Projet de vision artificielle : Manipulation de dessins

Bardisbanian Lucas
ENSC
lbardisbania@ensc.fr

Do Nicolas
ENSEIRB-MATMECA
ndo001@enseirb-matmeca.fr

Radji Waris
ENSEIRB-MATMECA
wradji001@enseirb-matmeca.fr

Abstract

Dans ce projet, nous avons exploré la possibilité de générer des dessins en nous basant sur le jeu "Quick, Draw!" développé par Google et disponible sur https://quickdraw.withgoogle.com. Cette application propose un vaste jeu de données de dessins réalisés par les utilisateurs. Pour ce faire, nous avons d'abord utilisé l'architecture `Sketch-RNN` *pour générer des dessins. Cependant, nous avons constaté que cette approche présentait des limites en ce qui concerne la génération de dessins sur plusieurs classes. Pour remédier à cela, nous avons étudié le modèle* `Sketch-Pix2Seq`*, une amélioration de* `Sketch-RNN` *qui utilise des réseaux de convolution au lieu de réseaux de neurones récurrents.*

1. Introduction

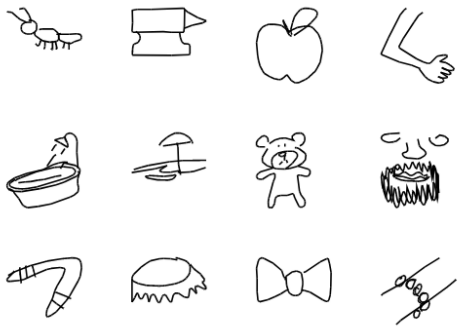

Figure 1. Exemples d'images du jeu de données "Quick, Draw !" (source : QuickDraw [5])

Le jeu "Quick, Draw!" de Google, consiste à faire deviner un dessin à une intelligence artificielle, et ceci en moins de vingt secondes (Figure 1). Les dessins réalisés par les utilisateurs ont permis de créer un grand jeu de données de 50 millions d'images réparties sur 345 catégories, tels que des animaux, des objets, des véhicules et des instruments de musique. Celui-ci est accessible en libre service et est accessible en plusieurs formats (vectoriel ou matriciel).

Il y a plusieurs défis potentiels liés à l'utilisation de ce jeu de données :

- Variabilité des dessins : les dessins du jeu de données sont réalisés par des personnes du monde entier et peuvent varier considérablement en terme de style, de qualité et de précision. Le jeu de données contient notamment des images qui n'ont pas été reconnu par le classifieur du jeu "Quick, Draw!". Par exemple, la plupart des dessins de la classe "chat" représentent une tête vue de face, mais certains dessins représentent un chat vu de profil.

- Dessins incomplets : lorsque que le classifieur du jeu "Quick, Draw!" réussi a trouvé la classe qui est associé au dessin de l'utilisateur, le dessin est ajouté au jeu de données, avec des traits potentiellement manquant

Il est possible d'appliquer un filtre très simple à cet ensemble de données, qui consiste à sélectionner les images en fonction d'une valeur `recognized` qui se trouve dans les meta-données. Cette valeur est simplement un booléen qui indique si le classifieur du jeu "Quick, Draw!" à réussi à trouver la classe de l'image.

Notre objectif principal est de développer une solution de génération d'images automatisée qui permettrait de produire des images de qualité d'une certaine classe en utilisant le jeu de données "Quick, Draw!". Nous avons par la suite explorer les possibilités de générer des images à partir de plusieurs classes.

Dans notre étude, nous avons d'abord exploré les GANs (Generative Adversarial Networks) et les VAEs (Variational Autoencoders) pour générer des images au format matriciel. Après avoir exploré ces approches, nous avons décidé de reprendre une architecture de `Sketch-RNN` pour générer des images au format matriciel, en utilisant des données séquentielles.

`Sketch-RNN` est un modèle de dessin génératif qui a été présenté dans un article de recherche publié en 2017 par Google [3]. Le modèle `Sketch-RNN` a été largement utilisé comme référence dans le domaine de la génération de dessins, et il a été appliqué à de nombreux problèmes

différents, notamment la génération de dessins de visages, de bâtiments et de paysages. Il a également été utilisé pour générer des dessins de manière interactive, permettant aux utilisateurs de dessiner des objets en donnant simplement quelques informations de départ.

