SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6345

NEPRIJATELJSKI NAPADI NA MODELE ZA KLASIFIKACIJU SLIKE

Iva Božić

Zagreb, lipanj 2019.

Zahvaljujem mentoru prof. dr. sc. Siniši Šegviću na ukazanom povjerenju i pruženim savjetima i pomoći pri izradi ovog rada. Hvala Ivanu na bezuvjetnoj ljubavi i potpori.

Sadržaj

1 Uvod

Uspjeh dubokog učenja u zadnjih nekoliko godina se pokazao primjenjiv u raznim sferama života poput prepoznavanja lica [1], raspoznavanja govora [2] i autonomnih vozila [3, 4, 5]. No unatoč tom uspjehu pokazalo se kako su duboki modeli izuzetno osjetljivi na neprijateljske perturbacije podataka [6]. Računalni vid predstavlja poseban izazov kada govorimo o neprijateljskim primjerima. Dodavanjem izrazito malih, ljudskom oku neprimjetnih, perturbacija možemo dobiti slike koje modeli krivo klasificiraju. Ovo, naravno, podiže pitanje o sigurnosti dubokih modela te predstavlja vrlo bitan problem. Zato je istraživanje napada i obrana dubokih modela od velikog interesa u području dubokog učenja.

Predmet ovog rada jest treniranje modela za prepoznavanje rukom pisanih znamenki na skupu MNIST (*engl. Modified National Institute of Standards and Technology database*) te konstruiranje neprijateljskih primjera korištenjem dva najpoznatija algoritma u ovom području: brza predznačna metoda temeljena na gradijentu (*engl. Fast Gradient Sign Method, FGSM*) i projicirani gradijentni spust (*engl. Projected Gradient Descent, PGD*). Također, razmotrit će se obrane od navedenih napada te testirati postignuta robusnost.

U 2. poglavlju dan je kratak uvod na novi pogled na neuronske modele, ukratko su opisane osnove i podjela strojnog učenja, te modeli koji su razmatrani u ovom radu. U 3. poglavlju dan je formalan opis neprijateljskih napada uz podjelu na dvije osnove skupine te su opisani FGSM i PGD. Postizanje robusnosti učenjem s neprijateljskim primjerima je opisano u 4. poglavlju. Programski okvir TensorFlow opisan je u 5. poglavlju. 6. poglavlje opisuje podatkovni skup MNIST. U poglavlju 7 opisana je programska izvedba, a u poglavlju 8 prikazani su eksperimentalni rezultati. U poglavljima 9 i 10 dani su zaključak i popis literature. Na kraju je dan kratak sažetak na hrvatskom i engleskom jeziku.

2 Diferencijabilno programiranje

U mnogim područjima danas, kao što su prepoznavanje na slikama [7], strojno prevođenje [8, 9], pretraživanje, preporuke, autonomna vozila, baze podataka [10], sinteza i prepoznavanje govora, diferencijabilno programiranje počinje preuzimati posao koji je do sad bio izvođen klasičnom programskom potporom (*engl. Software*). Diferencijabilno programiranje odnosi se na pisanje koda korištenjem parametriziranih diferencijabilnih operatora (*engl. differentiable operators*) kao što su modeli umjetnih neurona. Neuronski modeli su diferencijabilne funkcije sastavljene od jednostavnih diferencijabilnih građevnih elemenata. Konstruiranje programa pomoću takvih elemenata i njihovo treniranje korištenjem gradijenata dobivenih automatskim širenjem pogreške unatrag (*engl. backpropagation*) naziva se diferencijabilno programiranje.

Drugim riječima, cilj je oblikovati programe koji se mogu učiti na podatcima. Prilikom izrade klasične programske potpore (tradicionalno ručno programiranje algoritama) problem se prvo formalno specificira, zatim se pomno dizajniraju algoritmi i kompleksni problemi se dijele u manje potprobleme. Rezultat ovog procesa sastoji se od eksplicitnih instrukcija koje oblikuju program. S druge strane, diferencijabilno programiranje je slično klasičnom programiranju s tom razlikom što su koraci obrade diferencijabilnog programa parametrizirani i diferencijabilni. Prvo se prikupljaju podatci za treniranje koji se zatim daju kao ulaz u algoritam strojnog učenja. Na temelju tih podataka uče se parametri koji aproksimiraju željenu funkcionalnost programa. Zadavanjem ciljnog ponašanja programa (npr. pobijediti u partiji goa), i pisanjem kostura koda (kao što je jedan od neuronskih modela) pronalazi se podskup programskog prostora za koji program dobro radi. Konkretno, kada govorimo o neuronskim modelima pronalaženje efikasnog programa se obavlja metodom propagacije pogreške unatrag i gradijentnim spustom (*engl. gradient descent*).

Diferencijabilno programiranje naravno ima i svoja ograničenja. Jedna od najvećih prepreka je neinterpretabilnost istreniranih modela. Također, još uvijek se otkrivaju svojstva diferencijabilnog programiranja, kao što su neprijateljski primjeri i napadi, koji

naglašavaju nedovoljnu istraženost ovog pristupa. Ovo će ujedno biti u fokusu ovoga rada.

Diferencijabilno programiranje predstavlja novi pogled na neuronske mreže na način da ih više ne percipiramo kao biološki inspiriran alat koji uči kao ljudski mozak, već kao koncept koji proizlazi iz samog programiranja. Prema [11], pojava umjetnih neuronskih mreža predstavlja početak zaokreta u tome kako se piše programska potpora.

2.1 Osnove strojnog učenja

Strojno učenje se bavi proučavanjem kako izraditi komponente programa koji su u stanju elemente svoje funkcionalnosti naučiti na podatcima [12].

Pretpostavimo da na ulaz potprograma dobivamo D-dimenzionalni podatak $\mathbf{x}$. Također pretpostavimo da je ciljno ponašanje potprograma rasporediti ulazni podatak u jednu od C klasa. Drugim riječima rečeno, željeno ponašanje potprograma je dati predikciju y. Predikcija y odgovara indeksu klase ulaznog podatka. Ako uzmemo za primjer da je opisani potprogram dio sustava za raspoznavanje brojeva, klase bi u tom slučaju bile znamenke od 0 do 9 [12].

Računalni program koji omogućuje ovako opisano učenje na primjerima označenih podataka gradi se uz pomoć algoritma strojnog učenja [12].

Osnovni elementi koji definiraju algoritam strojnog učenja su model, gubitak i optimizacijski postupak [12].

Model predstavlja dio programa koji treba učiti u obliku matematičke funkcije sa slobodnim parametrima. Diferencijabilnost tog modela nam omogućuje učenje gradijentnim spustom [12].

Drugi element koji je bitan kod algoritma strojnog učenja je gubitak. Gubitak je funkcija kojom se opisuje koliko je model dobro naučio na temelju podataka, tj. u kojoj se mjeri predikcije modela poklapaju s podatcima za učenje [12].

Optimizacijski postupak podešava parametre na način da se postigne što manji gubitak na skupu za učenje [12].

2.2 Nadzirano i nenadzirano učenje

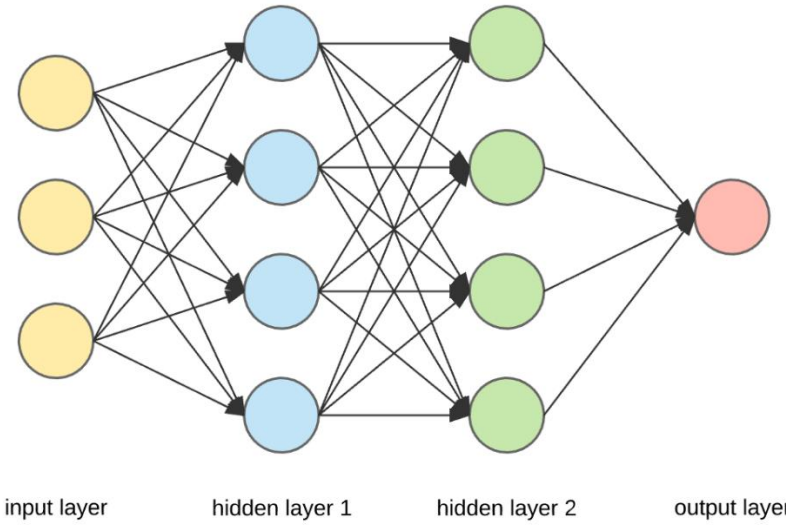
Učenje s obzirom na podatke možemo podijeliti na nadzirano i nenadzirano učenje.

Kod nadziranog učenja (*engl. supervised learning*) ulazni podatci dolaze u takvom obliku da za svaki ulaz postoji očekivani izlaz. Nenadzirano učenje (*engl. unsupervised learning*) dobiva podatke bez informacije o očekivanom izlazu.

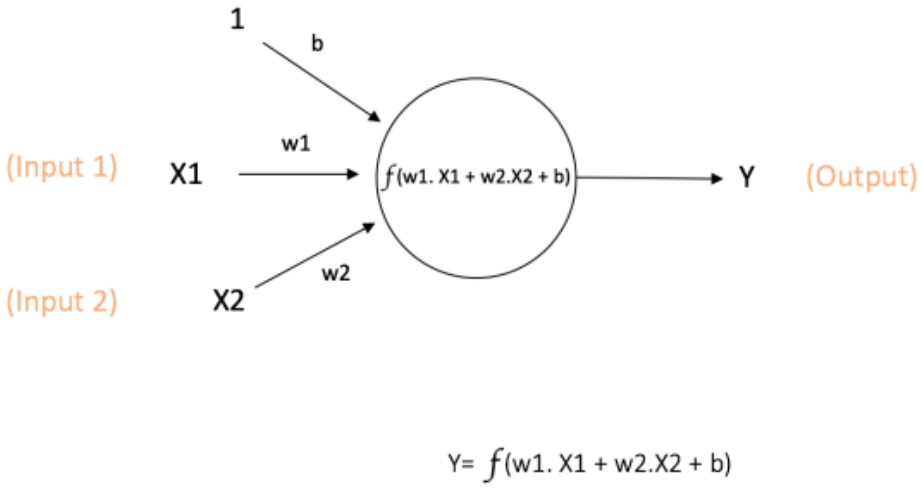
2.3 Potpuno povezani slojevi

Potpuno povezani sloj sastoji se od skalarnih čvorova koji su povezani sa svim čvorovima iz prethodnog sloja. Potpuno povezani sloj modelira afinu linearnu transformaciju prethodnog sloja pri čemu svaki njegov čvor odgovara skalarnom produktu gustog vektora težina i aktivacija prethodnog sloja uvećanom za skalarni pomak.

Kod osnovnih neuronskih modela razlikujemo 3 različita sloja: ulazni sloj (*engl. input layer*), skriveni sloj(evi) (*engl. hidden layer*) i izlazni sloj (*engl. output layer*). Primjer takvog modela prikazan je na slici 1. Ulazni sloj sadrži zadane podatke koje šalje kao ulaze skrivenom sloju. Izlazni sloj sadrži rezultat klasifikacije. Skriveni sloj(evi) se nalaze između ulaznog i izlaznog sloja.



Slika 1. Potpuno povezana mreža sastavljena od ulaznog sloja, dva skrivena sloja i izlaznog sloja [13].



Slika 2. Skalarni čvor koji na temelju danih ulaza računa izlaz pri čemu f predstavlja nelinearnu aktivacijsku funkciju [14].

Ulaz $x_1, x_2, \ldots, x_n$ u skalarni čvor može biti ili zadani ulazni podatak ili izlaz iz prethodnog skalarnog čvora. Svaki ulaz ima odgovarajuću težinu w_i (*engl. weight*) koja pokazuje koliko *i*-ta vrijednost ulaza utječe na skalarni čvor u odnosu na ostale ulaze. Težine su vrijednosti koje modeli uče prilagođavanjem viđenim podatcima. Na temelju viđenih podataka i učenju na njima generaliziraju određeno znanje [15]. Dodatan ulaz uz težine

je prag b (engl. bias). Izlaz iz skalarnog čvora je rezultat funkcije f koja predstavlja nelinearnu aktivacijsku funkciju (*engl non-linear activation function*). Najčešće aktivacijske funkcije su sigmoidalna funkcija (*engl. sigmoid*), tangens hiperbolni (*engl. tanh*) i zglobnica (*engl. Rectified Linear Unit, ReLU*)

Sigmoidalna funkcija preslikava realne vrijednosti u raspon $[0,1]$. Definirana je u izrazu (1) te grafički prikazana na slici 3.

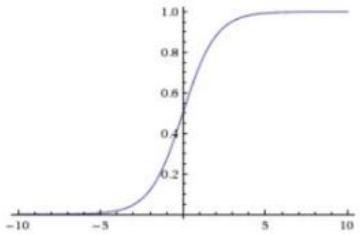
$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Tangens hiperbolni preslikava realne vrijednosti u raspon $[-1,1]$. Ova funkcija je definirana u izrazu (2) i grafički prikazana na slici 4.

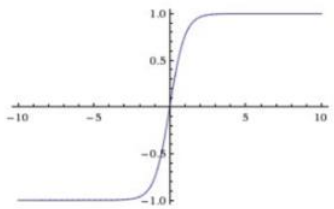
$$\tanh(x) = 2\sigma(2x) - 1 \tag{2}$$

Funkcija ReLU sve negativne realne vrijednosti zamjenjuje s nulom. ReLU je definirana u izrazu (3) i grafički prikazana na slici 5.

