
# Projet de recherche

Xiang LIU

Yitong HE

12/04/2018

École Centrale Lyon

# Table des matières

# La reference (contexte)

**Image-to-Image Translation with Conditional Adversarial Networks**
Il étudie les réseaux conditionnels en tant que la solution générale aux problèmes de traduction d'images en images. Ces réseaux apprennent le mappage entre l'image d'entrée et l'image de sortie. Après, il étudie la fonction de perte pour entraîner cette cartographie. Donc il est possible de trouver les fonctions de perte différents à chaque problème. Il démontre que cette méthode est utile pour synthétiser des photos à partir de cartes d'étiquettes, reconstruire des objets à partir de bord, et effectuer des images couleur entre les tâches.

**A non-local algorithm for image denoising**
Il propose une méthode de bruit pour évaluer les performances des méthodes de débruitage d'image numérique. Il calcule et analyse le bruit pour l'algorithme de débruitage à savoir les filtres de lissage locaux. Et puis, il propose NL-means sur la base de la moyenne non locale de tous les pixels de l'image. Enfin, il introduit quelques expériences.

**DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs**
Il utilise l'apprentissage profonde pour segmenter les images sémantiques. La convolution atrophique nous permet de contrôler la résolution des réponses des fonctions dans les réseaux de neurones convolutionnels profonds. Il élargit le champ de vision du filtre afin de pouvoir combiner des contextes plus grands sans augmenter le nombre de paramètres ou la quantité de calcul. Deuxièmement, il propose une façon pour segmenter les objets à plusieurs échelles. Troisièmement, en combinant le DCNN avec le modèle de carte de probabilité, il a amélioré la localisation des limites d'objets.

# Scrapping et constitution du jeu de donn ées

Le script est un robot internet qui capture automatiquement les informations Web selon certaines règles. Notre script ouvert le navigateur que nous choisissons. Il recherche les image sur l'image de Google avec le mot clé. Après monter la fin de résultat, nous obtenons un navigateur avec les images suffisantes. Par utiliser la code de source de la page Web, nous pouvons obtenir l'adresse de la photo et la télécharger. Le script permette de scrapper à la fois une image et un json. Nous choisissons les images en PNG pour bien accorder à notre programme après. Nous nommons les images comme requête_numérotation.format (Par exemple: bar_chart_20.jpg) pour traiter les images en même temps.
Avec un nouvel type de script, on peut annuler la limite sur le nombre d'images. Pour éviter d'être repéré comme un bot, on utilise une façon de scrolling avec le driver pour balancer le chrome et les paramètres de gestion du temps.
Enfin, nous obtenons les images suivant par le mot clé (histogramme).

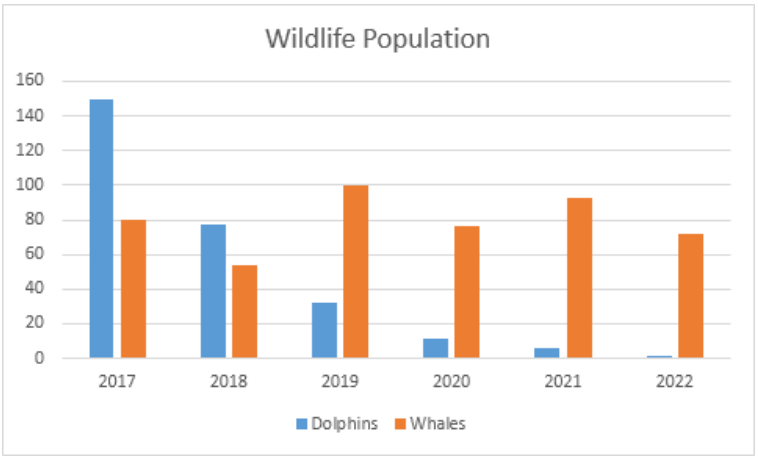



image 1 l'image d' histogramme

Avec la même façon, nous pouvons obtenir les images de Graphique linéaire et Dot plot.