2. Méthodologie

Afin de comprendre au mieux l'architecture de `Sketch-RNN`, nous allons faire un rapide résumé de c'est qu'un auto-encodeur variationnel (VAE) ainsi que les réseaux de neurones récurrents (RNN) et plus particulièrement de type Long Short-Term Memory (LSTM).

2.1. Auto-encodeur variationnel (VAE)

Un VAE nous intéresse dans notre cas pour son utilité dans la génération d'images. Basé sur les auto-encodeurs, l'idée basique est de réaliser un encodage qui fait une réduction de dimensions sur un espace latent puis un décodage pour revenir aux dimensions de base (Figure 2).

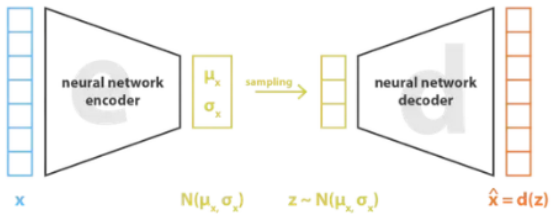

Figure 2. Principe d'un VAE (source : Towards Data Science [4])

Cet encodage sur l'espace latent permet plusieurs choses. Tout d'abord, réaliser des opérations sur un espace réduit afin de faciliter les calculs, mais également, dans le cas d'un VAE, de naviguer dans l'espace latent afin de générer de nouvelles données. Dans cette optique, il est nécessaire de régulariser cet espace latent afin de pouvoir générer des données cohérentes. Cette régularisation se fait en encodant selon une distribution normale permettant de centrer les différentes distributions ensemble et d'obtenir des variances ni trop fortes ni trop faibles (Figure 3). Pour générer une nouvelle image, on échantillonne un point depuis l'espace latent, puis on le décode grâce au décodeur.

Pour optimiser un VAE, nous nous basons sur deux termes:

- La divergence de Kullback-Leibler (KL divergence) est une mesure de la distance entre deux distributions de probabilité. Elle est souvent utilisée dans le cadre des VAEs pour mesurer la distance entre la distribution de l'espace latent du VAE et une distribution priori choisie (généralement une distribution gaussienne), pour encourager la distribution de l'espace latent à être proche de la distribution priori choisie.

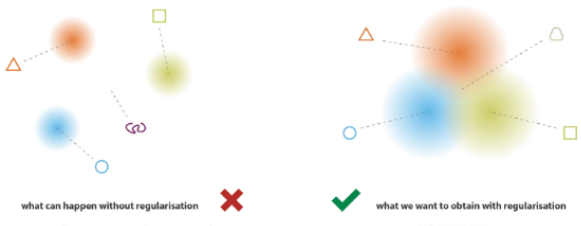

Figure 3. Régularisation de l'espace latent (source : Towards Data Science [4])

- La perte de reconstruction est une autre mesure de distance utilisée dans le cadre des VAEs. Elle mesure la distance entre l'entrée originale et sa reconstruction par le VAE. Elle est généralement calculée en utilisant une fonction de perte comme la distance Euclidienne ou la perte d'entropie croisée. La perte de reconstruction est utilisée pour s'assurer que le VAE est capable de reconstruire correctement l'entrée d'origine à partir de l'espace latent.

2.2. Réseaux de neurones récurrents (RNN)

Les RNN sont capables d'analyser des données séquentielles en retenant l'information des entrées précédentes. Ils nous intéressent car nos données comportent des données séquentielles (l'ordre dans lequel les traits ont été tracés) qui peuvent se révéler pertinentes. Néanmoins, un problème apparaît, le "vanishing gradient problem" : il se produit lorsque le gradient des poids d'un réseau de neurones tend à être très petit lorsqu'il est propagé à travers les couches du réseau. Cela peut entraîner une mauvaise convergence de l'algorithme de descente de gradient et rendre difficile l'optimisation des poids du réseau. Pour résoudre ce problème, les LSTM instaurent des portails d'entrée, de sortie et d'oubli qui permettent de gérer comment on se souvient de l'information et ainsi de mieux conserver les informations pertinentes (Figure 4).

