Équipe du projet
Nourhene Chaalia
Camille Méketyn
Jules Salzinger
École Centrale Marseille

Rapport du projet S8

Utilisation de modèles génératifs pour la classification ouverte

Tuteur : M. Artieres

Sommaire

Introduction générale **3**

Problématique **4**

1 Présentation de la solution apportée **6**
1.1 Le classifieur 6
1.2 Le réjecteur 7
1.3 Le GAN 8
1.4 La base de données d'entraînement 10
1.4.1 Base de données pour un entraînement classique 10
1.4.2 Base de données pour un entraînement progressif 11
1.4.3 Autres stratégies génératives 11

2 Description technique de l'implémentation **11**
2.1 Le classifieur 12
2.1.1 Architecture 12
2.1.2 Entraînement 12
2.2 Le réjecteur 13
2.2.1 Architecture 13
2.2.2 Entraînement 13
2.3 Le GAN 14
2.3.1 Architectures 14
2.3.1.1 Le générateur 14
2.3.1.2 Le discriminateur 15
2.3.2 Entraînement 15

3 Résultats des tests et interprétations **15**
3.1 Seuillage des probabilités et de l'entropie de Shannon 16
3.2 Entraînement et résultats du GAN 17
3.2.1 Autres méthodes 19
3.3 Entraînement du réjecteur et tests du classifieur 19
3.3.1 Résultats avec un seul seuil du discriminateur 19
3.3.2 Comparaison des différents seuils du discriminateur 21
3.4 Interprétation des résultats 23

Conclusion **24**

Bibliographie **25**

Introduction générale

L'une des problématiques les plus importantes dans le développement de l'intelligence artificielle a toujours été la résolution des problèmes de classification. Par exemple, un dispositif de collecte des déchets doit pouvoir trier entre différents matériaux, une voiture intelligente doit interpréter les différents signaux envoyés par ses capteurs pour reconnaître des objets, et un chatbot doit trouver le sens des mots employés dans une phrase. Pour cette raison, de nombreux algorithmes et réseaux neuronaux ont été développés dans le but de parvenir à une classification toujours plus précise, et beaucoup de moyens financiers sont déployés pour améliorer leurs performances : il s'agit d'une problématique à la fois importante et actuelle. Quand on parle de classification, la plupart du temps, on fait référence à la classification fermée, c'est à dire que l'on associe à des données d'entrée une classe parmi n, en ayant pour ce faire une base de données contenant des images de chacune des n classes et à partir de laquelle on "apprend" à reconnaître chaque classe. Mais que se passe-t-il quand le modèle rencontre une entrée n'appartenant à aucune des classes envisagées par la base de données d'apprentissage ? Le plus souvent, les modèles de classification fermée lui attribuent la classe la plus proche d'après leurs critères, ce qui peut mener à des erreurs lourdes de conséquences. Dans ce rapport, nous nous intéresserons à un autre type de classification, en pratique beaucoup plus puissant mais à ce jour peu exploré : la classification ouverte. La classification ouverte consiste, pour un jeu de n classes, à classer les entrées entre $n + 1$ classes : les n classes de la base d'apprentissage et la classe "autre", signifiant que le modèle n'a pas reconnu l'objet. Bien que difficiles à mettre en oeuvre, ces solutions présentent des atouts considérables lors de la prise de décision d'un système automatisé intelligent.

Remerciement

Nous tenons à remercier notre tuteur projet Thierry Artieres pour l'élaboration du sujet, toute l'aide qu'il nous a fournie et la grande quantité de code qu'il nous a donnée pour démarrer le projet dans les meilleures conditions.

Problématique et état de l'art

Le domaine de la classification ouverte étant très vaste, il est intéressant de dire ici quelques mots sur les différentes problématiques liées à la classification ouverte avant de préciser celle sur laquelle portera ce rapport. Le point de départ de ces problématiques est le suivant : on peut établir un modèle statistique pour reconnaître une classe donnée à partir d'exemples, mais comment entraîner un modèle à refuser "le reste" ? Le point fondamental est qu'on peut fournir des exemples de tout ce qu'est une chose, mais pas de tout ce qu'elle n'est pas. Quand bien même cela serait possible, l'objectif ne serait pas là : le cerveau humain, par exemple, n'a pas besoin de savoir qu'un lama n'est pas une pomme pour savoir en voyant un lama pour la première fois de sa vie qu'il n'a aucune chance d'être une pomme. Il faut donc trouver un autre moyen de qualifier la non-appartenance à une classe, et pour ce faire différentes approches ont déjà été proposées :

- quand le problème contient un grand nombre de classes, on peut appliquer l'algorithme OpenMax sur les deux dernières couches d'un réseau neuronal de classification. Celles-ci sont le plus souvent :
  - une couche correspondant à l'activation sortant du réseau pour chacune des classes
  - une couche réalisant la fonction softmax sur ces activations pour en déduire la classification

  Cette première méthode (Abhijit Bendale, Terrance E. Boult, 2015) propose de remplacer la fonction softmax par une reconnaissance des classes en fonction de la distribution des activations sur toute la couche : l'algorithme OpenMax. Ainsi, la classe n est reconnue non plus si l'activation du n^{eme} neurone est plus forte que les autres, mais si la distribution de toutes les activations de la pénultième couche correspond à une distribution caractéristique de cette classe. Certes, cette possibilité n'est pas une réponse parfaite, mais elle offre une solution dont les performances augmentent avec le nombre de classes et qui présente de très bons résultats ;
- une deuxième possibilité consiste à analyser la confiance du réseau dans son résultat. Si les résultats sont attendus entre 0 et 1 par exemple, il est intéressant de se demander si une image totalement différente serait placée peut-être dans la même classe, mais avec un score inférieur. A en juger par les résultats présentés par Lei Shu et al., 2017, cette méthode montre de bons résultats au moins dans le domaine de la classification de textes. Elle est équivalente à un seuillage sur les activations des neurones ;
- de nombreuses améliorations des deux précédentes solutions existent, mais on peut en citer citer notamment deux. Tout d'abord, un ajout intéressant à OpenMax

(Zongyuan Ge et al., 2017) est d'approcher la distribution théorique recherchée en générant des exemples de synthèse à l'aide d'un GAN (Generative Adversarial Network). Ensuite, on peut essayer d'améliorer le simple seuillage sur les activations du softmax par un seuillage sur les entropies de ces activations, afin de prendre en compte les importances relatives de chaque activation dans le résultat.

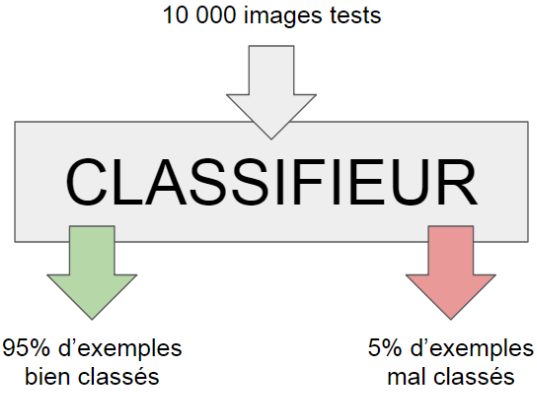
Dans ce rapport, la problématique étudiée est la suivante : on considère un classifieur entraîné sur la base de données d'entraînement de MNIST. Lorsque l'on le teste sur la base de test de MNIST, il commet des erreurs de classification. On fait l'hypothèse que les erreurs de classification proviennent du fait que les images sur lesquelles les erreurs sont commises sont plus éloignées que les autres de la distribution moyenne des exemples de leur classe réelle. Ainsi, en entraînant un réseau dit "réjecteur" à donner la probabilité qu'une image appartienne à MNIST, et donc en faisant de la classification ouverte sur MNIST, on pourrait améliorer les performances du classifieur en retirant de la base de test les exemples ayant le score le plus bas pour le réjecteur. Il est évident que la partie la plus difficile consiste à déterminer comment entraîner correctement le réjecteur, et c'est ce que nous verrons dans ce rapport.

