Rapport de projet :
5A. Mathématiques Appliquées et Modélisation 2017-2018

Modèles de mélange et traitement d'image :

Groupe de travail : ABOUADALLAH Mohamed Anwar

École Polytechnique Lyon
15 Boulevard Latarjet 69622 Villeurbanne

«Le domaine des sciences des données est tout juste en train de se cristalliser » Stéphane Mallat

Remerciements

Notre projet s'inscrit dans le domaine de la modélisation et de l'analyse des données, un sous domaine des statistiques qu'on a exploré durant les années passées au sein de Polytech'Lyon en plus des projets de statistique de 4A qui ont été une bonne introduction pour nous dans le domaine de la Data Science.
Par ce fait, nous souhaitons commencer nos remerciements par adresser un témoignage de la plus vive des reconnaissances vers Clément Marteau [1], notre responsable de projet, pour les différentes explications et le grand accompagnement qu'il nous a apporté tout au long de ce projet.
Nous souhaitons aussi remercier Madame Lola Etiévant [2] pour son accompagnement durant nos projets de l'année dernière qui nous été d'une aide capitale.
Et afin de clore cette partie remerciements, nous tenons absolument à remercier les membres de la communauté Python et Stack Overflow pour toute l'aide qu'ils nous ont fournie durant la partie implémentation de notre projet.

1. marteau@math.univ-lyon1.fr
2. etievant@math.univ-lyon1.fr

Table des matières

1 Introduction

1.1 Avant propos

Ce projet a nécessité l'utilisation de plusieurs logiciels :

- **Rédaction du rapport et présentation :** LaTeXpour la rédaction, Zotero pour la bibliographie et enfin antidote pour la correction des erreurs.
- **Langages et logiciels de programmation :** Python.
- **Management du projet :** Nous avons mobilisé Gantt Project

Les codes utilisés sont disponibles en annexe. Nous avons aussi des notebook Jupyter afin de faciliter la communication entre nous.

1.2 Contexte du projet

« *L*a peinture, ce n'est pas copier la nature mais c'est apprendre à travailler comme elle», une belle citation de Pablo Picasso, fondateur du cubisme et un compagnon d'art du surréalisme qui résume tout simplement l'état d'esprit créatif d'un vrai data scientist et d'un spécialiste du deep learning.
Ainsi, dans ce projet on va s'intéresser au traitement de l'image via des méthodes numérico-statistiques. Mais avant, on peut se poser plusieurs questions : Existe-t-il une manière « idéale » de traiter des images ? Si oui, laquelle ? Sinon, à quel point peut-on s'en approcher ?
Il faut tout d'abord définir la manière « idéale » de traiter l'image. Nous allons fixer la définition suivante : Une méthode de traitement d'image idéale consisterait à enlever le bruit sans pour autant entrer dans le fameux dilemme [3] « biais variance ». Actuellement, plusieurs méthodes existent : Les plus célèbres parmi elles restent le Modèle de Peronna-Malik, la déconvolution par filtre de Wiener, le débruitage par séries de Fourier et par ondelettes. Sur ces méthodes s'ajoute le modèle H.D.M.I., développe par ANTOINE HOUDARD, CHARLES BOUVEYRON et JULIE DELON en 2017 et sur lequel portera notre projet.

1.3 But du projet

A la suite de cette brève introduction, nous souhaitons discuter avant toute chose du domaine de l'imagerie dont le développement est intrinsèquement corrélé avec les avancées dans le domaine de l'informatique.
En effet, depuis plusieurs années, la capacité à accéder en temps utile à l'information est devenue un atout compétitif certain pour les multinationales. Pour y parvenir, il convient de se doter d'une stratégie et des méthodes statistiques permettant d'extraire les informations utiles regroupées dans un domaine portant le nom de "Data Science", une famille de méthodes statistiques dont les principales caractéristiques sont d'être multidimensionnelles et descriptives.
Parmi ces méthodes, on trouve les algorithmes d'apprentissage automatique, principalement utilisées dans le domaine de l'Intelligence Artificielle, elles permettent, dans une certaine mesure, à un système piloté ou assisté par ordinateur, d'adapter ses analyses et ses comportements en se fondant sur l'analyse

3. le problème de minimiser simultanément deux sources d'erreurs qui empêchent les algorithmes d'apprentissage supervisé de généraliser au-delà de leur échantillon d'apprentissage

de données empiriques provenant d'une base de données ou des capteurs.

Durant le vingt-et-unième siècle, avec l'augmentation de la puissance des processeurs, l'abondance de l'information (Données) et la création de langages de programmation simples et efficaces tels que Python et R, les méthodes de classification ont vu leurs champs d'application s'étendre jusqu'à devenir primordiales dans des domaines telles que l'imagerie médicale ou la reconnaissance faciale qu'on est accoutumé à appeler le "**Deep Learning**" ou "**Deep Structured Learning**" qui se traduit par "apprentissage profond".

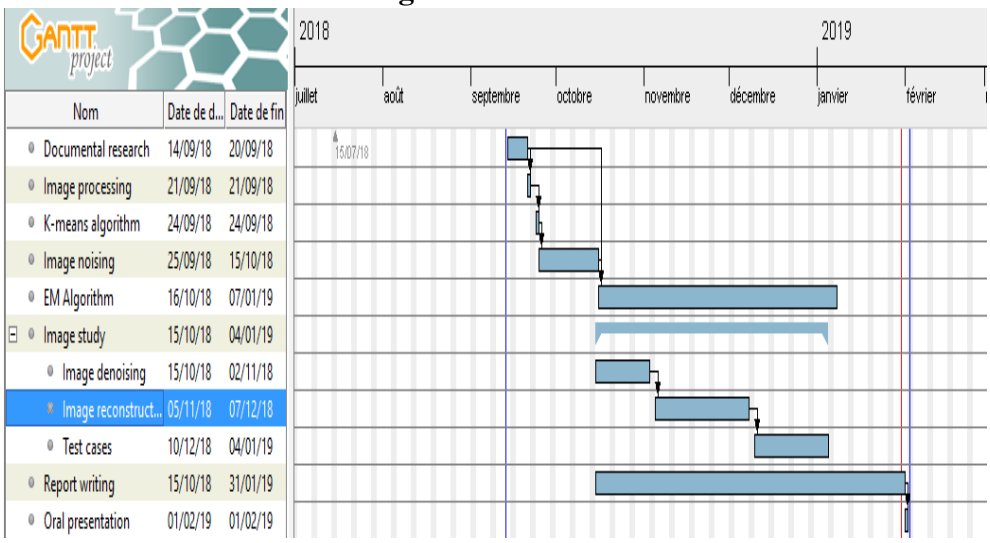
Suite à cette brève présentation du thème de notre projet, nous allons maintenant vous introduire notre objectif :
Notre projet consiste à résoudre le problème du débruitage d'image par l'intermédiaire d'une classification non supervisée basée sur l'utilisation de l'algorithme E.M. dans le cadre des modèles de mélanges probabilistes de grande dimension (HIGH-DIMENSIONAL MIXTURE MODELS FOR UNSUPERVISED IMAGE DENOISING) abrégé H.D.M.I. sur les images contenant un bruit. Ce modèle propose une modélisation complète du processus censé avoir généré le bruit des patchs.
Dans le cadre de notre projet, on compte s'intéresser à des modèles des mélanges gaussiens dans un premier temps puis développer un script habile à faire un débruitage d'image.
Mais avant toute chose, nous allons définir brièvement ce qu'est un modèle de mélange gaussien (GMM pour Gaussian Mixture Model) : il s'agit d'un modèle statistique exprimé selon une densité mélange qui est usuellement utilisée afin d'estimer paramétriquement la distribution de variables aléatoires en les modélisant comme une somme de plusieurs gaussiennes (appelées noyaux).
Ainsi, nous allons implémenter un script qui permettrait de bruiter une image, extraire sa matrice, la décomposer en patchs, classifier les patchs, calculer la moyenne pour chaque classe de patchs puis reconstruire l'image à partir de ces moyennes. En d'autre terme nous allons jouer aux chirurgiens avec des images.