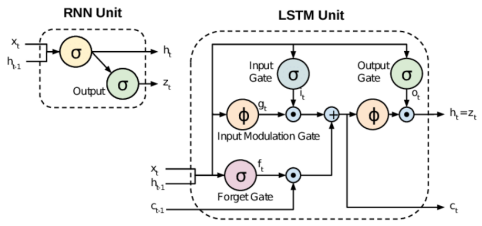

Figure 4. Fonctionnement d'un RNN et LSTM (source : arXiv [2])

2.3. Sketch-RNN

Sketch-RNN est un modèle de génération d'images utilisant un VAE basé sur des LSTM (Figure 5). Ce modèle permet différentes fonctionnalités: génération inconditionnelle d'images selon une classe donnée, reconstruction conditionnelle selon un dessin d'entrée, interpolation dans l'espace latent entre deux dessins d'entrées ainsi que la complétion d'un dessin incomplet donné en entrée, selon une classe.

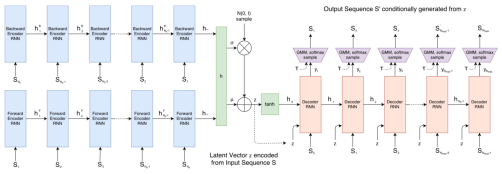

Figure 5. Architecture de Sketch-RNN (source : arXiv [3])

L'encodeur de Sketch-RNN est une LSTM bi-directionnelle. Une première LSTM prend en entrée la séquence de traits dans l'ordre dans lequel ils ont été réalisés tandis que la seconde les prend dans l'ordre inverse. La séquence de traits est représentée par une liste de vecteurs où chaque vecteur est composé de cinq éléments : $(\Delta x, \Delta y, p_1, p_2, p_3)$. Δx et Δy correspondent au décalage sur l'axe x et y par rapport au point précédent et p_1, p_2 et p_3 représentent trois états possibles du stylo (p_1 indique que le stylo touche actuellement le papier et qu'une ligne sera tracée pour relier le point suivant au point actuel, $p2$ indique que le stylo sera retiré du papier après le point actuel et qu'aucune ligne ne sera tracée ensuite et pour finir, p_3 indique que le dessin est terminé et que les points suivants, y compris le point actuel, ne seront pas pris en compte). Les deux LSTM s'entraînent avec des poids différents. Cette LSTM bi-directionnelle permet d'obtenir des informations à la fois sur le passé et sur le futur. Les deux sorties sont rassemblées pour encoder un nouveau vecteur z sur l'espace latent à partir d'une distribution normale. Le décodeur est une LSTM autorégressive permettant de reconstruire la séquence de traits au fur et à mesure, à partir de la sortie du temps précédent et du vecteur latent, et ainsi obtenir des informations temporelles sur comment est construit le dessin. A chaque pas de temps, la sortie est passée dans un mélange de modèle gaussien, qui est entraîné en parallèle du décodeur. Ce dernier permet de rendre les données de sorties plus cohérentes et semblables à celles en entrée.

L'entraînement de Sketch-RNN suit l'approche d'un VAE et repose sur la perte de reconstruction L_R et la divergence de Kulback-Leibler L_{KL}. La fonction de perte à optimiser correspond alors à $Loss = L_R + w_{KL} \cdot L_{KL}$ avec w_{KL} correspondant au poids que l'on accorde à la divergecne de Kulback-Leibler.

Plus précisément, la divergence de Kulback-Leibler L_{KL} correspond à l'équation 1.

$$L_{KL} = -\frac{1}{2N_z}(1 + \hat{\sigma} + \mu^2 + exp(\hat{\sigma})) \tag{1}$$

avec N_z la taille du vecteur latent, $\hat{\sigma}$ et μ étant les vecteurs obtenus à partir du dernier état caché de l'encodeur et correspondant à l'écart-type et à la moyenne.

