RAPPORT PROJET 4A IMDS

Analyse et génération d'empreintes digitales par ACP

Polytech Clermont
Mathis Baumard et Thomas Mestrou

Table des matières

Table des figures

Remerciements

Nous tenons à remercier notre tuteur M. Grollemund, pour nous avoir supervisé tout au long de notre projet. Nous le remercions plus particulièrement pour sa disponibilité et son engagement dans notre projet.

Glossaire

ACP : L'analyse en Composantes Principales (ACP), est une méthode statistique consistant à transformer des données corrélées entre elles (dans notre cas, des pixels de différentes couleurs), en des données non-corrélées. Cette méthode permet ainsi de résumer le nombre d'informations que nous avons à notre disposition.

Bruit : Le bruit est une variable que nous rajoutons à un vecteur pour en modifier la valeur sans pour autant trop modifié celui-ci.

CAH : La Classification Ascendante Hiérarchique (CAH) est une méthode statistique qui permet de rapprocher les individus en fonction de la distance avec les autres. Cette méthode s'applique avec une métrique que nous pouvons choisir.

Indice d'importance : Pour le chemin d'image, un indice d'importance correspond au coefficient a_{ij} de la matrice. Cet indice indique le poids de l'image correspondante dans le chemin d'images. Par exemple, si le coefficient vaut 1, alors l'image est la seul à peser dans la projection, mais si celui-ci vaut 0.25, alors l'image n'exerce qu'une influence mineure.

Projection : Une projection d'un axe est le fait d'afficher les valeurs d'un axes sur un graphique.

Vecteur multivarié : Un vecteur multivarié est un vecteur provenant d'une loi normale multidimensionnelle. Cette loi est simplement une généralisation de la loi normale. À la différence d'un loi normale qui a pour paramètres une moyenne et un écart type, la loi multinormale a pour paramètres un vecteur et une matrice de covariance.

Introduction

Les empreintes digitales, tout comme la reconnaissance vocale ou faciale, font partie de ce qu'on appelle la biométrie. La biométrie est une méthode d'authentification de plus en plus utilisée de nos jours, car elle permet une connexion rapide et sécurisée (pour débloquer son téléphone, il suffit de poser son doigt sur un bouton spécial, au lieu de rentrer son code). Mais son utilisation massive est aussi dû à une surévaluation de la sécurité de ces méthodes par les utilisateurs.

Cependant, ces méthodes ne sont que très récentes, et les premières recherches sur le sujet datent seulement du début du XXIème siècle. Alors a-t-on raison de faire une confiance aveugle en ces moyens d'authentification et de sécurité qui n'en sont qu'à leurs balbutiements ?

Dans ce contexte, nous voulons trouver des moyens de créer de nouvelles empreintes digitales, qui n'appartiennent pas à des personnes existantes, mais qui sont assez ressemblantes pour pouvoir être utilisées, pour verrouiller un téléphone par exemple. Pour cela, nous partons d'une base de données d'empreintes existantes, et nous appliquons une Analyse en Composantes Principales. Après celle-ci, nous faisons plusieurs modifications sur les données transformées par ACP, ce qui génère de nouvelles images. Nous mettons ensuite en place une méthode qui calcule la ressemblance de deux empreintes. Finalement, nous compilons toutes les étapes appliquées sur nos images dans un script Python qui automatise ses transformations et générations d'images.

1. Analyse des images par Analyse en Composantes Principales (ACP)

1.1. Les données et l'importation

Au préalable des explications concernant les détails de l'utilisation de l'ACP dans ce contexte, nous présentons succinctement les données que nous avons à notre disposition. La base de données est composée de douze images au format .bmp, ce qui correspond à des images en échelle de gris (voir la figure 1 pour un exemple d'images d'empreintes digitales). Autrement dit, chaque image correspond à un tableau de valeurs entières indiquant l'intensité de gris pour chacun des pixels, à savoir 0 pour un pixel noir et 255 pour un pixel blanc. L'avantage de disposer d'images dans ce format est que le traitement qui en découle ne se faite que sur un tableau. Au contraire, une image en couleur est numérisée en trois tableaux, correspondants pour chaque pixel aux intensités de bleu, de rouge et de vert, ce qui rend plus lourd et complexe l'analyse de l'image.

Figure 1 : Trois empreintes digitales différentes

D'un point de vue de code, pour ouvrir les images, nous utilisons la bibliothèque *PIL* de Python. Avec la fonction *Image* de cette bibliothèque, en spécifiant le chemin ver le dossier où est stocké l'image à importer, nous obtenons les valeurs numériques relatives à l'intensité de gris de chacun des pixels. De cette manière-là, nous importons l'ensemble des images de la base de données.