1.4 Répartition des taches

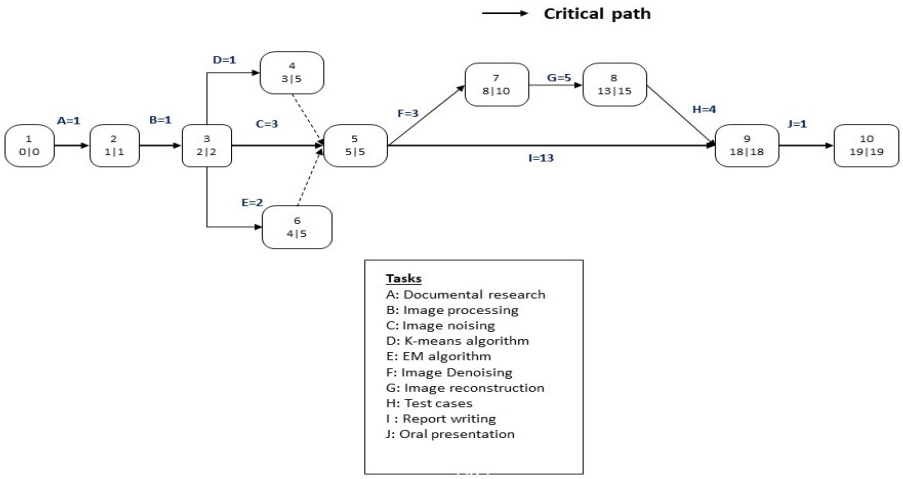
Notre équipe étant constituée de deux membres, nous avons essayé de répartir les tâches de manière égale afin que chacun d'entre nous puisse exprimer un peu de magie.
Dans cette section, nous allons exposer les diagrammes de Gantt et Pert.

1.4.1 Diagramme de Gantt

Diagramme de Gantt



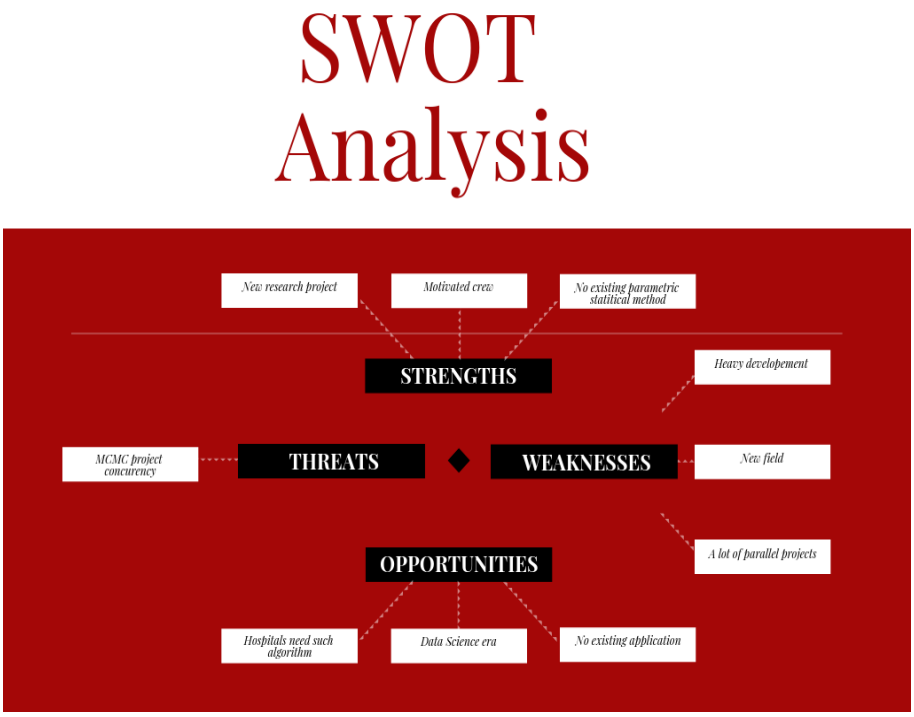
1.4.2 Diagramme de de Pert



Maintenant, après avoir expliqué le fonctionnement de notre équipe, il est temps d'expliquer l'importance de notre projet.

1.5 Management du projet

1.5.1 Analyse Swot du projet



2 Quelques bases mathématiques

2.1 Calcul de la vraisemblance

- La vraisemblance décrit la plausibilité d'une valeur des paramètres d'un modèle, étant donné l'observation d'un certain nombre de réalisations d'une variable aléatoire. Elle est calculée de cette manière : $L(X,\theta) = \prod_{k=1}^{n} f(x_i;\theta)$
- Pour (X_1,...,X_n) un échantillon de variables aléatoires i.i.d. de loi P_θ, on appel E.M.V. la quantité :$\widehat{\theta}_{MV}$ tel que $L(X,\widehat{\theta}_{MV}) = max \prod_{k=1}^{n} f(x_i;\theta)$ [4]
- Il arrive dans certains cas on se retrouve dans l'incapacité de calculer analytiquement ce maximum. On se retrouve contraint à approximer le zéros de $\frac{\partial L(X,\theta)}{\partial \theta}$ par des méthodes numériques (descente gradient, éléments finis ...)

2.2 Définitions

Définition 1 (Barycentre) *Soit un ensemble de points (x_1, x_2,..., x_n), on cherche à partitionner les n points en k ensembles S = ($S_1, S_2, \ldots, S_k$) en minimisant la distance entre les points à l'intérieur de chaque partition. La fonction à minimiser est la suivante :*
Soit μ_i est le barycentre des points dans S_i,

$$\underset{S}{ArgMin} \sum_{i=1}^{k} \sum_{X_j \in S_i} ||X_j - \mu_i||^2$$

Définition 2 (Critère B.I.C.) *Ce critère utilise un point de vue bayésien : on ne considère plus le paramètre inconnu comme un vecteur de $\mathbb{R}^K$ mais plutôt comme une variable aléatoire à valeurs dans $\mathbb{R}^K$*
Critère BIC (Bayesian Information Criterion) :

$$BIC(m) = n \log(\hat{\sigma}^2_{(m)}) + \log n \times |m|.$$

Il consiste à sélectionner le modèle vérifiant :

$$\hat{m}_{BIC} = \arg \min_{m \in \mathcal{M}} BIC(m).$$

4. Pour les variables continues, notre $\prod$ sera remplacée par une intégrale.

3 L'image numérique

3.1 Petit historique sur le traitement de l'image

Le besoin de traiter et synthétiser les images a commencé à être ressenti dès le début du 20 ème siècle, mais, ce n'est qu'en 1950 où grâce aux avancées de la physique des particules qui a contraint les physiciens à détecter des trajectoires particulièrement complexes et les pousser à partir des années 60 à analyser systématiquement entre 10.000 à 100.000 images par expérience dans les chambres à bulles dans un souci de détermination des trajectoires de milliers de particules grâce à plusieurs caméras réparties.
S'ensuit la période de guerre froide où les chercheurs américains et russes commencent à s'intéresser à la lecture optique pour reconnaître les caractères dactylographiés [5] d'un texte. En termes d'images, cette application semblait abordable : le nombre de caractères est limité et l'image est contrastée.
En France, le professeur René de Possel [6] fut l'un des premiers mathématiciens à orienter ses recherches sur le sujet. Ce dernier réussit à mettre au point un lecteur optique dès 1965 et fit fonctionner en 1969 la première machine au monde à lire automatiquement les textes imprimés. Ainsi en 1976, on disposait d'un système complet de reconnaissance de 1 000 caractères dactylographiés par seconde.
Pour l'écriture manuscrite, il a fallu attendre une vingtaine d'années de plus pour que de telles techniques soient opérationnelles dans certains domaines tels que la lecture des adresses postales ou les montants des chèques bancaires.
D'une manière générale, jusqu'à la fin des années 1960, les images, que ce soient des photos de satellites, des images d'ADN au microscope électronique ou des radiographies, étaient de mauvaise qualité et difficiles à exploiter. Les optiques étaient peu performantes, provoquant de nombreuses aberrations géométriques et chromatiques. Traiter des images a d'abord consisté à les restaurer en corrigeant tous ces défauts d'acquisition. En parallèle, se posaient deux problèmes :

- **Leur volume :** Difficile à stocker.
- **Leur traitement :** Très lent

Ainsi beaucoup de recherches ont été menées dans les années 1960 dans le domaine de la compression d'images, en particulier pour l'image animée. Les premiers travaux se situent entre 1957 et 1962, surtout aux États-Unis. Ensuite, dès 1965-1970, une large communauté de chercheurs à travaillé à comprimer ces gros volumes de données.
Actuellement, grâce aux grandes veilles technologiques dans les domaines de l'informatique, des statistiques et du deep learning, le traitement d'image est devenu capital, vu qu'on peut voir que certaines sociétés telle que Apple ont développé des algorithmes basés sur des réseaux de neurones capable de transformer une simple photo en un Van Gogh ou encore Esker dont l'efficacité en terme de numérisation des documents frôle les 98% avec un temps de traitement record.

3.2 L'image vue par les matheux !

Une image numérique en niveaux de gris [7] est un tableau de valeurs. Chaque case de ce tableau, qui stocke une valeur, se nomme un pixel. En notant n le nombre de lignes et p le nombre de colonnes de l'image, on manipule ainsi un tableau de np pixels.

5. action de saisir un texte sur un clavier de machine à écrire ou d'ordinateur
6. 1905-1974 directeur de l'Institut Blaise Pascal à Paris
7. En imprimerie, le niveau de gris désigne la concentration des points de trame et est donc directement en rapport avec le rendu de l'image. Un niveau de gris va alors varier du blanc au noir.