La perte de reconstruction L_R, elle, est la somme de la perte logarithmique des termes d'offset $(\Delta x, \Delta y)$, L_s, et de la perte logarithmique des termes de l'état du stylo (p_1, p_2, p_3), L_p décrites dans les équations 2 et 3 : $L_R = L_s + L_p$

$$L_s = -\frac{1}{N_{max}} \sum_{i=1}^{N_s} log(\sum_{j=1}^{M} \Pi_{j,i} \cdot p(\Delta x_{i,j} \Delta y_{i,j})), \quad où$$
$$p(\Delta x_{i,j} \Delta y_{i,j}) = \mathcal{N}(\Delta x_i, \Delta y_i | \mu_{x,j,i}, \mu_{y,j,i}, \sigma_{x,j,i}, \sigma_{y,j,i}, \rho_{xy,j,i})$$
$$et \quad \sum_{j=1}^{M} \Pi_{j,i} = 1 \tag{2}$$

N_{max} est la longueur maximale de séquences, $\mathcal{N}(\Delta x, \Delta y | \mu_x, \mu_y, \sigma_x, \sigma_y, \rho_{xy})$ est la fonction de distribution de probabilité d'une distribution normale à deux variables. Chacune des M distributions normales à deux variables est constituée de cinq paramètres : $(\mu_x, \mu_y, \sigma_x, \mu_y, \rho_{xy})$, où μ_x et μ_y sont les moyennes, σ_x et σ_y sont les écarts types, et ρ_{xy} est le paramètre de corrélation de chaque distribution normale à deux variables.

$$L_p = -\frac{1}{N_{max}} \sum_{i=1}^{N_{max}} \sum_{k=1}^{3} p_{k,i} log(q_{k,i}) \tag{3}$$

en considérant (q_1, q_2, q_3) en tant qu'une distribution catégorielle pour modéliser les données de vérité terrain (p_1, p_2, p_3) et où $q_1 + q_2 + q_3 = 1$.

Un des défis majeurs à surmonter avec l'utilisation de LSTMs pour la génération de séquences de traits est d'apprendre au modèle quand il doit arrêter de dessiner. En effet, étant donné que les probabilités des trois états possibles du stylo sont très déséquilibrées (la probabilité de l'événement p_1 est beaucoup plus élevée que p_2, et l'événement p_3 ne se produit qu'une fois par dessin), il est difficile d'entraîner le modèle pour qu'il prévoit efficacement l'événement p_3 correspondant à la fin du dessin. Une des méthodes pour parer à ce problème aurait été d'assigner des poids différents à chaque évènement lors du calcul de la fonction de perte. L'évènement p_3 aurait un poids grandement supérieur à celui de p_2 qui lui même supérieur à celui de p_1. Cependant la solution optée a été d'imposer une

longueur maximale N_{max} que peut prendre la séquence. De ce fait, un dessin s'arrête lorsque la séquence atteint N_{max} ou lorsque le modèle à prédit l'événement p_3 à la longueur $N_s < N_{max}$.

2.4. Sketch-Pix2Seq

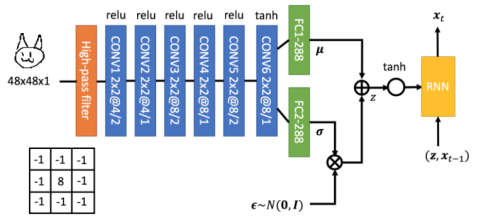

Figure 6. Encodeur de Sketch-Pix2Seq. Les couches convolutives sont représentées par h × w@d/s, où h, w, d et s représentent la hauteur et largeur du filtre, le nombre de canaux en sortie et la foulée (source : arXiv [1])

Sketch-Pix2Seq est un modèle de génération d'images qui s'inspire énormément de Sketch-RNN. En effet, ce modèle fournit les mêmes fonctionnalités que Sketch-RNN mais en essayant d'améliorer les performances de ce dernier, en particulier dans la génération d'images de plusieurs catégories.

Ce modèle reprend une grande partie de l'architecture de Sketch-RNN (VAE avec comme décodeur un LSTM) à la différence qu'il utilise en guise d'encodeur un réseau de neurones convolutif (Figure 6). Les données passées en entrée de l'encodeur ne sont donc plus constituées de séquences de traits des dessins mais d'images de dessins avec une résolution de 48 pixels par 48 pixels sur un seul canal que nous avons dû produire nous même à partir des séquences de traits, étant donné qu'il n'existait pas de jeu de données à disposition avec ces caractéristiques. La difficulté a vraiment été d'obtenir des images avec la résolution souhaitée tout en gardant une bonne qualité du dessin. Avant que ces images ne soient introduites dans l'encodeur, un filtre passe-haut est appliqué sur ces dernières. La sortie de la dernière couche convolutive est réarrangée en un vecteur unidimensionnel, qui est ensuite introduit dans deux couches "fully-connected" distinctes donnant en sortie μ et σ. μ et σ sont respectivement la moyenne et l'écart type de la distribution postérieure $p(z|X)$ apprise par l'encodeur où $z = \mu + \sigma \cdot \epsilon$ (avec ϵ un bruit gaussien) est le vecteur latent et X l'image d'entrée.

