

Compte Rendu Méthodes d'apprentissage

Réalisé par :
Meriam JARDAK
Saifeddine ELAMRI

Groupe : 2A ISN G2

I. Objectif :

Le but de ce Projet est de construire un classifieur de feuilles d'arbre par approche probabiliste en premier lieu et avec des réseaux de neurones convolutifs en second lieu.

Pour simplifier, on commencera dans un premier temps à considérer un problème à 4 classes (**"pimento", "papaya", "chrysanthemum", "chocolate tree"**) puis on enrichira progressivement le classifieur en disposant d'une base de données contenant 32 types de feuille.

Afin de réaliser ce classifieur, on va utiliser 3 méthodes :

- ✓ Supervisés
- ✓ Non supervisés
- ✓ Réseau de neurones convolutifs

II. Classification Probabiliste

➤ Extraction des caractéristiques

On commence tout d'abord par l'extraction et la sélection des caractéristiques (features) des images et qui se fait à l'aide de la fonction « **extractFeatures.m** » fournie sur Arche.

On obtient ainsi une matrice de données X de dimensions 60x38 contenant 38 caractéristiques morphologiques des 60 feuilles.

Code Matlab :

```
1 - clear all
2 - close all
3
4 - LeafType={'papaya','pimento','chrysanthemum','chocolate_tree'};%... à décommenter pour ajouter de nouvelles espèces
5 % 'duranta_gold','eggplant','figus','fruitcitere','geranium','guava',...
6 % 'hibiscus','jackfruit','ketembilla','lychee','ashanti_blood','mulberry_leaf',...
7 % 'barbados_cherry','beaunier_du_perou','betel','pomme_jacquot','bitter_orange',...
8 % 'rose','caricature_plant','star_apple','chinese_guava','sweet_olive','sweet_potato',...
9 % 'thevetia','coeur_demoiselle','vieux_garcon','coffee','crotton';
10
11 - K=length(LeafType);
12 - label=[];
13 - X=[];
14 - for LT=LeafType
15
16     filenames=dir([LT{1},filesep,'Training',filesep,'*.png']);
17
18
19 - for ifile=1:length(filenames)
20
21     img=imread([filenames(ifile).folder,filesep,filenames(ifile).name]);
22     X=[X;extractFeatures(img)];
23     label=[label,LT];
24     close all;
25
26 - end
27 - end
28
```

Exécution : La matrice X est bien générée



60x38 double

1. Visualisation des données

Dans une deuxième étape on va visualiser les données dans des sous-espaces de dimension 2 et ensuite 3 afin d'évaluer le caractère discriminant des caractéristiques extraites.

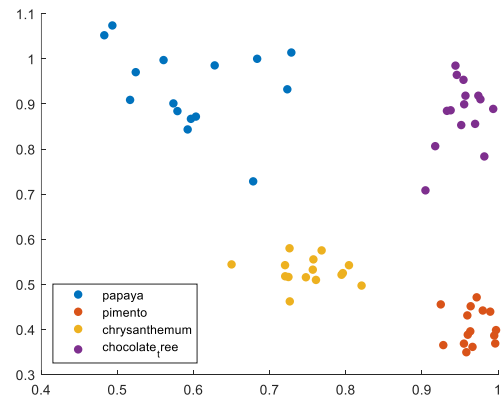
Le vecteur « **feat** » contient la dimension de l'espace (nombre de caractéristiques) sur lequel on aimerait visualiser les données.

Cas Dimension 2

```

32 %% Sélection manuelle des caractéristiques et visualisation
33
34 feat=[1 2]
35 Xs=X(:,feat);
36 figure(1), hold,
37 for LT=LeafType
38     Ilt=find(strcmp(label,LT));
39     scatter(Xs(Ilt,1),Xs(Ilt,2),'o','filled');
40     % pause
41 end
42 legend(LeafType(1:4),'Location','SouthWest');
43

```

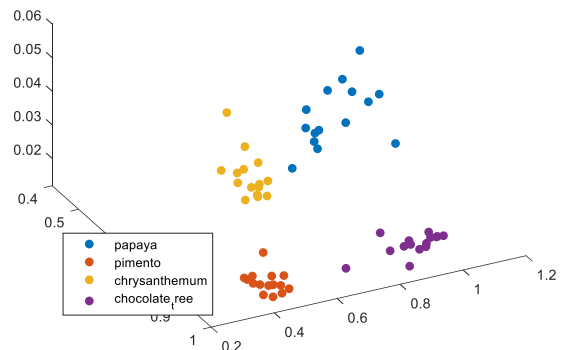


Cas Dimension 3

```

32 %% Sélection manuelle des caractéristiques et visualisation
33
34 feat=[1 2 3]
35 Xs=X(:,feat);
36 figure(1), hold,
37 for LT=LeafType
38     Ilt=find(strcmp(label,LT));
39     scatter3(Xs(Ilt,1),Xs(Ilt,2),Xs(Ilt,3),'o','filled');
40     % pause
41 end
42 legend(LeafType(1:4),'Location','SouthWest');
43

```



2. Réduction de dimension par ACP

Afin de simplifier le problème de classification, on va réduire la dimension de l'espace en procédant par l'analyse en composante principale sur la matrice de données X.

Cette méthode permet de déterminer les composantes principales qui permettent le mieux de distinguer entre les 4 classes.

Tout d'abord On va centrer les colonnes de la matrice X en retirant à chaque ligne de X la moyenne de cette matrice selon ses colonnes qu'on stockera dans la variable « **moyX** ».

Ensuite, on calcule à l'aide de la fonction « **eig()** » les valeurs et les vecteurs propres de la matrice de données X.

Puis à l'aide de la fonction « **svd** » de Matlab, on va obtenir la décomposition suivante :

$$\mathbf{X} = \mathbf{U} * \mathbf{D} * \mathbf{V}'$$

ou D contient les valeurs propres de la décomposition et V la matrice de passage dans l'espace des composantes principales.

Enfin on trie la matrice V par valeurs décroissantes des valeurs propres et on nomme Xp la matrice qu'on obtient par la multiplication de X avec la matrice des vecteurs propres V.

On restreindra Xp à ses ncp premières colonnes, correspondant aux composantes principales de X. Ce sont celles qui portent la plus grande partie de l'information contenu dans X.

Code Matlab :

```
46 %% Réduction de la dimension par ACP
47 moyX=mean(X);
48 X=X-moyX;
49 covV=X'*X;
50 [U,D,V]=eig(covV);
51 [d,Isort]=sort(diag(D),'desc');
52 V=V(:,Isort);
53 Xp=X*V;
```

Exécution :



60x38 double

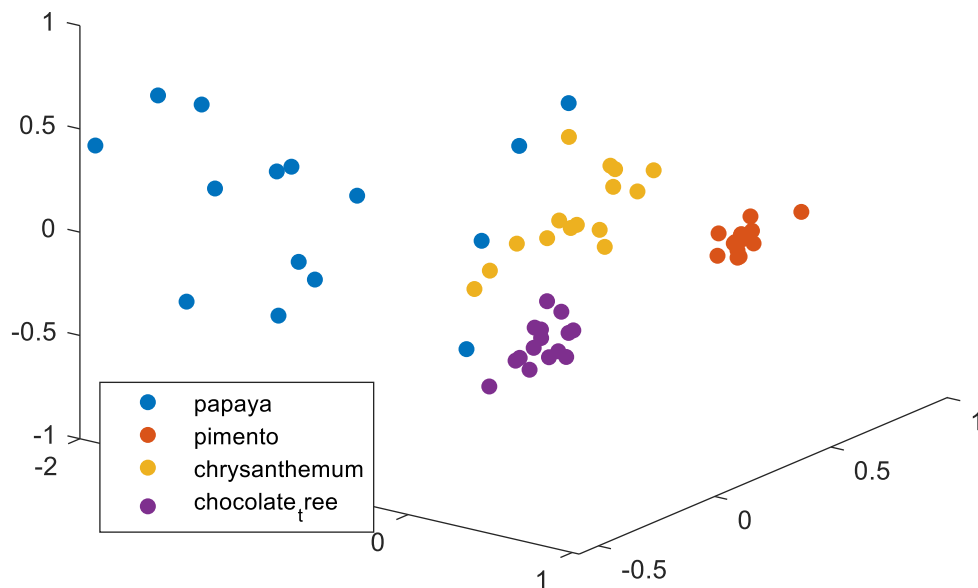
3. Visualisation des données après ACP

On fixe ncp =3

Code Matlab :

```
68 % visualisation en 3d: on regarde 3 composantes de l'ACP
69
70 comp=[1 2 3]; % 3 premières composantes par défaut, à modifier pour visualiser d'autres composantes
71
72 figure(2), hold,
73 for LI=LeafType
74     Ilt=find(strcmp(label,LI));
75     scatter3(Xp(Ilt,comp(1)),Xp(Ilt,comp(2)),Xp(Ilt,comp(3)),'o','filled');
76 end
77 legend(LeafType(1:4),'Location','SouthWest');
```

Figure :



On peut remarquer la différence de la distribution des données de celle sans ACP.

➤ Approches Supervisées

1. Méthode non paramétrique : Parzen

En adoptant un noyau gaussien on va calculer à l'aide de la fonction « **gaussParzen** » la vraisemblance d'une nouvelle observation x passée en paramètre ainsi que l'écart type du noyau gaussien et les données d'apprentissage.

Pour cela elle fait appel à la fonction « **mvnpdf** » qui retourne la densité de probabilité des différentes variables.

Code Matlab :

```
1 function z=gaussParzen(data,appr,sig)
2
3 % data: point(s) x
4 % appr: données d'apprentissage
5 % sig: std du noyau de parzen
6 %X=Ne*nc (nbre de caracteristiques)
7
8 Sigma=sig^2*eye(size(appr,2));
9 N=size(appr,1);
10 z=zeros(size(data,1),1);
11
12 for i=1:N
13     z=z+mvnpdf(data,appr(i,:),Sigma);%mvnpdf retourne la valeur de la densité de probabilité
14 end
15
16 z=z/N;
```

A l'aide de la fonction « **isocontoursParzen** » on peut visualiser les iso contours de ces vraisemblances et les superposer sur les données d'apprentissage.

