UFR
DE MATHÉMATIQUES
ET INFORMATIQUE

Projet Apprentissage Profond

- Prédiction des données MNIST avec uniquement 100 labels-

Professeur :

Blaise Hanczar

Étudiant :

Mohamed Abdelhadi Boudjemai

Année universitaire : 2020 – 2021

Table des matières

1. Introduction :

Dans ces dernières années, les réseaux de neurones profonds ont réalisé de grands succès dans les tâches difficiles d'IA (l'intelligence artificielle) surtout les réseaux antagonistes génératifs (GAN) qui sont devenus un axe de recherche de l'intelligence artificielle. Inspirés du jeu à somme nulle à deux joueurs, les GAN comprennent un générateur et un discriminateur, tous deux formés selon l'idée d'apprentissage contradictoire. Le but des GAN est d'estimer la distribution potentielle d'échantillons de données réels et de générer de nouveaux échantillons à partir de cette distribution.

Depuis leur lancement, les GAN ont été largement étudiés en raison de leur énorme perspective d'applications, y compris l'informatique d'image et de vision, le traitement de la parole et du langage, etc. Ainsi, un GAN ou Generative Adversarial Network est une technique d'intelligence artificielle permettant de créer des imitations parfaites d'images ou autres données.

Une autre méthode qui prend aussi du poids en réseaux de neurones profonds est la "Pseudo-étiquette" ou pseudo label. Cette méthode d'apprentissage semi-supervisé simple et efficace et utilise un petit ensemble de données étiquetées avec une grande quantité de données non étiquetées pour améliorer les performances d'un modèle.

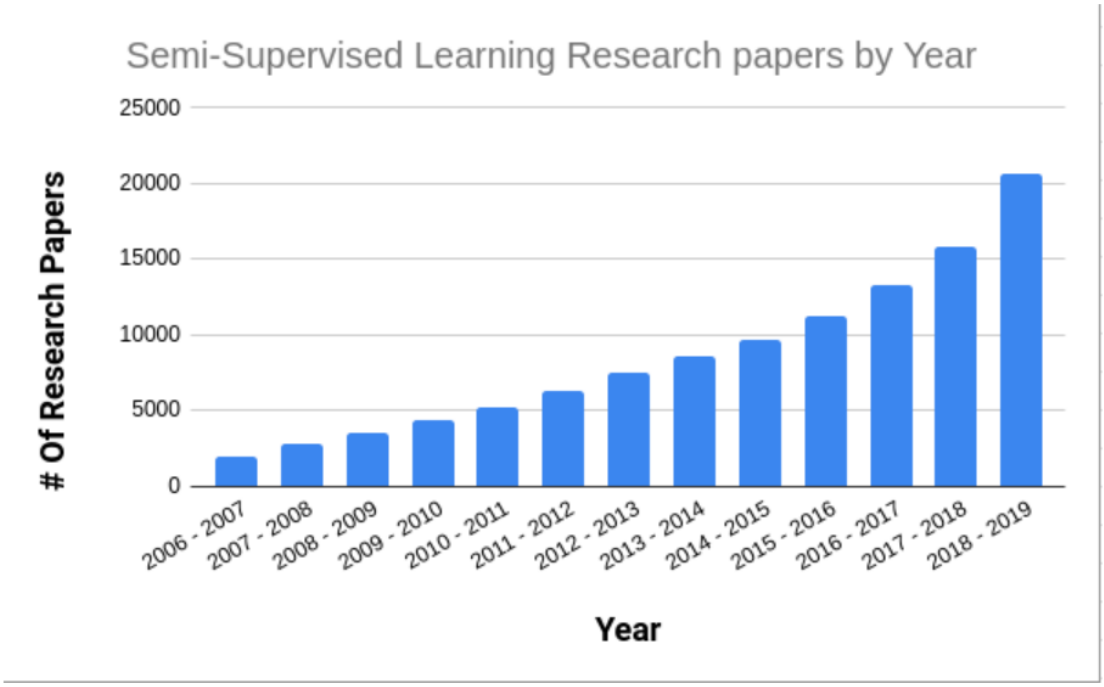
Dans ce rapport, nous décrivons une synthèse de la théorie des GAN et la méthode pseudo-label qui seront utilisé pour la génération d'images sur un jeu de données MNIST.

2. Objectif :

Dans ce projet, nous proposons des méthodes simples de formation réseau de neurones de manière semi-supervisée, notre jeu de donnée sera les données MNIST qui est une base de données de chiffres écrits à la main. C'est un jeu de données très utilisé en apprentissage automatique, ce dernier contiens une base d'apprentissage de 60000(x,y) et une base de test d'environ 10000(x,y) exemples.

L'objectif c'est de faire un réseau de neurones qui va prédire le label qui est sur l'image avec uniquement 100 labels ça veut dans notre base d'apprentissage on va avoir 59900(x) et 100(x,y) exemples ça c'est ce qu'on appelle l'apprentissage semi-supervisé qui consiste à apprendre un model avec à la fois des données qui contient de labels et des données qui non pas de labels.

Cependant, récemment, il y a eu un regain d'intérêt pour l'apprentissage semi-supervisé, qui se reflète à la fois dans la recherche universitaire et industrielle. Voici un graphique montrant le nombre d'articles de recherche liés à l'apprentissage semi-supervisé sur Google Scholar par année.



Documents de recherche sur l'apprentissage semi-supervisé par année

Nous nous intéressons à l'apprentissage d'un type de réseau de neurones appelé réseaux antagonistes génératifs (ou GAN pour *Generative Adversarial Network*). Ce type de réseau de neurones est particulièrement intéressant car il permet de générer de nouvelles données à partir d'une distribution inconnue. De plus, les GAN sont utilisés pour de nombreuses applications, principalement en image, comme la génération d'images à partir d'un texte, l'édition d'images existantes, ou la création de vidéo à partir d'images. Ils peuvent également être utilisés pour de la compression ou l'augmentation de la résolution d'images.

Dans ce projet, nous proposons des méthodes simples de formation réseau de neurones de manière semi-supervisée. Ainsi, le but de cette étude est de fournir une information complète sur le GAN et ses différents modèles dans le domaine de la synthèse d'images ainsi que la méthode pseudo-étiquette. Notre principale contribution à ce travail est la comparaison critique des variantes GAN populaires pour la génération d'images sur un jeu de données MNIST y compris l'efficacité de la méthode pseudo-étiquette.

La prédiction des données MNIST avec uniquement 100 labels car plusieurs documents ont mentionné que le pseudo-étiquetage est sensible aux prédictions initiales et que de bonnes prédictions initiales nécessitent un nombre suffisant de points étiquetés (par exemple 100 labels).

