

# Implémentation de l'algorithme InfoGAN

Bruno Curzi-Laliberte<sup>1</sup>, Thomas Rochefort-Beaudoin<sup>2</sup>, Antoine Martin<sup>3</sup>, Lucas Aubrun<sup>4</sup>

Polytechnique Montréal, Département de Génie Logiciel

<sup>1</sup>bruno.curzi-laliberte@polymtl.ca, <sup>2</sup>thomas.rochefort-beaudoin@polymtl.ca, <sup>3</sup>antoine.martin@polymtl.ca, <sup>4</sup>lucas.aubrun@polymtl.ca

## Abstract

InfoGAN est une avancée importante en apprentissage non supervisé de représentations. L'article publié par l'équipe d'OpenAI en 2016 propose une extension du GAN classique ayant comme objectif de rendre la méthode capable d'apprendre des représentations utiles et interprétables. Cet article se penche sur la reproduction des résultats obtenus par l'équipe d'OpenAI. Il a été impossible de les répliquer avec les configurations originales de l'article d'InfoGAN démontrant un problème de reproductibilité. Nous avons réussi à obtenir des résultats décents sur MNIST avec une implémentation d'InfoGAN maison, réussissant même à reproduire les manipulations de code latent présentées dans l'article. Notre implémentation n'a toutefois pas réussi à générer des images d'aussi bonne qualité sur les ensembles de données SVHN et CelebA. Bien qu'il fut possible d'observer l'impact de certaines variables conditionnelles, la qualité générale des images obtenues par le générateur est bien inférieure à celle présente dans l'article.

## 1 Introduction

Une vaste partie du domaine de recherche en apprentissage non supervisé se penche sur l'apprentissage de représentations des données ayant comme objectif d'exposer des caractéristiques distinctes de l'ensemble de données étudié. Lors de la publication de l'article InfoGAN en 2016 par des chercheurs de la firme OpenAI, les approches les plus populaires pour ce type de problème étaient les modèles génératifs tels que les autoencodeurs variationnels (VAE) [1] ainsi que les réseaux adversariaux génératifs (GAN) [2]. Ces deux méthodes ont démontré une performance impressionnante, mais ne permettent pas de générer des représentations dites "démêlées", permettant d'allouer une dimension latente à une caractéristique spécifique des données. Par exemple, un GAN appliqué à l'ensemble MNIST ne permet pas de contrôler le chiffre qui sera généré par le générateur, ce qui est peu utile en pratique. Idéalement, on voudrait pouvoir interagir avec le générateur afin de contrôler une caractéristique spécifique des données générées.

Les approches proposées jusqu'alors pour l'apprentissage de représentations "démêlées" se basaient toutes sur l'apprentissage supervisé. De ceux-ci, la méthode disBM [3] propose une machine de Boltzman où des sous-ensembles des poids du réseau de neurones sont tenus fixes pour des données présentant une variation dans une caractéristique spécifique. Cette méthode pousse ainsi des régions spécifiques du réseau de neurones à apprendre une représentation pour une caractéristique spécifique. L'approche "Deep Convolutional Inverse Graphics Network" (DC-IGN) [4] a ensuite amené ce concept de "fixer" certaines régions du réseau de neurones à un autoencodeur variationnel. Ces deux méthodes ont le défaut évident de nécessiter des données étiquetées, apportant un coût plus important. Lors de la publication de l'article, une seule méthode non supervisée se penchait sur l'apprentissage de représentations "démêlées". La méthode "hossRBM" [5] basée sur les "restricted Boltzmann machines" propose une approche non supervisée ayant démontré pouvoir "démêler" des facteurs latents discrets tel que l'émotion d'un visage sur l'ensemble "Toronto Face Dataset".

InfoGAN propose une approche non supervisée à l'apprentissage de représentations "démêlées" d'images. Autrement dit, l'objectif de l'algorithme est d'entraîner un générateur capable de générer des images suivant certaines caractéristiques sans avoir à indiquer manuellement ces caractéristiques pour chaque image. Notamment, on pense à la classe d'une image dans un ensemble de données, mais on peut aussi imaginer qu'un InfoGAN capture des caractéristiques continues telles que la luminosité, l'angle, la taille... Le principe est assez simple : des variables conditionnelles sont ajoutées au bruit à l'entrée du générateur et, pour forcer le modèle à apprendre à les utiliser, il doit être capable de les recréer en sortie du discriminateur. Au final, on peut utiliser un bruit aléatoire ainsi qu'une certaine combinaison de variables conditionnelles pour générer l'image voulue. Prenons l'exemple de MNIST : l'équipe OpenAI propose l'utilisation d'une variable latente catégorique de 0 à 9 ainsi que 2 variables latentes continues lors de l'entraînement. En forçant le modèle InfoGAN à apprendre une représentation pour ces trois variables conditionnelles, les chercheurs ont ainsi obtenu un "mapping" de la variables discrètes au type de chiffre (0 à 9), ainsi qu'une représentation de la rotation et de la largeur du chiffre pour les variables continues.

