

UNIVERSITATEA DIN BUCUREȘTI
FACULTATEA DE MATEMATICĂ ȘI
INFORMATICĂ

LUCRARE DE LICENȚĂ

Coordonator științific

Absolvent

Conf. Dr. Popescu Marius

Constantin Octavian-Florin

UNIVERSITATEA DIN BUCUREȘTI
FACULTATEA DE MATEMATICĂ ȘI
INFORMATICĂ

LUCRARE DE LICENȚĂ

Exemple adversariale în învățarea automată

Coordonator științific

Absolvent

Conf. Dr. Popescu Marius

Constantin Octavian-Florin

Rezumat

Într-o lume în care progresul științific reprezintă o modalitate necontestată de a privi spre viitor, tot mai mulți algoritmi ce folosesc învățarea automată își fac loc în viețile oamenilor. Astfel, siguranța folosirii acestora reprezintă un punct de interes prevalent în lumea științifică.

Cercetări recente au arătat că numeroase clasificatoare din domeniul *machine learning*ului pot fi ușor induse în eroare în multiple moduri. În situația *computer vision*-ului, asemenea atacuri se concentrează pe modificarea valorilor pixelilor. În foarte multe cazuri, aceste modificări pot fi atât de subtile încât un observator uman nu are capacitatea de a sesiza diferențele, dar predicțiile rețelelor ar fi complet modificate.

Primii pași în studierea fenomenului exemplelor adversariale au fost făcuți utilizând algoritmi ce aveau acces deplin asupra modelelor atacate. Nu după mult timp, oamenii de știință au reușit performanța de a perturba predicția unei arhitecturi fără a avea nicio informație în legătură cu aceasta. Prin urmare, protecția împotriva exemplelor adversariale reprezintă o nouă provocare pentru cercetători.

Această lucrare expune două modalități diferite prin care acuratețea celor mai utilizate arhitecturi de rețele convoluționale este degradată în mod semnificativ, folosind atât un atac de tip *white-box*, cât și unul de tip *black-box*. Cu scopul observării modului în care structura modelelor influențează generarea exemplelor adversariale, am utilizat diverse tipuri de arhitecturi. În cele din urmă, am utilizat un ansamblu construit din rețelele menționate anterior și am testat capacitățile acestuia în condiții adversariale cu scopul creșterii robusteții arhitecturii folosite.

Abstract

In a world where the scientific progress represents an unquestionable way of seeing the future, numerous machine learning algorithms are becoming more common in human lives. Thus, their security is playing a major role in the scientific field.

Recent researches have shown a great amount of ways in which machine learning algorithms can be deluded. Regarding to computer vision tasks, a remarkable technique of deceiving consists in modifying the values of pixels in images. In the majority of cases, these changes can be so subtle that a human annotator may not observe them but the classifier's predictions may be severely affected.

First steps in researching adversarial examples phenomenon were made using algorithms who got at their disposal full information about the target. Short time later, scientists have attained the performance of wrecking the predictions for an architecture about what they had no prior information. Therefore, the protection against these attacks constitute a new challenge for scientists.

This paper exposes two different ways of degrading the performance of some well-known deep learning algorithms using both white-box and black-box methods. Trying to observe the importance of a network's structure in generating adversarial examples, I used several different architectures. Finally, regarding to an increase in model's robustness I used an ensemble built of previously presented networks and I analyzed its capacities in adversarial conditions.

Cuprins

Rezumat	3
Abstract	4
Cuprins.....	5
Lista figurilor	7
1. Introducere	9
2. Prezentarea temei alese și diverse abordări	11
2.1. Taxonomie.....	11
2.2. Contextul apariției exemplelor adversariale	13
2.3. Apariția exemplelor adversariale	14
2.4. Pericolul existenței exemplelor adversariale	16
2.5. Tehnici de apărare împotriva atacurilor adversariale.....	18
2.5.1. O încercare eșuată – Mascarea Gradientilor.....	18
2.5.2. Antrenarea adversarială.....	19
2.5.3. Transformarea datelor de intrare	19
2.5.4. Utilizarea caracteristicilor robuste	19
3. Tehnologii folosite	21
3.1. Pytorch	21
3.2. Rețele Neurale Artificiale.....	21
3.3. Funcții de pierdere	22
3.3.1. Funcția de pierdere <i>cross-entropy</i>	23
3.4. Funcții de activare.....	23
3.4.1. ReLu.....	24
3.4.2. Softmax	24

3.5.	Deep-learning	25
3.6.	Rețele neurale convoluționale	26
3.6.1.	Arhitectura rețelelor neurale convoluționale	27
3.6.2.	Straturile de convoluție	28
3.6.3.	Straturile de unificare (<i>pooling</i>).....	30
3.6.4.	Operația de flattening și straturile fully-connected.....	31
4.	Prezentarea modului de abordare	32
4.1.	Setul de date.....	32
4.2.	Arhitecturi propuse	33
4.2.1.	ConvNet	34
4.2.2.	LeNet-5.....	34
4.2.3.	ResNet18	36
4.3.	FGSM.....	37
4.4.	SimBa	38
4.5.	Ansamblu de rețele	40
5.	Rezultate obținute.....	42
5.1.	SimBa vs. LeNet-5	42
5.2.	SimBa vs. Ansamblu.....	45
5.3	FGSM vs. ConvNet.....	50
6.	Concluzie	52
	Bibliografie.....	53

Lista figurilor

Figura 2. 1. – Toate imaginile de pe același rând împart proprietăți semantice asemănătoare [4]	14
Figura 2. 2. – O demonstrație a algoritmului FGSM [5].....	15
Figura 2. 3. – Demonstrație a unui atac de tip Black-Box folosind o aplicație de mobil pentru clasificarea imaginilor [15].....	17
Figura 2. 4. – Modele de exemple adversariale clasificate în mod eronat. Clasele prezise de rețea sunt ilustrate sub fiecare imagine [16]	17
Figura 2. 5. – Împărțirea setului de date în seturi robuste și seturi non-robuste [22]	20
Figura 3. 1. – Arhitectura unei rețele neurale cu multiple straturi.....	22
Figura 3. 2. – Funcția de activare ReLu	24
Figura 3. 3. – Modalitatea de extragere a caracteristicilor folosind o rețea neurală convoluțională ([24])	26
Figura 3. 4. – Exemplu de arhitectura a unei rețele convoluționale (LeNet5)	27
Figura 3. 5. – Operația de convoluție și modul în care fiecare câmp receptiv local este reprezentat de neuronii din stratul următor	28
Figura 3. 6. – Reprezentarea vizuală a efectelor filtrelor asupra unei imagini.....	29
Figura 3. 7. – <i>Procedeul de Max-Pooling</i>	30
Figura 3. 8. – <i>Procedeul de flattening</i>	31
Figura 3. 9. – Staturile <i>fully-connected</i>	31
Figura 4. 1. – Exemplu de imagini din setul de date MNIST.....	32
Figura 4. 2. – Distribuția claselor din seturile de antrenare și testare.....	33
Figura 4. 3. – Fiecare coloană arată ce <i>feature maps</i> din stratul 2 sunt combinate cu unitățile din stratul 3.....	35
Figura 4. 4. – Reprezentarea unui bloc rezidual ce folosește <i>identity mapping</i> [27].....	36
Figura 4. 5. – Exemplu adversarial în care clasa imaginii s-a schimbat prin modificarea valorilor unor pixeli.....	38

Figura 4. 6. – Pseudocodul algoritmului <i>SimBa</i> [28].....	39
Figura 4. 7. – Ansamblul construit.....	41
Figura 5. 1. – Acuratețea rețelei în funcție de gradul de perturbare ales	42
Figura 5. 2. – Exemple adversariale generate pentru diferite valori ale lui <i>epsilon</i> . Predicția inițială a modelului este prima valoare, cea de-a doua fiind predicția pentru exemplul dat.....	44
Figura 5. 3. – Numărul mediu de interogări pentru fiecare arhitectură folosită	45
Figura 5. 4. – Acuratețea rețelei ConvNet evaluată pe exemplele adversariale	46
Figura 5. 5. – Acuratețea rețelei LeNet5 evaluată pe exemplele adversariale	47
Figura 5. 6. – Acuratețea rețelei ResNet18 evaluată pe exemplele adversariale	47
Figura 5. 7. – Exemple adversariale ale aceleași imagini. Pe prima coloană, în ordine, se află modelele: Ansamblu, ConvNet, LeNet-5, ResNet18.....	48
Figura 5. 8. – Exemple adversariale ale aceleași imagini. Pe prima coloană, în ordine, se află modelele: Ansamblu, ConvNet, LeNet-5, ResNet18.....	49
Figura 5. 9. – Acuratețea rețelei ConvNet în condiții <i>white-box</i>	50
Figura 5. 10. – Imagini adversariale generate pentru diferite valori ale lui epsilon folosind FGSM	51

1. Introducere

Datorită succesului lor, rețelele neurale convoluționale adânci (*Deep Convolutional Neural Networks CNN*) au devenit în ultimii ani alegerea favorită a dezvoltatorilor în rezolvarea problemelor ce implică procesarea limbajului natural, înțelegerea scrisului și a vorbirii cât și recunoașterea imaginilor și a înțelesului acestora. Pe măsură ce aplicabilitatea acestora crește constant în domenii precum medicina (accelerarea descoperirii tratamentelor pentru anumite boli), sistemul financiar (tranzacțiile frauduloase sunt dezvăluite prin algoritmi de *Machine-Learning*) sau industria transporturilor (automobilele ce se pot conduce singure), siguranța și securitatea acestor modele devine o nouă arie importantă de cercetare.

În 2014, într-o lucrare intitulată „Intriguing properties of neural networks”, Christian Szegedy et al. descoperă o proprietate interesantă a numeroaselor modele de învățare automată, anume vulnerabilitatea acestora împotriva atacurilor adversariale. Prin modificarea unui număr relativ mic de pixeli dintr-o imagine, insesizabili ochiului uman, cercetătorii reușesc să ducă în eroare chiar și cele mai noi și performante arhitecturi ale vremii. Un an mai târziu, Ian J. Goodfellow et al. adaugă prin publicația lor, „Explaining and Harnessing Adversarial Examples”, faptul că un exemplu adversarial poate fi clasificat în mod greșit de o gamă variată de modele de *Machine-Learning* și *Deep-Learning*, chiar dacă acestea au o arhitectură foarte diferită sau au fost antrenate pe subseturi diferite ale aceluiași set de date. Mai mult de atât, aceștia sugerează că asemenea exemple dezvăluie puncte oarbe, fundamentale în algoritmii de învățare folosiți până acum.

Această lucrare are scopul de a prezenta și compara rezultatele a trei rețele neurale convoluționale în fața atacurilor adversariale. Antrenând și folosind doua dintre cele mai performante rețele neurale, LeNet-5 și ResNet18, câștigătoare a concursului ImageNet, cu rezultate impresionante din punctul de vedere al acurateții, și o a treia rețea cu o arhitectură mult mai simplă, mi-am propus să creez un ansamblu format din cele trei ce va fi supus ulterior atacurilor adversariale *black-box* de tip *SimBa*. Voi compara apoi performanțele individuale ale celor trei rețele supuse aceluiași tip de atac, cu rezultatele obținute de ansamblu. În cele din urmă voi prezenta un alt tip de atac adversarial, FGSM (*Fast-Gradient Sign Method*) și voi analiza performanțele acestuia. Am ales să desfășor

experimentele folosind setul de date MNIST datorită numărului mare de exemple și dimensiunii mici a imaginilor.

În următorul capitol voi încerca să prezint în detaliu problema atacurilor adversariale și diversele abordări ale cercetătorilor pe această temă. Capitolul al treilea va descrie tehnologiile și algoritmi folosiți. Totodată voi vorbi și despre tehnicile și metodele de învățare automată utilizate. Capitolele patru și cinci vor cuprinde descrierea setului de date, modalitatea de abordare a problemei și voi prezenta rezultatele obținute în urma studiului de caz. Ultimul capitol va expune o concluzie asupra rezultatelor prezentate anterior.

2. Prezentarea temei alese și diverse abordări

Acest capitol va prezenta tema aleasă și diversele abordări ale cercetătorilor în legătură cu aceasta. Deși apariția exemplelor adversariale este un eveniment destul de recent în lumea *Machine-Learningului*, voi încerca să ofer și un cadru temporal ce va expune evoluția subiectului ales. Totodată, acest capitol va cuprinde o serie de termeni explicați ce vor fi de folos în parcurgerea capitolelor următoare. În cele din urmă, voi include și o serie de modalități de apărare împotriva acestor atacuri.

2.1. Taxonomie

În această secțiune voi prezenta conceptele cheie în contextul atacurilor adversariale ce vor fi utile în parcurgerea materialelor următoare:

- **Adversar:** Entitate ce încearcă să atace modelul de *Machine-Learning* făcându-l să clasifice în mod eronat un set de date de intrare ce arată nealterat. Presupunând ca $i \in c$, unde i este imaginea de intrare, iar c este clasa originală, scopul adversarului este de a face modelul să prezică $i' \in c'$, unde i' este imaginea concepută de adversar astfel încât un observator uman ar clasifica-o ca și când ar aparține clasei c .
- **Perturbație:** Pentru a obține scopul menționat mai sus, un adversar va crea o intrare i' ce va fi obținută prin perturbarea imaginii i prin adăugarea unei impurități $\delta\epsilon$. Imaginea modificată $i' = i + \delta\epsilon$ este numită imagine perturbată sau, în cazul în care atacul are succes, este numită exemplu adversarial. Adăugarea acestei impurități va face imaginea inițială să treacă granița de clasificare. Constrângerile referitoare la perturbație reprezintă o provocare pentru adversar. Toate modificările aduse imaginii inițiale trebuie făcute sub o anumită valoare, astfel încât imaginea perturbată să poată fi totuși încadrată în clasa ei inițială de un adnotator uman.

Distanța $L2$ este adesea folosită în literatură pentru a defini gradul de impuritate admis.

- **Atac țintit:** Modelul clasifică în mod eronat imaginea i' ca făcând parte din clasa c' , iar c' este definită. În cazul unui atac țintit, atacatorul prelucrează imaginea i astfel încât predicția modelului la întâlnirea imaginii prelucrate este clasa c' .
- **Interogare:** O interogare este definită ca fiind o singură instanță de a trimite un set de date de intrare către model și a nota observațiile făcute. Minimizând numărul de interogări făcute va reduce timpul de construire a unui exemplu adversarial, prin urmare, numărul de interogări este un factor cheie în proiectarea unui astfel de exemplu.
- **Modelul atacat:** Modelul care este atacat va defini regulile atacului, care sunt resursele de care dispune adversarul și scopul atacului. Se remarcă două aspecte importante:
 - **Scopul adversarului:** Acesta este cel care stabilește ce dorește să obțină atacatorul din atac. Se disting ținte de securitate precum: atacuri de integritate, atacuri de disponibilitate, atacuri țintite sau atacuri de explorare. Un atacator poate urmări unul sau mai multe scopuri în timpul unui atac.
 - **Capacitățile atacatorului:** Informațiile ce stau la dispoziția atacatorului cum ar fi: datele de antrenare (cu sau fără etichete), parametrii modelului, numărul de interogări pe care le poate face, perioada de timp în care poate lansa atacurile.
- **Atac de tipul *white-box*:** În cazul unui atac *white-box*, adversarul știe orice despre modelul pe care îl va ataca, printre care și informații privind *weight-urile* învățate în urma antrenării, parametrii cu care a fost antrenat modelul, datele de antrenare cu tot cu etichete. Cu toate aceste informații, strategia principală a atacatorului este cea descrisă în [4] sau [5] în care se folosesc gradientii obținuți.
- **Atac de tipul *black-box*:** Prin contrast cu atacurile *white-box*, în cazul unui atac de tip *black-box*, atacatorul are informații limitate asupra modelului și chiar a datelor de antrenare, uneori fără ca acestea să fie etichetate. Un astfel de atac este modelat adesea în jurul observațiilor făcute pe baza predicțiilor modelului sau a gradului de confidență.

