brojeve. Mješovita preciznost može značajno ubrzati učenje, kao i smanjiti potrebnu količinu memorije potrebnu za učenje modela. Da bismo mješovitu preciznost mogli koristiti bez pada performansi modela, grafičke kartice na kojim učimo modele trebale bi imati tenzorske jezgre [12].

Algoritam 4 Brzo učenje s neprijateljskim primjerima. Prilagođeno iz [17]

Ulaz: x – ulazne slike, y – ispravne oznake ulaznih slika, N – broj epoha, M – veličina skupa za učenje, η – stopa učenja, ϵ – ograničenje perturbacije, α – veličina koraka

---

$\theta = initializeModelParams()$
for $(ep = 1 \dots N)$ **do**
  $grad = 0$
  for $(i = 1 \dots M)$ **do**
    $\delta = U(-\epsilon, \epsilon)$
    $gradAdv = \nabla_\delta L(x_i + \delta, y_i; \theta)$
    $\delta = \delta + \alpha \cdot sign(gradAdv)$
    $\delta = clamp(\delta, -\epsilon, \epsilon)$
    $grad \mathrel{+}= \nabla_\theta L(x_i + \delta, y_i; \theta)$
  end for
  $\theta = \theta - \eta \cdot grad$
end for

Pseudokod 4 prikazuje algoritam za brzo učenje s neprijateljskim primjerima. Kao i prije, parametri modela θ inicijaliziraju se pozivom funkcije *initializeModelParams*. Početni iznos perturbacije δ inicijalizira se pomoću uniformne distribucije s rasponom $[-\epsilon, \epsilon]$. Za razliku od "besplatnog" učenja, kod brzog učenja potrebna su dva unaprijedna prolaza, kao i dva zasebna izračuna iznosa gradijenta za svaki ulaz. Zbog ovoga brzo učenje nije jednako brzo kao i prirodno učenje modela, ali zato nudi određenu mjeru robusnosti.

Autori brzog učenja tvrde da njihova metoda nudi gotovo jednaku mjeru robusnosti protiv iterativnih napada kao i modeli učeni PGD-om, ali sa značajno kraćim vremenom učenja [17]. Čak i ako već navedene optimizacije (korištenje cikličke stope učenja i računanja u mješovitoj preciznosti) primijenimo na "besplatno" učenje i učenje PGD-om, da bismo postigli jednaku mjeru robusnosti modele je potrebno učiti dulje nego modele učene metodom brzog učenja.

Nažalost, korištenje brzog učenja s neprijateljskim primjerima ima i svoju manu. U slučaju da modele učimo velik broj epoha, u jednom trenutku gotovo sigurno će doći do značajnog pada u točnosti na neprijateljskim primjerima. Ovu pojavu zovemo katastrofalna prenaučenost (engl. *catastrophic overfitting*), a jedan od mogućih načina za sprječavanje te pojave je rano zaustavljanje učenja (engl. *early stopping*). Kako bismo na vrijeme mogli zaustaviti učenje modela, nakon svake epohe evaluiramo točnost modela na neprijateljskim primjerima konstruiranim iz nasumične mini-grupe. U slučaju da je točnost u trenutnoj epohi značajno manja od točnosti u prethodnoj epohi, zaustavljamo učenje modela i kao najbolji model vraćamo model iz prethodne epohe. Zbog ovog svojstva, korištenje brzog učenja s neprijateljskim primjerima nije prikladno za modele koje trebamo učiti velik broj epoha.

3.4.3. Nadogradnje brzog učenja s neprijateljskim primjerima

Kako bi brzo učenje s neprijateljskim primjerima bilo primjenjivo za proizvoljno dugo učenje, potrebno je riješiti problem katastrofalne prenaučenosti. Taj problem pojavljuje se kod svih postupaka koji uče FGSM-om, no manje je izražen kod brzog učenja nego kod običnog učenja FGSM-om. Rad [8] otkriva nam da brzo učenje u manjoj mjeri pati od problema katastrofalne prenaučenosti zbog nasumične inicijalizacije perturbacije. Kao i kod modela učenih običnim FGSM-om, i modeli učeni brzim učenjem susreću se s katastrofalnom prenaučenošću, ali se od nje u periodu od nekoliko mini-grupa mogu oporaviti te nastaviti dalje normalno učiti.

Ključ u oporavku leži u nasumičnoj inicijalizaciji koja pomaže u stvaranju kvalitetnih neprijateljskih primjera. Ipak, zbog nasumičnosti nemamo garanciju da će takav postupak uvijek biti dovoljan za oporavak pa nakon određene količine vremena i dalje dođe do katastrofalne prenaučenosti. Ako bismo mogli garantirati da će se model tijekom učenja uvijek moći oporaviti od katastrofalne prenaučenosti u periodu od nekoliko mini-grupa, mogli bismo proizvoljno dugo učiti model. U radu [8] oporavak se postiže kontinuiranim praćenjem točnosti na neprijateljskim primjerima konstruiranim iz nasumične mini-grupe svakih s mini-grupa. U slučaju da je točnost u trenutnom koraku značajno manja od točnosti u prethodnom koraku, do sljedeće provjere točnosti umjesto metode FGSM s nasumičnom inicijalizacijom, za konstrukciju neprijateljskih primjera koristimo metodu PGD. Ovakvim postupkom model se lako može oporaviti od pojave katastrofalne prenaučenosti pa stoga može i dulje učiti. Opisani način učenja nazivamo *FastAdv+* (engl. *Fast Adversarial Training Plus*).

Algoritam 5 *FastAdv+* učenje s neprijateljskim primjerima. Prilagođeno iz [8]

Ulaz: x – ulazne slike, y – ispravne oznake ulaznih slika, N – broj epoha, M – veličina skupa za učenje, η – stopa učenja, ϵ – ograničenje perturbacije, α – veličina koraka, c – prag detekcije, s – frekvencija detekcije

---

$\theta = initializeModelParams()$
$accLast = 0$
$accValid = 0$
for $(ep = 1 \dots N)$ **do**
  $grad = 0$
  for $(i = 1 \dots M)$ **do**
    if $(accLast > accValid + c)$ **then**
      $x_{adv} = PGD(x_i, y_i, \epsilon, \alpha, K)$
    else
      $x_{adv} = RandomFGSM(x_i, y_i, \epsilon, \alpha, K)$
    end if
    $grad \mathrel{+}= \nabla_\theta L(x_{adv}, y_i; \theta)$
    if $(i\%s == 0)$ **then**
      $accLast = accValid$
      $accValid = evaluateRobustness(x_{rand}, y_{rand}, \epsilon, \alpha, K)$
    end if
  end for
  $\theta = \theta - \eta \cdot grad$
end for

Pseudokod 5 prikazuje algoritam *FastAdv+* za učenje s neprijateljskim primjerima. Kako bismo pratili točnost modela na neprijateljskim primjerima, koristimo varijable *accLast* i *accValid* pri čemu prva varijabla predstavlja stariju točnost s kojom uspoređujemo, a druga varijabla točnost iz prethodnog koraka. U slučaju da je u prethodnom koraku točnost pala ispod vrijednosti određene pragom c, učit ćemo PGD-om, a inače ćemo učiti s metodom *RandomFGSM* koja predstavlja FGSM metodu s nasumičnom inicijalizacijom. Nakon provođenja učenja, u slučaju da smo na iteraciji određenoj frekvencijom detekcije s, evaluirat ćemo točnost modela na neprijateljskim primjerima pozivom funkcije *evaluateRobustness*. U originalnome radu [8], za provjeru je korištena metoda PGD, a isto je reproducirano i u ovome radu.

Ovakav način učenja omogućava nam proizvoljno dugo učenje modela, no i dalje

ne dostiže jednaku mjeru robusnosti kao i najnovije varijante učenja PGD-om. Kako bismo dodatno unaprijedili *FastAdv+* učenje, predložena je nadogradnja: zadnjih nekolicinu epoha model ćemo učiti isključivo PGD-om. Ova nadogradnja motivirana je hipotezom da je na početku treniranja modelu dovoljno imati slabije neprijateljske primjere za učenje, dok je za kasnije epohe potrebno koristiti jače neprijateljske primjere kako bi model dodatno mogao učiti. Opisani način učenja zovemo *FastAdvW* (engl. *Fast Adversarial Training Warmup*), a prema rezultatima rada [8] modeli učeni metodom *FastAdvW* postižu bolje rezultate čak i naspram najnovijih varijanta učenja PGD-om, pritom zahtijevajući značajno manje vremena za učenje.

