République Algérienne Démocratique et populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'électronique et de l'informatique
Département d'informatique

Master I

Systèmes Informatiques Intelligents

Apprentissage Automatique et Réseaux de Neurones

Rapport de mini projet

Generative Adversarial Networks

Réalisé par :

BENDJOUDI Meriem G3

HAMMOUCHE Manel Yasmine G2

2019/2020

1. **Introduction**

Machine learning est la capacité de la machine d'apprendre à faire des décisions toute seule sans l'avoir programmé. Et cela est fait avec des techniques plus profondes que la programmation classique. Parmi les techniques intéressantes de l'apprentissage de la machine est l'apprentissage profond.

L'apprentissage profond ou comme il est connu en anglais le « Deep learning » est une technique quel été inspiré par la structure et le fonctionnement du cerveau humain dans la circulation des connaissances à travers ces neurones. La structure artificielle obtenue connu en réseaux de neurones artificiels.

Des nouveaux techniques d'apprentissage profond ont été mises en lumière et les modèles génératifs sont connu un grand succès pour ces résultats impressifs à l'œil. L'un de ces modèles, les réseaux antagonistes génératifs qu'Ian GOODFLOW les déclarait comme la future du Deep learning.

Dans ce mini-projet nous nous sommes détaillé les différentes phases que nous avons faite pour construire un modèle GAN que nous allons par la suite implémenter dans une application.

2. **réseaux antagonistes génératifs:**

Réseaux antagonistes génératifs ou « Generative Adversiral Networks » en anglais connue par «GANs», est un nouveau modèle génératif proposé par « Ian GOODFLOW » dans une de ces publications en 2014[1].

Cette technique consiste à lancer un apprentissage de deux différents réseaux de neurones simultanément, connurent générateur et discriminateur.

Dans cet apprentissage contradictoire l'objectif du générateur est de former des nouvelles données aussi réelles que possibles pour tromper le discriminateur qui s'occupe de la classifier entre donnée réelle et donnée générer par le générateur qu'elle est une fausse donnée.

Initialement le générateur est si mauvais dans la génération de nouveaux échantillons, que le discriminateur peut le faire classer facilement. Et avec le temps le G et D devient de plus en plus bon.

Donc il est évident qu'il faut produire un équilibre entre les deux modèles afin d'avoir un bon résultat.

La figure ci-dessous montre l'architecture générale de GAN.

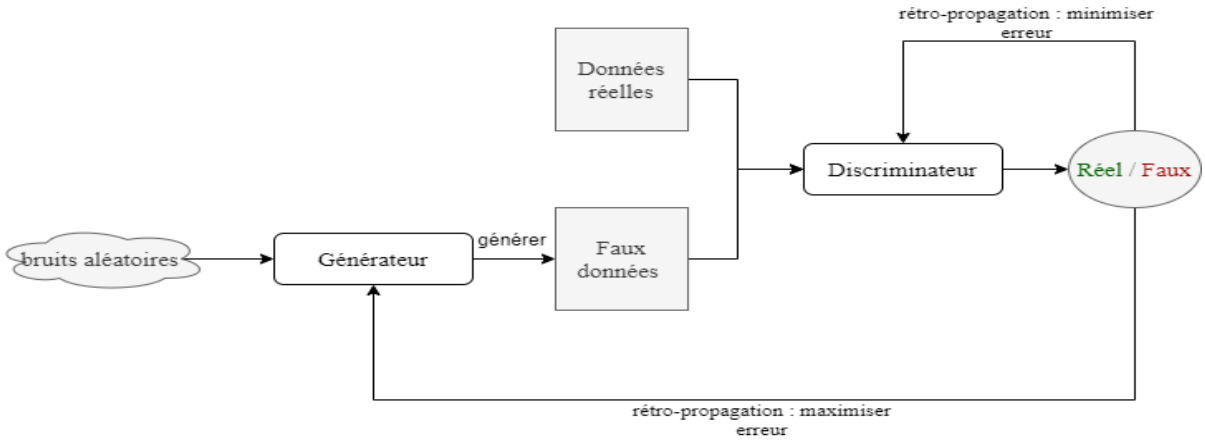

Figure 1 : l'architecture du GAN.