- **Transferabilitate:** Termenul de transferabilitate a fost introdus pentru prima dată în [5] și se referă la folosirea unui model antrenat pe același set de date cu modelul ce urmează să fie atacat, primul numindu-se și „model substituit”. Apoi, se încearcă atacuri de tip *white-box* pe modelul substituit obținându-se astfel exemple adversariale ce vor fi folosite pentru a ataca în condiții de *black-box* modelul țintă.
- **Robustețe:** Aceasta reprezintă abilitatea modelelor de a se apăra de atacurile adversariale. Cercetătorii încearcă să mărească gradul de robustețe al modelelor prin diferite tehnici precum: mascarea gradientilor[7, 8, 9], antrenarea cu exemple adversariale[5, 10, 11], modificarea setului de date de intrare [12] și antrenarea în ansamblu[13].

2.2. Contextul apariției exemplelor adversariale

Datorită algoritmilor de *gradient-descent* și *backpropagation*, descoperirii rețelelor neurale convoluționale și a progresului făcut în domeniul *Deep-Learningului* în ultimii ani, foarte multe probleme ce nu puteau fi rezolvate înainte de către calculator, sunt acum rezolvabile. Chiar dacă acești algoritmi au fost inventați acum mult timp (termenul de *back-propagation* și modul acestuia de funcționare a fost introdus pentru prima dată în anul 1986 [1], pe când ideea de *gradient-descent* exista încă din 1847 [2]), un avans considerabil a fost realizat de la prima folosire a procesoarelor grafice. În 2009, Rajat Raina, Anand Madhavan și Andrew Y. Ng, prin lucrarea lor, [3], obțin rezultate surprinzătoare și demonstrează că utilizarea GPU-urilor (*Graphics Processing Unit*) pot aduce performanțe considerabile, aceștia reușind să obțină timpi de învățare de 70 de ori mai rapizi decât folosirea unui procesor *dual-core*. Mai mult de atât, aceștia au redus timpul de așteptare al unei rețele DBN (*Deep Belief Network*) cu 100 de milioane de parametri de la câteva săptămâni, la o singură zi.

În jurul anului 2013 metodele de *Deep-Learning* au reușit să atingă performanțe ce reușesc să concureze cu cele ale oamenilor în domenii precum: *Computer-Vision*, *Object-Recognition* și *Natural Language Processing*. În contextul *Computer Vision-ului*, dacă până în acea perioadă de timp nimeni nu ar fi fost surprins în cazul în care un computer producea o greșeală, în zilele noastre este cel puțin neobișnuit să ne gândim ca un astfel de program ar putea greși în totalitate. Pe fondul acestor factori, problema generării unor imagini ce ar

putea face un clasificator să prezică în mod eronat, cu un grad de confidență ridicat, conținutul acelei imagini, era irelevantă.

2.3. Apariția exemplorilor adversariale

În anul 2014, în [4], Christian Szegedy, Ian Goodfellow et al., în încercarea de a înțelege mai bine rețelele convoluționale și modul în care ele funcționează, aceștia își propun ca, plecând de la o imagine dată, să o poată modifica în așa fel încât predicția unui clasificator să fie schimbată. Ei intenționau să observe caracteristicile definitorii ale unei clase. Spre exemplu, plecând de la o imagine ce indică cifra 5, cercetătorii au încercat să extragă caracteristicile definitorii clasei respective și, folosind aceste informații, își propun să modifice o altă imagine ce urmează să fie clasificată ca cifra 5.

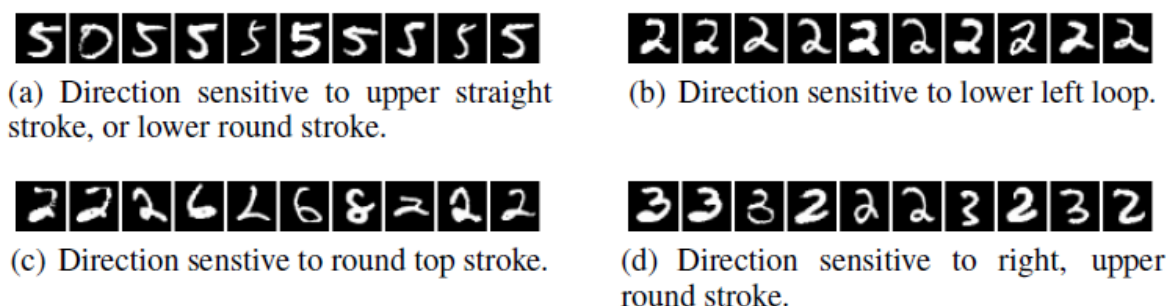


Figura 2. 1. – Toate imaginile de pe același rând împart proprietăți semantice asemănătoare [4]

Oamenii de știință descoperă că pot face modelele să prezică etichete greșite doar modificând mici valori ale pixelilor imaginii normale, fără ca imaginea inițială să fie distrusă, definind astfel noțiunea de exemplele adversariale.

La sfârșitul experimentelor, ei constată că rețelele neurale convoluționale au proprietăți neașteptate și că existența imaginilor adversariale pare a fi în contradicție cu abilitatea rețelor de a obține o performanță mare în generalizare. Prin urmare, dacă o rețea generalizează bine, cum ar putea fi aceasta dusă în eroare de o imagine aproape identică cu una din setul de

antrenare? Autorii oferă două posibile explicații pentru existența exemplilor adversariale: prima posibilitate propusă este aceea că setul exemplilor adversariale este foarte mic, prin urmare acestea nu se regăsesc aproape niciodată în setul de testare sau de antrenare. Cea de-a doua posibilitate este aceea că universul acestor exemple este foarte mare, iar acestea se găsesc aproape oriunde în vecinătatea fiecărui scenariu de testare.

Un an mai târziu, în [5], Ian J. Goodfellow și echipa sa publică o lucrare științifică prin care oferă o explicație pentru existența acestor exemple și totodată dezvăluie anumite direcții noi de cercetare în această arie.

Articolul atestă faptul că exemplele adversariale apar din cauza comportamentului liniar în spații multidimensionale ale modelelor. Modul acesta de a vedea problema le-a permis să creeze un algoritm numit FGSM (*Fast Gradient Sign Method*) pentru generarea exemplilor adversariale. Mai mult de atât, aceștia îl vor folosi pentru a crea exemple pe care ulterior le vor utiliza în a antrena rețelele și a vedea dacă acestea devin mai robuste.

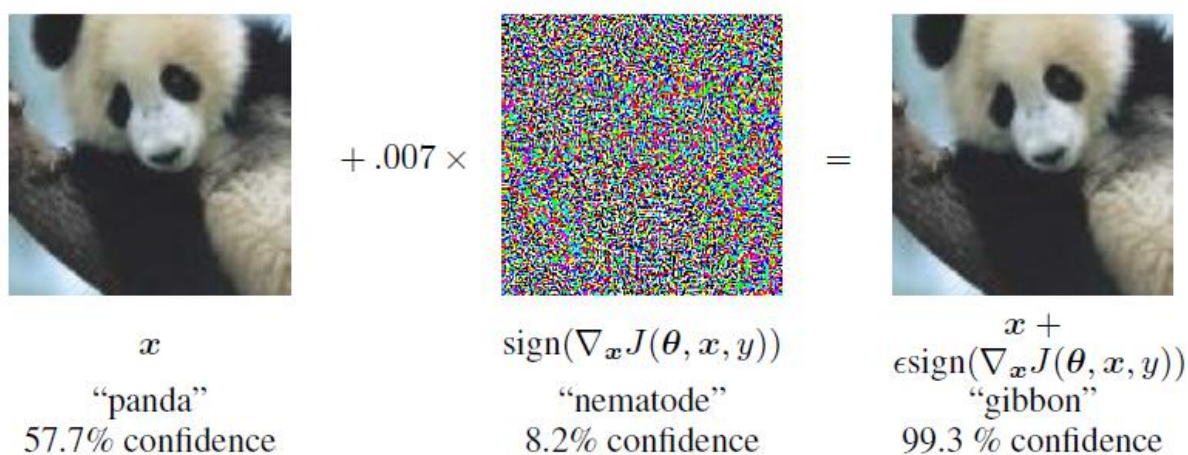


Figura 2. 2. – O demonstrație a algoritmului FGSM [5]

Figura 2.2. ilustrează o demonstrație a algoritmului FGSM aplicat rețelei GoogLeNet [5], în care, plecând de la o imagine cu panda și adăugând un vector mic ale cărui elemente sunt egale cu semnul elementelor gradientului dat de funcția de cost (*loss function*) specifice lui x , unde x este clasa din care face parte poza. Ce este de remarcat la **Figura 2.2** este faptul că gradul de încredere al rețelei în ceea ce a prezis a crescut considerabil chiar dacă prezicerea era una eronată.

Pornind de la setul de date MNIST și generând 60,000 de exemple adversariale, autorii au reușit să scadă rata de eroare a rețelei de la 89,4% pe exemple adversariale până la 17.9%. Din păcate, atunci când modelul antrenat pe exemple adversariale clasifica în mod greșit o imagine, gradul de încredere al acestuia era foarte mare. S-a obținut un grad de încredere mediu de 81.4%.

În urma experimentelor făcute, Ian J. Goodfellow et al. își prezintă concluziile și observațiile. Aceștia consideră ca exemplele adversariale sunt rezultatul faptului că modelele sunt prea liniare și remarcă proprietatea mai multor rețele de a fi duse în eroare de același exemplu, punând acest aspect pe seama că modelele învață funcții asemănătoare când sunt puse să execute aceeași sarcină. Se evidențiază că antrenarea adversarială poate regulariza modelul chiar mai mult decât dacă este folosită o tehnică de *dropout*. Un alt aspect important este acela că rețelele de tip RBF (*Radial Basis Function*) sunt rezistente la astfel de atacuri.

2.4. Pericolul existenței exemplurilor adversariale

Deși cele mai recente cercetări din domeniu asumau contextul în care modelul atacat era în contact direct cu atacatorul, acestea nu corespundeau situațiilor din lumea reală unde modelelor de *machine-learning* le sunt transmise imagini ce provin de la camere video sau alte tipuri de senzori. În [14] se discută aplicabilitatea exemplurilor adversariale în lumea reală. Autorii demonstrează că sistemele de învățare automată sunt vulnerabile și în condiții reale. Folosind imagini adversariale fotografiate cu un telefon mobil, iar apoi trimițându-le către clasificatorul ImageNet Inception, cercetătorii arată modul în care acestea au fost clasificate eronat.

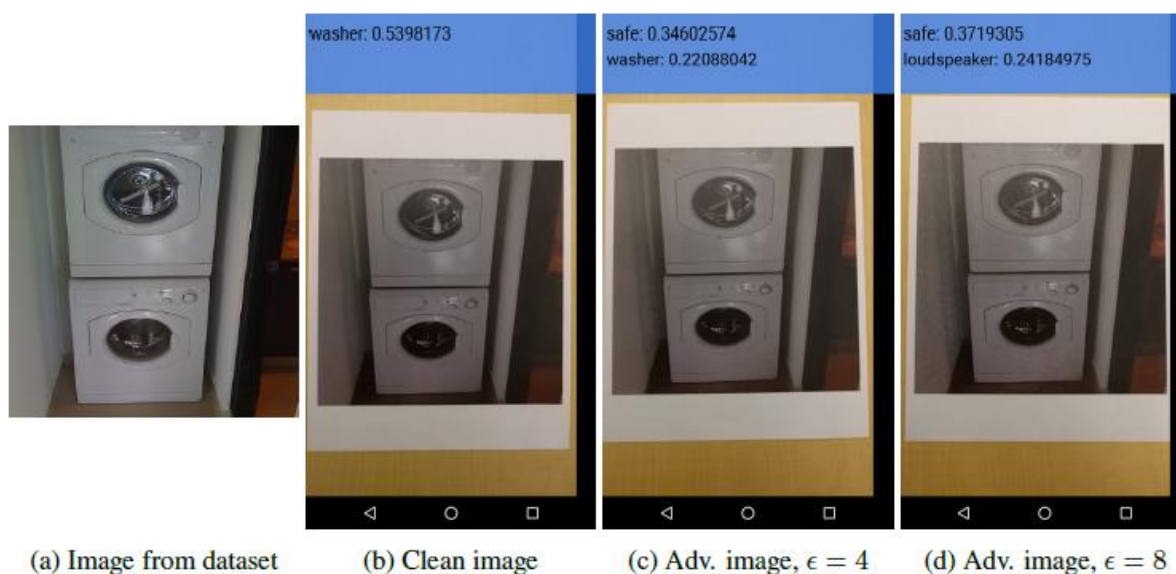


Figura 2. 3. – Demonstrație a unui atac de tip Black-Box folosind o aplicație de mobil pentru clasificarea imaginilor [15]

Exemplele adversariale au potențialul de a fi periculoase. Spre exemplu, oamenii de știință au încercat să atace mașinile ce se conduc singure folosind bucăți de hârtie colorată sau vopsea pe semnele de circulație. Ei au modificat un semn de „Stop” în așa fel încât automobilul l-a perceput ca fiind un semn de „Cedează trecerea”. În [16], antrenând un model substituit folosit la generarea de exemple adversariale, autorii au reușit să atace rețele neurale adânci folosite în lumea întreagă: DNN-ul (*Deep Neural Network*) găzduit de MetaMind a clasificat în mod eronat 84.24% din exemplele adversariale generate anterior. Aceștia au repetat același scenariu și împotriva rețelelor găzduite de Amazon și Google, Amazon obținând o rată de eroare de 96.19%, iar Google de 88.94%.



Figura 2. 4. – Modele de exemple adversariale clasificate în mod eronat. Clasele prezise de rețea sunt ilustrate sub fiecare imagine [16]

Potrivit noilor cercetări din cadrul UC Berkeley, OpenAI, și Pennsylvania State University [18], agenții de învățare automată prin recompensă au fost și ei dezvăluiți a fi vulnerabili împotriva atacurilor adversariale. [17] confirmă aceeași descoperire, adăugând algoritmi precum DQN (*Deep Q Learning*), TRPO (*Trust Region Policy Optimization*), A3C (*Asynchronous Advantage Actor-Critic*) pe lista algoritmilor ce pot fi duși în eroare.

2.5. Tehnici de apărare împotriva atacurilor adversariale

2.5.1. O încercare eșuată – Mascarea Gradienților

Printre primele metode de apărare a împotriva atacurilor adversariale a fost „Mascarea Gradienților”, termen ce a fost introdus în [16] și a dat naștere unei întregi game de algoritmi meniți să crească robustețea modelelor.

Remarcând faptul că numeroase modalități de generare a exemplelor adversariale folosesc gradientii modelelor pentru a lansa un atac eficient, cercetătorii au avut ideea de a masca gradientii. Algoritmii clasici ce utilizează gradientii modelelor, precum FGSM și I-FGSM erau folosiți pentru a-și planifica următoarele direcții în care vor fi modificați pixelii imaginilor [4, 5]. Cercetătorii au crezut că lipsa gradientilor va opri posibilitatea generării exemplelor adversariale.

Ulterior s-a dovedit că metoda mascării gradientilor nu îmbunătățește robustețea modelului, ci doar îngreunează timpul de generare al exemplelor. Folosind un model substituit ce simulează modelul pus sub atac, [19] au arătat că această tehnică de ascundere a gradientilor nu este utilă împotriva atacurilor.

2.5.2. Antrenarea adversarială

În [5], autorii au prezentat o metodă ce a constituit un fundament solid pentru cercetările legate de robustețea modelelor. Aceștia au propus termenul de antrenare adversarială prin care modelele erau antrenate pe seturi mari de exemple adversariale. Cercetătorii au redus rata de eroare a modelelor de la 89.4% până la 17.9%. Totuși, gradul de confidență mediu, atunci când modelul făcea o predicție greșită a fost 81.4%.