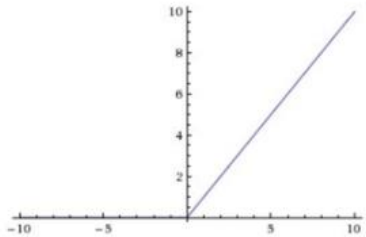
$$f(x) = \max(0, x) \tag{3}$$



Slika 3. Sigmoidalna funkcija prikazana na intervalu [-10,10] [14].



Slika 4. Funkcija tanh prikazana na intervalu [-10,10] [14].



Slika 5. ReLU funkcija prikazana na intervalu [-10,10] [14].

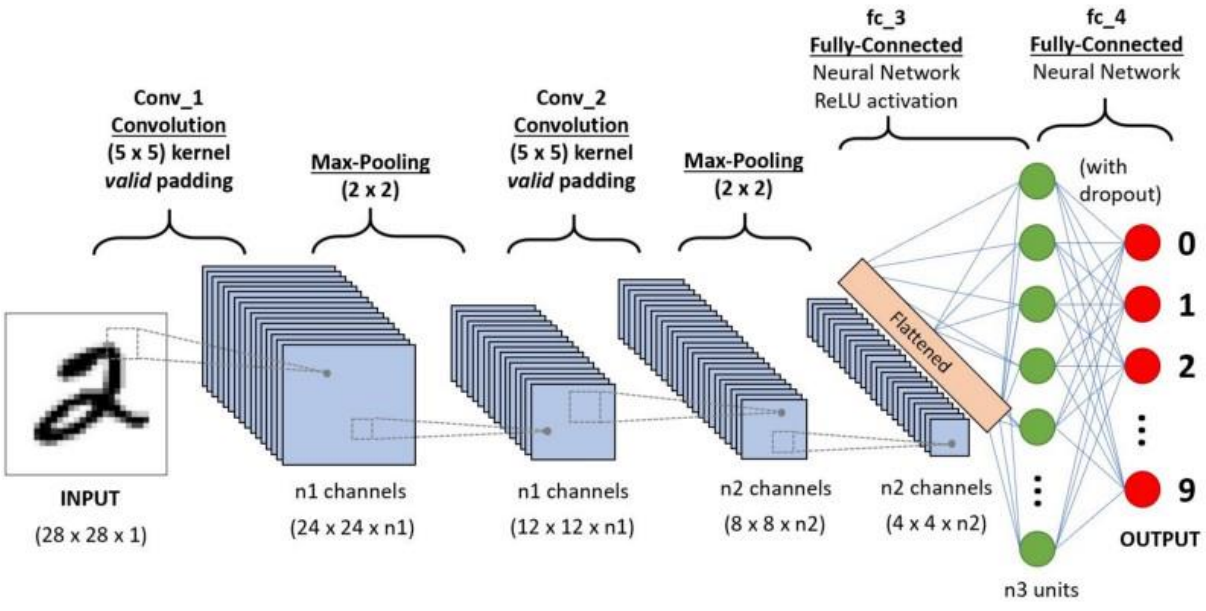
2.4 Konvolucijski modeli

Konvolucijski modeli su modeli koji uče na podatcima koji su u obliku višedimenzionalnih polja [15]. Konvolucijski modeli se sastoje od konvolucijskog sloja (*engl. convolutional layer*), sloja sažimanja (*engl. pooling layer*) i potpuno povezanog sloja (*engl. fully connected layer*).

Parametri konvolucijskog sloja su filtri koje predstavljamo tenzorima težina, uobičajeno puno manjih dimenzija od ulaza. Filter se ponaša kao pomično okno koje se pomiče od prvog do zadnjeg stupca nakon čega prelazi u novi red. Taj postupak se ponavlja dok se ne obiđu sva okna ulaznog tenzora. Za svaki pomak sve se težine okna množe s odgovarajućim poljima koja prekrivaju. Ono što se postiže ovim postupkom je prepoznavanje prostornih značajki [15].

Potpuno povezani slojevi su kao oni opisani u poglavlju 2.3.

Sloj sažimanja smanjuje dimenzionalnost ulaza. Dimenzionalnost se smanjuje mapiranjem prostorno bliskih značajki u jednu (kao npr. operacijom uzimanja najveće vrijednosti (*engl. max pooling*)) [15].



Slika 6. Arhitektura konvolucijskog modela [16].

3 Neprijateljski napadi

Tipična konstrukcija neprijateljskih napada je dodavanje perturbacija na sliku tako da model krivo klasificira tako zadani ulaz iako je perturbacija nevidljiva ljudskom oku. Proces kreiranja slika koje su u stanju prevariti model slijedi u nastavku.

Definirajmo model kao preslikavanje iz prostora ulaznih podataka u prostor izlaznih podataka kako je prikazano u (4).

$$h_\theta : X \to \mathbb{R}^k \tag{4}$$

Prostor izlaznih podataka je $\boldsymbol{k}$-dimenzionalni vektor gdje $\boldsymbol{k}$ označava broj mogućih klasa kojima ulazni podatci mogu pripadati. Vektor $\boldsymbol{\theta}$ označava parametre modela.

Treniranje modela postižemo određivanjem njegovih parametara $\boldsymbol{\theta}$ koji minimiziraju funkciju gubitka za zadani set ulaznih podataka $\boldsymbol{x_1, \ldots, x_n}$.

Funkcija gubitka definira se kao preslikavanje prikazano u izrazu (5).

$$\ell : \mathbb{R}^k \times \mathbb{Z}_+ \to \mathbb{R}_+ \tag{5}$$

$\mathbb{R}^k$ predstavlja izlaz iz modela, a $\mathbb{Z}_+$ oznaku prave klase podataka. Sukladno tome iz (5), za $y \in \mathbb{Z}_+$, slijedi izraz (6).

$$\ell(h_\theta(x), y) \tag{6}$$

Najčešća funkcija gubitka u dubokom učenje je specijalni slučaj unakrsne entropije (*engl. cross entropy loss*) u obliku negativne log-izglednosti, definirana u izrazu (7), gdje $h_\theta(x)_j$ označavaju j-tu komponentu vektora $\boldsymbol{h_\theta(x)}$.

$$\ell(h_\theta(x), y) = \log\left(\sum_{j=1}^{k} exp\left(h_\theta(x)_j\right)\right) - h_\theta(x)_y \tag{7}$$

Izraz (7) dolazi od funkcije softmax koja je definirana kao

$$\sigma(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \tag{8}$$

Pomoću softmax funkcije računamo aposteriorne vjerojatnosti razreda [15]. Klasifikacijske mjere (*engl. logits*) koje dobijemo kao izlaz iz (4) su ulaz u funkciju (8) koja kao rezultat daje aposteriorne vjerojatnosti razreda.

Prilikom treniranja modela želimo maksimizirati vjerojatnost pripadnosti podataka pojedinom razredu. Kako se vjerojatnosti kreću unutar intervala $[0, 1]$ i poprimaju vrlo male vrijednosti, umjesto maksimizacije vjerojatnosti pripadnosti maksimiziramo logaritamsku vjerojatnost razreda koja je dana u izrazu (9). Također, za veliki broj podataka puno je pogodnije zbrajati logaritamsku vjerojatnost nego množiti vjerojatnosti.

$$\log \sigma(h_\theta(x))_y = \log\left(\frac{exp\left(h_\theta(x)_y\right)}{\sum_{j=1}^{k} exp\left(h_\theta(x)_j\right)}\right) = h_\theta(x)_y - \log\left(\sum_{j=1}^{k} \exp\left(h_\theta(x)_j\right)\right) \tag{9}$$

Uobičajeno je da se umjesto maksimizacije vjerojatnosti minimizira gubitak, te se stoga uzima negativna logaritamska izglednost funkcije (9) kao funkcija gubitka iz čega dobivamo izraz (7). Minimizaciju gubitka definiramo kao

$$\theta^* = \arg\min_{\theta} \sum_{x} \ell(x, \theta) \tag{10}$$

Ako želimo da model $\boldsymbol{\theta}$ pogrešno klasificira ulazne podatke $\boldsymbol{x}$**,** određujemo ograničenu perturbaciju $\boldsymbol{\delta}$ te maksimiziramo gubitak za $\boldsymbol{x} + \boldsymbol{\delta}$**,** kao što je definirano u izrazu (11).

$$\delta^* = \arg\max_{\delta} \ell(x + \delta, \theta) \tag{11}$$

Ovaj rad se fokusira na dvije metode kojima možemo konstruirati neprijateljske napade, a to su FGSM i PGD. Njihov opis slijedi u poglavljima 3.2 i 3.3.

3.1 Vrste neprijateljskih napada

Neprijateljske napade možemo podijeliti u dva osnovna tipa: neusmjereni (*eng. non-targeted*) i usmjereni (*engl. targeted*) napadi. Neusmjereni napadi su najopćenitiji tip napada jer je jedini cilj ove vrste napada modificirati izvorne podatke (ulazni podatci su slike) na način da ih model pogrešno klasificira pri čemu nam nije važno kojoj će klasi model pridružiti zadani ulaz dok god ta klasa nije klasa kojoj podatak zaista pripada. Usmjereni napadi su tip napada kojim modifikacijom ulazne slike želimo postići da model predvidi točno određeni rezultat (klasu) na izlazu za dani ulaz.

3.1.1 Neusmjereni napadi

Treniranjem modela želimo optimizirati parametre $\boldsymbol{\theta}$. To radimo na način da minimiziramo prosječni gubitak nad nekim setom za treniranje $\{\boldsymbol{x_i} \in \boldsymbol{X}, \boldsymbol{y_i} \in \mathbb{Z}\}, \boldsymbol{i} = \boldsymbol{1}, \ldots, \boldsymbol{m}$. Navedeno opisujemo kao optimizacijski problem izrazom (12).

$$\underset{\theta}{\text{minimize}} \frac{1}{m} \sum_{i=1}^{m} \ell(h_\theta(x_i), y_i) \tag{12}$$

Uobičajeni način rješavanja optimizacije (12) je korištenje stohastičkog gradijentnog spusta. Mini-grupa $\boldsymbol{B}$ (*engl. minibatch*) je određena grupa uzoraka tako da je $\boldsymbol{B} \subseteq \{\boldsymbol{1}, \ldots, \boldsymbol{m}\}$ te se gradijent gubitka računa po parametrima $\boldsymbol{\theta}$ [17]. Parametre $\boldsymbol{\theta}$ se ažurira kako je napisano u izrazu (13)

$$\theta := \theta - \frac{\alpha}{|B|}\sum_{i \in B} \nabla_\theta \ell(h_\theta(x_i), y_i) \tag{13}$$

pri čemu je α veličina koraka, a $\boldsymbol{\nabla_\theta \ell(h_\theta(x_i), y_i)}$ gradijent. Postupak u izrazu (13) se ponavlja za različite mini-grupe do trenutka kada parametri počinju konvergirati. Kod dubokih modela gradijent se računa vrlo efikasno korištenjem propagacije pogreške unatrag (*engl. backpropagation*) koji se temelji na pravilu ulančavanja. Automatska diferencijacija koja se temelji na unazadnom prolazu omogućuje jednostavan izračun gradijenta gubitka ne samo po parametru $\boldsymbol{\theta}$, nego i po samom ulazu $\boldsymbol{x_i}$. Računanje gradijenta po parametru $\boldsymbol{\theta}$ pokazuje kako male promjene tog parametra utječu na funkciju gubitka dok gradijent po ulazu pokazuje kako male promjene u samom ulazu utječu na funkciju gubitka [17].

Kako bi odredili neprijateljski primjer, računat ćemo upravo gradijent po ulazu. Ulaz se prilagođava na način da maksimizira gubitak (za razliku od optimizacije parametara modela kada želimo minimizirati gubitak).

Prilikom konstrukcije neprijateljskih primjera osiguravamo da je neprijateljski primjer sličan originalnom ulazu $\boldsymbol{x}$. Ovo se postiže pomoću optimizacije nad ograničenom perturbacijom originalnog ulaza, pri čemu perturbaciju označavamo s δ. Optimizacijski problem koji želimo riješiti napisan je u izrazu (14).

$$\underset{\delta \in \Delta}{\text{maximize}}\, \ell\,(h_\theta(x\, +\, \delta), y) \tag{14}$$

U izrazu (14) Δ predstavlja dopušteni set perturbacija [17]. Taj set nije moguće matematički točno definirati. Razlog tome je što dopušteni set perturbacija može uključivati bilo što za što bi čovjek vizualno rekao da je isto ulaznom podatku (slici). To uključuje ne samo dodavanje male količine šuma, nego i skaliranje, traslaciju, rotaciju i slično.

Jedan od uobičajenih setova perturbacija je kugla norme ℓ_∞ (*eng.* ℓ_∞ *ball*) koja je definirana kao

$$\Delta = \{\delta : ||\delta||_\infty \leq \epsilon\} \tag{15}$$

pri čemu je ℓ_∞ norma vektor $\boldsymbol{z}$ koji je definiran kao

$$||z||_\infty = \max_i |z_i| \tag{16}$$

Iz izraza (15) i (16) vidimo da je dopuštena veličina perturbacije svakog piksela između $[-\epsilon, \epsilon]$. Prednost ovakvog seta perturbacija je što se dodavanjem toga šuma na originalnu sliku svaki piksel mijenja toliko malo da je promjena vizualno nevidljiva u odnosu na originalnu sliku. Duboki modeli su vrlo osjetljivi upravo na ovakve vrste napada.

3.1.2 Usmjereni napadi

Usmjereni napadi imaju za cilj sliku klasificirati kao određenu željenu klasu.