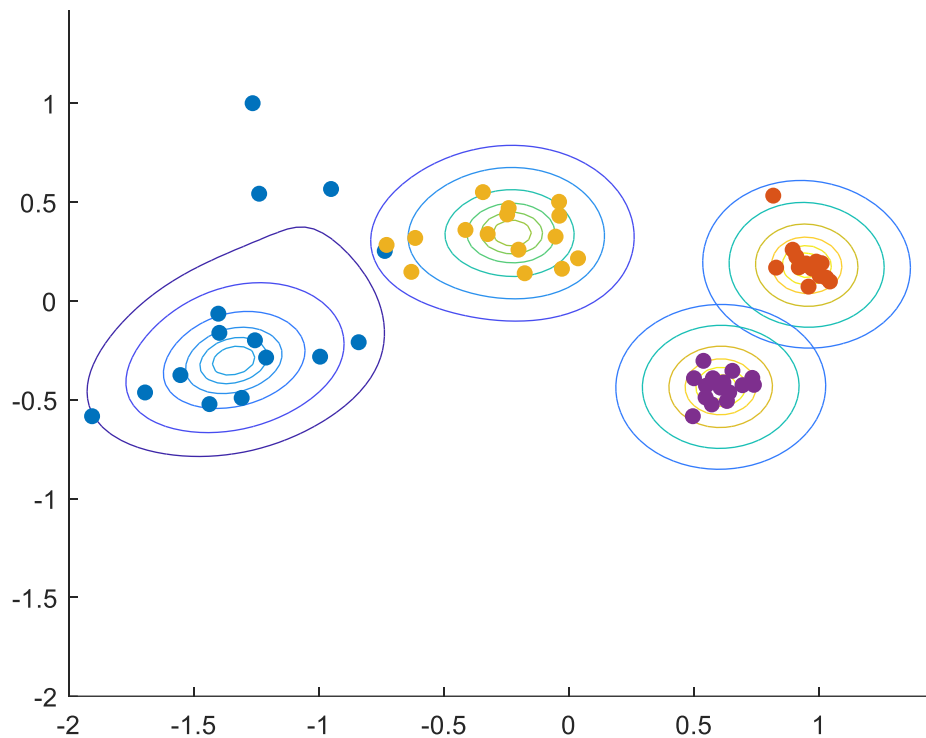
```
1 function [X,Y,Z]=isoContoursParzen(feats,sigParz)
2
3 % Aide pour l'affichage d'isocontours d'une loi en 2D
4
5 % determination du range de calcul
6 xmin=-2;
7 xmax=1.5;
8 ymin=-2;
9 ymax=1.5;
10
11 % discrétisation de chaque axe sur 100 points
12 pasx=(xmax-xmin)/100;
13 x=(xmin:pasx:xmax-pasx);
14 pasy=(ymax-ymin)/100;
15 y=(ymin:pasy:ymax-pasy);
16
17 % dtermination des coordonnées 2D (10000 points):
18 % le point (i,j) de l'espace a pour coordonnées (X(i,j), Y(i,j))
19 [X,Y]=meshgrid(x,y);
20
21 % vectorisation des 10000 coordonnées
22 XYvec=zeros(2,size(X,1)*size(X,2));
23 XYvec(1,:)=reshape(X,1,size(X,1)*size(X,2));
24 XYvec(2,:)=reshape(Y,1,size(X,1)*size(X,2));
25
26 % calcul de la distribution sur les 10000 points
27 Zvec=gaussParzen(XYvec',feats,sigParz);
28 Z=reshape(Zvec,size(X,1),size(X,2));
29
30
31
32 Nc=5; % nombre d'isocontours ? afficher
33 % figure,
34 M=max(max(Z));
35 v=M*[0.98 0.95 0.90 0.8 0.6 0.4];
36 contour(X,Y,Z,v);
```

Visualisation des contours :

Code Matlab :

```
95
96 % Visualisation du résultat en dimension 2
97 % isocontoursParzen fait appel à la fonction gaussParzen
98
99 sig=0.3; % écart-type du noyau gaussien
100
101 cp=[1 2]; % Choix de 2 dimensions à visualiser entre 1 et ncp
102 figure; hold on ;
103
104 for LT=LeafType
105     Ilt=find(strcmp(label,LT));
106     isoContoursParzen(Xp(Ilt,1:2),sig);
107     scatter(Xp(Ilt,cp(1)),Xp(Ilt,cp(2)),'o','filled');
108 end
```

Figure :

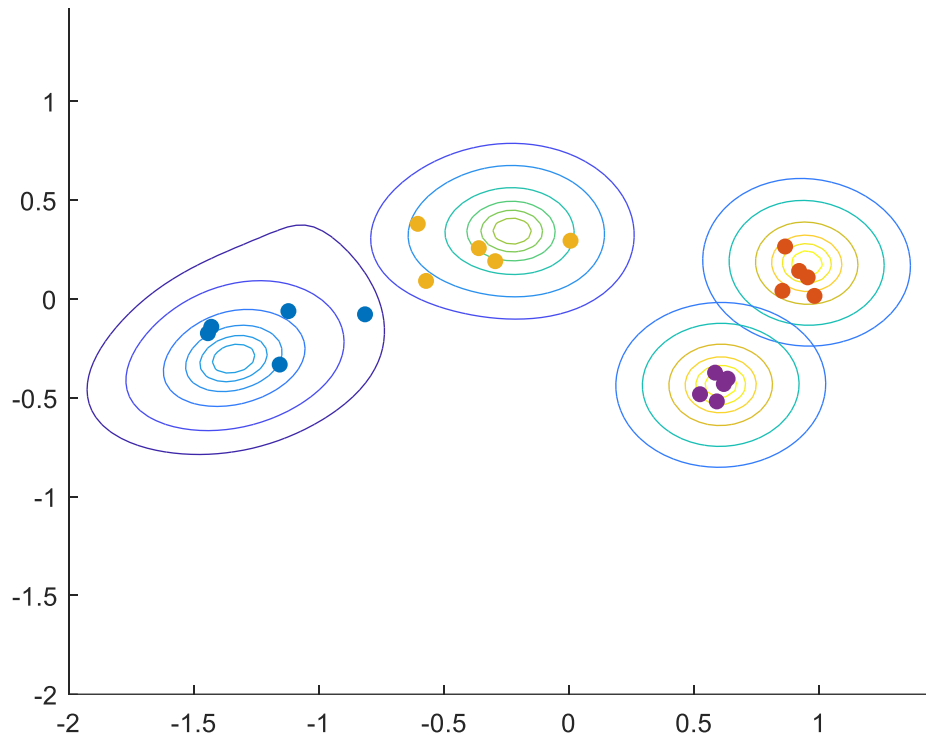


On passe maintenant à la phase de test, on commence tout d'abord par charger les données de test pour passer à la classification.

Code Matlab :

```
121
122 % Test sur les nouvelles données (non utilisées pour l'apprentissage)
123 %chargement des données de test
124 labeltest=[]
125 Xtest=[]
126 for LT=LeafType
127
128     filenames=dir([LT{1},filesep,'Test',filesep,'*.png']);
129
130
131     for ifile=1:length(filenames)
132
133         img=imread([filenames(ifile).folder,filesep,filenames(ifile).name]);
134         Xtest=[Xtest;extractFeatures(img)];
135         labeltest=[labeltest,LT];
136         close all;
137
138     end
139 end
140
141 %classification
142 Xtest=Xtest-moyX
143 Xtestp=Xtest*V
144
145 comp=[1 2]; % choix de 2 dimensions à choisir entre 1 et ncp
146
147 figure, hold,
148 for LT=LeafType
149     Ilt=find(strcmp(label,LT));
150     Ilt1=find(strcmp(labeltest,LT));
151     isoContoursParzen(Xp(Ilt,1:2),sig);
152     scatter3(Xtestp(Ilt1,comp(1)),Xtestp(Ilt1,comp(2)),Xtestp(Ilt1,comp(3)),'o','filled');
153 end
154
```

Figure :



Interprétation :

On constate que l'algorithme a bien réussi à classer toutes les données. On peut donc dire qu'on a un très bon taux de classification. On peut aussi essayer de classer encore plus de types de feuilles sur une base de données plus vaste et voir l'influence de cette variation.

Calcul des vraisemblances des données de test :

Afin de pouvoir classer les données de test on va calculer leurs vraisemblances à l'aide de la fonction « **gaussParzen** » qu'on stockera dans un tableau Z de taille 20 x 4

Code Matlab :

```
148 - Z=zeros(20,K);
149 - for i=1:20
150 -     for j=1:K
151 -         LT=LeafType(j);
152 -         Ilt=find(strcmp(label,LT));
153 -         Z(i,j)=mvnpdf(Xtestp(i,:),Xp(Ilt,:),sig);
154 -     end
155 - end
156 -
```

Exécution :

	1	2	3	4
1	0.3704	6.0371e-08	0.2371	2.4604e-05
2	0.4459	2.7427e-14	0.0078	3.1165e-10
3	0.5628	5.2756e-14	0.0015	1.8493e-10
4	0.5134	7.5491e-12	0.0197	1.2430e-07
5	0.5303	1.6664e-10	0.0410	1.1272e-07
6	8.5563e-08	1.7385	0.0081	0.4321
7	1.1974e-08	2.1214	0.0033	0.1871
8	6.8196e-09	1.8503	0.0020	0.2971
9	1.7016e-08	2.1192	0.0045	0.1342
10	4.9400e-08	2.0048	0.0092	0.0720
11	0.2006	4.1733e-06	0.8681	2.8113e-05
12	0.1811	3.6181e-06	0.6856	2.8543e-04
13	0.0657	3.6369e-04	1.1569	0.0037
14	0.0081	0.0100	0.9447	0.0184
15	0.0528	2.3554e-04	1.0095	7.2410e-04
16	3.7012e-06	0.0652	0.0081	1.9445
17	1.0355e-06	0.1316	0.0061	2.0899
18	2.3577e-06	0.1856	0.0120	1.9024
19	1.0719e-06	0.0628	0.0038	2.0170
20	7.8849e-07	0.1151	0.0054	2.0999

→ Chaque donnée de test appartient à la classe qui la plus grande vraisemblance

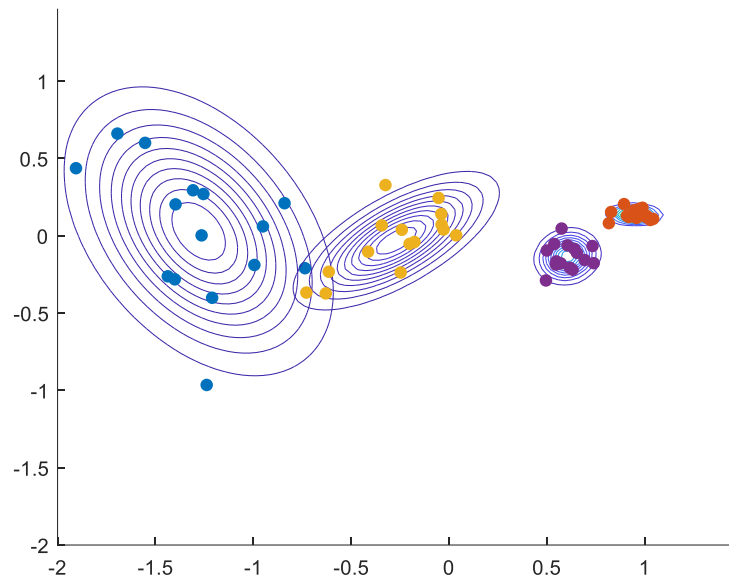
2. Méthode paramétrique Gaussienne

Dans cette méthode paramétrique les densités paramétriques employées seront des gaussiennes. On va déterminer dans un premier temps les moyennes et covariances des vraisemblances associées à chacune des quatre classes à l'aide des estimateurs du Maximum de Vraisemblance. On cherchera par la suite à faire correspondre les feuilles à leurs classes d'appartenance.