2.5.3. Transformarea datelor de intrare

Autorii [20] au testat mai multe modalități de a modifica datele de intrare cu scopul creșterii robusteții modelelor: tăierea imaginilor și redimensionarea acestora, reducerea numărului de culori, compresarea imaginilor și decompresarea lor, minimizarea varianței totale și algoritmi de *image quilting* [21]. Aceștia au concluzionat că minimizarea varianței totale și *image quilting* au fost mecanisme eficiente de apărare împotriva atacurilor atunci când modelele au fost antrenate pe imagini transformate, ei reușind să clasifice în mod corect o proporție de 80%-90% din exemple folosind 4 tipuri diferite de atacuri. S-a obținut o diferență L_2 între imaginile din setul de date și imaginile prelucrate de 0.08.

2.5.4. Utilizarea caracteristicilor robuste

În 2019, cercetătorii [22] propun o perspectivă nouă asupra fenomenului de exemple adversariale. În contradicție cu cercetările de până atunci, aceștia pun existența exemplelor adversariale pe baza învățării supervizate, afirmând ca ele sunt o consecință directă a modelelor de a generaliza caracteristici în datele pe care le primesc. Modelele sunt antrenate pentru a-și maximiza acuratețea, prin urmare ele folosesc orice semnal disponibil pentru a face asta, chiar dacă semnalele sunt sau nu inteligibile oamenilor. După toate acestea, prezenta unei cozi sau a unor urechi nu sunt niște caracteristici mai însemnate pentru un clasificator decât sunt altele pe care oamenii nu le pot înțelege.

Pentru a demonstra cele afirmate, oamenii de știință stabilesc trei tipuri de caracteristici fundamentale: caracteristicile utile, caracteristici robuste utile și caracteristici non-robuste utile. Utilitatea unei caracteristici este dată de existența unei corelări directe între aceasta și eticheta clasei. O caracteristică a unei imagini se definește ca fiind utilă robustă dacă, supusă unei perturbări, aceasta încă rămâne utilă, pe când o caracteristică non-robustă este o caracteristică ce își pierde utilitatea în cadrul perturbării imaginii. Autorii susțin că astfel de *features* ajută la clasificare în condiții standard, dar scad acuratețea modelelor în cazul atacurilor adversariale.

Plecând de la un set de date de antrenare, aceștia își propun să îl secționeze în două părți. Prima parte a lui va consta în caracteristici robuste, iar cealaltă va fi reprezentată de caracteristicile non-robuste. Apoi, antrenând o rețea doar cu datele robuste, aceasta ar trebui să obțină o acuratețe foarte bună în condiții standard, cât și în condiții adversariale.

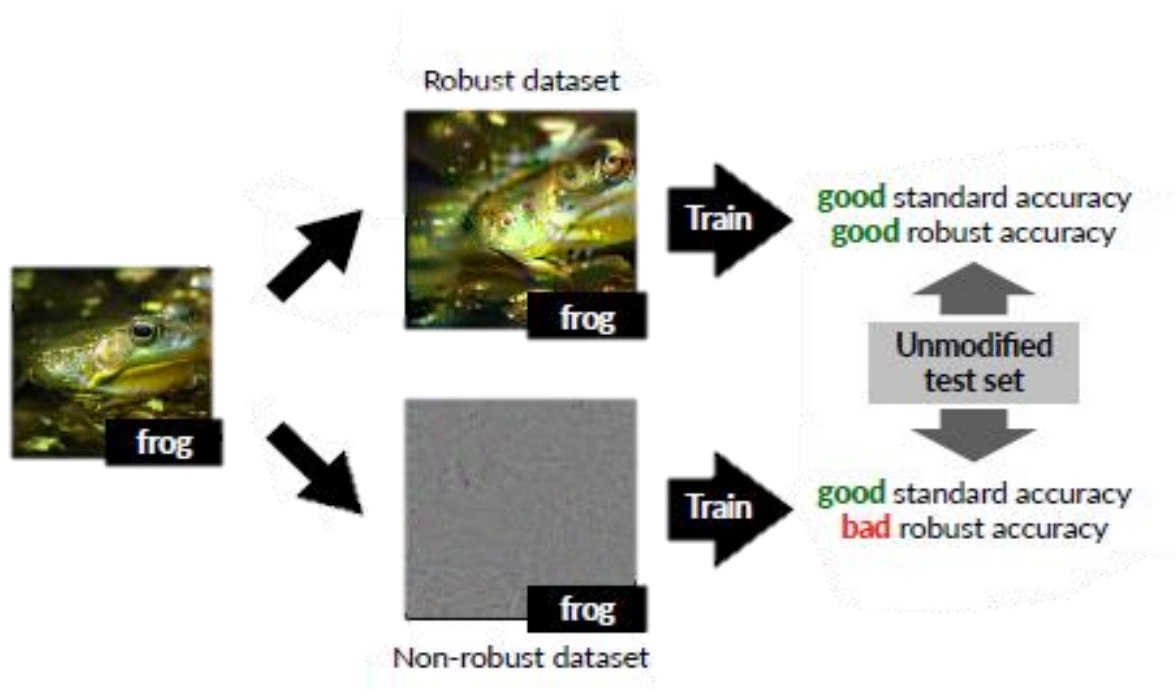


Figura 2. 5. – Împărțirea setului de date în seturi robuste și seturi non-robuste [22]

Deși autorii stabilesc o bază teoretică solidă pentru modul în care se pot folosi caracteristicile robuste și cele non-robuste împotriva exemplurilor adversariale, aceștia folosesc în teorie un clasificator robust, care nu există momentan. Astfel, cercetări ulterioare vor fi necesare pentru a clarifica această direcție.

3. Tehnologii folosite

3.1. Pytorch

Pytorch este un pachet științific ce se bazează pe Torch și Python, ce dispune de o diferită implementare a obiectelor încât acestea să poată utiliza placa video în realizarea calculelor. Acest lucru îmbunătățește considerabil timpii de rulare ai programelor. Pytorch a fost creat cu scopul de a servi utilizatorilor o platformă de *Deep-Learning* ce dispune de flexibilitate și viteză.

Întrucât Pytorch dispune de o interfață simplă, foarte asemănătoare Python-ului, grafuri computaționale, viteze foarte bune de calcul și suport în dezvoltarea proiectelor în domeniul *Machine-Learningului* și *Deep-Learningului*, incluzând sisteme de calcul diferențial automate și modele pre-antrenate, am ales a-l folosi în rezolvarea problemei propuse.

3.2. Rețele Neurale Artificiale

Plecând de la modelul biologic al procesării și transmiterii informației din creierele ființelor vii, unde fluxul de date este transmis în mod constant între neuroni, aceștia formând rețele de informații, cercetătorii și au propus să relice acest sistem. Astfel, a luat naștere conceptul de rețea neurală artificială ce reprezintă un sistem de unități numite neuroni artificiali (ce au scopul de a imita neuronii umani) ce sunt interconectați și procesează informația.

Un neuron artificial primește un semnal care de cele mai multe ori este reprezentat de un număr real. Ulterior, acest semnal este procesat de neuron și trece printr-o funcție non-liniară, numită funcție de activare. Dacă valoarea obținută este peste un anumit prag, aceasta este propagată în continuare. De cele mai multe ori, neuronii sunt organizați în straturi ce au ca scop diferite transformări ale datelor primite.

Utilitatea rețelelor neuronale este dată de faptul că acestea, prin ajustarea unor parametri interni numiți *weight* și *bias*, au capacitatea de a învăța funcții matematice complexe.

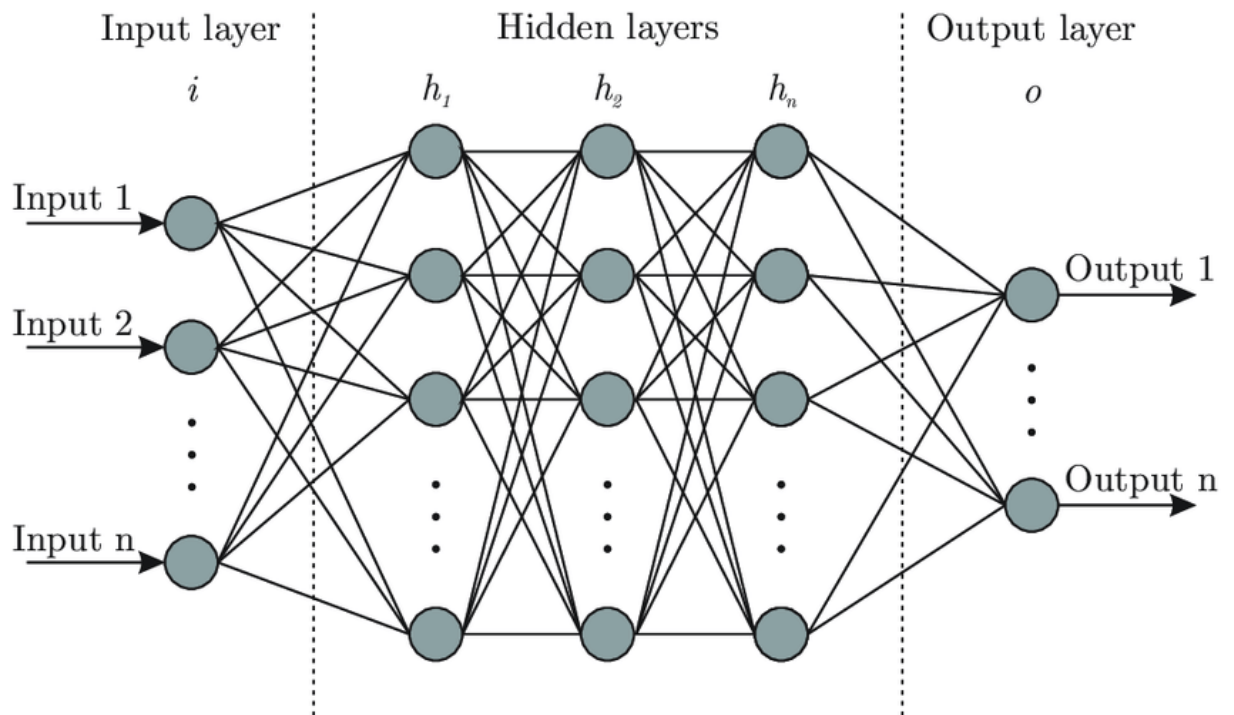


Figura 3. 1. – Arhitectura unei rețele neurale cu multiple straturi

Unul dintre cele mai utilizate tipuri de rețele sunt rețelele de tip *feed-forward* în care conexiunile dintre neuroni nu formează un ciclu, iar informația circulă de la neuronii de *input*, prin straturile ascunse, până la stratul de ieșire. Pentru antrenarea acestor rețele se folosește, de regulă, un algoritm de *backpropagation* care are rolul de calcula gradientul funcției de pierdere (*loss function*) în raport cu *weight-urile* rețelei pentru un anumit set de date de intrare și ieșire.

3.3. Funcții de pierdere

O funcție de pierdere este o metodă utilizată pentru a cuantifica distanța până la obiectivul ales. Acestea sunt folosite pentru antrenarea modelelor de *machine-learning*. Astfel, pentru fiecare pereche *input-output*, se va calcula cu ajutorul unei funcții de pierdere cât de aproape

este predicția algoritmului față de eticheta originală. Apoi, în funcție de rezultatul dat de funcția de pierdere, se vor ajusta parametrii modelului.

3.3.1. Funcția de pierdere *cross-entropy*

Se utilizează o funcție de pierdere *cross-entropy* pentru a măsura performanțele unui model al cărui *output* este o probabilitate între 0 și 1. Valoarea acestei funcții va scădea cu cât etichetele prezise de model sunt mai exacte. Astfel, atunci când clasificatorul va acorda probabilitatea de 0.19 pentru a fi o pisică, unei imagini în care chiar este o pisică, valoarea funcției *cross-entropy* va fi mare, ceea ce înseamnă că predicția modelului este departe de adevăr. În urma procesului de antrenare, se urmărește minimizarea constantă a funcției de pierdere.

3.4. Funcții de activare

Inspirate din modul în care neuronii umani funcționează, funcțiile de activare reprezintă o abstractizare a procesului de a transmite informații în interiorul creierului uman. În momentul în care un neuron primește destulă energie, acesta se activează și transmite impulsul nervos mai departe. Replicând acest proces, funcțiile de activare sunt utilizate pentru a procesa informația primită de la fiecare neuron al unei rețele, acestea având rolul de a decide dacă informația primită va fi transmisă sau nu în continuare celorlalți neuroni.

Funcțiile de activare sunt ecuații matematice ce au rolul de a determina *output*-ul unei rețele. Întrucât ele se află legate de mii sau milioane de neuroni, funcțiile de activare trebuie să fie eficiente din punct de vedere computațional. Mai mult de atât, ele au rolul de a adăuga non-linearitate în capacitățile modelului. În proiectul ales am folosit două tipuri de funcții de activare: *ReLU* și *Softmax*.

3.4.1. ReLu

Rectified linear unit (ReLu) este o funcție lineară ce se folosește preponderent în multiple tipuri de rețele convoluționale. În cazul în care aceasta primește o valoare negativă, va întoarce valoarea 0, iar pentru orice valoare pozitivă, va returna valoarea primită. *ReLu* este cea mai folosită funcție de activare datorită faptului că facilitează algoritmi de *gradient-descent* și rezolvă problema gradientilor care dispar (*vanishing gradients*).

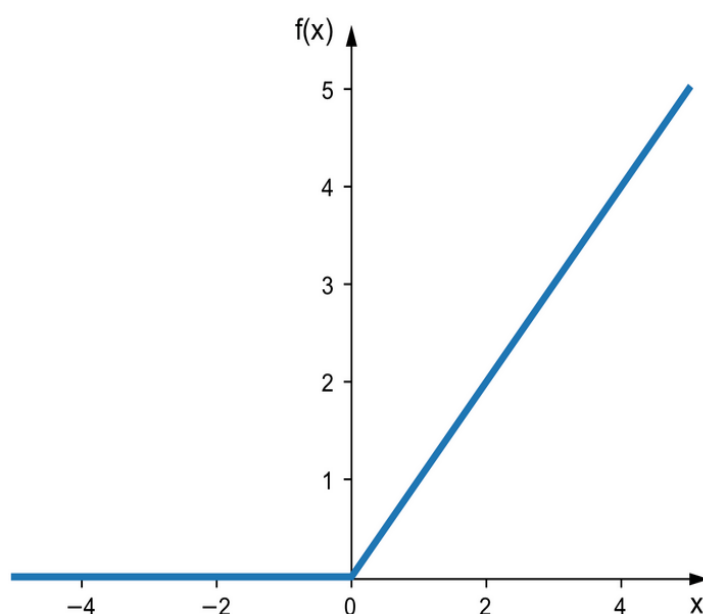


Figura 3. 2. – Funcția de activare ReLu

3.4.2. Softmax

Funcția de activare *Softmax* este folosită de cele mai multe ori pe straturile finale ale rețelelor ca o ultimă funcție de activare. Astfel, aceasta oferă o distribuție de probabilitate asupra claselor prezise. După aplicarea funcției *Softmax* unui vector, chiar dacă acesta are elemente negative sau nule, vectorul va conține elemente pozitive, în intervalul $(0,1)$, a căror sumă va fi 1.

În cazul în care se va folosi funcția *softmax* pentru ultimul strat, ea este urmată de o funcție *argmax* care va întoarce indicele clasei cu scorul cel mai mare. Astfel, obținându-se cel mai bun scor, se poate realiza o clasificare.

3.5. Deep-learning

Deși unii din algoritmi ce sunt astăzi folosiți cu precădere în metodele de *deep-learning* au fost inventați acum mult timp, utilizarea acestora a fost îngreunată considerabil. Întrucât obiectul central al modelelor de învățare automată este reprezentat de seturile de date de antrenare, evenimente precum creșterea semnificativă a volumului de informație pe care le poate procesa și stoca un computer și avansul tehnologiei prin dezvoltarea de procesoare grafice rapide, au avut un impact major în ceea ce înseamnă astăzi *deep-learningul*. Astfel, chiar dacă elemente centrale ale tehnicilor de *deep-learning* au fost teoretizate încă din 1980, progresul tehnologiei a fost cel care a rezolvat două din problemele substanțiale ale acestor algoritmi: nevoia de seturi mari de date și puterea de calcul.