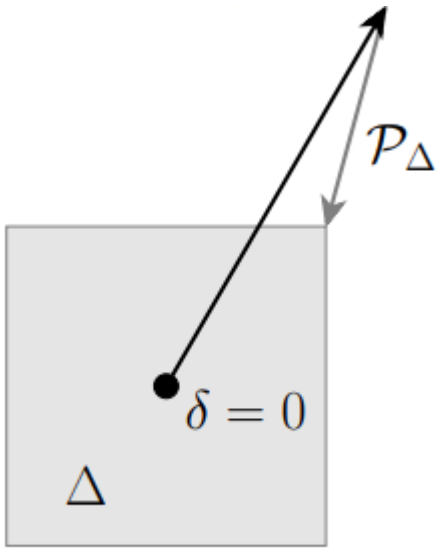
Kod usmjerenih napada jedina razlika u odnosu na postupak opisan u 3.1.1 je što želimo ne samo maksimizirati gubitak točne klase podatka nego želimo i minimizirati gubitak targetirane klase. Optimizacijski problem u ovom slučaju izgleda kako je napisano u izrazu (17).

$$\underset{\delta \in \Delta}{\text{maximize}} (\ell(h_\theta(x+\delta), y) \; - \; \ell(h_\theta(x+\delta), y_{targeted})) \tag{17}$$

Uvrštavanjem izraza (7) u izraz (17) i nakon sređivanja dobivamo izraz (18).

$$\underset{\delta \in \Delta}{\text{maximize}} (h_\theta(x+\delta)_{y_{target}} - \; h_\theta(x+\delta)_y) \tag{18}$$

3.2 Brza predznačna metoda temeljena na gradijentu (FGSM)



Slika 7. Projekcija gradijenta prilikom korištenja FGSM-a [17].

Cilj FGSM-a je prevariti model dodavanjem vrlo male perturbacije na sliku [17]. Ta perturbacija se računa na sljedeći način:

$$\delta = \epsilon \cdot sign(\nabla_\delta J(x + \delta, y, \theta)) \tag{19}$$

U formuli (19) $J(x, y)$ označava funkciju gubitka za trenirani model, $\boldsymbol{x}$ ulazne podatke, $\boldsymbol{y}$ izlazne podatke, a $\boldsymbol{\theta}$ parametre modela.

Dodavanjem izraza (19) originalnoj slici dobivamo brzu predznačnu metodu temeljenu na gradijentu koju možemo definirati kao:

$$x^{adv} = x + \delta \tag{20}$$

U gore navedenoj formuli x označava originalnu sliku, a x^{adv} sliku nakon napada (*eng. adversial image*).

Formula (20) proizlazi iz sljedećih izraza. Za neki ulazni podatak $\boldsymbol{x}$ treba podesiti $\boldsymbol{\delta}$ u smjeru gradijenta (21) kako bi se maksimizirao gubitak. Navedeno je formalno definirano u izrazu (22).

$$g := \nabla_{\delta} \ell(h_{\theta}(x + \delta), y) \tag{21}$$

$$\delta := \delta + \alpha g \tag{22}$$

α u izrazu (22) označava veličinu koraka.

Regija perturbacije Δ definira se kao kugla norme ℓ_{∞} koja označava područje unutar ℓ_{∞} norme. U slučaju ℓ_{∞} norme, koja je definirana u izrazima (15) i (16) projiciramo δ na normu kugle (*engl. projecting onto norm ball)* što konkretno u ovom slučaju znači da se podrezuje vrijednost δ na vrijednost koja leži u intervalu $[-\epsilon, \epsilon]$.

Kod FGSM-a, za dovoljno velik pomak u smjeru gradijenta, dobivena vrijednost izlazi iz okvira vrijednosti kugle norme ℓ_{∞}. Nakon projekcije za dovoljno veliki pomak projicirana vrijednost će uvijek biti u jednom od vrhova (kao što je prikazano na slici 7).

Uzmimo zbog jednostavnosti da je početna vrijednost δ nula, u tom slučaju nova vrijednost δ nakon projekcije je

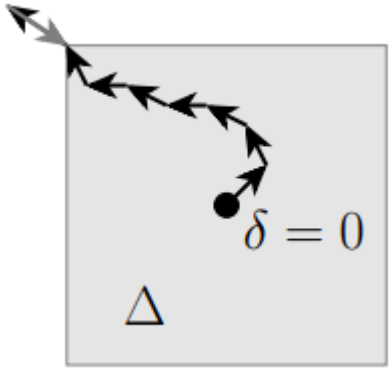
$$\delta := clip(\alpha g, [-\epsilon, \epsilon]) \tag{20}$$

Još treba riješiti veličinu koraka α. Iz logike problema da se želi povećati gubitak što je više moguće, slijedi da trebamo uzeti najveći mogući korak. Iz izraza (20) vidimo da za dovoljno velik α vrijednosti g zapravo neće imati utjecaja na δ, tj. za novu vrijednost δ se uzima $-\epsilon$ ili $+\epsilon$, ovisno o predznaku od g. Iz svega navedenog slijedi da za dovoljno velik α nova vrijednost δ postaje

$$\delta := \epsilon \cdot sign(g) \tag{21}$$

3.3 Projicirani gradijentni spust (PGD)

Prilikom konstrukcije PGD napada želimo optimizirati granicu perturbacije oko neke regije Δ (regija označena sivom bojom na slici 8). Da bi se to postiglo pomičemo se u pozitivnom smjeru gradijenta funkcije gubitka (jer želimo maksimizirati funkciju gubitka). Nakon pomaka radimo projekciju na regiju Δ, tj. pronalazimo točku koja je najbliža projiciranoj točki u Δ. Navedeni postupak se ponavlja kako bi pronašli maksimalan gubitak.



Slika 8. Optimizacija perturbacije korištenjem PGD-a [17].

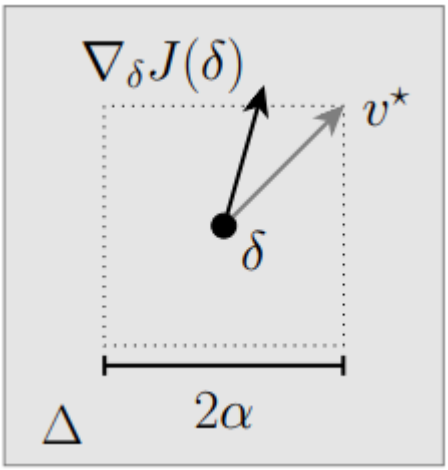
Osnovni algoritam PGD-a glasi kako slijedi:

$$Ponavljati: \qquad (22)$$
$$\delta := \mathcal{P}(\delta + \alpha \nabla_{\delta} \ell(h_{\theta}(x+\delta), y))$$

$\mathcal{P}$ u gornjoj formuli označava projekciju na kuglu (*engl. projection onto the ball*) koja u slučaju ℓ_{∞} norme označava podrezivanje (*engl. clipping*).

Kod PGD-a treba obratiti pažnju na veličinu koraka.

Pri računanju gradijenta u izrazu (21) po δ, za $\delta = 0$ primijećeno je da je gradijent u prosjeku vrlo mali, reda veličine 10e-06 [17]. Kako bi se ovo nadoknadilo, veličina koraka α mora biti vrlo velika, $\alpha = 1e4$, kako bi se pomaknuli u smjeru gradijenta [17]. Nakon uspješnog pomaka iz regije gdje je $\delta = 0$, magnituda gradijenta značajno naraste i u tom trenutku α postaje prevelik te se počinjemo prebrzo kretati prema granici što zapravo znači da se PGD ponaša isto kao FGSM. Kako bi se riješio opisani problem, optimizacijski problem će se opisati na drugačiji način metodom normaliziranog najstrmijeg spusta (*engl. normalized steepest descent method*).



Slika 9. Normalizini gradijentni spust [17].

Ako želimo minimizirati neku funkciju f po ulazu z, poznati algoritam gradijentnog spusta izgleda kao u izrazu (23).

$$z := z - \alpha \nabla_z f(z) \tag{23}$$

Kod normaliziranog najstrmijeg spusta umjesto izraza (23) koristimo izraz koji je definiran u (24).

$$z := z - \underset{||v||_\infty \leq \alpha}{\operatorname{argmax}} v^T \nabla_z f(z) \tag{24}$$

U slučaju ℓ_∞ norme argmax je definiran kao što je napisano u izrazu (25).

$$\underset{||v||_\infty \leq \alpha}{\mathrm{argmax}}\, v^T \nabla_z f(z) = \alpha \cdot sign(\nabla_z (f(z)) \tag{25}$$

Veličina koraka α i granica perturbacije ϵ su jednakog reda veličine te stoga odabiremo α takav da je jednak nekom razumno malom dijelu od ϵ [17].

Performanse PGD-a su ograničene mogućnošću pojave lokalnog optimuma unutar zadanog cilja [17]. Iako taj problem nije moguće potpuno izbjeći, može se ublažiti slučajnom inicijalizacijom oko početne točke. Ovo znači da PGD pokrećemo s različitih slučajnih lokacija unutar kugle norme ℓ_∞ koju smo odabrali. Poboljšanje koje se njegovom uporabom dobije je malo, ali nezanemarivo [17].

4 Postizanje robusnosti učenjem s neprijateljskim primjerima

Učenje s kojim postižemo robusnost možemo opisati kao optimizacijski problem prikazan izrazom (26).

$$\underset{\theta}{\text{minimize}} \frac{1}{|D_{train}|} \sum_{(x,y)\in D_{train}} \max_{\delta\in\Delta(x)} \ell(h_\theta(x+\delta), y) \tag{26}$$

Kao i kod uobičajenog treniranja modela, problem želimo riješiti stohastičkim gradijentnim spustom po parametru $\boldsymbol{\theta}$. Za svaku mini-grupu $\boldsymbol{B} \subseteq \boldsymbol{D_{train}}$ parametar $\boldsymbol{\theta}$ se ažurira na sljedeći način:

$$\theta := \theta - \frac{\alpha}{|\ B\ |} \sum_{(x,y)\in B} \nabla_\theta \max_{\delta\in\Delta(x)} \ell\,(h_\theta(x+\delta), y) \tag{27}$$

Sastavni dio izraza (27) je gradijent unutar kojeg se nalazi optimizacijski problem. Takav problem se rješava Danskinovim teoremom [18]. Više riječi o Danskinovom teoremu će biti u poglavlju 4.2. Kako se problem (26) sastoji od unutrašnje i vanjske petlje optimizacije, u nastavku je opisano rješenje problema sukladno toj podjeli.

4.1 Određivanje neprijateljskog primjera

Neprijateljski primjer se određuje rješavanjem sljedećeg optimizacijskog problema:

$$\underset{||\delta||\leq\epsilon}{\text{maximize}}\, \ell(h_\theta(x+\delta), y) \tag{28}$$

pri čemu je h_θ duboki model kao što je definirano u izrazu (4). Korištenjem propagacije

pogreške unatrag možemo izračunati gradijent funkcije gubitka po perturbaciji $\boldsymbol{\delta}$. Osim računanja gradijenta treba osigurati da $\boldsymbol{\delta}$ bude unutar granica $[-\epsilon, \epsilon]$.

Ovaj rad se fokusira na dvije metode kojima možemo odrediti neprijateljske primjere, a to su FGSM i PGD koji su opisani u poglavljima 3.2 i 3.3.

4.2 Učenje s neprijateljskim primjerima

Želimo istrenirati model koji je robustan u odnosu na neprijateljske napade, neovisno o napadu koji napadač koristi (pod uvjetom da su svi napadi unutar iste norme).

Formalno, za zadani skup S koji se sastoji od ulaznih i izlaznih parova podataka želimo riješiti optimizacijski problem napisan u izrazu (29).

$$\underset{\theta}{\text{minimize}} \frac{1}{|S|} \sum_{(x,y)\in S} \max_{||\delta||\leq\epsilon} \ell(h_\theta(x+\delta)), y) \tag{29}$$

U izrazu (29) pretpostavljamo da napadač poznaje parametre θ i sukladno tome prilagođava napade u ovisnosti o parametrima koje smo postavili. Cilj koji se želi postići je otpornost modela na napade čak i kada napadač posjeduje potpuno znanje o modelu.

Najjednostavnija i najintuitivnija metoda treniranja robusnog modela je učenje s neprijateljskim primjerima (*engl. adversial training with adversial examples*). Osnovna ideja je kreirati neprijateljske primjere te ih inkorporirati u proces treniranja.

Pitanje je nad kojim neprijateljskim primjerima provesti treniranje?

Želimo optimizirati (29) koristeći gradijentni spust. U slučaju ako želimo optimizirati θ koristeći stohastički gradijenti spust po θ za funkciju gubitka na nekoj mini-grupi, obavljamo sljedeće ažuriranje:

$$\theta \coloneqq \theta - \frac{\alpha}{|\ B\ |} \sum_{x,y \in B} \nabla_\theta \max_{||\delta|| \leq \epsilon} \ell\,(h_\theta(x+\delta), y) \tag{30}$$

Unutarnji gradijent se računa prema Danskinovom teoremu. Danskinov teorem kaže da ako želimo izračunati gradijent funkcije koja računa maksimum, moramo napraviti sljedeće: naći maksimum i zatim izračunati gradijent u toj točki. Opisano je formalno napisano u izrazu (31).

$$\nabla_\theta \max_{||\delta|| \leq \epsilon} \ell\,(h_\theta(x+\delta), y) = \nabla_\theta \ell(h_\theta(x+\delta^*(x)), y) \tag{31}$$

δ^* u izrazu (31) je definiran kao

$$\delta^*(x) = \underset{||\delta|| \leq \epsilon}{\operatorname{argmax}}\, \ell(h_\theta(x+\delta), y) \tag{32}$$

Važno je napomenuti kako Danskinov teorem tehnički vrijedi samo za slučaj gdje možemo točno izračunati maksimum. U praksi je pokazano da je kvaliteta robusnog gradijentnog spusta direktno povezana s kvalitetom izvedbe maksimizacije [4]. Drugim riječima, što je rezultat maksimizacije točniji to je aproksimacija Danskinovim teoremom točnija [17]. Zato je glavni aspekt neprijateljskog treniranja uključiti snažne napade u unutarnju petlju optimizacije. Najsnažniji napadi pronađeni u dubokom učenju su upravo varijante PGD-a [4].