Note : le discriminateur aide le générateur à produire de meilleur résultat après chaque itération et cela car le générateur mis à jour ces poids selon le résultat du discriminateur.

L'architecture du générateur et du discriminateur se diffère d'un type à un autre selon le problème qu'on veut le résoudre.

3. Domaine d'utilisation des GANs

GANs étant données un thème de recherche de nos jours, plusieurs types de GANs ont été proposé pour résoudre un/des problèmes du plusieurs domaine.

Nous citons quelques exemples:

- StackGans : qui sont utilisé dans les problèmes de la traduction du texte à une image.
- CycleGans : sont principalement fait pour la traduction imagée à image (jour ← → nuit, cheval ← → zèbre, photo réelle ← → dessin)
- AnoGans :qui peut détecter les anomalies dans un scanner.
- Et bien d'autres.

4. Data set :

Nous nous somme documenté sur les différents data proposé, et nous avons choisi «MNIST handwritten digit data base » qui contient les différents styles d'écritures de digits par main.

Notre choix a été base sur les critères suivants :

- Préférence du thème.
- La difficulté d'entrainement de la technique.
- Les capacités de nos machines.
- Le choix de l'application à concevoir.

4.1. Description des données : [2]

Le MNIST (Modified ou Mixed National Institute of Standards and Technology) est une base de données comportant des chiffres manuscrits. Elle comporte 2 ensembles d'exemples:

- le premier ensemble comporte 60.000 exemples, représentent les données d'apprentissage ;
- le second 10.000 exemples qui sont les données de test.

Les chiffres sont normalisés en taille et centrés dans des images de 28 x 28 pixels.

4.2. Traitement des données

La data set n'a pas besoin d'un traitement spécial ; nous avons juste normalisé la valeur des images de [0, 255] à [-1, 1] et cela a été basé sur le fait que le discriminateur reçoit en entrée une image qui été produite par le générateur qui est à son tour utilise la fonction tangente hyperbolique (Tanh) comme fonction d'activation.

```
images_train = load_Data( directory )

images_train = (X_train - 127.5) / 127.5      #X_train sont les données d'apprentissage
```

5. Outils utilisés pour l'implémentation :

Nous avons utilisé le langage de programmation python avec sa bibliothèque d'apprentissage automatique « TENSORFLOW » pour l'implémentation et « Google Colaboratory » comme un environnement pour lancer l'apprentissage du modèle. Notre choix du Framework a été basé sur le fait qu'on peut accéder à tensorflow et keras au même temps.

6. Conception de l'architecture :

Nous nous sommes choisis la version originale du GANs présenté par Goodflow et ses collègues pour l'implémenter. Ce type est connu avec VannilaGAN.

Dans les sections suivantes nous allons détailler l'architecture de notre générateur G et le discriminateur D ainsi que la description du trainement et l'évaluation du modèle.

6.1. Discriminateur

Le discriminateur prend une image de taille 28 * 28 pixels et renvoie une probabilité entre 0 et 1.
Pour avoir un modèle qui répond au fonctionnement du discriminateur nous avons construit un réseau de neurone on utilisant 4 couches cachées entièrement connectés, avec une fonction d'activation "**leakyrelu**" pour chacune, ce qui donne de meilleurs résultats par rapport aux autres fonctions d'activation et cela aussi permet d'éviter le "vanishing gradient". Kailash Ahirwar a expliquer la cause de ce choix dans son livre intitulé *generative adversiral networks project* où il a dit « presque le gradient de cette fonction ne sature pas lors de la rétropropagation » [3]. Et pour la dernière couche, une couche dense avec **sigmoïde** comme fonction d'activation, car la valeur de cette fonction peut être interpréter comme une probabilité que le générateur attribue un input réel
Il faut noter que nous avons ajouté un "**dropout**" avant la couche de sortie pour éviter le débordement.

