Ce rapport

présente la thématique, l'état de l'art, la publication référente et la démonstration de faisabilité

Introduction

Une recherche avancée dans le domaine du deep learning a mis en lumière un sujet encore inexploré au cours de cette formation.

Des réseaux de neurones sont utilisés dans le but d'améliorer la résolution spatiale d'une image, c'est-à-dire le niveau de détails d'une image pour une dimension donnée.

La super-résolution à image unique est ce processus qui restaure une image "high-resolution" (HR) à partir d'une image "low-resolution" (LR).

L'association de protection des animaux qui a fait appel à nous pour le projet précédent pourrait avoir besoin d'utiliser ses photos de chiens, mais certaines sont de mauvaises qualités visuelles après recadrement.

Mais pouvons-nous améliorer la qualité des photos de chiens ?

Etat de l'art

L'état de l'art du domaine est complexe, mais l'utilisation d'une structure particulière émerge à la lecture des récents papiers.

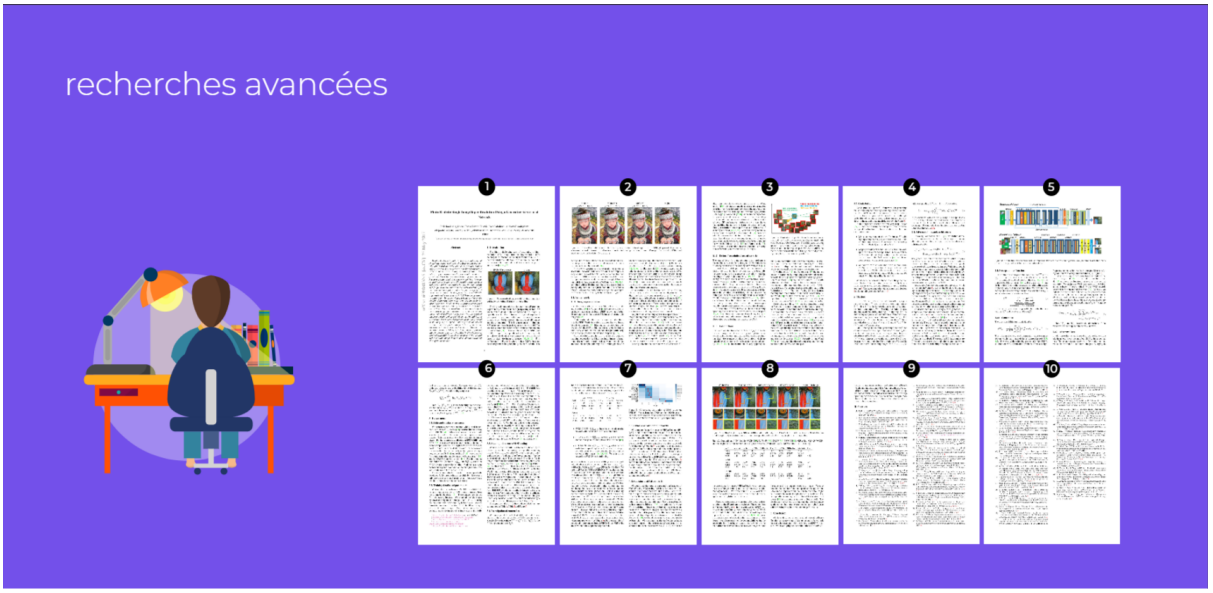
Ce sont les "Generative Adversarial Networks" (GANs) :

- [1] TWIST-GAN: Towards Wavelet Transform and Transferred GAN for Spatio-Temporal Single Image Super Resolution
- [2] ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks
- [3] SRGAN : Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network

L'article [1] le plus récent est le plus complexe et ne dispose pas d'implémentation. Nous allons donc privilégier l'étude de l'article [3] plus accessible et dont une implémentation du modèle SRGAN précurseur aux autres modèles a déjà été réalisée par un tiers.

L'étude de ce papier ainsi que d'autres sources vont nous permettre de comprendre le fonctionnement de l'algorithme, et ainsi de réaliser une démonstration de faisabilité (POC).

Lecture de la publication



page 1

Le SRGAN est un réseau adverse génératif capable de restaurer des images hautes résolutions réalistes pour un agrandissement de facteur x4.