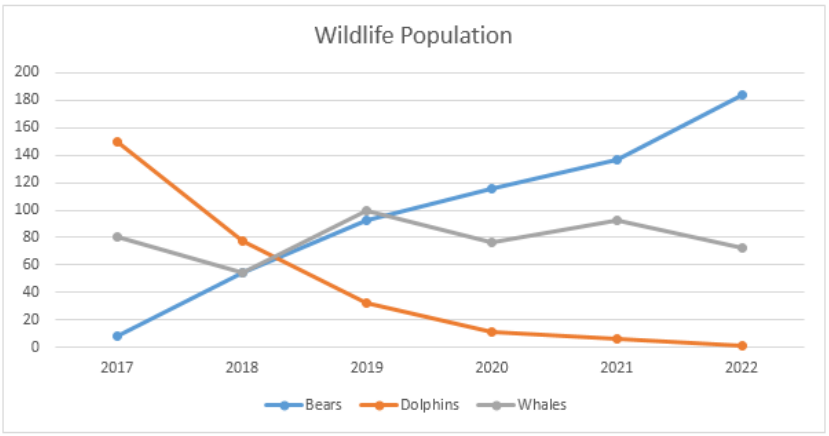



image 2 l'image de Graphique linéaire

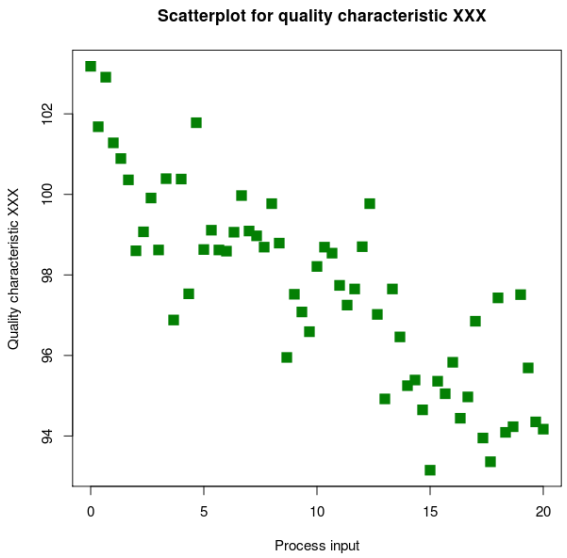



image 3 l'image de dot plot

Nous avons obtenu un groupe d'image. Après on va les traiter pour traîner notre programme.

# Extraire les contours d'images

Pour analyser les images, nous devons extraire les contours d'images ce qui nous permet d'avoir un ensemble d'images au bon format pour notre réseau de neurones. Le programme est divisé par trois fonctions.

D'abord, nous diminuons la taille d'image en 256x256 pix avec le code en resize.cpp. Nous transformons les images de PNG en JPG.

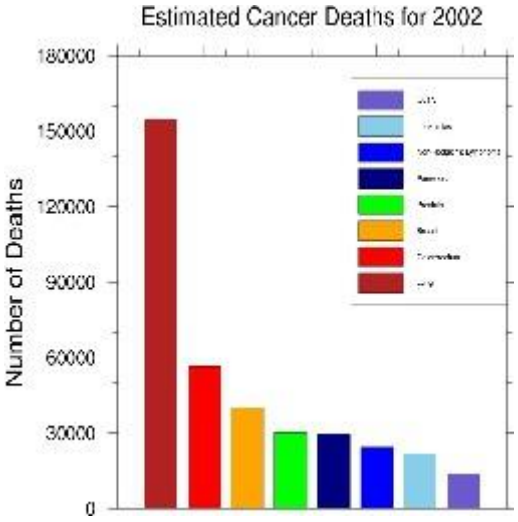



image 4 l'image en 256 to 256 pix

Ensuite, nous Extrayons les contours d'image comme le fil blanc sur le papier noir avec le code en findcontour.cpp. Enfin, nous posons les résultats de l'étape 1 et étape 2 dans la même image avec le code en combine_A_and_B.py.