Code Matlab :

```
156
157 %% Paramétrique - Gaussiennes
158
159 % Training
160
161 K=length(LeafType);
162 muK=zeros(ncp,K);
163 CovK= repmat(eye(ncp,ncp),1,1,K);
164
165 % Évaluez ici muK et covK sur les données d'apprentissage Xp
166 for i=1:K
167     LT=LeafType(i)
168     Ilt=find(strcmp(label,LT));
169     muK(:,i)=mean(Xp(Ilt,1:ncp));
170     CovK(:,i)=cov(Xp(Ilt,1:ncp));
171 end
172
173 % code pour visualisation des iso-contours en 2d:
174
175 dim_visu=[1 3]; % on choisit 2 dimensions, ici par exemple la 1ère et la 3ème
176
177 figure(4); hold on;
178 i=1;
179 for LT=LeafType
180     Ilt=find(strcmp(label,LT));
181     isoContoursGauss(muK(dim_visu,i),CovK(dim_visu,dim_visu,i));
182     scatter(Xp(Ilt,dim_visu(1)),Xp(Ilt,dim_visu(2)),'o','filled');
183     i=i+1;
184 end
185
186
187
188 % Évaluation du classifieur au sens du Maximum de Vraisemblance
189 % sur les données d'apprentissage
190
191
```


Figure :

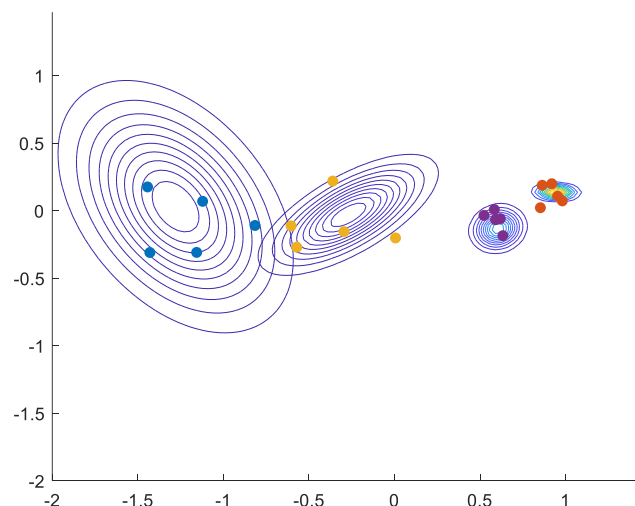


On passe maintenant à la phase de test, on commence tout d'abord par charger les données de test pour passer à la classification.

Code Matlab :

```
185 -  
186 - figure(5); hold on;  
187 - i=1;  
188 - for LT=LeafType  
189 -     Ilt=find(strcmp(labelTest,LT));  
190 -     isoContoursGauss(muK(dim_visu,i),CovK(dim_visu,dim_visu,i));  
191 -     scatter(Xtestp(Ilt,dim_visu(1)),Xtestp(Ilt,dim_visu(2)),'o','filled');  
192 -     i=i+1;  
193 - end  
194  
195
```

Figure :



Interprétation :

On remarque que l'algorithme a réussi à classer toutes les données de test en 4 classes puisque on voit bien que toutes les données sont superposées sur les iso contours des données d'apprentissage.

➤ Approches non Supervisées

Maintenant, on ne dispose pas de données d'apprentissage, la classification des données se fait à partir leur répartition dans l'espace des caractéristiques.

On va utiliser l'union des données d'apprentissage et de test en formant une matrice

$$X_{ns} = [X_p; X_{testp}]$$

1. Méthode des K-moyennes

L'objectif de cette méthode est de minimiser la variance intra-classe qui est équivalent à maximiser la variance inter-classe.

L'algorithme consiste à affecter itérativement chaque échantillon à la classe dont le centre lui est le plus proche.

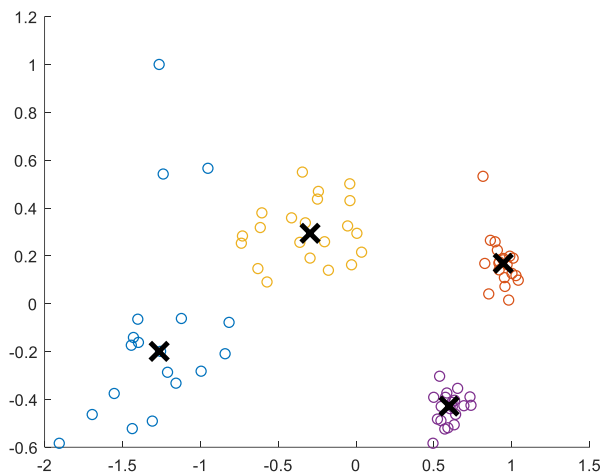
Code Matlab :

```
192 %% K-moyennes
193
194 Xns=[Xp;Xtestp] ;
195
196 opts = statset('Display','final');
197 [idx,C] =kmeans(Xns,K, 'Distance','cityBlock','Replicates',20,'Options',opts);
198 figure(10);
199 hold on
200
201 for k=1:K
202     plot (Xns(idx==k,1),Xns (idx==k,2),'o')
203 end
204 plot(C(:,1),C(:,2),'kx', 'MarkerSize',15,'LineWidth',3)
205 hold off ;
```

Remarque :

On a utilisé les matrices Xp et Xtestp des parties précédentes pour construire la matrice Xns et à l'aide de la matrice C construite.

Figure :



- ➔ Les centroïdes de chaque classe sont mis à jour à la fin de chaque itération.
- ➔ On utilise la fonction «**k-means**» et on choisit 7 initialisations différentes

- ➔ L'algorithme converge et il nous a permis de trouver le centre des différents groupements. Ainsi, on peut classer les 4 espèces.

2. Algorithme Expectation-Maximization

On va utiliser cet algorithme pour déterminer les maximums des vraisemblances. Il prend en considération les distances entre tous les points pour classifier les 4 espèces.

On l'initialise avec les centroïdes calculés avec la méthode de k-moyenne.

Afin d'obtenir un résultat meilleur on centre et on normalise chaque dimension.

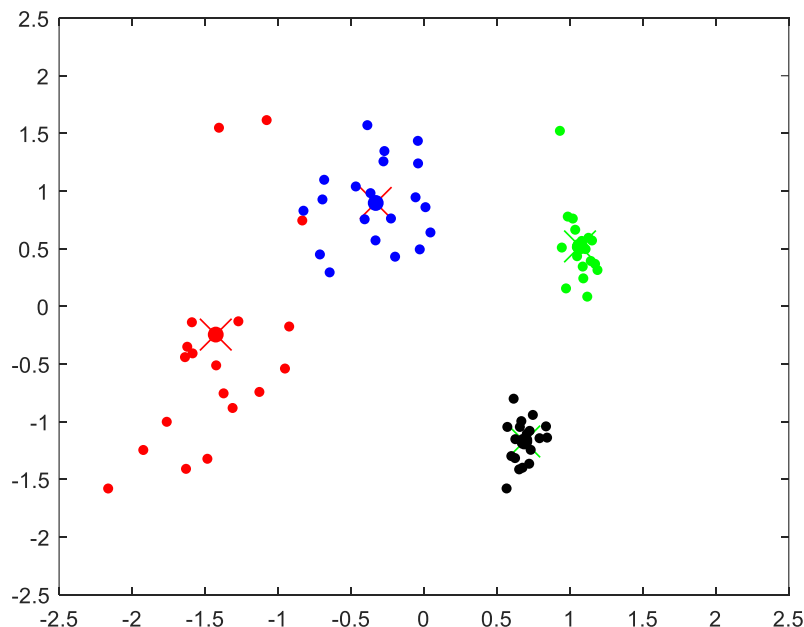
```

1  function Xns= centrer(Xns)
2  -   n=size(Xns(1,:),2);
3  -   for i=1:n
4      Xns(:,i)=(Xns(:,i)-mean(Xns(:,i)))/std(Xns(:,i));
5  -   end
6  -   end

208 % EM
209 Xn=[Xp;Xtestp];
210 N=size(Xn,1);
211 D=size(Xn,2);
212 K=4;
213 % Center and normalize each dimension
214 Xn=centrer(Xn);
215 % Initialisation des moyennes et covariances des classes
216 mu=C;
217 Sigma(1,:)=eye(D);
218 Sigma(2,:)=eye(D);
219 Sigma(3,:)=eye(D);
220 Sigma(4,:)=eye(D);
221 pi=ones(1,K)/K;
222 apost=zeros(K,N);
223 figure, scatter(Xn(:,1),Xn(:,2),20,'b','fill');
224 gr=[1,2,3,4];
225 hold;
226 h1=gscatter(mu(:,1),mu(:,2),gr,'rgbk',10,[],'off');
227 set(h1,'MarkerSize',20);
228 h1=scatter(mu(1,1),mu(1,2),50,'r','0','filled');
229 h2=scatter(mu(2,1),mu(2,2),50,'g','0','filled');
230 h3=scatter(mu(3,1),mu(3,2),50,'b','0','filled');
231 h4=scatter(mu(4,1),mu(4,2),50,'k','0','filled');
232 axis([-2.5 2.5 -2.5 2.5]);
233 % code de mise en oeuvre de l'EM (limité à 50 itérations)
234 for i=1:50
235     pause(0.2)
236     % Etape E
237     for k=1:K
238         muk=mu(k,:);
239         Sigmak(:,:)=Sigma(k,:,:);
240         apost(k,:)=mvnpdf(Xn,muk,Sigmak)*pi(k);
241     end
242     apost=apost./repmat(sum(apost),K,1);
243     hold on;
244     color=apost(1:3,:);
245     scatter(Xn(:,1),Xn(:,2),20,color,'fill');
246     axis([-2.5 2.5 -2.5 2.5]);
247     pause(0.2)
248     % Etape M
249     for k=1:K
250         mu(k,:)=sum(repmat(apost(k,:),1,D).*Xn)./repmat(sum(apost(k,:),1,D),1,D);
251         Sigma(k,:,:)=(repmat(apost(k,:),1,D).*(Xn-repmat(mu(k,:),N,1)).*(Xn-repmat(mu(k,:),N,1))./(repmat(sum(apost(k,:),D,D));
252         pi(k,:)=sum(apost(k,:))/N;
253     end
254     % Décision du MAP pour affichage
255     [~,IDX1]=max(apost);
256     hold off;
257     h1=gscatter(mu(:,1),mu(:,2),gr,'rg',[],[],'off');
258     hold on;
259     h1=scatter(mu(1,1),mu(1,2),50,'r','0','filled');
260     h2=scatter(mu(2,1),mu(2,2),50,'g','0','filled');
261     h3=scatter(mu(3,1),mu(3,2),50,'b','0','filled');
262     h4=scatter(mu(4,1),mu(4,2),50,'k','0','filled');
263     color=apost(1:3,:);
264     scatter(Xn(:,1),Xn(:,2),20,color,'fill');
265     axis([-2.5 2.5 -2.5 2.5]);
266 end

```

Figure :



➔ L'algorithme converge vers un maximum local lorsqu'on utilise les résultats de la méthode K-moyenne et on peut donc identifier les 4 types de feuilles.

