# debruitage_autoencodeur

January 5, 2025

## 1 Débruitage à l'aide d'un autoencodeur

#### 1.0.1 Quelques instructions :

Le devoir peut être réalisé seul ou en binôme et doit être rendu sous forme d'un pdf **prenom_nom.pdf** et **prenom1_nom1_prenom2_nom2.pdf** (dans le cas d'un binôme). Il est indispensable d'inclure les sorties des calculs dans le pdf.

#### 1.0.2 Introduction

Ce projet est inspiré d'un problème d'exploitation du texte des comptes rendus médicaux de la cohorte CKD-REIN. Ces derniers sont disponibles sous forme de scans en pdf avec du bruit à éliminer pour faciliter la lecture optique. Il est indispensable d'utiliser **google colab** pour réaliser ce projet avec une GPU.

Les données à utiliser sont disponibles ici. Il est indispensable de décompresser le dossier et de charger le dossier vers votre google drive.

Une **GPU** est indispensable pour ce projet. `Exécution -> Modifier le type d'exécution -> T4 GPU`

**Objectifs** - Comment définir un jeu de données personnalisé avec les mêmes transformations aléatoires - Comment construire une architecture d'autoencodeur : - Création d'un encodeur personnalisé - Création d'un décodeur personnalisé - Utilisation d'un réseau pré-entraîné comme encodeur - Entraîner et évaluer les autoencodeurs d'images

### 1.1 Se placer sur drive pour exécuter le notebook

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

### 1.2 Importation des librairies

[ ]:
```
import torch
import torchvision

from torch import nn
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets, transforms
from torch.utils.tensorboard import SummaryWriter
```

```
from torch import optim

from tqdm import tqdm
import os
import gc
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
from PIL import Image

%matplotlib inline
```

[ ]:
```
## commandes pour vider le cache de la gpu si nécessaire (pas indispensable)
gc.collect()
torch.cuda.empty_cache()
```

### 1.3 Une classe de paramétrisation

[ ]:
```
class ma_config:
    # training
    seed = 21
    num_epochs=20
    lr=0.0005
    batch_size=8
    clip_value=1
    logdir = 'logdir'

    # data
    data_dir='/content/drive/My Drive/documents_avec_bruits'
    eval_size=0.2
    stack_ratio=0.5
    transformed_size=(400, 400)
    device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
```

[ ]:
```
def set_seed(seed):
    np.random.seed(seed)
    torch.manual_seed(seed)

set_seed(ma_config.seed)
```

### 1.4 Préparation des données

Nous allons utiliser un jeu de données de la plateforme **Kaggle** pour éviter la perte de temps avec des questions réglementaires pour l'utilisation de bouts de comptes rendus de la cohorte CKD-REIN.

Le jeu de données `documents_avec_bruits` est composé des trois dossiers :

- `apprentissage_avec_bruit` - contient 144 images de documents bruités (froissés, rayés ou sales). Le texte des images est à peu près le même, mais chaque image a une police de caractères et un bruit différents. Le nom de l'image unique est `[numero_image].jpg`
- `apprentissage_sans_bruit` - contient les mêmes 144 images de documents, mais sans aucun bruit. Le nom de l'image unique est également `numero_image.jpg`, ce qui correspond aux images bruitées de `apprentissage_avec_bruit` -`test` - contient 72 images de documents bruités (froissés, rayés, sales). Le texte sur les images est différent de `apprentissage_avec_bruit` mais est le même dans toutes les images `test`. Le nom de l'image unique est `numero_image.jpg`

Il est utile de noter que la taille des images peut varier.

La plupart des images des jeux de données `apprentissage_avec_bruit`, `apprentissage_sans_bruit` et `test` sont de taille `(540, 420)`, mais environ 30% sont de taille `(540, 258)`.