Termenul de „*deep*” face referire la numărul de straturi ascunse pe care le are o rețea. Rețelele neurale tradiționale aveau inițial 2-3 straturi ascunse, pe când rețelele neurale adânci pot depăși 100.

Deep-learningul reprezintă o parte a *machine-learningului* care presupune o tehnică de a învăța computerul să acționeze într-o manieră umană, anume de a-l face să învețe după un anumit exemplu. *Deep-learningul* este o variație modernă ce presupune, în teorie, posibilitatea unui număr nemărginit de straturi ascunse. Totodată, deviații de la modelele clasice de învățare umană sunt posibile.

Odată cu popularizarea ideii de *deep-learning*, utilizarea rețelelor neurale convoluționale a luat amploare. În cazul unui astfel de model de învățare automată, straturile acestuia sunt organizate pe mai multe nivele ce au scop de a extrage și conserva caracteristicile cele mai importante din imaginea inițială. Totodată acestea au scopul de a își îmbunătăți performanțele.

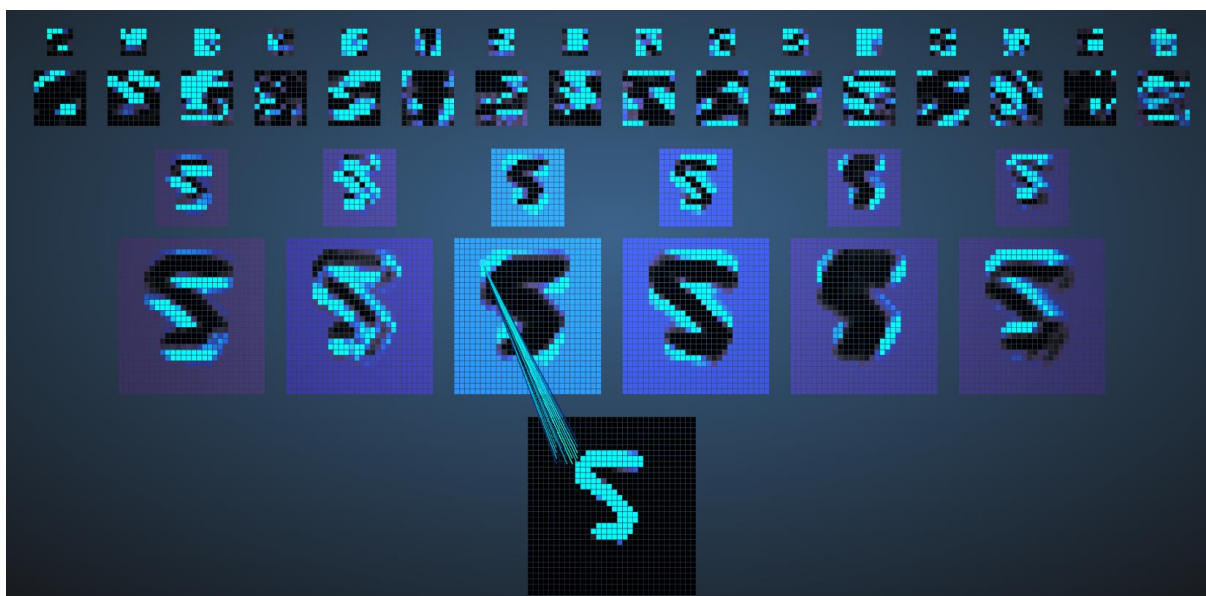


Figura 3. 3. – Modalitatea de extragere a caracteristicilor folosind o rețea neurală convoluțională ([24])

3.6. Rețele neurale convoluționale

Inspirați de modul în care oamenii percep imaginile, cercetătorii au încercat să transpună acest proces către computere. Rețelele neurale artificiale obțineau scoruri nesatisfăcătoare în prelucrarea imaginilor, acestea nefiind capabile să recunoască și să modeleze caracteristici definitorii ale unei clase. Chiar dacă rețelele neurale de tip *feed-forward* pot fi folosite pentru clasificarea imaginilor, având scoruri acceptabile, această abordare nu este deloc practică. Un număr foarte mare de neuroni ar fi necesari pentru procesarea imaginilor, deoarece, în cazul utilizării acestui tip de rețele, fiecare pixel reprezintă o caracteristică definitorie. Așadar, spre exemplu, pentru o imagine ce ar avea o dimensiune de 100 x 100 ar fi necesară ajustarea a 10,000 de *weights* pentru fiecare neuron din stratul al 2 lea. Dar ce s-ar fi întâmplat dacă imaginea era și coloră? Numărul parametrilor s-ar fi triplat. Operația de convoluție se remarcă a fi o soluție pentru această problema, întrucât aceasta reduce numărul de parametri, dând capabilitatea rețelei de a fi mai adâncă cu mai puțini parametri. Astfel, a apărut ideea de convoluție și unificare (*pooling*), concepte ce stau la baza rețelelor convoluționale.

Rețelele neurale convoluționale sunt folosite în preponderență în aplicații ce includ recunoașterea imaginilor și a videoclipurilor, clasificare de imagini, analize medicale și

procesarea limbajului natural. Datorită performanțelor acestora, ele au devenit rapid abordarea predominantă în rezolvarea problemelor propuse anterior.

3.6.1. Arhitectura rețelelor neurale convoluționale

Arhitectura acestora este compusă dintr-un strat de intrare, un strat de ieșire și unul sau mai multe straturi ascunse. Straturile ascunse conțin o serie de straturi convoluționale, straturi de unificare (*pooling*) și funcții de activare.

Spre deosebire de straturile ascunse ale rețelelor neurale artificiale, straturile ascunse ale rețelelor convoluționale sunt concepute pentru a extrage caracteristici importante ale pixelilor centrali, cât și a celor din jurul lor. Totodată acestea au rolul de a reduce dimensiunea imaginii.

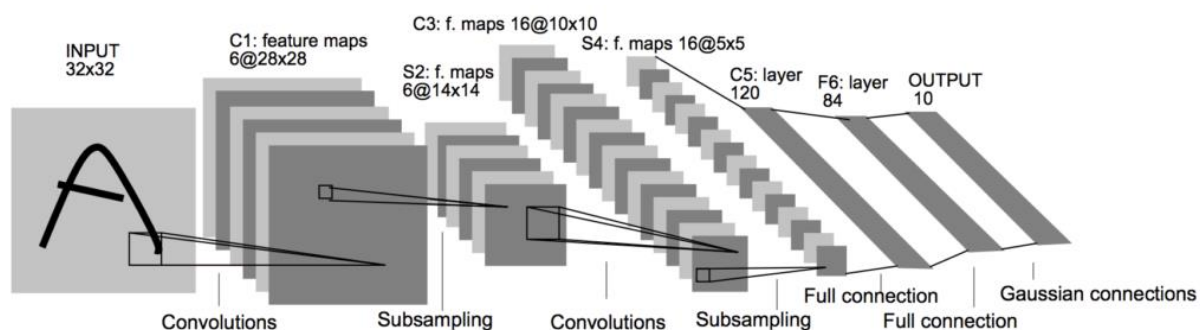


Figura 3. 4. – Exemplu de arhitectura a unei rețele convoluționale (LeNet5) [23]

În timp ce straturile ascunse ale unei rețele servesc ca modalități de extragere a trăsăturilor, straturile finale au scopul de a produce o clasificare a caracteristicilor extrase, iar apoi de a calcula scorul fiecărei clase. Întrucât există numeroase tipuri de reprezentări ale unei rețele ce folosește convoluții, imaginea 3.4 exemplifică una dintre primele și cele mai simple arhitecturi [23].

3.6.2. Straturile de convoluție

Unul dintre procedeele ce stau la baza rețelelor neurale convoluționale este procesul de convoluție, iar în spatele procesului de convoluție, stă ideea de conectivitate locală. Încercând să rezolve problema numărului costisitor de mare a numărului de parametri folosiți de o rețea neurală artificială, a luat naștere conceptul de conectivitate locală. Realizând că există o redundanță în corelarea unui neuron cu toți ceilalți neuroni din stratul anterior, cercetătorii au decis să reducă numărul acestor legături. Astfel, aceștia decid să conecteze fiecare neuron doar cu o mică porțiune de neuroni din stratul anterior. Această mică regiune poartă numele de câmp receptiv local. Apoi, suprapunând în mod constant aceste câmpuri receptive locale, se poate procesa un număr mai mare de informație globală. Acest proces se numește convoluție, iar câmpurile receptive locale ce străbat întreaga imagine se numesc filtre.

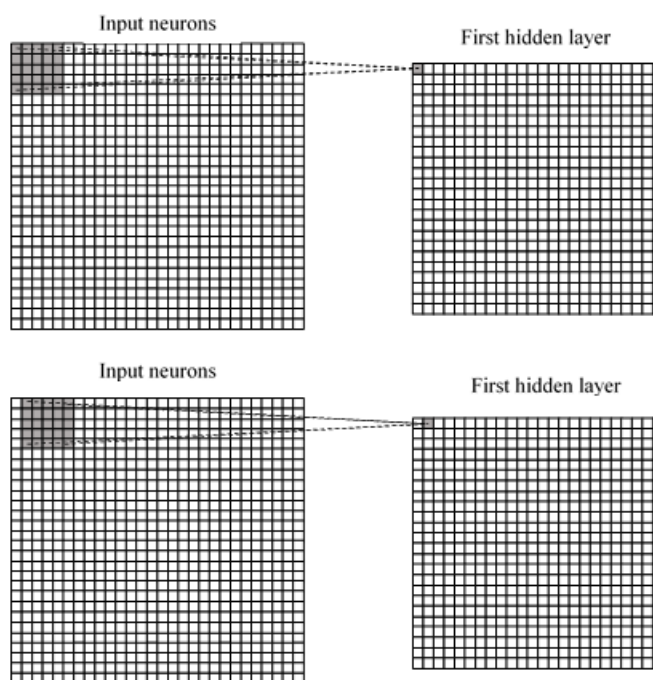


Figura 3. 5. – Operația de convoluție și modul în care fiecare câmp receptiv local este reprezentat de neuronii din stratul următor

De regulă, primul strat al unei rețele este dat de un strat de convoluție. Acesta are ca rol extragerea caracteristicilor predominante. Procesul de convoluție conservă relațiile dintre pixeli învățând *features* ale imaginii inițiale folosind diferite filtre. Presupunând că dimensiunea unui set de date de intrare are forma $(number\ of\ images) \times (image\ width) \times (image\ height)$, la ieșirea

din stratul de convoluție se va obține o imagine sau un set de imagini abstracte ce poartă numele de hartă de caracteristici (*feature map*) și va avea dimensiunile $(number\ of\ images) \times (feature\ map\ width) \times (feature\ map\ height)$, unde $feature\ map\ width < image\ width$ și $feature\ map\ height < image\ height$. Se obține, așadar o comprimare a trăsăturilor imaginii. În funcție de tipul filtrelor folosite, se pot extrage o multitudine de caracteristici precum: detectarea muchiilor, blurarea imaginii sau efectul de *sharpening* etc. Figura 3.6 reprezintă o descriere vizuală a acestor efecte.



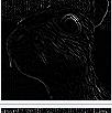
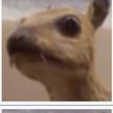
Operation	Kernel w	Image result $g(x,y)$	Operation	Kernel w	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		Gaussian blur 5×5 (approximation)	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$		Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$		Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$		Gaussian blur 3×3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figura 3. 6. – Reprezentarea vizuală a efectelor filtrelor asupra unei imagini

În cazul imaginilor foarte mari, o bună parte din informația caracteristică unei clase se va afla în anumite regiuni ale imaginii, iar celelalte regiuni vor fi inutile. Astfel apare ideea de a folosi *stride-ul*, ce reprezintă numărul de pixeli pe care îi va evita un filtru atunci când străbate imaginea. Prin urmare, un *stride* de mărime 1 va muta filtrul cu un singur pixel pe poziția următoare, pe când un *stride* de lungime 2, va sări peste câte un rând la fiecare mutare a filtrului.

Uneori, unele filtre nu se potrivesc perfect peste imaginea inițială, în special în cazul în care se folosește *stride-ul*. Astfel vom avea două opțiuni: prima opțiune presupune bordarea imaginii cu 0-uri (*zero-padding*) până filtrul se va potrivi. Cea de-a doua opțiune este de a renunța la părțile peste care filtrul nu poate ajunge. Aceasta se bazează pe ideea că de cele mai multe ori, părțile cele mai semnificative ale imaginii nu se vor afla niciodată la marginea acesteia.

3.6.3. Straturile de unificare (*pooling*)

Un alt concept important regăsit în arhitectura rețelelor convoluționale îl reprezintă *pooling-ul*. Imaginile din setul de antrenare ce aparțin aceleiași clase prezintă particularități aparte, astfel, caracteristicile definitorii ale clasei vor fi ușor modificate. Spre exemplu, într-o mulțime de poze cu leoparzi, unele animale vor fi mai depărtate, iar altele mai apropiate, unele vor fi într-o parte total opusă a imaginii față de celelalte. Prin urmare, rețelele vor avea nevoie de un anumit grad de flexibilitate pentru a putea face o predicție corectă și pentru a recunoaște aceste caracteristici. Astfel apare ideea de unificare ce este o formă non-liniară de compresie.

Există numeroși algoritmi de *pooling* dintre care se remarcă: *max-pooling*, *average pooling*, *sum-pooling*. Cel mai folosit este *max-pooling-ul*. Procedeu de max-pooling este descris în imaginea 3.7. și presupune extragerea valorilor maxime dintr-o anumită porțiune a imaginii, traversând întreaga imagine. Astfel, se conservă cele mai importante aspecte, pe când cele mai neînsemnate se pierd, lucru ce este benefic și ajută la reducerea numărului de parametri, prin urmare acesta poate fi considerat și un procedeu de control al *overfitting-ului*.

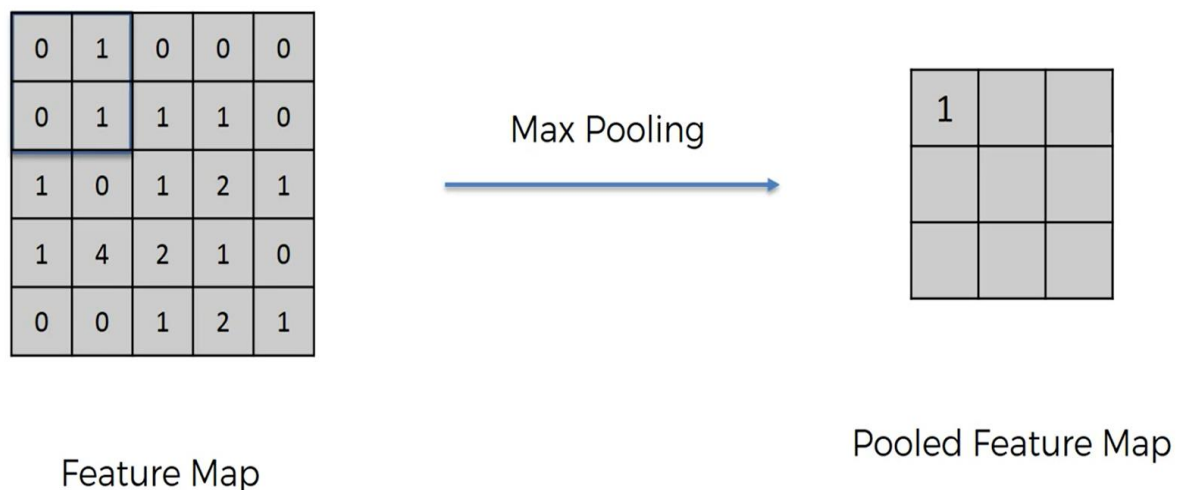


Figura 3. 7. – Procedeu de *Max-Pooling*

3.6.4. Operația de flattening și straturile fully-connected

După numeroase convoluții și straturi de unificare urmează procedeul de *flattening*. Acesta este un proces foarte simplu ce are scop pregătirea datelor pentru a putea parcurge straturile *fully-connected*. Prezentată în figura 3.5, operația de *flattening* înseamnă modificarea formei datelor primite pentru a fi potrivite ca *input* pentru straturile full-conectate.