3.Description des méthodes utilisées :

Il existe plusieurs méthodes qui prennent du poids en réseaux de neurones profonds parmi la GAN et la Pseudo label.

3.1 Les réseaux antagonistes génératifs (GAN)

Les réseaux antagonistes génératifs (RAG ou GAN en anglais) sont des réseaux utilisant des couches de convolutions. Cependant, ils ne sont pas conçus pour analyser ou classer des données mais pour en générer. Ces réseaux ont été popularisés en 2014 par Ian J. Goodfellow et al. [1]

Le principe des réseaux antagonistes est la minimisation de multiples fonctions avec des objectifs antagonistes. Ils ont été utilisés pour faire des contre-exemples afin de tromper des réseaux de neurones classiques, ou pour trouver de manière automatique des techniques de cryptage. Ainsi, l'idée consiste à apprendre à un réseau de neurones à générer des exemples réalistes qui pourraient appartenir à la base de données d'apprentissage. Ce réseau de neurones est décomposé en deux modèles différents : un générateur et un discriminateur (voir Figure 1). Le générateur prend en entrée un vecteur de variables aléatoires et l'associe à une sortie dans l'espace des données X. Le discriminateur est un classifieur binaire qui prend en entrée des données de la base d'apprentissage et des données de la sortie du générateur. Son but est d'apprendre à différencier ces deux types de données. Le but du générateur est de tromper le discriminateur lorsqu'il génère des données. Au cours de l'apprentissage, la distribution de ces données en sortie se rapproche peu à peu de la distribution des données de la base d'apprentissage (voir Figure 2).

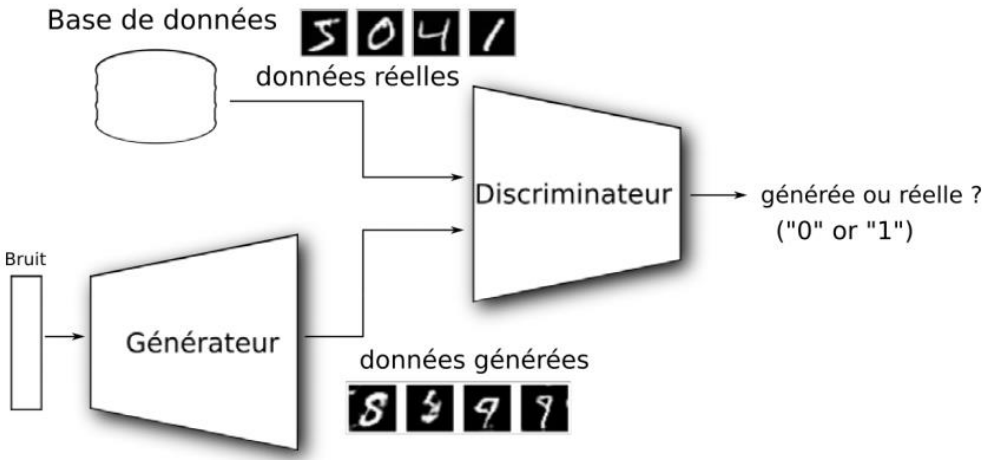


FIGURE 1: Architecture du GAN avec le générateur et le discriminateur

Le simple but de générer des nouvelles données a un intérêt limité dans l'utilisation des GAN. Des variantes existantes dans lesquelles des informations supplémentaires sont données au générateur afin de produire des données. Par exemple, dans les travaux de A. Odena et al. [2], les auteurs proposent d'ajouter la classe de la donnée en entrée du générateur. Cela permet au générateur de générer un type de donnée particulier, en plus d'améliorer son apprentissage. Avec un principe assez proche, il est possible de générer une image à partir d'un texte [3]. Celui-ci est encodé et donné à l'entrée du générateur ainsi qu'à l'intérieur du discriminateur. Les GAN peuvent également être utilisés pour augmenter la résolution d'une image en utilisant celle-ci en entrée [4]. De même, certains GAN peuvent être utilisés afin de changer le style d'une image par celui d'un artiste [5].

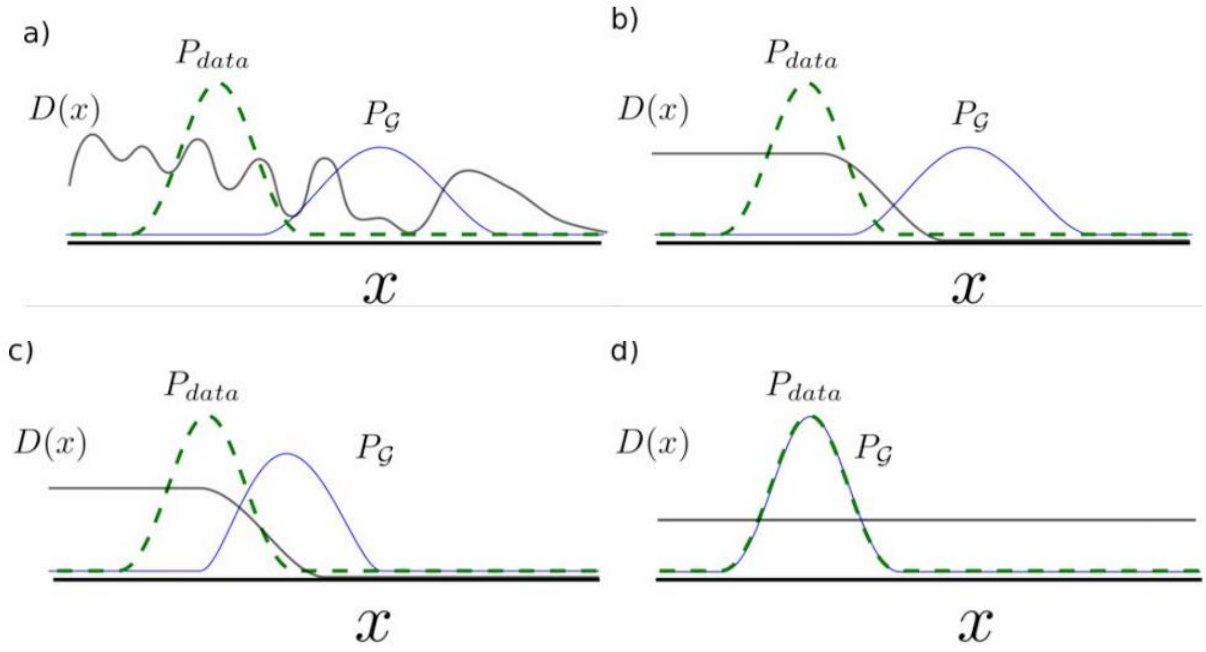


FIGURE 2:Distribution des données réelles P_{data}, des données générées P_G et sortie du discriminateur D(x) dans l'espace X au cours de l'apprentissage. Au début de l'apprentissage (a), le discriminateur est incapable de différencier les données réelles des données générées. Après quelques itérations, le discriminateur est capable de trouver la frontière entre les données réelles et les données générées (b). Afin de tromper le discriminateur, le générateur doit modifier ses paramètres afin de générer des données de plus en plus proches des données réelles (c), jusqu'à avoir la même distribution que celles-ci (d). Le discriminateur est alors incapable de différencier les données générées des vraies.

