Laboratoire 3

Noise2Noise

Durée : 2 séances

Par Christophe Lamarche

Mise à jour: 23 mai 2021

1 Description

L'utilisation de filtre pour retirer du bruit dans un signal est amplement vue dans le curriculum d'ingénierie électrique. Les filtres observés sont statiques et cernent des fréquences particulières. Pour le traitement et la réduction de bruit à l'intérieur d'images, l'ÉTS offre le cours ELE747 (École de Technologie Supérieure , ÉTS).

Ce laboratoire vise à initier l'étudiant à interpréter des papiers scientifiques pour reproduire des modèles neuronaux et d'introduire l'utilisation de banques de données publiques.

1.1 Exposé du problème

Le but de ce laboratoire est de réduire le bruit d'images en utilisant les technologies mises de l'avant par le papier de recherche **Noise2Noise** (Lehtinen et al., 2018). L'équipe du laboratoire de recherches chez Nvidia a utilisé l'architecture Unet (Ronneberger et al., 2015) afin de retirer le bruit d'images. La particularité de leur méthode vient à utiliser uniquement des données bruitées pour entraîner leur modèle (Lehtinen et al., 2018).

L'utilisation de données bruitées offre la possibilité d'entraîner des modèles pour des situations où la récolte de données sans bruit est difficile.

2 Conception de Noise2Noise

2.1 Base de données

Plusieurs services existent pour trouver et télécharger des bases de données. La compagnie **Google** offre un moteur de recherche pour les bases de données avec leur site web :

https://datasetsearch.research.google.com/

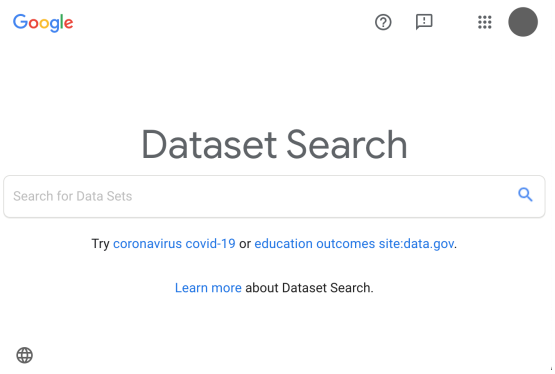

FIGURE 1 – Moteur de recherche pour des bases de données

À l'aide de ce site, il est possible de rechercher notre base de données d'images.

Dans notre cas, nous cherchons à télécharger plusieurs images. Par exemple, avec les mots clés **images** et **flickr**, un lien vers le site **Kaggle** est offert.

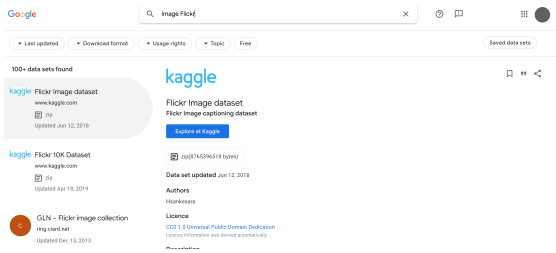

FIGURE 2 – Résultats pour les mots "Image Flickr"

Kaggle est un site offrant des services pour la communauté d'analyste de données (*data analyst*) et de scientifiques en apprentissage machine (*machine learning practitioners*). Les utilisateurs du site peuvent accéder et offrir des bases de données à la communauté. En plus, de donner accès à des *notebooks* tels que ceux produits avec *Google Colaboratory*, **Kaggle** offre aux utilisateurs de participer à des cours en ligne et des compétitions d'analyse de données.

2.1.1 Accéder à la clé de l'API de Kaggle

Les *notebooks* de *Google Colaboratory* sont configurés pour interfacer avec l'API de **Kaggle**. Pour ce faire, il est nécessaire de télécharger notre clé de l'API pour avoir accès aux fonctionnalités.

Sur le site web de **Kaggle**, connectez-vous ou inscrivez-vous pour un compte de Kaggle. Les comptes **Kaggle** sont gratuits et il est possible d'en créer un directement à l'aide d'un compte *Google* ou bien un compte *Facebook*.