Figura 3. 8. – Procedeul de *flattening*

Următorul pas este introducerea datelor într-un sistem *fully-connected*. Această parte a rețelei este similară rețelelor neurale artificiale. Ea este utilizată pentru a produce predicțiile rețelei, reprezentând partea de final a acesteia. Există numeroase rețele neurale convoluționale ce folosesc multiple *layere fully-connected*.

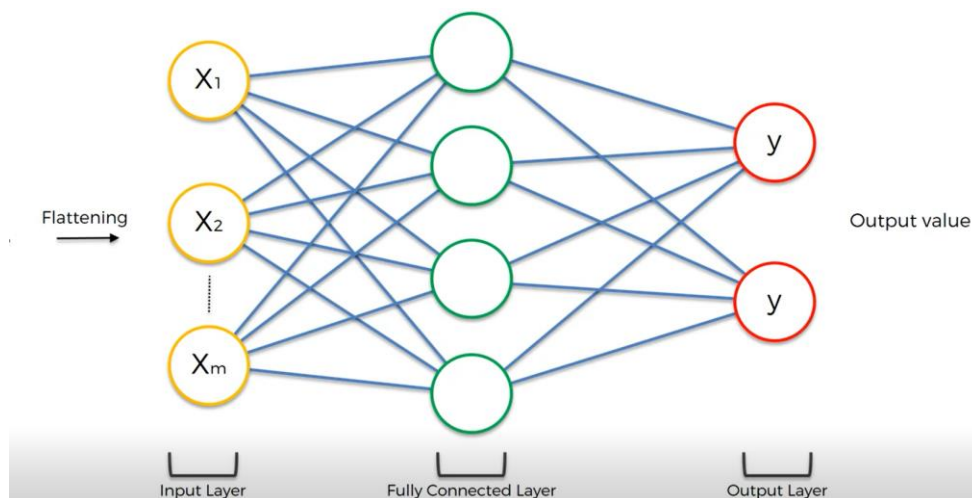


Figura 3. 9. – Staturile *fully-connected*

4. Prezentarea modului de abordare

4.1. Setul de date

Pentru abordarea subiectului ales, am decis să folosesc setul de date MNIST, deoarece imaginile au o dimensiune mică, de 28x28 de pixeli, fiind astfel ideale în generarea exemplelor adversariale. Setul de date MNIST reprezintă o bază de date de 70,000 de exemple cu cifre scrise de mână, dintre care 60,000 sunt destinate antrenării modelelor, iar restul de 10,000 sunt utilizate pentru testare.

Baza de date MNIST este utilizată adesea în antrenarea sistemelor de procesare a imaginilor și este totodată foarte populară în rândul practicanților de *machine-learning*. Aceasta a fost creată prin schimbarea dimensiunii imaginilor din setul de date NIST. Setul de date de antrenare NIST a fost preluat de la Biroul de Recensământ al Statelor Unite, pe când setul de testare a fost obținut de la elevii de liceu. Yan LeCun, Corinna Cortes și Christopher J.C. Burges, au preluat aceste date, le au normalizat și au introdus niveluri de gri. Astfel, fiecare pixel are valori între 0 (alb) și 255 (negru). Pentru că setul de date de antrenare era diferit față de cel de testare, cel din urmă fiind mult mai greu recunoscut, ele provenind de la studenți, autorii au construit setul de antrenare pentru MNIST amestecând cele doua seturi de date.



Figura 4. 1. – Exemplu de imagini din setul de date MNIST

Baza de date MNIST conține 10 clase cu cifre de la 0 la 9. Distribuția claselor se poate observa în imaginea 4.2. Setul de date poate fi descărcat de la [25].

Digit	MNIST	
	Train	Test
0	5,923	980
1	6,742	1,135
2	5,958	1,032
3	6,131	1,010
4	5,842	982
5	5,421	892
6	5,918	958
7	6,265	1,028
8	5,851	974
9	5,949	1,009
Total	60,000	10,000

Figura 4. 2. – Distribuția claselor din seturile de antrenare și testare

4.2. Arhitecturi propuse

În realizarea acestui proiect am folosit 3 tipuri de rețele convoluționale care au fost antrenate de la 0 pe același set de date. Prima dintre ele este o rețea convoluțională ce are doar două straturi, la care mă voi referi în continuare cu termenul de ConvNet. Cea de-a doua rețea folosită se numește LeNet-5, ce a fost concepută de către Yann LeCun, Leon Bottou, Yosuha Bengio și Patrick Haffner pentru a recunoaște caracterele scrise de mână și cele scrise de tipar [23]. Ultima și cea de-a treia rețea este diferită prin arhitectura ei de primele doua. Pe când primele doua au o arhitectură mai simplă, ultima rețea va fi mult mai adâncă și va avea 18 straturi. Aceasta se numește ResNet și a câștigat primul loc în 2015 la ILSVRC (ImageNet)

într-un *task* de clasificare, locul 1 la aceeași competiție, supusă de data aceasta unei sarcini de localizare și tot locul 1 la COCO 2015 pentru detectare și segmentare de imagini.

Motivația din spatele alegerii acestor arhitecturi a fost a ceea de a studia modul în care topologia unei rețele poate influența generarea de exemple adversariale și acela de a observa diferențe din punct de vedere al robusteții acestora.

După antrenarea celor 3 modele, am construit un ansamblu din ele. Acesta obține predicțiile fiecărei rețele, iar apoi alege predicția corectă din voturile date, fiecare model având o pondere egală.

4.2.1. ConvNet

Pentru antrenarea acestui model am folosit o întregul set de antrenare MNIST grupat în bucăți de 64 de imagini. După ce am încercat să îmbunătățesc acuratețea modelului folosind diverse rate de învățare, am ales valoarea 0.01 deoarece aceasta era cea care oferea cele mai bune rezultate. Antrenarea a fost efectuată de-a lungul a 10 epoci.

Arhitectura acestui model consta în două straturi convoluționale, primul având 20 de filtre, iar al 2 lea 50 de filtre de 5x5 cu un *stride* de 1 și *zero-padding*. În continuarea fiecărui strat de convoluție am adăugat o funcție de activare ReLu, iar apoi un strat de unificare de dimensiune 2x2, ce folosește *max-pooling-ul*. La sfârșitul rețelei, am folosit două straturi *fully-connected*, unul de 500 de neuroni urmat de ReLu și un altul asupra căruia este folosit *Softmax* pentru a obține o distribuție de probabilități a fiecărei clase.

În urma antrenării, această rețea a obținut o acuratețe pe datele de testare de 98.71%.

4.2.2. LeNet-5

Conform lucrării originale [23], arhitectura rețelei LeNet-5 a fost concepută pentru recunoașterea caracterelor de tipar din documentele oficiale și a cifrelor scrise de mână. Luând în considerare aceste lucruri am decis să o includ în lucrarea mea.

Intrarea pentru primul *layer* al rețelei este o imagine alb-negru de 32x32 de pixeli ce va trece printr-un strat de convoluție cu 6 filtre ce au dimensiunea de 5x5 și folosesc un *stride* de 1. Astfel, dimensiunea output-ului va fi de 28x28x6. Ca funcție de activare se folosește *tanh*. Următorul strat este un strat de *average-pooling* cu un filtru de 2x2 și *stride* de 2, reducând astfel dimensiunea datelor de intrare la 14x14x6.

Cel de-al treilea strat este un strat convoluțional ce are 16 filtre de dimensiune 5x5 și un *stride* de 1. În acest strat, fiecare unitate din fiecare filtru de convoluție este conectată la doar câteva din filtrele vecine din stratul anterior. Figura 4.3, preluată din [23] explică modul în care acestea sunt combinate. Autorii susțin că acest lucru ajută la ruperea simetriei rețelei și limitează numărul de conexiuni la un număr rezonabil.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

Figura 4. 3. – Fiecare coloană arată ce *feature maps* din stratul 2 sunt combinate cu unitățile din stratul 3

Stratul patru reprezintă din nou o operație de unificare cu un filtru de 2x2 și un *stride* de 2. Acesta este identic cu stratul 2, cu excepția faptului că, primind 16 *feature-maps*, va avea ca ieșire dimensiunea de 5x5x16.

Stratul cu numărul cinci va fi un strat convoluțional *fully-connected* cu 120 de filtre, fiecare având dimensiunea de 1x1, astfel realizându-se operația de *flattening*. Stratul următor este un strat full-conectat ce conține 84 de neuroni. La sfârșitul rețelei se află un strat full-conectat de dimensiune 10 ce folosește funcția *softmax* pentru a prezice probabilitățile fiecărei clase. Se folosește ca funcție de activare după fiecare strat de convoluție *tanh*.

Pentru a antrena această rețea, am folosit o mărime a *batch*-ului de 32 de imagini, iar acesta a fost antrenat pe 15 epoci. În urma antrenării, aceasta a obținut pe setul de date de testare o acuratețe de 99.05%.

4.2.3. ResNet18

După câștigarea concursului LSVRC2012 de către AlexNet [26] cu o arhitectură *deep* ce conținea doar 5 straturi, cercetătorii și au îndreptat mai mult atenția către mărirea numărului de straturi pe care le are o rețea. Totuși, aceștia s-au lovit de o problemă, cea a gradientilor care dispăreau. Mărind numărul de straturi, în momentul utilizării algoritmului de *back-propagation*, aceștia au observat ca straturile de la începutul rețelei primeau gradienti cu valori foarte mici, apropiate de 0. Astfel, cu cât rețelele erau mai adânci, performanțele acestora se degradau.

În 2015, ResNet152 câștigă competiția ImageNet, fiind o arhitectură cu 152 de straturi[27]. Competitorii reușesc această victorie datorită folosirii ideii de învățare reziduală (*Residual Learning*) care rezolvă problema dizolvării gradientilor. Această metodă presupune folosirea de blocuri reziduale și *identity mapping*. Aceasta presupune ca, după câteva straturi de convoluție, valorii care iese din straturile de convoluție sa îi fie adăugată valoarea de dinaintea straturilor. Presupunând că x este valoarea ce intră într-un bloc rezidual, iar $F(x)$ este valoarea ce iese din el, la ieșirea din blocul de convoluție, valorii $F(x)$ i se va adăuga x . Totuși, datorită faptului că $F(x)$ a trecut prin blocuri de convoluție, acesta va avea dimensiunea mai mică față de x . Prin urmare, va mai fi adăugat un strat de convoluție de dimensiune 1×1 pentru a-i micșora dimensiunea lui x . Figura 4.4. prezintă ideea de *residual learning*.

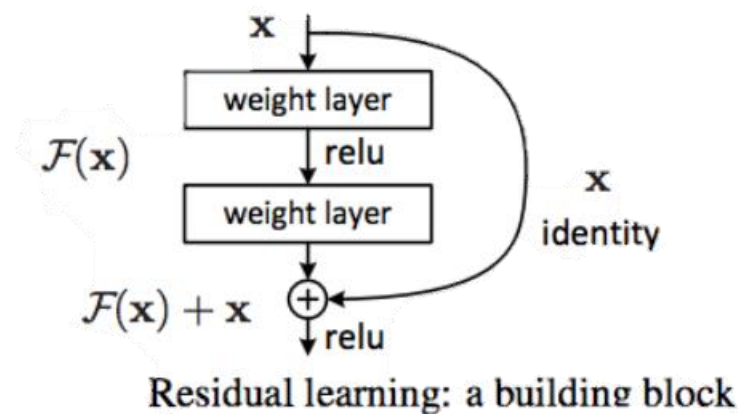


Figura 4. 4. – Reprezentarea unui bloc rezidual ce folosește *identity mapping* [27]

Întrucât arhitectura originală ResNet a fost creată pentru a procesa imagini de înaltă rezoluție de 224x224 de pixeli ce sunt RGB, a trebuit să schimb stratul de intrare pentru a primi un singur canal. Mai mult de atât, rețeaua fiind antrenată pe setul de date ImageNet, aceasta oferea 1000 de predicții, corespunzătoare fiecărei clase prezente în acel set de date. Așadar am modificat ultimul strat *fully-connected*, micșorându-l la doar 10 clase, făcându-l astfel, potrivit pentru tratarea problemei alese.

Rețeaua folosită în acest proiect, ResNet18 face parte dintr-o familie de rețele ce folosesc învățarea reziduală, de unde provine și numele acesteia. Există alte modele și cu 50, 101 sau 152 de straturi. În antrenarea acestei rețele am folosit un total de 50 de epoci, utilizând o rată de învățare de 0.01 și o dimensiune a *batch-ului* de 64, obținând astfel o acuratețe de 99.5%.

4.3. FGSM

Prezentat de către Ian J. Goodfellow în [5], FGSM (*Fast Gradient Sign Attack*) este unul dintre cei mai populari algoritmi pentru generarea exemplelor adversariale. Atacul este unul puternic și intuitiv. Acesta este un atac de tipul *white-box*, prin urmare folosește toate informațiile modelului de care are nevoie. După cum îi spune și numele, acesta utilizează modul în care rețelele învață, anume prin gradienti.

Ideea algoritmului este ca, în loc de a minimiza funcția de pierdere pentru o imagine dată, încercând să ajustăm *weights-urile* modelului în funcție de gradientii obținuți prin *backpropagation*, acesta încearcă să mărească valoarea funcției de pierdere bazându-se pe aceiași gradienti. Cu alte cuvinte, acest tip de atac folosește gradientul funcției de pierdere referitor la imaginea de intrare, apoi ajustează imaginea de intrare pentru a maximiza funcția de pierdere.

Atacul poate fi scris sub forma:

$$adv_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y)), \text{ unde:}$$

- adv_x este imaginea adversarială
- x este imaginea inițială
- θ reprezintă parametrii modelului

- y este eticheta inițială a lui x
- ϵ reprezintă gradul de perturbație al imaginii
- $J(\theta, x, y)$ este *loss-ul* modelului în raport cu imaginea inițială

Astfel, observând cât de mult influențează fiecare pixel valoarea funcției de pierdere, se poate construi o imagine adversarială.

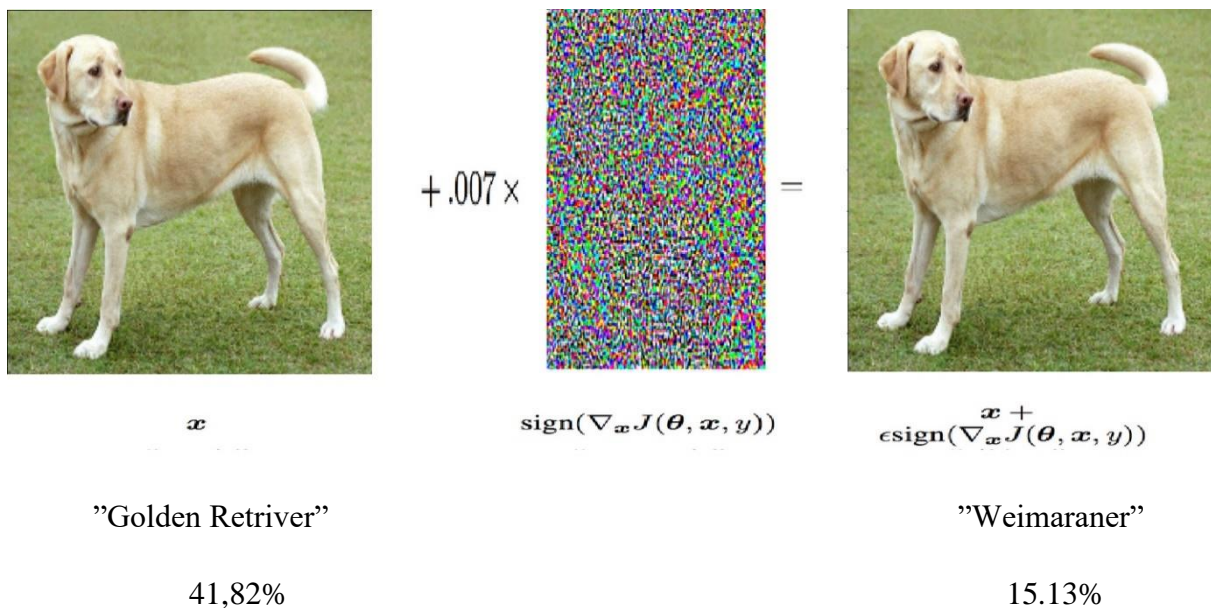


Figura 4. 5. – Exemplu adversarial în care clasa imaginii s-a schimbat prin modificarea valorilor unor pixeli

4.4. SimBa

În 2019, Chuan Guo et al. [28] tratează problema exemplelor adversariale în condiții *black-box*, stabilind și o nouă referință pentru viitoarele atacuri de acest tip din punctul de vedere al vitezei algoritmului cât și al numărului de interogări asupra modelului țintă. O altă particularitate a algoritmului este dată de faptul că acesta poate fi scris în aproximativ 20 de linii de cod.

