Verificarea rețelelor neuronale folosind cGAN

Laptedulce Anastasia
Morariu Ioana-Alexandra
Romaneț Rareș
Domșa Emanuel
Diaconu Laura

GitHub

Abstract

In this paper we explored and tested the use of Conditional Generative Adversarial Networks and it's features in the context of image recognition tasks. For testing matters we use the cGAN benchmark from VNNComp 2023. The paper includes description of the above mentioned benchmark, details of configuration and running alpha_beta_crown tool and a short interpretation of the obtained results.

Introducere

Verificarea rețelelor neuronale este un proces esențial în dezvoltarea și implementarea acestor modele avansate. Această practică reprezintă un pilon fundamental din mai multe motive cheie.

În primul rând, corectitudinea și fiabilitatea rețelelor neuronale sunt imperative. Acestea sunt utilizate într-o varietate de domenii, de la medicină și tehnologie la securitate cibernetică și vehicule autonome. Verificarea asigură că aceste rețele operează conform așteptărilor, furnizând rezultate precise și de încredere într-o gamă largă de scenarii.

Siguranța este un alt aspect crucial. În aplicații critice, cum ar fi cele medicale sau cele legate de siguranța vehiculelor, erorile în funcționarea rețelelor neuronale pot avea consecințe grave. Verificarea acestora este vitală pentru a identifica și a remedia potențialele vulnerabilități care ar putea compromite siguranța sistemelor.

De asemenea, verificarea rețelelor neuronale ajută la prevenirea bias-ului și discriminării. Aceste rețele pot fi susceptibile la prejudecăți încorporate din datele de antrenament. Prin testare și evaluare riguroasă, se poate identifica și corecta aceste bias-uri pentru a asigura obiectivitate și echitate în rezultatele furnizate.

Descrierea data set-ului cGAN

În contextul verificării rețelelor neuronale, un benchmark este constituit dintr-o serie de modele neurale testate și specificațiile riguroase ce trebuie satisfăcute pentru a valida corectitudinea și securitatea acestora.

Modelul neuronal este reprezentat sub forma fișierelor în format .onnx. Aceste fișiere cuprind detaliile arhitecturii modelului, precum straturile rețelei neuronale și parametrii, însă nu includ datele reale de antrenament.

Specificările care sunt supuse verificării sunt reprezentate printr-un set de fișiere în format .vnnlib, care conțin informații esențiale despre rețeaua neuronală. Aceste detalii includ dimensiunile stratului de intrare și de ieșire, precum și intervalul sau domeniul de valori admise pentru activările neuronale.

Benchmark-ul utilizat testează un model deosebit de reațea neuronală care se numește rețea adversară generativă condiționată. Pentru o înțelegere mai bună a celui din urmă vom descrie mai întâi ce este o rețea neurală adversară generativă.

GAN

Este un model neuronal mai special [1] a cărui concept poate fi reprezentat ca un joc între două rețele neuronale distincte adversare sau, altfel spus, între doi jucători. Primul jucator este numit generator, iar al doilea discriminator.

Rolul generatorului este de a genera date false, bazate pe datele de intrare, care să pară cât mai reale. Generatorul are scopul de a păcăli pe al doilea jucător - discriminantorul.

Scopul celui de-al doilea jucator este de a determina ce imagini sunt reale și care sunt false, adica realizate de generator. Dacă discriminatorul interpretează corect atunci primește feedback pozitiv, dacă interpretează greșit atunci primește feedback negativ. Discriminatorul are acces la datele de ce ies din generator, dar si la datele de antrenament.

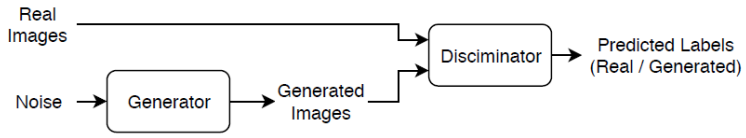

Figure 1: Mod de lucru al GAN

Ambii jucători învață și se îmbunătățesc în timp. Generatorul devine mai bun în a crea falsuri convingătoare, iar discriminatorul își îmbunătățește capacitatea de a spune dacă ceva este autentic. De-a lungul timpului, rețeaua ajunge într-un punct în care datele produse de generator vor arăta aproape imposibil de distins de datele din lumea reală.

cGAN

cGAN, prescurtare de la Conditional Generative Adversarial Networks [2], ghidează procesul de creare a datelor prin încorporarea unor etichete specificice în GAN. Rețele adversare - generatorul și discriminatorul - se orientează după aceste etichete.