L'intérêt des InfoGANs est bien sûr plus grand que ce

qu'on peut en faire avec MNIST. OpenAI réussi à appliquer l'algorithme sur CelebA par exemple, où l'on constate que les variables conditionnelles représentent des aspects critiques de la génération de visages, à savoir : "Azimuth (pose), Presence or absence of glasses, Hair style, Emotion" [6] (fig 6, Manipulating latent codes on CelebA).

## 2 Principes théoriques

### 2.1 De GAN à InfoGAN

L'algorithme InfoGAN étend le fonctionnement des *Generative Adversarial Networks* (GAN) développés par Ian Goodfellow et al [2]. Un GAN est constitué de deux réseaux de neurones, G (modèle génératif) et D (modèle discriminatif), qui sont en compétition l'un contre l'autre selon les étapes suivantes. Supposons que l'on possède un ensemble de données X, suivant une distribution  $\rho_X$ , et que l'on veuille générer des données artificielles qui pourrait appartenir à cet ensemble. À partir de données Z suivant une distribution quelconque  $\rho_Z$  (un bruit gaussien par exemple), le réseau G crée une distribution de données  $\rho_G$ . Des éléments  $w$  issus des deux distributions  $\rho_X$  et  $\rho_G$  sont ensuite donnés au réseau D afin qu'il détermine leur probabilité d'appartenance à la distribution  $\rho_X$  (en d'autres mots, si ce sont des vraies données ou des données artificielles). Les deux réseaux prennent alors part à un jeu minimax, régi par l'équation suivante:

$$\min_G \max_D V(D, G) = \mathbf{E}_X[\log(D(x))] + \mathbf{E}_Z[\log(1 - D(G(z)))] \quad (1)$$

Cependant, l'algorithme GAN utilise un vecteur de bruit continu z, mais n'a aucune contrainte sur la manière de l'utiliser. Il y a donc peu de contrôle sur le détail des images qui vont être générées. Comme mentionné plus haut, il serait intéressant de pouvoir contrôler certaines caractéristiques des images générées, et ainsi obtenir des représentations "démêlées".

Une extension proposée du modèle GAN dans l'article original proposait l'utilisation de variables conditionnelles pour représenter les informations sémantiques des données à générer. C'est ce que fait InfoGAN. On génère ces variables selon l'ensemble de données mais on les utilise de manière non supervisée. Autrement dit, on ne décide pas de manière supervisée quelles variables conditionnelles utiliser, mais bien uniquement selon par exemple le fait qu'on utilise MNIST on peut proposer l'utilisation d'une variable catégorique discrète de 0 à 9.

L'entrée du générateur  $G(c, z)$  est composée de deux parties: un bruit incompressible Z et des variables latentes  $c_i$  qui auront pour rôle de cibler des caractéristiques fixes des données. Dans l'article, des variables latentes de types catégoriques (one-hot) de dimensions 10 ainsi que des variables continues suivant une loi normale sont utilisées.

Pour s'assurer que le générateur n'ignore pas les variables conditionnelles et qu'elles apportent de l'information utile à la génération de données, l'information mutuelle entre les variables latentes et la distribution générée  $\rho_G$  doit être élevée. Cette information mutuelle  $I(c; G(c, z))$  est définie comme suit :

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X), \quad (2)$$

où  $H$  désigne l'entropie d'une variable. Ainsi, si deux variables X et Y sont indépendantes, leur information mutuelle  $I(X; Y) = 0$  c'est-à-dire qu'aucune ne révèle d'information sur l'autre. Donc pour maximiser  $I(c; G(c, z))$ , une nouvelle équation minimax régularisée est établie :

$$\min_G \max_D V_1(D, G) = V(G, D) - \lambda I(c; G(z, c)). \quad (3)$$

Ainsi, si le générateur G veut minimiser (3), il se retrouve à maximiser l'information apportée par les variables latentes.