Les GAN diffèrent également des réseaux de neurones classiques par leur fonctionnement. Comme on peut le voir sur la figure 3, ils sont composés de deux réseaux en compétition. Le réseau générateur "G" qui doit générer les données que l'on souhaite et un réseau discriminant "D" dont le rôle est de déterminer si la donnée qu'il reçoit est une vraie image ou si elle a été générée par le réseau précédent. Après quoi les poids des deux réseaux sont modifiés selon le résultat de la prédiction du réseau discriminant. Plus l'entraînement réalise d'epoch plus le réseau discriminant sera efficace pour discerner les

vraies données de celles qui ont été générées, ce qui permet au réseau générateur de produire des images de plus en plus vraisemblables rapidement.

La figure 3 montre le GAN dans le cas des images.

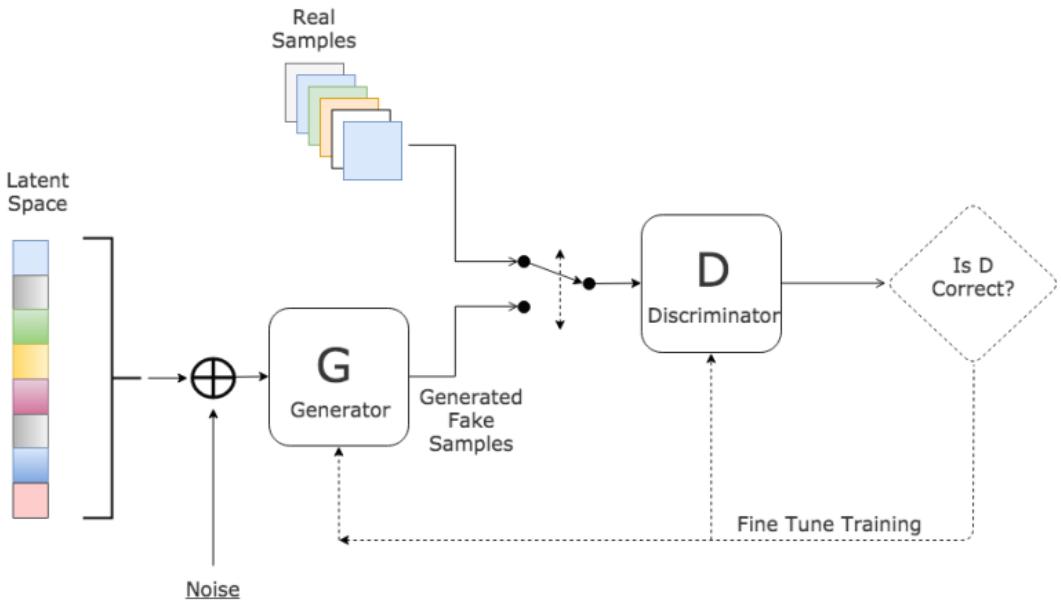


Figure 3: Schéma d'un GAN

La particularité des GAN est que leur phase d'apprentissage est non-supervisée, c'est-à-dire qu'aucun label (ou description) n'est requis pour l'apprentissage à partir des données. Un GAN classique est composé de deux éléments : un générateur G et un discriminateur D. Les deux sont des réseaux de neurones profonds. Le générateur prend en entrée un signal bruité (par exemple, des vecteurs aléatoires de taille k où chaque entrée suit une distribution normale N (0, 1)) et génère des données au même format que les données d'apprentissage en entrée (par exemple, une image de 128x128 pixels et 3 canaux de couleurs).
Le discriminateur est un classifieur binaire qui reçoit en entrée des données de ces deux sources : des données générées du générateur ou des données réelles de la base de données d'apprentissage. L'objectif du discriminateur est de déterminer de quelle source proviennent les données (Voir Figure 4).

Dans le cas des GAN, le but est de directement générer des échantillons provenant de P (x) à partir de la variable latente z < P (z). La distribution pdata(x) n'est donc jamais modélisée directement. Les GAN sont constitués de deux réseaux de neurones. Un Générateur Gθ(z) qui génère les échantillons x˜ < Gθ(z), ainsi qu'un Discriminateur Dϕ(x). Le rôle de Dϕ est de discriminer les x selon s'ils sont réels (x <Pdata), ou faux (x˜ < Gθ(z)). Le rôle du générateur est donc de tromper ce discriminateur en générant des x˜ le plus réaliste possible. Le moyen

classique de formuler l'interaction entre les deux modèles est à travers un jeu à somme nulle, où chaque joueur maximise ou minimise la récompense v(Gθ, Dϕ). Formellement, nous avons que :

$$\min_{G_\theta} \max_{D_\phi} v(G_\theta, D_\phi) = \min_{G_\theta} \max_{D_\phi} \mathbb{E}_{x \sim p_{data}}[\log D_\phi(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(z)}[\log(1 - D_\phi(G_\theta(\mathbf{z}))]$$

Où généralement la variable latente z < N(z; 0, 1). L'entraînement du GAN alterne donc entre la mise à jour du discriminateur et du générateur comme montre la figure ci-dessous.