Generatorul utilizează etichetele pentru a creea date false care imită datele reale și respectă condiția setată. Și la fel ca în modelul GAN, discriminatorul va distinge între datele falsificate produse de generator și datele autentice corespunzătoare condiției date.

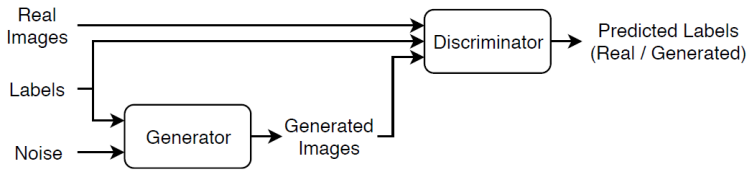

Figure 2: Mod de lucru al cGAN

De exemplu, să presupunem că ați folosit un spectru larg de imagini cu flori pentru a antrena un GAN capabil să producă imagini false cu flori. Deși puteți folosi modelul dvs. pentru a genera o imagine a unei flori aleatorii, nu îi puteți instrui să creeze o imagine a, de exemplu, a unei lalele sau a unei floarea soarelui.

GAN condiționat (cGAN) ne permite să condiționăm rețeaua cu informații suplimentare, cum ar fi etichetele de clasă. Înseamnă că în timpul antrenamentului, transmitem imagini în rețea cu etichetele lor reale (trandafir, lalele, floarea soarelui etc.) pentru ca aceasta să învețe diferența dintre ele. În acest fel, obținem capacitatea de a cere modelului nostru să genereze imagini cu anumite flori.

cGAN benchmark din vnncomp2023

Obiectivul [3] modelului neuronal din benchmark-ul cGAN din competiția vnncomp2023 este de a genera imagini ale camerei care conțin un obstacol al vehiculului situat la o anumită distanță în fața vehiculului ego, unde distanța este controlată de condiția distanței de intrare. În figura 3 sunt ilustrate exemple de imagini create de generatorul din cadrul retelei neurale.

Figure 3: Imagini generate

Generatorul primește două intrări:

- o condiție de distanță (1-d scalar), între 0 - 1(normalizat de la 0m la 30m). De exemplu, dacă condiția de distanță este 0,5, înseamnă că va fi generată o imagine ce are obstacolul la 15m in fața vehiculului ego.
- un vector de zgomot care controlează mediul (vector 4-d)

Ca și rezultat genratorul returnează o imagine generată.
Discriminatorul consideră imaginea returnată de generator ca și data de intrare și emite ca rezultat urmatoarele:

- un scor real/fals (1-d scalar), adică dacă trebuie să prezică dacă imaginea a fost realizată de generator sau nu
- o distanță prezisă (1-d scalar), distanță estimată de discriminant dintre vehiculul ego și vehiculul obstacol

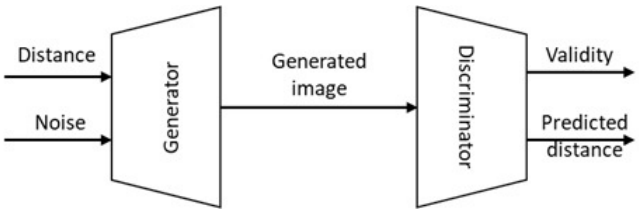

Figure 4: Mod de lucru cGAN benchmark

Pentru verificare, am putea combina aceste două componente împreună și am putea stabili specificații de verificare adecvate pentru distanța de intrare, zgomotul de intrare și distanța estimată.

Configurare alpha_beta_crown

Pentru rularea benchmark-ului cGan am ales ca și tool-uri alpha_beta_crown si NeuralSAT, verificând prima dată cu atenție dacă acestea au putut rula setul de date, figura 5. Am facut această alegere deoarece am dorit o comparație dintre primul tool care a obțiunut un timp de verificare mai eficient, iar cel de al doilea care a obținut cel mai ineficient timp de verificare.