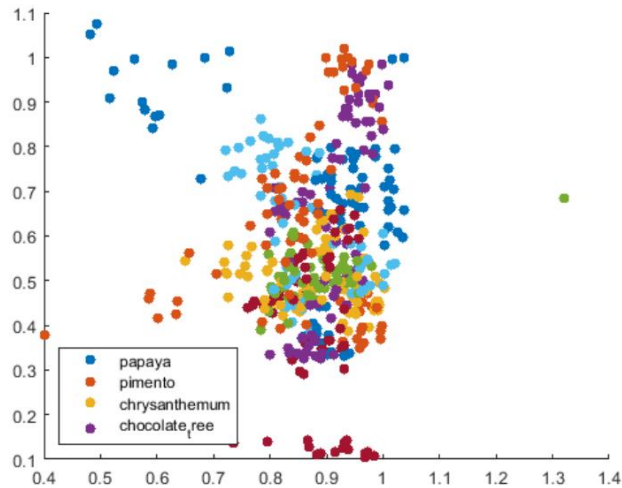
➤ *Extension du classifieur*

Dans la première partie du TP, on a utilisé des différentes méthodes pour classer 4 différents types de feuilles. On augmente maintenant le nombre de classes à 32.

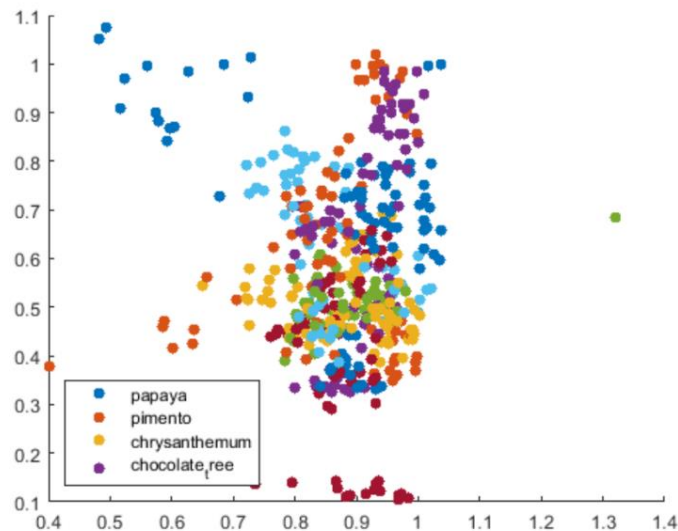
Sélection manuelle :

En tenant compte, à chaque fois, de différentes caractéristiques, on obtient :

➔ Avec feat = [1 2 3]

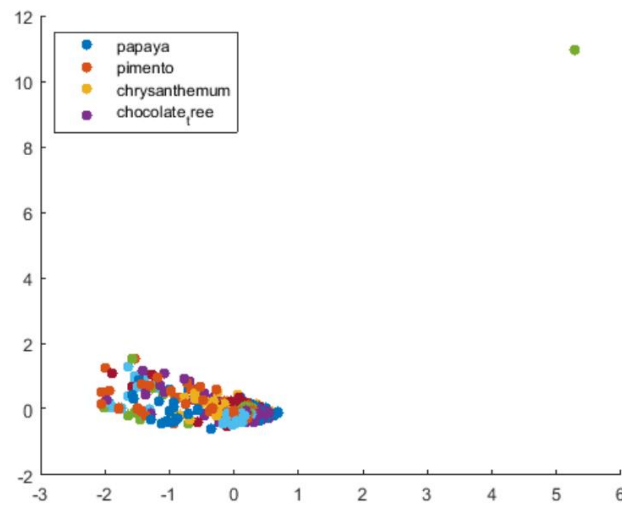


➔ Avec feat = [1 2 20 28]

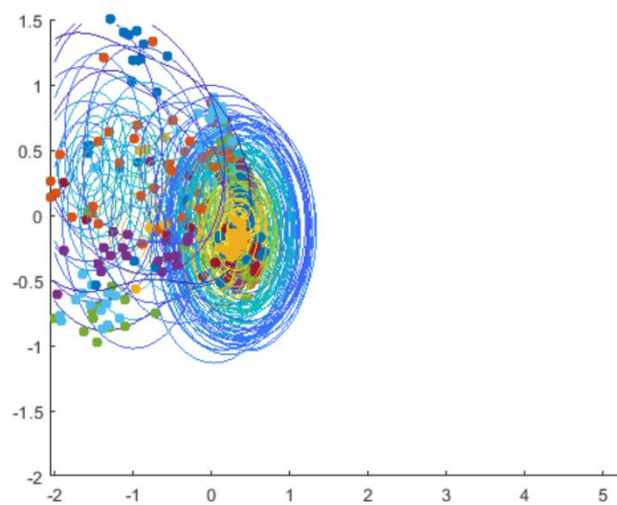


Contrairement au cas avec 4 types de feuilles, il est difficile de classer les 32 espèces. Il faut tenir compte de beaucoup plus de caractéristiques.

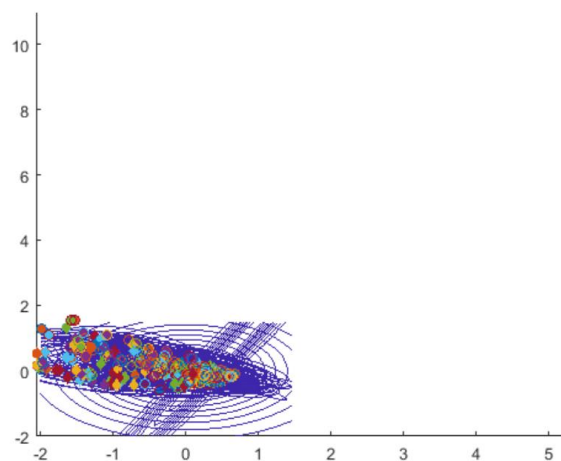
ACP : Avec l'analyse en composante principale, on obtient :



Approches supervisées : Méthode non paramétrique :

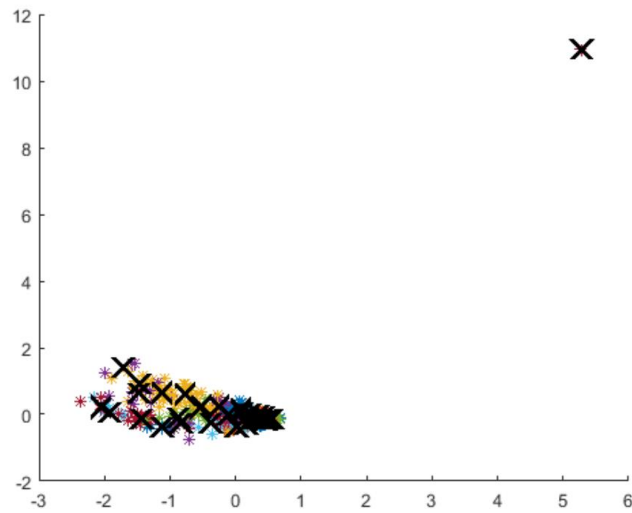


Approches supervisées : Méthode paramétrique :

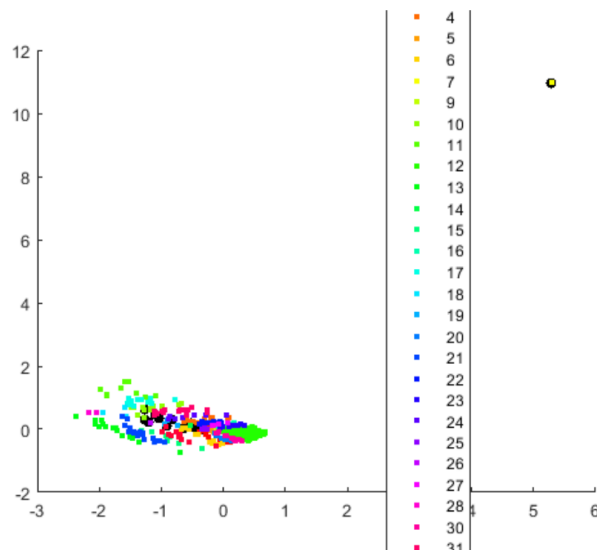


Le nombre d'erreurs dans ce cas est 313.

Approches supervisées : Kmeans :



Approches supervisées : EM :



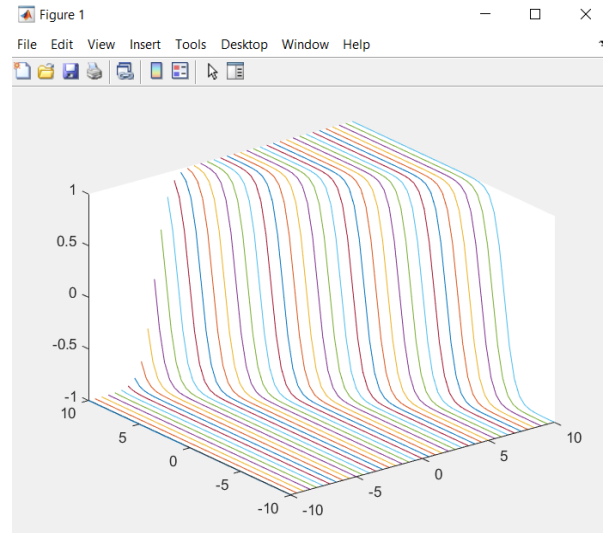
Pour un nombre élevé de classe, il est difficile de séparer les données et déterminer le type de chaque feuille, il vaut mieux donc les classer le via un réseau de neurone.