De plus, en guise de prétraitement avant l'application de l'ACP, nous vectorisons les matrices relatives à chacune des empreintes. En effet, la méthode de l'ACP s'applique sur une table où chaque vecteur ligne correspond aux mesures d'un individu statistique, évalué sur un ensemble de variables (en colonnes). Nous avons donc besoin de mettre les données des images sous ce format. Dans ce contexte, un individu statistique correspond à une image, et une variable correspond à l'intensité d'un pixel donné. Nous obtenons donc une table dont le nombre de colonnes est le nombre de pixels des images, à savoir « longueur x largeur », et dont le nombre de lignes correspond au nombre d'images dans la base de données.

À noter qu'étant donné la procédure que nous présentons, il est nécessaire d'avoir des images au même format et de même dimension. Cela implique une phase de prétraitement supplémentaire si la base d'images ne vérifie pas cette contrainte. Cette dernière consiste à effectuer des « zooms » ou des « dézooms » sur certaines des images, de sorte à disposer d'images de la même dimension et donc de vecteurs lignes de la même longueur.

1.2. Projection et reconstruction des images

Pour projeter les images, nous utilisons la méthode *PCA* de la librairie de *scikit-learn.* Elle nous permet de réaliser une ACP simplement, en choisissant le nombre d'axes sur lequel on souhaite projeter les images. De plus, nous pouvons facilement reformer une image à partir de la projection des images à l'aide de la fonction *inverse_transform* de cette même librairie.

L'ACP consiste à ne garder que les informations importantes d'un vecteur. Ce procédé transforme donc une image qui possède « longueur x largeur » informations, à une image comportant un nombre d'informations égal au maximum au nombre d'individus (images). Nous pouvons don appliquer des méthodes de transformations plus rapidement et plus facilement sur ces images « réduites ». Pour appliquer cette ACP, nous devons choisir le nombre d'axes à conserver.

Le choix du nombre d'axes dépend des individus, nous ne pouvons donc pas établir, une règle pour effectuer ce choix automatiquement. Nous faisons donc varier ce nombre entre 1 et le nombre d'images que nous avons dans notre dossier, c'est-à-dire douze. Nous obtenons les images ci-dessous, respectivement après avoir effectué une ACP avec deux, cinq et dix axes, ainsi que l'image de base.

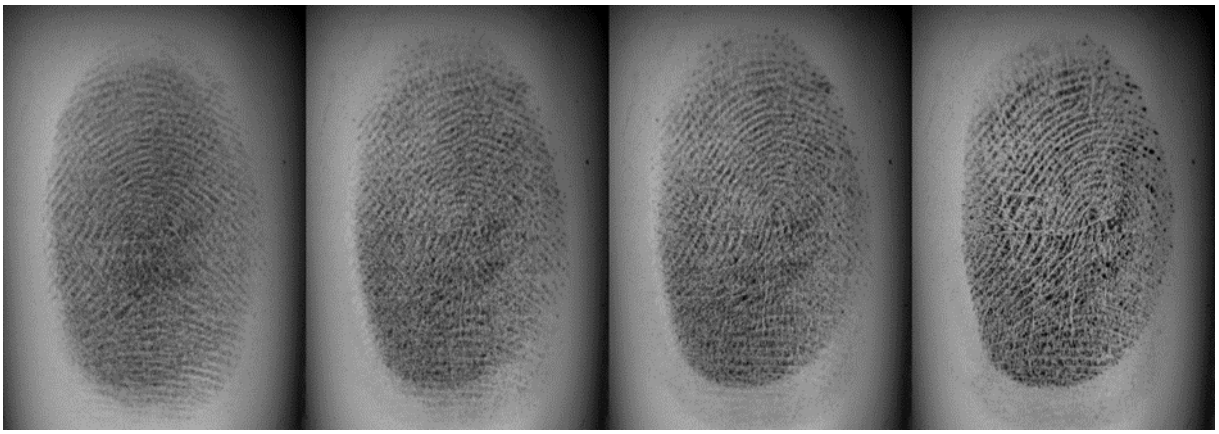


Figure 2 : Représentation des projections d'empreintes digitales avec différents nombres d'axes principaux conservés

Nous pouvons observer que la meilleure ACP se produit avec dix axes conservés. Certains axes rajoutent peu d'informations, et nous obligent donc de prendre un nombre plus élevé. À partir de onze axes principaux conservés, l'image est trop ressemblante à l'image de base, et nous perdons donc l'intérêt de notre étude, qui consiste à générer des empreintes différentes.