1 Présentation de la solution apportée

Afin de répondre à la problématique posée, nous suivons une méthode de résolution dont les grandes lignes ont été définies par notre tuteur de projet.

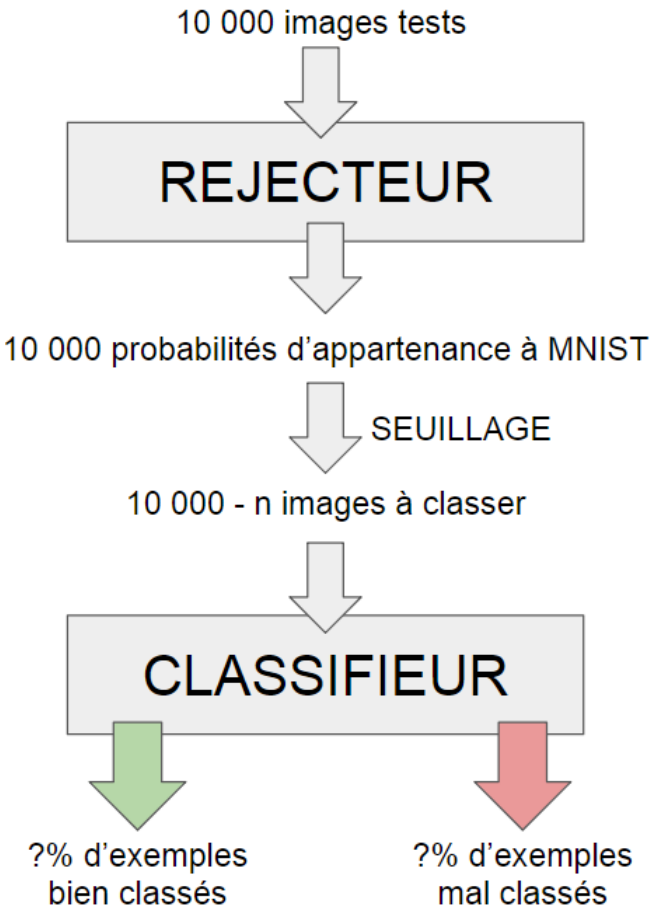
1.1 Le classifieur

Le point de départ de notre problématique est un classifieur qui opère une classification fermée sur la base de données MNIST. Ce classifieur, comme tous les modèles que nous utiliserons au cours de ce projet, est un réseau de neurones convolutif simple dont les caractéristiques seront détaillées en partie 2. Ce réseau est important car il nous permettra de mesurer la qualité des différentes méthodes que nous utiliserons. Pour cette raison, nous l'entraînons volontairement sur très peu d'exemples de façon à obtenir une précision de 95% (95.54 ± 0.01). Cette précision assez basse nous permettra de voir de façon plus claire et plus significative les améliorations apportées à la classification par le pré-filtrage effectué par le réseau réjecteur.

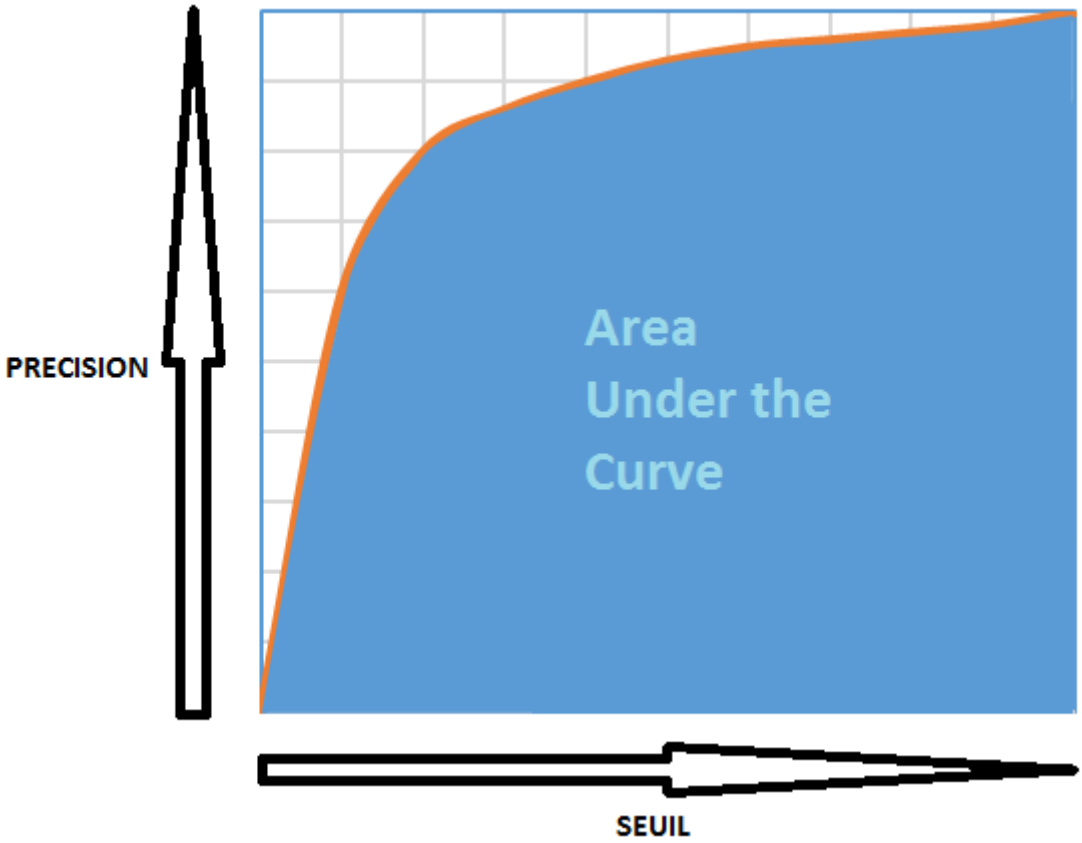


1.2 Le réjecteur

Comme on l'a vu dans la partie précédente, on cherche à entraîner un réseau neuronal que l'on appellera "réjecteur". Ce réseau aura pour but de renvoyer la probabilité qu'une image d'entrée fasse partie de la base de données MNIST. Plus la probabilité qu'elle en fasse partie est élevée et plus sa distribution correspond à la distribution moyenne des éléments des classes de MNIST, par conséquent plus cette probabilité est élevée et plus il est censé être simple pour le classifieur de reconnaître l'image.



Avec ce schéma de fonctionnement, on voit bien que le taux de précision du classifieur dépend du seuil choisi pour le réjecteur. Notre objectif est donc que le taux de précision augmente aussi vite que possible avec le seuil choisi : on veut obtenir une précision aussi bonne que possible en supprimant aussi peu d'éléments de la base de test que possible. Une bonne métrique de cette efficacité du réjecteur est donc l'aire sous la courbe représentant la précision du classifieur en fonction du seuil choisi. On appelle cette métrique l'AUC (Area Under the Curve).