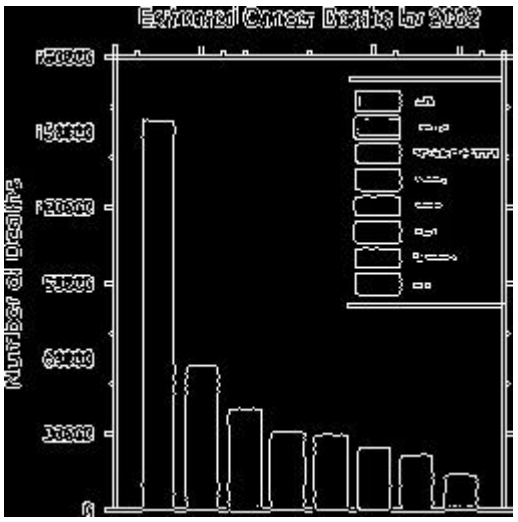


image 5 le contour d'image

Les deux premiers peut être utilisés directement ou en changeant les paramètres dans le code. Le dernier doit être utilisé selon l'implémentation au-dessus.

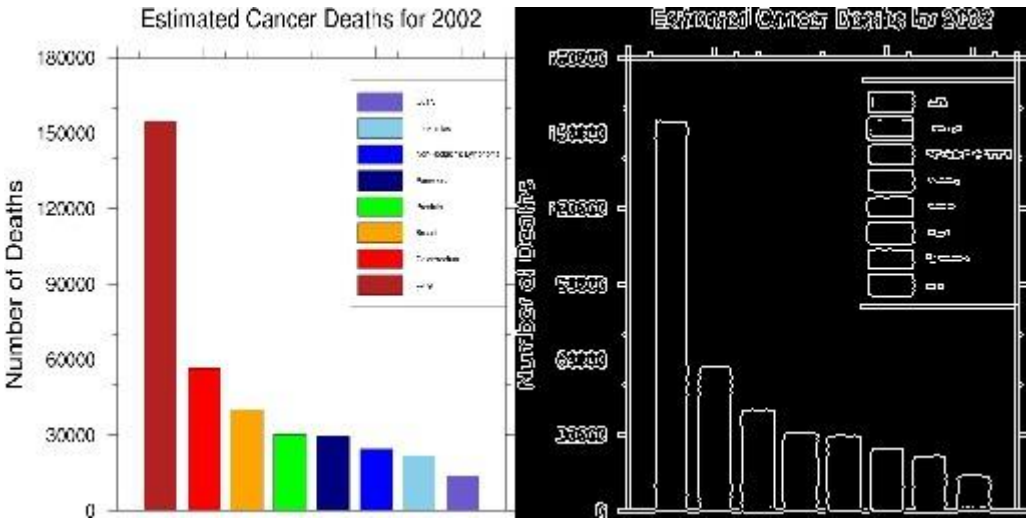



image 6 la combinaison d'image (le jeu de données)

# Projet Pix2Pix

## La méthode GAN

En intelligence artificielle, les réseaux antagonistes génératifs (en anglais generative adversarial networks ou GAN) sont une classe d'algorithmes d'apprentissage non-supervisé. Ils permettent de générer des images avec un fort degré de réalisme.

Le GAN commence par la thèse de lan Goodfellow. On suppose qu'il existe deux réseaux, G (Générateur) et D (Discriminateur).

G est un réseau qui génère les images. Il reçoit un bruit aléatoire z et génère une image par ce bruit, noté G (z). D est un réseau discriminant qui détermine si une image est "réelle". Ses paramètres d'entrée sont x. x représente une image. La sortie D (x) représente la probabilité que x est une image réelle. Si elle vaut 1, cela signifie que 100% est une image réelle. Si la sortie est 0. Cela signifie qu'elle ne peut pas être vraie.

Dans le processus d'apprentissage, l'objectif de G est de générer des images plus réelles pour tromper le réseau D. Le but de D est de séparer les images générées par G et des images réelles. De cette façon, G et D constituent un processus de jeu.

Dans l'état idéal, G peut générer une image G (z) presque réelle. Pour D, il est difficile de déterminer si l'image générée par G est vraie, donc D (G (z)) = 0.5.

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

Nous voyons que le premier symbole de la formule est min_G. G s'attend à ce que D (G (z)) soit plus grand et V (D, G) devienne plus petit. Si D est plus fort, D (x) doit être plus grand et D (G (x)) doit être plus petit. À ce moment, V (D, G) augmentera. Par conséquent, on prend max_D.

On renouvelle D avec augmentation de gradient stochastique pour mettre V(G,D) plus grand.