1.3. Distances entre les images

Après avoir créé de nouvelles images, nous calculons la distances entre ces dernières. Cette distance, que nous appelons aussi « score », compare deux images et renvoie un nombre représentant la différence entre celles-ci. Cette dernière varie entre 0 (les deux images sont identiques) et 1 (les images sont totalement différentes). Par exemple, si nous comparons une image entièrement blanche à une image entièrement noire, alors la distance entre ces deux images est de 1.

Pour calculer la distance entre ces deux images, nous les découpons en plusieurs rectangles que nous comparons un par un. Chaque pixel possédant une valeur entre 0 et 255, nous calculons la moyenne des valeurs du rectangle 1 de l'image 1, puis celle du rectangle 1 pour l'image 2, et finalement nous faisons la différence en valeur absolue de ces deux valeurs. Nous appliquons cette méthode sur tous les rectangles, et enfin nous divisons le « score » par le nombre de rectangles.

En ayant fait de nombreux tests entre des images -générées ou non- nous remarquons qu'une différence de l'ordre de 2% correspondait à une image assez différente mais qui reste acceptable.

```
img1 = cv2.imread('D:/Mathis/Mes Images/Empreintes/1_1.bmp',0)
img3 = cv2.imread('D:/Mathis/Mes Images/Empreintes/1_7.bmp',0)

print("La différence entre l'empreinte 1 et l'eempreinte 7 est : ",score(img1,img3,56*2,40))
```

```
La différence entre l'empreinte 1 et l'eempreinte 7 est :  0.05014569320451674
```

```
A = 255 * np.ones((560,400))
B = np.zeros((560,400))

print('La différence entre une image blanche et une image noire est : ',score(A,B,56*2,40))
```

```
La différence entre une image blanche et une image noire est :  1.0
```

Figure 3 : Représentation de la distance entre les images

Sur l'image ci-dessus, nous voyons que la différence entre une image blanche et une image noire est de 100%, tandis que la différence entre 2 images considérées comme différentes est de 5%.

1.4. Distance entre une et plusieurs images

Dans notre programme principal, nous sommes amenés à calculer le score entre une image créée et toutes celles à partir desquelles nous l'avons créé. Cela nous permet de vérifier que cette dernière ne ressemble pas totalement à une des images de la base de données, et de pouvoir récolter le maximum d'informations sur le score pour les utiliser par la suite. Nous implémentons alors la fonction *compare* qui effectue ses calculs et renvoie un tableau des scores.

2. Méthodes de génération de nouvelles images

2.1. Bruitage des projections

Pour générer de nouvelles images, plusieurs modèles se sont offerts à nous et l'un d'entre eux est le bruitage des projections. Celui-ci permet de modifier les projections que nous avons par l'ACP, de sorte à obtenir de nouvelles images étant considérées comme « différentes », sans pour autant rajouter un bruit trop important et donc rendre l'empreinte « illisible ».

La question principale est donc : comment bien choisir ce bruit ?

Nous réfléchissons dans un premier temps à la façon d'ajouter le bruit à cette projection. En effet, il faut que nous rajoutions un bruit qui ne soit pas le même sur chaque axe, car les valeurs sont très différentes. Si l'on prend la projection de l'image 6 sur l'axe 0 et sur l'axe 9, on voit que les valeurs sont de 7616 et -48. Ainsi, si on rajoute un bruit de 50 par exemple, cette modification exerce une influence mineure sur l'axe 0 (on rajoute un bruit de 0.65%) tandis que cette modification est majeure sur l'axe 9 (on rajoute un bruit de 104%, soit 160 fois supérieur à l'axe 1). On peut voir ces informations sur la figure ci-dessous.

	0	1	2	3	4	5	6	7	8	9
0	-4879.703636	-193.461038	-4393.917169	909.871143	2418.857198	229.271934	490.397136	-597.842264	-1993.037642	-2564.216769
1	-6696.522345	-855.841589	-4102.961130	-721.692269	-324.932203	-1548.576513	522.897261	3662.285199	7759.779566	410.652337
2	-3547.516990	3126.955261	-3090.721041	289.282223	1685.287061	-1125.705252	-366.668736	-175.369986	-3697.833973	7590.153971
3	844.850074	-7590.261582	1143.336344	-2778.795771	-3644.419240	1406.812954	7200.142853	2979.066698	-2792.916175	188.952298
4	-2344.825714	2417.006528	-3790.617769	946.945453	3175.219247	32.928063	668.203213	-1206.073899	-3093.138045	-4929.072404
5	7616.470694	-1860.480462	-4954.429135	-4544.795476	-5208.607338	-1552.016173	-2818.845157	-5216.358056	883.725901	-48.978516
6	2341.408872	-5550.513347	875.477633	10684.257142	-2622.385176	95.549920	-2404.827477	-132.441125	165.381245	305.390413
7	9264.611565	-335.820412	2401.494396	406.944254	7855.892516	-386.343949	3615.597036	-2412.699119	3113.857865	1006.055757