Pour créer un compte, cliquer le bouton "*Register*" dans le coin supérieeur droit de la page d'accueil de **Kaggle**.

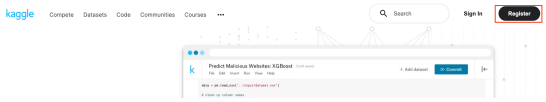

FIGURE 3 – Création d'un compte sur Kaggle

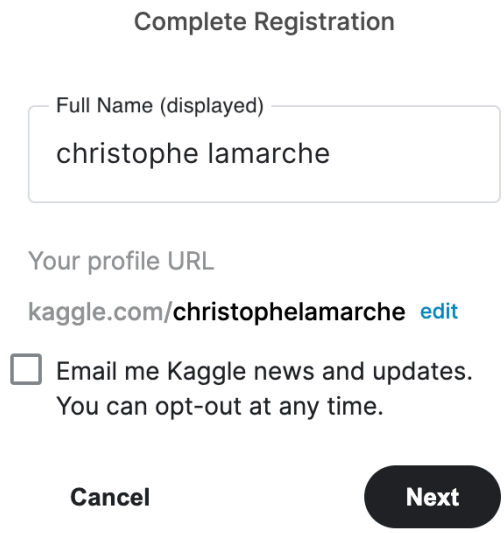

FIGURE 4 – Choisir le nom de l'utilisateur

À la suite que le compte soit fait, dirigez-vous vers les paramètres de compte situés sous la photo de profil.

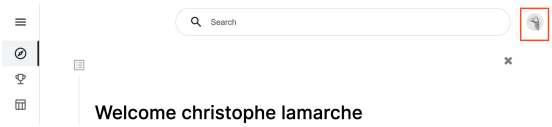

FIGURE 5 – Accéder les options de compte

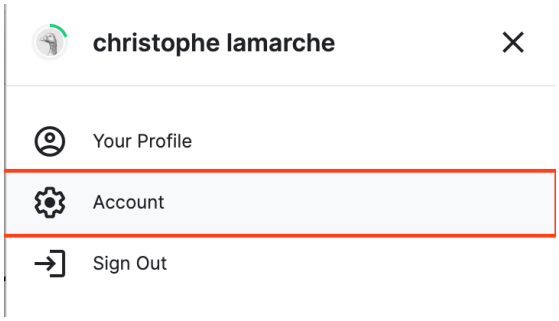

FIGURE 6 – Accéder aux paramètres de compte

Dans les paramètres de comptes, sous l'onglet comptes, déplacez-vous vers la section *API* et cliquez l'option "créer une clé API". Le fichier "kaggle.json" se téléchargera. Ce fichier contient la clé d'API pour travailler avec **Kaggle** à partir de Google Colaboratory et de la ligne de commande (*command line*). **Ce fichier devrait rester privé**. Afin d'être facilement accessible et privé, déposez ce document à l'intérieur de votre *Google Drive* sous un fichier n'ayant pas été partagé à d'autres utilisateurs.

2.1.2 Utilisation de l'API

Dans un nouveau *Notebook*, démarrer une cellule de code et copier le segment de code suivant (Segment de code 1 et 2).

```
from google.colab import drive
drive.mount('/content/drive')
```

Segment de codes 1 – Connecter le notebook avec Google Drive

```
# Mettre la cle API accessible pour le SDK de Kaggle
!kaggle -v &> ./file.tmp && rm ./file.tmp
!cp $chemin_de_la_cle_API$ /root/.kaggle/kaggle.json
!chmod 600 /root/.kaggle/kaggle.json

# Telecharger la base de donnees
!kaggle datasets download $nom-de-lutilisateur/nom-de-la-base-de-donnees$
# Decompresser les fichiers
!unzip -o $nom-de-la-base-de-donnees.*$
# Supprimer le fichier compresse
!rm ./*.zip
```

Segment de codes 2 – Télécharger la base de données avec Kaggle

Dans le segment de code 2, il est nécessaire de remplacer les informations entre les $$. La première information à inscrire est le chemin de la clé de l'API. Il est facile d'obtenir le chemin en sélectionnant le fichier dans le menu des fichiers.