Alpha_beta_CROWN a fost testat pe Python 3.11, sistemul de operare utilizat a fost Ubuntu 20.04 prin Windows Subsystem for Linux (WSL) pe Windows 10, folosind un GPU de laptop NVIDIA GeForce RTX 3060. A fost instalat într-un mediu miniconda. În urmatorii pasi se va explica cum a fost facută instalarea și rularea tool-ului.

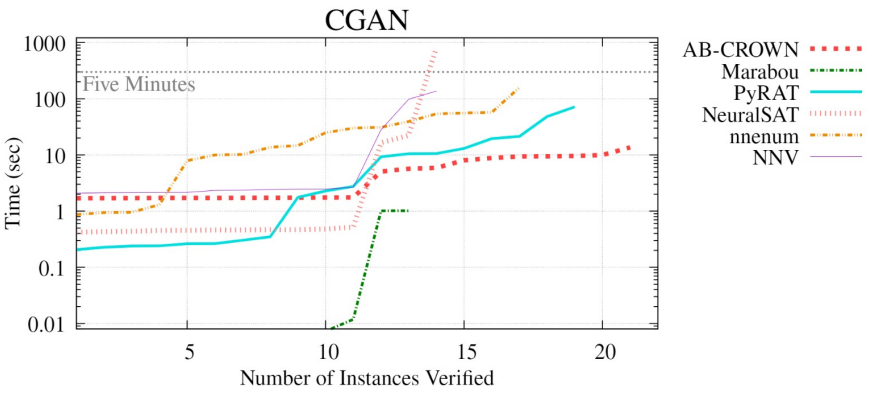

Figure 5: Rezultatele rulării tool-urilor pentru benchmark-ul cGan

Pai de Instalare:

1. Instalarea Miniconda a fost realizată prin linia de comandă (bash) cu următorii pași:

```
mkdir -p ~/miniconda3
wget https://repo.anaconda.com/miniconda/Miniconda3-
latest-Linux-x86_64.sh -O ~/miniconda3/miniconda.sh
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
rm -rf ~/miniconda3/miniconda.sh
~/miniconda3/bin/conda init bash
```

2. Clonarea repository-ului de la adresa [4]:

```
git clone --recursive https://github.com/Verified-
Intelligence/alpha-beta-CROWN.git
```

3. Clonarea submodulului `auto_Lirpa` în interiorul directorului `alpha-beta-CROWN`:

```
git clone https://github.com/Verified-
Intelligence/auto_LiRPA.git alpha-beta-CROWN/auto_LiRPA
```

4. Configurarea mediului Conda:

```
conda deactivate; conda env remove --name alpha-beta-crown
conda env create -f complete_verifier/environment.yaml
    --name alpha-beta-crown
conda activate alpha-beta-crown
```

5. Crearea directorului `vnncomp2023_benchmarks` și clonarea repository-ului [5] în interiorul acestuia. Modificarea **root path-ului** în fișierul `cgan.yaml` conform locației reale a directorului `cgan`.

6. Rularea verificatorului:

```
python abcrown.py --config exp_configs/vnncomp23/cgan.yaml
```

7. Pentru a rezolva eroarea legată de `libcudnn_cnn_infer.so.8`, adăugarea următoarei linii în `.bashrc`:

```
export LD_LIBRARY_PATH=/usr/lib/wsl/lib:$LD_LIBRARY_PATH
```

8. Rularea comenzii și salvarea rezultatelor în `results.txt`:

```
python abcrown.py --config exp_configs/vnncomp23/cgan.yaml
    > results.txt
```

Cel mai dificil pas a fost remedierea erorii legate de libraria libcudnn_cnn_infer.so.8., fiind si pasul care a durat cel mai mult timp. Am rezolvat aceasta eroare folosind multiple sugestii gasite pe diferite platforme online [6].

Interpretare rezultate

Rezultatele obținute după rularea toolului alpha_beta_crown pe benchmark-ul cGAN au fost sub forma unui stream în consolă, dupa cum se vede în figura 6.