Le critère d'optimisation des algorithmes de super résolution est généralement de minimiser une fonction de perte, l'erreur quadratique moyenne MSE entre les images restaurées (SR) et les images en haute résolution réelles (HR).

page 2

Le SRGAN est constitué d'un réseau neuronal résiduel pour générer des images.

Le critère d'optimisation est différent d'une MSE.

La mise en place d'un nouveau critère d'optimisation, une fonction de perte dite de perception utilisant les cartes de caractéristiques d'un réseau VGG.

On se concentre ici uniquement sur la super résolution à image unique.

page 3

La fonction de perte MSE a du mal à restaurer les détails de hautes fréquences, on génère des images floues et donc de qualité visuelle faible.

La fonction de perte MSE a du mal à restaurer les détails de hautes fréquences, on génère des images floues et donc de qualité visuelle faible.

L'emploi d'un GAN va permettre de surpasser ce problème.

page 4

Le GAN va pousser la reconstruction vers une solution spécifique et ainsi obtenir un rendu plus proche d'une image réelle, supprimant l'effet de flou.

Ce qui est confirmé par la note moyenne d'opinion (humain).

Le jeu d'entraînement est obtenu par une technique de réduction des images réelles HR en images LR.

Le GAN va consister à entraîner un réseau générateur G de paramètre $\theta = \{W,b\}$ et un réseau discriminateur D de paramètre $\theta = \{W,b\}$ de façon alternative.

On cherche à trouver les paramètres du générateur permettant d'obtenir le minimum de la fonction de perte de perception l sur l'ensemble N d'images générées SR et images réelles HR.

On cherche à générer les images les plus réalistes possible en haute résolution à partir du générateur mais aussi à différencier les images générées des images réelles grâce au discriminateur, l'optimisation va se réaliser en même temps sur les deux réseaux, ils sont adversaires dans un minimax problem.

page 5

La fonction de perte de perception est constituée d'une perte sur le contenu et d'une perte adverse.

Au lieu d'utiliser la fonction de perte MSE la plus utilisée, on définit la perte sur le contenu à l'aide d'un réseau VGG19 comme la distance entre les cartes de caractéristiques de convolution entre les images générées et les images réelles.

Un générateur qui est un réseau de neurones résiduels (ResNet) de B= 16 blocks.

Un discriminateur qui est un réseau de neurones convolutionnels similaire à un réseau VGG avec une fonction d'activation sigmoid en sortie

page 6

Un entrainement réalisé sur 350 000 images aléatoires depuis la base de données ImageNet.

Une réduction par interpolation bicubique.

Un entrainement par minibatch = 16 sur des bouts d'images du jeu d'entrainement de taille 96*96 pixels.

Le générateur est pré-entrainé avec une fonction de perte MSE sur le jeu d'entrainement et sert ainsi d'initialisation des poids du générateur pour l'entrainement du GAN.

page 7

L'évaluation des performances montre sur différents jeux d'entrainement, que le modèle SRGAN avec sa combinaison de fonction de perte de contenu sur VGG et fonction de perte adverse surpasse les autres en devenant l'état de l'art

page 8-9-10

conclusion et références

Expérimentation

Collecter les données

The Stanford Dogs dataset contains images of 120 breeds of dogs from around the world. This dataset has been built using images and annotation from ImageNet for the task of fine-grained image categorization.

lien : http://vision.stanford.edu/aditya86/ImageNetDogs/
nombre de races : 120
nombre d'images : 20 580

Pré-traitement

Les images sont stockées dans le dossier .stfdogs20580 du répertoire racine du projet.

Le dataset est téléchargé et préparé automatiquement grâce à une fonction implémentée par mes soins durant la phase de création du dataset de l'objet de classe STFDOGS20580

La préparation consiste à rassembler N images provenant du dataset en un jeu d'entrainement et un jeu de validation, chaque jeu devant avoir la version HR et la version LR.

La diminution de dimension (downsampling) est réalisée par interpolation bicubique à l'aide de la librairie OpenCV, d'un facteur 4 (= /16 pour une image 2d).