[ ]:
```
noisy_folder_path = os.path.join(ma_config.data_dir, 'apprentissage_avec_bruit')
clean_folder_path = os.path.join(ma_config.data_dir, 'apprentissage_sans_bruit')
eval_folder_path = os.path.join(ma_config.data_dir, 'test')

images_names = os.listdir(noisy_folder_path)
eval_names = os.listdir(eval_folder_path)

images_names.sort()
eval_names.sort()
```

### 1.5 Visualisation d'exemples de de figures (avec et sans bruit)

[ ]:
```
def plot_example(idx):
    noisy_path = os.path.join(noisy_folder_path, images_names[idx])
    clean_path = os.path.join(clean_folder_path, images_names[idx])
    noisy = Image.open(noisy_path)
    clean = Image.open(clean_path)

    fig, ax = plt.subplots(1, 2, figsize=(8,4))
    ax[0].imshow(noisy, cmap='gray')
    ax[0].set_title('Noisy text')
    ax[1].imshow(clean, cmap='gray')
    ax[1].set_title('Clean text')
    plt.show()

plot_example(0)
plot_example(23)
plot_example(110)
```

### 1.6 Bilan des dimensions des images

Nos images ont des hauteurs différentes. Visualisons à l'aide d'un histogrammes des hauteurs des images de l'ensemble du jeu de données.

[ ]:
```
heights = []
for img in images_names:
    path = os.path.join(clean_folder_path, img)
    w, h = Image.open(path).size
    heights.append(h)
plt.figure(figsize = (6, 6))
plt.hist(heights, align='right')
plt.title('Distribution des hauteurs des images du jeu de données')
plt.show()
```

L'histogramme ci-dessus indique qu'il y a  50 images qui ont une hauteur beaucoup plus petite que le reste. Nous avons besoin d'une transformation pour corriger cette différence.

### 1.7 Transformations

Ici, nous allons traiter un jeu de données d'images **appariées** et appliquer les mêmes transformations aléatoires à l'image propre et à l'image bruitée afin de préserver la structure de l'ensemble de données.

Nous allons donc avoir besoin des transformations suivantes :

1. Une classe `PairedToTensor()` - fait appel à `transforms` pour transformer les images `noisy` et `clean` en `torch.Tensor`
2. Une classe `PairedStack(stack_ratio)` - si le rapport hauteur/largeur de l'image est inférieur au rapport `stack_ratio` donné, l'image est empilée verticalement avec elle-même (appliqué aux deux images à la fois).
3. Une classe `PairedRandomCrop(size)` - applique **le même** recadrage aléatoire avec une `taille` donnée à une image bruitée et à une image propre en même temps (utiliser une méthode du module `transforms`).
4. Une classe `PairedCenterCrop(size)` - applique un recadrage central avec la taille donnée à l'image bruitée et sa version propre en même temps (utiliser une méthode du module `transforms`).

**À faire :** (on peut utiliser `__init__` pour le constructeur d'une classe et `__call__` pour appeler l'objet avec un argument)

**Q1**. Compléter la classe `PairedToTensor`

**Q2**. Compléter la classe `PairedStack`

**Q3**. Compléter la classe `PairedRandomCrop`

**Q4**. Compléter la classe `PairedCenterCrop`

### 1.8 Fonction d'affichage de la première paire d'images d'un batch

[30]:
```python
# Fonction pour tracer la première paire d'un batch
def show_tensor_batch(batch, title=None):
    fig, ax = plt.subplots(1, 2, figsize = (10, 10))
    if title:
        fig.suptitle(title, y=0.7)
    ax[0].imshow(batch[0][0], cmap='gray')
    ax[1].imshow(batch[1][0], cmap='gray')

    plt.show()
```

[ ]:
```python
class PairedToTensor:
    def __call__(self, batch):
        noisy, clean = batch
        ### Début de votre code



        ### Fin de votre code
        return noisy, clean
```