După cum exemplele adversariale există aproape mereu, atacul asupra unui clasificator se va transforma într-o problemă de căutare în jurul imaginii folosite. În condiții *white-box*,

această căutare poate fi eficientizată folosind algoritmul *gradient descent* [4]. Totuși, un atac de tipul *black-box* este mult mai apropiat de lumea reală. În astfel de scenarii, numărul de interogări la care este supus un model poate implica deseori atât un cost financiar, cât și un timp de așteptare mai mare (pentru API-ul oferit de Google, prețul a 1000 de interogări este de 1.5\$). Prin urmare, am ales să folosesc acest algoritm datorită vitezei lui și numărului mic de interogări asupra modelului țintă.

Presupunând că distanța până la o graniță de decizie este mică, algoritmul *SimBa* se folosește de scorurile claselor obținute pentru a face o următoare decizie. În mod concret, alegând în mod repetat o direcție întâmplătoare dintr-un set predefinit de direcții ortogonale de căutare și utilizând scorurile claselor ca mod de ghidare (pentru a verifica dacă modificarea adusă ne apropie de granița de clasificare sau ne îndepărtează), algoritmul modifică imaginea adăugând sau scăzând vectorul direcție din imagine. Astfel, fiecare modificare aduce imaginea mai aproape de granița de decizie și mai departe de eticheta inițială.

Algorithm 1 SimBA in Pseudocode

```

1: procedure SIMBA( $x, y, Q, \epsilon$ )
2:    $\delta = 0$ 
3:    $p = p_h(y \mid x)$ 
4:   while  $p_y = \max_{y'} p_{y'}$  do
5:     Pick randomly without replacement:  $q \in Q$ 
6:     for  $\alpha \in \{\epsilon, -\epsilon\}$  do
7:        $p' = p_h(y \mid x + \delta + \alpha q)$ 
8:       if  $p'_y < p_y$  then
9:          $\delta = \delta + \alpha q$ 
10:       $p = p'$ 
11:    break
  return  $\delta$ 

```

Figura 4. 6. – Pseudocodul algoritmului *SimBa* [28]

Intuiția ce a ghidat la inventarea algoritmului a fost aceea că, pentru oricare direcție q și pentru un grad de perturbare ales, ϵ , una dintre $x + \epsilon q$ sau $x - \epsilon q$ este posibilă să reducă $p_h(y|x)$, adică probabilitatea modelului de a clasifica imaginea x ca aparținând clasei y . Prin urmare, vom selecta în mod aleatoriu direcții q pe care le vom adăuga sau scădea.

Cercetătorii au arătat că putem minimiza numărul de interogări dacă mai întâi vom încerca să adăugăm direcția q , iar apoi să o scădem. Pentru a garanta o eficiență maximă din

punctul de vedere al numărului de interogări, trebuie să ne asigurăm că oricare două direcții alese nu se anulează între ele, prin urmare vom selecta în mod unic direcțiile și nu le vom repeta.

În încheierea lucrării, autorii menționează faptul că au evitat să folosească tehnici mai sofisticate tocmai pentru a păstra simplitatea algoritmului și că i se pot aduce diverse modificări. Astfel, pentru a reduce numărul de interogări, i-am adăugat o condiție de sfârșit atunci când exemplul a trecut granița de clasificare și am setat o limită superioară pentru numărul de interogări pe care le poate face algoritmul.

4.5. Ansamblu de rețele

Construcția unui ansamblu de rețele este o paradigmă a învățării rețelelor neurale adânci (*Deep-learning*) ce presupune utilizarea mai multor rețele pentru a forma un comitet ce va lucra împreună în rezolvarea unei sarcini prin balansarea acurateții fiecărui model. Scopul folosirii unei astfel de arhitecturi este aceea de a îmbunătăți performanțele de predicție.

Diversitatea modelelor folosite este utilă atunci când erorile de predicție ale ansamblului sunt uniform distribuite. Astfel, o diversitate mai mare va duce la o îmbunătățire a acurateții ansamblului.

O altă posibilă proprietate a ansamblelor, în contextul exemplelor adversariale, ar putea fi aceea ca ele sunt rezistente atacurilor sau că ele ar putea avea o performanță crescută în condiții adversariale. Ideea ce stă la baza acestei proprietăți este aceea că un exemplu adversarial ar putea modifica predicția unei rețele, însă este posibil să nu reușească același lucru cu un număr mai mare de rețele.

Predicția unui astfel de ansamblu se formează combinând predicțiile fiecărei rețele printr-o metodă comună. O astfel de metodă poate fi una simplă, precum votul majorității, suma predicțiilor obținute sau media acestora. Alte metode mai complexe presupun adăugarea unei ponderi a votului fiecărei rețele în funcție de performanțele lor.

Plecând de la această ipoteză, în scopul studierii comportamentului ansamblelor în condiții adversariale, am construit un ansamblu format din cele trei rețele. Astfel, deoarece fiecare rețea produce un vector de 10 elemente cu valori între 0 și 1, am decis să folosesc suma predicțiilor ca metodă de calcul a predicției finale a modelului.

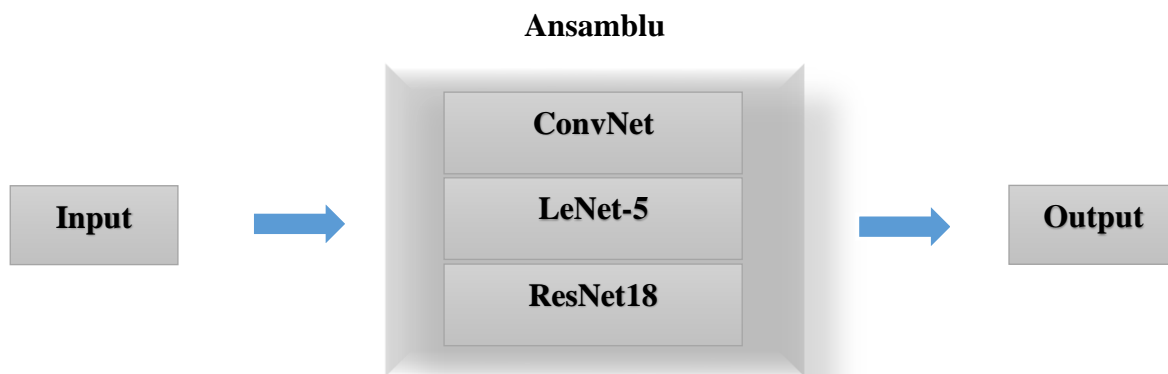


Figura 4. 7. – Ansamblul construit

5. Rezultate obținute

În acest capitol voi detalia experimentele realizate. Totodată voi prezenta rezultatele obținute.

5.1. SimBa vs. LeNet-5

După ce am antrenat toate rețelele și am ajuns la o acuratețe ridicată, am implementat algoritmul *SimBa* și am inițiat un atac fără țintă în condiții *black-box* împotriva celei de-a doua rețele convoluționale cu scopul studierii degradării performanței în funcție de gradul de perturbare. Pentru aceasta, am ales în mod întâmplător 100 de exemple care sunt clasificate corect (pentru ca o imagine clasificată în mod incorect să nu fie percepută ca un exemplu adversarial) și am selectat 5 grade diferite de perturbare.

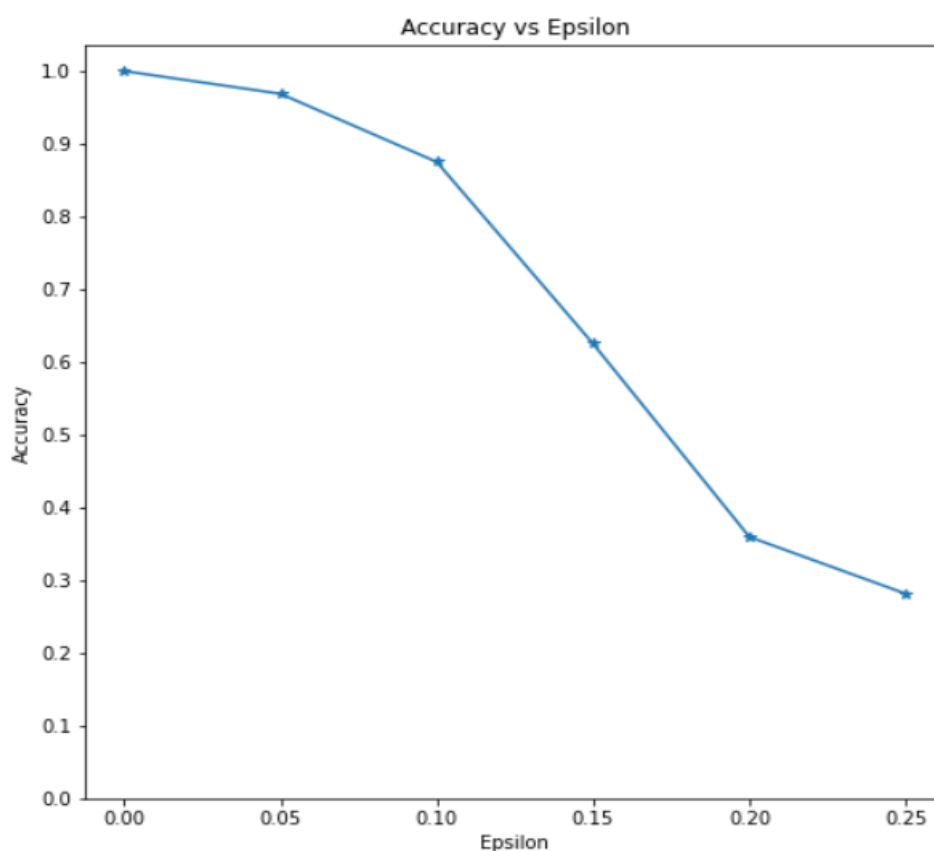


Figura 5. 1. – Acuratețea rețelei în funcție de gradul de perturbare ales

Pentru fiecare din cele 5 grade de perturbare, am generat exemple adversariale plecând de la imaginile selectate și am testat acuratețea modelului. În cazul de față, deoarece imaginile au doar 28x28 pixeli și au un singur canal de culoare (*gray-scale*), valoarea lui *epsilon* este direct proporțională, sau chiar egală cu procentul modificării pixelilor selectați, de la negru către alb.

Imaginea 5.1 arată cum, pentru o perturbare de doar 0.15, acuratețea modelului ajunge la 62%, clasificând corect 65 din cele 100 de imagini, ceea ce face rețeaua să se comporte mai slab decât o mare parte din modelele de *machine-learning* pe acest set de date. În schimb, pentru o perturbare de 0.2, acuratețea modelului este degradată până la 35%, reușind să clasifice în mod corect doar 35 de imagini.

Cu toate acestea, alegerea unui *epsilon* mic nu garantează modificarea imaginii astfel încât aceasta să treacă granița de clasificare și, deși există exemple adversariale ce pot fi generate folosind o valoare mică, acestea sunt foarte rare. Pe de altă parte, selectarea unui *epsilon* mare va ajuta algoritmul să găsească o astfel de imagine adversarială într-un timp scurt, dar va face ca aceasta să fie cu mult diferită de imaginea originală. Figura 5.2 va cuprinde o serie de imagini generate pentru diferite valori ale lui, care pot explica modul în care valoarea lui *epsilon* influențează calitatea imaginii.

În cazul în care am selectat $\epsilon = 0.5$, din 100 de imagini, doar 5 au putut modifica predicția rețelei. Totuși, modificările aduse imaginilor nu sunt vizibile cu ochiul liber. Pe de altă parte, atunci când valoarea lui *eps* este una ridicată, modificările imaginilor se pot observa cu ușurință.

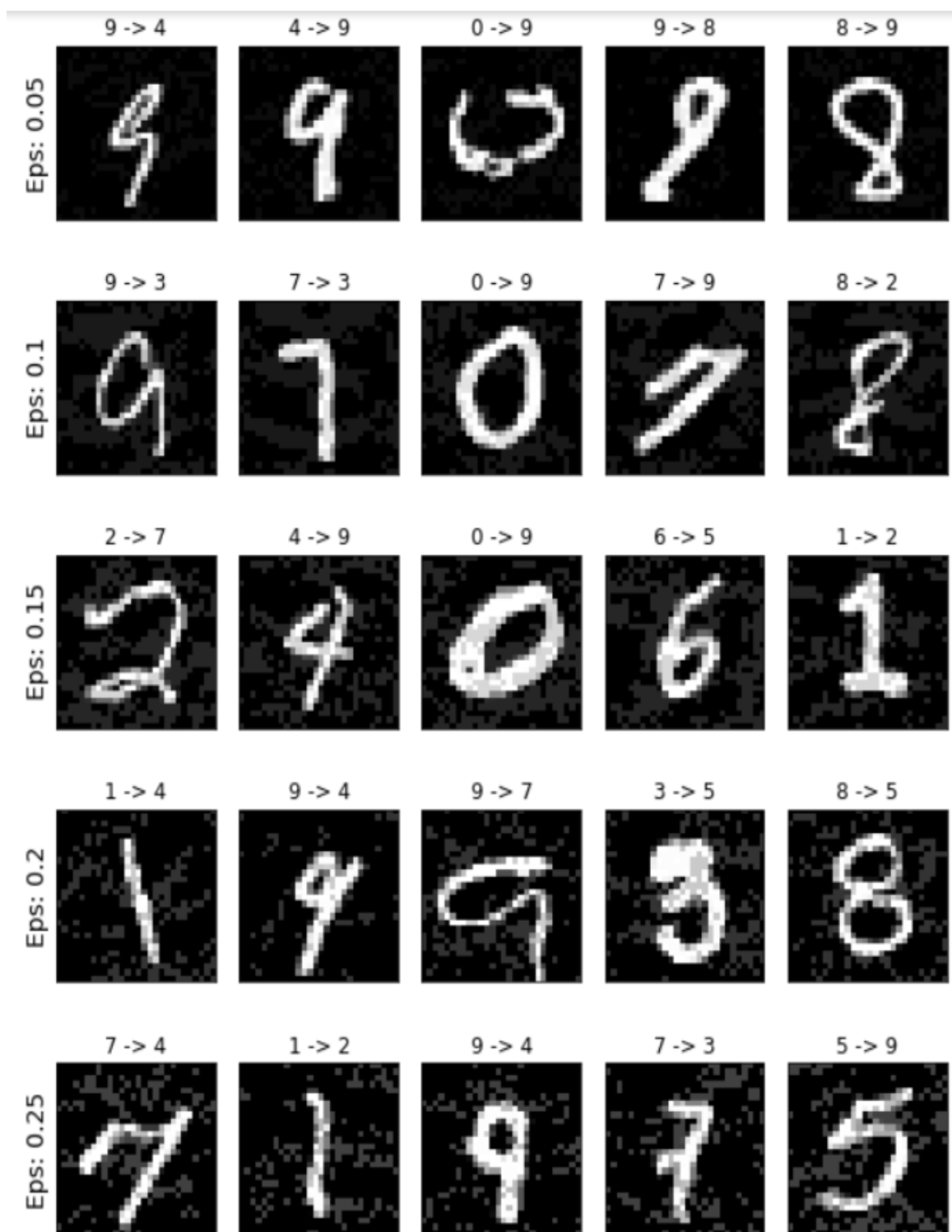


Figura 5. 2. – Exemple adversariale generate pentru diferite valori ale lui *epsilon*. Predicția inițială a modelului este prima valoare, cea de-a doua fiind predicția pentru exemplul dat.