Figure 4 : Dataframe avec l'image 6 mise en lumière

Nous devons aussi prendre en compte que ce bruit doit être rajouté de manière aléatoire. Si nous voulons créer plusieurs nouvelles images à partir d'une seule, il faut obligatoirement que ce bruit soit aléatoire, et non pas une valeur fixe.

Finalement, pour répondre à ces deux contraintes, nous décidons de choisir un vecteur multivarié. Ce vecteur possède donc le même nombre de valeurs que d'axes choisis dans l'ACP et ces valeurs sont cohérentes avec l'ordre de grandeur de l'axe. Ce vecteur a donc pour moyenne 0 et pour écart-type la matrice de covariance entre les différents axes. Cette matrice équivaut à :

```
[[126.33738128   0.           0.           0.           0.
    0.           0.           0.           0.           0.        ]
 [  0.         140.87088855   0.           0.           0.
    0.           0.           0.           0.           0.        ]
 [  0.           0.         112.50378822   0.           0.
    0.           0.           0.           0.           0.        ]
 [  0.           0.           0.         123.40604774   0.
    0.           0.           0.           0.           0.        ]
 [  0.           0.           0.           0.         114.30004311
    0.           0.           0.           0.           0.        ]
 [  0.           0.           0.           0.           0.
  128.22338589   0.           0.           0.           0.        ]
 [  0.           0.           0.           0.           0.
    0.         113.16685135   0.           0.           0.        ]
 [  0.           0.           0.           0.           0.
    0.           0.         111.7807752    0.           0.        ]
 [  0.           0.           0.           0.           0.
    0.           0.           0.         107.04024262   0.        ]
 [  0.           0.           0.           0.           0.
    0.           0.           0.           0.         111.8893488 ]]
```

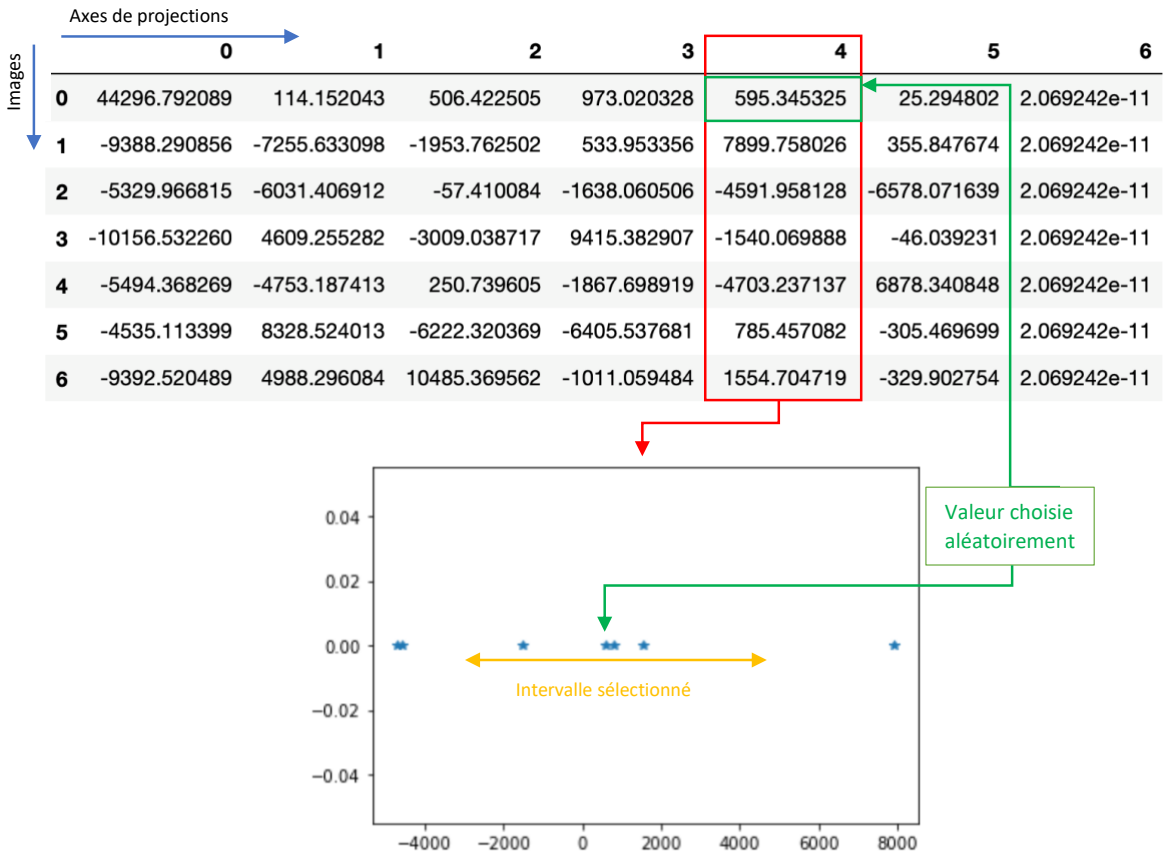
Figure 5 : Matrice de covariance du vecteur multivarié