4. Zatrovani podatci

4.1. Općenito o zatrovanim podatcima

Kako bismo u nekom sustavu koristili modele dubokog učenja za postizanje određene funkcionalnosti, modele je prvo potrebno naučiti na velikom skupu podataka. Iako danas često imamo goleme količine podataka koje možemo koristiti za učenje modela, sve podatke trebalo bi provjeravati. U slučaju da koristimo neprovjerene podatke, izlažemo naš sustav brojnim prijetnjama. Jedna od takvih prijetnji ubacivanje je zatrovanih podataka u skup za učenje modela.

Zatrovani podatci namjerno su dizajnirani podatci čiji cilj je zavaravanje sustava. Slični su neprijateljskim primjerima, ali postoje i određene razlike koje se većinom očituju u namjeri. Dok neprijateljske primjere koristimo kako bi za neki konkretan ulaz izmijenili klasifikaciju nakon što je model već naučen, zatrovane podatke ubacujemo u skup za učenje modela kako bismo izmijenili decizijsku granicu modela - ako model učimo na zatrovanom skupu podataka, kada ga nakon učenja koristimo za klasifikaciju, njegove odluke za određene ulaze bit će drugačije nego što bi bile da smo model učili na čistom skupu. Uz to, zatrovani podatci mogu biti ručno dizajnirani, dok neprijateljske primjere većinom konstruiramo koristeći napade zasnovane na gradijentu funkcije gubitka modela po ulazu.

Slika 4.1: Pojednostavljeni primjer zatrovanih podataka. Preuzeto iz [11].

Slika 4.1 prikazuje pojednostavljene primjere zatrovanih podataka. Na originalne slike dodan je bijeli pravokutnik u nadi da će model naučiti prepoznavati takav pravokutnik i na temelju njegove prisutnosti kategorizirati sve ulaze u pojedini razred. Osim modifikacije samih ulaznih slika, radi se i modifikacija ispravnih oznaka: za sve modificirane slike napadač bi mogao postaviti istu oznaku. U slučaju slike 4.1, napadač bi mogao za sve tri slike dodijeliti razred *dog* kao ispravnu oznaku. Ako nakon učenja model na nekoj slici prepozna bijeli pravokutnik, lako je moguće da će ju klasificirati u razred *dog* iako ona zapravo pripada nekom drugom razredu. Naravno, ovaj primjer veoma je pojednostavljen - u stvarnosti izmjene originalnih slika mogu biti veoma suptilne, baš kao i promjene prisutne kod neprijateljskih primjera. Izmijenjeni dio ulaza zvat ćemo okidačem (engl. *trigger*). U slučaju da je na jednostavan način moguće ubaciti podatke u skup za učenje našeg modela, napadači tu ranjivost mogu iskoristiti za razne ciljeve. Općenito govoreći, ubacivanje zatrovanih podataka kao posljedicu može imati pojavu jedne od dviju značajnih mana sustava:

1. Pad točnosti modela
2. Ugradnja stražnjih vrata u model

U prvome slučaju, napadač ubacivanjem zatrovanih podataka želi degradirati performanse modela, time čineći njegov rad nepouzdanijim. U drugome slučaju, napadač pažljivim dizajnom zatrovanih podataka može do određene mjere manipulirati ponašanjem modela. Ako model nauči klasificirati sve zatrovane podatke u određeni razred te nakon učenja kao ulaz dobije novi zatrovani podatak ili podatak koji ima određenu mjeru sličnosti s naučenim okidačem, postoji velika vjerojatnost da će ulaz biti krivo klasificiran.

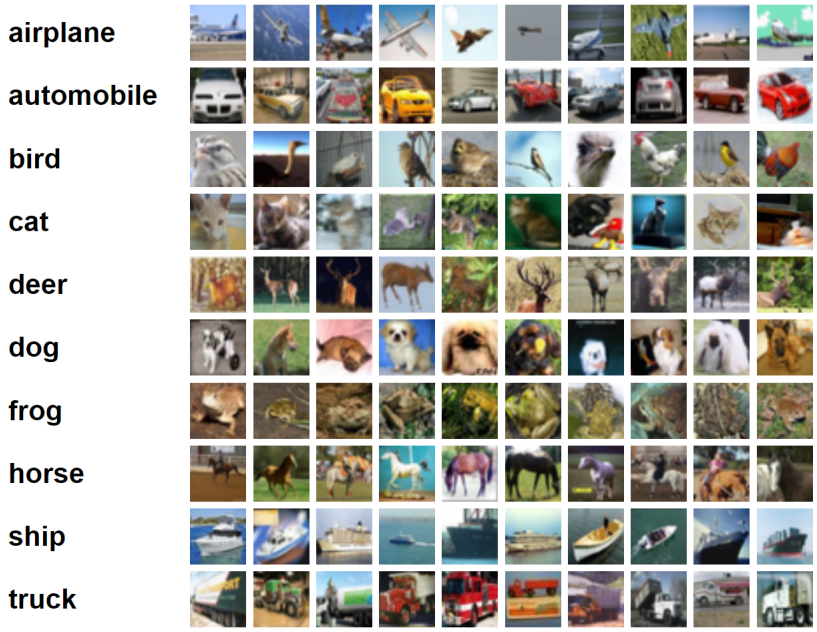
4.2. BackdoorBox

BackdoorBox Toolbox [9] skup je alata otvorenog koda koji služi za edukaciju o napadima čiji je cilj ugradnja stražnjih vrata u modele. Osim napada, u skupu alata postoje i brojne obrane. U okviru ovog rada, koristit ćemo *BadNets* alat iz skupa alata *BackdoorBox* kako bismo konstruirali zatrovane skupove podataka. Konstruirane zatrovane skupove podataka koristit ćemo za evaluaciju otpornosti robusnih modela na zatrovane podatke, kao i učenje novih modela.

5. Eksperimenti

5.1. Skup podataka CIFAR-10

Skup podataka CIFAR-10 [7] sastoji se od 60000 32x32 slika u boji. Svaka slika ima oznaku jednog od 10 razreda, a cijeli skup podataka sastoji se od 6000 slika svakog razreda. Skup podataka podijeljen je na 50000 slika u skupu za učenje i 10000 slika u skupu za testiranje.



Slika 5.1: Primjeri slika i oznaka iz skupa podataka CIFAR-10. Preuzeto iz [7].

Slika 5.1 prikazuje razrede iz skupa podataka CIFAR-10, kao i po 10 nasumično odabranih slika iz svakog razreda. Svi razredi međusobno su isključivi tj. nijedna slika ne pojavljuje se u skupu podataka s više različitih oznaka razreda. Uz CIFAR-10, postoji i skup podataka CIFAR-100 koji sadrži 100 različitih razreda. U ovom radu eksperimente provodimo na skupu CIFAR-10 zbog brzine učenja, no metode korištene u radu lako se mogu primijeniti i na druge skupove podataka.

5.2. Korištene tehnologije

Za provođenje eksperimenata u ovome radu, kod je pisan u programskom jeziku Python. Osim standardnih biblioteka Pythona, korištene su biblioteke NumPy, Matplotlib i radni okvir PyTorch. Sami kod izvršavan je primarno na platformi Kaggle koristeći dvije grafičke kartice NVIDIA T4.

5.2.1. NumPy

NumPy je biblioteka pisana za programski jezik Python. Služi za učinkovito provođenje operacija na matricama i tenzorima, a nudi i implementacije brojnih matematičkih funkcija koje se lako mogu primijeniti na matrice i tenzore. Korištenjem biblioteke NumPy, moguće je značajno ubrzati izvođenje koda vezanog uz neuronske mreže zbog optimiranih implementacija u programskim jezicima niže razine i strojnom kodu, kao i eliminiranja potrebe za petljama u Pythonu.

5.2.2. Matplotlib

Kao i NumPy, i Matplotlib je biblioteka pisana za programski jezik Python. Služi za jednostavno iscrtavanje grafičkih objekata. Osim klasičnih grafova, koristeći Matplotlib moguće je prikazati i slike (npr. slike učitane kao dio skupa podataka CIFAR-10). Uz iscrtavanje, Matplotlib nudi i mogućnost pohranjivanja u raznim grafičkim formatima.