Les valeurs des pixels sont enregistrées dans l'ordinateur ou l'appareil photo numérique sous forme de nombres entiers entre 0 et 255, ce qui fait 256 valeurs possibles pour chaque pixel. La valeur 0 correspond au noir, et 255 au blanc. Les valeurs intermédiaires correspondent à des niveaux de gris allant du noir au blanc.
Ainsi par exemple, on va prendre une image quelconque et on va générer notre tableau de pixels :

```
In [73]: from PIL import Image
         import numpy as np
         imgpil = Image.open("image.pgm")
         imgpil
```

Out[73]:

3.2.1 Traitement de l'image via Python

Nous commencerons par la lecture de l'image, plusieurs manières sont possibles, la première étant d'utiliser des librairies classiques telles que Image Pil, OpenCv ou matplotlib.image.
Dans le cadre de notre projet nous avons choisi d'utiliser ScikitImage, une librairie similaire à Scikit-Learn et dont les fonctions s'accordent parfaitement avec cette dernière.
Ainsi pour notre image test de base :

On l'affiche puis on la transforme en matrice afin de faire notre traitement dessus.

```
    def affiche(chemin) :
    imgpil = Image.open(chemin)
    return imgpil

    def matim(chemin):
    image=io.imread(chemin,as_grey=True)
    return image
```

Ce qui nous donne un tableau de dimension $512x512$ de cette forme :

```
array([[ 0.69411765,  0.69411765,  0.69411765, ...,  0.72156863,
         0.67058824,  0.57254902],
       [ 0.69411765,  0.69411765,  0.69411765, ...,  0.72156863,
         0.67058824,  0.57254902],
       [ 0.69411765,  0.69411765,  0.69411765, ...,  0.72156863,
         0.67058824,  0.57254902],
       ...,
       [ 0.22352941,  0.22352941,  0.25490196, ...,  0.48235294,
         0.46666667,  0.45882353],
       [ 0.22745098,  0.22745098,  0.27843137, ...,  0.48235294,
         0.48627451,  0.49803922],
       [ 0.22745098,  0.22745098,  0.27843137, ...,  0.48235294,
```

Dans le cadre de notre projet, nous avons créé plusieurs fonctions qui permettent de traiter cette image :

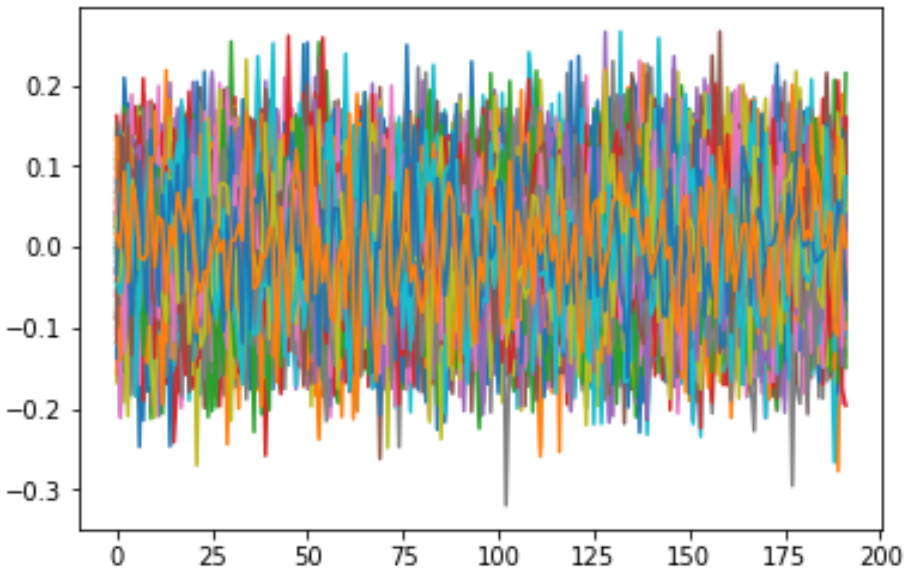
- **Distord :** Permet de tordre l'image au cas où on se retrouve avec des couleurs qui changent la dimension matricielle (pour ne pas avoir à gérer des matrices N-dimensionnelles).
- **Image_rescale :** Permet de réduire la dimension matricielle.

3.3 Bruitage d'une image en Python

3.3.1 Bruit Gaussien

Sur une photographie, des millions de pixels mesurent l'intensité lumineuse reçue. Cependant, ils le font uniquement avec une certaine précision. Typiquement, si l'intensité lumineuse qui frappe un certain pixel est égale à 10, la mesure effectuée par le capteur risque d'être soit inférieure soit supérieure à ce nombre, cela se produisant de façon non prévisible et variable d'un pixel à l'autre.
C'est ce que l'on appelle un bruit. Dans le cadre de notre projet, nous allons nous intéresser aux bruits gaussiens, c'est-à-dire un bruit suivant un processus gaussien sous la forme suivante :

Exemple de bruit gaussien tracé en Python : [8]



8. Ici $\sigma = 0.07$

3.3.2 Bruitage d'une image

On applique le bruit sur un image simple via la fonction suivante :

```
def bruitage2(sigma, image) :
x0=image
y=x0 + sigma*np.random.standard_normal(x0.shape)
return(y)

```

Ici on crée une simple matrice de bruit gaussien puis on ajoute chacun de ses termes sur notre matrice image. Ceci nous donne des résultats de cette forme :

Image de base :

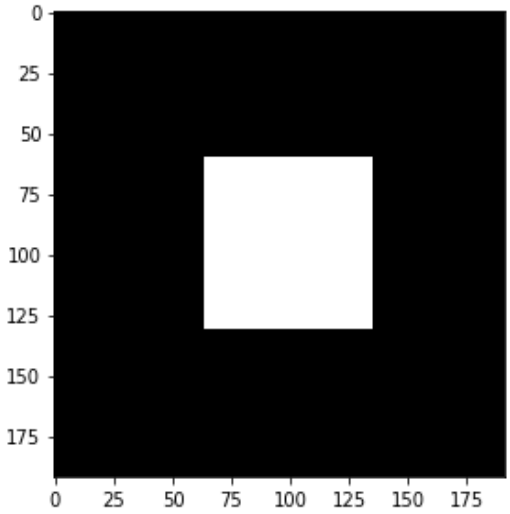
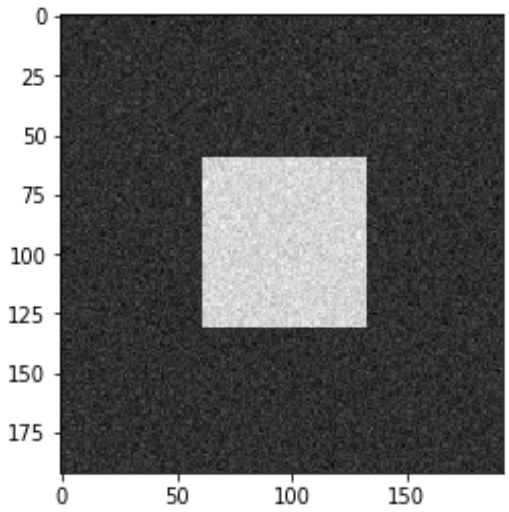


Image bruitée :



Pour une image plus complexe :

Image de base :

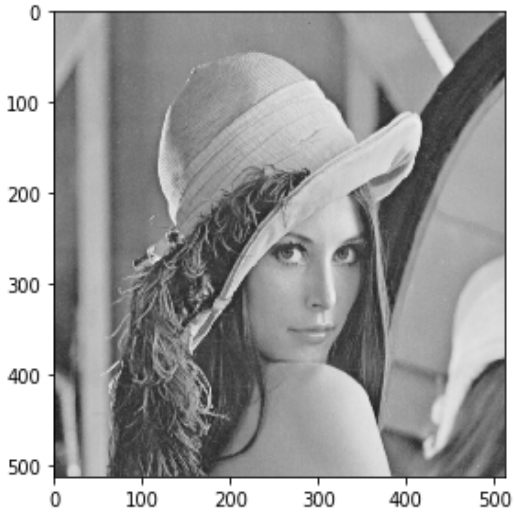
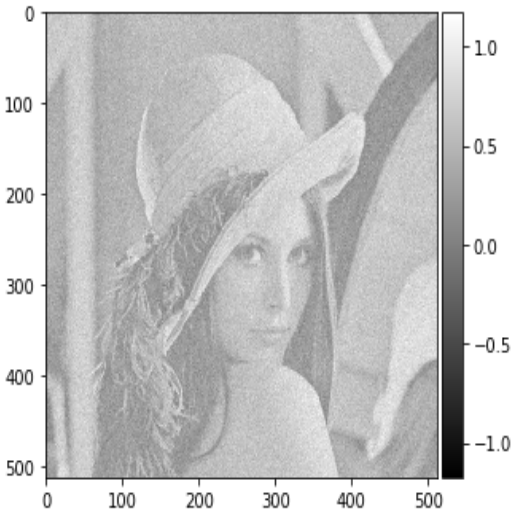


Image bruitée :



Nous avons utilisé plusieurs manières de bruitage comme l'ajout d'un filtre gaussien ou les fonctions de bruitage Python mais cette manière simple reste la meilleure.