6.2. Générateur :

Le générateur est un réseau de neurones qui prend un bruit de taille 128 et retourne une image de taille 28 * 28 pixels.
Pour la conception de ce modèle, nous avons utilisé 4 couches cachées qui sont entièrement connectées, tel que chaque couche cachée à un double nombre de neurones que la couche qui la précède et afin d'éviter le déséquilibre des gradients et stabiliser l'entraînement, nous avons utilisé pour chaque couche un « **batchNormalization**» avec un « **momentum** » qui égale à 0.8
Nous avons « **Relu** » comme fonction d'activation pour toutes les couches caches sauf la dernière couche où on a utilisé « tanh » pour écraser les valeurs d'entrée dans une plage comprise entre -1 et 1.

Pour former l'architecture finale du GAN nous avons regroupé le générateur et le discriminateur dans un seul modèle à l'aide du réseau de neurones Sequentiel de l'API keras.

```
Model=Sequentiel()

Model.add(discriminator)

Model.add(generator)
```

Le tableau récapitulatif suivant montre les architectures utilisées dans notre modèle GAN.

Composant	**Architecture**	**Fct d'act couche(s) cachée(s)**	**Fct d'act couche de sortie**	**Optimize**	**Fct calcule d'erreur**
Générateur	RN entièrement connectée	Relu	Tanh	Adam	binary_crossentropy

Discriminateur	RN entièrement connectée	Leaky relu	Sigmoid	adam	Binary crossentropy

7. Description de trainement :

Dans le modèle GAN nous sommes face à l'apprentissage de deux modèles contre eux.

Nous commençons par préparer les deux composants du GAN en utilisant l'optimisateur Adam et **Binary crossentropy** comme fonction d'erreur nous lançons l'apprentissage du discriminateur.

```
Dis.compile(loss='binary_crossentropy',optimizer=Adam(lr=0.0002, beta_1=0.5))
```

Après avoir fait le trainement du D on bloque Lui bloque pour faire l'apprentissage de G.

```
#bloquer D
dis.trainable=false
#lancer l'apprentissage de G
gan.compile(loss='binary_crossentropy',optimizer=Adam(lr=0.0002, beta_1=0.5))
```

Maintenant, les deux modèles sont prêts pour le grand trainement. Nous reste de charger notre ensemble de données et de normaliser grâce a « tensorflow.keras »qui nous donne la possibilité de charger la dataset mnist.

```
#importer la dataset
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
#normaliser les images de trainement
X_train = ( X_train -  127.5)  / 127.5
X_train = np.expand_dims(X_train, axis=3)
#normaliser les images de test
X_test=(X_test- 127.5) / 127.5
X_test = np.expand_dims(X_test, axis=3)
```

Il faut avoir des exemplaires du fausse donnée pour que le discriminateur peut commencer à apprendre à différencier et classifier la donnée entre réel et faux, et comme nous allons lancer un apprentissage par batch pour le discriminateur donc nous avons besoin de label pour les images réelles avec un vecteur de 1 et pour les fausses un vecteur à 0 pour. Ces labels dérige le discriminateur à la sortie qu'il doit la prédire. La bibliothèque numpy nous a aidé à initialiser les vecteurs.

```
#label pour la réelle donnée
real=np.ones((BATCH_SIZE,1))
#label pour la fausse donnée
fake=np.zeros((BATCH_SIZE,1))
```

Pour améliorer l'apprentissage nous avons utilisé un « smoothing label » qui est recommandé pour GANs par Soumith Chintala qui a expliqué « label smoothing, c'est-à-dire si vous avez deux Target labels: Real = 1 et Fake = 0, Alors pour chaque échantillon entrant, s'il est réel, remplacez label par un nombre aléatoire entre 0,7 et 1,2, et s'il s'agit d'un faux, remplacez-le par 0,0 et 0,3 »[5]

```
real = real- 0.3 + (np.random.random_sample(real.shape) * 0.5)
fake = fake+ np.random.random_sample(fake.shape)*0.3
```

Pour que le générateur améliore de la qualité de ses échantillons générer il à besoin du résultat du discriminateur (la probabilité retourner par sa fonction d erreur) pour mettre à jour ces poids pour avoir des résultats meilleurs que l'itération précédente ce qui fait loss function du générateur est calculé après l'apprentissage du discriminateur. Il faut noter que nous avons lancé l'apprentissage que notre architecture GAN avec la fonction d'erreur « Binary_crossentropy » comme nous avons choisi l'optimisateur « Adam » ou on a fixé le taux d'apprentissage du générateur et discriminateur à 0.0002 et le paramètre bêta à 0.5.L'utilisation de binary crossentropy pour mesure de la différence entre probabilités calculées et probabilités réelles pour prédictions plus nos prédictions sont éloignées proviennent des vraies images.