5.2.3. PyTorch

PyTorch je radni okvir pisan za programski jezik Python. Služi za laku izgradnju i učenje dubokih modela, kao i općenito provođenje izračuna s tenzorima. Pruža nam mogućnost automatske diferencijacije veoma značajnu za izračun gradijenta funkcije gubitka, kao i pristup brojnim optimizacijskim algoritmima, ali i slojevima neuronskih mreža koje s lakoćom možemo kombinirati za izgradnju vlastitih mreža.

Uz navedene mogućnosti, radni okvir PyTorch nudi nam podršku za korištenje grafičkih kartica za provođenje izračuna. U slučaju da na računalu imamo prikladnu grafičku karticu, kao i instaliranu prikladnu programsku podršku, koristeći PyTorch lako možemo prebacivati provođenje izračuna s procesora na grafičku karticu koja je uobičajeno specijalizirana za paralelno procesiranje. PyTorch nam također nudi i mogućnost provođenja izračuna sa 16-bitnim brojevima s pomičnim zarezom. Bitno je napomenuti da je za ovo potrebno imati grafičku karticu s tenzorskim jezgrama.

5.3. Brzo učenje s neprijateljskim primjerima

Kako bismo usporedili učinkovitost različitih pristupa brzom učenju s neprijateljskim primjerima, učili smo nekolicinu modela koristeći varirajuće hiperparametre i načine učenja. Svi naučeni modeli neuronske su mreže arhitekture ResNet-18, a učeni su 80 epoha. Jedina iznimka ovome je model učen "besplatnim" učenjem. Taj model učen je ukupno 10 epoha, ali s hiperparametrom ponavljanja iznosa 8. Ovime postižemo jednak broj iteracija učenja svih modela. Kao mjeru gubitka kod svih modela korištena je unakrsna entropija, a kao optimizator korišten je stohastički gradijentni spust sa zamahom (engl. *momentum*) iznosa 0.9 i propadanjem težina iznosa $5 \cdot 10^{-4}$. Korištenje zamaha ubrzava konvergenciju modela, a korištenje propadanja težina regularizacijska je tehnika koja služi za smanjivanje složenosti modela, a time i vjerojatnosti pojave prenaučenosti. Kako bismo što bolje reproducirali originalni rad [17], za brzo učenje s neprijateljskim primjerima (*FastAdv*) korištena je ciklička stopa učenja. Pritom stopa učenja maksimum dosegne na pola koraka učenja te se nakon toga do kraja učenja njen iznos smanjuje. Važno je napomenuti da se kod cikličke stope učenja iznos stope učenja mijenja nakon svake mini-grupe, a odozdo je ograničen iznosom 0. Svi ostali modeli koristili su stopu učenja s kosinusnim kaljenjem. Pritom je maksimalan broj koraka postavljen na ukupan broj epoha, a iznos stope učenja mijenja se nakon svake epohe.

Kako bismo ubrzali eksperimente, sve modele učili smo u mješovitoj preciznosti. Osim modela učenih s neprijateljskim primjerima, za usporedbu je učen i model na prirodnom skupu podataka. Nakon učenja, za sve modele izračunata je točnost na prirodnom skupu za testiranje, ali i točnost na neprijateljskim primjerima konstruiranim iz prirodnog skupa za testiranje PGD-om s 20 iteracija. Pritom je napad PGD kao hiperparametre imao ograničenje perturbacije (ϵ) iznosa $8/255$ te veličinu koraka (α) iznosa $1/255$. Uz točnost, mjereno je i vrijeme potrebno za učenje modela.

U tablici 5.1 stupac *LR* označava osnovnu stopu učenja, stupac *Točnost* točnost na skupu za testiranje, a stupac *20-PGD* točnost na neprijateljskim primjerima konstruiranim iz skupa za testiranje PGD-om s 20 iteracija. Točnosti su prikazane postotkom, a vrijeme učenja prikazano je brojem minuta. Pritom su točnost, ali i vrijeme učenja zaokruženi na jedno decimalno mjesto. U stupcu *Način učenja*, *Natural* predstavlja prirodno učenje, *FreeAdv* "besplatno" učenje, *FastAdv* brzo učenje, *FastAdv+* brzo učenje s povremenim PGD-om, a *FastAdvW* brzo učenje s povremenim PGD-om te PGD-om zadnjih 10 epoha učenja. Ako stupac *Način učenja* ima sufiks *Early*, korišteno je učenje s ranim zaustavljanjem.

Tablica 5.1: Rezultati raznih načina brzog učenja s neprijateljskim primjerima.

Način učenja	LR	Točnost [%]	20-PGD [%]	Vrijeme učenja [min]
Natural	0.01	90.4	0	43.3
Natural	0.02	92.3	0	42.7
PGD-7	0.1	83.4	43.7	202.7
FreeAdv	0.1	84.7	30.8	25
FastAdv	0.2	89.6	0	66.5
FastAdv, Early	0.2	82.2	40.2	59.8
FastAdv+	0.2	85.7	40.7	90.7
FastAdv+, Early	0.2	85.7	41.2	91.5
FastAdvW	0.2	85.1	43.2	115.5
FastAdvW, Early	0.2	85.3	42.8	113.4

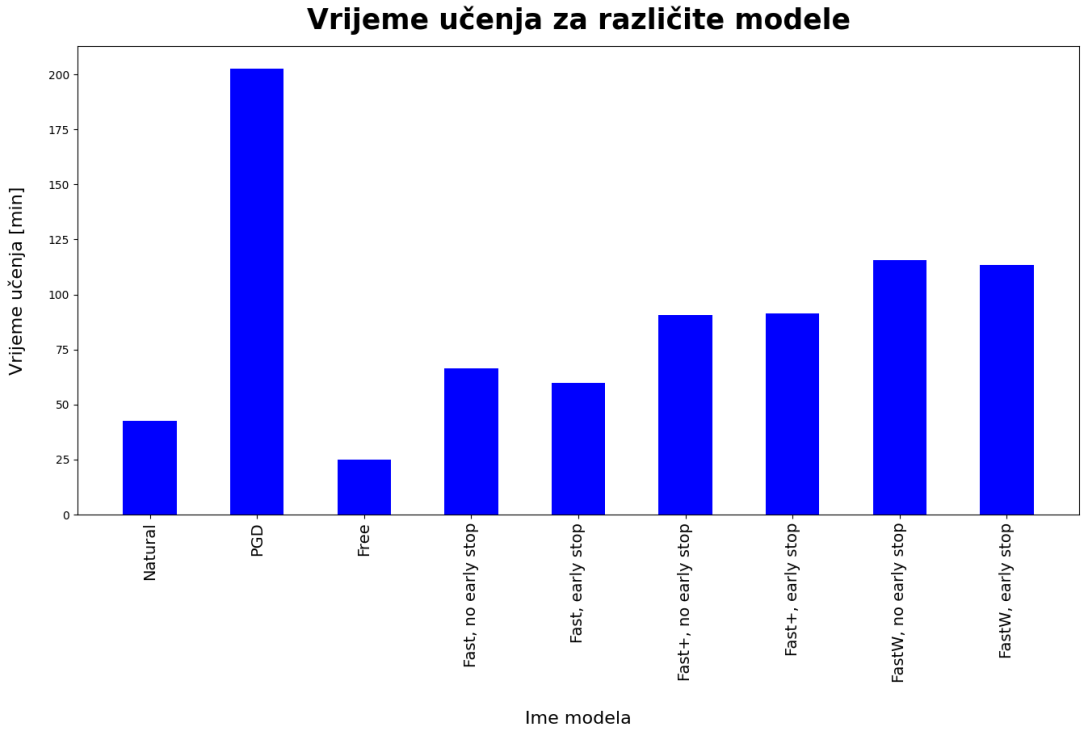
Kao što možemo vidjeti u tablici 5.1, najvišu točnost na prirodnom skupu za testiranje ima model učen na prirodnom skupu podataka. Nažalost, taj model nije nimalo robustan - njegova točnost na neprijateljskim primjerima konstruiranim PGD-om s 20 iteracija iznosi 0%. Kada govorimo o robusnim modelima, najvišu točnost na neprijateljskim primjerima ima model učen PGD-om, no on se i daleko najdulje uči. Pritom su tijekom učenja PGD metodom neprijateljski primjeri konstruirani PGD-om sa 7 iteracija.