Modéliser

Le pré-entrainement du générateur sur l'erreur quadratique moyenne

pré-entrainement du générateur sur MSE loss

Entrée [9]:
```
pre_trainer = SrganGeneratorTrainer(model=generator(), checkpoint_dir=f'.ckpt/pre_generator')
pre_trainer.train(train_ds,
                  valid_ds.take(100),
                  steps=20000,
                  evaluate_every=1000,
                  save_best_only=False)

pre_trainer.model.save_weights(weights_file('pre_generator.h5'))
```

```
1000/20000: loss_train = 695.528, loss_val = 455.302155, psnr_val = 22.832457 (99.92s)
2000/20000: loss_train = 471.605, loss_val = 451.819458, psnr_val = 23.171009 (92.54s)
3000/20000: loss_train = 439.607, loss_val = 406.658142, psnr_val = 23.792353 (92.14s)
4000/20000: loss_train = 427.463, loss_val = 389.210876, psnr_val = 24.158199 (92.76s)
5000/20000: loss_train = 414.197, loss_val = 394.336487, psnr_val = 24.186033 (92.69s)
6000/20000: loss_train = 400.656, loss_val = 385.960175, psnr_val = 23.474201 (92.33s)
7000/20000: loss_train = 396.774, loss_val = 396.300751, psnr_val = 24.632648 (92.65s)
8000/20000: loss_train = 390.889, loss_val = 391.532104, psnr_val = 24.041616 (92.92s)
9000/20000: loss_train = 391.229, loss_val = 379.361420, psnr_val = 24.671635 (92.87s)
10000/20000: loss_train = 386.358, loss_val = 367.339905, psnr_val = 24.556221 (92.63s)
11000/20000: loss_train = 381.680, loss_val = 366.331238, psnr_val = 25.380821 (92.70s)
12000/20000: loss_train = 380.388, loss_val = 374.230377, psnr_val = 24.895226 (92.87s)
13000/20000: loss_train = 380.089, loss_val = 375.147003, psnr_val = 24.793724 (92.67s)
14000/20000: loss_train = 375.905, loss_val = 365.611511, psnr_val = 25.389008 (92.99s)
15000/20000: loss_train = 376.517, loss_val = 367.176575, psnr_val = 25.069117 (92.71s)
16000/20000: loss_train = 375.429, loss_val = 355.473328, psnr_val = 24.747904 (93.35s)
17000/20000: loss_train = 370.777, loss_val = 385.038513, psnr_val = 24.078451 (92.77s)
18000/20000: loss_train = 369.196, loss_val = 353.026093, psnr_val = 24.819422 (93.05s)
19000/20000: loss_train = 368.123, loss_val = 360.204041, psnr_val = 26.084274 (92.94s)
20000/20000: loss_train = 368.140, loss_val = 373.766479, psnr_val = 24.730148 (92.63s)
```

L'entrainement du générateur sur la perception et du discriminateur sur l'entropie croisée binaire