Note : toutes valeurs indiquées dans les descriptions sont prises après une séquence d'expérimentation que nous allons détailler plus en bas.

8. évaluation du modèle :

L'évaluation des modèles génératifs y compris les modèles GANs, est trop difficile et la performance ne peut pas être jugée. Dans ce cadre, plusieurs thèses ont été publiées avec des solutions à ce problème.

Pour cela nous avons opté à « Fréchet Inception Distance » pour l'évaluation de notre modèle avec l'observation des qualités générer car il y a des cas où fid elle est petite mais les images générer sont loin d'être bien.

Fréchet Inception Distance :

« Le score FID est utilisé pour évaluer la qualité des images générées par les réseaux antagonistes génératifs, et des scores inférieurs se sont avérés bien corrélés avec des images de meilleure qualité. »[4]

De manière générale, le score FID est le calcul d'un rapport entre les données de test et les nouveaux échantillons générés par le modèle GAN. Le plus le score petit le plus la performance est bon ce qui est fait le générateur génère des exemples aussi réels que les données originales.

9. Expérimentation

Nous nous avons sélectionné quelque architecture pour régler les hyper-paramètre afin d'avoir un modèle avec une meilleure performance.

D'abord nous avons commencé par régler quelques paramètres qui sont montrés dans le tableau suivant :

Optimizer	G-lr	Fonction activation G	D-lr	Fonction activation D	Batch-size	noise	Qualite_Image_Generted	FID
Adam	0.0002	leakyRelu	0.0002	leakyRelu	128	100	Bien	48.144
Adam	0.0002	leakyRel	0.0006	LeakyRel	128	100	Moyenne	44.847

		u		u				
Adam	0.0006	leakyRelu	0.0002	leakyRelu	128	100	Moyenne	48.251
Adam	0.002	leakyRelu	0.002	leakyRelu	128	100	Mauvaise	139.832
Adam	0.00001	leakyRelu	0.00001	leakyRelu	128	100	Mauvaise	43.281
Adam	0.0002	leakyRelu	0.0002	LeakyRelu	100	100	Moyenne	49.730
Adam	0.0002	Relu	0.0002	Relu	128	100	Moyenne	49.882
Adam	0.0002	LeakyRelu	0.0002	Relu	128	100	Moyenne	54.143
Adam	0.0002	Relu	0.0002	leakyRelu	128	100	bien	53.305

Après avoir choisi les meilleures valeurs pour le taux d'apprentissage, la fonction d'activation, taille du bruit et la taille de lot. Le tableau ci-dessous montre les résultats des expérimentations qui ont été lancés pour régler les variantes de couches cachées et le nombre de neurone par couches.

Generateur hidden layer	Generateur number of neuron par layer	Discriminateur hidden layer	Discriminateur number of neuron par layer	Observation visuelle	FID
1	1024	1	1024	Moyenne mais vous peux les détecte	49.20
3	128,256,1024	3	128,256,1024	Bien	51.698
3	128,256,1024	4	128,256,512,1024	Bien et visible	52.494
4	128,256,512,1024	3	128,256,1024	moins Clair que precedents	54.175
4	128,256,512,1024	4	128,256,512,1024	Tres visible et bien	55.97

Meilleur modelé :

Après avoir réglé tous les hyperparamètres nous avons obtenu une meilleure performance que nous montres ces graphes de la fonction d'erreur et la fonction d'évaluation FID :