Od metoda brzog učenja s neprijateljskim primjerima izloženih u radovima [15], [17] i [8], najbolje performanse ima model učen *FastAdvW* metodom. Njegova točnost na prirodnom skupu za testiranje druga je najviša od svih modela koji nude određenu mjeru robusnosti, a njegova točnost na neprijateljskim primjerima usporediva je s točnosti modela učenog PGD-om. Važno je napomenuti da modeli učeni metodom *FastAdvW* trebaju skoro dvostruko manje vremena za učenje naspram modela učenih PGD-

om, ali se nažalost uče najdulje od svih predloženih varijanti brzog učenja. S obzirom na to da modeli učeni metodom *FastAdvW* zadnjih 10 epoha uče isključivo PGD-om, takav rezultat je i očekivan. Modeli učeni *FastAdv+* metodom dobra su alternativa korištenju modela učenih metodom *FastAdvW* jer se uče kraće i postižu malo bolje rezultate na prirodnom skupu za testiranje. Uočimo da rano zaustavljanje učenja kod modela učenih metodama *FastAdv+* i *FastAdvW* nema nikakav učinak na performanse. Gledajući da je glavni cilj tih metoda rješavanje problema katastrofalne prenaučenosti prisutnog kod modela učenih metodom *FastAdv*, ovakav rezultat je očekivan, a govori nam da ove nadogradnje stvarno uspijevaju u svojoj namjeri.

U slučaju da nam je glavni cilj brzo naučiti model koji je barem u nekoj mjeri otporan na neprijateljske primjere, kao najbolja opcija ističe se učenje metodom *FreeAdv*. Kao što je već rečeno, model učen metodom *FreeAdv* učen je koristeći ponavljanje (engl. *replay*) iznosa 8 pa je stoga ukupno učen 10 epoha - ukupan broj epoha smanjen je za faktor jednak ponavljanju. Ovaj model nudi nam konkurentnu točnost na prirodnim podatcima, ali je njegova točnost na neprijateljskim primjerima niža od svih ostalih robusnih modela. Nižu točnost na neprijateljskim primjerima tumačimo kao posljedicu učenja na neprijateljskim primjerima nedovoljne kvalitete tj. jačine. U slučaju da dodatno povećamo iznos parametra ponavljanja, točnost na neprijateljskim primjerima povećala bi se. Naravno, istovremeno bi došlo do pada točnosti na prirodnim podatcima. Dobra strana metode *FreeAdv* svakako je vrijeme učenja - ovaj model uči se kraće čak i od modela učenog na prirodnom skupu podataka. Toliku kratkoću učenja pripisujemo tome da je tijekom učenja metodom *FreeAdv* provedena evaluacija nakon epoha samo 10 puta, dok je za ostale metode evaluacija provedena punih 80 puta. Ovo proizlazi iz činjenice da modele učene metodom *FreeAdv* učimo samo 10 epoha, dok ostale modele učimo 80 epoha. Pritom evaluaciju izvodimo nakon svake epohe.

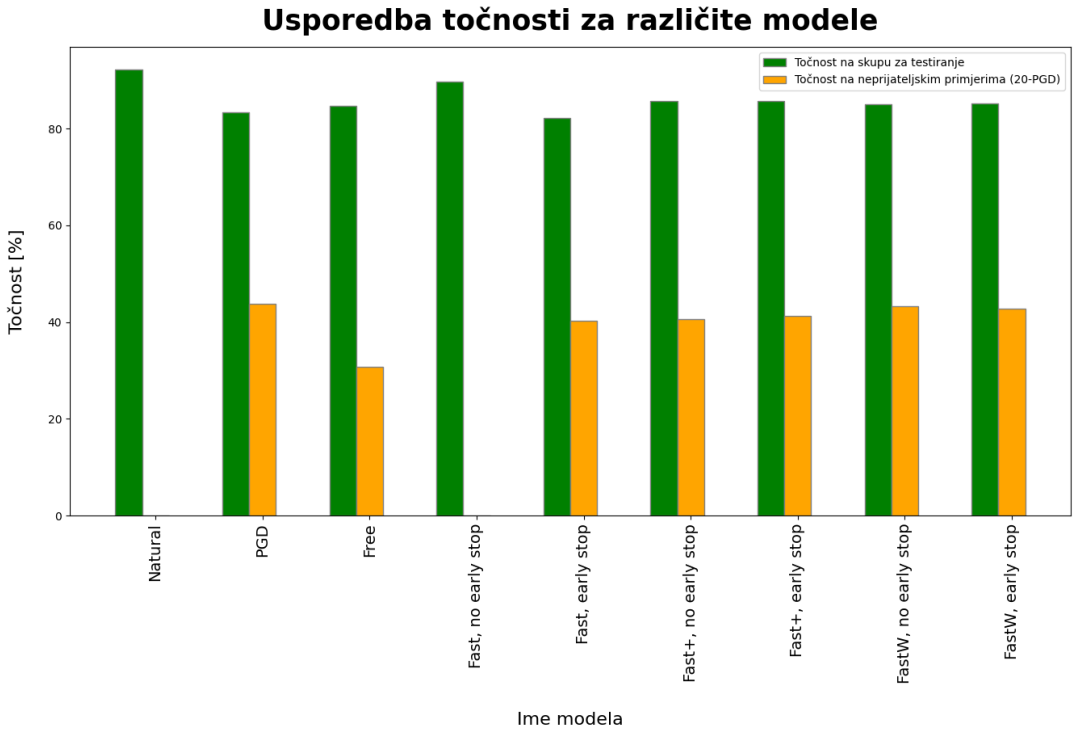
Modeli učeni metodom *FastAdv* ograničeni su pojavom katastrofalne prenaučenosti. Ovo svojstvo veoma je vidljivo kod modela učenih metodom *FastAdv* bez ranog zaustavljanja - takav model postiže visoku točnost na prirodnom skupu podataka, ali zato nije nimalo otporan na neprijateljske primjere. Korištenjem metode *FastAdv* s ranim zaustavljanjem, točnost na neprijateljskim primjerima podjednaka je s točnosti modela učenih metodom *FastAdv+*, ali je takav model ograničen zbog nemogućnosti proizvoljno dugog učenja.



Slika 5.2: Usporedba vremena učenja raznih načina brzog učenja s neprijateljskim primjerima.

Slika 5.2 prikazuje usporedbu vremena učenja raznih načina brzog učenja s neprijateljskim primjerima. Zbog usporedbe, na grafu je prikazano i vrijeme učenja modela učenog na prirodnim podatcima, kao i vrijeme učenja modela učenog PGD-om. Kao što je bilo vidljivo i iz tablice 5.1, daleko najdulje vrijeme učenja ima model učen PGD-om. S druge strane, najkraće vrijeme učenja ima model učen metodom *FreeAdv*. Ako uspoređujemo modele učene metodom *FastAdv*, *FastAdv+* ili *FastAdvW*, najkraće vrijeme učenja ima model učen metodom *FastAdv* s ranim zaustavljanjem. Gledajući da model učen metodom *FastAdv+* pri uočavanju pada točnosti na neprijateljskim primjerima s sljedećih mini-grupa uči PGD-om, a model učen metodom *FastAdvW* uz to zadnjih 10 epoha uči isključivo PGD-om, ovakav rezultat je i očekivan.

