# Rapport - semaine 1 - juin 2020

## Création de la base des donnés

J’ai écrit une script .m pour créer plusieurs hologrammes en utilisant Matlab / Octave. L’objectif est de générer plusieurs hologrammes avec une quantité différente des sources dans la scène 3D, on commence avec 1 source ponctuelle dans une position 3D calculé de manière aléatoire et on monte jusqu’à 5 sources ponctuelles.

Par rapport à la génération aléatoire des positions 3D des particules, les positions x, y respectent les limites du plan d’hologramme et la position z respect une profondeur maximale et minimale.

Il est important de dire qu’on enregistre trois fichiers, un fichier avec les hologrammes, un fichier avec les images restituées dans une profondeur fixée et un fichier avec toutes les positions des sources.

#### Exemple :

Pour un numéro de classes égal à 5 et avec 200 hologrammes de dimension 2mm x 2mm et de résolution 200 pixels x 200 pixels par classe, les tailles des fichiers avec les hologrammes et les images restituées ont environ 600 Mo chaque une. L’image suivante détaille l’exemple précédent :

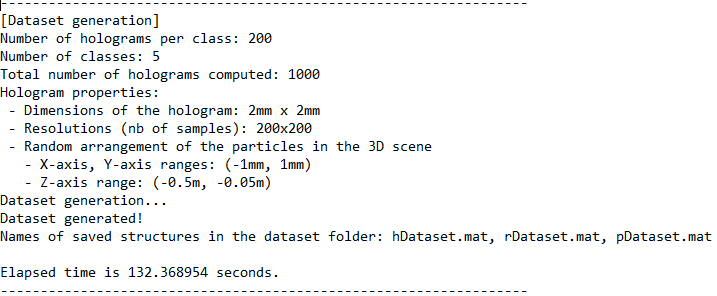


Figure 1 : sortie de la fenêtre de commande du script

#### Exemple :

Exemple des hologrammes générés :



Figure 2: code pour charger la base de données

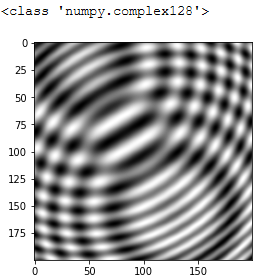
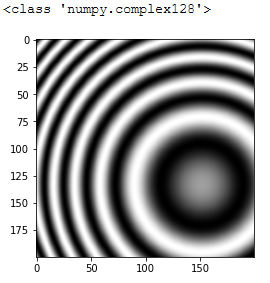


Figure 3 : exemples des hologrammes avec 1 source ponctuelle (à gauche) et deux sources ponctuelles (à droite)

#### Problèmes résolues :

Le script a généré des valeurs réelles pour les hologrammes et les images restituées, par contre ce problème a été déjà résolu et maintenant on a des valeurs complexes comme on peut vérifier avec l’image ci-dessus.

#### **Problèmes à résoudre :**

Quelques améliorations à faire :

* Pour l’instant, la restitution de tous les hologrammes est faite pour une même profondeur définie au début du code. De cette façon, les images restitués ne sont pas utiles, car la fonction de restitution d’image focalise la restitution seulement pour une distance dans l’axe z, donc on n’arrive pas à trouver la source avant de faire la restitution. Peut-être, il n’est pas nécessaire cette donnée dans l’étude.

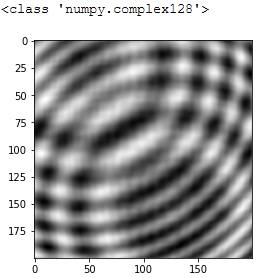
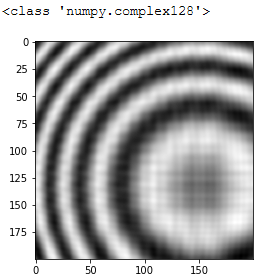


Figure 4 : images restituées pour les exemples ci-dessous à la distance 0.2m du plan de l’hologramme

#### Observations :

Quelques remarques à faire :

* Quand j’ai utilisé l’Octave pour exécuter le script pour la génération de la base de données, j’ai trouvé quelques limitations et problèmes dans le processus de sauvegarde. Si on veut faire, par exemple, une base des données avec 5 classes et avec plus de 25 hologrammes par classe, on a des erreurs pour sauvegarder les dictionnaires dans le format Matlab. Je remarque aussi que les fichiers sont plus gros quand ils sont générés par l’Octave. Donc, c’est intéressant d’éviter l’Octave.