### 2.2 Adaptation pratique

En pratique, il est difficile de calculer l'information mutuelle exacte car cela requiert d'accéder à  $H(c; G(z, c))$ , l'entropie des variables latentes étant donnée une distribution  $\rho_G$  générée par G à posteriori. Une approximation est alors utilisée, sous la forme d'une distribution auxiliaire  $Q(c|G(z, c))$ . Cette approximation est connue sous le nom de *Variational Information Maximization* [7].

En d'autres mots, un réseau de neurones auxiliaire Q est ajouté. Il partage les mêmes paramètres et hyperparamètres que le modèle discriminatif D mais, là où le modèle D cherche à prédire si une image est réelle ou artificielle, le modèle Q cherche à prédire les variables latentes utilisées en entrée du générateur G. Le générateur est donc forcé d'apprendre à utiliser ces variables latentes de façon représentative, afin que Q puisse les reconnaître.

## 3 Expériences

Dans cette section, nous présenterons la démarche entreprise pour tenter de reproduire les résultats de l'article InfoGAN.

### 3.1 Code source d'OpenAI

Nous avons premièrement tenté d'exécuter le code d'OpenAI disponible sur Github afin d'obtenir une référence. Cependant, le code présent dans le répertoire est plutôt vieux et difficile à exécuter à cause des versions des librairies utilisées. On peut y retrouver la configuration utilisée, décrite dans l'article, comme les architectures de réseaux neuronaux, les taux d'apprentissage, les tailles de batch, etc... mais il n'a pas été possible de ré-implementer l'algorithme en se basant uniquement sur ce code source.

Le fichier README fourni par OpenAI propose une procédure pour exécuter le code avec une image docker. Le lien étant incorrect, d'autres moyens pour obtenir un code de référence ont été explorés. Après avoir essayé diverses versions de Tensorflow et rencontré de multiples erreurs de Tensorflow/Prettytensor/etc., cette démarche a été abandonnée.

### 3.2 Implémentation de l'équipe

Par la suite, une structure de base pour entraîner un InfoGAN a été cherchée. À partir de [8], un code fonctionnel a pu être intégralement implémenté et exécuté. Cependant, les résultats obtenus sur MNIST étaient de très mauvaise qualité comparé à ceux de l'article original (Figure 1).

Figure 1: Résultats obtenus par l’implémentation de [8]



Partant de cette base, plusieurs modifications ont été apportées afin de permettre une meilleure paramétrisation et l’utilisation de combinaisons de variables latentes discrètes et continues. L’algorithme résultant s’exécute correctement sur tous les ensembles de données utilisés (MNIST, CelebA et SVHN) mais fourni des résultats nettement moins bon que ceux d’OpenAI. Deux différences importantes existent entre l’InfoGAN original et notre implémentation.

Premièrement, les configurations des réseaux de neurones ont été modifiées, car celles décrites dans l’article ne nous ont jamais permis d’obtenir de bons résultats.

Deuxièmement, la fonction de perte utilisée est différente car nous n’avons pas réussi à reproduire celle appliquée par OpenAI. Il semble que leur perte soit créée séquentiellement : un écart *loglikelihood* est tout d’abord calculé par rapport aux résultats du discriminateur D, puis des estimateurs *discrete & continuous mutual information estimators* sont calculés (soustraction d’un terme d’entropie avec un terme d’entropie croisée) pour appliquer la formule d’information mutuelle (2). La perte finalement obtenue est propagée à D et G pour leur apprentissage. Or nous n’avons pas réussi à implémenter le calcul des termes d’entropie et d’entropie croisée (code source difficile à lire et impossible à exécuter).

Nous nous sommes donc inspiré de [9] pour implémenter une fonction de perte tenant compte de l’information mutuelle. L’utilisation de cette fonction est justifiée car, en exécutant le code fourni par [9], on remarque que la perte converge vers 2.3 pour l’ensemble MNIST. Il s’agit du seuil identifié par OpenAI à la figure 1 de leur article. Cependant, cette formule fonctionne pour l’information mutuelle des variables catégoriques discrètes, mais ne s’applique pas aux variables continues.

Nous avons donc finalement utilisé une fonction de minimisation de *Categorical Cross-Entropy Loss* pour les variables catégoriques car cela produisait de meilleurs résultats que la fonction de maximisation d’information mutuelle. Pour les variables conditionnelles continues, nous utilisons

une fonction de perte *Mean Squared Error*. Le générateur et discriminateur sont entraînés avec une fonction de perte *Binary Cross-Entropy* et non la fonction de perte logarithmique proposée par OpenAI.