L'approche de ce modèle se focalise ainsi plus sur la capture des caractéristiques se trouvant sur le dessin que sur la mémorisation de la séquence de trait permettant de reproduire un dessin.

La fonction de perte à optimiser sur le modèle Pix2seq, à la différence de celle de Sketch-RNN, n'est constitué que de la perte de reconstruction ; la divergence de Kulback-Leibler disparaît. Cette suppression du terme de divergence permettrait à l'encodeur d'avoir une meilleur image de l'espace latent reflétant les caractéristiques des différentes catégories. En effet, selon les auteur de l'article de recherche présentant ce modèle [1], l'utilisation de la divergence Kulback-Leibler dans la fonction de perte de Sketch-RNN forcerait tous les vecteurs latents à être tirés de la même gaussienne, ce qui pourrait être le principal facteur menant à des résultats insatisfaisants pour l'apprentissage de catégories multiples car il est peu probable que les données de différentes catégories soient tirées de la même distribution.

3. Résultats

Les deux modèles ont été entraîné sur une Tesla T4, sur 200 itérations, des jeux de données de 7500 images et une taille de lot de 256. Après plusieurs expérimentations, nous avons décidé de mettre en place une dégradation du taux d'apprentissage de 0.001 à 0.0001 au fil des itérations. Cela permet d'améliorer la convergence de notre modèle et de minimiser le risque de divergence.

3.1. Génération d'image mono-classe

Notre première expérience avait pour but d'entraîner le modèle sur des dessins de chats. La figure 8 présente l'évolution de la valeur des fonctions de pertes pour les deux modèles. Bien que les termes des fonctions de pertes soient différent, ce qui rend leurs comparaison difficile.

Au niveau de la perte d'entraînement, les deux modèles convergent, ce qui signifie que plus augmenter le nombre d'itérations aurait sûrement produit un modèle de meilleur qualité. C'est aussi le cas pour la perte de validation du modèle Sketch-RNN. À l'opposé, le modèle Sketch-Pix2Seq semble faire du sur-apprentissage, la valeur de la perte croît considérablement à partir d'un nombre d'itérations. Afin d'expliquer ce comportement nous pouvons émettre plusieurs hypothèses :

- Un grand nombre de données est nécessaire à la bonne généralisation d'un CNN.
- La variabilité intra-classe n'est pas équilibré, autrement dit il n'y a pas forcement la même proportion de tête de chats et de chats dessinées en entier, dans les jeux de données d'entraînement et de validation.
- La suppression de la KL-divergence.

D'après nos expérimentations, les courbes ont une évolution similaire, quels que soient les paramètres de l'expérimentation ou le jeu de données choisi.

La figure 7 présentent des images qui ont été générées par les deux modèles. Il est possible de différencier les