III. TD Réseaux de neurones artificiels

➤ Compréhension d'un perceptron :

Code Matlab commenté et affichage :

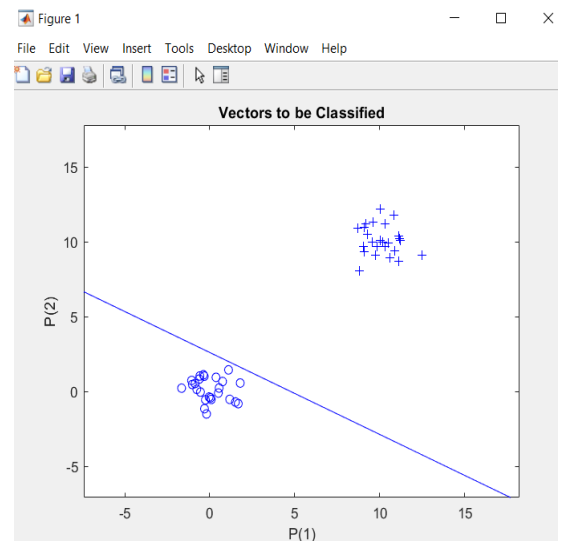
```
1 close all
2 clear all
3 clc
4
5 %1. Créer le vecteur des poids nécessaires.
6 w1=1;
7 w2=1;
8 poids=[w1;w2];
9 %2. Ajouter un biais.
10 b=-5;
11 %3. Choisir la fonction à tester.
12 f="tansig";
13 %observations
14 d=[3;4];
15 %4. Créer et former un perceptron
16 somme=d'*poids+b;
17 %5. Évaluer à la fin la fonction de sortie de ce perceptron. (feval)
18 sortie=feval(f,somme);
19
20 %1. Créer une grille sur un carré entre -10et 10 par exemple. (meshgrid)
21 [abs,ord]=meshgrid(-10:0.5:10);
22 %2. Évaluer la totalité des points sur ce perceptron.
23 X=[abs(:) ord(:)];
24 sortiegrille=feval(f,X*poids+b);
25 sortiegrille=reshape(sortiegrille,length(abs),length(ord));
26 %3. Tracer la sortie en 3D. (plot3)
27 figure;
28 plot3(abs,ord,sortiegrille)
```



➤ Implémentation d'un perceptron et la mise à jour des poids :

Code Matlab commenté et affichage :

```
4 %1. Définir les données d'entrée et de sortie en utilisant un certain nombre d'échantillons pour chaque
5 %classe (25 par exemple).
6 N=25;
7 %2. Créer deux ensembles aléatoires de données d'entrées avec un décalage entre eux. (randn)
8 offset=10;
9 X=[randn(N,2);randn(N,2)+offset];
10 %3. La première classe a le label 1, tandis que la seconde a le label 0.
11 C=[zeros(N,1); ones(N,1)];
12 %4. Visualiser les échantillons d'entrée/cible. (plotpv)
13 figure;
14 plotpv(X',C')
15 %5. Implémenter l'algorithme de mise à jour des poids pour créer un perceptron (avec un biais).
16 % algorithme du gradient descendant
17 %initialisation du poids
18 W=randn(2+1,1)% +1 pour le biais
19 pas=0.1
20 for j=1:5
21 for i=1:size(X,1)
22 erreur=10;
23 while(erreur> 0.001)
24 O = [X(i,:) 1]*W
25 if(O>0)
26 O=1;
27 else
28 O=0;
29 end
30 erreur=C(i)-O
31 W=W+2*pas*erreur*[X(i,:) 1]';
32 end
33 end
34 %6. Tracer la limite de decision (plotpc)
35 plotpc(W(1:2)',W(3))
36
```



➤ Classification des données linéairement séparables à l'aide d'un perceptron:

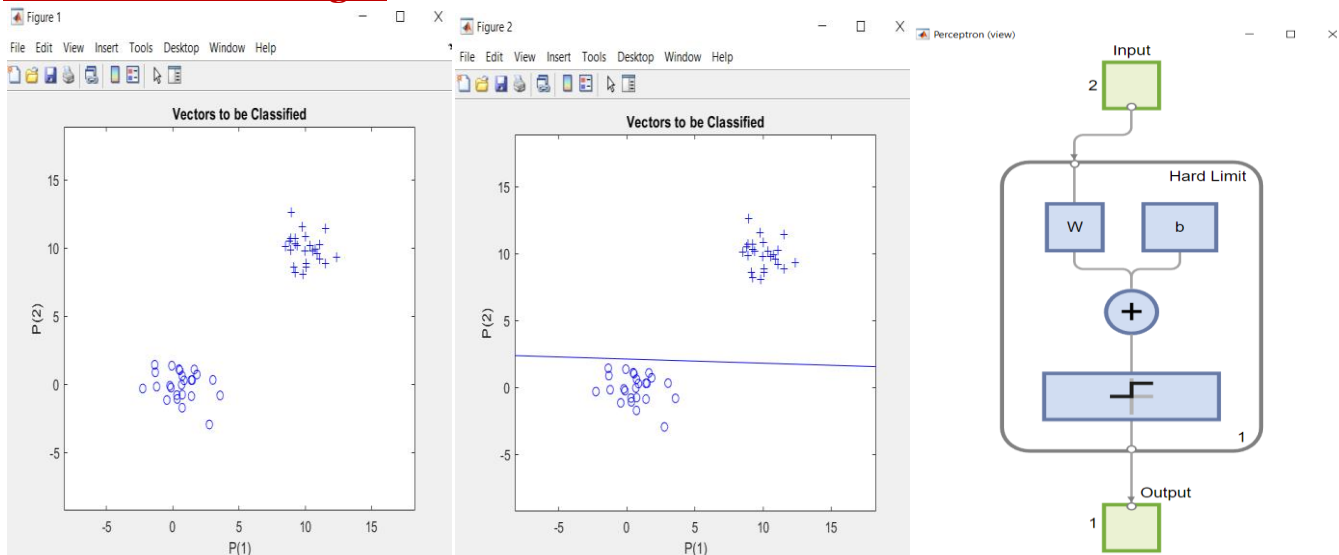
Code Matlab commenté:

```

1 close all
2 clear all
3 clc
4 %1. Définir les données d'entrée et de sortie en utilisant un certain nombre d'échantillons pour chaque
5 %classe (25 par exemple).
6 N=25;
7 %2. Créer deux ensembles aléatoires de données d'entrées avec un décalage entre eux. (randn)
8 offset=10;
9 X=[randn(N,2);randn(N,2)+offset];
10 %3. La première classe a le label 1, tandis que la seconde a le label 0.
11 C=[zeros(N,1); ones(N,1)];
12 %4. Visualiser les échantillons d'entrée/cible. (plotpv)
13 figure;
14 plotpv(X',C')
15 %5. Créer et former un perceptron (perceptron, train, configure, view)
16 net=perceptron;
17 net=train(net,X',C');
18 view(net);
19 figure;
20 plotpv(X',C')
21 %6. Tracer la limite de decision (plotpc)
22 plotpc(net.iw{1,1},net.b{1})
23 %7. Tester avec xtest = [0.7; 1.2]; et visualiser la solution
24 xtest=[0.7;1.2];
25 ytest=net(xtest);

```

Exécution et affichage :



➤ *Classification d'un problème à 4 classes avec un perceptron:*

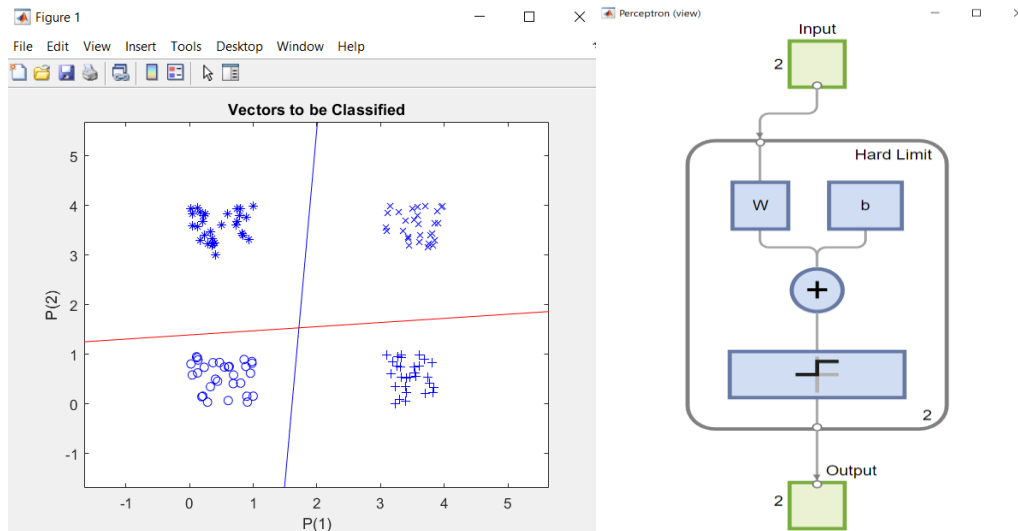
Code Matlab commenté:

```

1 close all
2 clear all
3 clc
4 %1. Définir les données d'entrée et de sortie en utilisant un certain nombre d'échantillons de chaque
5 %catégorie (30 par exemple).
6 N=30;
7 %2. Créer quatre ensembles aléatoires de données d'entrée avec différents décalages entre eux (rand)
8 offset=3;
9 A=[rand(N,2)];
10 B=[rand(N,1)+offset rand(N,1)];
11 C=[rand(N,1) rand(N,1)+offset];
12 D=[rand(N,1)+offset rand(N,1)+offset];
13 X=[A;B;C;D];
14 %3. Les étiquettes prennent en considération la position des données pour chaque classe. Définir 4
15 %étiquettes sur un espace 2-D.
16 a=[0;0];
17 b=[1;0];
18 c=[0;1];
19 d=[1;1];
20 %4. Visualiser les échantillons d'entrée/cible (plotpv). Pour les cibles, veiller à avoir la bonne
21 %dimension de votre matrice cible (repmat peut être utile).
22 L=[repmat(a,1,N) repmat(b,1,N) repmat(c,1,N) repmat(d,1,N)]';
23 %5. Créer un perceptron (perceptron)
24 net=perceptron;
25 net=train(net,X',L');
26 %7. Visualiser la structure (view)
27 view(net);
28 %8. Tester avec xtest = [0.7; 1.2]; et visualiser la solution
29 xtest=[0.7;-1];
30 ytest=net(xtest);
31 figure;
32 plotpv(X',L')
33 plotpc(net.iw{1,1},net.b{1})