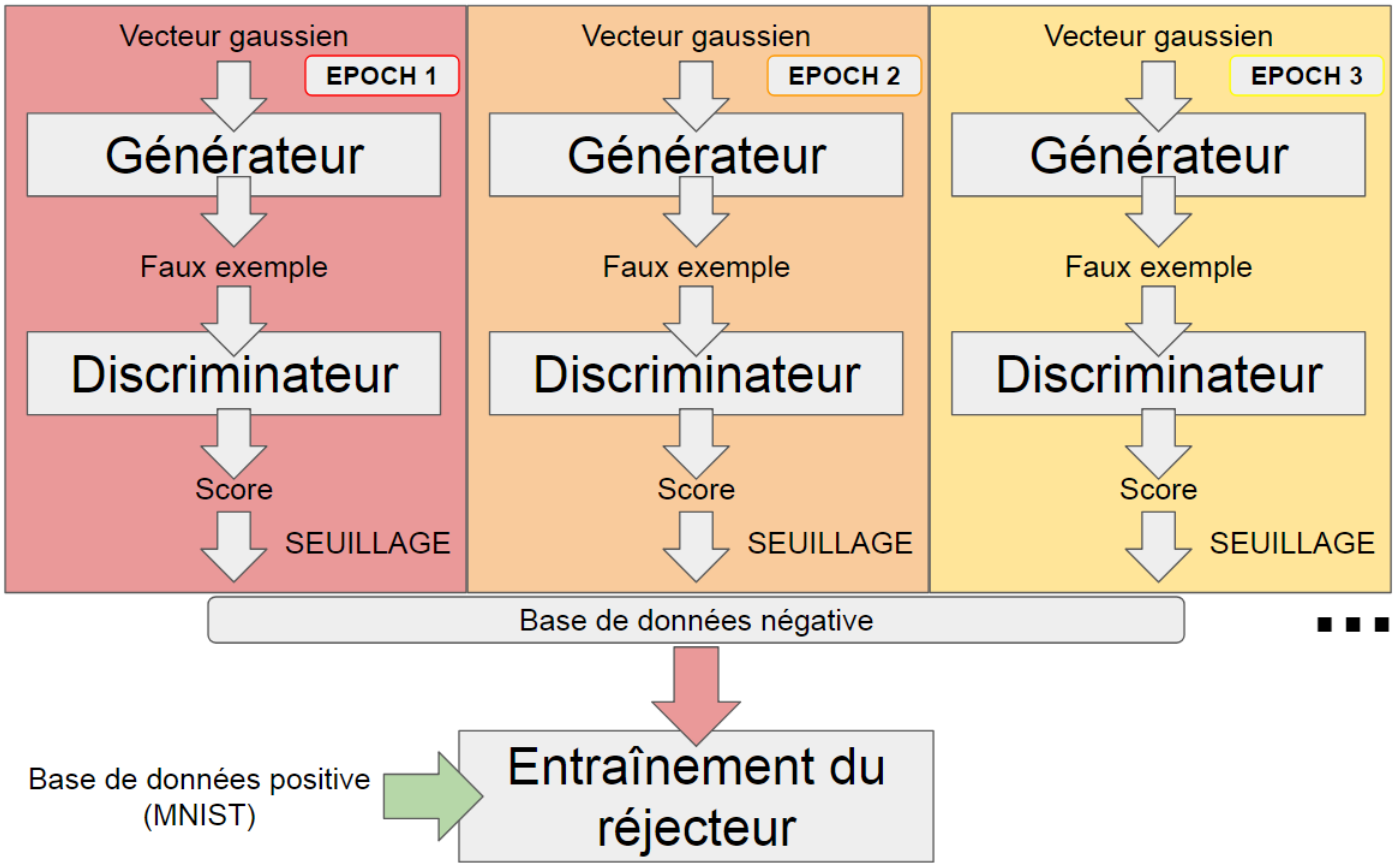


Mais comment entraîner ce réjecteur à accomplir cette tâche ? Pour réaliser l'entraînement, nous avons besoin d'une base de données. Les premiers éléments de cette base seront bien sûr des éléments de MNIST, auxquels sera attribuée la probabilité 1 puisqu'ils font partie de MNIST. Il faut ensuite fournir au réjecteur des exemples ne faisant pas partie de MNIST, et ayant donc la probabilité 0. Ces exemples doivent se rapprocher de la distribution d'un élément de MNIST tout en ayant une distribution différente. Un très bon critère serait : un humain dirait-il que l'image est un chiffre écrit à la main ? Nous cherchons donc à générer la deuxième partie de la base de données qui consiste en des exemples qui se rapprochent de ceux de MNIST sans toutefois en être. Pour cela, nous utiliserons un réseau de type GAN.

1.3 Le GAN

Un GAN, ou Generative Adversarial Network, est un réseau neuronal composé de deux sous-réseaux : un réseau dit "discriminateur" et un réseau dit "générateur". Le réseau générateur prend en argument un vecteur gaussien généré aléatoirement et renvoie une image. Son objectif est que cette image soit aussi fidèle que possible à la distribution

d'une image issue de MNIST. Le réseau discriminateur prend en argument une image et renvoie la probabilité qu'elle soit issue de MNIST. Son objectif est de distinguer les vraies images de MNIST de celles générées par le premier réseau. Les deux réseaux sont entraînés simultanément de sorte qu'au cours de l'entraînement chacun devient de plus en plus performant. Ainsi, à la fin de l'entraînement, le générateur renvoie des images fidèles à MNIST et très difficiles à différentier des images originales. Le discriminateur, lui, a alors une précision proche de 50% car les exemples générés sont devenus quasiment identiques aux exemples initiaux. Notre objectif va donc être de récupérer les exemples générés par le générateur au cours de son entraînement, en commençant au début et en s'arrêtant un petit peu avant la fin. Notre but étant de récupérer des exemples ne faisant à coup sûr pas partie de MNIST, on ne gardera que les exemples pour lesquels le score du discriminateur est suffisamment bas.



Mais cette façon de générer la base de données n'est qu'une possibilité. Sur le conseil de M.Artieres, nous proposons plusieurs protocoles afin d'affiner notre base de données.

1.4 La base de données d'entraînement

Dans cette partie, nous expliquons les différentes méthodes envisagées pour générer la base de données, ainsi que les raisons qui nous ont poussées à nous tourner vers l'une d'elles en particulier pour nos tests.

1.4.1 Base de données pour un entraînement classique

Dans le cas de ce type d'entraînement, on commence par générer une base de données de taille n, puis on la mélange et on entraîne le réjecteur sur e epochs en lui présentant à chaque fois $\frac{n}{e}$ exemples issus de la base de données et $\frac{n}{e}$ exemples de MNIST. Nous avons à notre disposition plusieurs paramètres à choisir pour la génération de cette base de données.

— Le discriminateur chargé de donner le score de l'exemple généré doit-il être le discriminateur de l'epoch correspondant à l'epoch choisie pour le générateur, ou faut-il plutôt utiliser pour tous les exemples le discriminateur final après toutes ses epochs d'entraînement ?

— Faut-il pondérer les exemples lors de l'entraînement du réjecteur en fonction des scores obtenus par le discriminateur ?

— Faut-il mélanger la base de données ou non ?

— Faut-il privilégier certaines epochs du GAN par rapport aux autres, c'est-à-dire par exemple générer plus d'exemples avec les premières epochs et moins avec les suivantes ?