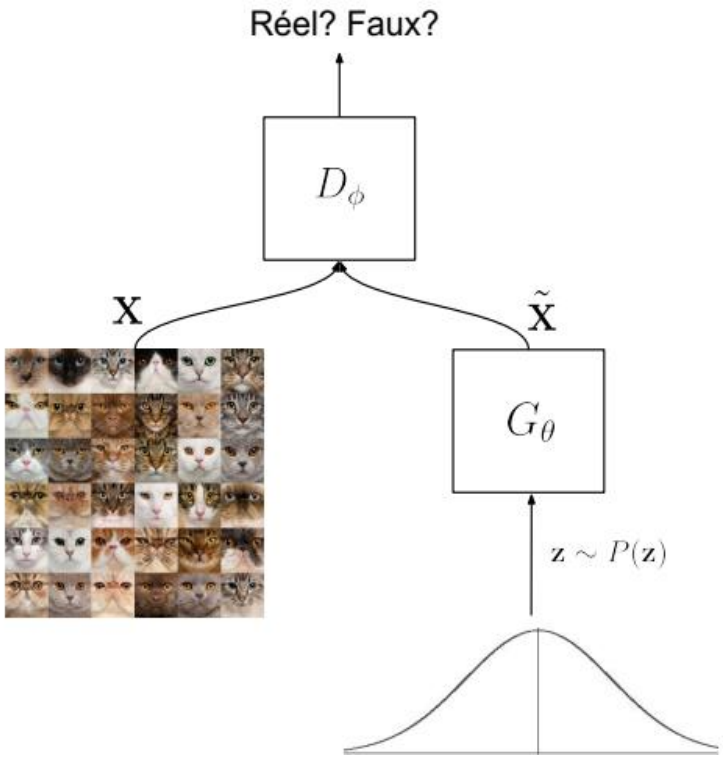


Figure 4: Exemple de GAN: Le générateur produit des images de chats et le discriminateur détermine si elles sont réalistes ou non

Générateur cherche à produire des images quelconques ressemblant à celles d'un jeu de données. Cette méthode ne cherche pas à modéliser explicitement la densité (manifold), c'est une approche inspirée de la théorie des jeux :jeu minimax à 2 joueurs telle que:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

L'apprentissage profond fait partie des méthodes d'apprentissage automatique. Cette dernière motivée par la réalisation de tâches difficiles à définir de manière exhaustive ou par

des règles simples dans des programmes classiques. Par exemple, développer une intelligence artificielle (IA) pour jouer au jeu de Go en respectant les règles du jeu est relativement simple à programmer car l'ensemble des règles peut facilement être défini (Chaque joueur joue un seul pion à son tour, il ne peut poser son pion que sur une case valide, etc). Cependant, faire jouer l'IA de manière optimale afin de remporter la victoire est impossible à définir simplement. Ceci est dû au fait que le jeu de Go n'a pas de stratégie optimale connue, ce qui ne permet pas de créer une succession de règles à suivre pour l'IA afin de gagner. De plus, le nombre d'états possibles du jeu ainsi que les possibilités à chaque état sont tellement importants qu'il est impossible de tout décrire dans un programme classique pour une IA.

Le discriminateur D (sortie 0 à 1) change ses poids afin de maximiser V, c'est -à-dire:

$$\min_G \max_D V(D,G) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}}_{\text{Vraies données}}[\log \overbrace{D(\boldsymbol{x})}^{\uparrow 1}] + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}}_{\text{Fausses données}}[\log(1 - \overbrace{D(G(\boldsymbol{z}))}^{\downarrow 0})]$$

Le générateur G (sortie image) cherche à confondre le discriminateur D, afin de minimiser V, soit:

$$\min_G \max_D V(D,G) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}}_{\text{Vraies données}}[\log \overbrace{D(\boldsymbol{x})}^{\downarrow 0}] + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}}_{\text{Fausses données}}[\log(1 - \overbrace{D(G(\boldsymbol{z}))}^{\uparrow 1})]$$

Pour cette fonction objective V, il s'agit de:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

C'est une alternance entre :

- la montée du gradient pour le discriminateur:

$$\max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- et la descente du gradient pour le générateur, c'est-à-dire:

$$\min_G \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Cette fonction de perte de G est peu commode avec

$$\min_G \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

L'algorithme GAN se présente sous cette forme, Figure 5:

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Figure 5: principe de l'algorithme GAN

3.2 Méthode Pseudo-étiquette :

La méthode d'apprentissage semi-supervisé simple et efficace pour les réseaux de neurones profonds. Proposée pour la première fois par Lee en 2013 [6], la méthode de pseudo-étiquetage utilise un petit ensemble de données étiquetées avec une grande quantité de données non étiquetées pour améliorer les performances d'un modèle. La technique elle-même est incroyablement simple et ne suit que 4 étapes de base :

1. Former le modèle sur un lot de données étiquetées

2. Utilisez le modèle entraîné pour prédire les étiquettes sur un lot de données non étiquetées

3. Utilisez les étiquettes prévues pour calculer la perte sur les données non étiquetées

4. Combinez perte étiquetée avec perte non étiquetée et rétropropagation … et répétez.

Le pseudo-étiquetage forme le réseau avec des données étiquetées et non étiquetées simultanément dans chaque lot. Cela signifie que pour chaque lot de données étiquetées et non étiquetées, la boucle d'apprentissage fait :

1. Un seul passage avant sur le lot étiqueté pour calculer la perte → Ceci est la perte étiquetée

2. Un passage avant sur le lot non étiqueté pour prédire les « pseudo étiquettes » pour le lot non étiqueté

3. Utilisez cette « pseudo étiquette » pour calculer la perte non étiquetée.

Maintenant, au lieu d'ajouter simplement la perte non étiquetée avec la perte étiquetée, Lee propose d'utiliser des poids. La fonction de perte globale ressemble à ceci :

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} L(y_i^m, f_i^m) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} L(y_i'^m, f_i'^m)$$

Dans l'équation, le poids (alpha) est utilisé pour contrôler la contribution des données non étiquetées à la perte globale. De plus, le poids est fonction du temps (époques) et augmente lentement pendant l'entraînement. Cela permet au modèle de se concentrer davantage sur les données étiquetées au départ lorsque les performances du classificateur peuvent être mauvaises. Au fur et à mesure que les performances du modèle augmentent au fil du temps (époques), le poids augmente et la perte sans étiquette met d'avantage l'accent sur la perte globale. Lee propose d'utiliser l'équation suivante pour alpha (t):

$$\alpha(t) = \begin{cases} 0 & t < T_1 \\ \frac{t-T_1}{T_2-T_1}\alpha_f & T_1 \le t < T_2 \\ \alpha_f & T_2 \le t \end{cases}$$

où alpha_f = 3, T1 = 100 et T2 = 600. Ce sont tous des hyperparamètres qui changent en fonction du modèle et de l'ensemble de données.

L'organigramme de la méthode est donné par la figure 6.