## 4 Résultats

Notre implémentation a été testée sur les ensembles de données MNIST [10], ”Street View House Number” (SVHN) [11] et CelebA [12]. Pour les trois ensembles, il fut décidé d’utiliser une seule variable latente catégorique (c1) ainsi que deux variables latentes continues (c2 c3). Les mêmes hyperparamètres pour l’entraînement ont été utilisés pour tous les ensembles (Table 1). L’architecture du Q-network fut également pareille pour tout les ensembles, soit une couches pleinement connectée de 128 unités cachées suivi d’une ”BatchNormalisation” et d’une activation ”LeakyRelu” avec une pente de 0.2.

Table 1: Hyperparamètres utilisés

Hyperparamètres	Valeur
Taux d’apprentissage (Gen et Discr)	2e-4
Adam - Beta1	0.5
Nb epochs	100
Batch size	128

Le code utilisé pour ce projet est disponible sur GitHub: <https://github.com/BrunoC-L/8225-project>.

### 4.1 MNIST

Les architectures du générateur et du discriminateur/q-network utilisés pour l’ensemble MNIST sont présentées aux deux figures ci-dessous:

Figure 2: Architecture du générateur pour MNIST.

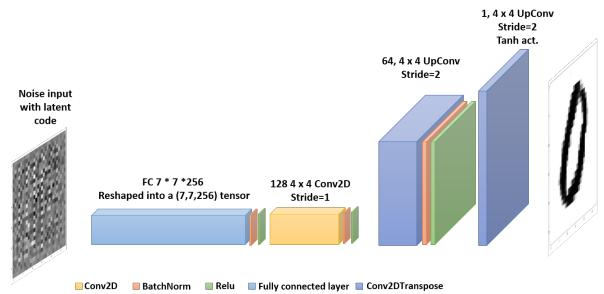
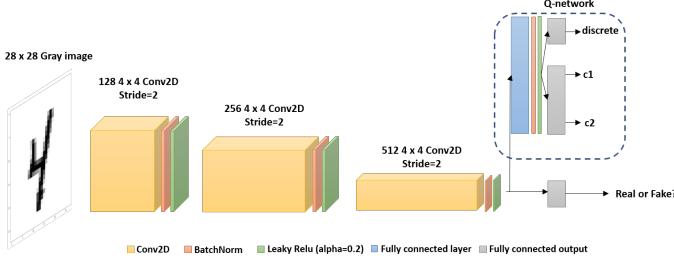


Figure 3: Architecture du discriminateur et Q-network pour MNIST.



Les figures suivantes présentent nos meilleurs résultats. Nous faisons varier les codes C1 (discret), C2 (continu) et C3 (continu) pour MNIST, qui représentent respectivement le chiffre généré, l'angle de celui-ci et la largeur du chiffre. Chaque figure contient 10 colonnes et 10 lignes, où chaque colonne représente une valeur de C1.

La figure 4 présente une variation uniquement sur C1, ainsi qu'une valeur différente de bruit Z pour chaque ligne. On remarque que même en changeant le bruit, on peut généralement associer une colonne à un chiffre. Il y a quelques exceptions avec la 2<sup>e</sup> colonne qui génère des 3, 5 et 8, puis la 4<sup>e</sup> qui génère des 3 et 5. Les codes C2 et C3 sont maintenus constants.

Figure 4: Images générées de MNIST en variant la variable discrète C1 (type de numéro).

Varying discrete latent code									
4	3	6	3	1	2	0	8	7	9
4	5	6	5	1	2	0	8	7	9
4	3	6	5	1	2	0	8	7	9
4	8	6	5	1	2	0	8	7	9
4	3	6	3	1	2	0	8	7	9
4	6	6	5	1	2	0	8	7	9
4	5	6	5	1	2	0	8	7	9
4	3	6	5	1	2	0	8	7	9
4	5	6	5	1	2	0	8	7	9
4	3	6	3	1	2	0	8	7	9

La figure 5 présente une variation du premier code continu, soit C2. Celui-ci représente la rotation du chiffre dans notre cas. Encore une fois, nous varions C1 pour chaque colonne, mais ne varions pas le bruit pour chaque ligne. Cette fois-ci, les lignes représentent les valeurs de C2 de -2 à 2. Il est plutôt intéressant de remarquer que les chiffres varient tous entre 0 degré et environ 30 degrés en sens horaire. On aurait pu s'attendre à ce que la variation soit plutôt entre -30 et 30 degrés par exemple. Nous pouvons poser l'hypothèse qu'un biais existe dans l'ensemble de données, dû à une majorité de la population étant droitière. Il serait donc normal d'observer une inclinaison dominante dans l'ensemble de données original.