```


Exécution et affichage :



➤ *Classification d'un problème à 4 classes avec un perceptron multicouche:*

Code Matlab commenté:

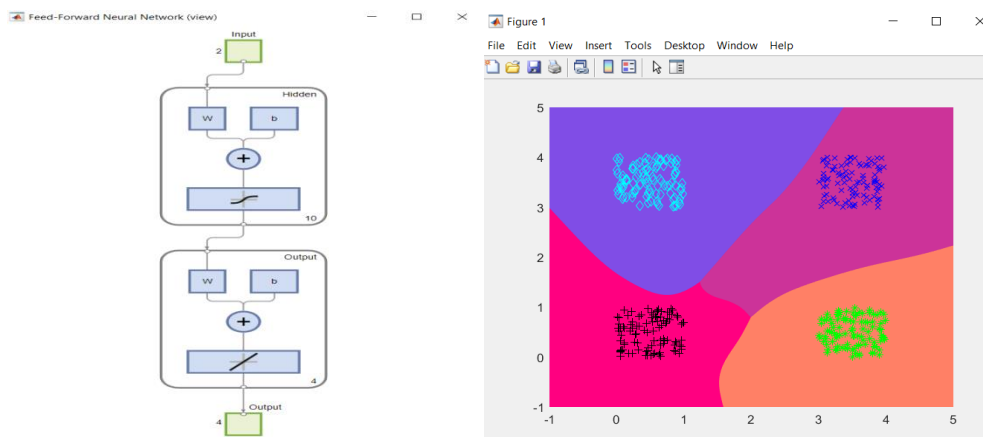
```
1 close all
2 clear all
3 clc
4 %1. Définir 4 ensembles de données d'entrée en utilisant un certain nombre d'échantillons de chaque
5 %classe (100 par exemple) (rand). Visualiser les données (plot, text)
6 N=100;
7 offset=3;
8 A=[rand(N,2)];
9 B=[rand(N,1)+offset rand(N,1)];
10 C=[rand(N,1) rand(N,1)+offset];
11 D=[rand(N,1)+offset rand(N,1)+offset];
12 X=[A;B;C;D];
13 %2. Définir le codage de sortie pour les 4 ensembles. (codage +1/-1)
14 a=[1;-1;-1;-1];
15 b=[-1;1;-1;-1];
16 c=[-1;-1;1;-1];
17 d=[-1;-1;-1;1];
18 %3. Préparer les entrées et les sorties pour l'apprentissage du réseau (repmat pour les cibles).
19 L=[repmat(a,1,N) repmat(b,1,N) repmat(c,1,N) repmat(d,1,N)];
20 %4. Créer un perceptron multicouche (feedforwardnet).
21 net=feedforwardnet(10,'traingd');
22 %5. Entraîner le réseau de neurones (train).
23 net.divideParam.trainRatio=0.7;
24 net.divideParam.valRatio=0.2;
25 net.divideParam.testRatio=0.1;
26 net=train(net,X',L');
27 %6. Visualiser la structure (view)
28 view(net)
29 %7. Evaluer la performance du réseau et tracer les résultats en comparant la classe prédite à la classe
30 %cible
31 xtest=[0.7; 1.2];
32 ytest=net(xtest);
33 figure(1)
```

```

34 hold on
35 plot(A(:,1),A(:,2),'k+')
36 plot(B(:,1),B(:,2),'g*')
37 plot(C(:,1),C(:,2),'cd')
38 plot(D(:,1),D(:,2),'bx')
39 %évaluation sur une grille
40 [abs,ord]=meshgrid(-1:0.01:5);
41 Xd=[abs(:) ord(:)];
42 sortie=net(Xd');
43 %8. Tracer le résultat de la classification pour l'espace d'entrée complet en créant une grille (meshgrid,
44 %mesh, reshape)
45 figure(1)
46 m=mesh(abs,ord,reshape(sortie(1,:),length(abs),length(abs))-5);
47 set(m,'facecolor',[1 0 0.5],'linestyle','none')
48 hold on
49 m=mesh(abs,ord,reshape(sortie(2,:),length(abs),length(abs))-5);
50 set(m,'facecolor',[1 0.5 0.4],'linestyle','none')
51 m=mesh(abs,ord,reshape(sortie(3,:),length(abs),length(abs))-5);
52 set(m,'facecolor',[0.5 0.3 0.9],'linestyle','none')
53 m=mesh(abs,ord,reshape(sortie(4,:),length(abs),length(abs))-5);
54 set(m,'facecolor',[0.8 0.2 0.6],'linestyle','none')
55 view()

```

Exécution et affichage :



- *Classification à l'aide d'un simple réseau d'apprentissage profond pour les chiffres manuscrits:*

Code Matlab commenté:

```

1 close all
2 clear all
3 clc
4 %6.1. Charger et examiner les données
5 digitDatasetPath=fullfile(matlabroot,'toolbox\nnet\ndemos\ndatasets\DigitDataset');
6 imds=imageDatastore(digitDatasetPath,'IncludeSubfolders',true,'LabelSource','foldernames');
7 figure;
8 imshow(imds.Files{4822})
9 labelCount=countEachLabel(imds)
10 img=readimage(imds,500);
11 size(img)
12 %6.2. Préciser les ensembles d'apprentissage et de validation
13 [trainDigitData,valDigitData]=splitEachLabel(imds,750,'randomized')
14 %6.3. Définir l'architecture du réseau
15 layers = [
16 imageInputLayer([28 28 1])
17 convolution2dLayer(3,8,'Padding','same')
18 batchNormalizationLayer
19 reluLayer
20 maxPooling2dLayer(2,'Stride',2)
21 fullyConnectedLayer(10)
22 softmaxLayer
23 classificationLayer];
24 %6.4. Préciser les options d'apprentissage
25 options = trainingOptions('sgdm', ...
26 'InitialLearnRate',0.01, ...
27 'MaxEpochs',4, ...
28 'Shuffle','every-epoch', ...
29 'ValidationData',valDigitData, ...
30 'ValidationFrequency',30, ...
31 'Verbose',false, ...
32 'Plots','training-progress');
33 %6.5. Apprendre le réseau avec les données d'apprentissage

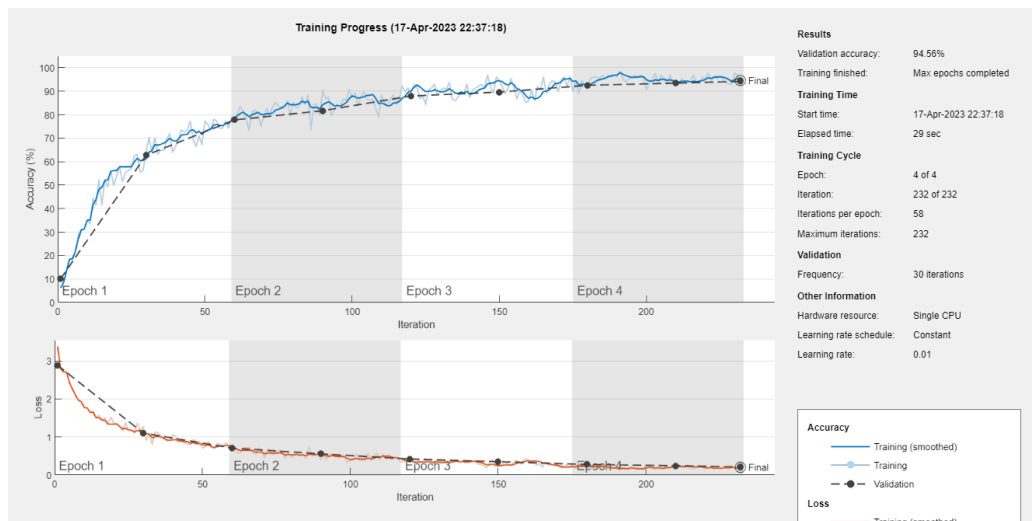
```

```

34 net=trainNetwork(trainDigitData, layers, options);
35 %6.6. Classer les images de validation images et calculer la précision
36 Ypred=classify(net, valDigitData);
37 Yvalidation=valDigitData.Labels;
38 accuracy=sum(Ypred==Yvalidation)/length(Ypred)

```

Exécution et affichage :



IV. Classification en utilisant un réseau de neurones convolutif

➔ L'objectif dans cette partie est de manipuler des réseaux de neurones convolutifs qui sont adaptés à la reconnaissance des images pour la classification des feuilles d'arbre suivant l'arbre duquel elles proviennent. Dans un premier temps, on va se limiter à 4 types de feuilles à classer (bitter orange, rose, sweet potato et vieux garçon) puis on va passer à 32 types de feuilles. Pour pouvoir créer et entraîner un réseau de neurones, il faut une base de données assez riche pour que la classification soit de bonne qualité. Notre base de photos de feuilles contient, pour chaque classe, entre 148 et 160 images.

➤ Cas de 4 Classes de feuilles :

Chargement des données

En premier lieu, on commence par télécharger la base de données des images dont on va se servir pour la phase d'apprentissage et la phase de test, ensuite on va charger les données numériques comme un entrepôt de données d'images avec la fonction (ImageDatastore) qui étiquette automatiquement les images en fonction des noms de dossiers et stocke les données comme un objet ImageDatastore.

Code Matlab :

```

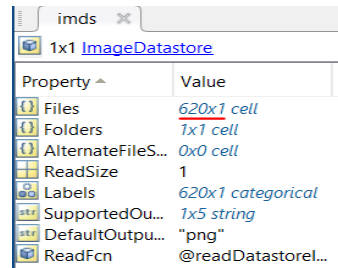
5 %Chemin d'accès et Chargement du données
6 digitDatasetPath='C:\Users\LENOVO\OneDrive\Bureau\Leaf';
7 imds=imageDatastore(digitDatasetPath, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');

```

Exécution :

Workspace	
Name ^	Value
digitDatasetP...	'C:\Users\LENOV...
imds	1x1 ImageDatast...