Nous décidons d'effectuer la majorité de nos tests avec les choix suivants :

— le discriminateur chargé de donner les scores est toujours le dernier, à la fin de l'entraînement du GAN, afin d'être sûr que les exemples sont choisis par un discriminateur abouti et de moins risquer d'intégrer dans la base de données des exemples non-représentatifs ;

— nous ne pondérons pas l'entraînement du réjecteur, car nous estimons qu'il s'agit d'une éventuelle piste supplémentaire à creuser si nous essayons d'améliorer nos résultats mais que ce n'est pas nécessaire dans un premier temps ;

— nous décidons de mélanger la base de données, afin d'éviter les biais liés à l'évolution temporelle du GAN ;

— nous générons le même nombre d'exemples à toutes les epochs, car le seuillage par les scores éliminera automatiquement plus d'exemples provenant des epochs moins intéressantes, et nous estimons que forcer la distribution des provenances des exemples constitue une éventuelle piste supplémentaire inutile à creuser dans un premier temps.

C'est donc ce type d'entraînement que nous utilisons principalement dans la suite. Cette méthode présente l'intérêt majeur d'être la "méthode étalon" la plus basique. Ainsi, toute nouvelle méthode testée par la suite pourra efficacement être comparée à celle-ci pour mettre en évidence lesquels de nos choix étaient bons et lesquels de nos paramètres étaient correctement choisis.

1.4.2 Base de données pour un entraînement progressif

Les solutions présentées dans cette section ont été codées et testées superficiellement, mais nous ne disposons que de peu de données car d'une part ces tests sont très longs et d'autre part nous avons préféré réaliser des tests sur les premiers types de bases de données pour les raisons évoquées ci-dessus. L'idée de base de l'entraînement progressif consiste à envoyer d'abord des exemples plus simples au réjecteur durant l'entraînement, puis des exemples de plus en plus difficiles. Cette méthode est particulièrement adaptée à notre génération de base de données car on peut supposer que plus un exemple est généré tôt dans l'apprentissage et plus il est "simple" à distinguer d'un élément réel de MNIST. Nous disposons une fois encore de multiples options. Les quelques tests que nous réaliserons plus tard sur cette méthode essayeront rapidement chacune de ces options.

— Le discriminateur chargé de donner le score de l'exemple généré doit-il être le discriminateur de l'epoch correspondant à l'epoch choisie pour le générateur, ou faut-il plutôt utiliser pour tous les exemples le discriminateur final après toutes ses epochs d'entraînement ?

— Pour générer la n^{eme} epoch de l'entraînement du réjecteur, on peut soit s'intéresser aux exemples générés par la n^{eme} epoch d'entraînement du GAN, soit prendre toutes les epochs du GAN inférieures à n. Une dernière possibilité est de générer d'abord des exemples avec toutes les epochs du GAN puis de recréer une échelle de complexité grâce aux scores par le discriminateur final (ou du discriminateur correspondant).

1.4.3 Autres stratégies génératives

On aurait également pu s'intéresser à d'autres stratégies génératives pour générer les éléments de notre base de données afin de la diversifier. Par exemple, on aurait pu utiliser plusieurs GAN (un entraîné sur chaque classe de MNIST séparément). Nous avons aussi envisagé à certains stades d'utiliser des GANs avec des architectures différentes, ou encore des architectures similaires avec des initialisations différentes. Nous avons d'ailleurs finalement conservé l'une de ces idées puisque notre base de données est générée à partir de deux architectures de GAN. Nous avons aussi imaginé, sur une suggestion de M. Artieres, des stratégies ne s'appuyant pas sur des GANs (par exemples découper des morceaux d'éléments de MNIST pour en créer des chimères qui y ressemblent mais n'en sont pas). Bien que nous ayons dû faute de temps renoncer à tester ces options, les codes permettant leur mise en application figurent dans notre rendu.

2 Description technique de l'implémentation

2.1 Le classifieur

2.1.1 Architecture

Voici la structure que nous avons utilisée pour le classifieur :

Couche	0	1	2	3	4	5
Type	Input	Conv2D	ReLU	Pool	Conv2D	ReLU
Format entrant	28*28*1	28*28*1	26*26*64	26*26*64	13*13*64	11*11*128
Noyau	-	3*3	-	2*2	3*3	-
Format sortant	28*28*1	26*26*64	26*26*64	13*13*64	11*11*128	11*11*128
Stride	-	1*1	-	2*2	1*1	-
Padding	-	non	-	non	non	-
Couche	6	7	8	9	10	
Type	Pool	Dropout (0.5)	Flatten	Fc (ReLU)	Fc (sigmoid)	
Format entrant	11*11*128	5*5*128	5*5*128	3200	256	
Noyau	2*2	-	-	-	-	
Format sortant	5*5*128	5*5*128	3200	256	10	
Stride	2*2	-	-	-	-	
Padding	non	-	-	-	-	

FIGURE 2.1 – Conv2D : couche de convolution ; ReLU : unité de rectification linéaire ; Pool : max pooling ; Fc (fonction d'activation) : couche densément connectée

2.1.2 Entraînement

Le réseau est entraîné en 50 epochs à partir des 2000 premiers éléments de la base d'entraînement de MNIST, normalisés entre -1 et 1. L'algorithme d'optimisation utilisé pour l'entraînement est la descente de gradient stochastique, avec un taux d'apprentissage de 0.01, et la fonction de coût utilisée est l'entropie croisée.

2.2 Le réjecteur

2.2.1 Architecture

Voici la structure que nous avons utilisée pour le réjecteur :

Couche	0	1	2	3	4	5	6
Type	Input	Conv2D	ReLU	Pool	Conv2D	ReLU	Pool
Format entrant	28*28*1	28*28*1	28*28*64	28*28*64	9*9*64	7*7*128	7*7*128
Noyau	-	3*3	-	3*3	3*3	-	3*3
Format sortant	28*28*1	28*28*64	28*28*64	9*9*64	7*7*128	7*7*128	2*2*128
Stride	-	1*1	-	3*3	1*1	-	3*3
Padding	-	oui	-	non	non	-	non
Couche	7	8	9	10	11	12	
Type	Dropout (0.25)	Flatten	Fc (ReLU)	RBF	RBFLayer (2000)	Fc (sigmoid)	
Format entrant	2*2*128	2*2*128	512	128	300	300	
Noyau	-	-	-	-	-	-	
Format sortant	2*2*128	512	128	300	300	1	
Stride	-	-	-	-	-	-	
Padding	-	-	-	-	-	-	

FIGURE 2.2 – Conv2D : couche de convolution ; ReLU : unité de rectification linéaire ; Pool : max pooling ; Fc (fonction d'activation) : couche densément connectée

Ce réseau contient une couche appelée couche RBF. Si l'on note α_i l'activation du neurone i de la couche précédente et X le vecteur d'activation ($X = [\alpha_1, \alpha_2, ...]$), la sortie d'un neurone classique, s'écrit $f_a(\sum_i w_i \alpha_i)$. En revanche, la sortie d'un neurone RBF s'écrit $f_a(||X - c||^2)$, avec $f_a(x) = exp(-\frac{x}{\alpha})$ et c un vecteur de paramètres appris. L'intérêt de ce type de couches est de détecter dans la couche précédente des distributions particulières, avec une décroissance rapide des activation quand celle-ci n'y correspond pas. On fera par la suite décroître progressivement α au cours de l'entraînement pour resserrer progressivement les fonctions d'activations autour de leur moyenne.