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

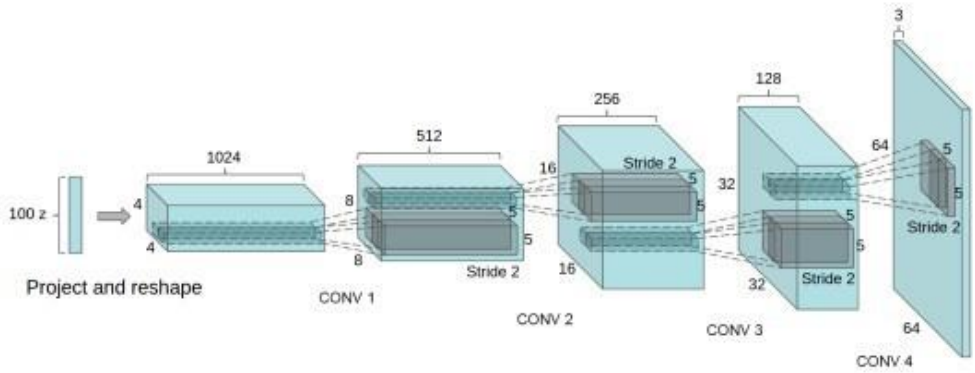
On renouvelle G avec descente de gradient stochastique pour mettre V(G,D) plus petit.

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} log\left(1 - D\left(G(z^{(i)})\right)\right)$$

Après on recommence les deux étapes.

# La méthode DCGAN

On veut combiner le CNN et GAN. Il remplace G et D avec deux réseaux de neurones convolutif (CNN). Cependant, DCGAN a apporté quelques changements à la structure du réseau de neurones convolutionnels pour améliorer la qualité des échantillons et la vitesse de convergence.
On annule toutes les couches de regroupement. Dans le réseau G, une couche convolutionnelle transposée est utilisée pour le suréchantillonnage. Dans le réseau D, la convolution avec stride est utilisée à la place des couches de regroupement. On utilise la normalisation par lots dans D et G. On supprime la couche FC et rendre le réseau du réseau convolutionnel complet. ReLU est utilisé comme une fonction d'activation dans le réseau G, mais la dernière couche utilise tanh. On utilise LeakyReLU comme fonction d'activation dans le réseau D




# Le projet

Ce répertoire permet à partir d'une base de données d'images contenant 3 classes différentes (bar chart, line chart et scatter plot) d'entraîner un modèle de réseaux de GANs et de générer les images sur la base de la même style par le modèle entraînée. Avec l'extraire des contours d'images, on combine les image et les contours comme le jeu de données d'entraîner et du test.
On a mis 4 dossiers dans ce répertoire :

1. google_pictures

Il contient 3 types des images concernées (bar chart, line chart et scatter plot) obtenu par git googlescrapper.

2. data

Il contient 3 dossiers d'images de bar chart, pour entraîner le réseau de GAN, et puis tester le modèle finale.

3. pix2pix-master

Il contient le git pix2pix.

4. résultat

Il contient le premier résultat obtenu à la base du jeu de donnée.

# Lancement

Afin d'installer l'ensemble des packages nécessaires au fonctionnement du script, exécuter **resize.cpp** pour changer la taille des images. Et puis exécuter **findcontour.cpp** pour extraire les contours sous forme d'image blanc et noir. Dans le terminal, créér un dossier **/path/to/data** avec sous-dossiers A et B. A et B doit contenir ses propres sous-dossiers train, val, test, etc. Mettre les images du style A dans **/path/to/data/A/train**. Mettre les images du style B dans **/path/to/data/B/train**. Répéter pour les autres (val, test, etc). Et puis executer la commande ci-dessous:

*python combine_A_and_B.py --fold_A /path/to/data/A --fold_B /path/to/data/B --fold_AB /path/to/data*

Après l'exécution du script, 3 fichiers "test", "train" et "val" sont créés. Celui-ci contient le jeu de donnée pour être entraîné et testé. En fin, on suit le lancement de git pix2pix à obtenir le résultat avec l'algorithme de GAN.

À partir du dossier **/data**, on répart nos images. Celui-ci sera lu pour entraîner le réseau de neurones. À la fin de l'entraînement, qui peut être plus ou moins long suivant les paramètres que vous avez choisi, le modèle est enregistré. Il est ensuite utilisé automatiquement sur le dossier **/test** créé ultérieurement et en créant un dossier results. Dans son sous-dossier **latest_net_G_val**, vous pouvez voir les résultats dans le web **index.html**. Comme chaque variable intermédiaire est enregistrée (modèle GAN entraîné), il est possible de les réutiliser à générer des images sans réaliser toutes les étapes.