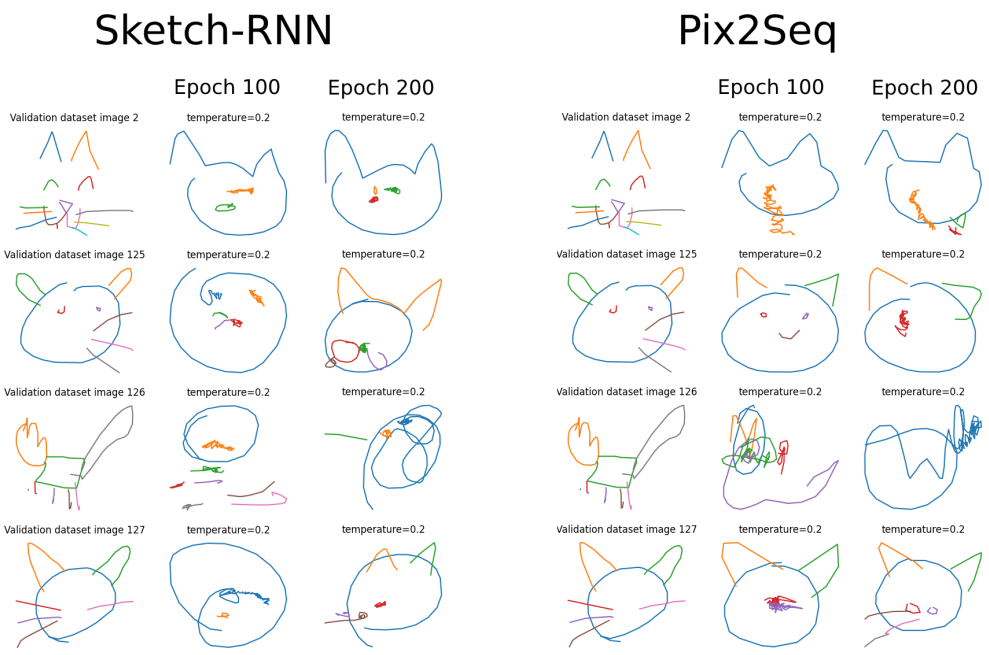

Figure 7. Images de chats générées par `Sketch-RNN` et `Sketch-Pix2Seq`

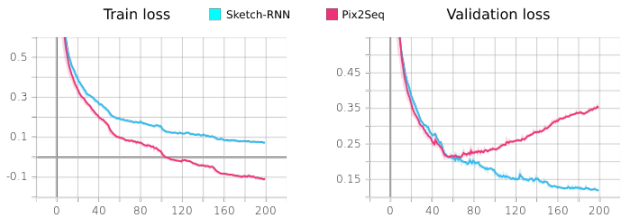

Figure 8. Évolution des pertes (Chats)

traits dessinées par les générateur, grâce à leurs couleurs. L'objectif du générateur est d'essayer de reproduire une image du jeu de données de validation, des images qui n'ont pas été vu par le modèle pendant l'entraînement, en l'encodant dans l'espace latent, puis en la décodant. Au bout de la 200ème itérations, les deux modèles ont l'air d'avoir assimilés le concept de chats, et arrivent à reproduire des dessins qui peuvent être reconnu comme chat. Les trois premiers traits dessinés sont respectivement de couleurs bleu, orange et vert, ce qui nous permet de voir que les modèles ont tendances à commencer à dessiner un cercle pour la tête puis des triangles pour les oreilles, ce qui est très proche de la façon dont les utilisateurs de "Quick, Draw!" dessinent leurs chats.

Étant donné que le jeu de données est en grande partie constituer de visage de chats, les modèles ont beaucoup de mal à reproduire les corps de chats. À vu d'oeil, `Sketch-Pix2Seq` semble produire des dessins de meilleurs qualités que `Sketch-RNN`, mais il est difficile de vraiment quantifier cette différence.

3.2. Génération d'image multi-classe

Afin d'évaluer la capacité de `Sketch-Pix2Seq` à générer des images d'un jeu de données multi-classe, nous avons reproduis l'expérience précédente avec des images de cochons et chats, ce qui rend les tâches d'encodage et de décodage beaucoup plus difficiles pour les modèles.

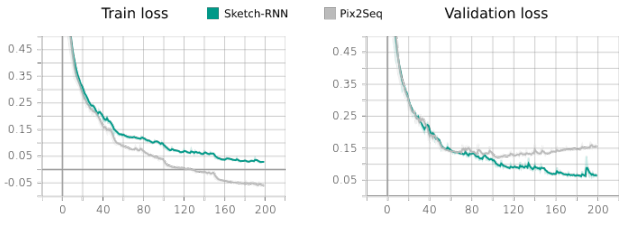

Figure 9. Évolution des pertes (Chats+Cochons)

La figure 9 montre que le modèle converge moins vite que lorsqu'il y a une seule classe. Les deux modèles ont l'air de mieux se généraliser au jeu de données de validation, le sur-apprentissage de `Sketch-Pix2Seq` est beaucoup moins marqué.

Néanmoins, avec nos configurations il est très difficile d'avoir des images de bonnes qualités, même si

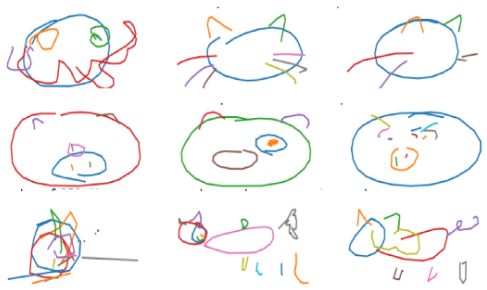

Figure 10. Images de chats/cochons générées par `Sketch-Pix2Seq`

`Sketch-Pix2Seq` (Figure 10) semble plus performant que `Sketch-RNN` (Figure 11) sur ce point.