Sur la diagonale, nous avons mis une valeur qui est égale à la racine carrée de l'addition des valeurs absolues de la plus petite valeur de l'axe concerné et de la plus grande. La matrice est nulle pour les coefficients car les axes sont indépendants les uns des autres. En ayant fait de nombreux tests sur les valeurs de ce vecteur, nous déterminons que ce vecteur multivarié doit être multiplié par le nombre d'axes de l'ACP pour être le plus efficace.

2.2. Images moyennes

2.2.1 Moyenne évoluée par axe

Dans cette méthode de moyenne d'image, nous partons du constat que pour chaque image, les axes de l'ACP sont des valeurs réelles. Il est donc possible de faire des calculs statistiques simples sur ces valeurs telles qu'une moyenne, un écart-type… À parti de ce constat, l'objectif est de créer une nouvelle valeur pour chaque axe en fonction de celles des images, sans qu'elles soient trop éloignée de celles-ci. Nous avons donc choisi la méthode suivante :pour chaque axe de l'ACP, nous prenons aléatoirement une valeur x (correspondant à la projection d'une image sur l'axe étudié). Puis nous calculons l'écart-type std de toutes les valeurs d'un axe (nous sélectionnons les valeurs ci-dessous).



	0	1	2	3	4	5	6
0	44296.792089	114.152043	506.422505	973.020328	595.345325	25.294802	2.069242e-11
1	-9388.290856	-7255.633098	-1953.762502	533.953356	7899.758026	355.847674	2.069242e-11
2	-5329.966815	-6031.406912	-57.410084	-1638.060506	-4591.958128	-6578.071639	2.069242e-11
3	-10156.532260	4609.255282	-3009.038717	9415.382907	-1540.069888	-46.039231	2.069242e-11
4	-5494.368269	-4753.187413	250.739605	-1867.698919	-4703.237137	6878.340848	2.069242e-11
5	-4535.113399	8328.524013	-6222.320369	-6405.537681	785.457082	-305.469699	2.069242e-11
6	-9392.520489	4988.296084	10485.369562	-1011.059484	1554.704719	-329.902754	2.069242e-11

Figure 6 : Représentation d'une projection

À partir de ces données, nous sélectionnons les valeurs des axes se situant dans l'intervalle : $[x - std; x + std]$. Une fois ces valeurs récupérées, nous faisons une simple moyenne de ces dernières qui sera la valeur de l'image pour l'axe étudié.

Dans le cas ci-dessus, nous étudions l'axe 4 des projections de l'ACP. L'algorithme a sélectionné aléatoirement la valeur 595,34 et l'écart-type est de 3988,79. L'intervalle dans lequel nous sélectionnons les valeurs est donc : [-3393,44 ; 4584,13]. Les valeurs de l'axe récupérées sont alors : {-1540,07 ; 595,34 ; 785,46 ; 1554,70}, dont nous faisions la moyenne pour obtenir la valeur de la projection sur l'axe 4 de notre nouvelle image : 346,86.

Cette méthode a l'avantage de pouvoir créer des empreintes très différentes les unes des autres, puisqu'on choisit aléatoirement la valeur centrale. Cependant, l'image créée est souvent peu ressemblante à une empreinte. C'est pour cela que cette méthode est à privilégier lorsque la base de données est composée d'énormément d'images ou après une autre méthode statistique, telle que la Classification Ascendante Hiérarchique (CAH) que nous allons désormais voir.

2.2.2. Classification Ascendante Hiérarchique

Afin de créer plusieurs empreintes différentes et relativement proches de celles dans la base données, nous avons mis en place une méthode statistique bien connue : une Classification Ascendante Hiérarchique.