# Le résultat

**La première fois**

Les paramètres clés sont dans le tableau au-dessous.

| | |
|---|---|
| batchSize (number of image in batch) | 1 |
| fineSize(crop the image input to this size) | 256 |
| Dataset Size | 5 |
| niter (number of iter at starting learning rate) | 200 |
| Ndf(number of discrim filters in first conv layer ) | 64 |
| Ngf(number of gen filters in first conv layer) | 64 |

On obtient le résultat au-dessous.

| | |
|---|---|
| Err_G | 1.1576 |
| Err_D | 0.5484 |
| ErrL1 | 0.0752 |

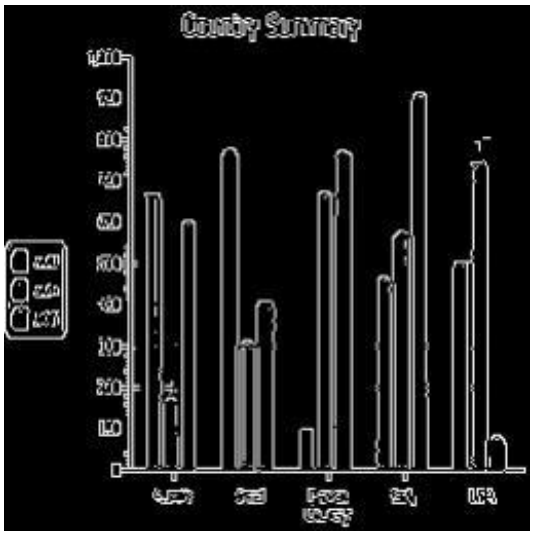



image 7 L’image d’entrée

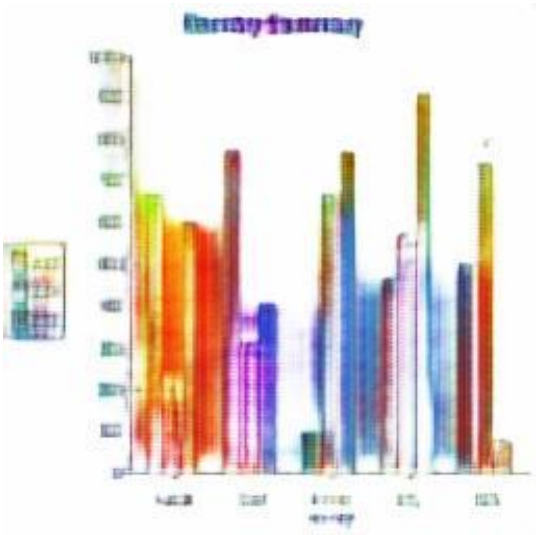


image 8 L’image de sortie

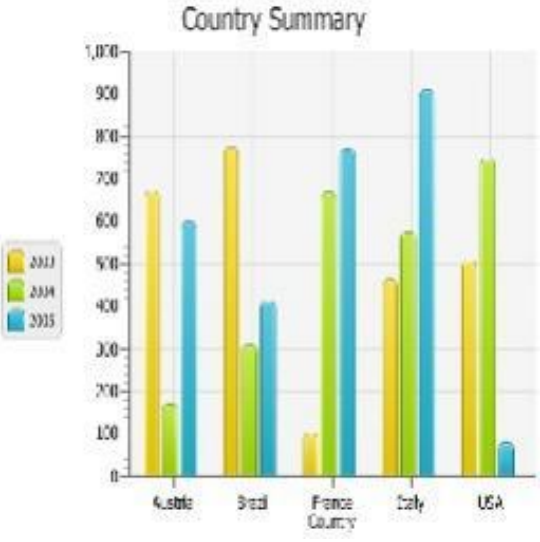



image 9 L’image original

Dans cette combinaison d'images, l'image de sortie est presque comme l'image original. Nous pouvons clairement distinguer les parties de l'histogramme (en-têtes, axes, coordonnées, barres de couleur et légendes). Autrement dit, les gens peuvent distinguer cet image.
Cependant, les couleurs de remplissage de la carte ne peuvent pas toujours rester dans la plage de contour. Certaines des couleurs se répandent sur le contour, entraînant une distorsion de l'image. En raison d'une certaine distorsion des contours qui ont traités précédemment, les détails de l'image présentent également un changement. Il faut 23 seconds pour calculer chaque image.