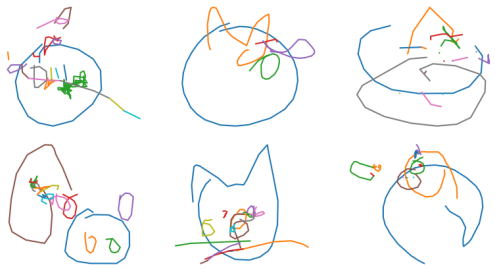

Figure 11. Images de chats/cochons générées par SketchRNN

4. Discussion

À cause de la limitation au niveau des ressources computationnelles, nos modèles ont été entraînés sur beaucoup moins de données et d'itération que dans les articles de références, mais nos résultats restent tout de même cohérents. Nous remarquons également une certaine cohérence dans l'ordre dans lequel les traits sont dessinés par le modèle. Par exemple, pour les chats, on remarque que le premier trait est souvent le rond de la tête, suivi par les deux oreilles et des détails sont ajoutés par la suite. Pour les cochons, c'est la même idée. Ce sont la tête et le groin qui sont dessinés en premier. Il est à noter que nos résultats ont des difficultés avec les variabilités intra classes et que notre modèle essaye parfois de représenter un chat de côté comme un chat de face, par exemple. Pour améliorer ce problème il faudrait trié les données, ou faire de l'augmentation de données pour rendre plus commun les types de dessins rares.

Si le temps nous l'avait permis, nous aurions bien aimé pouvoir faire la génération inconditionnelle d'images selon une classe donnée, c'est à dire générer une image sans entrée (ne pas encoder une séquence de traits avec l'encodeur puis utiliser le décodeur pour générer une image mais vraiment laisser le décodeur se débrouiller par lui même). En plus de cela, nous aurions voulu expérimenter l'interpolation sur des vecteurs latents afin de voir quelles caractéristiques des dessins sont associés à certains vecteurs latents.

Aujourd'hui la génération d'images est un des sous-domaines les plus tendances de l'intelligence artificielle, et plein de nouvelles architecture sont sorties depuis la création de `Sketch-RNN` et `Sketch-Pix2Seq`. Même si ces nouveaux modèles ont pour but de générer des images réaliste, il pourrait être intéressant de voir comment ils s'appliquent à un jeu de données de dessin. Les architecture de type `Transformers` commencent à bien s'adapter aux traitements de séries temporelles et pourrait remplacer les réseaux de neurones récurrents. Il serait aussi possible de combiner `Transformers` et les diffusions stables afin de pouvoir générer des dessins avec du textes.

5. Conclusion

Ce projet nous a permis de nous familiariser avec des architectures de réseau de neurones qui nous étaient inconnues et de comprendre les enjeux et la complexité de la génération de dessins. Nous avons d'abord utilisé l'architecture `Sketch-RNN`, mais avons constaté qu'elle présentait des limites en ce qui concerne la génération de dessins sur plusieurs classes. Pour remédier à cela, nous avons étudié le modèle `Sketch-Pix2Seq`, une amélioration de `Sketch-RNN` qui utilise des réseaux de convolution plutôt que des réseaux de neurones récurrents. Grâce à cette étude, nous avons acquis de nouvelles connaissances sur les différentes approches de génération de dessins et sur leurs avantages et inconvénients respectifs.

References

[1] Yajing Chen, Shikui Tu, Yuqi Yi, and Lei Xu. Sketch-pix2seq: a model to generate sketches of multiple categories, 2017. 4

[2] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description, 2014. 2

[3] David Ha and Douglas Eck. A neural representation of sketch drawings, 2017. 1, 3

[4] Joseph Rocca and Baptiste Rocca. *Understanding Variational Autoencoders (VAEs)*. Towards Data Science, 2019. 2

[5] Henry Rowley, Ruben Thomson, Jongmin Kim, Takashi Kawashima, Jonas Jongejan, and Nick Fox-Gieg. *Quick, Draw !* Google, 2016. 1