5 Programski okvir TensorFlow

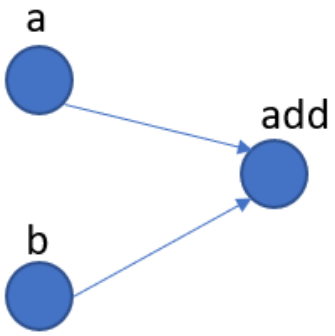
TensorFlow je besplatna biblioteka otvorenog koda (*engl. open-source software library*) za protok podataka (*engl. dataflow*) i diferencijabilno programiranje. Tensorflow je razvijen od strane Google Braina te se koristi u Googleu i mnogim drugim kompanijama za istraživanje i produkciju. Prva verzija izašla je 2015. godine. Verzija korištena u ovom radu je 1.13.1. Napisan je u programskom jeziku C++ i ubrzan je korištenjem CUDA-e (*engl. Compute Unified Device Architecture*). CUDA je paralelna računalna platforma i programski model koji omogućuje korištenje grafičkih procesorskih jedinica (GPU) za računalnu upotrebu opće namjene.

Tensorflow biblioteka dostupna je za programski jezik Python što omogućuje brz razvoj i prototipiranje rješenja baziranih na strojnom učenju.

Uporaba TensorFlowa je široko rasprostranjena u strojnom učenju, pogotovo kada je riječ o neuronskim mrežama. TensorFlow je simbolička matematička biblioteka koja omogućuje numeričke izračune stvaranjem računskog grafa toka podataka (*engl. dataflow graph*). Primjer jednostavnog grafa u TensorFlowu dan je sljedećim kodom:

```
1. a = tf.placeholder(tf.int8, name = ‘a’)
2. b = tf.placeholder(tf.int8, name = ‘b’)
3. add = tf.add(a, b, name = ‘add’)
4. sess = tf.Session()
5. sess.run(add, feed_dict={a: 2, b: 3})
```

Linije 1 i 2 stvaraju prazne čvorove sa simboličkim imenom i tipom podatka, koji će biti inicijalizirani na vrijednosti u nastavku. Linija 3 stvara čvor koji predstavlja operaciju zbrajanja. U liniji 4 stvara se Session, a u liniji 5 provodi evaluaciju TensorFlow koda. Računski graf koji nastaje ovako definiranim kodom prikazan je slikom 10.



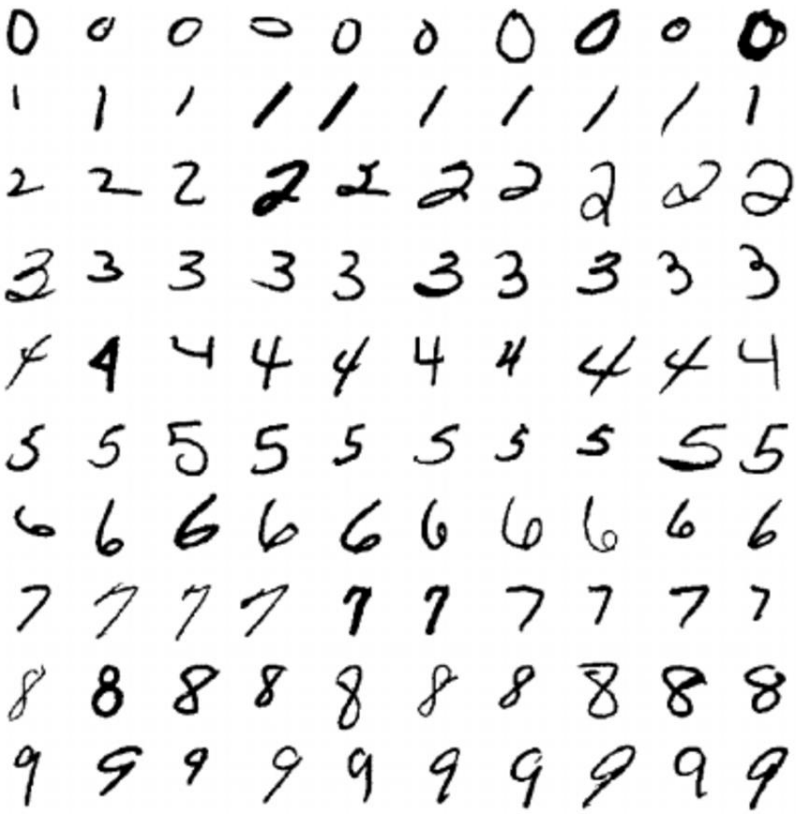
Slika 10. Shema jednostavnog računskog grafa konstruiranog u TensorFlowu. Graf se sastoji od dva ulazna čvora a i b te jednog izlaznog čvora koji računa operaciju zbrajanja nad danim ulazima.

6 Podatkovni skup MNIST

MNIST (*engl. Modified National Institute of Standards and Technology database*) je baza rukom pisanih znamenki. Sastoji se od skupa slika za treniranje veličine 60 000 primjera i skupa slika za testiranje veličine 10 000 primjera. Uobičajeno se koristi za treniranje i testiranje u području strojnog učenja.

Znamenke su normalizirane i centrirane, a svaka slika je dimenzija 28×28 piksela.

Primjeri znamenki iz skupa MNIST prikazani su na slici 11.



Slika 11. Primjeri znamenki iz skupa MNIST.

7 Programska izvedba

Programski kod je organiziran u IPython bilježnice i pokretan na GPU-u (*engl. Graphics Processing Unit*) na servisu Google Colab.

U nastavku slijedi programski kod dva najvažnija algoritma (FGSM i PGD) te programski kod treniranja robusnog modela.

```
1.  def fgsm(model, x, y_, epsilon, session):
2.      loss = tf.nn.softmax_cross_entropy_with_logits(logits=model.output_layer,
3.          labels=y_)
4.      gradient = tf.gradients(loss, model.x)
5.      grad = session.run(gradient, {model.x: x})
6.      adv = np.clip(x + epsilon*np.sign(grad), 0, 1)
7.
8.      return adv, grad
```

Funkcija *fgsm* kao argumente prima *model* (što je u slučaju ovog rada jedan od modela definiranih u poglavlju 8.1), *x* koji predstavlja ulazne podatke, *y_* koji sadrži oznake stvarnih klasa podataka, parametar *epsilon* i *session* koji predstavlja TensorFlow Session (čija funkcionalnost je objašnjena u poglavlju 5).

Nakon evaluacije gubitka i izračuna pripadnog gradijenta uz pomoć TensorFlowa, u varijablu *adv* se sprema izračunata vrijednost neprijateljskog primjera kao što je objašnjeno u poglavlju 3.2.

```
1.  def pgd(model, x, y_, epsilon, session, num_iter):
2.      loss = tf.nn.softmax_cross_entropy_with_logits(logits=model.output_layer,
3.              labels=y_)
4.      grad = tf.gradients(loss, model.x)
5.
6.      adv = np.copy(x)
7.
8.      for i in range(num_iter):
9.          loss_, grad_ = session.run([loss, grad],
10.             {model.x: adv})
11.
12.         adv += 0.01 * np.sign(grad_)[0]
13.         adv = np.clip(adv, x - epsilon, x + epsilon)
14.         adv = np.clip(adv, 0, 1)
15.
16.     return adv
```

Funkcija *pgd* prima iste argumente kao što su opisani za funkciju *fgsm* uz dodatak argumenta *num_iter* kojim se određuje broj iteracija. Također se evaluiraju gubitak i pripadni gradijent pomoću TensorFlowa. U liniji 6 u varijablu *adv* kopiramo originalne ulazne podatke kako bi ih promijenili u neprijateljski primjer. U for petlji za zadani broj iteracija evaluiraju se gubitak i pripadni gradijent pomoću TensorFlowa. Evaluacija se u svakoj iteraciji radi nad varijablom *adv* čija je vrijednost stvorena u prethodnoj iteraciji petlje. Vrijednosti na kraju podrezujemo na interval $[-\epsilon, \epsilon]$, odnosno $[0, 1]$ kako bi dobivene vrijednosti bile unutar dopuštene vrijednosti perturbacije odnosno validne vrijednosti piksela.

```
1.  def train_model(sess, model, num_iter, saver):
2.      for i in range(num_iter):
3.          batch_x, batch_y = mnist.train.next_batch(batch_size)
4.
5.          batch_xx = pgd(model, batch_x, batch_y, 0.3, sess)
6.
7.          sess.run(model.train_opt, feed_dict={model.x: batch_xx, model.y_: batch_
    y})
8.
9.          if i%100==0:
10.             saver.save(sess=sess, save_path='model_cnn', global_step=model.glo
    bal_step, write_meta_graph=False)
11.             loss, acc = sess.run([model.cross_entropy, model.accuracy], feed_dict
    ={model.x: batch_x, model.y_: batch_y})
12.             print("Iteration", str(i), "\t-> Loss =", str(loss), "\t, Accuracy =", str(acc))
```

Funkcija *train_model* kao argumente prima *sess* koji označava TensorFlow Session, *model* koji u ovom slučaju predstavlja konvolucijski model kako je opisan u poglavlju 8.1, *num_iter* koji predstavlja broj iteracija te *saver* koji je objekt TensorFlow klase Saver koja nam omogućuje spremanje modela kojeg treniramo. U petlji za zadani broj iteracija za svaku mini-grupu originalnih podataka pozivamo funkciju *pgd*. Vrijednosti koje vrati funkcija *pgd* predstavljaju podatke koje koristimo za treniranje robusnog modela. Svakih 100 iteracija spremamo model te računamo gubitak i točnost kako bi vidjeli kako model napreduje prilikom treniranja.

8 Eksperimentalni rezultati

8.1 Arhitektura modela

Za testiranje opisanih problema korištene su 3 arhitekture: potpuno povezani model s jednim skrivenim slojem, potpuno povezani model s dva skrivena sloja i konvolucijski model s dva sloja. Arhitektura konvolucijskog modela odgovara modelu koji je opisan u [18]. Detaljan opis arhitekture slijedi u nastavku.

Arhitektura potpuno povezanog modela s jednim skrivenim slojem:

- Ulazni sloj širine 784
- Potpuno povezani sloj širine 1024
- Potpuno povezani sloj širine 10

Arhitektura potpuno povezanog modela s dva skrivena sloja:

- Ulazni sloj širine 784
- Prvi potpuno povezani sloj širine 1024
- Drugi potpuno povezani sloj širine 512
- Potpuno povezani sloj širine 10

Arhitektura konvolucijskog modela:

- Ulazni sloj dimenzija $28 \times 28 \times 1$
- Prvi konvolucijski sloj od 32 filtra dimenzija 5×5 s prijenosnom funkcijom ReLu, uz pomak 1
- Prvi sloj sažimanja s filterom dimenzija 2×2 uz pomak 2
- Drugi konvolucijski sloj od 64 filtra dimenzija 5×5 s prijenosnom funkcijom ReLu (*engl. Rectified Linear Unit*) uz pomak 1
- Drugi sloj sažimanja s filterom veličine 2×2 uz pomak 2

- Potpuno povezani sloj s ulazom 64 matrica dimenzija 7×7, dimenzija 1024, uz prijensnu funkciju ReLU
- Potpuno povezani sloj od 10 čvorova

8.2 Postignuti rezultati

Sva implementacija je testirana na skupu MNIST koji je opisan u poglavlju 6. uz korištenje arhitektura opisanih u poglavlju 8.1.

Postignuta točnost modela prilikom standardnog treniranja je oko 87% za potpuno povezani model s jednim skrivenim slojem, oko 88% za potpuno povezani model s dva skrivena sloja i oko 97% za konvolucijski model.

Isti ti modeli za apsolutnu vrijednost neprijateljske perturbacije 0.1, koja se dodaje pikselima koji su u intervalu $[0,1]$, uz FGSM napad postižu sljedeće točnosti: oko 3.6% za potpuno povezani model s jednim skrivenim slojem, oko 1.3% za potpuno povezani model s dva skrivena sloja i oko 27.9% za konvolucijski model.

Za jednaku vrijednost neprijateljske perturbacije, ali uz PGD napad, točnost potpuno povezanog modela s jednim skrivenim slojem je oko 0.6%, potpuno povezanog modela s dva sloja oko 0.8% i konvolucijskog modela oko 2.3%.

Treniranje s neprijateljskim primjerima je provedeno na konvolucijskom modelu prema [18]. Točnost konvolucijskog modela nakon napada za vrijednost neprijateljske perturbacije 0.1 podignuta je na oko 63% za model treniran kroz 1000 iteracija odnosno na oko 95% za model treniran kroz 40 000 iteracija.