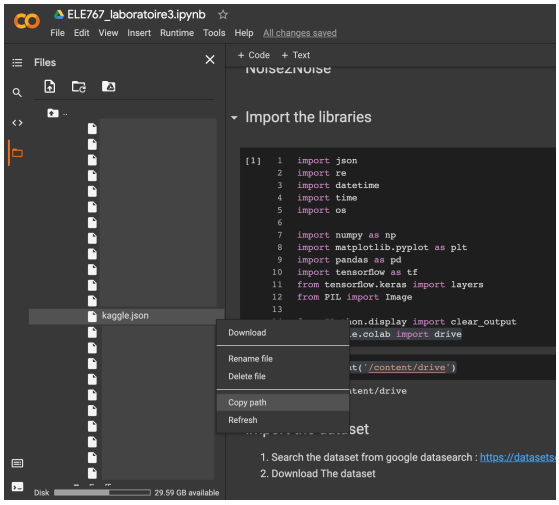

FIGURE 7 – Copier le chemin de la clé de l'API

Le chemin de la clé de l'API offert par le menu des fichiers doit être modifié afin d'être inséré dans la commande du segment de code 2. Pour ce faire, les espaces " " par une barre oblique inverse jointe d'un espace "\ ".

Note : Dans le segment de code 2, les lignes commençant par un point d'exclamation "!" correspondent à des commandes de terminal.

2.1.3 Lecture d'images

Pour lire les images, nous allons utiliser la librairie "Pillow (PIL)". Pour lire les images, nous importons le module "Image" de la librairie "PIL".

```
from PIL import Image
image = Image.open(chemin_image)

# Transformer l'image en matrice
import numpy as np
m_image = np.array(image)
```

Segment de codes 3 – Ouverture d'une image avec PIL

2.1.4 Concaténer plusieurs matrices d'images

La librairie *Numpy* offre la méthode "numpy.concatenate" pour joindre plusieurs matrices. Dans notre cas, nous voulons que les images soient accessibles en indexant une matrice. Alors, nous privilégions cette méthode à la méthode "numpy.stack".

```python
import numpy as np

# Creer les matrices a concatener
a = np.random.random((256,256,3))
b = np.random.random((256,256,3))

# Mauvaise concatenation pour notre situation
c = np.concatenate([a,b])
print(c.shape) # >> (512,256,3)

# Bonne concatenation pour notre situation
c = np.concatenate([a.reshape((1,*a.shape)),b.reshape((1,*b.shape))])
print(c.shape) # >> (2,256,256,3)
c = np.concatenate([c,b.reshape((1,*b.shape))])
print(c.shape) # >> (3,256,256,3)
```

Segment de codes 4 – Exemple d'utilisation de la méthode *numpy.concatenate*

2.2 Lecture et interprétation du papier scientifique

Afin de reproduire le modèle *Noise2Noise* (Lehtinen et al., 2018), il est nécessaire de lire le papier "Noise2Noise: Learning Image Restoration without Clean Data".

(*https ://arxiv.org/pdf/1803.04189.pdf*)

2.2.1 Conception du modèle

Dans le papier scientifique, les auteurs ont produit un tableau (tableau 2) afin de décrire l'architecture du modèle (Lehtinen et al., 2018). L'architecture est bien décrite dans la description du modèle et contient toutes les informations nécessaire pour la conception à l'aide de *Tensorflow*.

Dans cette description, les variables "m" et "n" correspondent aux nombres de dimensions de codage. Dans le cas des images téléchargées, le codage des couleurs est de trois dimensions (RGB).