3.3.3 Extraction des patchs

Il existe une infinité de méthodes pour extraire les patchs d'une image. Nous avons testé plusieurs méthodes pré-existantes mais nous avons préféré utiliser la nôtre.
Elle consiste à se fixer une taille pour les patchs, créer une nouvelle matrice vide puis parcourir la matrice image à l'aide d'une double boucle for. Et enfin, remplir notre matrice vide avec les patchs.

```
    def patches_extractor(image, patch_size) :
m,n = image.shape
    l,k = patch_size
    vec=[]
    patchs=np.zeros((n*m/(l*k), l*k))
    a,b = patchs.shape
    compteur=0
    for i in range (m) :
    for j in range (n) :
    vec.append(image[i,j])
    for i in range (a) :
    for j in range (b) :
    patchs[i,j]=vec[compteur]
    compteur += 1
    return patchs
    print("extraction du data set")


```

3.3.4 Reconstruction de l'image

La manière avec laquelle on reconstruit l'image sera expliquée ultérieurement dans ce rapport.

4 Modèles de mélange

4.1 Concepts théoriques

Nous allons maintenant nous concentrer sur le modèle de mélange gaussien, un cadre important pour les problèmes de regroupement. Contrairement aux autres méthodes, il s'agit d'une approche probabiliste de la classification. L'un des principaux avantages de la mise en cluster basée sur un modèle est que la partition résultante peut être interprétée de manière statistique. Il suppose que les observations sont tirées d'une distribution de mélange dont les composants sont gaussiens avec des paramètres (μ_k, Σ_k), la densité du $k-th$ composant du mélange est :

$$\boxed{\phi_{\mu_k,\Sigma_k}(x) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} exp(-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k))}$$

Soit θ la liste contenant tous les paramètres inconnus d'un modèle de mélange gaussien : la famille des moyennes $\mu = (\mu_1, ..., \mu_K) \in (\mathbb{R}^p)^K$, la famille des matrices de covariance $\Sigma = (\Sigma_1, ..., \Sigma_K) \in (S_{++}^p)^K$ et le vecteur des probabilités de cluster $\pi = (\pi_1, ..., \pi_K) \in [0,1]^K$ tel que $1_K^T\pi = 1$. La densité d'une observation X_1 est alors donnée par :

$$\boxed{p_\theta(x) = \sum_{k=1}^K \pi_k \phi_{\mu_k,\Sigma_k}(x),\ \forall x \in \mathbb{R}^p \text{ où } \theta = (\mu, \Sigma, \pi).}$$

Soit Z une variable aléatoire discrète prenant ses valeurs dans l'ensemble $[K]$ et telle que $P(Z=k) = \pi_k$ pour tout $k \in [K]$. La variable aléatoire Z indique le groupe duquel l'observation X est tirée. Considérant que toutes les distributions conditionnelles $X|Z=k$ sont gaussiennes, on obtient la formule suivante pour la densité marginale de X :

$$\boxed{p_\theta(x) = \sum_{k=1}^K P(Z=k)p_\theta(x|Z=k) = \sum_{k=1}^K \pi_k \phi_{\mu_k,\Sigma_k}(x),\ \forall x \in \mathbb{R}^p}$$

Dans le problème de clustering, l'objectif est d'affecter X à un cluster ou, de manière équivalente, de prédire le cluster Z du vecteur X. Une fonction de prédiction dans un tel contexte est $g : \mathbb{R}^p \to [K]$ telle que $g(X)$ soit aussi proche que possible de Z. Si nous mesurons le risque d'une fonction de prédiction g en termes de taux d'erreur de classification $R_\theta(g) = P_\theta(g(X) \neq Z)$, il est bien connu que le prédicteur optimal (Bayes) $g_\theta^* \in argmin_g R_\theta(g)$ est fourni par la règle :

$$\boxed{g_\theta^*(x) = \arg\max_{k\in[K]} \tau_k(x,\theta)}$$

où $\tau_k(x,\theta) = p_\theta(Z=k|X=k)$ représente la probabilité conditionnelle de la variable Z sachant X. Dans le modèle de mélange gaussien, la règle de Bayes implique que :

$$\boxed{\tau_k(x,\theta) = \frac{p_\theta(x|Z=k)P(Z=k)}{p_\theta(x)} = \frac{\pi_k\phi_{\mu_k,\Sigma_k}(x)}{\sum_{k'=1}^K \pi_{k'}\phi_{\mu_{k'},\Sigma_{k'}}(x)}}$$

La vraie valeur du paramètre θ n'étant pas disponible, cette formule ne peut pas être utilisée directement pour résoudre le problème de la mise en cluster. Au lieu de cela, une stratégie naturelle consiste à estimer θ à l'aide d'un vecteur $\hat{\theta}$, sur la base d'un échantillon $X_1, ..., X_n$ tiré de la densité p_θ, puis à définir la règle de classification par :

$$\hat{g}(x) = g^*_{\hat{\theta}}(x) = \arg\max_{k\in[K]} \tau_k(x,\hat{\theta}) = \arg\max_{k\in[K]} \hat{\pi}_k \phi_{\hat{\mu}_k,\hat{\Sigma}_k}(x)$$

Une approche commune pour estimer le paramètre θ consiste à s'appuyer sur la maximisation de la probabilité. Soit $X_1,...,X_n$ avec $X_i \in \mathbb{R}^p$ un ensemble d'observations iid tirées de la densité p_θ donnée. La log-vraisemblance du modèle de mélange gaussien est :

$$l_N(\theta) = \sum_{i=1}^N log p_\theta(x_i) = \sum_{i=1}^N log\left(\sum_{k=1}^K \pi_k \phi_{\mu_k,\Sigma_k}(x_i)\right)$$

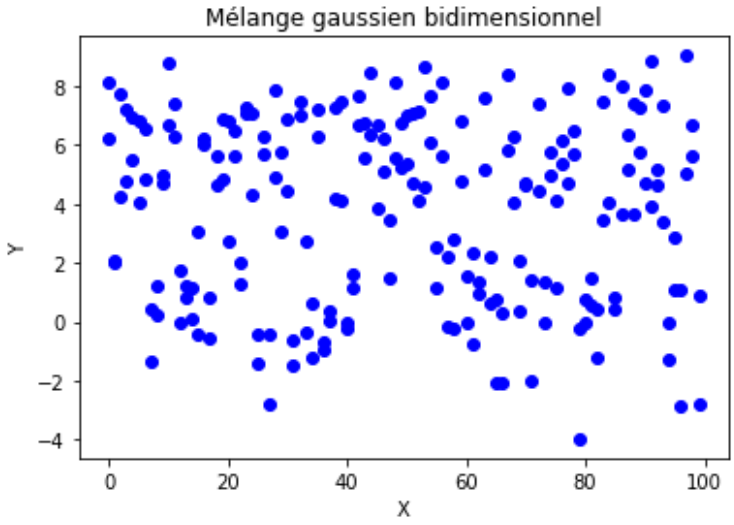
En raison de la présence dans cette équation du logarithme d'une somme, la maximisation de la log-vraisemblance est un problème diffcile non linéaire et non convexe. En particulier, il ne s'agit pas d'une distribution familiale exponentielle donnant des expressions simples. Une approche couramment utilisée pour maximiser approximativement cette équation en ce qui concerne θ est l'algorithme EM (anticipation-maximisation) que nous allons étudier dans la partie suivante.

4.2 Simulation d'un modèle de mélange

Dans le cadre de notre projet, nous avons choisi de faire quelques simulations à l'aide de Python afin de bien visualiser et s'approprier ces notions.
Ainsi, nous allons les simuler dans le cadre bi et multidimensionnel.