**La deuxième fois**

On change les images originales en 128x128 pix pour tester la qualité des images et la vitesse d'opération.

| | |
|---|---|
| batchSize (number of image in batch) | 1 |
| fineSize(crop the image input to this size) | 128 |
| Dataset Size | 5 |
| niter (number of iter at starting learning rate) | 200 |
| Ndf(number of discrim filters in first conv layer ) | 64 |
| Ngf(number of gen filters in first conv layer) | 64 |

Il faut 6 seconds pour calculer chaque image. On compare la même image dans les conditions différentes.

| | |
|---|---|
| Err_G | 0.8073 |
| Err_D | 0.6730 |
| ErrL1 | 0.0510 |

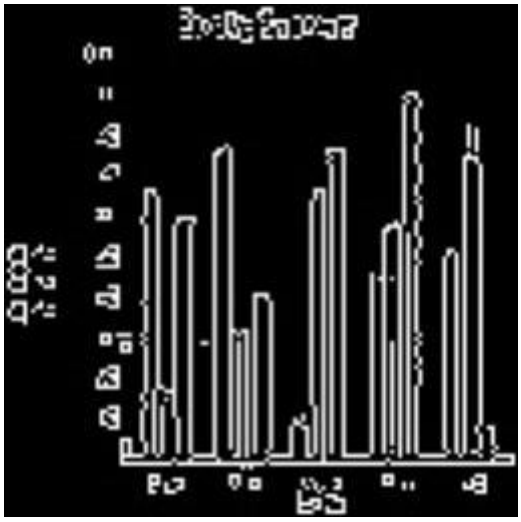


image 10 l'image d'entrée en 128 pix

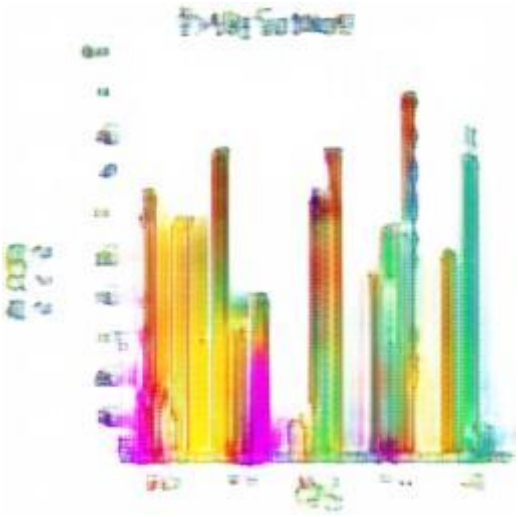


image 11 L’image de sortie en 128 pix

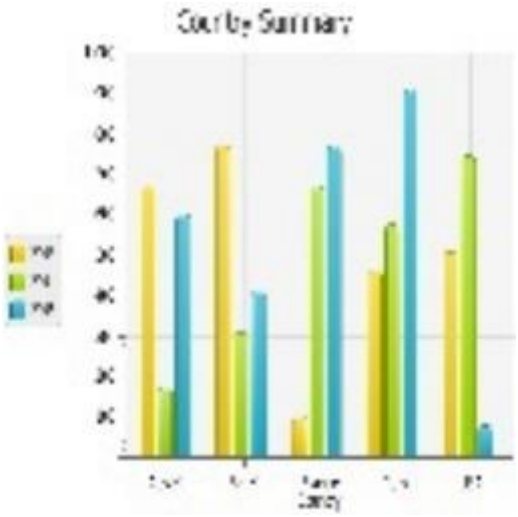


image 12 L’image originale

Nous pouvons voir que les textes et les axes de l'image originale deviennent flous, ce qui met le contour plus faible. La partie colorée ne déborde pas beaucoup et le résultat final ne conserve que les informations de barre coloré.