2.2.2 Entraînement

Nous avons utilisé un réjecteur à une sortie qui sera entraîné sur 50 epochs sur la base de données générée et la base d'apprentissage de MNIST normalisée entre -1 et 1. Les images de MNIST auront une valeur de la variable cible égale à 0 et celles générées égale à 0.9. Ce réjecteur permet de faire une régression pour estimer la valeur de la variable cible entre 0 et 1. Ainsi la fonction de coût que l'on a utilisée est l'écart quadratique moyen (EQM). Nous avons opté pour une régression afin de pouvoir comparer plus facilement

les performances du classifieur en fonction du seuil qu'on va appliquer à la sortie de ce réjecteur. En fonction de ce seuil, nous allons pouvoir choisir pour un exemple d'entrée entre deux classes : images à rejeter (supérieur au seuil) et images à garder (inférieur au seuil). La fonction d'activation de la dernière couche sature pour des valeurs proches de 0 ou 1, donc afin d'éviter la saturation du modèle pendant l'entraînement nous avons utilisé la valeur cible 0.9 au lieu de 1. En effet, si le modèle se sature, le gradient diminue voire s'annule, et le modèle ne s'entraîne plus. L'algorithme d'optimisation utilisé pour cet entraînement est l'algorithme Adagrad, avec un taux d'apprentissage de 0.001. Nous appliquons également une réduction d'un facteur deux au taux d'apprentissage chaque fois que le coût stagne pendant plusieurs epochs consécutives. Après chaque epoch d'entraînement, on divise enfin le paramètre des couches RBF par 1.1.

2.3 Le GAN

2.3.1 Architectures

2.3.1.1 Le générateur

Voici la structure que nous avons utilisée pour le générateur :

Couche	0	1	2	3	4
Type	Input	Dense	LeakyReLU (0.2)	Reshape	UpSampling2D
Format entrant	100	100	6272	6272	7*7*128
Noyau	-	-	-	-	2*2
Format sortant	100	6272	6272	7*7*128	14*14*128
Stride	-	-	-	-	-
Padding	-	-	-	-	-
Couche	5	6	7	8	
Type	Conv2D	LeakyReLU (0.2)	UpSampling2D	Conv2D	
Format entrant	14*14*128	14*14*64	14*14*64	28*28*64	
Noyau	5*5	-	2*2	5*5	
Format sortant	14*14*64	14*14*64	28*28*64	28*28*1	
Stride	1*1	-	-	1*1	
Padding	oui	-	-	oui	

FIGURE 2.3 – Conv2D : couche de convolution ; ReLU : unité de rectification linéaire ; Pool : max pooling ; Fc (fonction d'activation) : couche densément connectée

Comme les valeurs en sortie du générateur doivent être comprises entre -1 et 1, on applique la fonction d'activation tanh dans la dernière couche de convolution.

2.3.1.2 Le discriminateur

Voici la structure que nous avons utilisée pour le discriminateur :

Couche	0	1	2	3	4
Type	Input	Conv2D	LeakyReLU (0.2)	Dropout (0.3)	Conv2D
Format entrant	28*28*1	28*28*1	14*14*64	14*14*64	14*14*64
Noyau	-	5*5	-	-	5*5
Format sortant	28*28*1	14*14*64	14*14*64	14*14*64	7*7*128
Stride	-	2*2	-	-	2*2
Padding	-	oui	-	-	oui
Couche	5	6	7	8	
Type	LeakyReLU (0.2)	Dropout (0.3)	Flatten	Fc (sigmoid)	
Format entrant	7*7*128	7*7*128	7*7*128	6272	
Noyau	-	-	-	-	
Format sortant	7*7*128	7*7*128	6272	1	
Stride	-	-	-	-	
Padding	-	-	-	-	

FIGURE 2.4 – Conv2D : couche de convolution ; ReLU : unité de rectification linéaire ; Pool : max pooling ; Fc (fonction d'activation) : couche densément connectée

2.3.2 Entraînement

Le générateur doit apprendre la distribution P_g sur la base de données x. Le générateur permettra de définir une fonction G(z,θ_g), avec G la fonction représentée par le réseau et θ_g les poids. On note la fonction du discriminateur D(x, θ_d), avec encore une fois θ_d les poids du réseau. Le discriminateur est à une seule sortie. Celle-ci représente la probabilité que x provienne de la base d'apprentissage plutôt que de la base générée. On entraîne D à maximiser la probabilité d'attribuer le bon label aux deux types d'images, et on entraîne en même temps G à minimiser $\log(1-D(G(z)))$. En d'autres termes, G et D jouent successivement le rôle de min et max dans la fonction $V(G,D)$ ainsi définie :

$$min_G max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

Le générateur sera entraîné sur 200 epochs sur la base d'apprentissage de MNIST normalisée entre -1 et 1. A chaque epoch le générateur génère une nouvelle base de données qui va passer par le discriminateur afin de l'entraîner lui aussi à distinguer entre les images réelles (MNIST) et celles générées.

3 Résultats des tests et interprétations

Afin d'améliorer résultats obtenus par le classifieur à 95% sur les images de MNIST, nous avons utilisé trois méthodes différentes pour rejeter les images mal classées. Nous avons appliqué des seuils sur trois valeurs : la probabilité donnée en sortie du classifieur, l'entropie de Shannon des sorties du classifieur et le résultat du réjecteur. Nous avons ensuite comparé ces trois méthodes. Cette comparaison nous permettra de vérifier si notre méthode obtient de meilleurs résultats qu'une méthode "naïve".

3.1 Seuillage des probabilités et de l'entropie de Shannon

Dans un premier temps, nous avons regardé les probabilités que donne le classifieur pour une image. Chacune d'elles correspond à la probabilité que l'image appartienne à une classe donnée. Le classifieur se sert de ces valeurs pour choisir la classe à laquelle appartient l'image : celle de probabilité la plus élevée. Nous avons choisi de ne conserver que les images pour lesquelles cette probabilité est supérieure à un certain seuil. Ainsi, nous gardons les images pour lesquelles le classifieur est "plus sûr de lui" et rejetons celles que le classifieur semblent avoir plus de mal à classer. Puis nous regardons les performances du classifieur sur les images conservées.

En augmentant ce seuil de réjection, nous gardons un nombre plus petit d'images et espérons donc obtenir des meilleures performances. L'évolution de la précision du classifieur en fonction du nombre d'images rejetées est donnée par la figure 3.1.

Nous nous sommes ensuite intéressés à l'entropie de Shannon des sorties du classifieur. Si on considère un vecteur $X = [x_1, x_2, ..., x_n]$, chaque x_i ayant une probabilité P_i, alors l'entropie de Shannon de ce vecteur est donnée par :

$$H(X) = -\sum_{i=1}^{n} P_i log(P_i)$$

De même que précédemment, nous avons choisi un seuil et avons conservé uniquement les images qui avaient une entropie inférieure à ce seuil. Nous avons ainsi pu obtenir l'évolution de la précision du classifieur en fonction du nombre d'images rejetées, donnée par la figure 3.1.