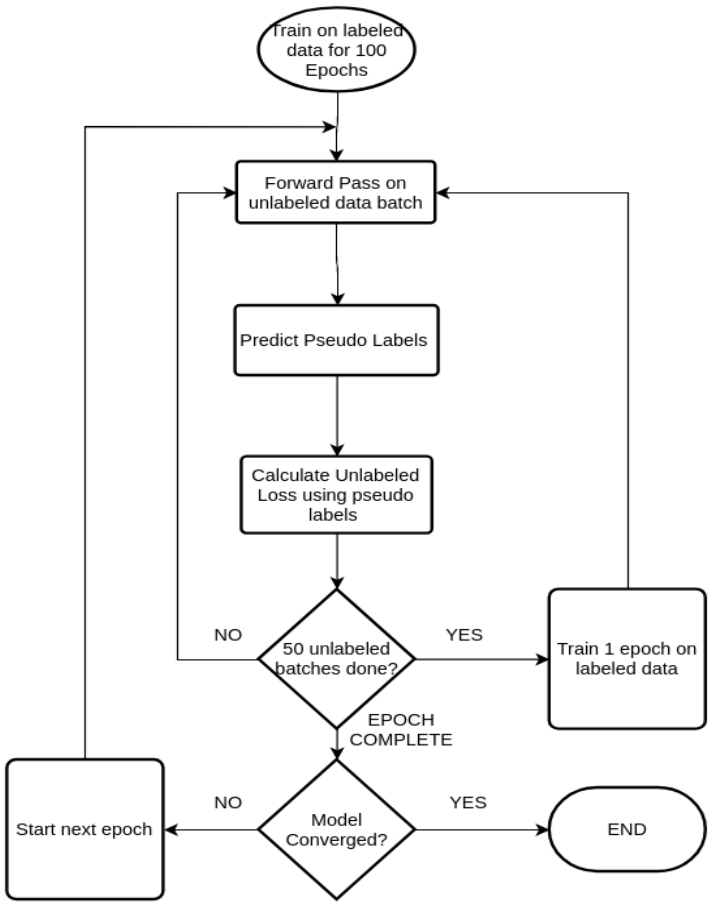


Figure 6: L'organigramme de la méthode Pseudo-étiquette

La Méthode Pseudo-étiquette est très utilisée dans la visualisation des clusters du jeu de données MNIST (voir figure 7).

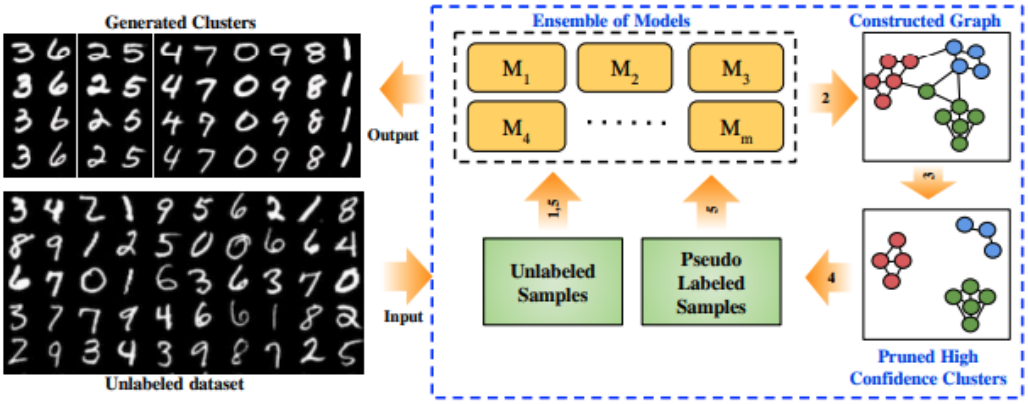


Figure 7: Visualisation des clusters du jeu de données MNIST

Dans cette étude, nous introduisant la méthode pseudo-étiquette par un réseau de neurones de manière semi-supervisée. Fondamentalement, le réseau proposé est formé de manière supervisée avec des données étiquetées et non étiquetées simultanément. Pour les données non étiquetées, les pseudo-étiquettes, prenant simplement la classe qui a la probabilité maximale prévue à chaque mise à jour des poids, sont utilisées comme s'il s'agissait de vraies étiquettes. En principe, cette méthode peut combiner presque tous les modèles de réseaux neuronaux et les méthodes de formation. Cette méthode équivaut en fait à la régularisation de l'entropie. Le conditionnel l'entropie des probabilités de classe peut être utilisée pour mesurer le chevauchement de classes. En minimisant l'entropie pour les données non étiquetées, le chevauchement de la distribution de probabilité de classe peut être réduit. Elle favorise une séparation de faible densité entre les classes, un préalable communément supposé pour l'apprentissage semi-supervisé. Plusieurs expériences sur le jeu de données MNIST bien connu prouvent que la méthode proposée montre les performances de pointe.

4. Baseline (réseaux sur uniquement 100 exemples) et Résultats Obtenues :

La première méthode utilisé SGAN (code1 annexe) :

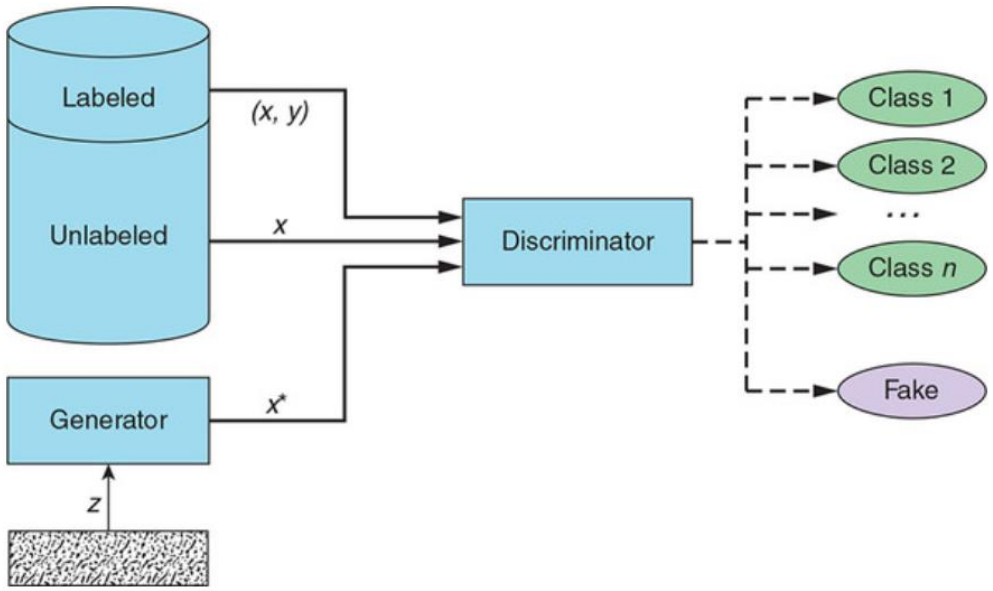
L'apprentissage semi-supervisé est l'un des domaines d'application pratique les plus prometteurs des GAN. Contrairement à l'apprentissage supervisé, dans lequel nous avons besoin d'une étiquette pour chaque exemple de notre ensemble de données, et à l'apprentissage non supervisé, dans lequel aucune étiquette n'est utilisée, l'apprentissage semi-supervisé a une étiquette de classe pour seulement un petit sous-ensemble de l'ensemble de données d'entraînement. En internalisant les structures cachées dans les données, l'apprentissage semi-supervisé s'efforce de généraliser à partir du petit sous-ensemble de points de données étiquetés pour classer efficacement de nouveaux exemples inédits. Il est important de noter que pour que l'apprentissage semi-supervisé fonctionne, les données étiquetées et non étiquetées doivent provenir de la même distribution sous-jacente.