Dans notre contexte, le principe de cette méthode est le suivant : nous calculons la distance euclidienne entre les images (après ACP) grâce à la fonction *linkage* de la librairie *SciPy*. Nous allons ensuite former le dendrogramme associé à se distances. Une fois cette partie effectuée, l'enjeu est de faire intervenir l'aléatoire. En effet, si nous prenions un critère constant pour la formation des groupes, nous créerions les mêmes images. Pour répondre à cette problématique, nous avons opté pour une solution semblable à celle vue dans la partie Moyenne évoluée par axe.

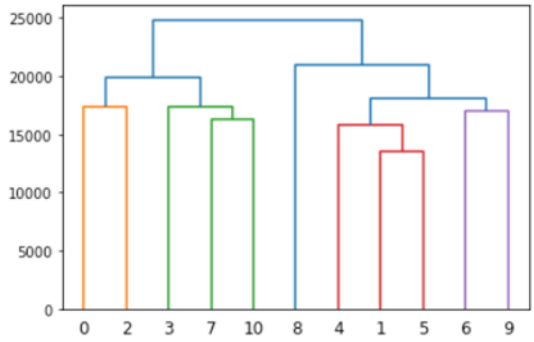


Figure 7 : Dendrogramme d'une CHA sur 11 empreintes digitales

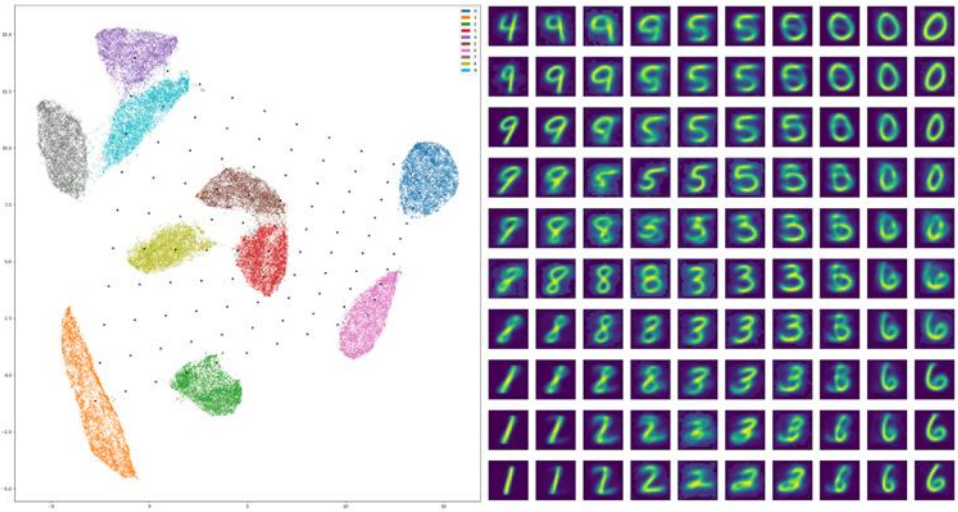
Nous avons récupéré les hauteurs de chaque nœud (que vous pouvez voir sur la figure ci-dessous), calculé la moyenne m et l'écart-type std de ces dernières, puis pris une valeur aléatoire dans l'intervalle $[m - std; m + std]$. Cette valeur sera la hauteur à laquelle nous formerons les groupes. Nous choisirons ensuite aléatoirement un groupe parmi ceux créés et ferons une moyenne simple ou par axe, qui nous donnera les valeurs de l'image nouvellement créée.

2.2.3. Moyenne simple par axes

Dans notre programme final, nous avons implémenté une deuxième méthode de calcul d'un nouvel axe. Cette dernière est très simple et consiste uniquement à calculer la moyenne de chaque axe pour former un vecteur sur lequel on applique l'inversion de l'ACP pour obtenir une image. Cette méthode étant très basique, elle ne sert qu'après une autre méthode comme la CAH ou comme référentiel pour construire une image que l'on considère comme très mauvaise.

2.3. Chemin d'images

Durant une de nos réunions avec notre professeur référent, une image nous a beaucoup intéressé. Celle-ci représente un chemin d'images provenant d'une étude sur les réseaux de neurones et sur la reconnaissance et génération d'images par rapport à des nuages de points. Nous appliquons cette méthode à notre étude de cas pour observer si cette solution produit des empreintes que nous pouvons considérer comme acceptables.