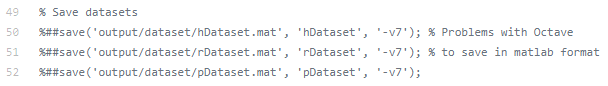
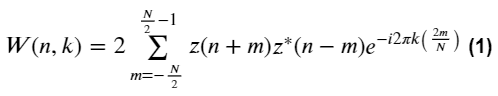


Figure 5 : code exemple pour sauvegarder en format Matlab

## Distribution de Wigner (WD)

Dès le début, j'ai cherché des références pour trouver des implémentations de la fonction Wigner, mais j'ai trouvé de nombreux articles et implémentations d'applications dans le domaine temps-fréquence et non pour l’espace-fréquence comme nous le souhaitions. Par contre, j’ai trouvé quelques références pertinentes, où je mets en évidence la référence [1] qui utilise la distribution de Wigner comme outil d’analyse de l’image avec une implémentation en python.

La WD a été approchée pour des signaux discrets sous le nom de pseudo-distribution de Wigner (PWD) :



Dans l’équation (1), la variable z(n) représente la valeur de gris du pixel n dans une image donnée z. Une PWD directionnelle peut être calculée en utilisant cette équation dans une fenêtre inclinée de 1D, pixel par pixel, L, conforme à une chaîne de valeurs z = z (-N/2), ..., z(0), ..., z (N/2). La position centrale de la fenêtre correspond exactement au pixel n de l'image. Equation (1) peut être interprété comme la transformée de Fourier discrète (TFD) du produit r (n, m) = z (n + m) z∗ (n - m). Ici z∗ indique le complexe-conjugué du signal z (notez que z = z∗ pour les images à valeur réelle). La somme est limitée à un intervalle spatial (-N/2, N/2-1). Dans l'équation (1), n et k représentent respectivement les variables discrètes d'espace et de fréquence, et m est un paramètre de décalage, qui est également discret. Ainsi, une PWD de l'image orientée pixel par pixel peut être calculée en décalant la fenêtre sur tous les pixels de l'image.

Dans les figures suivantes, on a appliqué la fonction de distribution de Wigner implémenté dans la référence [1] avec une fenêtre de taille 9 et un angle de 0 degrés.

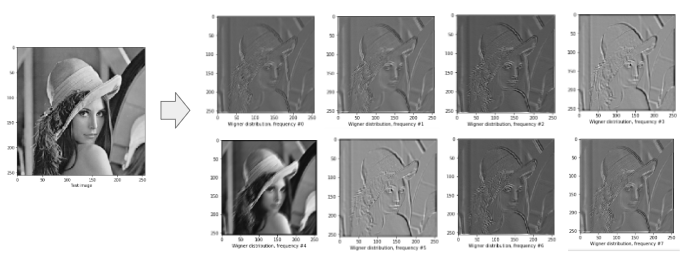


Figure 6 : 1D PWD sur l’image Lena

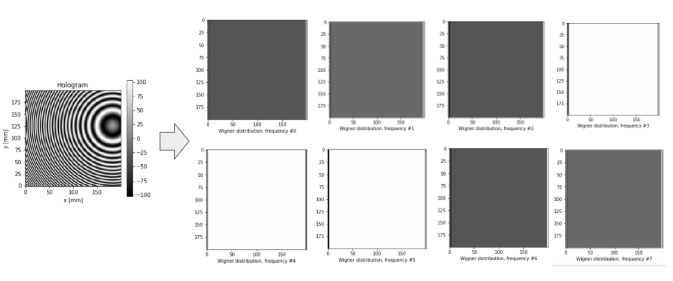


Figure 7 : 1D PWD sur les valeurs absolues de l’hologramme

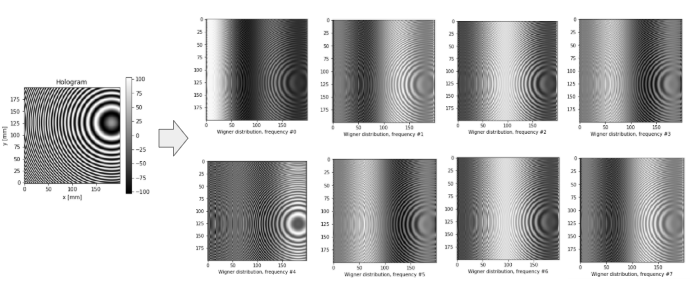


Figure 8 : Figure 2: 1D PWD sur les valeurs réelles de l’hologramme

#### **Problèmes à résoudre :**

Comme on peut bien noter avec les images ci-dessus, les résultats de la distribution de Wigner ne sont pas conformes aux attentes. Par contre, je suis en train de faire une nouvelle fonction pour calculer la distribution de Wigner, mais je n’arrive pas aux résultats attendus.

#### **Modification de l’équation :**

Je suis en train de faire.