[ ]:
```python
## tester la classe PairedToTensor
noisy_path = os.path.join(noisy_folder_path, images_names[0])
clean_path = os.path.join(clean_folder_path, images_names[0])
noisy = Image.open(noisy_path)
clean = Image.open(clean_path)
batch = noisy, clean

ToTensor = PairedToTensor()
print(f'Type image avec bruit : {ToTensor(batch)[0].type()}', \
      f'\n Type image sans bruit : {ToTensor(batch)[1].type()}')
```

[ ]:
```python
class PairedStack:
    def __init__(self, stack_ratio):
        self.stack_ratio = stack_ratio

    def __call__(self, batch):
        noisy, clean = batch
        ### Début de votre code


        ### Fin de votre code
        return noisy, clean
```

[ ]:
```python
# tester la classe PairedStack
noisy_path_1 = os.path.join(noisy_folder_path, images_names[0])
clean_path_1 = os.path.join(clean_folder_path, images_names[0])
```

```
noisy_1 = Image.open(noisy_path_1)
clean_1 = Image.open(clean_path_1)
batch_1 = noisy_1, clean_1

noisy_path_2 = os.path.join(noisy_folder_path, images_names[6])
clean_path_2 = os.path.join(clean_folder_path, images_names[6])
noisy_2 = Image.open(noisy_path_2)
clean_2 = Image.open(clean_path_2)
batch_2 = noisy_2, clean_2

ToTensor = PairedToTensor()
batch_1 = ToTensor(batch_1)
batch_2 = ToTensor(batch_2)

Stack = PairedStack(ma_config.stack_ratio)
batch_1 = Stack(batch_1)
batch_2 = Stack(batch_2)

print(f'dimension des images avec et sans bruit du premier batch: {batch_1[0].
 ↪shape}, {batch_1[1].shape}')
print(f'dimension des images avec et sans bruit du second batch: {batch_2[0].
 ↪shape}, {batch_2[1].shape}')

show_tensor_batch(batch_1, title="Premier batch:")
show_tensor_batch(batch_2, title="Second batch:")
```

[ ]:
```
class PairedRandomCrop:
    def __init__(self, size):
        self.size = size

    def __call__(self, batch):
        noisy, clean = batch
        ### Début de votre code



        ### Fin de votre code
        return noisy, clean
```

[ ]:
```
# tester la classe PairedRandomCrop
noisy_path = os.path.join(noisy_folder_path, images_names[0])
clean_path = os.path.join(clean_folder_path, images_names[0])
noisy = Image.open(noisy_path)
clean = Image.open(clean_path)
batch = noisy, clean

ToTensor = PairedToTensor()
```

```
batch = ToTensor(batch)

Stack = PairedStack(ma_config.stack_ratio)
batch = Stack(batch)

Crop = PairedRandomCrop(ma_config.transformed_size)
batch = Crop(batch)

show_tensor_batch(batch, title="Randomly cropped batch:")
```

**Résultat attendu :** *Les images doivent représenter la découpe de la même zone d'un document*

[ ]:
```
class PairedCenterCrop:
    def __init__(self, size):
        self.size = size

    def __call__(self, batch):
        noisy, clean = batch
        ### Début de votre code
        ### Fin de votre code
        return noisy, clean
```

[ ]:
```
# Tester la classe PairedCenterCrop

noisy_path = os.path.join(noisy_folder_path, images_names[0])
clean_path = os.path.join(clean_folder_path, images_names[0])
noisy = Image.open(noisy_path)
clean = Image.open(clean_path)
batch = noisy, clean

ToTensor = PairedToTensor()
batch = ToTensor(batch)

Stack = PairedStack(ma_config.stack_ratio)
batch = Stack(batch)

Crop = PairedCenterCrop(ma_config.transformed_size)
batch = Crop(batch)

show_tensor_batch(batch, title="Center cropped batch:")
```

**Résultat attendu:** *Les images doivent représenter la zone centrale d'un document*

### 1.9 Tout mettre ensemble

Maintenant nous allons `Composer` les transformations pour les jeux de données d'apptentissage et évaluation

**Pour l'apprentissage:** 1. Transformer les images d'entrée en `torch.Tensor` 2. Si l'image est

trop large, l'empiler verticalement avec elle-même 3. Appliquer le même recadrage aléatoire aux images bruitées et propres pour augmenter les données et obtenir la taille (400, 400).