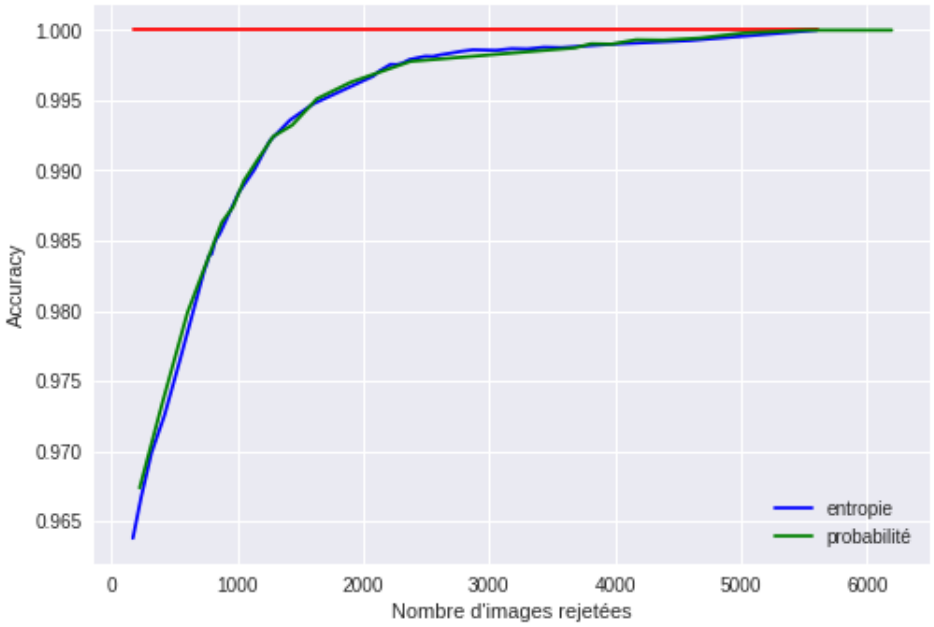


FIGURE 3.1 – Précision du classifieur en fonction du nombre d'images rejetées, avec un seuillage des probabilités (en vert) ou un seuillage de l'entropie (en bleu)

3.2 Entraînement et résultats du GAN

Nous avons choisi deux types de GAN : Un GAN avec des couches denses et un GAN avec des couches de convolution. Nous avons entraîné simultanément les deux générateurs ainsi que les discriminateurs sur 200 epochs et nous avons enregistré au fur et à mesure les images générées et leurs probabilités en sortie du discriminateur comme vu en partie 1. Ceci nous a permis de générer une base de données que nous avons utilisée pour entraîner notre réjecteur.

Après avoir entraîné le GAN, nous avons pu l'utiliser pour générer des images différentes de celles de MNIST, mais relativement proches. Pour chaque epoch de l'entraînement du générateur, nous avons généré des images et les avons stockées pour constituer une nouvelle base de données. Puis nous avons passé toutes ces images dans le discriminateur de la dernière epoch afin de pouvoir quantifier leur ressemblance aux images de MNIST. Les figures 3.2 et 3.3 sont des exemples d'images générées à différentes epochs.

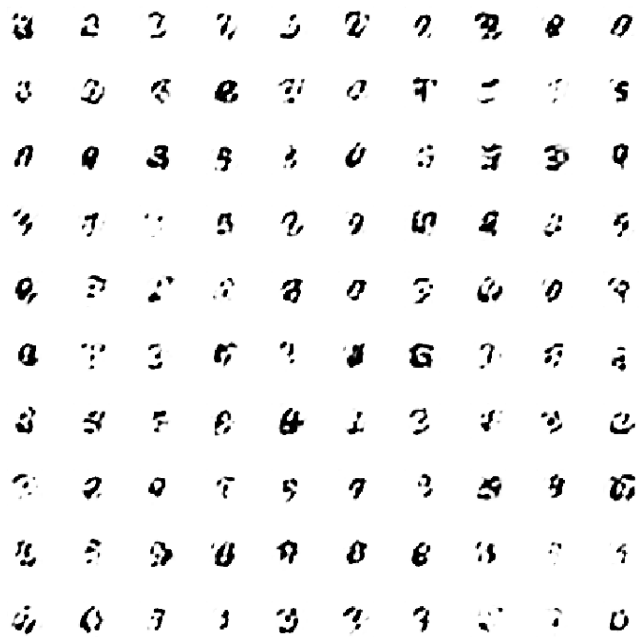


FIGURE 3.2 – Images générées par le générateur du GAN à l'epoch 1

FIGURE 3.3 – Images générées par le générateur du GAN à l'epoch 60

3.2.1 Autres méthodes

Après test de chacune des méthodes proposées pour l'apprentissage progressif, il semble que le réjecteur soit incapable d'apprendre ces données correctement. Les bases de données générées produisent des réjecteurs avec des performances très mauvaises qui renvoient systématiquement les mêmes valeurs quelles que soient les exemples proposés : il semble donc que ces bases de données ne soient pas adaptées, ou qu'elles nécessitent d'autres paramètres pour fonctionner correctement. Les tests sur des bases de données générées sans le GAN avec couches denses montrent les mêmes résultats que ceux avec, ce qui laisse à penser que la qualité des résultats ne dépend pas de ce GAN. Il serait intéressant de réaliser des tests plus poussés pour confirmer cette hypothèse.

3.3 Entraînement du réjecteur et tests du classifieur

Pour chaque seuil appliqué aux scores du discriminateur pour générer une base de donnée, on refait l'entraînement du réjecteur comme vu dans la partie 2. Pour chaque entraînement du réjecteur, nous avons testé notre classifieur. Pour cela, nous avons fait passer la base de données de test de MNIST (10 000 images) dans le réjecteur, et avons rejeté les images qui avait un résultat trop faible en sortie du réjecteur. Nous avons ainsi seuillé ces résultats pour ne garder qu'une partie des images, que nous avons ensuite classées à l'aide du classifieur.

3.3.1 Résultats avec un seul seuil du discriminateur

Après avoir généré une base d'apprentissage pour notre réjecteur permettant son entraînement, nous réalisons par la suite la courbe présentant l'évolution de la précision par rapport au seuil et nous évaluons la performance de notre réjecteur à l'aide de la mesure d'AUC comme nous allons le voir.

Les images gardées sont passées dans le classifieur, qui avait une précision de 95% sans le réjecteur. Nous calculons donc la performance du classifieur en fonction du seuil du réjecteur pour obtenir la courbe suivante et l'aire correspondante.

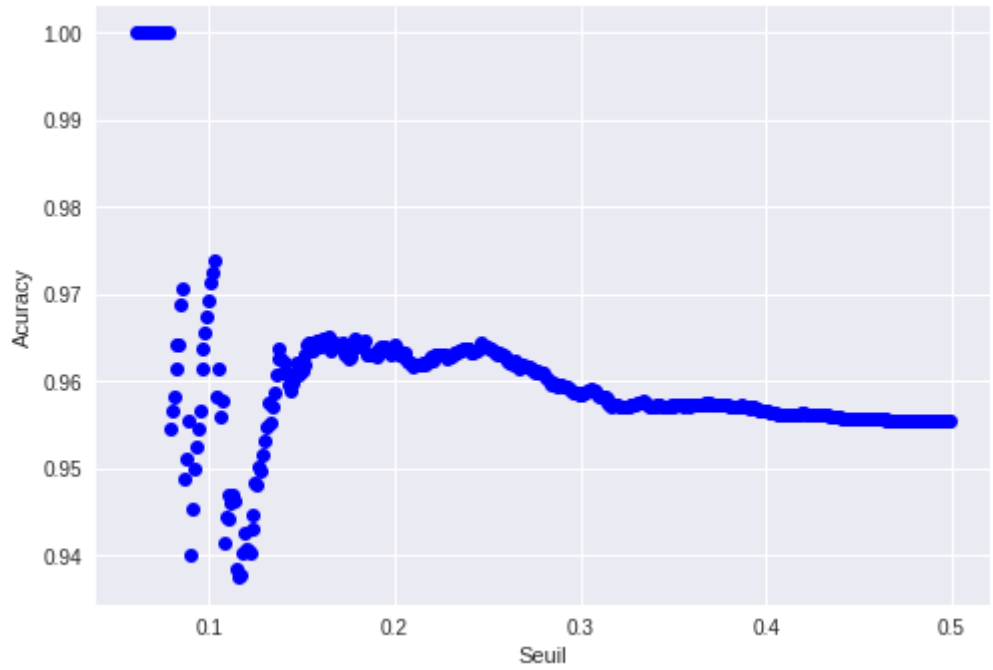


FIGURE 3.4 – Précision du classifieur en fonction du seuil du réjecteur

Afin de comparer les différentes méthodes, nous analysons les performances du classifieur (précision) en fonction du nombre d'images rejetées.