Figure 5: Images générées de MNIST en variant la variable continue C2 de -2 à 2 (rotation).

Varying discrete latent code										
Varying c1 from -2 to 2	4	3	6	3	1	2	0	8	7	9
	4	8	6	3	1	2	0	8	7	9
	4	3	6	3	1	2	0	8	7	9
	4	3	6	5	1	2	0	8	7	9
	4	3	6	5	1	2	0	8	7	9
	4	5	6	5	1	2	0	8	7	9
	4	5	6	6	1	2	0	8	7	9
	4	6	6	6	1	2	0	8	7	9
	4	8	6	6	1	2	0	8	7	9
	4	8	6	6	1	2	0	8	7	9
	4	3	6	6	1	2	0	8	7	9
	4	3	6	6	1	2	0	8	7	9

La figure 6 présente une variation du dernier code, C3, qui est lui aussi continu. Le fonctionnement de ce code est identique à C2, mais ils prennent des valeurs indépendantes lors de l'entraînement donc représentent forcément des caractéristiques différentes. Cette fois, nous remarquons qu'il semble y avoir une tendance à représenter la largeur du chiffre, soit plus grande pour C3 négatif et graduellement plus fine vers C3 positif.

Figure 6: Images générées de MNIST en variant la variable continue C3 de -2 à 2 (largeur).

Varying discrete latent code										
Varying c3 from -2 to 2	4	8	6	5	1	2	0	8	7	9
	4	3	6	5	1	2	0	8	7	9
	4	3	6	5	1	2	0	8	7	9
	4	3	6	5	1	2	0	8	7	9
	4	3	6	5	1	2	0	8	7	9
	4	3	6	3	1	2	0	8	7	9
	4	3	6	3	1	2	0	8	7	9
	4	3	6	3	1	2	0	8	7	9
	4	3	6	3	1	2	0	8	7	9
	4	3	6	3	1	2	0	8	7	9
	4	3	6	3	1	2	0	8	7	9
	4	3	6	3	1	2	0	8	7	9
	4	3	6	3	1	2	0	8	7	9

On peut donc conclure que notre algorithme fonctionne très correctement sur l'ensemble MNIST : les chiffres générés ressemblent aux originaux, et deux caractéristiques (largeur et inclinaison) peuvent être contrôlées de manière similaire à l'article InfoGAN.

## 4.2 SVHN

Les architectures du générateur et du discriminateur/q-network pour l'ensemble SVHN sont présentées aux deux figures ci-dessous:

Figure 7: Architecture du générateur pour SVHN.

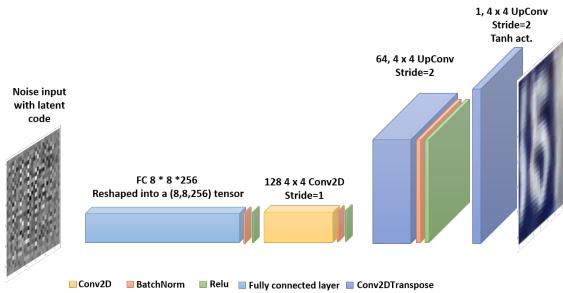
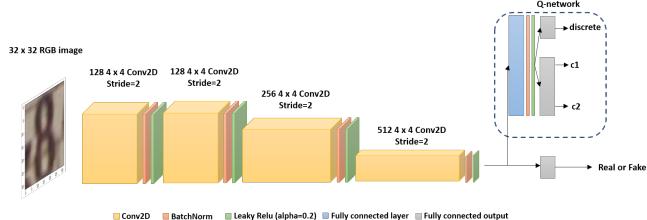
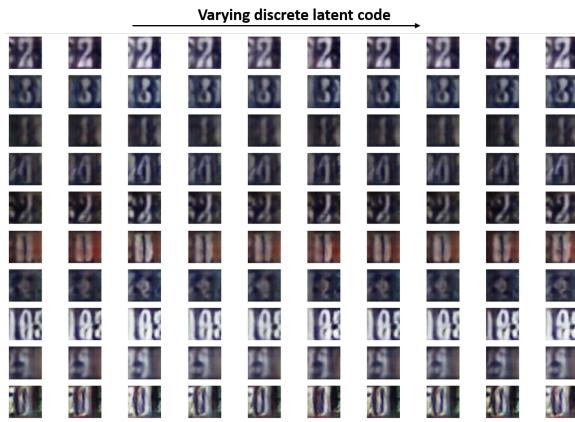


Figure 8: Architecture du discriminateur et Q-network pour SVHN.