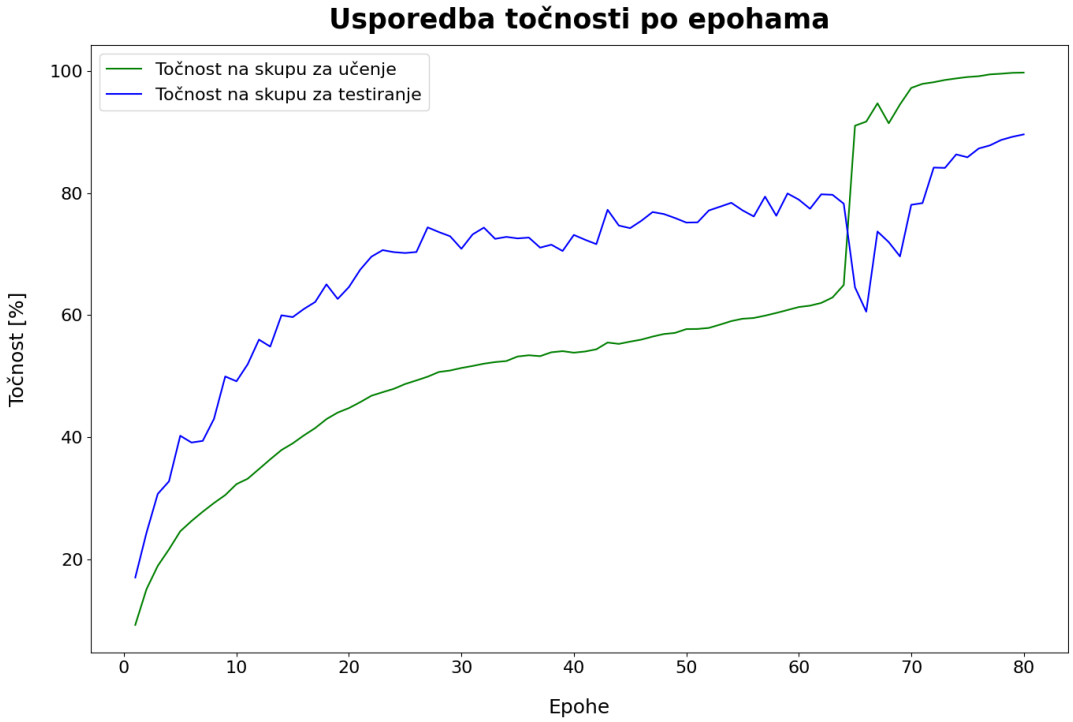
Slika 5.3 prikazuje usporedbu točnosti na skupu za testiranje i točnosti na neprijateljskim primjerima konstruiranim iz prirodnog skupa PGD-om s 20 iteracija. Kao i na prethodnom grafu, i ovdje su prikazane točnosti i za model učen na prirodnim podatcima, kao i za model učen PGD-om. Možemo vidjeti da najvišu točnost na prirodnom skupu imaju model učen na prirodnim podatcima i model učen metodom *FastAdv* kod kojeg je došlo do katastrofalne prenaučenosti. Istovremeno, ta dva modela ne nude nikakvu otpornost na neprijateljske primjere - njihova točnost na neprijateljskim primjerima jednaka je 0%. Ostali modeli imaju podjednake točnosti na prirodnom skupu, ali i na neprijateljskim primjerima. Pritom najvišu točnost na prirodnom skupu imaju



Slika 5.3: Usporedba točnosti raznih načina brzog učenja s neprijateljskim primjerima.

modeli učeni metodom *FastAdv+*, dok najvišu točnost na neprijateljskim primjerima imaju modeli učeni metodom *FastAdvW*. Model učen *Free* metodom ima najnižu točnost na neprijateljskim primjerima, ali zato, kao što smo vidjeli na slici 5.2, ima najkraće vrijeme učenja.

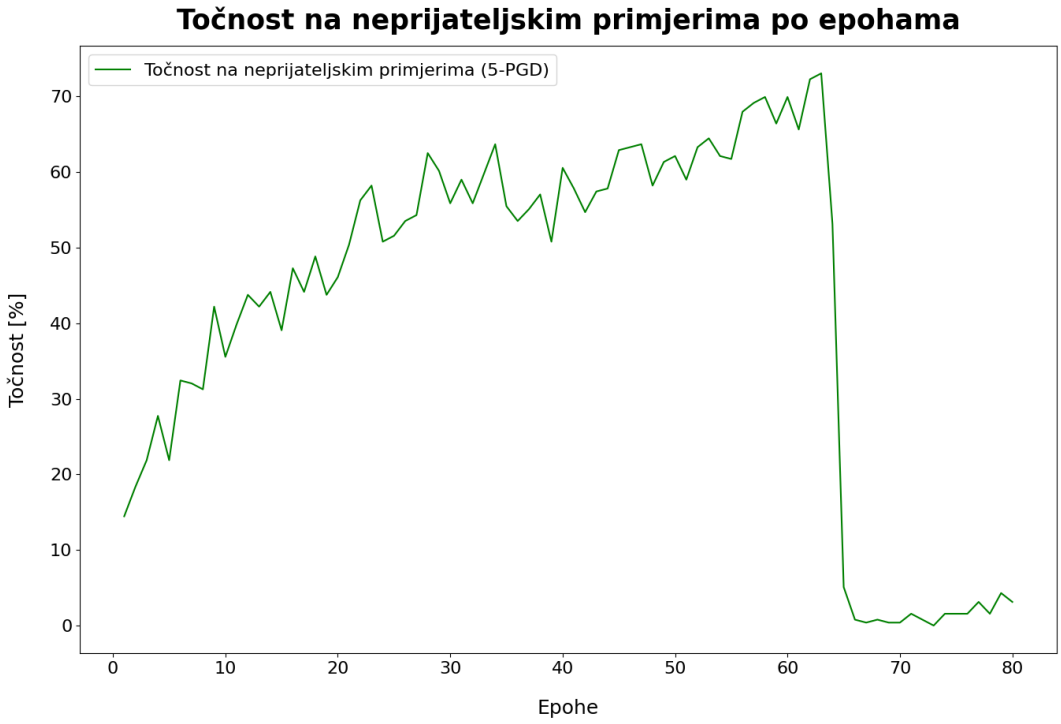
Slika 5.4 prikazuje usporedbu točnosti na skupu za učenje i točnosti na skupu za testiranje po epohama za model učen metodom *FastAdv*. Oko 65. epohe moguće je vidjeti velik porast u točnosti na skupu za učenje, kao i istovremeni pad u točnosti na skupu za testiranje. Ova pojava posljedica je katastrofalne prenaučenosti - model gubi na moći generalizacije i počinje pamtiti ulazne podatke. Kako bismo dodatno provjerili da je to istina, provjerit ćemo i točnost na neprijateljskim primjerima po epohama. U slučaju da oko 65. epohe dolazi do velikog pada u točnosti na neprijateljskim primjerima, u pitanju je katastrofalna prenaučenost od koje pate svi modeli učeni metodama zasnovanim na FGSM-u.



Slika 5.4: Točnost modela učenog metodom *FastAdv* po epohama.

Slika 5.5 prikazuje točnost modela učenog metodom *FastAdv* na neprijateljskim primjerima po epohama. Kako evaluacija tijekom učenja ne bi bila prespora, neprijateljski primjeri konstruirani su iz nasumično odabrane mini-grupe PGD-om s 5 iteracija. Kao što smo očekivali, oko 65. epohe dolazi do velikog pada u točnosti na neprijateljskim primjerima - već epohu nakon točnost na neprijateljskim primjerima iznosi otprilike 0%. Ovime potvrđujemo da je stvarno došlo do katastrofalne prenaučenosti. Kako bismo ovo izbjegli, model učen metodom *FastAdv* možemo učiti s ranim zaustavljanjem. Ipak, rano zaustavljanje ograničava nas jer modele ne možemo proizvoljno dugo učiti. Kao što je već prethodno rečeno, jedno od mogućih rješenja ovog problema korištenje je načina učenja *FastAdv*+ ili *FastAdvW*.

Kako bismo potvrdili da korištenje modela učenog metodom *FastAdv*+ rješava problem katastrofalne prenaučenosti, na slici 5.6 možemo vidjeti usporedbu točnosti na skupu za učenje i točnosti na skupu za testiranje po epohama za model učen metodom *FastAdv*+. Iako se oko 65. epohe može uočiti početak brzog porasta u točnosti na skupu za učenje, kao i istovremeni početak brzog pada u točnosti na skupu za testiranje, već u sljedećih nekoliko epoha točnosti se ispravljaju. Mogućnost modela učenog metodom *FastAdv*+ da se oporavi od pojave katastrofalne prenaučenosti proizlazi iz kontinuiranog praćenja točnosti na neprijateljskim primjerima te korištenja PGD-a za konstrukciju neprijateljskih primjera kada se uoči pad u točnosti.