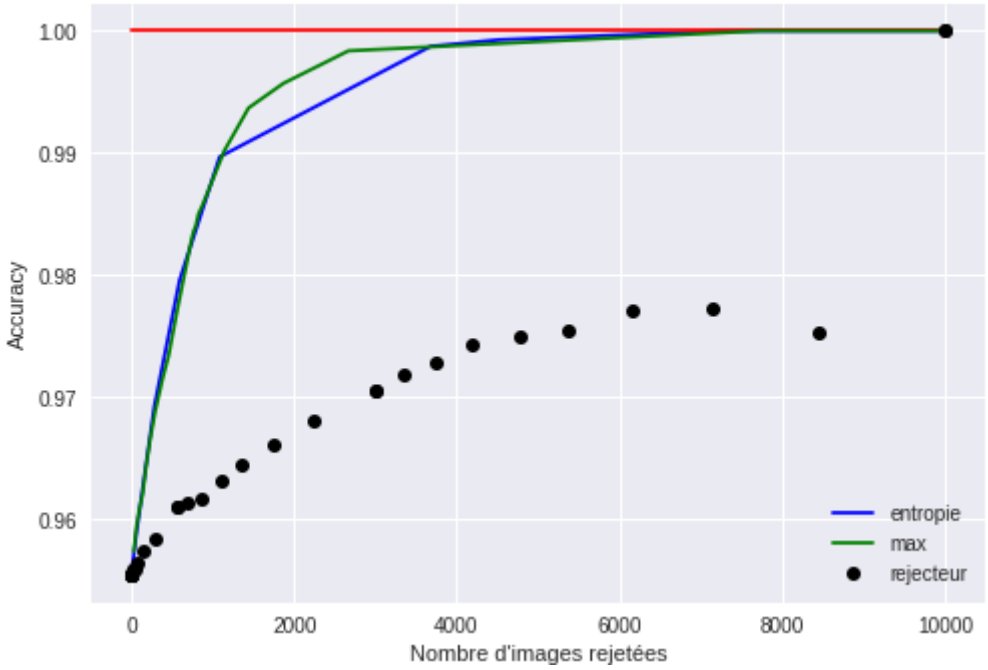


FIGURE 3.5 – Précision du classifieur en fonction du nombre d'images rejetées par le réjecteur (en noir) pour un seuil du discriminateur de 0.3, seuillage sur les probabilités (en vert) et sur l'entropie (en bleu)

3.3.2 Comparaison des différents seuils du discriminateur

Nous pouvons procéder de même pour d'autres seuils du discriminateur, et avons ainsi différentes bases d'entraînement pour le réjecteur. Pour chaque entraînement du réjecteur, nous obtenons des performances différentes du classifieur. Les figures 3.6 et 3.7 montrent ces performances pour des seuils du discriminateur de 0.2 et de 0.4.

Pour chaque seuil du discriminateur, nous calculons la performance du classifieur (après sélection du réjecteur) avec la métrique AUC comme le montre la figure 3.8. Cette figure permet d'avoir une idée du seuil à appliquer au discriminateur pour maximiser les performances du classifieur. Elle ne permet cependant pas de le déterminer précisément : il faudrait pour chaque seuil réaliser plusieurs entraînements du réjecteur et moyenner les valeurs de l'AUC pour avoir des données plus représentatives (des contraintes temporelles nous en ont empêché). On peut tout de même déjà voir qu'un seuil entre 0.25 et 0.45 est un bon seuil pour le discriminateur.

Nous remarquons que pour ces différents seuils les résultats sont assez proches et que le réjecteur permet d'améliorer la précision du classifieur, mais les méthodes de seuillage probabilistes et utilisant l'entropie de Shannon restent nettement meilleures.

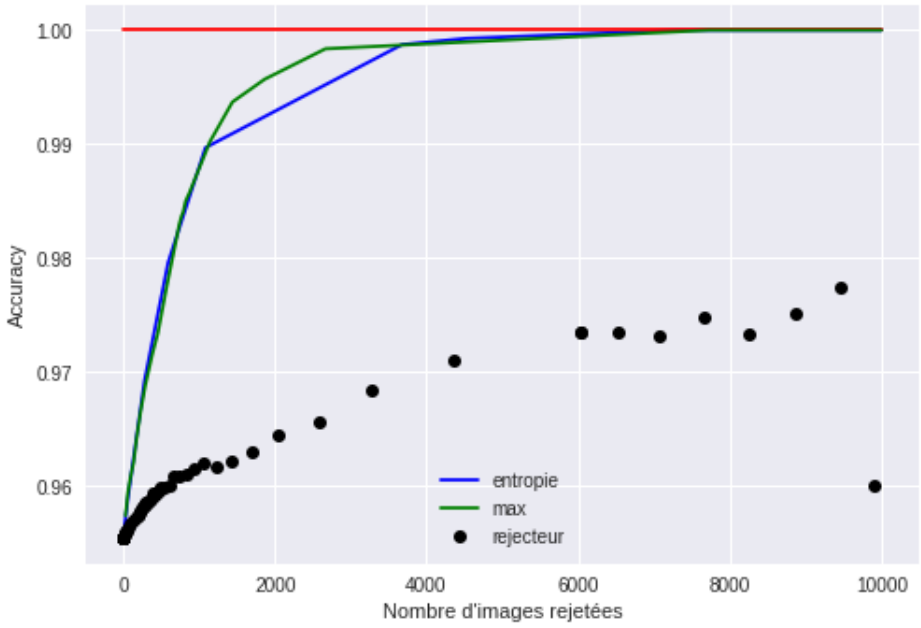


FIGURE 3.6 – Précision du classifieur en fonction du nombre d'images rejetées par le rejecteur (en noir) pour un seuil du discriminateur de 0.2