Imaginea 5.2. prezintă compromisul dintre alegerea unei valori mici pentru epsilon ce vine odată cu alterarea imaginii inițiale într-o manieră insesizabilă unui adnotator uman, dar cu

timpi de calcul mari și alegerea unei valori ridicate pentru *epsilon* ce oferă timpi scăzuți, însă modificările sunt vizibile.

5.2. SimBa vs. Ansamblu

Pentru testarea performanțelor ansamblului, am selectat 100 de imagini în mod egal din fiecare clasă, din setul de date de testare și am comparat numărul mediu de interogări ale fiecărei rețele cu numărul mediu de interogări ale ansamblului. Am setat un număr maxim de interogări de 12,000, deoarece, în cazul în care nu alegeam un număr maxim, găsirea unui exemplu adversarial ar fi durat foarte mult, imaginea inițială ar fi putut fi distrusă, sau procesul ar fi putut intra într-o buclă infinită în cazul în care nu exista un asemenea exemplu.

În mod surprinzător, cea mai simplă rețea a obținut cel mai ridicat număr de interogări, aceasta atingând numărul mediu de 7034 de interogări. Cea de-a doua rețea, LeNet-5 a obținut un număr mediu de 975 de interogări, iar ResNet18 a obținut un număr de 893 în medie pentru toate cele 100 de imagini. Ansamblul a avut nevoie de un număr mediu de 1601 interogări.

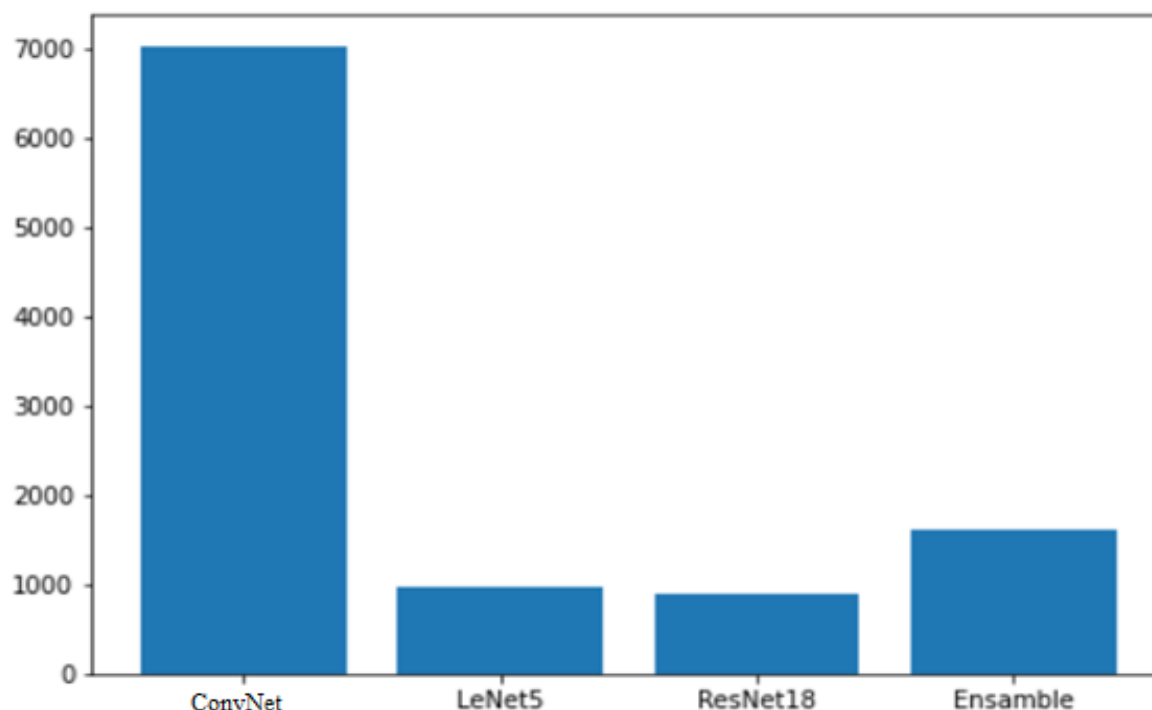


Figura 5. 3. – Numărul mediu de interogări pentru fiecare arhitectură folosită

O primă explicație pentru numărul crescut de interogări pentru ConvNet ar fi aceea că rețelele cu mai puține straturi sunt mai robuste atacurilor adversariale, chiar dacă au o performanță mai scăzută. În următorul subcapitol voi testa această ipoteză. O altă explicație ar putea fi aceea că, pentru această arhitectură, algoritmul ales a găsit mai greu exemple adversariale.

Totodată se remarcă faptul că ansamblul are nevoie de un număr mai mare de interogări decât arhitecturile LeNet-5 și ResNet18. Pentru a putea face ca ansamblul să clasifice o imagine în mod eronat, cel puțin două dintre rețelele componente vor trebui să clasifice greșit acea imagine.

Figurile 5.4., 5.5. și 5.6 ilustrează acuratețea fiecărui model pentru cele 100 de imagini. Astfel putem observa rețelele care au fost induse în eroare cel mai frecvent.

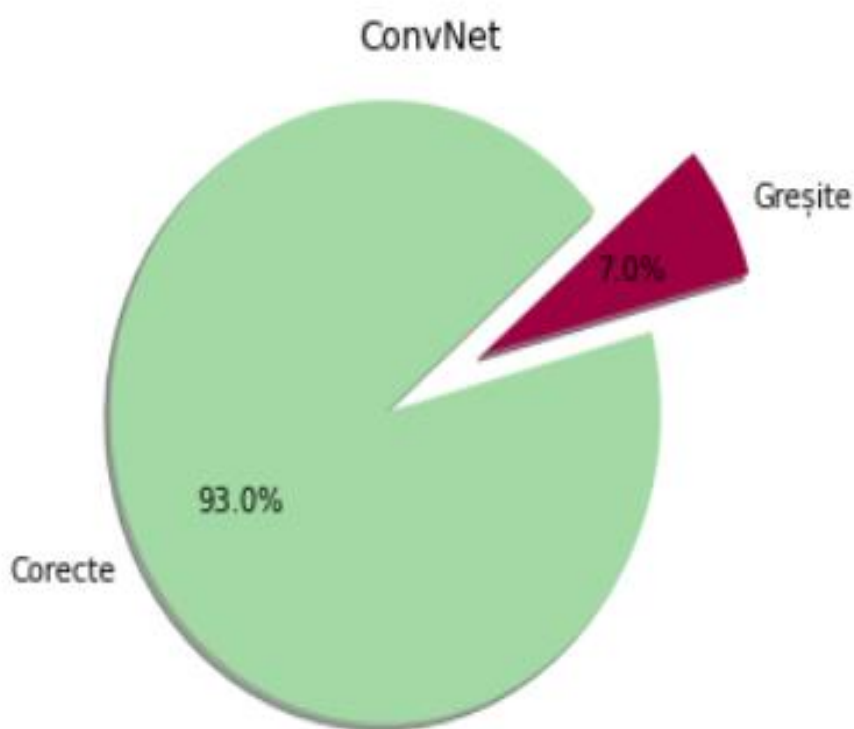


Figura 5. 4. – Acuratețea rețelei ConvNet evaluată pe exemplele adversariale

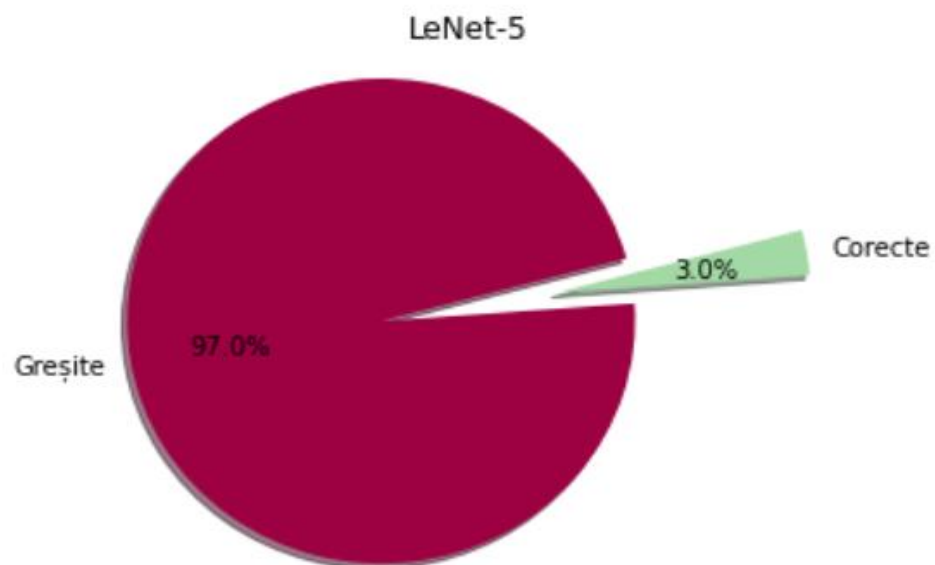


Figura 5. 5. – Acuratețea rețelei LeNet5 evaluată pe exemplele adversariale

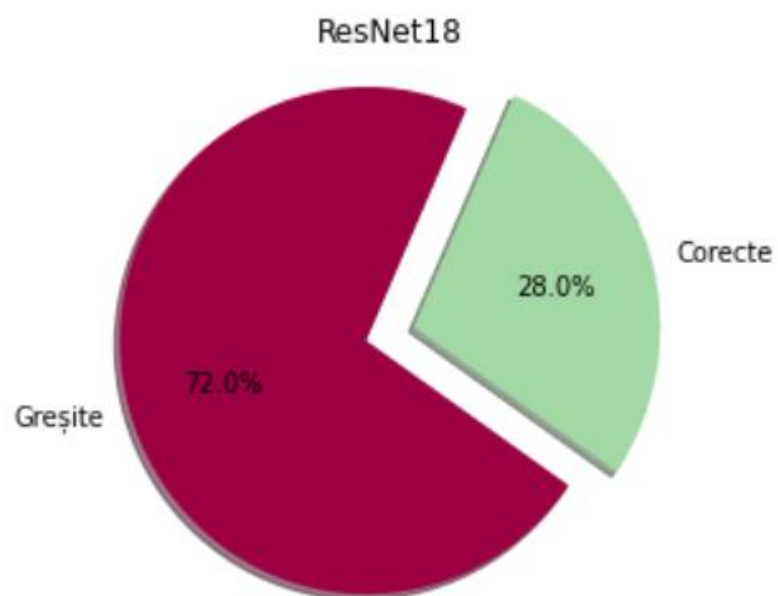


Figura 5. 6. – Acuratețea rețelei ResNet18 evaluată pe exemplele adversariale

Figurile 5.7. și 5.8. prezintă pe linii aceeași imagine trecută prin algoritmul de generare al exemplelor adversariale. Fiecare coloană reprezintă, în ordine, un model: Ansamblul, ConvNet, LeNet-5, ResNet18. Se poate observa cum, de regulă, imaginile generate pentru ansamblu sunt cele mai distorsionate.

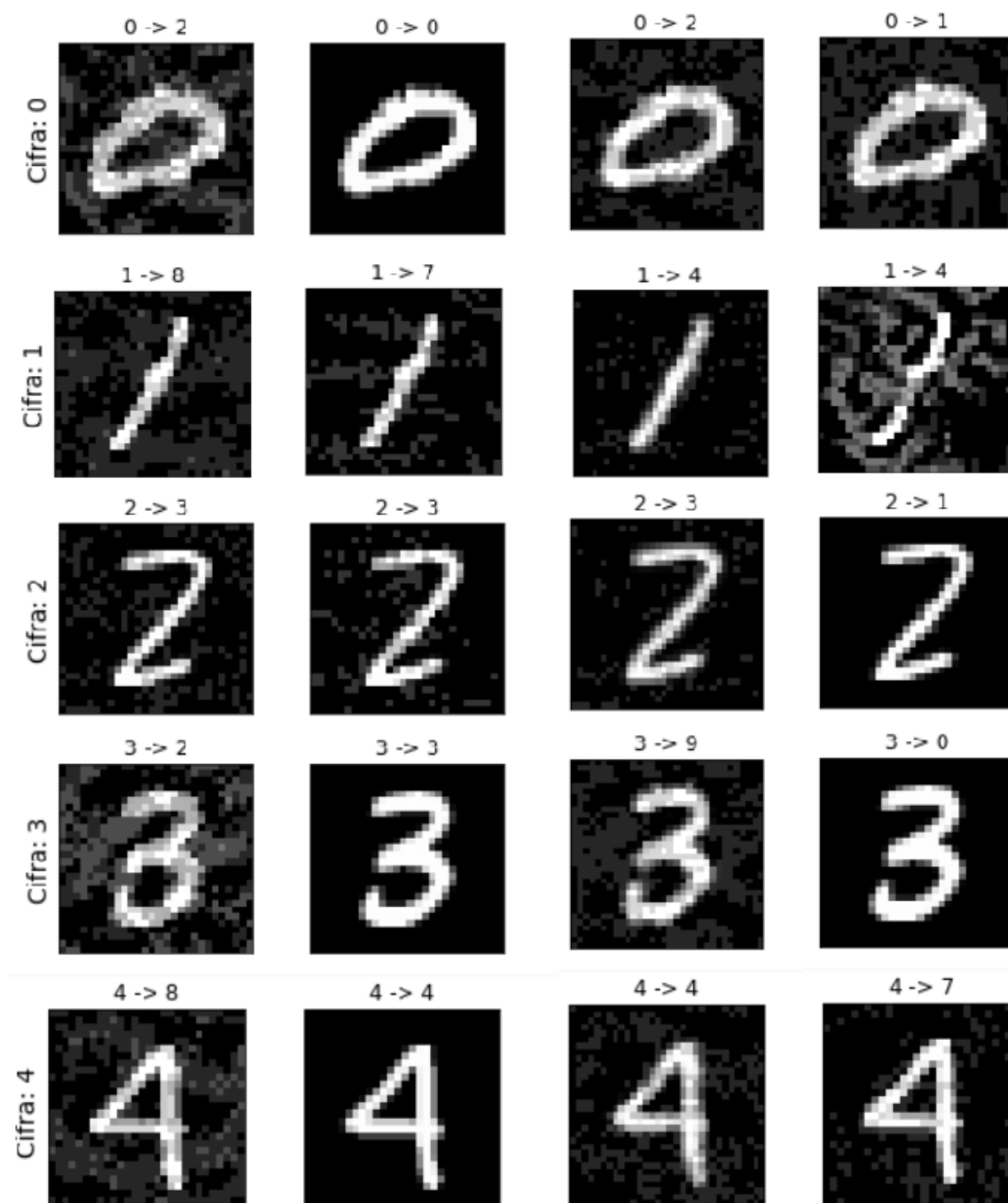


Figura 5. 7. – Exemple adversariale ale aceleași imagini. Pe prima coloană, în ordine, se află modelele: Ansamblu, ConvNet, LeNet-5, ResNet18