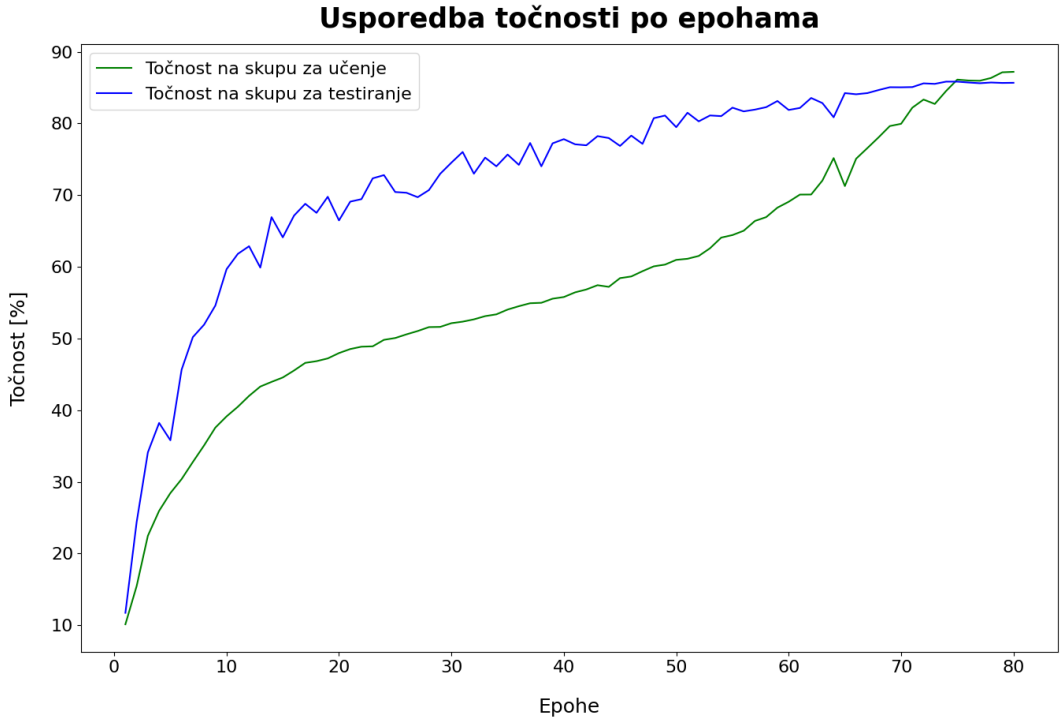


Slika 5.5: Točnost modela učenog metodom *FastAdv* na neprijateljskim primjerima po epohama.

5.4. Prelimarno istraživanje primjene učenja s neprijateljskim primjerima za detekciju zatrovanih podataka

Robusni modeli nude nam određenu mjeru otpornosti na neprijateljske primjere. Ipak, značajniji problem predstavljaju nam zatrovani podatci ubačeni u skup za učenje jer zbog njih model može imati lošije performanse, a i služiti kao stražnja vrata napadačima. Obični, nerobusni modeli tijekom učenja na zatrovanom skupu podataka veoma brzo nauče prepoznavati okidač, time postajući ranjivi. Idealno, modeli učeni s neprijateljskim primjerima pokazivali bi određenu mjeru otpornosti na zatrovane podatke.

Kako bismo provjerili imaju li robusni modeli doista mogućnost detekcije zatrovanih podataka, učili smo modele PGD-om, kao i metodom *FastAdv*. Modeli učeni PGD-om odabrani su zbog visoke otpornosti na neprijateljske primjere, a modeli učeni metodom *FastAdv* odabrani su zbog veoma kratkog trajanja učenja. Uz robusne modele, učeni su i prirodni modeli. Pritom su svi modeli učeni 60 epoha kako bismo izbjegli pojavu katastrofalne prenaučenosti do koje često dolazi u kasnijim epohama učenja modela metodom *FastAdv*.



Slika 5.6: Točnost modela učenog metodom *FastAdv+* po epohama.

Zatrovani skup podataka za učenje, kao i zatrovani skup za testiranje, konstruirani su iz prirodnog CIFAR-10 skupa podataka koristeći *BadNets* alat iz skupa alata *BackdoorBox*. Pritom je za skup za učenje stopa trovanja iznosila 0.3, a za skup za testiranje 1. Stopa trovanja određuje koliki će postotak podataka iz originalnog skupa biti zatrovan. I za skup za učenje i za skup za testiranje kao ciljni razred odabran je razred *automobile*. U slučaju da na ulaznoj slici postoji okidač, model koji ga je tijekom učenja naučio takvu sliku klasificirat će u razred *automobile*. Arhitektura modela, mjera gubitka, optimizator i hiperparametri za učenje jednaki su kao i u prethodnim eksperimentima. Kao i prije, svi modeli osim modela učenih metodom *FastAdv* učeni su koristeći stopu učenja s kosinusnim kaljenjem, dok modeli učeni metodom *FastAdv* koriste cikličku stopu učenja. Osim na zatrovanom skupu podataka, svi modeli učeni su i na prirodnom skupu podataka.

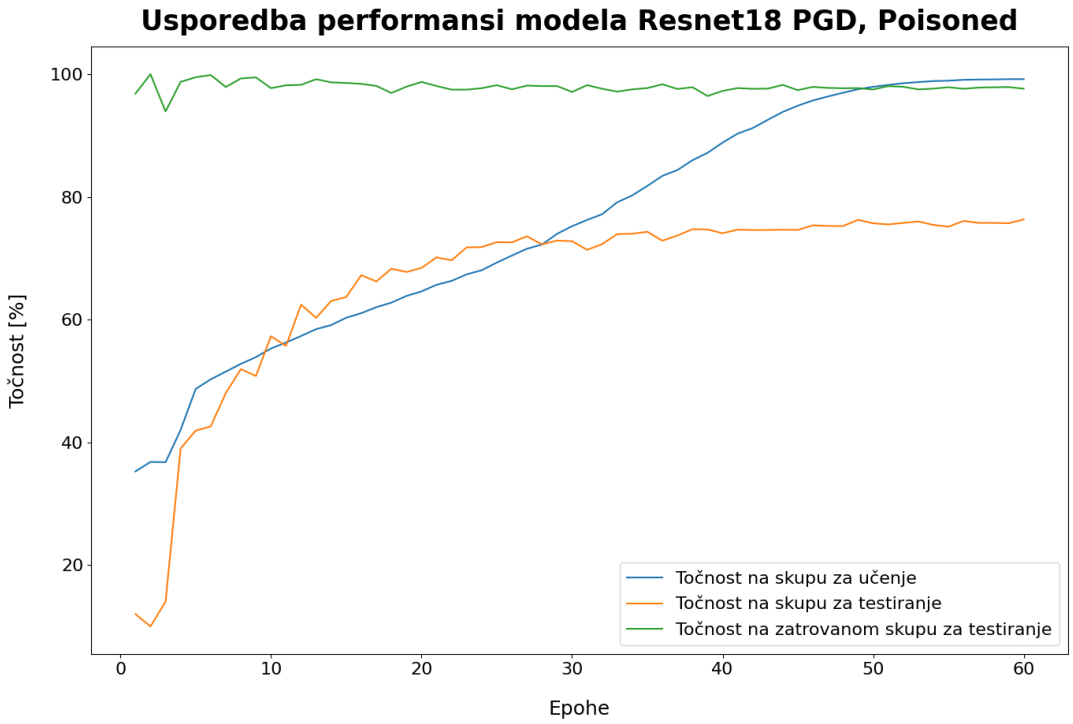
Kao i prije, u tablici 5.2 stupac *Način učenja* označava korišteni način učenja modela. Ako stupac *Način učenja* ima sufiks *Poison*, model je učen na zatrovanom skupu podataka. Stupac *Točnost* ponovno označava točnost na prirodnom skupu podataka, a stupac *Točnost, otrov* označava točnost na zatrovanom skupu podataka. Iznosi točnosti, kao i vremena učenja zaokruženi su na jedno decimalno mjesto.

Kao što možemo vidjeti u tablici 5.2, svi modeli učeni na prirodnom skupu podataka imaju točnost na zatrovanim podatcima iznosa otprilike 10%. Ovaj rezultat

Tablica 5.2: Rezultati raznih načina učenja na zatrovanom i prirodnom skupu podataka.