Pour faire ce chemin d'images, nous choisissons 4 images. Après ce choix, nous créons les matrices nécessaires pour celui-ci. Nous calculons ensuite des indices d'importance, ou poids, pour chaque valeur. Les images étant dans les coins de notre image reconstituée, il faut donc une matrice par image et que la somme des différentes matrices pour les mêmes coordonnées (en i et en j) vaille 1. Ainsi, les matrices nommées M, N, O et P sont créées.

```
M
 [[1.         0.75       0.5        0.25       0.        ]
 [0.75       0.5        0.375      0.16666667 0.        ]
 [0.5        0.375      0.25       0.125      0.        ]
 [0.25       0.16666667 0.125      0.16666667 0.        ]
 [0.         0.         0.         0.         0.        ]]
```

Figure 9 : Matrice d'importance

Ici, on peut voir la matrice M, qui correspond à la matrice de l'image située en haut à gauche de notre chemin d'images. Les matrices N, O et P sont obtenues en effectuant des rotations à 90 degrés de M. Ainsi, la matrice N aura la valeur 1 en haut à droite, O en bas à droite et P en bas à gauche.

Une fois les matrices créées, nous multiplions chaque image – et plus exactement les axes obtenus par l'ACP – par la valeur dans la matrice correspondante. Pour la première image, nous multiplions les axes de l'image 1 par M[1][1], ceux de l'image 2 de N[1][1], etc. Nous additionnons les axes des 4 images et opérons une ACP inverse pour recréer l'image. Nous obtenons donc une nouvelle image qui est considérée comme un mélange des 4 images sélectionnées.

3. Utilisation du script

3.1. Fichier .json

Le fichier .json est nécessaire à la bonne utilisation de notre programme car il permet de rendre notre code transportable. La raison principale réside dans le fait que les images des empreintes sont stockées dans des dossiers dont le chemin d'accès diffère selon les ordinateurs. Il faut donc trouver un moyen pour que l'utilisateur précise le dossier dans lequel se situent les images, en amont de la compilation. De même, d'autres paramètres peuvent être modifiés dans ce fichier.

```
{
    "DossierRecupImages": "/Users/Thomas/_4A_21_22/Proj_4A/sous_echantillon",
    "DossierNouvellesImages": "/Users/Thomas/_4A_21_22/Proj_4A/sous_echantillon_cree",
    "tailleImageY": 400,
    "tailleImageX": 560,
    "Nombre_axe_ACP": 10,
    "NbImagesACreer": 15,
    "Methode": 3,
    "Afficher_images" : false,
    "Affichage_score": false
}
```

Figure 10 : Paramètres du fichier .json

Pour modifier le chemin du dossier dans lequel se situe les images, il faut modifier la ligne *DossierRecupImages*. Et pour modifier le chemin du dossier dans lequel seront créés les dossiers contenant les images créées, il faut modifier la ligne *DossierNouvellesImages*. Il est aussi nécessaire de modifier les dimensions des empreintes de la base de données, que l'utilisateur précise dans les lignes *tailleImageX* et *tailleImageY*. L'utilisateur doit préciser le nombre d'axes qu'il souhaite conserver pour l'ACP, ce chiffre étant obligatoirement compris entre 1 et le nnombre d'images de la base de données. La valeur *NbImagesACreer* correspond au nombre d'images que l'usager souhaite créer par la méthode *Methode*. Cette dernière est un entier correspondant à une méthode mise en place avec pour légende :

1 : Moyenne évoluée par axe ;
2 : Moyenne simple par axe ;
3 : CAH puis moyenne simple par axe ;
4 : CAH puis moyenne évoluée par axe ;
5 : Bruitage ;
6 : Vecteur multivarié.

Enfin, il est aussi possible de modifier les paramètres *Afficher_Image* et *Afficher_score*. Si l'utilisateur les initialise à *true*, ces derniers affichent plus d'informations. Le paramètre *Afficher_Image* affiche les images lors de la compilation et *Afficer_score* présente le « score » entre les images qui ont été crées et celles de la base de données.

3.2. Programme principal

Le programme principal se divise en 3 parties. L'enjeu est d'offrir à l'utilisateur du programme un affichage du déroulement interne de l'algorithme, sans pour autant surcharger l'écran de celui-ci. En effet, cela lui ferait perdre les informations importantes.

Dans un premier temps se déroule la phase d'initialisation. On affiche le début du programme avec des informations sur le jour et l'heure actuelle. Nous allons aussi ouvrir le fichier .json pour en récupérer les informations et appeler la fonction *ouvrir* qui va récupérer toutes les images, effectuer la projection par ACP et renvoyer des Dataframes contenant respectivement les informations et les projections des images par ACP.