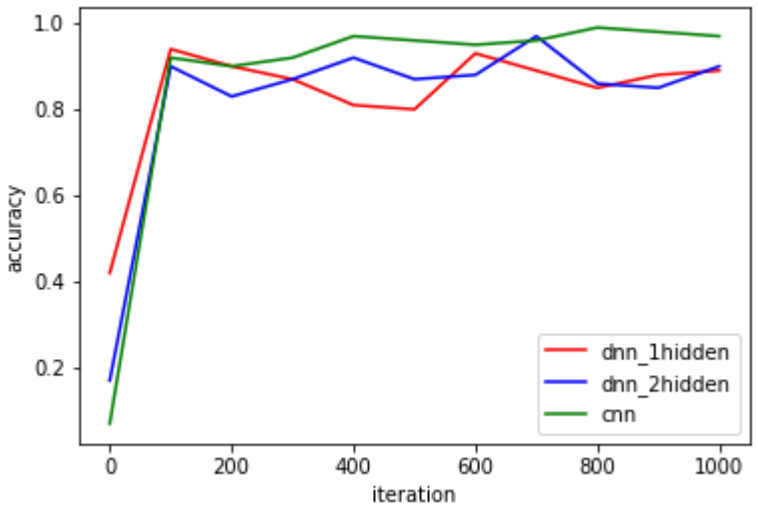
U nastavku će se prikazati učinak usmjerenih i neusmjerenih napada korištenjem algoritama FGSM i PGD.

Također će se prikazati grafovi napredovanja učenja modela nakon standardnog treniranja te točnost nakon napada na tako istrenirane modele uz istu vrijednost

neprijateljske perturbacije. Prikazat će se i graf točnosti ovisno o vrijednosti neprijateljske perturbacije za model istreniran na neprijateljskim primjerima.

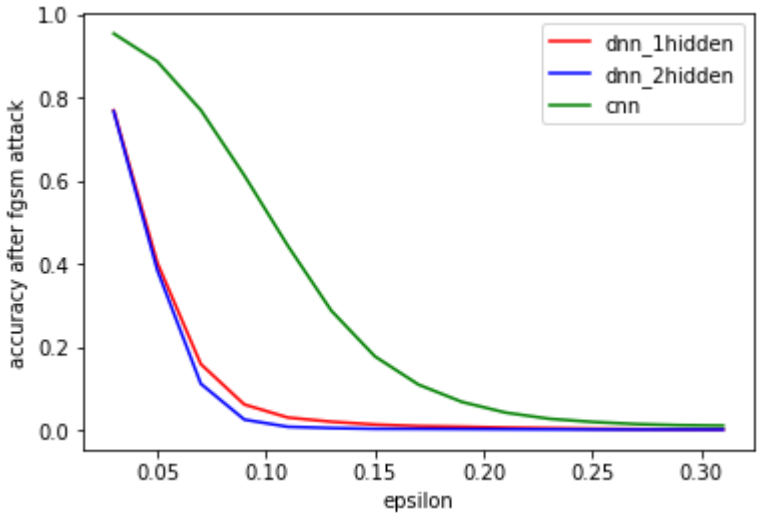
8.3 Prikaz rezultata

8.3.1 Rezultati standardnog treniranja modela i neprijateljskih napada na standardno istrenirane modele



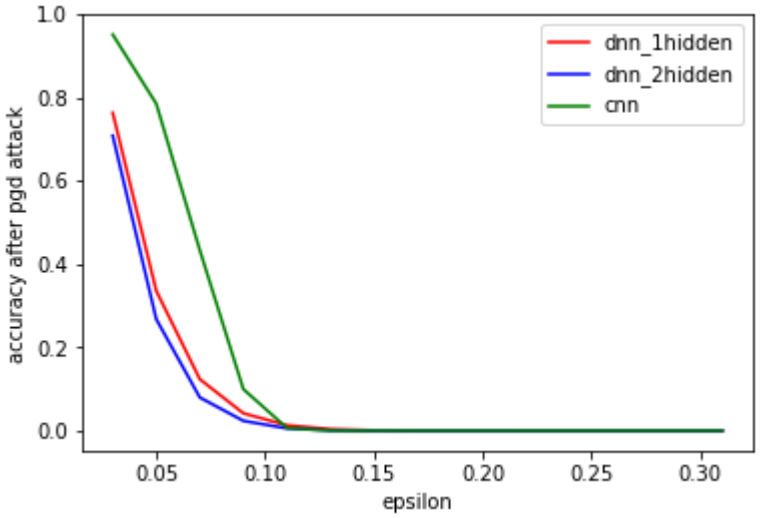
Slika 12. Napredovanje učenja prilikom standardnog treniranja na modelima opisanima u poglavlju 8.1. Crvena linija prikazuje napredovanje učenja potpuno povezanog modela s jednim skrivenim slojem, plava linija napredovanje učenja potpuno povezanog modela s dva skrivena sloja, a zelena linija napredovanje učenja konvolucijskog modela.

Na slici 12 prikazan je graf napredovanja učenja prilikom standardnog treniranja na modelima koji su opisani u poglavlju 8.1. Iz slike vidimo da između tri implementirana modela konvolucijski model postiže najbolje rezultate prilikom standardnog treniranja.



Slika 13. Točnost modela opisanih u poglavlju 8.1 ovisno o neprijateljskoj perturbaciji za FGSM. Crvena linija prikazuje točnost potpuno povezanog modela s jednim skrivenim slojem, plava linija prikazuje točnost potpuno povezanog modela s dva skrivena sloja, a zelena linija točnost konvolucijskog modela ovisno o vrijednosti parametra epsilon.

Na slici 13 vidimo kako se standardno istrenirani modeli ponašaju nakon neprijateljskih napada konstruiranih korištenjem FGSM-a. Vidimo kako je konvolucijski model malo otporniji od potpuno povezanih modela no da u konačnici sva tri modela pokazuju izuzetnu osjetljivost na neprijateljske napade.

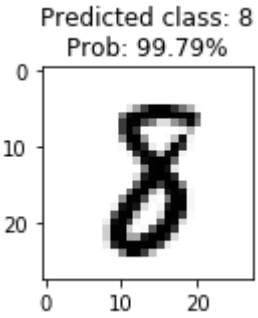


Slika 14. Točnost modela ovisno o neprijateljskoj perturbaciji za PGD. Crvena linija prikazuje točnost potpuno povezanog modela s jednim skrivenim slojem, plava linija prikazuje točnost potpuno povezanog modela s dva skrivena sloja, a zelena linija točnost konvolucijskog modela ovisno o vrijednosti parametra epsilon.

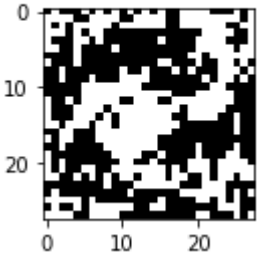
Slika 14 prikazuje kako se ovisno o apsolutnoj promjeni neprijateljske perturbacije mijenja točnost modela za PGD. Iz grafova na slikama 13 i 14 jasno se vidi kako je PGD opasniji napad od FGSM-a, kao što je i spomenuto u poglavlju 4.

8.3.2 Učinak parametra epsilon na izgled neprijateljskog primjera

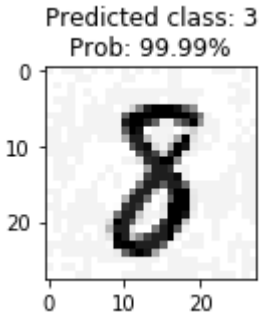
Slike 15 – 17 prikazuju jedan primjer perturbacije koja se dodaje na originalnu sliku te rezultat dodavanja konstruirane perturbacije u originalnu sliku koja tako postaje neprijateljski primjer.



Slika 15. Originalna slika.



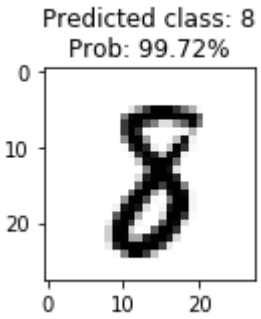
Slika 16. Perturbacija koju dodajemo na originalnu sliku za epsilon 0.1.



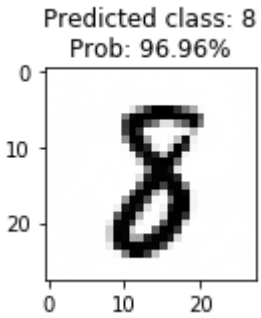
Slika 17. Originalna slika kojoj je dodana perturbacija.

.

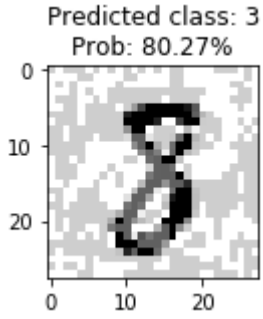
Slike 18 – 22 prikazuju na analogan način dobivene neprijateljske primjere za različite vrijednosti epsilona kako je navedeno u opisu svake slike. Primjećujemo da za određene vrijednosti epsilona slike postaju toliko izmijenjene da ih ni ljudsko oko ne može ispravno prepoznati.



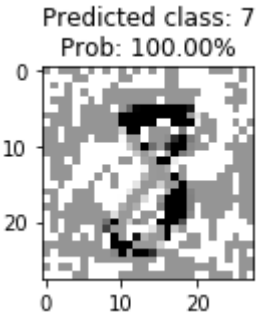
Slika 18. Originalna slika kojoj je dodana perturbacija uz epsilon 0.001.



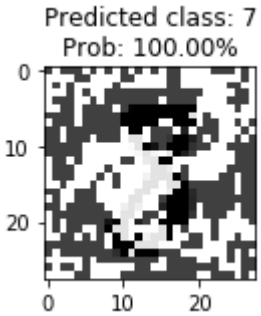
Slika 19. Originalna slika kojoj je dodana perturbacija uz epsilon 0.01.



Slika 20. Originalna slika kojoj je dodana perturbacija uz epsilon 0.3.



Slika 21. Originalna slika kojoj je dodana perturbacija uz epsilon 0.5.

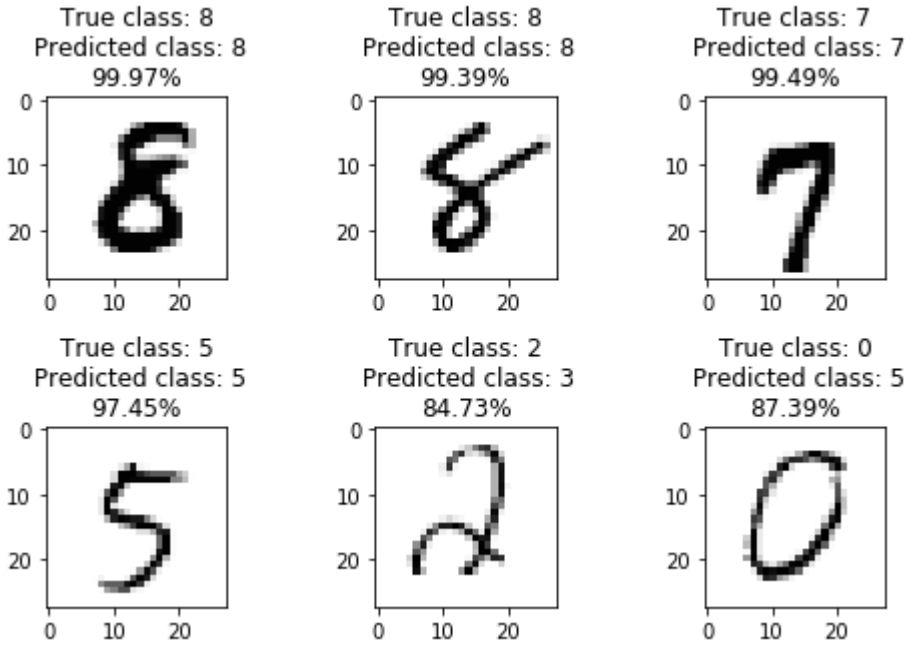


Slika 22. Originalna slika kojoj je dodana perturbacija uz epsilon 0.8.

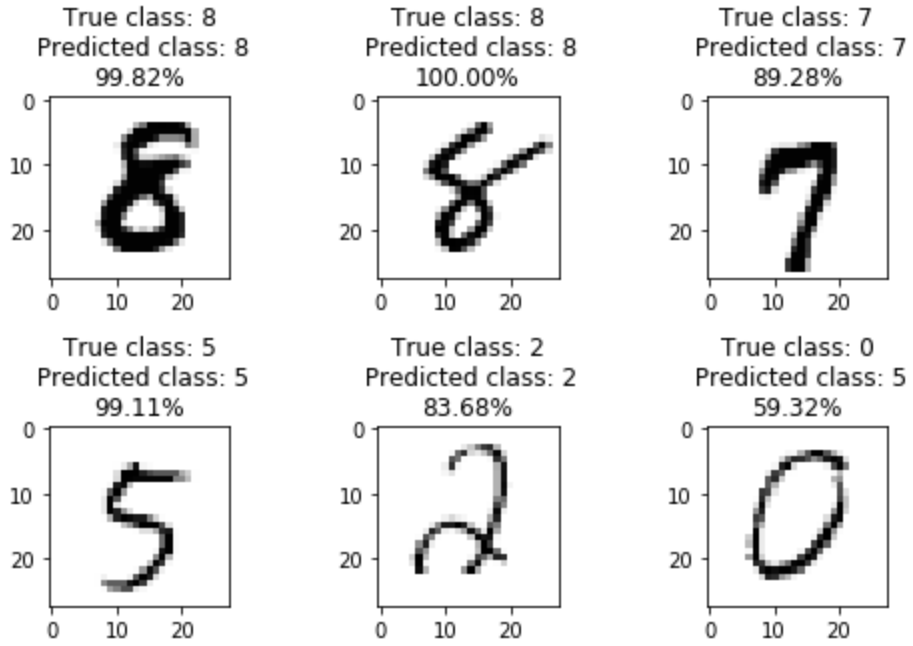
8.3.3 Rezultati na čistim slikama

Na sljedeće tri slike prikazani su rezultati predviđanja na čistim slikama tri modela koja su opisana u poglavlju 8.1.