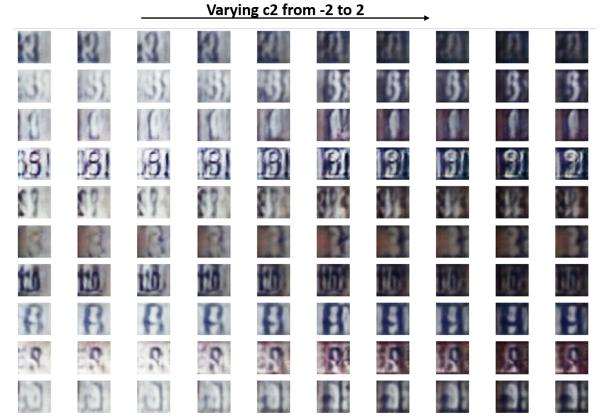
Les figures suivantes présentent les résultats obtenus sur l'ensemble SVHN. Les résultats globalement obtenus ne sont pas de très bonne qualité, donc il n'a pas été possible de déterminer précisément quelle influence avait le code discret ( $C_1$ ) sur la génération de données. En effet, nos résultats pour la variation de  $C_1$  (horizontalement, figure 9) ne montrent presque pas de différences pour les différentes valeurs. Dans l'article original, les auteurs utilisent quatre variables catégoriques discrètes. Ils indiquent que l'une d'entre elles réussit à capturer le contexte entourant le chiffre central de l'image générée, c'est-à-dire des parties d'autres chiffres (coupées par le bord de l'image) que l'on peut identifier sur la gauche ou sur la droite de l'image. Ceci prouve qu'InfoGAN peut apprendre à séparer un chiffre de son contexte.

Figure 9: Images générées en variant le code latent discret  $C_1$  (influence indéterminée).



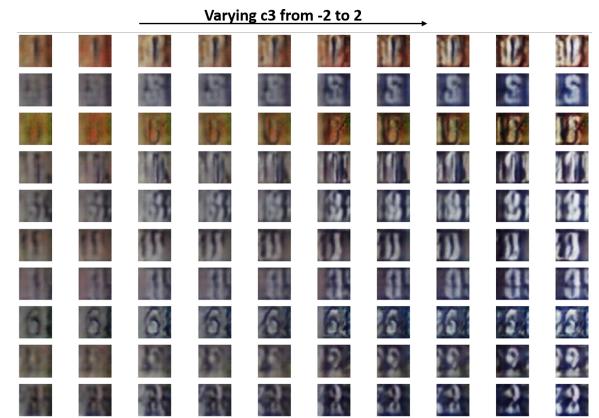
La figure 10 présente une variation du premier code continu  $C_2$ , qui représente la variation de luminosité sur l'image générée. On constate en effet que l'image s'assombrit lorsque  $C_2$  varie de -2 à 2 (horizontal, de gauche à droite).

Figure 10: Images générées en variant le code latent continu  $C_2$  (luminosité).



La figure 11 présente une variation du deuxième code continu  $C_3$ , qui représente la variation de contraste sur l'image générée. Les chiffres (ou symboles qui sont parfois difficilement assimilables à des chiffres) au centre des images se distinguent de plus en plus à mesure que  $C_3$  augmente de -2 à 2. Étant donné que les deux variables continues semblent avoir très bien rempli leur rôle, il est assez étonnant que la variable discrète n'ait pas provoqué de variation (clairement visible du moins).

Figure 11: Images générées en variant le code latent continu  $C_3$  (contraste).



Ces résultats montrent des performances assez limitées sur l'ensemble de données SVHN. L'utilité de la variable discrète n'a pas pu être mise en avant, et la génération des images est d'assez mauvaise qualité de manière générale. En effet, on ne parvient pas à identifier des chiffres sur chacune d'entre elles. En revanche, les variables continues parviennent à capturer des caractéristiques spécifiques (luminosité, contraste).

### 4.3 CelebA

Les architectures du générateur et du discriminateur/q-network utilisés pour l'ensemble CelebA sont présentées aux deux figures ci-dessous:

Figure 12: Architecture du générateur pour CelebA.