Pentru a putea interpreta datele rezultate le-am clasificat manual într-un fișier excel. Câpurile de tabel sunt urmatoarele:

- benchmark(benchmark-ul pentru care avem înregistrate rezultatele)
- model_neuronal(fișierul ONNX pentru care a rulat tool-ul)
- specificații(fieșierul VNNLIB pentru care a rulat tool-ul)
- rezultat(poate avea valori sat/unsat; sat înseamnă că modelul indeplinește condițiile din fișierul VNNLIB, adică distanța aproximată de discriminator, este aceeași cu distanța pe care a primit-o generator-ul ca și data de intrare; unsat înseamnă că discriminatorul nu a reușit să facă o aproximare corectă)

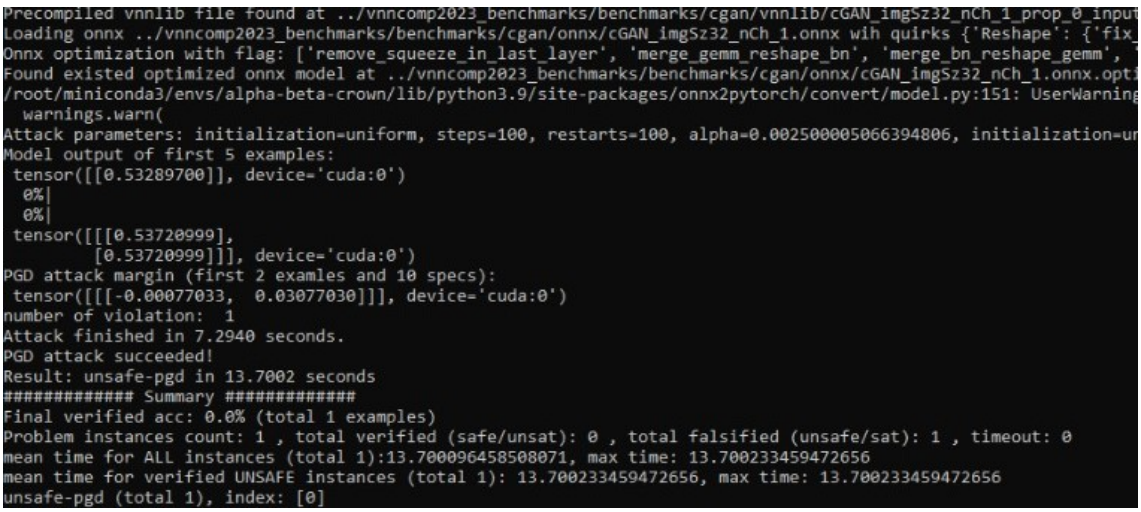

Figure 6: Rezultat stream consola

- timp_de_verificare(timpul de rulare a fișierului ONNX pentru specificatiile din fișierul VNNLIB, exprimat în secunde).

Aici se pot vedea rezultatele pe care le-am obținut. Pentru a ne verifica corectitudinea rezultatelor obținute, am facut o analiză comparativă între fișierul obținut la rularea noastră, aici, și cel obținut din competiție, aici.

Am observat că pentru fiecare intrare, în ambele fișiere, s-a obținut același rezultat(sat/unsat). O diferență dintre cele două fișiere o constituie coloana ce reprezinta timpul de verificare. Am remarcat că în fișierul obținut de noi, timpul de verificare înregistrat este mai mic cu aproximativ 3 secunde pentru intrările unde rezultatul este satisfiabil. În schimb, pentru intrările cu rezultat nesatisfiabil timpul de verificare este considerabil mai mare.

Concluzii

Bibliography

[1] "What are generative adversarial networks?" [Online]. Available: https://dzone.com/articles/what-is-a-conditional-generative-adversarial-netwo

[2] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," 2014.

[3] "vnncomp2023 chat." [Online]. Available: https://github.com/stanleybak/vnncomp2023/issues/2

[4] "Crown repository." [Online]. Available: https://github.com/Verified-Intelligence/alpha-beta-CROWN

[5] "cgan repository." [Online]. Available: https://github.com/ChristopherBrix/vnncomp2023_benchmarks/tree/main/benchmarks/cgan

[6] ".bashrc fix." [Online]. Available: https://discuss.pytorch.org/t/libcudnn-cnn-infer-so-8-library-can-not-found/164661