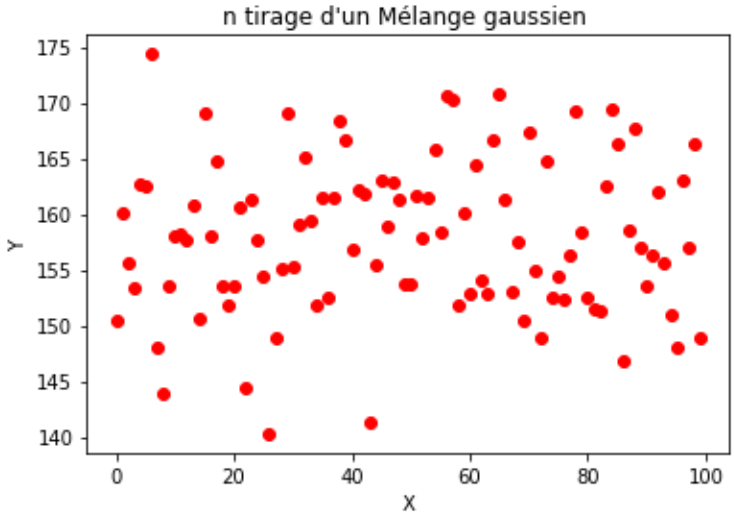
♣ Dans le cadre bidimensionnel on obtient un nuage de cette forme :

```
    def melangegaussbid(n,m):
    X = np.zeros((n,m))
    for i in range (n) :
    Z = np.random.binomial(1,2/5,1) # choix de la loi par tirage de Benoull
    if (Z == 1) :
    X[i,0] = np.random.normal(2/3,2,1)
    X[i,1] = np.random.normal(2/3,1,1)
    else :
    X[i,0] = np.random.normal(7,1,1)
    X[i,1] = np.random.normal(5,1,1)
    return (X)
    n,m=(100,2)
    X=melangegaussbid(n,m)
    plt.plot(X, 'bo')
    plt.title("Melange gaussien bidimensionnel")
    plt.ylabel("Y")
    plt.xlabel("X")
    plt.show()

```



♣ Ensuite nous avons simulé un modèle de mélange multi-dimensionnel :

```
    def melangegaussnvar(n):
X = np.zeros((100,1))### Boucle permettant de tirer 100 valeurs issues
    for k in range(100) :
Z = np.random.normal(1,2/5,1) # choix de la loi par tirage de Benoulli
    if (Z == 1) :
X[k] = np.random.normal(124,8,1)
    else :
X[k] = np.random.normal(157,7,1)
    return(X)
Y= melangegaussnvar(n)
    plt.plot(Y, 'bo')
plt.title("n tirage d'un Melange gaussien ")
    plt.ylabel("Y")
plt.xlabel("X")
    plt.show()

```



On peut facilement voir que dans le cadre d'un modèle de mélange multi-dimensionnel, on retrouve un ensemble de classe de points.
Dans la section suivante, nous allons présenter l'algorithme E.M. et dans celle qui suivra nous allons montrer la différence entre ces deux nuages de points.

5 Algorithme E.M.

"Le seul moyen de faire une méthode instructive et naturelle, est de mettre ensemble des choses qui se ressemblent et de séparer celles qui diffèrent les unes des autres" est une magnifique citation issue du Tome 1 de l'encyclopédie "Histoire Naturelle" écrite par Buffon en 1749.
Cette belle phrase nous révèle la raison première derrière l'apparition de la classification telle qu'on la connait aujourd'hui. La classification est alors devenue le thème central de l'histoire naturelle, puis de la biologie. À la fin du 19ème siècle, enfantée par les idées de Pascal et Condorcet, la théorie statistique apparaît, ce qui ne fera qu'accroître les performances des méthodes de classification, et ce, jusqu'à nos jours.
Ainsi l'apprentissage non supervisé est une méthode d'apprentissage automatique. Il s'agit de diviser un groupe hétérogène de données en sous-groupes, de manière à ce que les données considérées comme les plus similaires soient associées au sein d'un même groupe (homogène) et qu'au contraire les données considérées comme différentes se retrouvent dans d'autres groupes distincts. Ceci sera effectué sans qu'au préalable la nature des classes ait été choisie ; il s'agit donc d'une classification objective des données.

5.1 Présentation de l'algorithme

5.1.1 Présentation générale

Dans le cas non-supervisé, la maximisation de la log-vraisemblance d'un modèle de mélange peut conduire à des équations de vraisemblance qui ne possèdent pas de solutions analytiques. A dessein de contourner cela, on peut utiliser des algorithmes permettant de maximiser la log-vraisemblance quand les labels sont inconnus. Le plus utilisé d'entre eux est l'algorithme itératif Expectation-Maximization ou espérance maximisation dont le nom est souvent abrégé EM. Cet algorithme a été proposé par Dempster, Laird et Rubin en 1977, il est principalement utilisé en imagerie médicale dans le cadre de la reconstruction tomographique.

5.1.2 La construction de l'algorithme

On suppose que nos observations sont notées $X \in \mathbb{R}^n$ et que nous disposons d'un ensemble de variables latentes Z. L'ensemble des paramètres du modèle sera noté θ. L'objectif de notre algorithme E.M. est de déterminer les paramètres qui maximisent la vraisemblance de $p(X|\theta)$ quand ni cette expression ni son logarithme ne peuvent être maximisés par les méthodes de type MLE [9] classiques.
Soit notre jeu de données $\{X,Z\}$ qu'on ne connait pas totalement, on connait seulement les valeurs des X_n, sa vraisemblance est donnée par $p(X,Z|\theta)$.
Pour cela, on construit le postérior $p(Z|X,\theta)$, puis par le biais de $p(Z|X,\theta)$, on peut calculer la vraisemblance du set complet :

$$p(X,Z|\theta) = p(Z|X,\theta)p(X|\theta)$$

On suppose aussi qu'on connait une estimation θ_i pour θ ce qui nous permet de calculer le posterior $p(Z|X,\theta_i)$.

9. **Méthode MLE :** Méthodes d'estimation par maximum de vraisemblance

5.2 Initialisation

5.2.1 Initialisation aléatoire

L'algorithme commence d'abord par faire une assignation aléatoire des valeurs $\theta_i = \theta_0$. Ceci nous permet d'avoir une log-vraisemblance de cette forme :

$$\boxed{\log p(X|\theta) = \log\{\sum_z p(X, Z = z|\theta)\} = \log\{\sum_z p(X|Z = z, \theta)p(Z = z|\theta)\}}$$

Pour les variables latentes continues, notre somme sera remplacée par une intégrale, l'algorithme E.M. devrait maximiser $\log p(X|\theta)$ puis comme log est strictement croissante alors l'algorithme E.M. maximisera aussi $p(X|\theta)$. Par la suite, nous allons montrer le fonctionnement des deux étapes de notre algorithme.

5.2.2 Initialisation via Kmeans

Afin d'améliorer notre algorithme, on peut aussi commencer par une assignation via Kmeans. Dans ce rapport, nous n'allons pas trop tarder sur cet algorithme proposé par MacQueen en 1967.
Tout d'abord, il s'agit de l'un des algorithmes d'apprentissage non supervisé les plus simples et permet de résoudre le problème bien connu du clustering. La procédure suit un moyen simple et facile de classer un ensemble de données donné à travers un certain nombre de classes fixées a priori.
Il consiste à définir k barycentres, un pour chaque cluster. Ces barycentres doivent être placés de manière intelligente car leur emplacement diffère, et les résultats sont différents. Le meilleur choix est donc de les placer le plus loin possible les uns des autres. L'étape suivante consiste à prendre chaque point appartenant à un ensemble de données donné et à l'associer au barycentre le plus proche. Lorsqu'aucun point n'est en attente, la première étape est terminée et un groupage précoce est effectué. À ce stade, nous devons recalculer k nouveaux barycentres en tant que barycentres des grappes résultant de l'étape précédente. Une fois que nous avons ces k nouveaux barycentres, une nouvelle liaison doit être établie entre les mêmes points de jeu de données et le nouveau centre de gravité le plus proche. Une boucle a été générée. À la suite de cette boucle, nous pouvons remarquer que les k barycentres changent d'emplacement par étape jusqu'à ce qu'aucune autre modification ne soit apportée. En d'autres termes, les barycentres ne bougent plus. D'une manière plus formelle, on peut dire :

Algorithm 1 Algorithme K_means

Require: X, k

Initialisation aléatoire des barycentres $\mu_1, ...\mu k \in \mathbb{R}^n$.