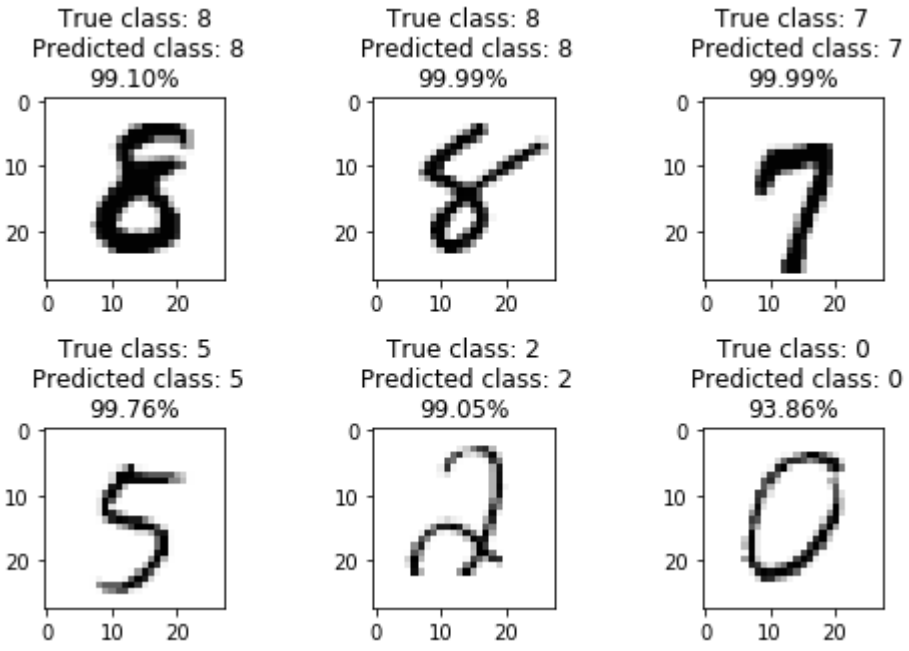
Uzeli smo šest slučajno odabranih brojeva kako bismo prikazali rezultate nakon normalnog treniranja modela. Iznad svake slike napisana je točna klasa, predviđena klasa i točnost s kojom je model predvidio klasu. Na slici 23 prikazani su rezultati klasifikacije potpuno povezanog modela s jednim skrivenim slojem, na slici 24 rezultati klasifikacije potpuno povezanog modela s dva skrivena sloja, a na slici 25 rezultati konvolucijskog modela.



Slika 23. Predviđanje potpuno povezanog modela s jednim skrivenim slojem nakon standardnog treniranja.



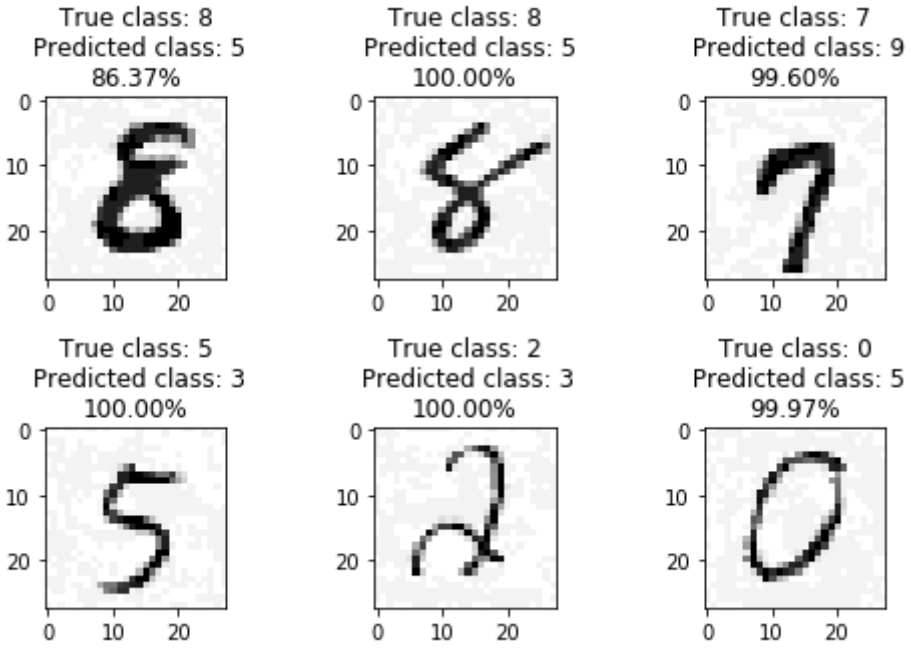
Slika 24. Predviđanje potpuno povezanog modela s dva skrivena sloja nakon standardnog treniranja.



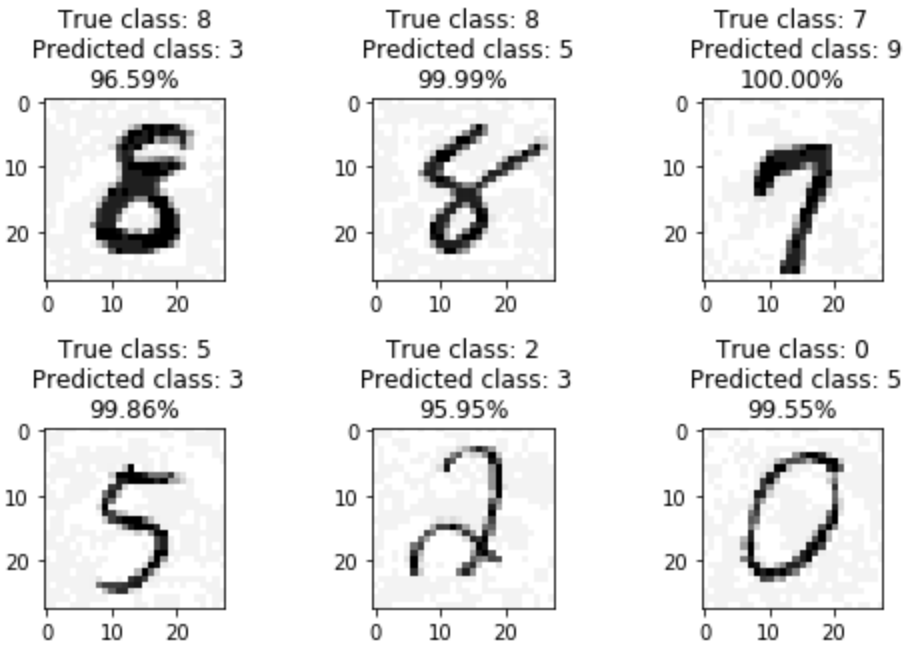
Slika 25. Predviđanje konvolucijskog modela nakon standardnog treniranja.

8.3.4 Učinak FGSM napada

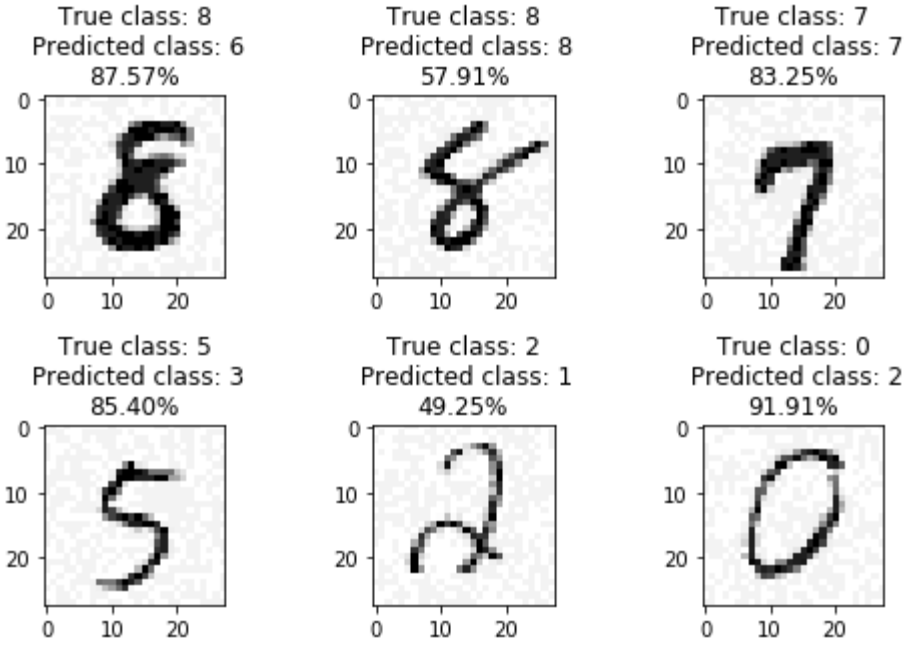
Sljedeće tri slike prikazuju iste primjere kao u poglavlju 8.3.3 nakon što je na njih izvršen FGSM napad. Primjećujemo da modeli s velikom (konvolucijski model) ili vrlo velikom (potpuno povezani modeli) pouzdanošću predviđaju pogrešne rezultate.



Slika 26. Predviđanje potpuno povezanog modela s jednim skrivenim slojem nakon FGSM-a.



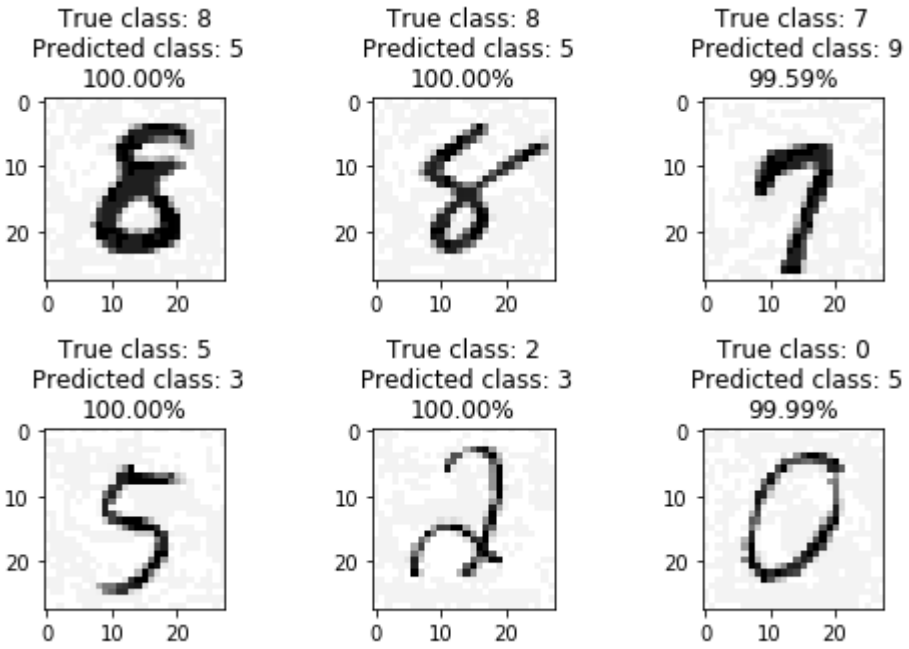
Slika 27. Predviđanje potpuno povezanog modela s dva skrivena sloja nakon FGSM-a.



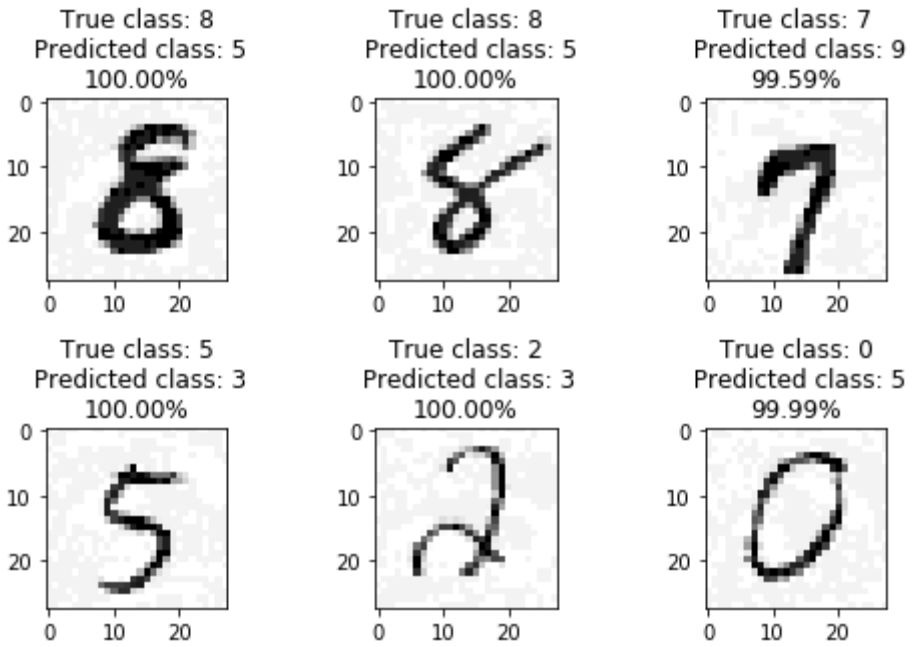
Slika 28. Predviđanje konvolucijskog modela nakon FGSM-a.