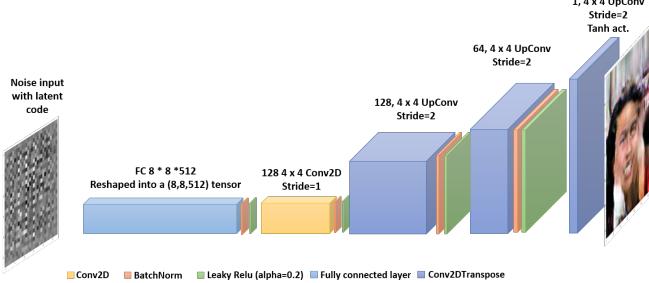
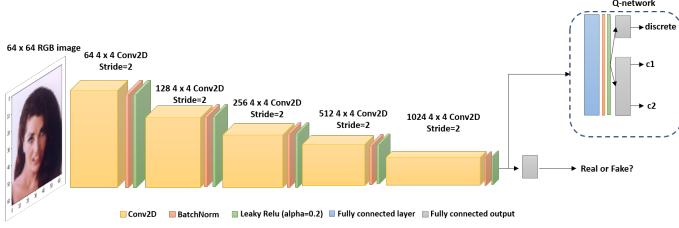


Figure 13: Architecture du discriminateur et Q-network pour CelebA



Les résultats pour cet ensemble de données sont présentés sur les figures 14, 15 et 16. Les images générées semblent être de bonne qualité, car des visages sont clairement identifiables. Cependant, en regardant plus précisément, on constate que chaque visage est en réalité un mélange de deux ou trois visages qui paraissent fusionner ensembles.

La figure 14 montre la variation du code discret C1, qui joue sur les coiffures générées. En effet, sur une même colonne, les visages ont (presque tous) des coiffures aux caractéristiques similaires (longueur, coloration), et ces caractéristiques varient d'une colonne à l'autre.

La figure 15 montre la variation du premier code continu C2, qui joue sur le ton général de l'image générée. On peut ainsi voir les images devenir de plus en plus claires à mesure que C2 augmente, entre -2 et 2.

La figure 16 montre la variation du second code continu C3. Son impact est assez difficile à déterminer précisément mais on pourrait croire qu'il joue sur l'unification d'un visage, c'est-à-dire sur le fait de ne pas avoir plusieurs visages fusionnés en un seul. En effet, si l'on regarde de près les images des colonnes de gauche, les visages semblent très brouillés (plusieurs couleurs de peau différentes, formes inhabituelles comme le menton écrasé sur la troisième ligne, etc...). Puis en se déplaçant vers la droite, ce type de défauts semble disparaître et les visages deviennent de meilleure qualité. La variable C3 contrôlerait alors à quelle personne appartient le visage généré (de façon pondérée).

Figure 14: Images générées en variant le code latent discret C1 (chevelure).

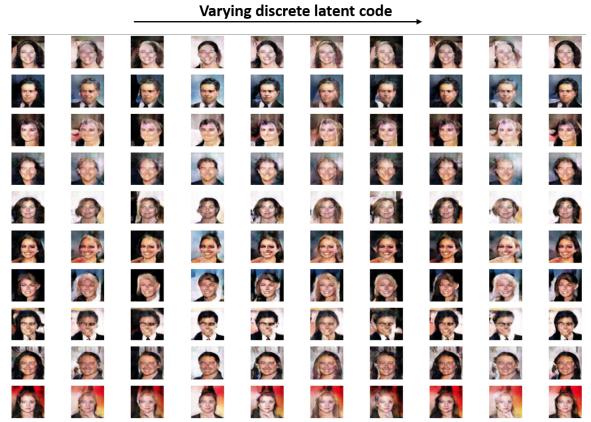


Figure 15: Images générées en variant le code latent continu C2 (ton général).



Figure 16: Images générées en variant le code latent continu C3 (visage unifié).



L'algorithme obtient donc des performances correctes sur cet ensemble de données : la plupart des images sont d'assez

mauvaise qualité mais certaines sont acceptables, et les trois codes latents testés ont su prouver leur utilité.

## 5 Analyse de l'approche utilisée

Dans le cadre de ce projet, notre premier objectif était de réussir à reproduire les résultats d'OpenAI par quelconque moyen. La théorie derrière InfoGan n'étant pas très imposante, nous espérions être en mesure d'implémenter assez rapidement une version satisfaisante et éventuellement reproduire des figures du niveau d'OpenAI.