Le manque d'ensembles de données étiquetés est l'un des principaux goulots d'étranglement dans la recherche sur l'apprentissage automatique et les applications pratiques. Bien que les données non étiquetées soient abondantes (Internet est une source pratiquement illimitée d'images, de vidéos et de textes non étiquetés), leur attribuer des étiquettes de classe est souvent prohibitif, peu pratique et prend du temps. Il a fallu deux ans et demi pour annoter manuellement les 3,2 millions d'images originales dans Image Net une base de données d'images étiquetées qui a permis de nombreuses avancées en matière de traitement d'images et de vision par ordinateur au cours de la dernière décennie.

Le GAN semi-supervisé (SGAN) est un réseau conflictuel génératif dont le discriminateur est un classificateur multi classe. Au lieu de distinguer seulement deux classes (réelle et fausse), il apprend à faire la distinction entre N + 1 classes, où N est le nombre de classes dans l'ensemble de données d'entraînement, avec un ajouté pour les faux exemples produits par le générateur.

Par exemple, l'ensemble de données MNIST de chiffres manuscrits a 10 étiquettes (une étiquette pour chaque chiffre, de 0 à 9), de sorte que le discriminateur SGAN formé sur cet ensemble de données prédirait entre 10 + 1 = 11 classes. Dans notre implémentation, la sortie du discriminateur SGAN sera représentée comme un vecteur de 10 probabilités de classe (qui totalisent 1,0) plus une autre probabilité qui représente si l'image est réelle ou fausse.

Faire passer le discriminateur d'un classificateur binaire à un classificateur multi classe peut sembler un changement trivial, mais ses implications sont plus profondes qu'il n'y paraît à première vue. Commençons par un diagramme. Ce qui suit montre l'architecture SGAN.



La deuxième méthode utilisé (code2 annexe) :

Pour réaliser la prédiction des données MNIST avec 100 données label, nous avons commencé par sélectionner 10 labels de chaque classe et entrainer un réseau de convolution sur les 100 données label. Avec ce modèle, nous avons réalisé une prédiction sur les données non labelisé par groupe de 2000 échantillons.

A chaque fois que nous avons fait une prédiction sur un groupe d'échantillons, nous ajouton le résultat a la liste des donnée labelisé et on recommence le processus avec un autre groupe d'échantillons.

De ce fait, notre model aura été entrainé par les 100 donnée labelisé et les 59900 données prédit par le model.

La figure 8 résume le principe du code de la 2éme méthode :

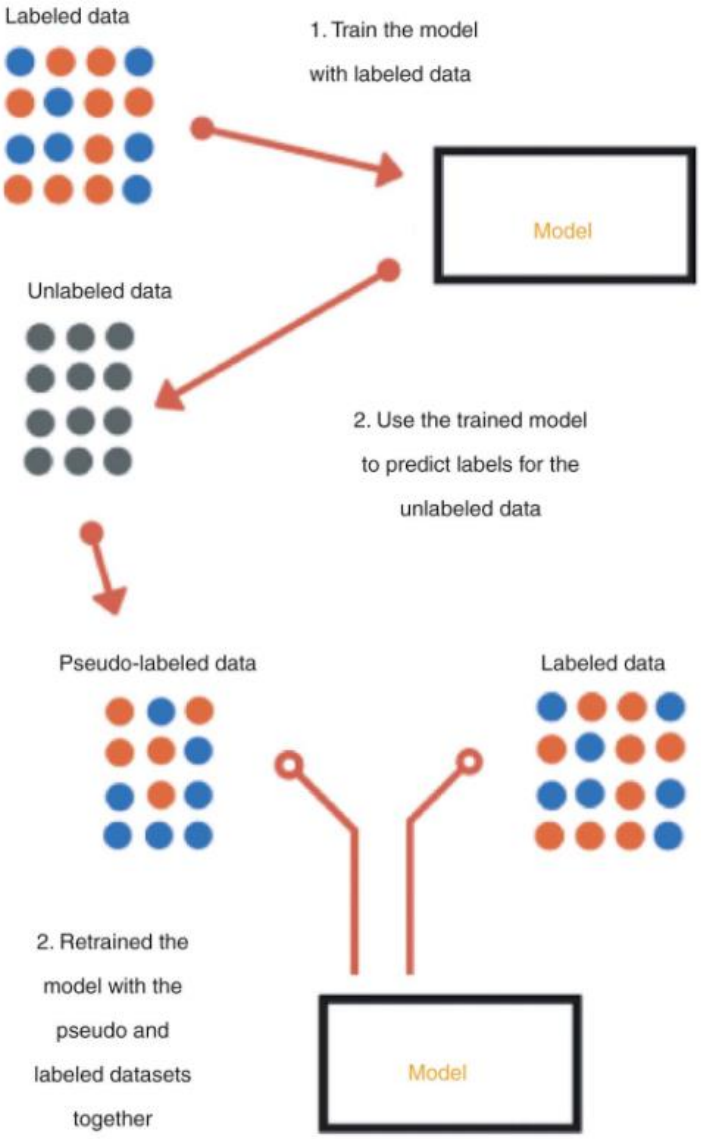


Figure8

Les résultats obtenus et les Baseline sont presque pareil pour les deux méthodes utilisées.

Pour la première méthode SGAN on a eu 68% de précision sur les Baseline et 70% de précision sur le model.

Pour la deuxième méthode on a eu 78% de précision sur les Baseline et 81% de précision sur le model.

Les 100 données MNIST labélisé sont pris aléatoirement cela peut avoir un effet sur notre model, les résultats du model sont dépondant du nombre de données labélisé les nombre de données labélisé pour chaque classe.