8.3.5 Učinak PGD napada

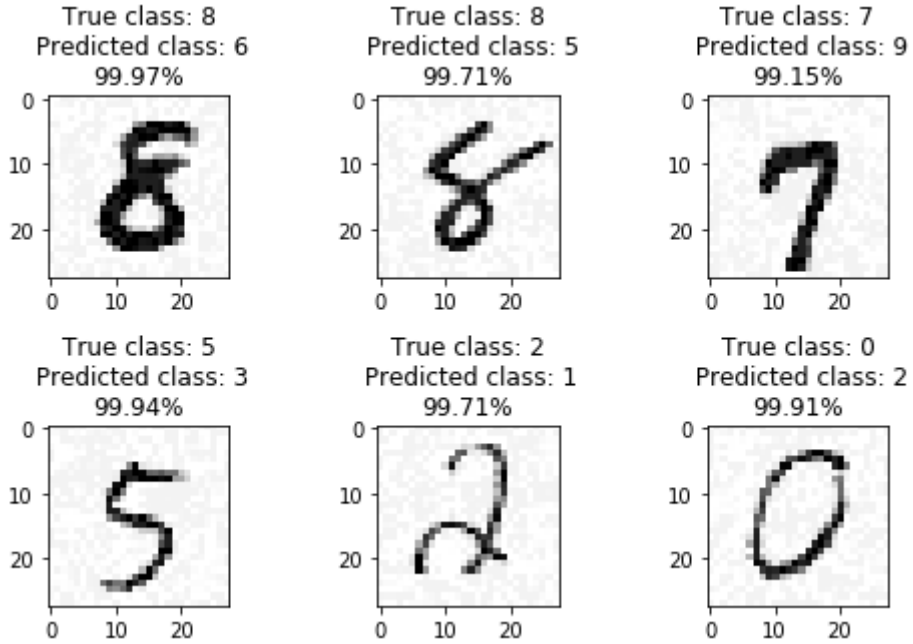
Sljedeće tri slike prikazuju iste primjere kao u poglavlju 8.3.3 nakon što je na njih izvršen PGD napad. Primjećujemo da svi modeli s vrlo velikom pouzdanošću predviđaju pogrešne rezultate, uključujući i konvolucijski model.



Slika 29. Predviđanje potpuno povezanog modela s jednim skrivenim slojem nakon PGD-a.



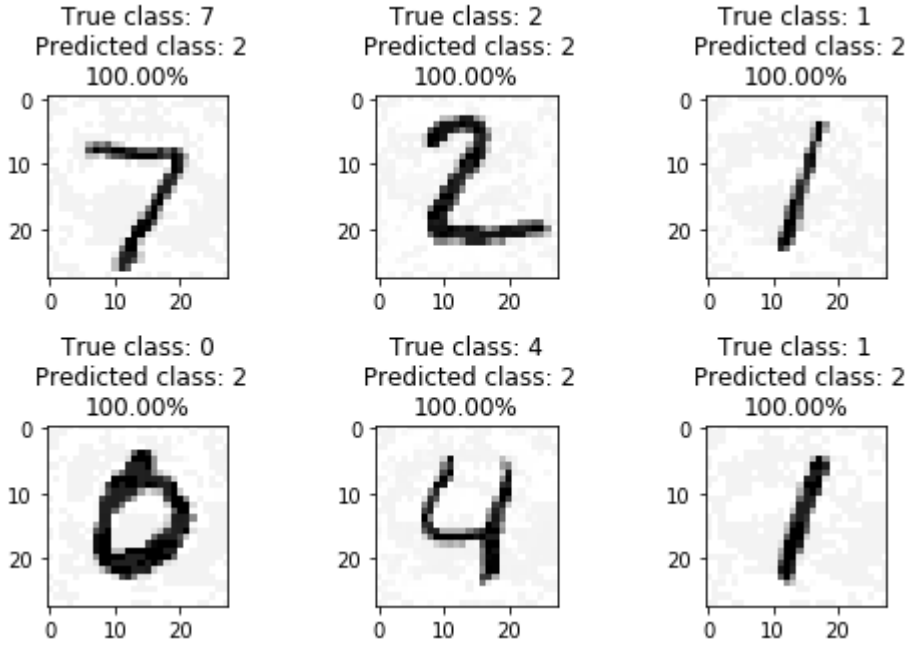
Slika 30. Predviđanje potpuno povezanog modela s dva skrivena sloja nakon PGD-a.



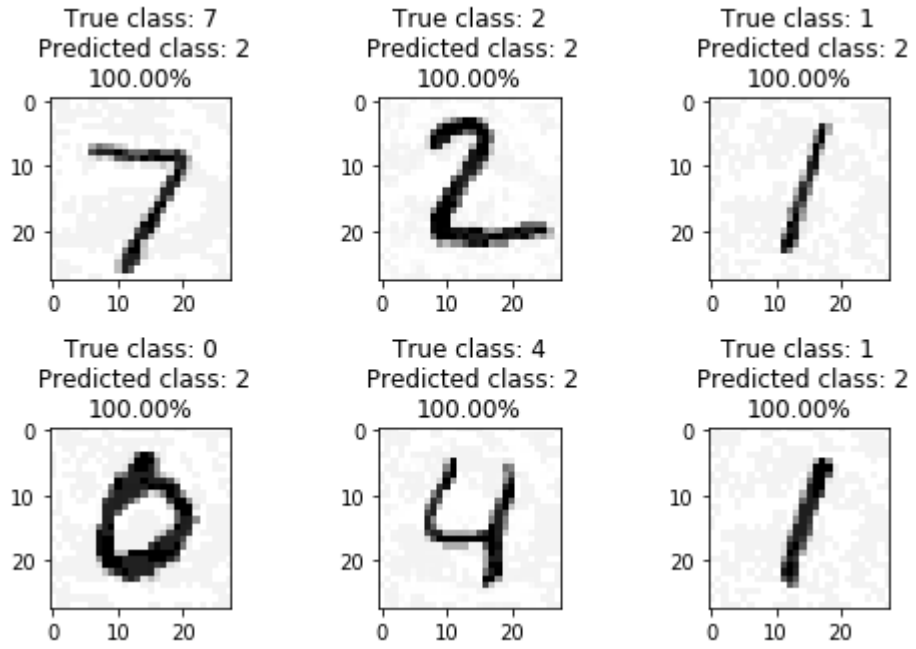
Slika 31. Predviđanje konvolucijskog modela nakon PGD-a.

8.3.6 Učinak usmjerenih napada

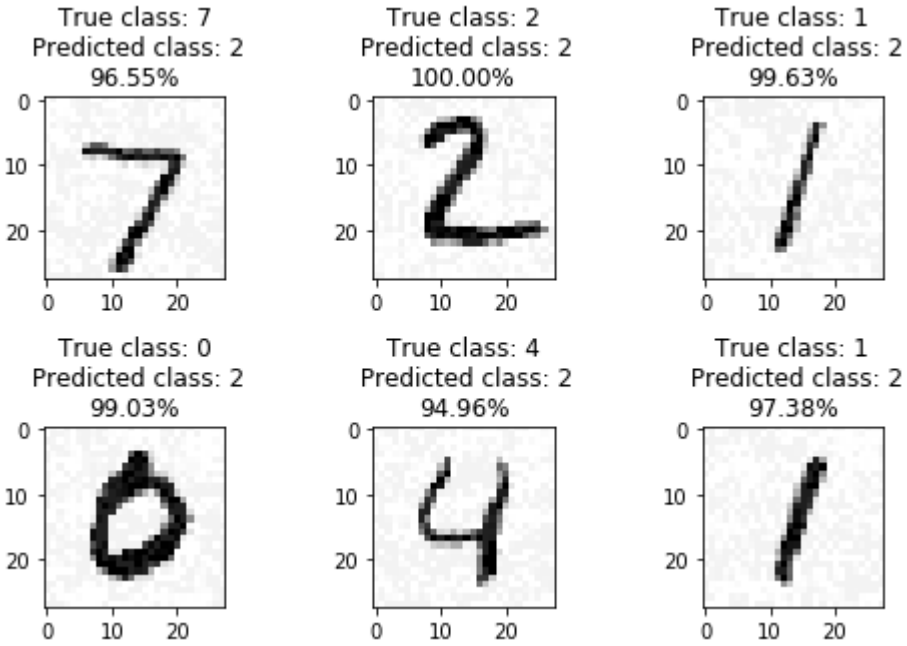
Slike 32, 33, i 34 prikazuju uspješno provede usmjerene napade na sva tri modela. Sva tri modela sve prikazane brojeve klasificiraju kao broj 2 što je upravo rezultat koji smo htjeli postići.



Slika 32. Predviđanje potpuno povezanog modela s jednim skrivenim slojem nakon usmjerenog napada.

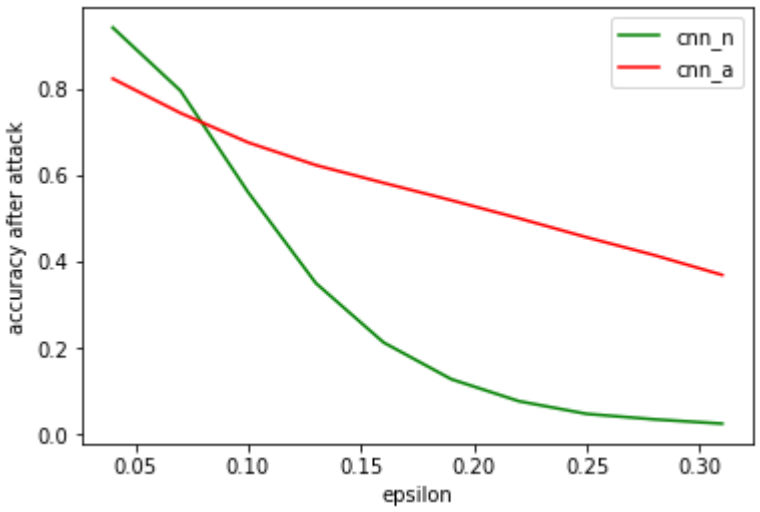


Slika 33. Predviđanje potpuno povezanog modela s dva skrivena sloja nakon usmjerenog napada.



Slika 34. Predviđanje konvolucijskog modela nakon usmjerenog napada.

8.3.7 Točnost robusnih modela

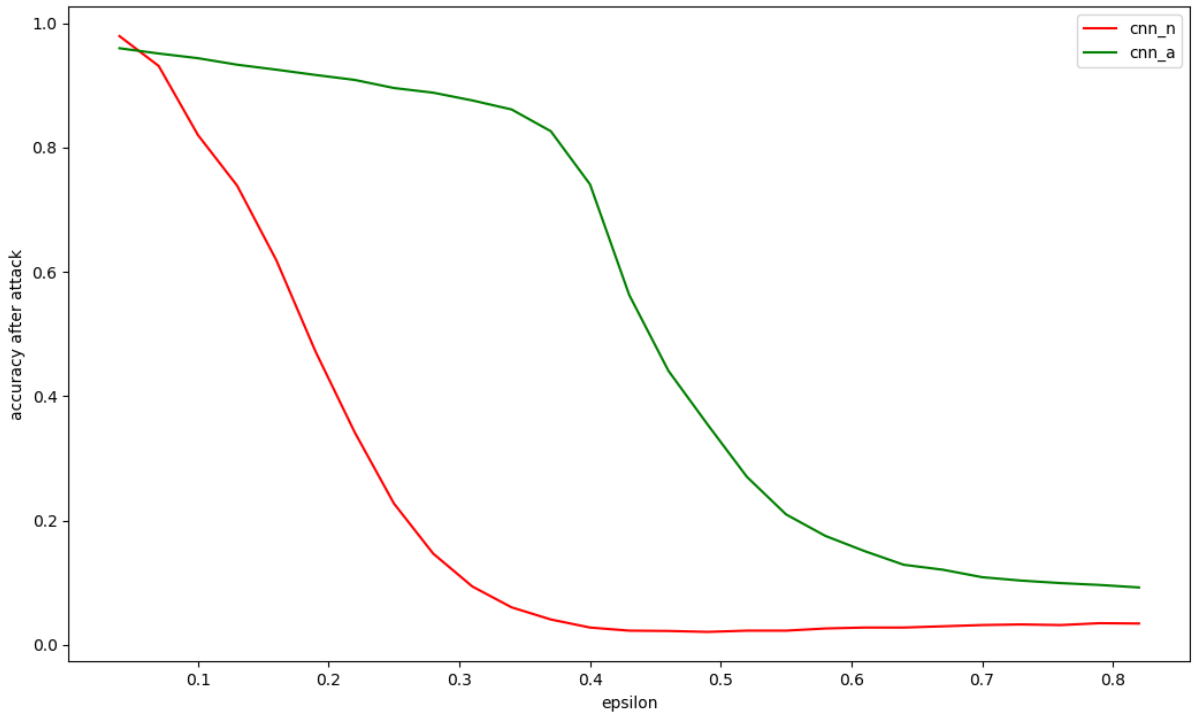


Slika 35. Promjena točnosti konvolucijskog modela istreniranog na standardan način i istreniranog na neprijateljskim primjerima ovisno o vrijednosti parametra epsilon. Oba modela su istrenirana kroz 1000 iteracija. Zelena linija prikazuje standardno istrenirani kovolucijski model, a crvena linija model istreniran na neprijateljskim primjerima.