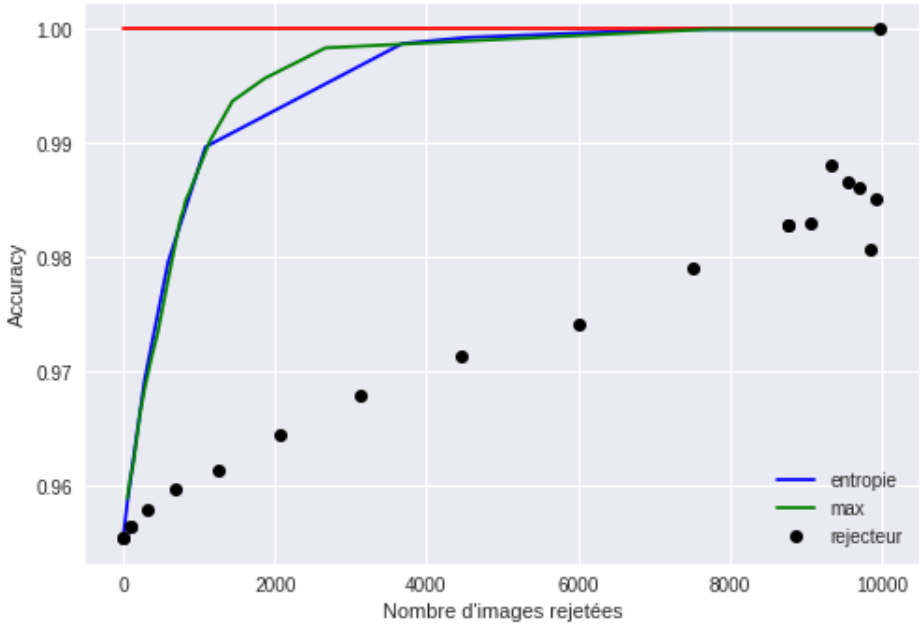


FIGURE 3.7 – Précision du classifieur en fonction du nombre d'images rejetées par le rejecteur (en noir) pour un seuil du discriminateur de 0.4

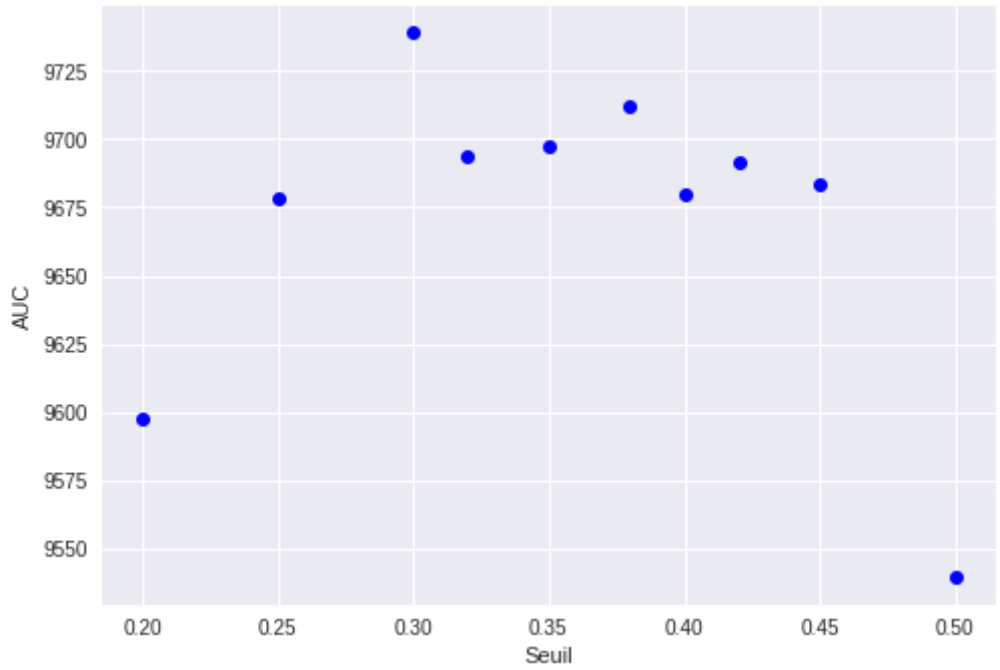


FIGURE 3.8 – Courbe de l'AUC du classifieur en fonction du seuil du discriminateur

3.4 Interprétation des résultats

A partir de ces résultats, on peut tirer plusieurs conclusions :

- pour optimiser la méthode choisie, même en fixant le mode de sélection de la base de données et les architectures des réseaux, il nous faudrait encore plus de résultats ;
- les architectures des réseaux et leurs hyper-paramètres n'ayant pas été optimisés, il y a fort à parier que ces résultats pourraient être largement améliorés même en supposant que la méthode choisie était la meilleure ;
- les images difficiles à classer, soit de distribution un peu éloignée de celle de MNIST, peuvent exister aussi dans la base d'entraînement de MNIST et nous les avons considérées comme des exemples à garder en apprenant notre réjecteur. Une amélioration serait de tester le classifieur sur la base d'apprentissage et de considérer les images mal classées comme des exemples négatifs durant l'apprentissage du réjecteur ;
- cette méthode repose en très grande partie sur la constitution d'une base de données représentative du problème. D'après les tests que nous avons réalisés, une approche fondée sur l'apprentissage progressif ne semble pas intéressante. Dans le cas d'une approche classique, nous nous sommes concentrés sur la méthode de sélection d'exemples la plus simple afin d'obtenir un point de départ. De meilleures performances peuvent sans aucun doute être atteintes avec des méthodes plus

complexes ;

— d'une manière générale, les résultats restent largement inférieurs aux méthodes naïves. On peut interpréter cet écart par l'hypothèse faite au début : nous avons considéré que les erreurs de classification étaient dues à un écart entre les distributions moyennes de MNIST et celles des exemples mal classés. Or, elles peuvent aussi être due à une confusion entre deux classes appartenant toutes deux à MNIST. Par conséquent, l'approche de la classification ouverte pour augmenter les performances d'un classifieur ne semble pas justifiée. En réalité, l'approche dite "naïve" que nous avons utilisée, c'est-à-dire un simple seuillage, semble plus intéressante d'un point de vue théorique : le réseau de classification est le plus à même de donner son degré de certitude sur les résultats qu'il donne puisque c'est justement à classer qu'il a été entraîné. Cette intuition théorique étant soutenue par les résultats observés, nous pensons que cette méthode restera meilleure quelles que soient les bases de données utilisées car il restera toujours un grand nombre d'exemples mal classés par le classifieur mais correspondant presque exactement à la distribution moyenne de MNIST.

Conclusion

Nous avons donc bien mis en place une méthode répondant à la problématique posée. Après analyse des résultats, il apparaît que la solution envisagée pour améliorer les résultats d'un classifieur semble très difficile à mettre en place pour obtenir des résultats ne serait-ce que comparables à ceux d'une solution très simple de seuillage. Nous pensons que cela est dû à notre hypothèse de départ disant que l'erreur de classification est due à la différence entre la distribution moyenne des exemples de MNIST et les exemples ambigus. En effet, il apparaît que ces exemples ambigus le sont plus vraisemblablement à cause d'une distribution trop proche de celle d'autres classes de MNIST, ce qui rend notre méthode difficile à appliquer et nécessairement moins efficace qu'un seuillage effectué sur les activations d'un réseau entraîné à classer. Si toutefois cette méthode devait être applicable, nous avons constaté qu'elle nécessiterait un très grand nombre d'essais pour trouver de façon assurée ses paramètres optimaux, qui sont très nombreux. Cette problématique nous aura néanmoins permis de nous confronter à celle plus vaste de la classification ouverte, dont nous avons pu voir toute la profondeur et la difficulté.

Bibliographie

— Towards Open Set Deep Networks, Abhijit Bendale, Terrance E. Boult, 2015

— Generative OpenMax for Multi-Class Open Set Classification, Zongyuan Ge et al., 2017

— DOC : Deep Open Classification of Text Documents, Lei Shu et al., 2017

— Article wikipédia sur les couches RBF

— Generative Adversarial Nets, Ian J. Goodfellow et al., 2014

— Keras Documentation