Puis nous effectuons le traitement des images et la création des nouvelles empreintes. Nous commençons par créer et ouvrir un nouveau dossier, qui contiendra toutes les nouvelles images. Pour que ce dossier soit unique, nous lui donnons comme nom le jour et l'heure de sa création. Ensuite, nous séparons le cas où l'utilisateur souhaite utiliser la méthode du chemin d'image des autres. En effet, cette méthode crée un nombre d'images fixe (25). On ne peut donc pas effectuer une boucle qui tourne autant de fois que le nombre d'images qui est indiqué dans le fichier .json, comme c'est le cas dans la deuxième partie.

Pour les deux cas, nous implémentons une boucle *for* qui va répéter les instruction suivantes autant de fois que d'images à créer. Ainsi, pour chaque image créée, nous sauvegardons cette dernière dans le dossier prévu à cet effet. Puis nous calculons la distance (« score ») entre l'image nouvellement construite et toutes les images qui ont servi à la construction avant de l'ajouter à un Dataframe. À la fin de la boucle *for*, nous sauvegardons ce Dataframe dans un fichier dans le dossier.

Pour que l'affichage soit plus esthétique, nous utilisons la librairie *alive-progress* qui permet d'afficher la barre de progression sur le terminal. Cette barre de chargement progresse à chaque image ouverte (au début du programme) et créée, et ainsi indique à l'tilisateur l'avancement du programme.

Enfin, si l'utilisateur le souhaite, le programme va afficher les « scores » calculés précédemment. Nous séparons encore une fois le cas où l'utilisateur utilise la méthode 6 des autres. La seule différences est le nombre d'itération de la boucle (25 pour la méthode 6, le nombre choisi par l'utilisateur pour les autres). Pour chaque image réalisée, le programme affiche la moyenne, le minimum et le maximum du « score » avec toutes les autres images.

Lorsque le programme se termine, il affiche un message de fin afin de le notifier à l'utilisateur.

Pour appeler le programme depuis le terminal d'un ordinateur, il suffit alors d'écrire la ligne de code suivante :

```
python3 Projet4A_v2.py Donnees.json
```

Conclusion

Tout au long de ce projet, nous avons mis en place différentes méthodes pour créer de nouvelles empreintes à partir d'une base de données quelconque. Nous avons entre autres mis en place des méthodes mathématiques telles que la Classification Hiérarchique Ascendante, la moyenne évoluée et simple par axes d'ACP et le chemin d'images. Nous avons créé une méthode purement informatique qui ajoute de bruit à une image. Nous avons remarqué que ces méthodes s'adaptent bien à la base de données que nous possédons car les images se ressemblent suffisamment. Toutefois, lorsque les images sont différentes entre elles, nos méthodes sont assez faibles et les images créées sont pour la plupart extrêmement floues et inutilisables, ou alors trop ressemblantes à celles de la base de données. Il est donc nécessaire de faire un prétraitement des empreintes utilisées afin d'avoir un résultat satisfaisant.

Une fois ces méthodes instaurées, nous avons établi une fonction permettant de calculer la différence entre ces deux images. Le calcul de ce « score » étant relativement simple, le résultat n'est cependant pas très précis et peu représentatif de la différence entre deux images. Pour remédier à cela, nous pourrions mettre en place une méthode basée sur la reconnaissance des crêtes et des plis des empreintes afin de pouvoir réellement comparer les images créées et les anciennes. De plus, nous n'avions pas de moyen sûr et sur lequel nous pouvions nous baser pour pouvoir être certains de créer des empreintes utilisables.

Enfin, nous avons remarqué qu'après avoir créé quelques empreintes, certaines réapparaissaient. Il serait donc intéressant d'effectuer un contrôle après création des images, vérifiant qu'elles n'ont pas déjà été fabriquées.

Annexe

Annexe 1 : Reconstruction d'images

Après la création de nos nouvelles empreintes pour le chemin d'images, nous devons reconstruire une image complète présentée sur la page de garde. Pour cela, nous avons un nouveau fichier .json et un nouveau programme Python. Dans ce fichier .json, nous retrouvons les mêmes paramètres que dans le fichier .json de notre programme principal, avec le nom du fichier où récupérer les images, celui où remettre l'image mais aussi le nom de l'image.

Dans le fichier Python, nous avons d'abord la fonction *ouvrir* qui renvoie une liste contenant le nom des images.

Après cela, nous faisons un test qui nécessite d'avoir une liste de 25 éléments pour lancer le programme. Si cette condition est vérifiée, alors le programme tourne, et crée tout d'abord 5 images composées elles-mêmes de 5 images qui ont été concaténées verticalement. Finalement, nous mettons les 5 grandes images les unes sur les autres pour créer l'image finale.