6. Conclusion

Les réseaux GAN ainsi que la méthode pseudo-étiquette sont des méthodes génératives les plus prometteuses dans diverses applications. Dans l'ensemble, le GAN est une idée très intéressante et c'est la raison pour laquelle de nombreux chercheurs y travaillent et proposent divers modèles basés sur le GAN.

Dans ce projet, nous avons effectué utiliser les réseaux GAN pour un exemple de prédiction des données MNIST avec uniquement 100 labels et montré à quel point ils sont utiles dans la synthèse d'images.

Nous n'avons touché que quelques variantes populaires de GAN pour la synthèse d'images. On peut voir que les dernières variantes de GAN sont non supervisées et présentent des résultats acceptables.

Quant à la méthode pseudo-étiquette, elle peut aider sur l'ensemble de données MNIST en affinant le modèle mais les résultats sont faits avec seulement 100 exemples étiquetés car le pseudo-étiquetage est sensible aux prédictions initiales et que de bonnes prédictions initiales nécessitent un nombre suffisant de points étiquetés.

7. Références

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David WardeFarley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680, 2014.

[2] Augustus ODENA, Christopher OLAH et Jonathon SHLENS, « Conditional Image Synthesis with Auxiliary Classifier GANs », in : Proceedings of the 34th International Conference on Machine Learning, sous la dir. de Doina PRECUP et Yee Whye TEH, t. 70, Proceedings of Machine Learning Research, International Convention Centre, Sydney, Australia : PMLR, 2017, p. 2642-2651.

[3] S. REED et al., « Generative Adversarial Text to Image Synthesis », in: ArXiv e-prints (mai 2016), arXiv : 1605.05396.

[4] C. LEDIG et al., « Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network », in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, p. 105-114, DOI : 10.1109/CVPR.2017. 19.

[5] J. ZHU et al., « Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks », in : 2017 IEEE International Conference on Computer Vision (ICCV), 2017, p. 2242-2251, DOI : 10.1109/ICCV.2017.244.

8.Code commenté en annexes

Code 1 SGAN:

```
# -*- coding: utf-8 -*-
"""Untitled0.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1wIkx_Vhv3Ksd6wCiWzmpJZa_iBCMuH-1

# ***Importation de bibliothèques***
"""

# Commented out IPython magic to ensure Python compatibility.
from keras import backend as K

from keras.datasets import mnist
from keras.layers import Input, Dense, Reshape, Flatten, Dropout, Lambda
from keras.layers import BatchNormalization, Activation
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.models import Model, Sequential
from keras.optimizers import Adam
from keras.utils import to_categorical

import matplotlib.pyplot as plt
import numpy as np

import sys

# %matplotlib inline
print('done importing libraries')

"""# ***Les dimensions des images cibles***"""

img_rows = 28
img_cols = 28
channels = 1
img_shape = (img_rows, img_cols, channels)
z_dim = 100 #random noise input for generator
num_classes = 10  #no. of classes to predict for semi supervised algo


print('done defining variables')
```

```
"""# **Le jeu de données MNIST **"""

def load_mnist_data():

    return mnist.load_data()

class Dataset:
    def __init__(self, num_labeled):
        self.num_labeled = num_labeled
        (self.x_train, self.y_train), (self.x_test, self.y_test) =
load_mnist_data()

        def preprocess_images(x):
            x = (x.astype(np.float32) - 127.5) / 127.5
            print(x.shape)
            x = np.expand_dims(x, axis=3)
            print(x.shape)

            return x
        def preprocess_labels(y):
            return y.reshape(-1, 1)

        self.x_train = preprocess_images(self.x_train)
        self.x_test = preprocess_images(self.x_test)

        self.y_train = preprocess_labels(self.y_train)
        self.y_test = preprocess_labels(self.y_test)

    def batch_labeled(self, batch_size):
        idx = np.random.randint(0, self.num_labeled, size=batch_size)
        images = self.x_train[idx]
        labels = self.y_train[idx]

        return images, labels

    def batch_unlabeled(self, batch_size):
        idx = np.random.randint(self.num_labeled, self.x_train.shape[0],
batch_size)

        return self.x_train[idx]

    def training_set(self):
        x_train = self.x_train[:self.num_labeled]
        y_train = self.y_train[:self.num_labeled]

        return x_train, y_train

    def test_set(self):

        return self.x_test, self.y_test

print('done defining the dataset class')

num_labeled = 100
dataset = Dataset(num_labeled)

print('done instantiating the dataset class')
```

"""# ***Le générateur***"""

```
#the Generator
def build_generator(z_dim):

    model = Sequential()

    model.add(Dense(256*7*7, input_dim=z_dim))
    model.add(Reshape((7, 7, 256)))

    # 7*7*256 -> 14*14*256
    model.add(Conv2DTranspose(128, kernel_size=3, strides=2, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))

    # 14*14*128 -> 14*14*64
    model.add(Conv2DTranspose(64, kernel_size=3, strides=1, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))

    # 14*14*64 => 28*28*1
    model.add(Conv2DTranspose(1, kernel_size=3, strides=2, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))

    model.add(Activation('tanh'))

    z = Input(shape=(z_dim,))
    img = model(z)

    return Model(z, img)
```

"""### ***Le discriminateur de base ***"""

```
#The Core Discriminator
def build_discriminator(img_shape):

    model = Sequential()

    # 28*28*1 => 14*14*32
    model.add(Conv2D(32, kernel_size=3, strides=2, input_shape=img_shape, pad-
ding='same'))
    model.add(LeakyReLU(alpha=0.01))

    # 14*14*32 => 7*7*64
    model.add(Conv2D(64, kernel_size=3, strides=2, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))

    # 7*7*64 => 3*3*128
    model.add(Conv2D(128, kernel_size=3, strides=2, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))

    model.add(Dropout(rate=0.5))
    model.add(Flatten())
```

```
    model.add(Dense(num_classes))

    return model
```

"""# ***Le discriminateur supervisé***"""

```
def build_discriminator_supervised(discriminator_net):

    model = Sequential()

    model.add(discriminator_net)
    model.add(Activation('softmax'))

    return model
```

"""# ***Le discriminateur non supervisé***"""

```
def build_discriminator_unsupervised(discriminator_net):

    model = Sequential()

    model.add(discriminator_net)

    def predict(x):
        prediction = 1.0 - (1.0 / (K.sum(K.exp(x), axis=-1, keepdims=True) + 1.0))

        return prediction

    model.add(Lambda(predict))

    return model
```