Način učenja	LR	Točnost [%]	Točnost, otrov [%]	Vrijeme učenja [min]
Natural	0.02	86	10.2	27.1
Natural, Poison	0.02	81.7	98.2	29.1
PGD	0.1	79.2	9.9	144.6
PGD, Poison	0.1	76.3	97.6	146.4
FastAdv	0.2	79.6	9.9	44.1
FastAdv, Poison	0.2	75.8	97.6	51.4

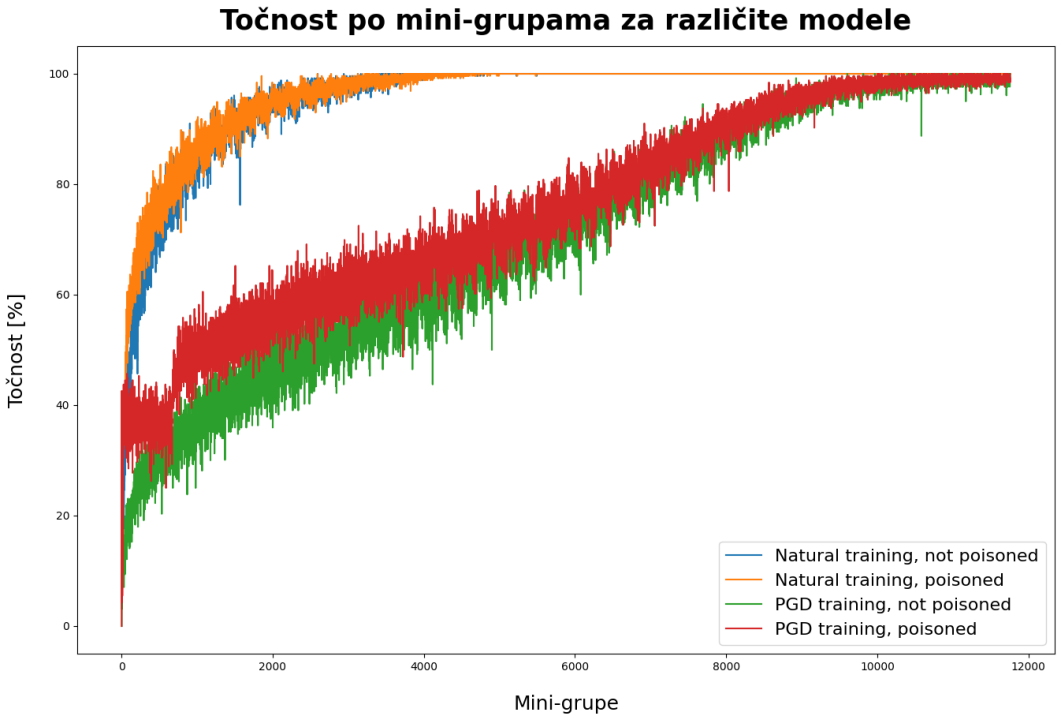
interpretiramo tako da modeli učeni na prirodnom skupu ispravno klasificiraju većinu zatrovanih slika koje uistinu i potječu iz ciljnog razreda, pritom se ne obazirući na okidač. Modeli učeni na zatrovanom skupu podataka imaju malo nižu točnost na prirodnom skupu, ali i veoma visoku točnost na zatrovanim podatcima. Visoka točnost na zatrovanim podatcima govori nam da su svi modeli učeni na zatrovanom skupu naučili okidač. Drugim riječima, čak ni robusni modeli poput modela učenih PGD-om nisu otporni na zatrovane podatke.



Slika 5.7: Usporedba točnosti zatrovanog modela učenog PGD-om po epohama.

Slika 5.7 prikazuje usporedbu točnosti na zatrovanom skupu za učenje, točnosti na skupu za testiranje i točnosti na zatrovanom skupu za testiranje za model učen PGD-om na zatrovanom skupu podataka. Već u prvih nekoliko epoha, točnost na zatrovanom skupu za testiranje stabilizira se na iznosu od skoro 100% te do kraja učenja neznatno pada. Točnost na skupu za testiranje već oko 20. epohe dostigne svoj vrhunac, dok točnost na zatrovanom skupu za učenje kontinuirano raste. Da je ovaj model u mogućnosti detektirati zatrovane podatke, očekivali bismo značajno nižu točnost na zatrovanom skupu za testiranje, kao i višu točnost na prirodnom skupu za testiranje. Ipak, postoji mogućnost da promatranje točnosti na razini epoha nije dovoljno precizno. Zbog ovoga ćemo dodatno pogledati točnost na zatrovanom skupu za učenje po mini-grupama.

Slika 5.8 prikazuje usporedbu točnosti na skupu za učenje po mini-grupama za 4 različita modela. Modeli u čijem nazivu se nalazi riječ *poisoned* učeni su na zatrovanom skupu podataka pa je stoga na slici za iste prikazana točnost na zatrovanom skupu za učenje. Za preostale modele, na slici je prikazana točnost na prirodnom skupu za učenje. Prirodni model učen na zatrovanim podatcima po točnosti je gotovo nerazlučiv od prirodnog modela učenog na prirodnim podatcima. Pošto je za modele učene na zatrovanim podatcima prikazana točnost na upravo tim podatcima, iz grafa saznajemo da prirodni model gotovo odmah nauči prepoznavati okidač.

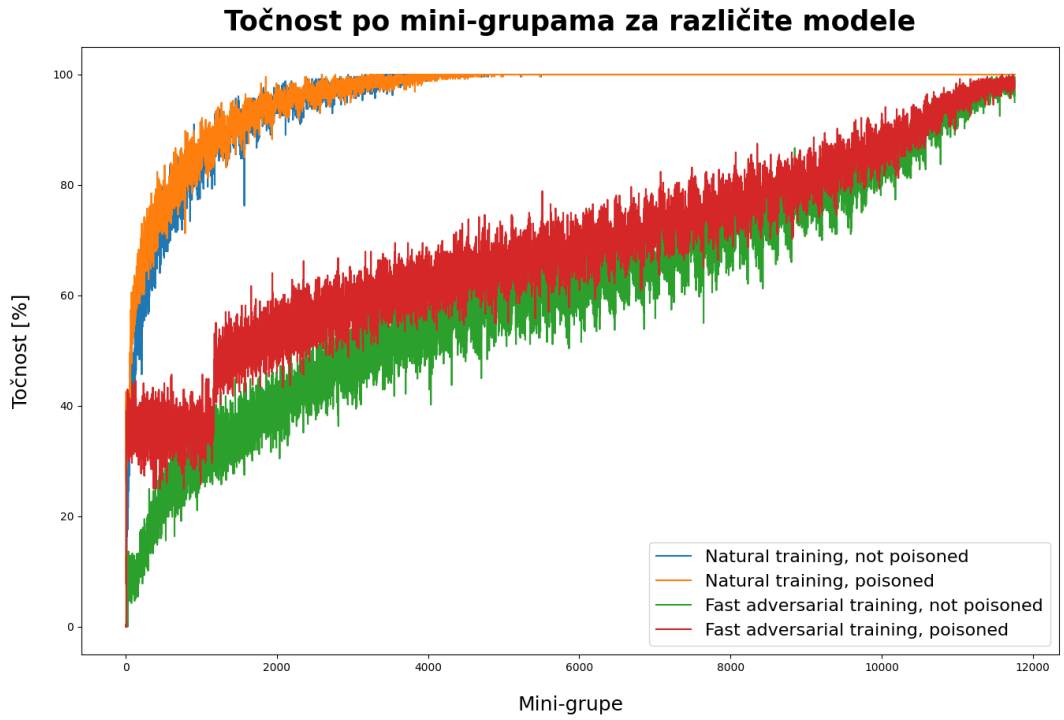


Slika 5.8: Usporedba točnosti po mini-grupama za prirodne modele i modele učene PGD-om.

S druge strane, točnosti za modele učene PGD-om na početku se značajno razlikuju. Kada je u pitanju model učen PGD-om na zatrovanim podatcima, oko 1000. mini-grupe dolazi do brzog skoka u točnosti. Ovu pojavu interpretiramo kao trenutak kada model učen PGD-om nauči prepoznavati okidač - nakon tog trenutka, njegova točnost na zatrovanim podatcima krene brzo rasti, a pritom se rast u točnosti podudara s rastom točnosti modela učenog PGD-om na prirodnom skupu podataka. Iako naučeni model nema moć detektiranja zatrovanih podataka, mogli bismo reći da se na početku učenja model "opire" učenju okidača. U tom periodu, modelova točnost na zatrovanim podatcima stagnira, no nakon otprilike 1000. mini-grupe, modelova točnost na zatrovanim podatcima započinje svoj brzi rast. Kako bismo vidjeli događa li se ovo samo kada je u pitanju učenje modela PGD-om, promotrit ćemo i usporedbu točnosti po mini-grupama prirodnih modela i modela učenih metodom *FastAdv*.