Note : Afin de faciliter la création du modèle, je vous conseille d'utiliser la création de modèles sous la forme du segment de code 5 afin de réutiliser les entrées de couche. De plus, les couches "keras" à utiliser sont énumérées à la table. 1

```python
from tensorflow.keras import layers

def model(input)
    x_input = layers.Input(shape=input.shape)
    x_flatten = layers.Flatten()(x_input)
    x = layers.Dense(6)(x_flatten)
    x = layers.Concatenate(axis=-1)([x_flatten, x])
    return tf.keras.Model(x_input, x)
```

Segment de codes 5 – Création de modèle pour la réutilisation d'entrée de couche

Nom de la couche
Input
Conv2D
Concatenate
MaxPool2D
UpSampling2D

TABLE 1 – Couches à utiliser pour produire le U-Net

Layer (type)	Output Shape	Param #	Connected to
INPUT (InputLayer)	[(None, 256, 256, 3)	0	
ENC_CONV0 (Conv2D)	(None, 256, 256, 48)	1344	INPUT[0][0]
ENC_CONV1 (Conv2D)	(None, 256, 256, 48)	20784	ENC_CONV0[0][0]
POOL1 (MaxPooling2D)	(None, 128, 128, 48)	0	ENC_CONV1[0][0]
ENC_CONV2 (Conv2D)	(None, 128, 128, 48)	20784	POOL1[0][0]
POOL2 (MaxPooling2D)	(None, 64, 64, 48)	0	ENC_CONV2[0][0]
ENC_CONV3 (Conv2D)	(None, 64, 64, 48)	20784	POOL2[0][0]
POOL3 (MaxPooling2D)	(None, 32, 32, 48)	0	ENC_CONV3[0][0]
ENC_CONV4 (Conv2D)	(None, 32, 32, 48)	20784	POOL3[0][0]
POOL4 (MaxPooling2D)	(None, 16, 16, 48)	0	ENC_CONV4[0][0]
ENC_CONV5 (Conv2D)	(None, 16, 16, 48)	20784	POOL4[0][0]
POOL5 (MaxPooling2D)	(None, 8, 8, 48)	0	ENC_CONV5[0][0]
ENC_CONV6 (Conv2D)	(None, 8, 8, 48)	20784	POOL5[0][0]
UPSAMPLE5 (UpSampling2D)	(None, 16, 16, 48)	0	ENC_CONV6[0][0]
CONCAT5 (Concatenate)	(None, 16, 16, 96)	0	UPSAMPLE5[0][0] POOL4[0][0]
DEC_CONV5A (Conv2D)	(None, 16, 16, 96)	83040	CONCAT5[0][0]
DEC_CONV5B (Conv2D)	(None, 16, 16, 96)	83040	DEC_CONV5A[0][0]
UPSAMPLE4 (UpSampling2D)	(None, 32, 32, 96)	0	DEC_CONV5B[0][0]
CONCAT4 (Concatenate)	(None, 32, 32, 144)	0	UPSAMPLE4[0][0] POOL3[0][0]
DEC_CONV4A (Conv2D)	(None, 32, 32, 96)	124512	CONCAT4[0][0]
DEC_CONV4B (Conv2D)	(None, 32, 32, 96)	83040	DEC_CONV4A[0][0]
UPSAMPLE3 (UpSampling2D)	(None, 64, 64, 96)	0	DEC_CONV4B[0][0]
CONCAT3 (Concatenate)	(None, 64, 64, 144)	0	UPSAMPLE3[0][0] POOL2[0][0]
DEC_CONV3A (Conv2D)	(None, 64, 64, 96)	124512	CONCAT3[0][0]
DEC_CONV3B (Conv2D)	(None, 64, 64, 96)	83040	DEC_CONV3A[0][0]
UPSAMPLE2 (UpSampling2D)	(None, 128, 128, 96)	0	DEC_CONV3B[0][0]
CONCAT2 (Concatenate)	(None, 128, 128, 144	0	UPSAMPLE2[0][0] POOL1[0][0]
DEC_CONV2A (Conv2D)	(None, 128, 128, 96)	124512	CONCAT2[0][0]
DEC_CONV2B (Conv2D)	(None, 128, 128, 96)	83040	DEC_CONV2A[0][0]
UPSAMPLE1 (UpSampling2D)	(None, 256, 256, 96)	0	DEC_CONV2B[0][0]
CONCAT1 (Concatenate)	(None, 256, 256, 99)	0	UPSAMPLE1[0][0] INPUT[0][0]
DEC_CONV1A (Conv2D)	(None, 256, 256, 64)	57088	CONCAT1[0][0]
DEC_CONV1B (Conv2D)	(None, 256, 256, 32)	18464	DEC_CONV1A[0][0]
DEC_CONV1C (Conv2D)	(None, 256, 256, 3)	867	DEC_CONV1B[0][0]