**Pour l'évaluation:** 1. Transformer les images d'entrée en `torch.Tensor` 2. Si l'image est trop large, l'empiler verticalement avec elle-même 3. Appliquer un recadrage central aux images bruitées et propres pour obtenir une taille de (400, 400) (nous n'avons pas besoin d'aléatoire dans le cas des données d'évaluation).

[ ]:
```
train_transforms = transforms.Compose([
    PairedToTensor(),
    PairedStack(ma_config.stack_ratio),
    PairedRandomCrop(ma_config.transformed_size),
])

eval_transforms = transforms.Compose([
    PairedToTensor(),
    PairedStack(ma_config.stack_ratio),
    PairedCenterCrop(ma_ma_config.transformed_size)
])
```

### 1.10 Jeux de données d'apprentissage et évaluation

[ ]:
```
class DocumentsDataset(Dataset):
    def __init__(self, image_names, noisy_folder, clean_folder,␣
 ↪transforms=None):
        self.noisy_path = [os.path.join(noisy_folder, img) for img in␣
 ↪image_names]
        self.clean_path = [os.path.join(clean_folder, img) for img in␣
 ↪image_names]
        self.transforms = transforms

    def __len__(self):
        return len(self.noisy_path)

    def __getitem__(self, idx):
        noisy = Image.open(self.noisy_path[idx])
        clean = Image.open(self.clean_path[idx])
        batch = (noisy, clean)

        if transforms is not None:
            batch = self.transforms(batch)

        return batch
```

[ ]:
```
train_names, eval_names = train_test_split(
    images_names, random_state=ma_config.seed, test_size=ma_config.eval_size)
```

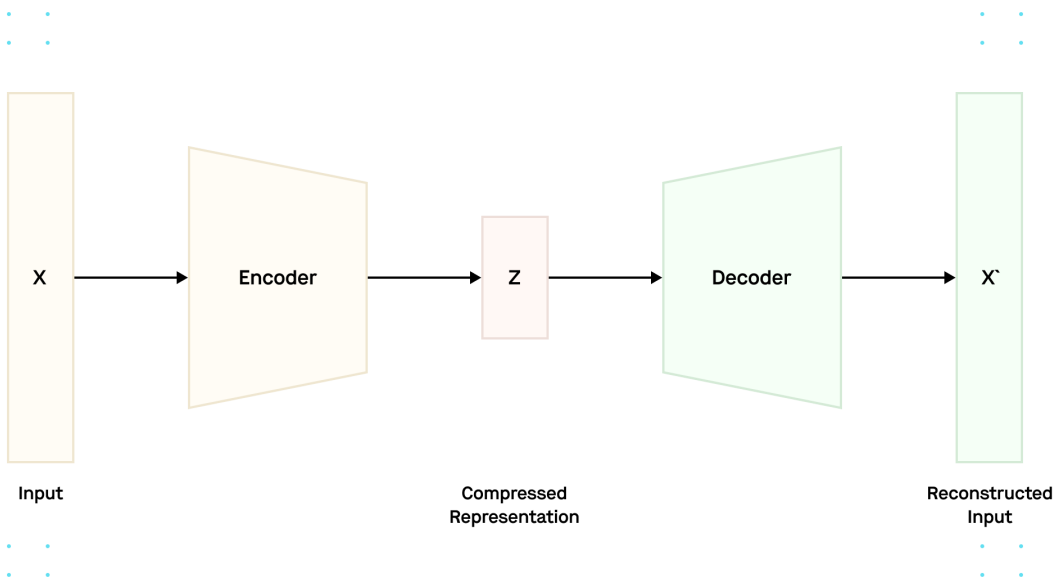
```
train_dataset = DocumentsDataset(train_names, noisy_folder_path,␣
 ↪clean_folder_path, train_transforms)
eval_dataset = DocumentsDataset(eval_names, noisy_folder_path,␣
 ↪clean_folder_path, eval_transforms)

train_loader = DataLoader(train_dataset, batch_size=ma_config.batch_size,␣
 ↪shuffle=True, num_workers=2)
eval_loader = DataLoader(eval_dataset, batch_size=ma_config.batch_size,␣
 ↪shuffle=False, num_workers=2)
```