Kao i na prethodnoj slici, i na slici 5.9 moguće je vidjeti usporedbu točnosti na skupu za učenje po mini-grupama. Pritom su modeli učeni PGD-om zamijenjeni modelima učenim metodom *FastAdv* kako bi proučili događa li se i kod ovog načina učenja brzi rast točnosti na zatrovanom skupu za učenje. Možemo uočiti da se i kod modela učenih metodom *FastAdv* događa ista pojava - model učen na zatrovanim podatcima na početku se "opire" učenju okidača. Kao i kod modela učenih PGD-om, u tom periodu točnost modela učenih metodom *FastAdv* na zatrovanim podatcima stag-

nira, no nakon nekog vremena, njen iznos počinje brzo rasti.



Slika 5.9: Usporedba točnosti po mini-grupama za prirodne modele i modele učene metodom *FastAdv*.

Prema dobivenim rezultatima, ni modeli učeni PGD-om, a ni modeli učeni metodom *FastAdv* na zatrovanim podatcima nemaju mogućnost detekcije zatrovanih podataka. Ipak, pojava vrlo brzog rasta točnosti na zatrovanom skupu za učenje kod robusnih modela veoma je zanimljiva. U slučaju da otprilike znamo kako bi se trebala povećavati točnost na skupu za učenje za model koji učimo, praćenjem točnosti po mini-grupama potencijalno bismo mogli detektirati trenutak kada model nauči prepoznavati okidač. Ako uspijemo detektirati takav trenutak, možemo reći da su u skup za učenje vjerojatno ubačeni zatrovani podatci.

6. Zaključak

Kako bi se korištenje dubokih modela moglo integrirati u važne aspekte svakodnevnog života, važno je razmišljati o sigurnosti samih modela. Neke od glavnih prijetnji sigurnosti dubokih modela su neprijateljski primjeri i zatrovani podatci. U ovome radu, napravili smo kratak pregled osnovnih načina za postizanje modela otpornih na neprijateljske primjere. Dva takva načina su učenje FGSM-om ili PGD-om. Istražili smo probleme vezane uz brzinu učenja robusnih modela otpornih na neprijateljske primjere konstruirane iterativnim napadima. Implementirali smo i evaluirali točnost, kao i brzinu učenja predloženih nadogradnji na klasične načine učenja s neprijateljskim primjerima: metoda *FreeAdv*, *FastAdv*, *FastAdv+* i *FastAdvW*. Kako bismo ispravno implementirali navedene načine brzog učenja s neprijateljskim primjerima, istražili smo korištenje računanja u mješovitoj preciznosti s ciljem ubrzanja učenja modela. Korištenjem načina učenja *FastAdvW*, postigli smo modele robusnosti otprilike jednake robusnosti modela učenih PGD-om, ali s potrebnih skoro dvostruko manje vremena za učenje.

Nakon implementacije različitih načina brzog učenja s neprijateljskim primjerima, istražili smo imaju li robusni modeli naučeni na zatrovanom skupu podataka moć detekcije zatrovanih podataka. Nažalost, došli smo do zaključka da klasične implementacije robusnih modela poput modela učenih PGD-om ili metodom *FastAdv* nemaju moć detekcije zatrovanih podataka jer, kao i modeli učeni prirodnim učenjem, tijekom učenja nauče prepoznavati okidač i na temelju toga klasificirati ulaz. Ipak, uočili smo da ako za robusne modele tijekom učenja pratimo točnost na skupu za učenje po minigrupama, potencijalno možemo izdvojiti modele učene na skupu podataka s ubačenim zatrovanim podatcima. Prema dobivenim rezultatima, robusni modeli učeni na zatrovanim podatcima u jednom trenutku će imati nagli rast u točnosti na zatrovanom skupu za učenje. Ovo svojstvo potencijalno bismo mogli koristiti kao detektor postojanja zatrovanih podataka.

U budućim istraživanjima bilo bi zanimljivo primijeniti metode brzog učenja s neprijateljskim primjerima na kompleksnije arhitekture mreža i veće skupove podataka jer su u okviru ovog rada istraživanja zbog brzine učenja bila izvođena isključivo na ResNet-18 arhitekturi i CIFAR-10 skupu podataka. Zanimljivo bi bilo istražiti performanse modela učenih različitim metodama sa zadanim vremenskim budžetom za sve metode. Uz ovo, vrijedilo bi istražiti mogućnost kombiniranja "besplatnog" i brzog učenja kako bi se dobio model visokih performansi, kao i vrlo kratkog vremena učenja. Po pitanju detekcije zatrovanih podataka, istraživanje bi trebalo proširiti na dodatne načine učenja s neprijateljskim primjerima poput metoda *FastAdv+* i *FastAdvW*. Za sve načine učenja trebalo bi provesti praćenje točnosti na zatrovanom skupu za učenje po mini-grupama kako bi se provjerila hipoteza da do naglog rasta u točnosti dolazi kod svih robusnih modela. U slučaju da se pokaže da ovo svojstvo imaju svi robusni modeli učeni na zatrovanom skupu podataka, trebalo bi dodatno istražiti mogućnost primjene ovog svojstva kao detektora postojanja zatrovanih podataka.

LITERATURA

[1] Bojana Dalbelo Bašić, Marko Čupić, i Jan Šnajder. Umjetne neuronske mreže, prezentacija, 2020.

[2] Hossein Gholamalinezhad i Hossein Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020.

[3] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] Ian J Goodfellow, Jonathon Shlens, i Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning for image recognition. U *Proceedings of the IEEE conference on computer vision and pattern recognition*, stranice 770–778, 2016.

[6] Diederik P Kingma i Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[8] Bai Li, Shiqi Wang, Suman Jana, i Lawrence Carin. Towards understanding fast adversarial training. *arXiv preprint arXiv:2006.03089*, 2020.

[9] Yiming Li, Mengxi Ya, Yang Bai, Yong Jiang, i Shu-Tao Xia. Backdoorbox: A python toolbox for backdoor learning. *arXiv preprint arXiv:2302.01762*, 2023.

[10] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, i Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[11] Arjun Menon. Data poisoning and its impact on the ai ecosystem, 2023.

[12] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[13] Keiron O'Shea i Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[14] Farheen Ramzan, Muhammad Usman Khan, Asim Rehmat, Sajid Iqbal, Tanzila Saba, Amjad Rehman, i Zahid Mehmood. A deep learning approach for automated diagnosis and multi-class classification of alzheimer's disease stages using resting-state fmri and residual neural networks. *Journal of Medical Systems*, 44, 12 2019. doi: 10.1007/s10916-019-1475-2.

[15] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, i Tom Goldstein. Adversarial training for free! *Advances in Neural Information Processing Systems*, 32, 2019.

[16] Leslie N Smith. Cyclical learning rates for training neural networks. U *2017 IEEE winter conference on applications of computer vision (WACV)*, stranice 464–472. IEEE, 2017.

[17] Eric Wong, Leslie Rice, i J Zico Kolter. Fast is better than free: Revisiting adversarial training. *arXiv preprint arXiv:2001.03994*, 2020.

Algoritmi za brzo učenje na neprijateljskim primjerima

Sažetak

Učenje s neprijateljskim primjerima važno je za postizanje sigurnih modela. Proučavamo osnovne načine učenja s neprijateljskim primjerima (FGSM, PGD). Uočavamo i istražujemo probleme prisutne pri korištenju načina učenja PGD na kompleksnijim modelima. Razmatramo predložene nadogradnje na osnovne načine učenja s neprijateljskim primjerima (*FreeAdv*, *FastAdv*, *FastAdv+*, *FastAdvW*) i evaluiramo performanse naučenih modela. Evaluacija pokazuje da najbolje performanse postižu modeli učeni načinima učenja *FastAdv+* i *FastAdvW*. Razmatramo mogućnost korištenja robusnih modela za detekciju zatrovanih podataka.

Ključne riječi: robusni modeli, učenje s neprijateljskim primjerima, FGSM, PGD, "besplatno" učenje s neprijateljskim primjerima, brzo učenje s neprijateljskim primjerima, zatrovani podatci

Algorithms for fast adversarial learning

Abstract

Adversarial learning is important for achieving secure models. We study the basic adversarial learning methods (FGSM, PGD). We observe and investigate the problems present when using the PGD learning method on more complex models. We consider proposed upgrades to the basic adversarial learning method (*FreeAdv*, *FastAdv*, *FastAdv+*, *FastAdvW*) and evaluate the performance of the trained models. The evaluation shows that the best performance is achieved by models trained using the *FastAdv+* and *FastAdvW* learning methods. We consider the possibility of using robust models for the detection of poisoned data.

Keywords: robust models, adversarial learning, FGSM, PGD, free adversarial learning, fast adversarial learning, poisoned data