Affichage des données



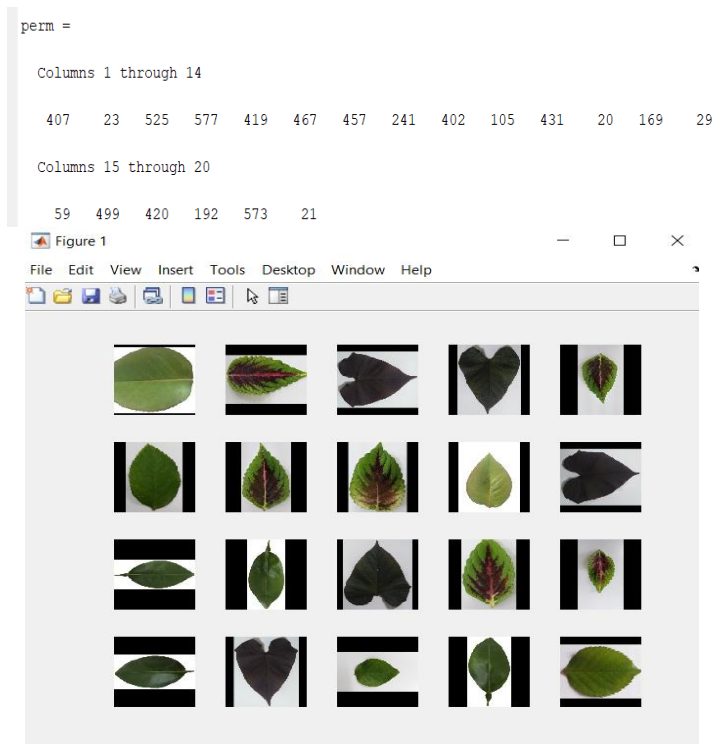
Property	Value
Files	620x1 cell
Folders	1x1 cell
AlternateFileS...	0x0 cell
ReadSize	1
Labels	620x1 categorical
SupportedOu...	1x5 string
DefaultOutpu...	"png"
ReadFcn	@readDatastorel...

On voit bien qu'il existe au total 620 images, donc afin de vérifier que la base de données a été bien chargée, on affiche 20 images choisies aléatoirement à partir des 620 images avec la fonction (randperm).

Code Matlab :

```
9 %Affichage des données
10 perm=randperm(620,20) % générer d'une manière aléatoire 20 entiers entre 1 et 620
11 for i=1:20
12     subplot(4,5,i)
13     imshow(imds.Files{perm(i)})
14     hold on
15 end
```

Exécution et affichage:



Nombre des images dans chaque catégories et taille des images

Pour calculer le nombre des images dans chaque catégorie on utilise la fonction (countEachLabel).

Pour déterminer la taille des images on utilise la fonction (size). La taille des images sert comme donné pour la couche d'entrée de notre réseau convolutif.

Code Matlab :

```
17 %Nombre des images pour chaque catégories
18 labelCount=countEachLabel(imds)
19 %Taille des images
20 img=readimage(imds,500);
21 size(img)
```

Exécution :

```
labelCount =
4x2 table
      Label      Count
      ----      -
bitter orange    160
      rose        148
sweet potato     152
vieux garcon     160

ans =
175  175   3
```

Division des données

On divise les données en un ensemble de données d'apprentissage et un ensemble de données validation de sorte que l'ensemble d'apprentissage contient 75% des données.

Code Matlab :

```
23 %Division des données
24 [trainDigitData,valDigitData]=splitEachLabel(imds,0.75,0.25,'randomized');
```

Définition de l'architecture du réseau

Maintenant nous allons définir l'architecture de notre réseau convolutif, pour cela nous allons définir les couches qui vont servir à entraîner notre modèle pour la classification des feuilles d'arbre :

imageInputLayer

C'est la couche d'entrée du réseau dans laquelle on va préciser la taille de des images d'entrées.

```
imageInputLayer([175 175 3])
```

convolution2dLayer

C'est la couche dans laquelle on va préciser la taille et le nombre de filtres sur les images On peut aussi spécifier si on souhaite ajouter du padding ou pas et ainsi que si on souhaite que la taille des images de sortie reste la même ou pas.

```
convolution2dLayer(3,10,'Padding','same')
```

taille des filtres

nombre des filtres

batchNormalizationLayer

C'est la couche qui normalise les activations et les gradients qui se propagent dans un réseau, ce qui facilite le problème d'optimisation pour apprendre les réseaux

`batchNormalizationLayer`

ReLU Layer

C'est la fonction d'activation qui permet d'éliminer les nombres négatifs des pixels.

`reluLayer`

Max Pooling Layer

C'est la couche qui permet de renvoyer les valeurs maximales des régions rectangulaires d'entrées

`maxPooling2dLayer(2, 'Stride', 2)`

Dans notre cas :

→ On choisit de répéter le processus suivant 4 fois :

`convolution2dLayer`

`batchNormalizationLayer`

`ReLU Layer`

→ On augmente à chaque itération le nombre de filtre de la manière suivante : 10, 18, 26, 34 et on prend des filtres de taille 3×3 .

→ Ce processus est toujours suivi de la couche Max Pooling Layer sauf pour la dernière itération.

Finalement on applique les couches :

Fully Connected Layer

C'est la couche dans laquelle les neurones se connectent à tous les neurones de la couche précédente. Elle combine toutes les caractéristiques apprises sur l'image qui permet de classer les données selon le nombre classes spécifié en paramètre qui est 5 dans notre cas.

`fullyConnectedLayer(4)`

Softmax Layer

C'est la couche qui permet de normaliser la sortie de la couche entièrement connectée. Elle est constituée des probabilités de classification.

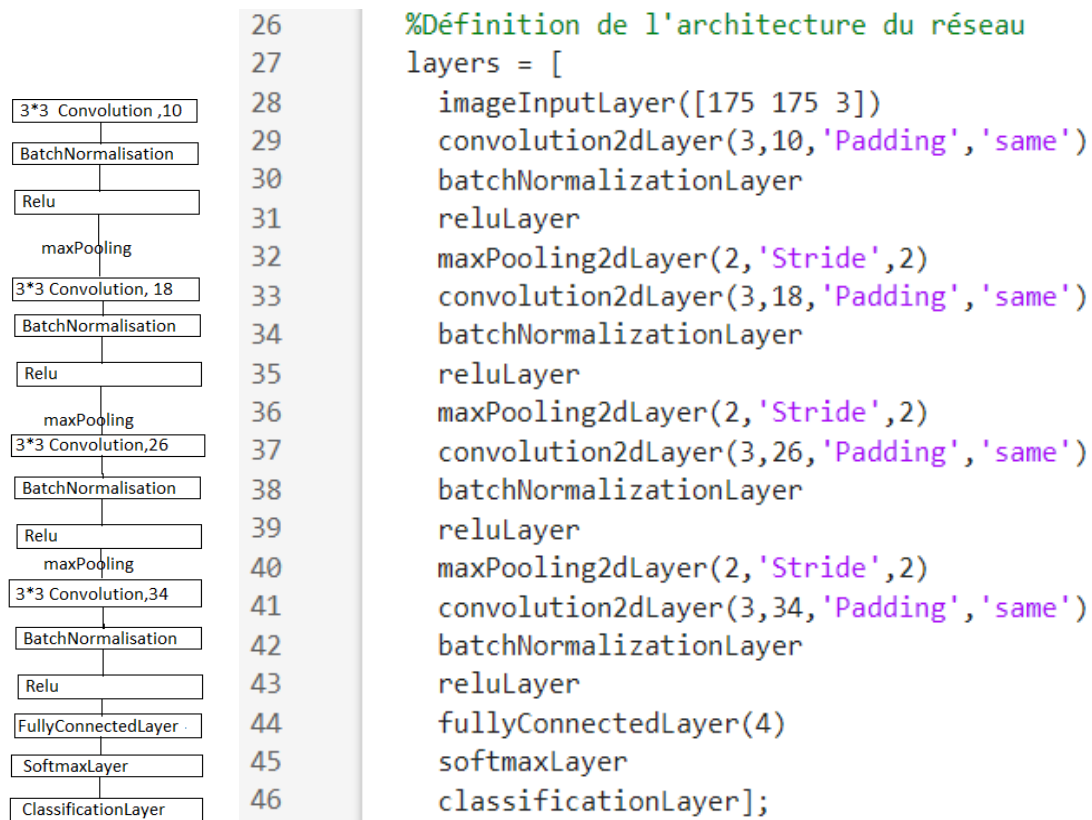
`softmaxLayer`

Classification Layer

Cette couche utilise les probabilités renvoyées par la fonction d'activation softmax pour chaque entrée afin d'affecter l'entrée à l'une des classes mutuellement exclusives et de calculer la perte.

`classificationLayer];`

Architecture final et code Matlab :



Apprentissage du réseau

Dans cette partie on doit indiquer dans les options d'apprentissage le nombre d'époques qu'on choisit 4 pour notre cas et pour apprendre le réseau on utilise la fonction (trainNetwork).

Code Matlab :

```

49 %Option d'apprentissage(Nombre d'époques)
50 options = trainingOptions('sgdm', ...
51     'InitialLearnRate',0.01, ...
52     'MaxEpochs',5, ...
53     'Shuffle','every-epoch', ...
54     'ValidationData',valDigitData, ...
55     'ValidationFrequency',30, ...
56     'Verbose',false, ...
57     'Plots','training-progress');
58
59 %Apprentissage du réseau
60 net=trainNetwork(trainDigitData,layers,options);

```

Exécution et affichage:



Performance du réseau

Après avoir entraîné le modèle on procède à la phase de Test sur les données de validation et on calcul la précision de la classification comme suit :

Code Matlab :

```
62 %Performance du réseau
63 Ypred=classify(net,valDigitData);
64 Yvalidation=valDigitData.Labels;
65 accuracy=sum(Ypred==Yvalidation)/length(Ypred)
```

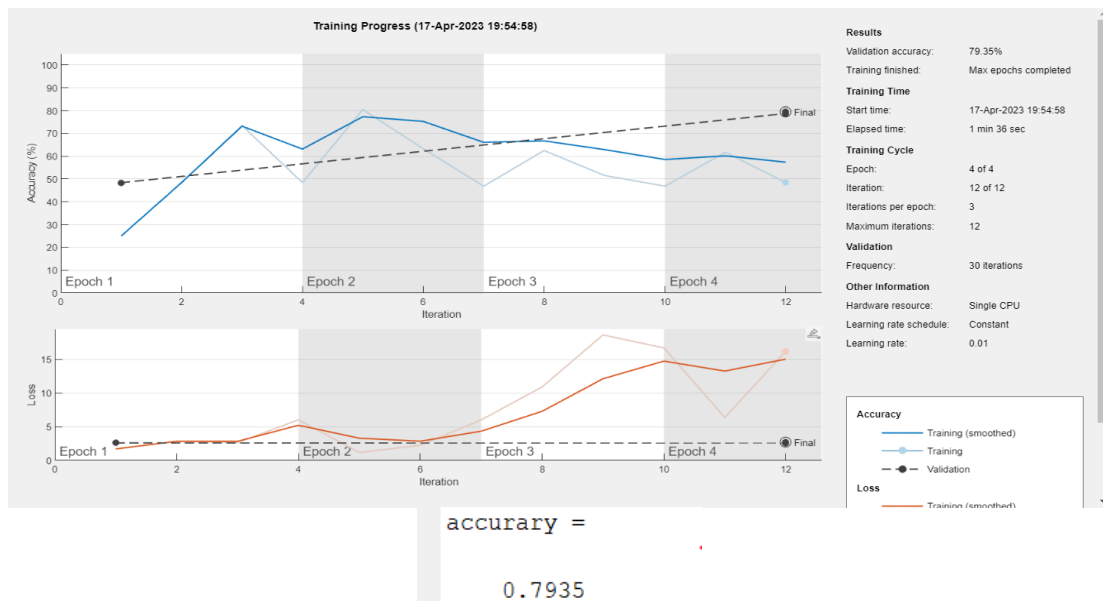
Exécution :

```
accuracy =
0.9742
```

Après la simulation, les résultats montrent qu'on a obtenu une précision qui vaut 97.42% dans une 1 min 31 secondes d'apprentissage. Un résultat rapide avec un taux élevé de précision ce qui nous permet de dire que notre réseau est performant.