## 2 Modèle de débruitage

L'architecture d'un autoencodeur se compose de deux parties :

**Encodeur** - un réseau neuronal feedforward qui calcule un `encodage` de faible dimension et extrait les caractéristiques qui capturent la structure utile dans l'entrée.

**Décodeur** - un réseau neuronal de type feedforward qui reconstruit les entrées à l'aide du `décodage`.




L'optimisation des paramètres de l'autoencodeur se fait en minimisant l'erreur de reconstruction, qui mesure la différence entre l'entrée et la sortie reconstruite. Cette famille de modèle a été introduite pour une tâche d'apprentissage purement **non supervisé** - la compression d'images. Cependant, les autoencodeurs peuvent également être utilisés pour des tâches d'apprentissage **semi-supervisé** ou **supervisé** telles que le débruitage ou la coloration d'images.

### 2.1 Autoencodeur de débruitage

Les autoencodeurs de débruitage fonctionnent comme les autoencodeurs classiques, à la différence que les entrées de l'autoencodeur sont des images bruitées et corrompues, alors que la sortie

souhaitée est une image nettoyée. Par conséquent, lors du calcul de la perte de reconstruction, nous mesurons la différence entre la sortie de l'autoencodeur et l'image nettoyée souhaitée. Ce fait nous permet de modifier l'architecture habituelle de l'autoencodeur. Au lieu d'utiliser un autoencodeur **under complete** (la taille cachée de l'encodage est inférieure à la taille de l'entrée), nous pouvons utiliser un autoencodeur **over complete** (la taille cachée de l'encodage est égale ou supérieure à la taille de l'entrée). Cela est possible parce que nos entrées sont différentes de nos sorties, de sorte que l'autoencodeur ne peut pas se contenter d'apprendre une correspondance triviale.

### 2.1.1 L'encodeur

Nous allons implémenter un réseau de neurones de convolutif comme encodeur. Nous allons utiliser des couches de convolution avec le même `padding`.

**Structure de l'encodeur:**

1. Couche de convolution avec `input channels = 1`, `output channels = 64`, `kernel de taille (3x3)` et `padding=1`
2. Fonction d'activation `ReLu`
3. Couche de convolution avec `input channels = 64`, `output channels = 128`, `kernel de taille (3x3)` et `padding=1`
4. Batch Norm pour `128 channels`
5. Fonction d'activation `ReLU`
6. `Max Pooling` avec `kernel de taille (2x2)`

**À faire :**

**Q5**. Compléter la méthode `__init__` de la classe `CustomEncoder`

**Q6**. Complete method `forward` de la classe `CustomEncoder`

[ ]:
```python
class CustomEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        ### Début de votre code


        ### Fin de votre code

    def forward(self, x):

        ### Début de votre code


        ### Fin de votre code

        return x
```

### 2.1.2 Le décodeur

**Structure du décodeur:**

1. Couche de convolution avec `input channels = 128`, `output channels = 64`, `kernel de taille (3x3)` et `padding=1`
2. Fonction d'activation `ReLU`
3. `Upsample` avec un `scale factor = 2`
4. Couche de convolution avec `input channels = 64`, `output channels = 1`, `kernel de taille (3x3)` et `padding=1`
5. Fonction d'activation `Sigmoid`

Les couches convolutives conservent cette fois la même taille en entrée. L'encodeur divise par deux la forme de vos images, nous devons donc les remettre à leur forme initiale en utilisant `nn.Upsample` avec `scale factor = 2`.