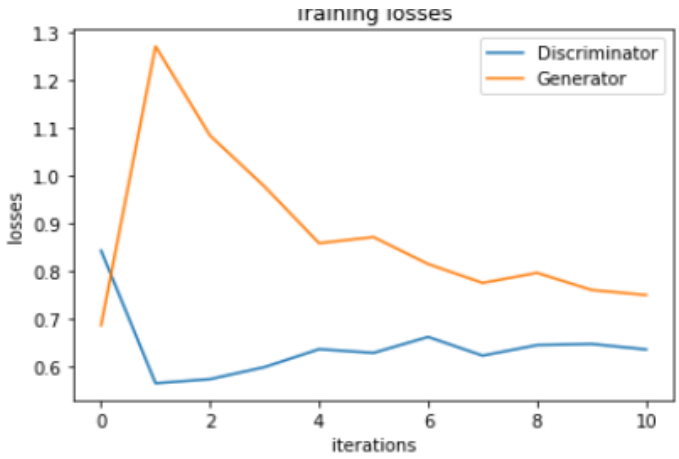

Figure 2: l'erreur du générateur et du discriminateur en fonction d'itérations du meilleur modèle.

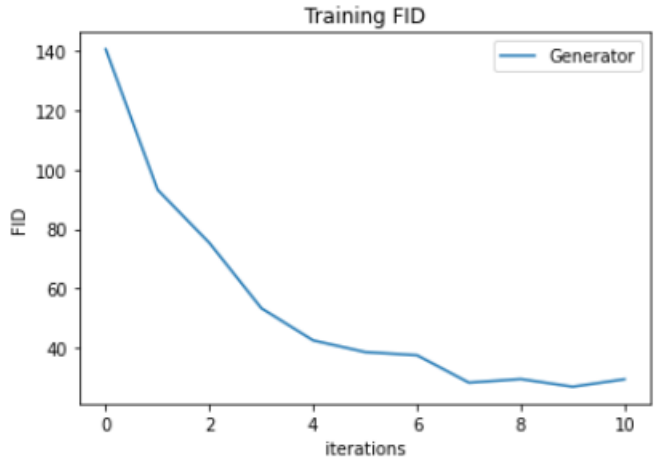

Figure 3: la fonction d'évaluation FID en fonction d'itérations du meilleur modèle.

Et voici un échantillon des données générer par notre modèle:

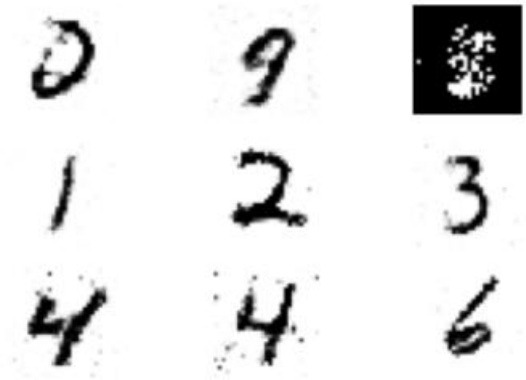

Figure 4 : un échantillon des images générer par notre modèle.

Application :

Pour l'exploit de notre solution nous avons réfléchi aux plusieurs idées :

- Utilisée les images générer pour faire une augmentation
- Bijoux personnalisez par des dates :
  L'idée est de données à l'utilisateur un champ de texte ou il rentre une date et choisit son modèle de bijoux préféré. Après on prend cette date et vérifiez la date entrée avec les numéros génère par notre modèle si c'est le cas on load le bijou et on écrit la date génèrer par notre générateur.
- Vêtement personnalisez par des digits (comme les vêtements sportifs)
- Jeux sodoku, Hitori

On a décidé de l'implémentation de notre solution avec le jeu sodoku.

9.1. Conception

Pour la conception d'application nous avons utilisé un modèle CNN (convolution neural network) qui a pour travail la prédiction des valeurs des digits générés par notre générateur qui sont par la suite utilisés pour construire la table de Sudoku.

Bien sûr si les images sont bien donc il va le prédite avec un taux d'erreur de 1% qui est notre cas, et si le cas contraire il est impossible de les prédite.

Nous avons pensé à produire un Sudoku avec les images générer ce qui n'été pas beau à l'œil et le joueur peut se tromper de reconnaitre le digit à cause de la qualité des échantillons générer (ces images peuvent être utilisées dans le cas d'augmentation des données).

9.2. Implementation

Afin de construire notre application web, nous avons utilisé l'outil Flask.