Notre sujet était assez facile à comprendre, mais difficile à entraîner correctement. Il est bien connu que les GANs sont des modèles pouvant être très instables et énormément sensibles aux nombreux hyperparamètres [13] [14]. Or, nous avons passé beaucoup trop de temps à essayer de reproduire exactement l'implémentation d'OpenAI, alors qu'il aurait été plus judicieux de passer plus de temps sur la recherche d'hyperparamètres afin d'améliorer la qualité des images produites par notre implémentation. D'autre part, nous ne nous sommes pas assez attardés sur la qualité des ensembles de données utilisés. Nous avons remarqué que l'ensemble CelebA original ne semble pas correspondre à l'ensemble utilisé dans l'article d'OpenAI. Les chercheurs d'OpenAI ont utilisé un "cropping" centré sur le visage afin de diminuer la résolution à 32x32 alors que nous avons utilisé un simple "downsampling" par interpolation. Ceci fait que nous avions probablement beaucoup plus de "bruit" dans les images causé par l'arrière-plan, alors que l'article InfoGAN semble présenter des images centrées entièrement sur un visage. Encore une fois, il aurait été intéressant d'avoir les détails sur le "pre-processing" utilisé par les chercheurs dans l'article, un autre exemple du défi de reproductibilité rencontré. Finalement, l'ensemble de données SVHN de TensorFlow Datasets semble être de très mauvaise qualité, présentant plusieurs images floues avec une luminosité très sombre. Nous aurions ainsi dû nous pencher sur le "pre-processing" impliqué par la librairie TensorFlow Datasets afin de voir si notre source de données était adéquate.

## 6 Conclusion

En sommes, nous pouvons affirmer avoir eu du succès à reproduire les résultats de l'article InfoGAN pour l'ensemble MNIST. Bien que les images obtenues par notre générateur soient de moins bonnes qualité que celles présentées dans l'article, nous avons obtenu le même comportement pour les variables latentes utilisées. Pour les ensembles SVHN et CelebA, nous avons eu énormément de difficultés à obtenir des résultats décents, à la fois pour les images générées que pour les manipulations de codes latents. Pour SVHN, il était impossible d'interpréter l'influence de la variable catégorique. Pour CelebA, la variable catégorique semble influencer la chevelure des visages générées ce qui concorde avec une partie des résultats du papier. Il est fort probable que la faible qualité des images générées par le générateur pour SVHN et CelebA cause une mauvaise utilisation des variables conditionnelles latentes.

Pour conclure, nous pouvons affirmer que nous avons rencontré un gros problème de reproductibilité en tentant

d'implémenter l'algorithme proposé InfoGAN. Bien que le code original d'OpenAI soit open source, et qu'un minimum d'effort a été fait pour documenter leur version, nous n'avons pas été capable d'exécuter leur code, et ce même avec 2 finissants en génie logiciel dans l'équipe. La qualité du code est également assez faible, de meilleures pratiques de conception logicielle auraient pu être utilisées au moment de la publication d'un papier de telle ampleur. Notre situation reflète très bien le problème de reproductibilité et de manque de transparence qui existe dans le domaine de recherche en intelligence artificielle et qui sévit depuis quelques années [15]. Bien qu'il soit honorable pour les auteurs de faire partie du maigre 13% des chercheurs en IA qui publient leur codes de recherche en 2016 [16], il serait également préférable que ce code soit capable de reproduire les résultats de l'article de manière simple et rapide.

## References

- [1] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.
- [2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.
- [3] Scott Reed, Kihyuk Sohn, Yuting Zhang, and Honglak Lee. Learning to disentangle factors of variation with manifold interaction. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page II–1431–II–1439. JMLR.org, 2014.
- [4] Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. Deep convolutional inverse graphics network, 2015.
- [5] Guillaume Desjardins, Aaron Courville, and Yoshua Bengio. Disentangling factors of variation via generative entanglement, 2012.
- [6] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets, 2016.
- [7] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. <https://arxiv.org/pdf/1605.09674.pdf>, 2017.
- [8] Jason Brownlee. How to develop an information maximizing gan (infogan) in keras. <https://machinelearningmastery.com/how-to-develop-an-information-maximizing-generative-adversarial-network-infogan-in-keras/>, 2021.
- [9] EmilienDupont. Keras infogan (work in progress). <https://github.com/EmilienDupont/infogan/tree/88bc0ef33ff76d3d96296ed9352f681fe59f4ea8>, 2021.
- [10] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

- [11] Yuval Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. 2011.
- [12] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
- [13] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.
- [14] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.
- [15] Benjamin Haibe-Kains, George Alexandru Adam, Ahmed Hosny, Farnoosh Khodakarami, Levi Waldron, Bo Wang, Chris McIntosh, Anna Goldenberg, Anshul Kundaje, and et al. Transparency and reproducibility in artificial intelligence. *Nature*, 586(7829):E14–E16, Oct 2020.
- [16] Papers with code. Trends: Code availability. <https://paperswithcode.com/trends>, 2021.