Répéter jusqu'à la convergence

for i=1 **to** k **do**

$ArgMin_j \sum_{i=1}^{k} \sum_{X_j \in S_i} ||X_j - \mu_i||^2$

end for

for j=1 **to** k **do**

$\mu_j = \frac{\sum_{i=1}^{n} 1_{(C^{(i)}=j)} x^{(i)}}{\sum_{1}^{n} 1_{(C^{(i)}=j)}}$

end for

5.3 L'étape E ou E-step

À partir de notre estimation θ_i, nous essayerons d'améliorer la valeur de θ afin d'avoir $p(X|\theta) > p(X|\theta_i)$ puis si possible nous maximiserons la différence entre $p(X|\theta)$ et $p(X|\theta_i)$.
Cette étape constitue la première étape d'une itération de l'algorithme, elle consiste tout simplement à calculer $E_{Z|X,\theta_i}[\log p(X,Z|\theta)]$

Demonstration 3 *Notre θ_i calculé vérifie :* $\log p(X|\theta) \geq \log p(X|\theta_i) + \Delta(\theta|X,\theta_i) \equiv Q(\theta|\theta_i)$
Notre nouvelle fonction $Q(\theta|\theta_i)$ est inférieure à $\log p(X|\theta)$ *et sont égales en $\theta = \theta_i$.*

$$\begin{aligned}
& \log p(X|\theta) - \log p(X|\theta_i) \\
= \; & \log\{\textstyle\sum_z p(X|Z=z,\theta)p(Z=z|\theta)\} - \log p(X|\theta_i) \\
= \; & \log\left\{\textstyle\sum_z p(X|Z=z,\theta)p(Z=z|\theta)\frac{p(Z=z|X,\theta_i)}{p(Z=z|X,\theta_i)}\right\} - \log p(X|\theta_i) \\
= \; & \log\left\{\textstyle\sum_z p(Z=z|X,\theta_i)\frac{p(X|Z=z,\theta)p(Z=z|\theta)}{p(Z=z|X,\theta_i)}\right\} - \log p(X|\theta_i) \\
\geq \; & \textstyle\sum_z p(Z=z|X,\theta_i)\log\left\{\frac{p(X|Z=z,\theta)p(Z=z|\theta)}{p(Z=z|X,\theta_i)}\right\} - \log p(X|\theta_i) \\
= \; & \textstyle\sum_z p(Z=z|X,\theta_i)\log\left\{\frac{p(X|Z=z,\theta)p(Z=z|\theta)}{p(Z=z|X,\theta_i)p(X|\theta_i)}\right\} \\
\equiv \; & \Delta(\theta|X,\theta_i)
\end{aligned}$$

Bilan : E-step : consiste à calculer $Q(\theta|X,\theta_i) = E_{Z|X,\theta_i}[\log p(X,Z|\theta)]$

5.4 L'étape M ou M-step

Dans cette étape, on souhaite affecter la valeur θ_{i+1} à θ qui maximise $Q(\theta|\theta_i)$.

Demonstration 4 *Comme on a* $\log p(X|\theta_{i+1}) \geq Q(\theta_{i+1}|\theta_i)$ *alors la vraisemblance va aussi augmenter.*
Puis montrons $\arg\max_\theta Q(\theta|\theta_i) = \arg\max_\theta \left\{E_{Z|X,\theta_i}[\log p(X,Z|\theta)]\right\}$
On a

$$\begin{aligned}
\theta_{i+1} &= \arg\max_\theta Q(\theta|\theta_i) \\
&= \arg\max_\theta \left\{\log p(X|\theta_i) + \textstyle\sum_z p(Z=z|X,\theta_i)\log\frac{p(X|Z=z,\theta)p(Z=z|\theta)}{p(Z=z|X,\theta_i)p(X|\theta_i)}\right\} \\
&= \arg\max_\theta \left\{\textstyle\sum_z p(Z=z|X,\theta_i)\log p(X|Z=z,\theta)p(Z=z|\theta)\right\} \\
&= \arg\max_\theta \left\{\textstyle\sum_z p(Z=z|X,\theta_i)\log\frac{p(X,Z=z,\theta)p(Z=z,\theta)}{p(Z=z,\theta)p(\theta)}\right\} \\
&= \arg\max_\theta \left\{\textstyle\sum_z p(Z=z|X,\theta_i)\log p(X,Z=z|\theta)\right\} \\
&= \arg\max_\theta \left\{E_{Z|X,\theta_i}[\log p(X,Z|\theta)]\right\}
\end{aligned}$$

Bilan : M-step : consiste à calculer $\theta_{i+1} \leftarrow \text{argmax}_\theta\, Q(\theta|X,\theta_i)$

Enfin il reste à déterminer une condition d'arrêt pour notre algorithme, pour cela on va choisir arbitrairement un ε petit, puis on continue l'algorithme jusqu'à ce que cette condition soit réalisée :

$$\|\theta_i - \theta_{i+1}\| > \varepsilon$$

5.5 Pseudo-code

Ensuite, à partir de cela on va établir un pseudo-code pour l'algorithme E.M., on a choisi de le faire via une boucle "if" bien qu'il soit possible de le faire en boucle "while", de plus on a choisi d'utiliser un ε comme condition d'arrêt.

Algorithm 2 Algorithme EM

Require: X, $p(Z|X,\theta)$, ε
  $\theta_i \leftarrow \theta_0$, ou θ_0 est une affectation via l'algorithme Kmeans.
  E-step :
  $Q(\theta|X,\theta_i) = E_{Z|X,\theta_i}[\log p(X,Z|\theta)]$
  M-step :
  $\theta_{i+1} \leftarrow \text{argmax}_\theta\, Q(\theta|X,\theta_i)$
  if $\|\theta_i - \theta_{i+1}\| > \varepsilon$ **then**
    $\theta_i \leftarrow \theta_{i+1}$
  end if

5.6 Implémentation sous Python

Après avoir expliqué ce qu'est l'algorithme E.M. et ce qu'est l'algorithme des K-moyennes, nous allons maintenant l'implémenter sous Python sous forme de classe. En effet, notre algorithme sera différent du pseudo-code étant donné qu'il sera exclusivement utilisable pour le traitement des images. Ainsi, il sera implémenté de cette manière :

- On initialise nos labels θ à zero, on utilise ensuite l'algorithme Kmeans pour calculer notre θ_0
- On implémente notre fonction expectation pour calculer nos labels.
- On implémente notre fonction maximisation pour maximiser nos paramètres à partir des labels calculés auparavant..
- On fixe ensuite notre critère d'arrêt, en ce qui nous concerne nous avons pris $\|\theta_i - \theta_{i+1}\| > \varepsilon$ puis nous avons créé une fonction qui utilise les fonctions que nous avons explicitées auparavant pour lire la matrice de l'image bruitée, extraire les patchs puis faire la classification comme nous l'avons expliqué au début de cette section.

6 État d'art : Implémentation du modèle H.D.M.I.

6.1 Présentation générale

Le modèle H.D.M.I. donne une modélisation complète du processus censé avoir généré le bruit des patchs. Pour surmonter les problèmes d'estimation potentiels dûs à la grande dimension des modèles, le modèle HDMI adopte une modélisation parcimonieuse c'est à dire une modélisation n'utilisant que le minimum de causes élémentaires pour expliquer nos variables. Ceci suppose que les données résident dans des sous-espaces de groupes de faible dimension.
Ainsi, cette modélisation parcimonieuse permet d'obtenir une stabilité numérique dans le calcul de l'espérance conditionnelle de l'image qui est appliquée pour débruitage.
L'utilisation d'un tel modèle permet également de s'appuyer sur la sélection d'outils, tels que BIC, pour déterminer automatiquement les dimensions intrinsèques du sous-espace et la variance du bruit. Cela donne un algorithme de débruitage aveugle qui démontre des performances de pointe, à la fois lorsque le bruit est connu ou inconnu.

6.2 Modèle utilisé :

Dans le cadre de notre projet, nous avons pris quelques images simples puis nous les avons bruitées.
Ainsi, pour une image nous prenons sa matrice, puis nous rajoutons un bruit gaussien.

Image de base :

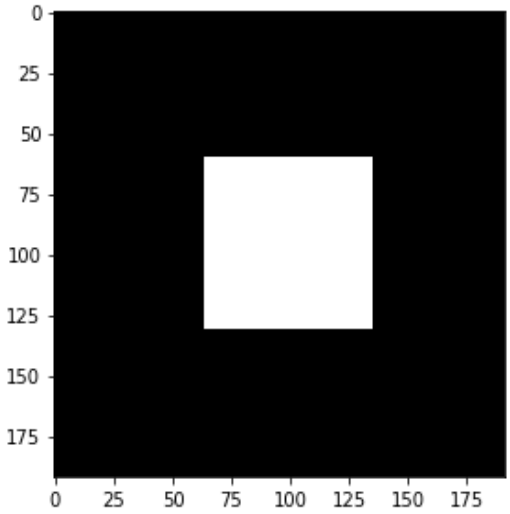
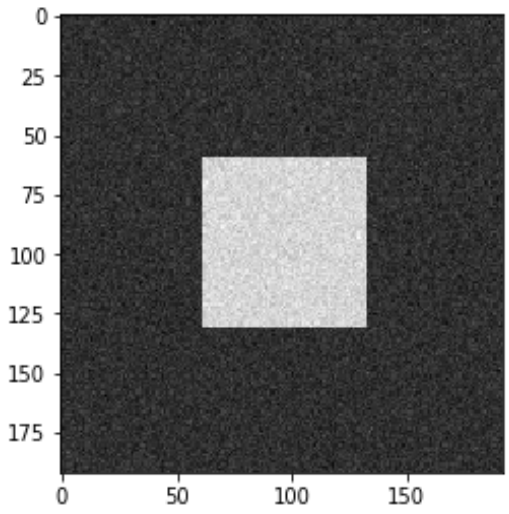


Image bruité :



Ensuite nous allons commencer notre traitement de l'image bruitée en suivant les étapes développées dans le papier de recherche. Tout d'abord, dans l'article de High-Dimensional Mixture Models For Unsupervised Image Denoising (HDMI) publié par Antoine Houdard, Charles Bouveyron, Julie Delon. Ils utilisent cet algorithme pour la classification.