Total params: 991,203
Trainable params: 991,203
Non-trainable params: 0

FIGURE 8 – Résumé du modèle à produire

2.2.2 Ajout de bruit

Le but du papier *Noise2Noise* (Lehtinen et al., 2018) est de débruiter les images en entrée. Afin d'obtenir des images davantage bruitées, il est intéressant de produire notre propre bruit. Le type de bruit choisi est intrinsèquement lié au choix de la fonction de perte qui sera utilisée lors de l'entraînement.

Le choix du type de bruit et de la fonction de perte est votre décision. Basé vous sur le papier *Noise2Noise* (Lehtinen et al., 2018) pour aider dans votre décision.

Le bruit consiste à l'ajout de valeur aléatoire au signal. Afin de produire des valeurs selon une distribution normale, la librairie *numpy* offre la méthode "**randn**" sous le module "**random**". Tandisque pour la production de bruit selon une distribution uniforme, la librairie *numpy* offre la méthode "**random**" et "**randint**" sous le module "**random**" dépendamment de la situation d'utilisation.

2.3 Entraînement du modèle

Avec les connaissances acquises lors des deux derniers laboratoires, nous allons faire l'entraînement du modèle. L'important dans ce laboratoire sont de bien reproduire les méthodes du papier *Noise2Noise* (Lehtinen et al., 2018).

Dans le cadre du laboratoire, vous devez utiliser la bonne fonction de perte (*loss function*) selon le type du bruit qui a été ajouté aux images.

L'idée principale derrière *Noise2Noise* est l'utilisation de données bruitées comme données cibles. Le principe est que dans une le cas où l'entraînement est fait sur plusieurs données bruitées différemment, le réseau de neurones apprendra une représentation moyenne du bruit similaire aux images non bruitées.

2.4 Performance du réseau débruiteur

Afin de déterminer la performance du réseau pour débruiter une image, nous pouvons utiliser l'équation de "*Peak Signal-to-Noise Ratio* (PSNR)" (Wikipedia contributors, 2021).

$$PSNR = 20 * log(\frac{MAX(Image_{originale})}{MSE(Image_{originale}, Image_{Bruitee})})$$

Cette équation offre une valeur élevée lorsque le bruit est faible et inversement, une valeur basse lorsque le bruit est élevé grâce au *MSE* au dénominateur. De cette manière, en utilisant le résultat du réseau de neurones sur une image précise, il est possible de comparer les performances durant l'entraînement.

3 À mettre dans le rapport

Avec le rapport, il faut remettre le notebook ('.ipynb') et le dossier "modeles"

Pour télécharger le fichier du notebook, cliquez l'option "Télécharger .ipynb" (*"Download .ipynb"*) sous l'onglet fichier.