L'application de `transforms.ToTensor()` met à l'échelle toutes les valeurs d'une image entre 0 et 1, c'est pourquoi la fonction d'activation `Sigmoid` est choisie pour la dernière couche.

**À faire :**

**Q7**. Compléter la méthode `__init__` de la classe `CustomDecoder`

**Q8**. Compléter la méthode `forward` de la classe `CustomDecoder`

[ ]:
```python
class CustomDecoder(nn.Module):
    def __init__(self):
        super().__init__()

        ### Début de votre code


        ### Fin de votre code

    def forward(self, x):

        ### Début de votre code


        ### Fin de votre code
        return x
```

**Combinaison d'un encodeur et d'un décodeur :**

[ ]:
```python
class Autoencoder(nn.Module):
    def __init__(self, encoder, decoder):
        super().__init__()

        self.encoder = encoder()
        self.decoder = decoder()


    def forward(self, x):
        code = self.encoder(x)
```

```
        output = self.decoder(code)

        return output
```

## 2.2 Entraînement du modèle

### 2.2.1 Boucle d'apprentissage

**À faire :**

**Q9**. Compléter la fonction `train_loop` qui permet de faire un cycle d'apprentissage et retourne l'erreur d'apprentissage.

[ ]:
```
# VALIDATION_FIELD[func] train_loop

def train_loop(model, optimizer, criterion, train_loader, device=ma_config.
 ↪device):
    running_loss = 0
    model.train()
    pbar = tqdm(train_loader, desc='Itération sur jeu de données train')
    for noisy, clean in pbar:
        ### Début de votre code

        # pass to device
        # forward
        # optimize

        ### Fin de votre code
    running_loss /= len(train_loader.sampler)
    return running_loss
```

### 2.2.2 Boucle d'évaluation

**À faire :**

**Q10**. Compléter la fonction `eval_loop` qui permet de calculer l'erreur d'évaluation sur le jeu de données d'évaluation.

[ ]:
```
def eval_loop(model, criterion, eval_loader, device=ma_config.device):
    running_loss = 0
    model.eval()
    with torch.no_grad():
        total_loss = []
        pbar = tqdm(eval_loader, desc='Iterating over evaluation data')
        for noisy, clean in pbar:
            ### Début de votre code

            # pass to device
            # forward
```

```
            ### Fin de votre code
    mean_loss = sum(total_loss)/len(total_loss)
    running_loss /= len(eval_loader.sampler)
    return running_loss
```

### 2.2.3 Création d'une fonction combinée train/eval, qui affichera les résultats intermédiaires :

**À faire :**

**Q11**. Compléter la fonction `train` qui permet de faire l'entraînement de l'autoencodeur.

[ ]:
```
def train(model, optimizer, criterion, train_loader, valid_loader,
          device=ma_config.device,
          num_epochs=ma_config.num_epochs,
          eval_loss_min=np.inf,
          logdir=ma_config.logdir):

    for e in range(num_epochs):


        # boucle d'apprentissage pour calculer l'erreur d'apprentissage

        # boucle d'évaluation pour calculer l'erreur d'évaluation

        # afficher la progression des deux erreurs

        # affichage d'une image du jeu de données de test (valid_loader) et le␣
↪résultat de son débruitage avec les paramètres obtenus

        # sauvegarder le modèle s'il améliore l'erreur d'évaluation
```

### 2.2.4 Entraînement et évaluation

[ ]:
```
set_seed(ma_config.seed)
model = Autoencoder(CustomEncoder, CustomDecoder).to(ma_config.device)
optimizer = optim.Adam(model.parameters(), lr=ma_config.lr)
criterion = nn.MSELoss()
train(model, optimizer, criterion, train_loader, eval_loader)
```