Algorithm 2 The HDMI inference algorithm

Require: the noisy patches $\{y_1, \dots, y_n\}$, the number K of groups, the noise variance σ^2.

Ensure: parameter estimates $\{\hat{\mu}_k, \hat{Q}_k, \hat{a}_{kj}, \hat{d}_k;\ k = 1, ..., K,\ j = 1, ..., d_k\}$ and BIC value for the HDMI model.

Initialisation Run the k-means algorithm for K groups on $\{y_1, \dots, y_n\}$.

Set $t_{ik} = 1$ if y_i is in group k and 0 otherwise.

Set $lex \leftarrow -\infty$, $dl \leftarrow \infty$.

while $dl > \epsilon$ **do**

M step Update the estimates for $\theta = \{\pi_k, \mu_k, Q_k, a_{kj}, d_k;\ k = 1, ..., K,\ j = 1, ..., d_k\}$.

$$\hat{\pi}_k = \frac{1}{n}\sum_i t_{ik},\ \hat{\mu}_k = \frac{1}{n\hat{\pi}_k}\sum_{i=1}^{n} t_{ik} y_i,\ (\hat{Q}_k, \hat{\lambda}_k) = \text{eigendec}(S_k).$$

where $S_k = \frac{1}{n\hat{\pi}_k}\sum_{i=1}^{n} t_{ik}(y_i - \hat{\mu}_k)(y_i - \hat{\mu}_k)^t$.

Compute the intrinsic dimension $\hat{d}_k$ thanks to algorithm 1.

Set $\hat{a}_{kj} = \hat{\lambda}_{kj}$ for $j = 1, \dots, \hat{d}_k$. Set the $p - \hat{d}_k$ last column of $\hat{Q}_k$ to 0.

E step Compute the probabilities $t_{ik} = P(Z = k|y_i, \hat{\theta})$ as follows

$$t_{ik} = \frac{\hat{\pi}_k p(y_i; \theta_k)}{\sum_{\ell=1}^{K} \hat{\pi}_\ell p(y_i; \theta_\ell)}.$$

Update the likelihood $l = \sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k p(y_i; \theta_k)$ and compute the relative error between the two successive likelihoods $dl = |l - lex|/|l|$.

$lex \leftarrow l$.

end while

Compute the BIC $\leftarrow 2l - m\log(n)$, where m is the number of free parameters of the model.

Ensuite celui là pour la reconstruction :

Algorithm 3 The unsupervised HDMI image denoising algorithm.

Require: A noisy grey image u, a patch size s, a range $[\sigma_{min}, \sigma_{max}]$ and a discretization step σ_{step} for the noise standard deviation, a number of groups K.

Ensure: A denoised image $\widehat{u}$.

Patch Extraction Extract all $s \times s$ patches from u, to obtain $\{y_1, \dots, y_n\}$.

Inference and model selection

for σ from σ_{min} to σ_{max} by σ_{step} **do**

Model inference Run algorithm 2 to obtain $\hat{\theta}_\sigma$ and the corresponding BIC value.

end for

Select the model $\hat{\theta} = \hat{\theta}_\sigma$ with the largest BIC.

Denoising

for $i = 1$ to n **do**

compute $\widehat{y}_i = \sum_{k=1}^{K} \hat{\pi}_k \left(\hat{\mu}_k + \hat{\widetilde{Q}}_k(\mathrm{I}_p - \hat{\sigma}^2\hat{\Delta}_k^{-1})\hat{\widetilde{Q}}_k^t(y_i - \hat{\mu}_k)\right)$,

end for

Aggregate all patches $\widehat{y}_i$ to compute $\widehat{u}$.

En ce qui nous concerne, nous avons choisi d'utiliser un critère d'arrêt plus simple, puis on utilise les moyennes des patchs de même classe pour reconstituer notre image.

6.3 Implémentation via Python

6.3.1 Dispatching d'Image

On commence par par décomposer notre image en un ensemble de patchs de taille $n \times m$ qu'on définira ultérieurement. Pour cela nous créons une fonction qui permet de créer nos patchs.

6.3.2 Initialisation des labels

Suivant les pas du papier de recherche, nous choisissons d'initialiser nos labels par l'intermédiaire de Kmeans.

Définition 5 (Kmeans) *Coutumièrement appelé l'algorithme des K moyennes en français, c'est un algorithme de classification non supervisé bâti sur ce principe :*
Étant donnés des points et un entier k, le problème consiste à diviser les points en k groupes, souvent appelés clusters, de façon à minimiser une certaine fonction. On considère la distance d'un point à la moyenne des points de son groupe ; la fonction à minimiser est la somme des carrés de ces distances.

Implémentation :

```
    def dist(a, b, ax=1):
    return np.linalg.norm(a - b, axis=ax)

    def k_means(patchs, K, rseed=2):
    X=patchs
    n_clusters=K
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]
    while True:
    labels = pairwise_distances_argmin(X, centers)
    for j in range(n_clusters) :
    new_centers = np.array([X[labels == j].mean(0)])
    if np.all(centers == new_centers):
    break
    centers = new_centers
    return centers, labels
    centers, labels2 = k_means(patch_bruite, 2, rseed=2)

```

Explication 6 *Cet algorithme prend en entrée la matrice contenant les patchs ainsi que le nombre de clusters.*
Il commence par une assignation aléatoire des patchs, ensuite on minimise la distance via la méthode décrite en haut :

$$ArgMin_{S} \sum_{i=1}^{k} \sum_{X_j \in S_i} ||X_j - \mu_i||^2$$

Ensuite nous passons au calcul des centres en moyennant sur les points jusqu'à ce que les centres ne varient plus.

Cette première étape faite, elle permet de nous enlever au moins une itération de l'e.m. (5 min de temps de calcul économisées pour une image simple.)

6.3.3 Classification des patchs

Pour cette étape nous utilisons l'algorithme E.M. que nous avons implémenté via quatre fonction :

- **Premièrement, l'étape expectation :**

  Calcul des probabilités du mélange :

```
    def prob(val, mu, sig, lamba):
    p = lamba
    for i in range(len(val)) :
    p = p*norm.pdf(val[i], mu[i], sig[i][i])
    return p

```

```
    def expectation(dataFrame, parameters):
    dataFrame2=dataFrame
    for i in range(dataFrame2.shape[0]):
    x = dataFrame2[1][i]
    y = dataFrame2[2][i]
    p_cluster1 = prob([x, y], list(parameters['mu1']), list(parameters['sig1
'']), parameters['lambda'][0] )
    p_cluster2 = prob([x, y], list(parameters['mu2']), list(parameters['sig2
'']), parameters['lambda'][1] )
    if p_cluster1 > p_cluster2:
    dataFrame2[0][i] = 0
    else:
    dataFrame2[0][i] = 1
    return dataFrame2

```

- Maximisation :

```
    def maximization(dataFrame, parameters):
    cluster1 = assign_pt(dataFrame, 0)
    cluster2 = assign_pt(dataFrame, 1)
    percent_assigned_to_cluster1 = len(assign_indexe(dataFrame , 0)) / float
(len(dataFrame))
    percent_assigned_to_cluster2 = 1 - percent_assigned_to_cluster1
    parameters2= parameters
    parameters2['lambda'] = [percent_assigned_to_cluster1 ,
percent_assigned_to_cluster2 ]
    parameters2['mu1'] = [cluster1[1].mean(),cluster1[2].mean()]
    parameters2['mu2'] = [cluster2[1].mean(), cluster2[2].mean()]
```

```
    parameters2['sig1'] = [ [cluster1[1].std(), 0 ], [ 0, cluster1[2].std() ] ]
    parameters2['sig2'] = [ [cluster2[1].std(), 0 ], [ 0, cluster2[2].std() ] ]
    return (parameters2)


```

♣ Sélection du modèle :

```
def distance(old_params, new_params):
dist = 0
for i in ['mu1','mu2']:
for j in range(old_params.shape[0]):
dist += (old_params[i][j] - new_params[i][j]) ** 2
return dist ** 0.5

```

Cette fonction permet de calculer notre critère d'arrêt. Ensuite nous allons créer notre fonction qui fait jouer Kmeans et l'algorithme e.m. jusqu'à ce que notre critère choisi soit satisfait.

```
def algo_em_2coul(dataFrame, params, epsilon):
shift = 0
iters = 0
x=np.ones((dataFrame.shape[0],2))
# randomly assign points to their initial clusters
x[:,0]=dataFrame[1]
x[:,1]=dataFrame[2]
centers, labels = k_means(x, 2)
dataFrame[0] = labels
df_copy = dataFrame
while shift > epsilon:
iters += 1
# E-step
updated_labels = expectation(df_copy.copy(), params)
# M-step
updated_parameters = maximization(updated_labels, params.copy())
# verifie si nos estimation ont change
# selection du modele
shift = distance(params, updated_parameters)
df_copy = updated_labels
params = updated_parameters
return df_copy



df3= algo_em_2coul(df, guess, 0.01)

```

♣ Enfin on calcule la moyenne de chaque classe de patchs. Pour cela, on utilise une première fonction qui prend les labels générés par l'e.m. puis qui retourne les indices des patchs (exemples pour deux

classes 0 et 1 : on va retourner les indices des membres de la classe 0 puis ceux de la classe 1).
Ensuite, grâce aux indices, on calcule les moyennes des patchs de chaque classe.
Enfin, on remplace les patchs par leurs moyennes.