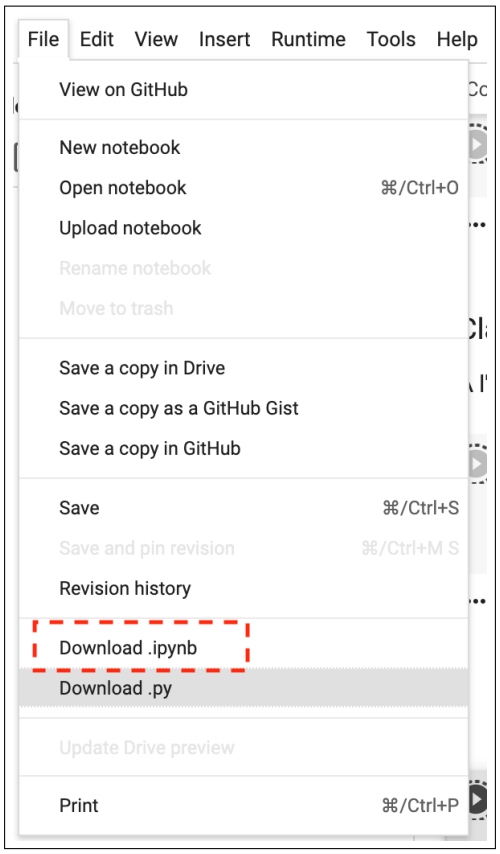

FIGURE 9 – Télécharger le notebook

Afin de télécharger le dossier "modeles", il faut le compresser auparavant. Pour se faire, ouvrer une nouvelle cellule de code et insérer la commande ci-dessous.

```
!zip -r modeles.zip ./modeles
```

Segment de codes 6 – Compresser un fichier sur une session

Faites attention au point d'exclamation [!] au début de la commande

Par la suite, il est possible de télécharger en se dirigeant sur l'icône de fichier dans la barre de gauche du notebook tel que représenté dans la figure 10.

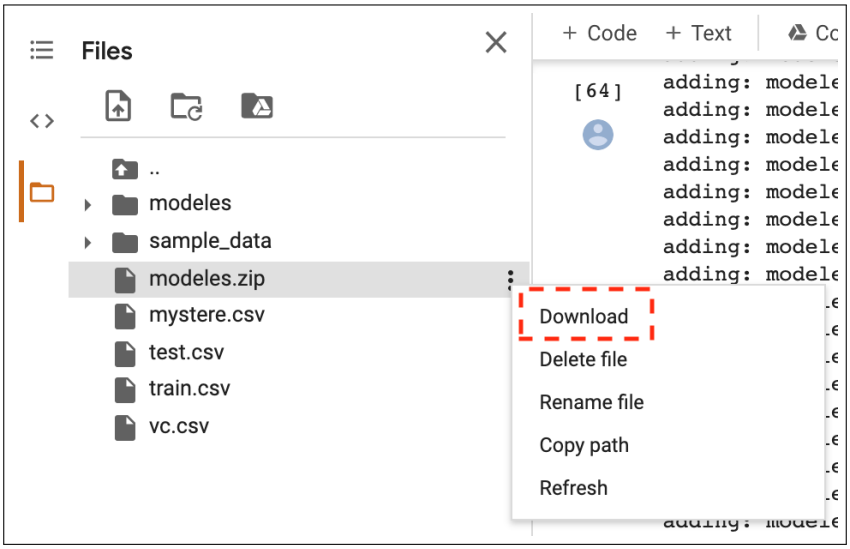

FIGURE 10 – Télécharger le dossier compressé

TABLE 2 – Contenu du rapport

Section	Taille minimum	Pondération
Introduction	3 lignes	5%
Théorie	5 lignes	20%
Résultats	5 lignes	40%
Conclusion	5 lignes	35%

3.1 Description des sections

3.1.1 Introduction

Dans l'introduction, il faut décrire le but du laboratoire et de résumer la procédure du laboratoire.

3.1.2 Théorie

Dans la section théorique, il est nécessaire d'expliquer les théories et les concepts qui sont mis de l'avant par le laboratoire.

3.2 Résultats

La section des résultats doit inclure les graphiques, les tableaux obtenus durant le laboratoire. Il est nécessaire d'inclure une figure décrivant l'entrainement pour chaque modèle. De plus, il est nécessaire d'expliquer le résultat observé à l'intérieur de la figure.

3.2.1 Conclusion

Finalement, la conclusion doit contenir un simple retour sur le but, une évaluation des résultats et une ouverture sur l'amélioration du laboratoire.

Références

Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise : Learning image restoration without clean data, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net : Convolutional networks for biomedical image segmentation, 2015.

Wikipedia contributors. Peak signal-to-noise ratio — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio. [En ligne ; Accéder le 18 May 2021].

École de Technologie Supérieure (ÉTS). Ele747 - analyse et traitement d'images. https://www.etsmtl.ca/etudes/cours/ele747/, 2021.