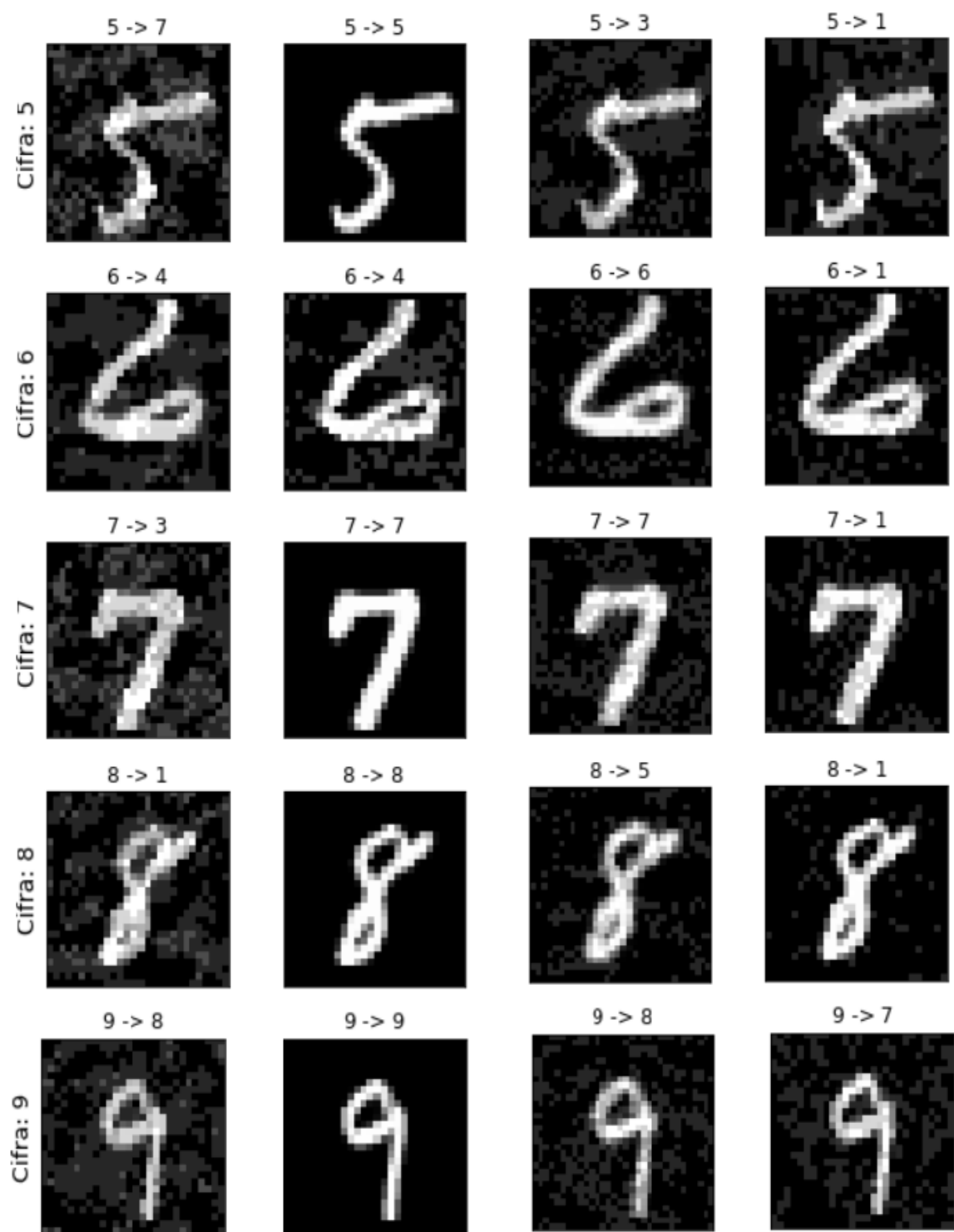


Figura 5. 8. – Exemple adversariale ale aceleași imagini. Pe prima coloană, în ordine, se află modelele: Ansamblu, ConvNet, LeNet-5, ResNet18

În cazul în care, pentru o imagine, unul dintre modele depășea pragul de 12,000 de interogări, întrucât conținutul acesteia nu mai era lizibil, imaginea obținută nu se considera adversarială ci era marcată ca fiind distrusă. Totuși, se păstra imaginea originală. Acest lucru se poate observa pe coloana a 2-a a figurilor 5.7. și 5.8.

5.3 FGSM vs. ConvNet

Deoarece cea mai simplă arhitectură, ConvNet, s-a comportat cel mai bine împotriva atacurilor adversariale, am decis să verific robustețea acesteia folosind o altă metodă, de această dată fiind un atac de tipul *white-box*. Astfel, dacă rețeaua este robustă, ar obține o acuratețe bună și împotriva acestui atac. Pe de altă parte, dacă atacul reușește, am putea considera că metoda aplicată anterior nu a putut găsi exemple adversariale cu ușurință pentru parametrii aleși, însă exemplele adversariale există.

Pentru această metodă, am folosit aceleași imagini utilizate anterior.

Pentru un *epsilon* de 0.1, acuratețea modelului s-a degradat la 82%, iar pentru o valoare de 0.25 a ajuns la 26%. Graficul de mai jos descrie acest fenomen.

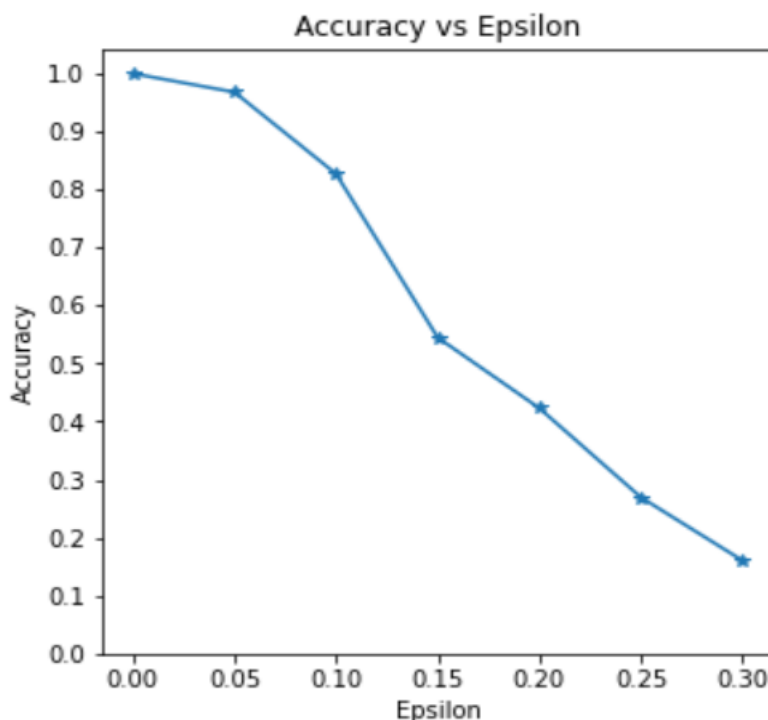


Figura 5. 9. – Acuratețea rețelei ConvNet în condiții *white-box*

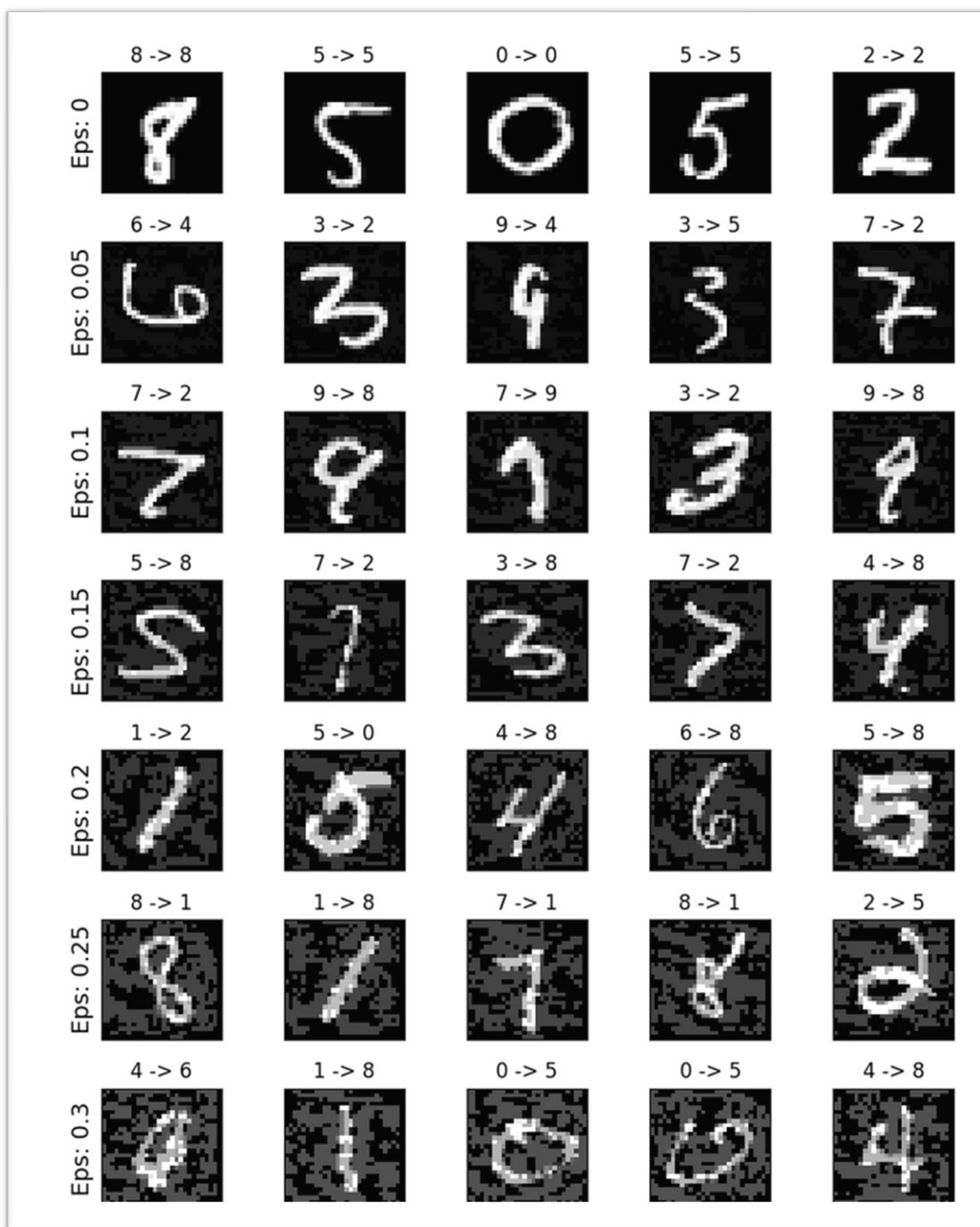


Figura 5. 10. – Imagini adversariale generate pentru diferite valori ale lui epsilon folosind FGSM

6. Concluzie

În această lucrare am încercat să abordez problema securității algoritmilor de *machine-learning* prin perspectiva utilizării ansamblurilor de modele ca mijloace robuste împotriva atacurilor adversariale.

În capitolele de început am prezentat potențialul dăunător al exemplelor adversariale. Deși acestea reprezintă o provocare relativ nouă pentru cercetători, studii recente au dovedit că foarte multe tehnologii ce folosesc algoritmi de *deep-learning* sunt predispuse acestor atacuri.

Ca urmare a experimentelor prezentate anterior am observat modul în care două tipuri de atacuri diferite pot influența semnificativ performanțele rețelelor convoluționale. Mai mult de atât, în urma utilizării ansamblului ca un mijloc de creștere a robusteții modelelor s-a observat că ansamblele ar putea fi utile împotriva exemplelor adversariale, întrucât numărul de interogări mediu pentru ansamblu a fost mai mare decât numărul mediu de interogări pentru două din cele trei rețele. Cu toate acestea, rețeaua care a atins cel mai mare număr de interogări, a obținut cea mai mică acuratețe pe datele de testare.

La momentul scrierii acestei lucrări nu există încă tehnici sigure împotriva atacurilor adversariale ce reușesc o apărare eficientă atât în condiții *white-box*, cât și în condiții *black-box*. Astfel, problema exemplelor adversariale rămâne deschisă.

Sunt de părere că, pentru îmbunătățirea rezultatelor obținute se pot încerca următoarele abordări, sau chiar o combinație între acestea. În cazul de față se pot folosi metode de prelucrare a datelor de intrare precum *noise-reduction* sau o tehnică în care, fiecărui pixel i se poate atribui media pixelilor din jurul lui, iar astfel modificările realizate de un algoritm de generare a exemplelor adversariale ar fi, în mare parte, eliminate. În cazul utilizării altor seturi de date, se poate antrena fiecare rețea din ansamblu în condiții adversariale. O altă idee ar fi aceea de a realiza un ansamblu compus dintr-o parte de detectare a exemplelor adversariale și o altă parte ce conține mai multe rețele, dintre care doar unele să fie antrenate pe exemple adversariale. În situația în care *inputul* este catalogat ca fiind posibil adversarial, rețelele ce au fost antrenate în aceste condiții vor avea o pondere mai mare în deciderea rezultatului final, iar dacă *inputul* este considerat nealterat, rețelele antrenate în condiții normale vor hotărî predicția ansamblului. Cu toate acestea, asemenea idei sunt foarte costisitoare din punct de vedere al timpului, întrucât antrenarea adversarială reprezintă un proces lent.

Bibliografie

1. Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986). <https://doi.org/10.1038/323533a0>
2. Lemaréchal, C. "Cauchy and the Gradient Method" (PDF). *Doc Math Extra*: 251–254 (2012)
3. Rajat Raina, Anand Madhavan, Andrew Y. Ng, [ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning](#), Pages 873–880, June 2009 <https://doi.org/10.1145/1553374.1553486>
4. Christian Szegedy, Dumitru Erhan, Ian Goodfellow et al., Intriguing properties of neural networks, <https://arxiv.org/abs/1312.6199>
5. Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy, Explaining and Harnessing Adversarial Examples, 2015
6. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Going deeper with convolutions, 2014, <https://arxiv.org/abs/1409.4842>
7. Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016, pages 582–597, 2016
8. Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In International Conference on Learning Representations, 2018.
9. Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In International Conference on Learning Representations, 2018.
10. Hossein Hosseini, Yize Chen, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. Blocking transferability of adversarial examples in black-box learning systems. CoRR, abs/1703.04318, 2017.
11. Aamir Mustafa, Salman Khan, Munawar Hayat, Roland Goecke, Jianbing Shen, and Ling Shao. Adversarial defense by restricting the hidden space of deep neural networks. International Conference on Computer Vision (ICCV), 2019.

12. Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten.
Countering adversarial images using input transformations. CoRR, abs/1711.00117, 2017.
13. Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In International Conference on Learning Representations, 2018.
14. Alexey Kurkain, Ian Goodfellow, Samy Bengio, Adversarial examples in the physical world, 2016, <https://arxiv.org/abs/1607.02533>
15. Alexey Kurakin, Ian Goodfellow, Samy Bengio, Demo to paper "Adversarial examples in the physical world", 2016, https://youtu.be/zQ_uMenoBCK
16. Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, Ananthram Swami, Practical Black-Box Attacks against Machine Learning, 2017, <https://arxiv.org/abs/1602.02697>
17. Vahid Behzadan, Arslan Munir, Vulnerability of Deep Reinforcement Learning to Policy Induction Attacks, 2017, <https://arxiv.org/abs/1701.04143>
18. Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, Pieter Abbeel, Adversarial Attacks on Neural Network Policies, 2017, <https://arxiv.org/abs/1702.02284>
19. Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, Michael Wellman, Towards the Science of Security and Privacy in Machine Learning, 2016, <https://arxiv.org/abs/1611.03814>
20. Chuan Guo, Mayank Rana, Moustapha Cisse, Laurens van der Maaten, Countering Adversarial Images using Input Transformations, 2018, <https://arxiv.org/abs/1711.00117>
21. Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. Proceedings of SIGGRAPH 2001, pages 341–346, August 2001.
22. Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, Aleksander Madry, Adversarial Examples Are Not Bugs, They Are Features, 2019, <https://arxiv.org/abs/1905.02175>
23. Yann LeCun, Gradient-Based Learning Applied to Document Recognition, 1990, <https://bit.ly/3hKr7vD>
24. <https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html>
25. Yann LeCun, Corinna Cortes, Christopher J.C.Burges, The MNIST Database, <http://yann.lecun.com/exdb/mnist/>

26. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.
27. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. 2015 arXiv:1512.03385.
28. Chuan Guo, Jacob R. Gardner, Yurong You, Andrew Gordon Wilson, Kilian Q. Weinberger, Simple Black-box Adversarial Attacks, 2019, <https://arxiv.org/abs/1905.07121>