6.4 Reconstruction de l'image

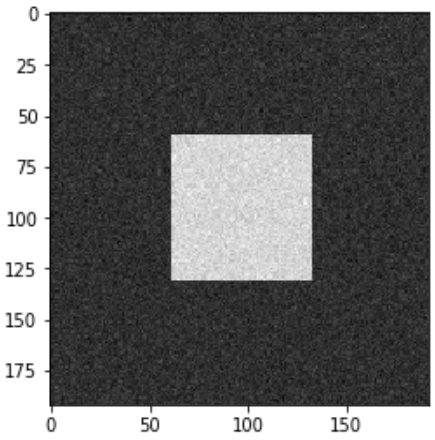
Notre classification terminée, nous avons des patchs moyens. Il est grand temps de recréer notre image supposée nette.
Pour cela, on crée une première matrice qu'on remplira avec nos nouveaux patchs. Puis, on reconstruit notre matrice image.

```
    df_moyene= nv_data_creator(df3)
    mat_im_moy=np.zeros((df_moyene.shape[0],df_moyene.shape[1]-1))
    for i in range (df_moyene.shape[0]) :
    mat_im_moy[i, :] = df_moyene.loc[i,1:9]

    def patch_dextractor(patchs, patch_size, image) :
    l,k = patch_size
    a, b = patchs.shape
    m, n = image.shape
    imagerec= np.zeros((m,n))
    vec=[]
    compteur=0
    for i in range (a) :
    for j in range (b) :
    vec.append(patchs[i,j])
    for i in range (m) :
    for j in range (n) :
    imagerec[i,j]=vec[compteur]
    compteur += 1
    return imagerec

```

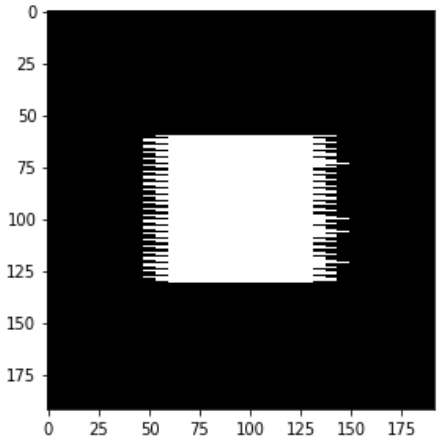
7 Résultats obtenus

7.1 Application sur une image simple

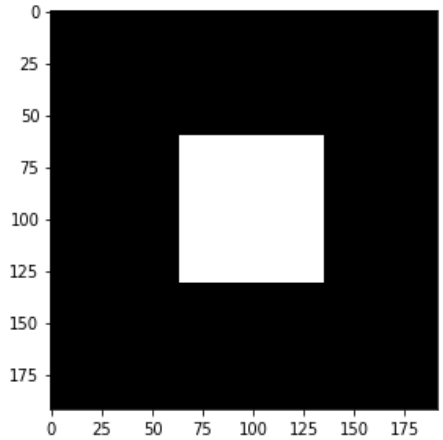
A partir d'une image de base bruitée avec une gaussienne :



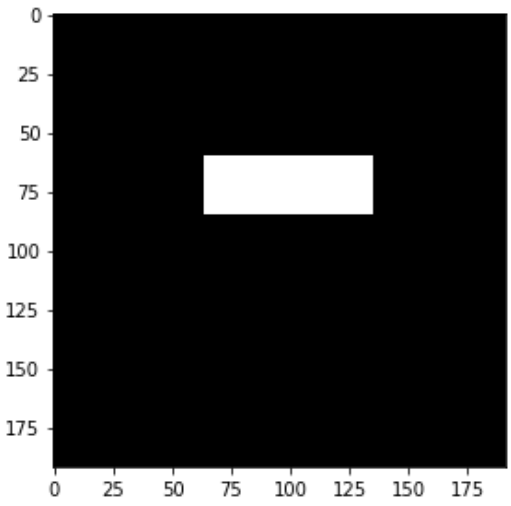
Reconstruction de cette image via des patchs 3x3 :



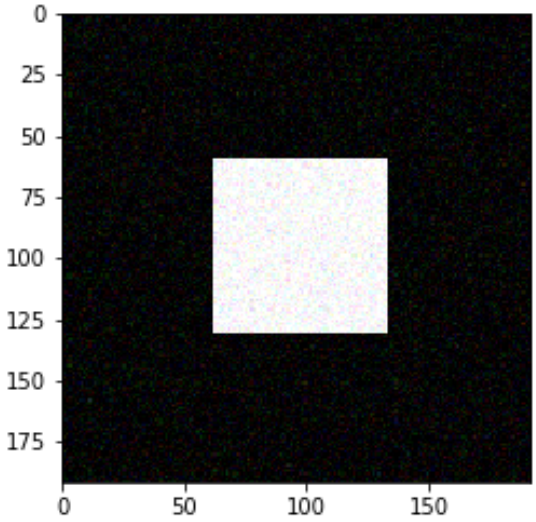
Reconstruction de cette image via des patchs 2x2 :



On peut interpréter ce résultat en disant que l'image du milieu a été reconstituée grâce à des patchs de taille 3,3.
Une interprétation simpliste consisterait à dire que dans les zones où on a deux ou plusieurs classes de patchs, on se retrouve dans le cas du sous apprentissage.
Afin d'améliorer notre classification nous passons à des patchs 2,2 ce qui réduit le taux d'erreur pour des images simples à 0 %.
Pour clore cette section nous pouvons parler du fait que plus on augmente la taille des patchs plus on tombe sur une mauvaise image. Le meilleur exemple serait notre résultat avec des patchs (8,8).



Ensuite un autre résultat pertinent qu'on souhaiterai exposer serait :



Dans cette image, on peut remarquer qu'on a un très mauvais débruitage cela est du au fait que l'on réduit nos patchs à des points.
On rentre ainsi dans ce que l'on appelle le dilemme biais-variance : En quelques mots, il provient du problème de minimiser simultanément deux sources d'erreurs qui empêchent les algorithmes d'apprentissage de généraliser au-delà de leur échantillon d'apprentissage.
Dans notre cas, ce dilemme peut s'intituler nombre de patchs/taille des patchs.

8 Conclusion

L'objectif des travaux présentés dans ce projet est la mise en place d'un code Python permettant de prendre une image bruitée, et de la débruiter après l'avoir découpée en patchs et utilisé les algorithmes de classification k-means et EM. La dernière étape de notre programme est de pouvoir moyenner ces patchs par classe puis de balayer la matrice obtenue pour reconstruire la matrice image et donc obtenir une image non bruitée.

Tout au long de ce projet, nous nous sommes heurtés à quelques difficultés. La première d'entre elles était la compréhension de l'article sur lequel nous nous sommes basés. En effet, l'article était parfois très compliqué au vu de l'avancement de ses auteurs dans le sujet. De plus, nous avons dû apprendre la programmation en Python avant de pouvoir appréhender le problème.

Malgré les difficultés rencontrées, nous avons pu faire tourner notre code, et avons pu vérifier qu'il fonctionnait bien sur des images simples (carré blanc sur fond noir par exemple). Cependant, il nous a été très compliqué de faire fonctionner notre programme sur des images plus réalistes car le temps d'exécution était trop long (dû au nombre important de classes contenues dans les images).

Références

[1] E.M.

[2] K-moyennes.

[3] Tutoriel Scit-image.

[4] Tutoriel Scit-learn.

[5] Charles Bouveyron. *Modélisation et classification des données de grande dimension : application à l'analyse d'images.* Mathématiques [math], Université Joseph-Fourier - Grenoble I,, Grenoble, 2006.