"""# ***Construction et compilation du modèle***"""

```
#Building and Compiling the Model
disc = build_discriminator(img_shape)

sup_disc = build_discriminator_supervised(disc)
sup_disc.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimiz-
er=Adam())

unsup_disc = build_discriminator_unsupervised(disc)
unsup_disc.trainable = False
unsup_disc.compile(loss='binary_crossentropy', metrics=['accuracy'], optimiz-
er=Adam())

gen = build_generator(z_dim)

def combined(generator, discriminator):

    model = Sequential()

    model.add(generator)
    model.add(discriminator)

    return model
```

```
sgan = combined(gen, unsup_disc)
sgan.compile(loss='binary_crossentropy', optimizer=Adam())
```

```
"""# ***Training***"""
```

```
d_accuracies = []
d_losses = []

def train(iterations, batch_size, sample_interval):
    real = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))

    for iteration in range(iterations):
        imgs, labels = dataset.batch_labeled(batch_size)
        labels = to_categorical(labels, num_classes=num_classes)

        imgs_unlabeled = dataset.batch_unlabeled(batch_size)

        z = np.random.normal(0, 1, (batch_size, z_dim))
        gen_imgs = gen.predict(z)

        d_loss_supervised, accuracy = sup_disc.train_on_batch(imgs, labels)
        d_loss_real = unsup_disc.train_on_batch(imgs_unlabeled, real)
        d_loss_fake = unsup_disc.train_on_batch(gen_imgs, fake)

        d_loss_unsupervised = 0.5 * np.add(d_loss_real, d_loss_fake)

        z = np.random.normal(0, 1, (batch_size, z_dim))
        gen_imgs = gen.predict(z)

        g_loss = sgan.train_on_batch(z, real)

        d_losses.append(d_loss_supervised)
        d_accuracies.append(accuracy)

        if iteration % sample_interval == 0:
            print('{} [D loss supervised: {:.4f}, acc: {:.2f}]'.format(iteration,
d_loss_supervised, 100 * accuracy))

iterations = 8000
batch_size = 32
sample_interval = 800

train(iterations, batch_size, sample_interval)
```

```
"""# **Évaluation du discriminateurs supervisée**"""
```

```
x, y = dataset.test_set()
y = to_categorical(y, num_classes)

_, accuracy = sup_disc.evaluate(x, y)
print('Test accuracty = {:.2f}%'.format(accuracy * 100))
```

```
"""# ***Comparaison avec un classificateur
```

```
entier ***"""

base_disc = build_discriminator(img_shape)
mnist_classifier = build_discriminator_supervised(base_disc)
mnist_classifier.compile(loss='categorical_crossentropy', metrics=['accuracy'],
optimizer=Adam())
for i in range(100):
    x, y = dataset.batch_labeled(batch_size)
    y = to_categorical(y, num_classes=num_classes)
    sup_loss, sup_acc = mnist_classifier.train_on_batch(x, y)

    if i % 20 == 0:
        print('iteration = {} / loss = {:.4f} / accuracy = {:.2f}'.format(i,
sup_loss, sup_acc * 100))
```

```
"""# ***##Evaluation du classificateur entier***"""

x, y = dataset.test_set()
y = to_categorical(y, num_classes)

_, accuracy = mnist_classifier.evaluate(x, y)
print('Test accuracty = {:.2f}%'.format(accuracy * 100))
```

Code du 2ème méthode:

```
# -*- coding: utf-8 -*-
"""TPDeepL.ipynb

Automatically generated by Colaboratory.

Original file is located at
   https://colab.research.google.com/drive/1Za7vsyfnS2sJvXTvCMwM8WITL5DpX7FN
"""

from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D,BatchNormalization
from keras import backend as K
import numpy as np
import random
from sklearn.metrics import confusion_matrix

"""chargement du dataset et pretraitement"""

num_classes = 10
```

```
# input image dimensions
img_rows, img_cols = 28, 28

num_train_samples = 100
samples_per_class = int(num_train_samples/9)

(x_train, y_train), (x_test, y_test) = mnist.load_data()
if K.image_data_format() == 'channels_first':
  x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
  x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
  input_shape = (1, img_rows, img_cols)
else:
  x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
  x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
  input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y=y_train.tolist()
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# only select 100 training samples
#idxs_annot = list(range(x_train.shape[0]))
#random.seed(0)
#random.shuffle(idxs_annot)
#idxs_annot = idxs_annot[ :100 ]

idxs_annot = list()
for j in range(10):
  for i in range(10):
    ind = y.index(j)
    y[ind]=None
    idxs_annot.append(ind)

x_train_unlabeled=x_train
x_train_labeled = x_train[ idxs_annot ]
```

```
for i in idxs_annot :
  x_train_unlabeled = np.delete(x_train_unlabeled,i,axis=0)


print(x_train_labeled.shape, 'x train label')
print(y_train_labeled.shape, 'y tain label')
print(x_train_unlabeled.shape, 'x train unlabel')

"""Baseline du model avec 100 données MNIST labelisé"""

epochs = 1000
batch_size = 128

model = Sequential()
model.add(Conv2D(30, (5, 5), input_shape=input_shape, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(15, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adam',
metrics=['accuracy'])

model.fit(x_train_labeled, y_train_labeled,
batch_size=batch_size,
epochs=epochs)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

"""entrainement du model sur les données non labelisé"""

new_x_train = x_train_labeled
new_y_train = y_train_labeled
listPred = list()
start = 0
for i in range(x_train_unlabeled.shape[0]):
  if i%2000 == 0 and i!=0:
    predict = keras.utils.to_categorical(listPred, num_classes)
    new_x_train = np.concatenate((new_x_train, x_train_unlabeled[start:i]), axis=0)
    new_y_train = np.concatenate((new_y_train, predict), axis=0)
    start = i
    listPred = list()
```

```
   model.fit(new_x_train, new_y_train,batch_size=128,epochs=10)
 x_pred = x_train_unlabeled[i].reshape(1, img_rows, img_cols, 1)
 listPred.append(model.predict_classes(x_pred))

predict = keras.utils.to_categorical(listPred, num_classes)
new_x_train = np.concatenate((new_x_train, x_train_unlabeled[start:]), axis=0)
new_y_train = np.concatenate((new_y_train, predict), axis=0)

"""construction du nouveau model sur les données labelisé + non labélisé predit"""

epochs = 10
batch_size = 128
model = Sequential()
model.add(Conv2D(30, (5, 5), input_shape=input_shape, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(15, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adam',
metrics=['accuracy'])

model.fit(new_x_train, new_y_train,
batch_size=batch_size,
epochs=epochs,
verbose=1,)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```