#### Reference :

[1] DIGITAL IMAGE PROCESSING IN THE SPACE-FREQUENCY DOMAIN :

<https://notebooks.azure.com/salva/projects/Digital-Image-Processing>

[2] Analyse spectrale locale de l'image par transformation de Wigner-Ville :

<http://documents.irevues.inist.fr/bitstream/handle/2042/1797/005.PDF%20TEXTE.pdf?sequence=1>

[3] Image Quality Assessment Using the Joint Spatial/Spatial-Frequency Representation :

<https://www-l2ti.univ-paris13.fr/~beghdadi/jasp2006_SNR_W.pdf>

## Machine Learning

Dans un premier moment, pour essayer de résoudre le problème de classification, j’ai essayé quelques algorithmes d’apprentissage automatique. Comme on a beaucoup des données (dans notre cas, la distribution de Wigner gère 8 images de sortie pour une image d’entrée), la première chose à faire c’est choisir les caractéristiques à utiliser, car comme nous sommes dans une application de *machine learning* si on a beaucoup des *features* on peut avoir le problème d’overfitting*.*

J’ai décidé travailler avec les 12 *features* utilisés dans un projet d’apprentissage automatique avec la base des données MNIST (images 28x28 pixels des lettres manuscrites) fait à l’école. Les *features* sont :

* la variance de l'image
* la variation standard de l'image
* la moyenne arithmétique de la magnitude du filtre sobel
* la variation standard de l'ampleur du filtre sobel
* la moyenne arithmétique de l'angle du filtre sobel
* la variation standard de l'angle du filtre sobel
* la moyenne arithmétique du nombre de pixels sur l'axe des x
* la variation standard du nombre de pixels sur l'axe des x
* le nombre de valeurs non nulles dans le tableau du nombre de pixels sur l'axe des x
* la moyenne arithmétique du nombre de pixels sur l'axe des y
* la variation standard du nombre de pixels sur l'axe des y
* le nombre de valeurs non nulles dans le tableau du nombre de pixels sur l'axe des y

Comme on a 8 « couches » pour chaque exemple, les *features* pour un exemple est la moyenne arithmétique des *features* des 8 images résultats de la fonction de distribution de Wigner, et à cause de cette façon de calculer, le temps de calculer est augmenté.

Après, j’ai fait la division de la base de données pour entrainer l’algorithme (80% de la base originale) et le tester (20%) et ensuite j’ai fait la normalisation des caractéristiques afin d’éviter que certains caractéristiques soient plus fortes que d’autres.

J’ai fait l’implémentation des méthodes *K-Nearest Neighbors*, *Decision Tree* et *Support Vector Machine* (SVM) en utilisant la bibliothèque scikit-learn en python. Les résultats de ces méthodes sont affichés ci-dessous, il faut noter que comme on a utilisé une quantité petite pour entraîner les méthodes, 100 exemples utilisés pour le *trainset*, on a obtenu une précision de 100% pour les 3 méthodes.

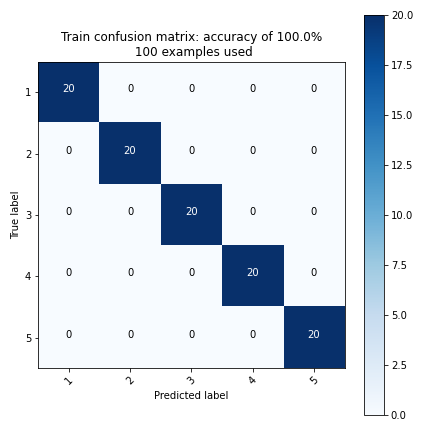
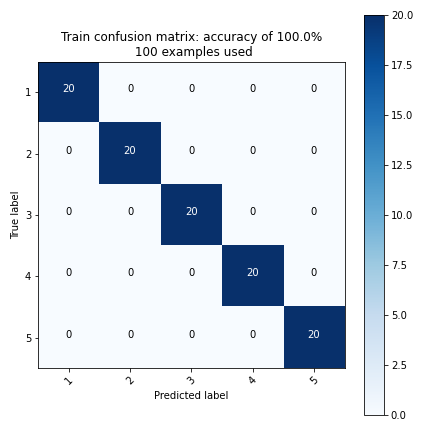
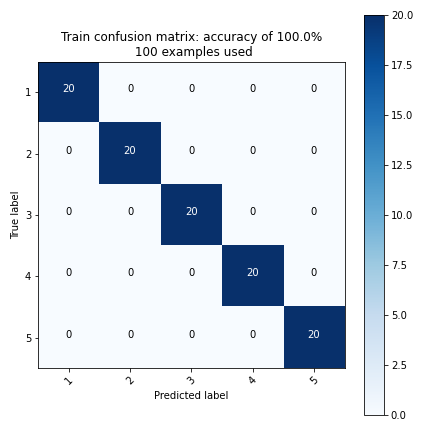


Figure 9 : Matrices de confusion avec les résultats d’entrainent des méthodes KNN, Decision Tree et SVM, respectivement

#### **Problèmes à résoudre :**

Quelques situations à surmonter :

* Si on va continuer à travailler avec des méthodes d’apprentissage automatique il faut mieux choisir les paramètres pour caractériser la base des données.
* Il faut améliorer la fonction de Wigner utilisé car elle ne donne pas le résultat attendue et le calcul demande beaucoup de temps.