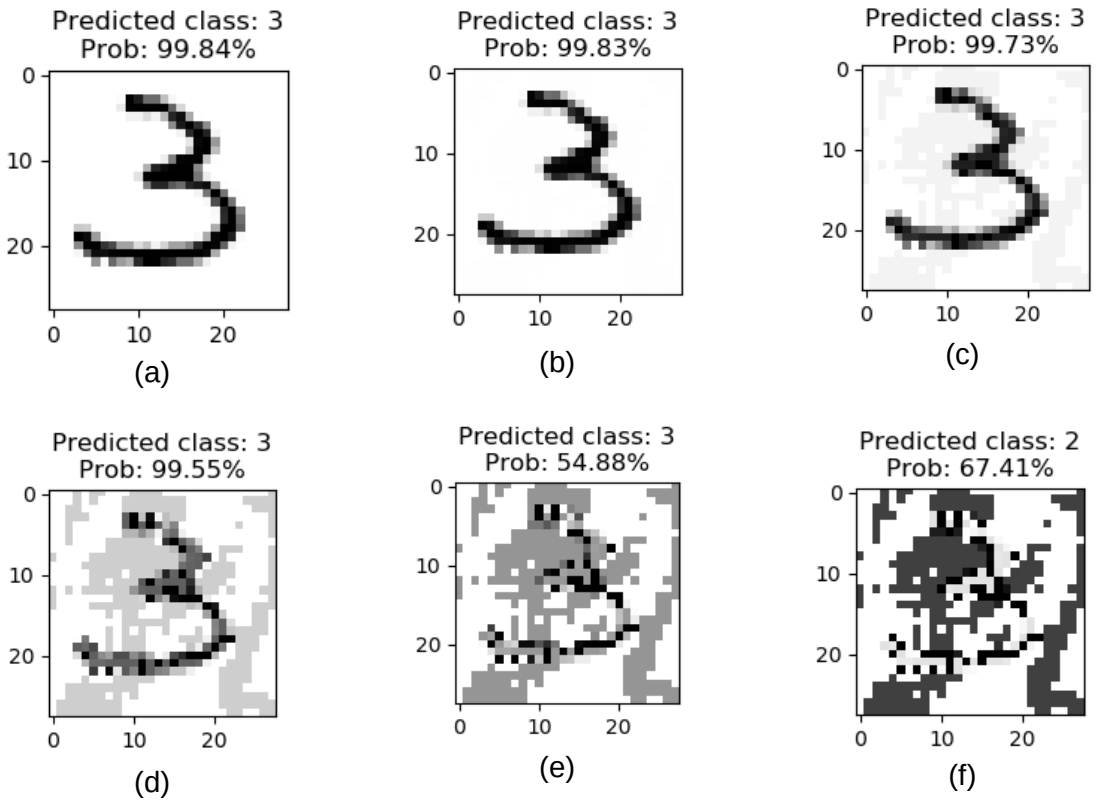
Na slici 35 zelenom linijom prikazan je graf na kojem vidimo kako se mijenja točnost prilikom napada na standardno istrenirani konvolucijski model ovisno o promjeni apsolutne vrijednosti neprijateljske perturbacije. Crvena linija pokazuje kako se u tom slučaju mijenja točnost konvolucijskog modela istreniranog na neprijateljskim primjerima. Oba modela istrenirana su kroz tisuću iteracija. Vrijednosti hiper-parametara korištenih prilikom treniranja konvolucijskog modela na neprijateljskim primjerima su sljedeći: vrijednost koraka učenja je 0.01, vrijednost epsilona 0.3, veličina mini-grupe 50, a broj iteracija PGD-a 40 [18]. Također je prema [18] prije PGD-a za svaku mini-grupu izvršena slučajna inicijalizacija.

Sa slike je vidljiv napredak u otpornosti na neprijateljske napade konvolucijskog modela treniranog na neprijateljskim primjerima u odnosu na modele trenirane na uobičajeni način.

Iako prilikom standardnog treniranja konvolucijskog modela kroz tisuću iteracija naš model postiže točnost oko 97%, te gubitak kod takvog treniranja brzo počinje kovergirati prema 0, takvi rezultati nisu ostvarivi kada je riječ o treniranju s neprijateljskim primjerima. Vrijednost gubitka za konvolucijski model treniran s neprijateljskim primjerima kroz tisuću iteracija je oko 1.2. Taj rezultat nije na razini vrijednosti koju želimo postići kako bi bili sigurni da je optimizacijski problem dobro riješen. Za razliku od standardnog treniranja, vrijednost gubitka prilikom treniranja s neprijateljskim primjerima pada daleko sporije. Za postizanje željenih rezultata potreban je puno veći broj iteracija prilikom treniranja s neprijateljskim primjerima. U [18] vidimo kako je gubitak pao do vrijednosti oko 0.1 kroz 100 000 iteracija. Na slici dolje prikazani su rezultati treniranja na naprijateljskim primjerima koje smo postigli kroz 40000 iteracija. Vidimo kako je postignuto željeno poboljšanje u robusnosti konvolucijskog modela.



Slika 36. Rezultati napada na konvolucijski model ovisno o apsolutnoj vrijednosti neprijateljske perturbacije. Crvena linija prikazuje standardno istrenirani kovolucijski model, a zelena linija model istreniran na neprijateljskim primjerima. Oba konvolucijska modela istrenirana su kroz 40 000 iteracija.



Slika 37. Neprijateljski primjeri dobiveni za epsilon = 0.001 (a), 0.01 (b), 0.1 (c), 0.3 (d), 0.5 (e), 0.8 (f).

Analogno slikama 18 – 22 koje prikazuju neprijateljske primjere kod standardno treniranih modela, na slici 37 vidimo slučaj neprijateljskih primjera za robusni model za različite vrijednosti epsilona kako je navedeno u opisu slike. Primjećujemo kako u ovom slučaju robusni model s vrlo velikom pouzdanošću predviđa točnu klasu podatka do oko vrijednosti epsilona za koji je bio treniran (što je u ovom slučaju vrijednost 0.3) iako slike za tako velik epsilon postaju vidljivo izmijenjene.

9 Zaključak

Ovim radom opisani su optimizacijski problemi s kojima se susrećemo prilikom pokušaja obrane od neprijateljskih napada na modele za prepoznavanje slika. Napadi kao što su FGSM i PGD, koji su detaljno opisani u radu, predstavljaju vrlo velik problem za duboke modele te su zato i bili u fokusu ovog rada. Kao rezultat ovih napada, modeli vrlo često s većom sigurnošću predviđaju netočne rezultate na izlazu nego što predviđaju točne rezultate. S obzirom na uspješnost koju ovi napadi postižu u primjeni na duboke modele, od posebnog je interesa bilo istražiti poznate obrane. Iako je izrazito lako prevariti modele koji uobičajenim treniranjem postižu točnost od 97% na skupu MNIST, obraniti se od ovih napada nije ni približno lagan zadak. Trenutno najbolji rezultat koji smo postigli treniranjem s neprijateljskim primjerima na konvolucijskom modelu istreniranom kroz 40 000 iteracija u trenutku pisanja ovog rada je oko 87% za apsolutnu vrijednost neprijateljske perturbacije od 0.3. Također je prilikom izrade rada primijećeno da hiper-parametri modela imaju izrazit utjecaj na postignutu robusnost modela. Tako je prvotni konvolucijski model sa širinom potpuno povezanog sloja od 128 čvorova nakon treniranja s neprijateljskim primjerima za apsolutnu vrijednost neprijateljske perturbacije 0.3 postizao točnost od 11.3%. Proširivanjem modela na 1024 čvorova postignuto je poboljšanje u točnosti. To pokazuje da robusni modeli zahtijevaju veći kapacitet od modela dobivenih standardnim učenjem. Daljnjim proširenjem modela nisu uočena veća poboljšanja u točnosti.

Sljedeći koraci koji bi se mogli poduzeti u smjeru dodatnih poboljšanja performansi mogli bi biti detaljno provjeravanje kako se model ponaša prilikom različitih kombinacija finog podešavanja parametara poput koraka učenja, veličine mini-grupe i broja iteracija prilikom treniranja. S obzirom da arhitekura modela koji bez imalo teškoća pronalazi optimalno rješenje prilikom normalnog treniranja postiže izrazito loše rezultate prilikom treniranja s neprijateljskim primjerima, bilo bi dobro istražiti kako optimalno oblikovati robusnu arhitekturu. Također bi bilo zanimljivo vidjeti i kako se modeli ponašaju na nekim drugim skupovima podataka kao što je npr. CIFAR.

Prilikom istraživanja ovog, trenutno vrlo aktivnog, područja, postavlja se pitanje mogu li se duboki modeli istrenirati tako da budu otporni na neprijateljske napade? Danas odgovor na to pitanje još uvijek nije potvrdan jer robusni modeli još uvijek ne mogu stati uz bok standardnih modela po svojim performansama. Bit će zanimljivo vidjeti kada će se to dogoditi i što će označiti preokret u istraživanju ovog područja.

10 Literatura

[1] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition." U Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, stranice 1528-1540. ACM, 2016.

[2] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, et al. “Deep speech 2: End-to-end speech recognition in english and mandarin.” in International Conference on Machine Learning, stranice 173-182. 2016.

[3] A. Kurakin, I. Goodfellow, and S. Bengio. “Adversarial examples in the physical world.” arXiv preprint arXiv:1607.02533, 2016.

[4] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song. “Robust physical-world attacks on deep learning models.”, arXiv preprint arXiv:1707.08945, vol. 1., 2017.

[5] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille. “Adversarial examples for semantic segmentation and object detection.” u International Conference on Computer Vision. IEEE, 2017.

[6] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy. Explaining and Harnessing Adversarial Examples, 2014.

[7] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, Alex Kurakin. Large-Scale Evolution of Image Classifiers, 2017.

[8] Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, Jeffrey Dean. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation, 2016.

[9] Mikel Artetxe, Gorka Labaka, Eneko Agirre, Kyunghyun Cho. Unsupervised Neural Machine Translation, 2017.

[10] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis. The Case for Learned Index Structures, 2017.

[11] A. Karpaty. *Software 2.0.* URL https://medium.com/@karpathy/software-2-0-a64152b37c35, datum pristupa: 12.06.2019.

[12] S. Šegvić, *Laboratorijske vježbe iz predmeta duboko učenje*.

[13] A. Dertat. Applied Deep Learning – Part 1: Artificial Neural Networks. URL https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6, datum pristupa: 12.06.2019.

[14] U. Karn. A Quick Introduction to Neural Networks. URL https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/, datum pristupa: 13.06.2019.

[15] I. Šego. *Klasifikacija slika kućnih brojeva dubokim konvolucijskim modelima.* URL http://www.zemris.fer.hr/~ssegvic/project/pubs/sego18bs.pdf, datum pristupa: 13. 06. 2019.

[16] S. Saha. *A Ccomprehensive Guide to Convolutional Neural Networks – the ELI5 way. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53,* datum pristupa: 13.06.2019.

[17] Z.Colter, A. Madry. *Adversarial Robustness -Theory and Parctice*. URL https://adversarial-ml-tutorial.org, datum pristupa: 13.06.2019.

[18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks, 2017.

Neprijateljski napadi na modele za klasifikaciju slike

Sažetak

Ovaj rad dotiče temu diferencijabilnog programiranja kao novog pogleda na područje neuronskih mreža. Definiraju se neprijateljski napadi i opisuje njihova podjela na usmjerene i neusmjerene napade. Implementirana su tri modela na kojima je testirana dana teoretska podloga: dva potpuno povezana modela (s jednim i dva skrivena sloja) i konvolucijski model. U nastavku je opisano treniranje robusnih modela. Detaljno su opisani FGSM i PGD napadi. Predstavljen je kratak osvrt na programski okvir TensorFlow i skup podataka MNIST koji je korišten za testiranje teoretskih kocepata opisanih u radu. Potpuno povezani modeli (s jednim odnosno dva skrivena sloja) i konvolucijski model prilikom standardnog treniranja postižu točnost od oko 87%, 88% odnosno 97%. Prilikom neprijateljskih napada isti modeli za apsolutnu vrijednost neprijateljske perturbacije od 0.1 postižu točnost od 0.6%, 0.8% odnosno 2.3% kada se radi o PGD napadima. Za FGSM napade i jednaku vrijednost perturbacije točnosti su 3.6%, 1.3% i 27.9%. Treniranjem robusnih modela postignuto je poboljšanje u točnosti prilikom napada na 87% za vrijednost perturbacije 0.3. Dodatno, u radu je prikazana i programska implementacija opisanih napada.

Ključne riječi: diferencijabilno programiranje, neprijateljski primjeri, neprijateljski napadi, robusnost, FGSM, PGD, usmjereni napadi, neusmjereni napadi

Adversarial Attacks in Image Classification

Abstract

This work addresses the topic of differentiable programming as a new shift in perspective on the field of neural networks. The definition of adversarial attacks are presented, as well as their classification into two major groups: targeted and untargeted attacks. In the scope of this work, three different models were implemented and used for testing the theoretical background: two dense fully connected neural networks (with one and two hidden layers, respectively), and one convolutional model. The process of training these models is given, as well as a detailed description of the FGSM and PGD

types of attacks. Further, the work presents a brief introduction to the TensorFlow library and the MNIST dataset. During the standard training procedure, the tested models achieved accuracies of 87%, 88% and 97% (respectively), while under adversarial attacks with perturbation of 0.1 the same models achieve accuracies of 0.6%, 0.8% and 2.3% (respectively) for the PGD type, and 3.6%, 1.3% and 27.9% for the FGSM type (respectively). By training the robust model, the accuracy was significantly improved, up to 87% for the perturbation value of 0.3. Additionally, this work also presents the software implementation of the described adversarial attacks.

Key words: differentiable programming, adversarial examples, adversarial attacks, robustness, FGSM, PGD, targeted attacks, untargeted attacks