**La troisième fois**

On change la batchSize et le Dataset Size pour tester la qualité des images et la vitesse d’opération.

| | |
|---|---|
| batchSize (number of image in batch) | 2 |
| fineSize(crop the image input to this size) | 128 |
| Dataset Size | 10 |
| niter (number of iter at starting learning rate) | 200 |
| Ndf(number of discrim filters in first conv layer ) | 64 |
| Ngf(number of gen filters in first conv layer) | 64 |

On obtient le résultat :

| | |
|---|---|
| Err_G | 0.5373 |
| Err_D | 0.6497 |
| ErrL1 | 0.1164 |

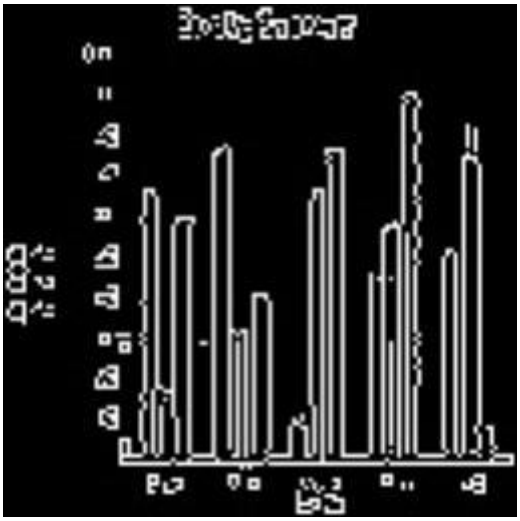


image 13 l'image d'entrée en batchSize 2

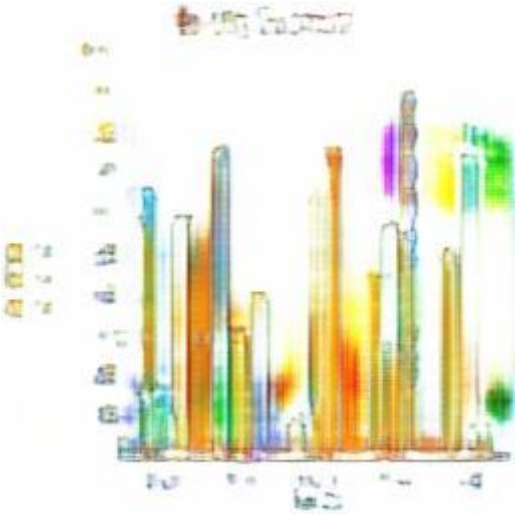


image 14 l'image de sortie en batchSize 2

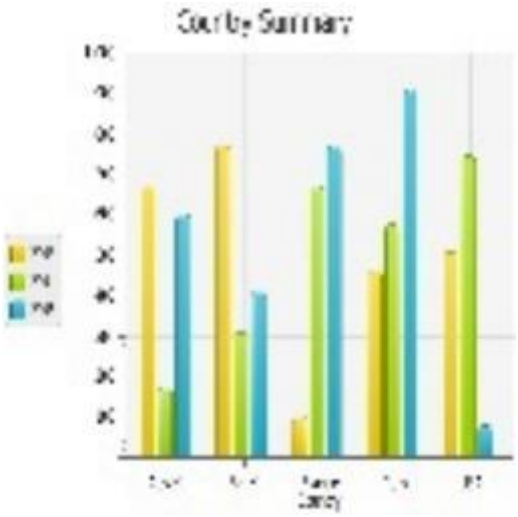


image 15 L’image original en batchSize 2

On trouve que si on change la batchSize et le Dataset Size. Les images deviennent plus faibles.

# La conclusion

Nous utiliserons le script issu du git image-scrapers afin de peupler notre base de données. Celui-ci permet de récupérer des images issues de google.

À partir d'une base de données contenant 3 classes d'images de tailles et de formats quelconques, on redimensionne chaque image à la taille 256x256 et on convertit en JPG. Et puis, on extrait les contours sous forme d’image noir et blanc. En fin, on les combine comme le jeu de données d’entraîner et du test.

Avec la méthode GANs, on traîne deux réseaux, G (Générateur) et D (Discriminateur) pour générer les image plus réel. Le modèle de CNN peut améliorer l’algorithme. Après changer la clarté d’image, on obtient les résultats faibles. Mais il reste les formes principales.

A cause de limite de mémoire interne, le modèle est faible.