#### Reference :

[4] Scikit-learn :

<https://scikit-learn.org/stable/>

## Convolutional Neural Network (CNN)

Pour essayer de résoudre le problème de classification, j’ai utilisé un réseau de neurones. J’ai décidé d’utiliser l’architecture d’un réseau neural convolutif car ce réseau est beaucoup utilisé pour les applications avec des images. Par rapport à la bibliothèque choisie pour faire l’implémentation, j’ai choisi utiliser la bibliothèque Keras, car elle est une API facile à utiliser et est le plus populaire *framework* pour l’apprentissage profond (*Deep learning*).

Il faut rappeler que les réseaux des neurones sont des algorithmes stochastiques, c’est-à-dire que le même algorithme sur les mêmes données peut entraîner un modèle différent à chaque fois que le code est exécuté.

#### Tests :

Pour une base de données avec 125 hologrammes, où pour chaque classe (5 classes au totale) on avait 25 hologrammes, j’ai créé une CNN avec les paramètres suivantes, il faut remarquer que j’ai utilisé les mêmes paramètres de la référence utilisé [2].

Les paramètres à souligner sont  que pour la première couche Conv2D, on a mis 64 nœuds et une matrice de filtrage 3x3, et pour la deuxième couche Conv2D, on a mis 32 nœuds et une matrice de filtrage 3x3.

Après entrainer le model pour un numéro de epoch égal à 30, epoch représente le nombre de fois que l'algorithme parcourt tout la base des données. Pour le test fait, l’entrainent a pris 20 minutes dans mon ordinateur personnel et on a obtenu une précision de 96%.



Figure 10 : Dernier epoch de l'exemple précédente

L’image suivante est un résumé des caractéristiques du CNN implémenté :

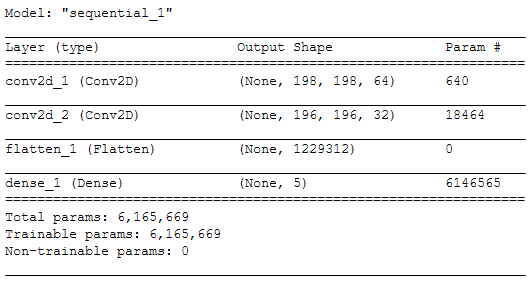


Figure 11 : résumé du réseau implémenté

#### **Problèmes à résoudre :**

Les problèmes urgents à résoudre sont :

* La fonction de la distribution de Wigner 1D utilisée donne pour une image comme entrée plusieurs sorties. Dans notre étude, la distribution crée 8 images comme sortie pour une image comme entrée, mais j’ai pris une des images crées pour entrainer le réseau. De plus le temps de calcul de la distribution est exorbitant, c’est pour ça qu’on a utilisé qu’une base des données avec 25 hologrammes par classe. Le temps de calcul pour 125 hologrammes était environ 20 minutes.

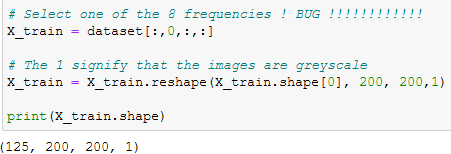


Figure 12 : exemple du problème cité ci-dessous

* Pour choisir les paramètres du réseau de neurones on doit faire plus des tests, comme par exemple, le batch, le epoch, la taille des filtres pour les couches convolutifs, la quantité des nœuds dans le couche.

#### **Conclusion :**

J’ai discuté avec Antonin et il m’a dit que utiliser il n’est pas nécessaire d’utiliser un réseau de neurones pour résoudre notre problème, car on peut exploiter les lignes gérés pour la distribution de Wigner, par contre comme je n’ai pas obtenu cette type d’image comme sortie, je ne suis pas sûr par rapport à la fonction utilisé.

#### Référence :

[5] Bibliothèque Keras :

<https://keras.io/>

[6] Building a Convolutional Neural Network (CNN) in Keras :

<https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5>

[7] Your First Deep Learning Project in Python with Keras Step-By-Step :

<https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>