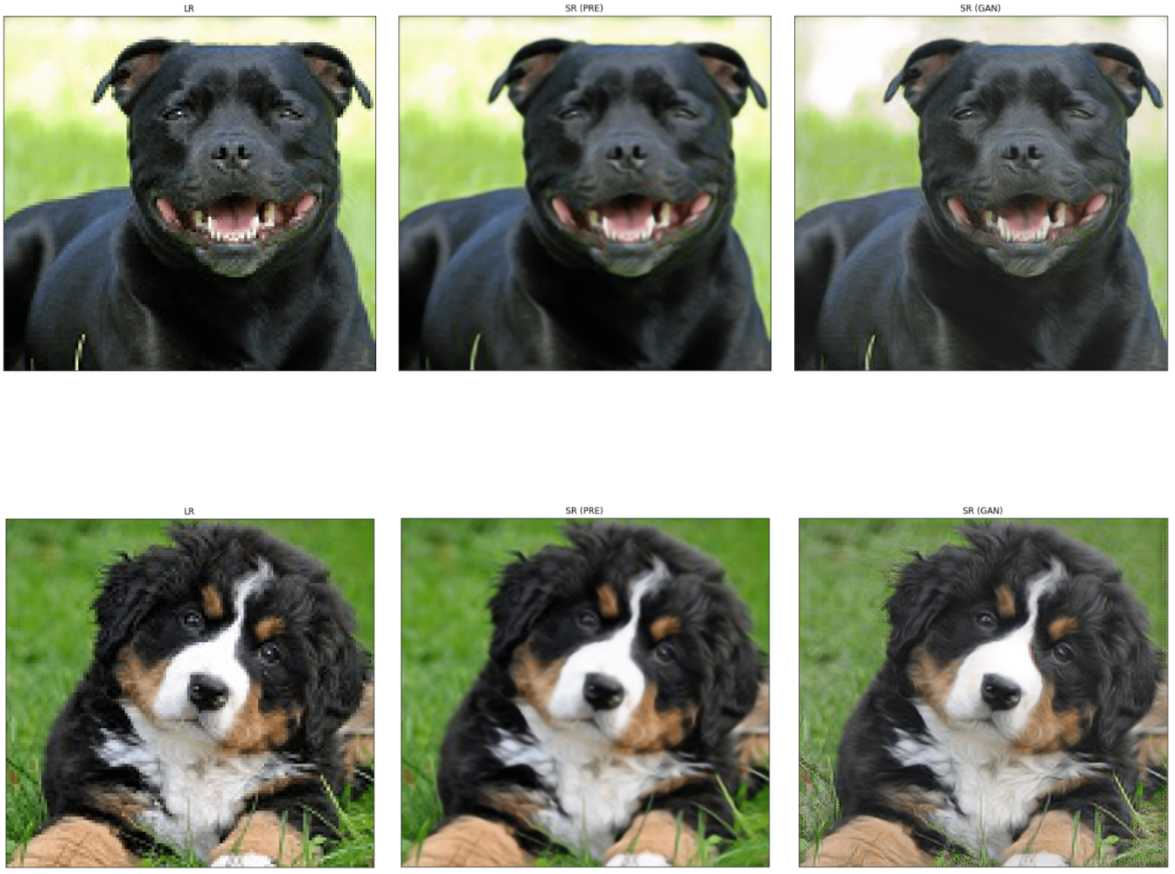
entrainement du générateur et discriminateur sur perception loss

Entrée [10]:
```
gan_generator = generator()
gan_generator.load_weights(weights_file('pre_generator.h5'))

gan_trainer = SrganTrainer(generator=gan_generator, discriminator=discriminator())
gan_trainer.train(train_ds,
                  steps=5000)
```

```
4050/5000, perceptual loss = 0.3276, discriminator loss = 0.2267
4100/5000, perceptual loss = 0.3371, discriminator loss = 0.3225
4150/5000, perceptual loss = 0.3145, discriminator loss = 0.0181
4200/5000, perceptual loss = 0.3320, discriminator loss = 0.1885
4250/5000, perceptual loss = 0.3208, discriminator loss = 0.0950
4300/5000, perceptual loss = 0.3415, discriminator loss = 0.0836
4350/5000, perceptual loss = 0.3392, discriminator loss = 0.1850
4400/5000, perceptual loss = 0.3131, discriminator loss = 0.2113
4450/5000, perceptual loss = 0.3118, discriminator loss = 0.1370
4500/5000, perceptual loss = 0.3289, discriminator loss = 0.2936
4550/5000, perceptual loss = 0.3318, discriminator loss = 0.0800
4600/5000, perceptual loss = 0.3271, discriminator loss = 0.1643
4650/5000, perceptual loss = 0.3284, discriminator loss = 0.1447
4700/5000, perceptual loss = 0.3018, discriminator loss = 0.5600
4750/5000, perceptual loss = 0.3063, discriminator loss = 0.0362
4800/5000, perceptual loss = 0.3249, discriminator loss = 0.0987
4850/5000, perceptual loss = 0.3119, discriminator loss = 0.3555
4900/5000, perceptual loss = 0.3115, discriminator loss = 0.0731
4950/5000, perceptual loss = 0.2988, discriminator loss = 0.0333
5000/5000, perceptual loss = 0.3192, discriminator loss = 0.1142
```


Démonstration de faisabilité



Conclusion

Nous avons réussi à réaliser une démonstration de faisabilité après la lecture d'un article de recherche sur la super résolution

L'implémentation du SRGAN fonctionne mais de nombreuses optimisations sont possibles en modifiant les hyper paramètres des modèles, en réalisant un entraînement complet et une augmentation du nombre d'images sélectionnées pour l'entrainement, et bien plus encore