Flask : est un web framework écrit en python qui nous fournit des outils, des bibliothèques et des technologies qui nous permettons de construire une application web.

9.3. Application

Voici notre application.

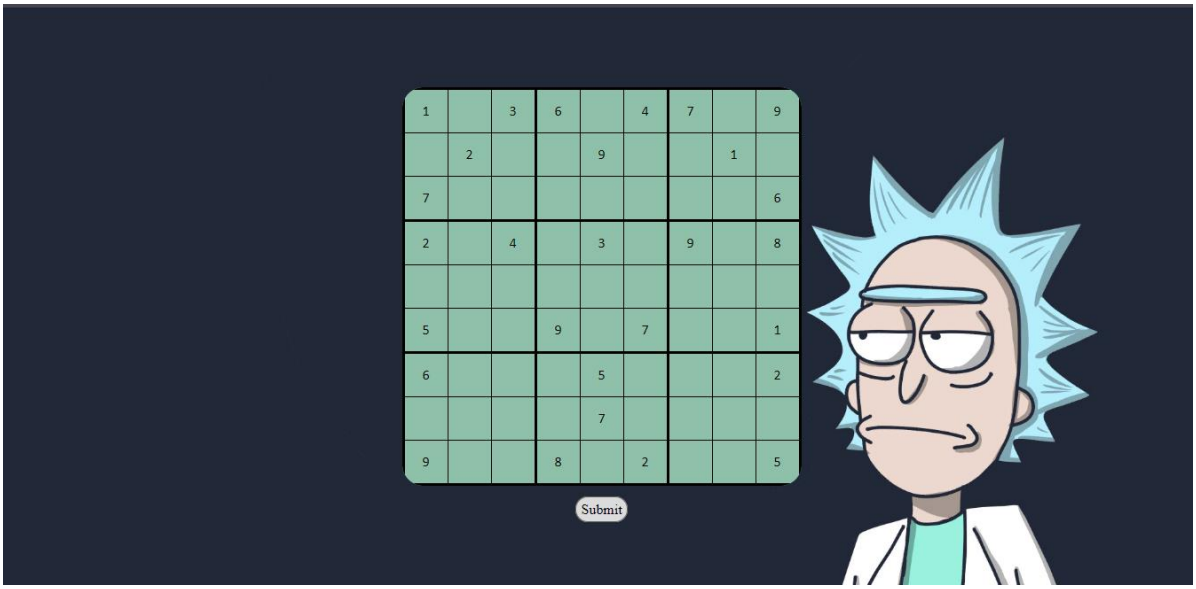

Figure 5 : page web de notre sudoku

10. Conclusion :

A travers ce mini-projet nous avons pus apprendre les notions de l'apprentissage profond et découvrit un nouveau modèle qui nous trouvons très intéressent et riche avec les informations, nous avons fait un nombre important d'expérimentations sur différentes architectures, ce qui nous a permis de voir de diverses résultats et décider par la suite l'architecture la plus adéquate à notre apprentissage.

Références :

[1] Generative Adversarial Networks Ian J. Goodfellow, Jean Pouget-Abadie∗ , Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio,2014,https://arxiv.org/abs/1406.2661

[2] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges." [Online].

Available:http://yann.lecun.com/exdb/mnist

[3]generative adversiral networks project ,Kailash Ahirwar, Packt Publishing 2019, http://gen.lib.rus.ec/

[4] GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, 2017.

[5] https://github.com/soumith/ganhacks,7months ago

Répartition des taches :

Tache	Membres
Documentation sur dataset	M.Hammouche, *M.Bendjoudi*
Documentation sur le modelé	M.Hammouche, *M.Bendjoudi*
Décision d'architecture a utilisée	M.Hammouche, *M.Bendjoudi*
Ecriture de programme et traitement de donnée	M.Hammouche
Documentation sur les paramètres et le choix a utilisée	M.Hammouche, *M.Bendjoudi*
Rédaction de rapport	M.Hammouche, *M.Bendjoudi*
Rédaction des slides	M.Hammouche, *M.Bendjoudi*

Implémentation d'application	*M.Bendjoudi*