Explication du choix de l'architecture :

Le choix était au début avec 3 blocs (convolution2dLayer ,batchNormalizationLayer ,ReLU Layer) et 4 époques pour l'apprentissage du réseau . Cette architecture a donné une précision de 79.35%.Pour l'améliorer on ajouté un bloc (convolution2dLayer,batchNormalizationLayer ,ReLU Layer) et une époque pour l'apprentissage du réseau ce qui a fait que la précision a passé de 79.35% à 97.42%.



accuracy =
0.7935

➤ Cas de 32 Classes de feuilles :

Chargement des données

Code Matlab :

```

5 %Chemin d'accès et Chargement du données
6 digitDatasetPath='C:\Users\LENOVO\OneDrive\Bureau\Imagors';
7 imds=imageDatastore(digitDatasetPath,'IncludeSubfolders',true,'LabelSource','foldernames');

```

Exécution :

digitDatasetP... 'C:\Users\LENOV...
imds 1x1 ImageDatast...

Affichage des données

imds	
1x1 ImageDatastore	
Property	Value
Files	4955x1 cell
Folders	1x1 cell
AlternateFileS...	0x0 cell
ReadSize	1
Labels	4955x1 categorical
SupportedOu...	1x5 string
DefaultOutput...	"png"
ReadFcn	@readDatastore...

On voit bien qu'il existe au total 4955 images, donc afin de vérifier que la base de données a été bien chargée, on affiche 20 images choisies aléatoirement à partir des 4955 images avec la fonction (randperm).

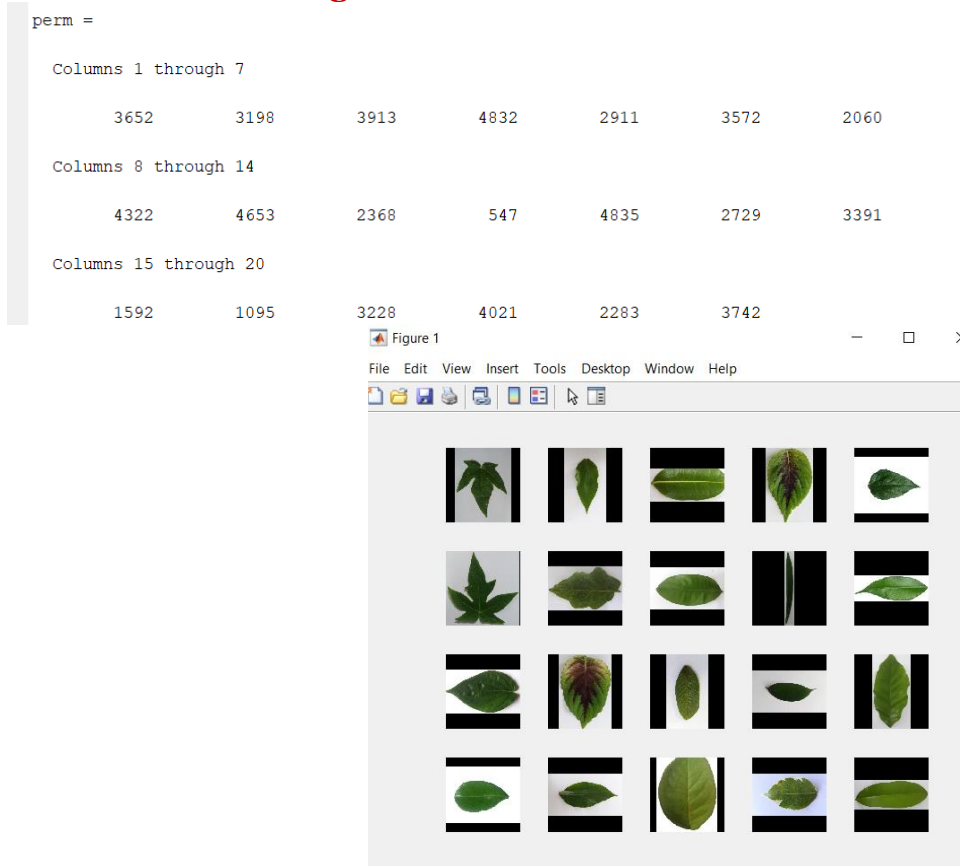
Code Matlab :

```

9      %Affichage des données
10     perm=randperm(4955,20); % generer d'une manière aléatoire 20 entiers entre 1 et 4955
11     for i=1:20
12         subplot(4,5,i)
13         imshow(imds.Files{perm(i)})
14         hold on
15     end

```

Exécution et affichage:



Nombre des images dans chaque catégories et taille des images

Code Matlab :

```

17 %Nombre des images pour chaque catégories
18 labelCount=countEachLabel(imds);
19 %Taille des images
20 img=readimage(imds,500);
21 size(img);

```

Exécution :

```

labelCount =

32x2 table

    Label    Count
    _____
    ashanti blood    160
    barbados cherry  160
    beaumier du perou 148
    betel            160
    bitter orange    160
    :                :
    star apple      160
    sweet olive      160
    sweet potato     152
    thevetia         160
    vieux garcon     160

Display all 32 rows.

ans =

175    175     3

```

Division des données

Code Matlab :

```

23 %Division des données
24 [trainDigitData,valDigitData]=splitEachLabel(imds,0.75,0.25,'randomized');

```

Définition de l'architecture du réseau

On inspire de la même architecture du réseau précédent (cas de 4 types de feuilles).

Code Matlab :

```

27 %Définition de l'architecture du réseau
28 layers = [
29     imageInputLayer([175 175 3])
30     convolution2dLayer(3,10,'Padding','same')
31     batchNormalizationLayer
32     reluLayer
33     maxPooling2dLayer(2,'Stride',2)
34     convolution2dLayer(3,18,'Padding','same')
35     batchNormalizationLayer
36     reluLayer
37     maxPooling2dLayer(2,'Stride',2)
38     convolution2dLayer(3,26,'Padding','same')
39     batchNormalizationLayer
40     reluLayer
41     maxPooling2dLayer(2,'Stride',2)
42     convolution2dLayer(3,34,'Padding','same')
43     batchNormalizationLayer
44     reluLayer
45     fullyConnectedLayer(32)
46     softmaxLayer
47     classificationLayer];

```

Apprentissage du réseau

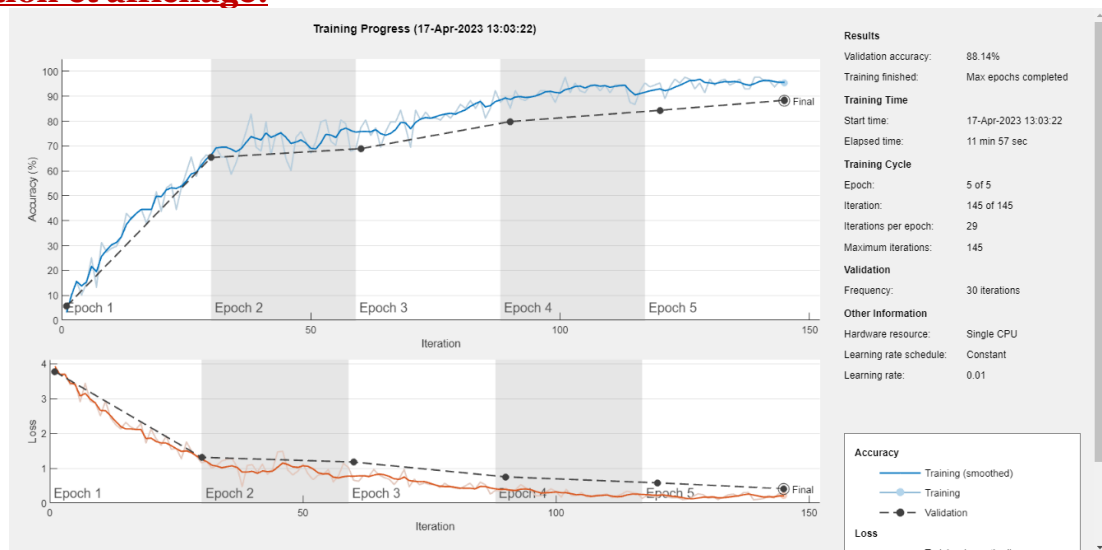
Code Matlab :

```

49 %Option d'apprentissage(Nombre d'époques)
50 options = trainingOptions('sgdm', ...
51     'InitialLearnRate',0.01, ...
52     'MaxEpochs',5, ...
53     'Shuffle','every-epoch', ...
54     'ValidationData',valDigitData, ...
55     'ValidationFrequency',30, ...
56     'Verbose',false, ...
57     'Plots','training-progress');
58 %Apprentissage du réseau
59 net=trainNetwork(trainDigitData,layers,options);

```

Exécution et affichage:



Performance du réseau

Code Matlab :

```
62 %Performance du réseau
63 Ypred=classify(net,valDigitData);
64 Yvalidation=valDigitData.Labels;
65 accuracy=sum(Ypred==Yvalidation)/length(Ypred)
```

Exécution :

```
accuracy =  
  
0.8814
```

Après la simulation, les résultats montrent qu'avec la même architecture du cas de 4 espèces, le réseau dans notre cas (de 32 espèces) est moins performant et plus lent mais on a obtenu un bon résultat.

On a obtenu une précision qui vaut 88.14% et le réseau a pris 11min 52 sec (résultat évident sachant le grand nombre de types d'espèces) pour l'apprentissage.

On peut encore améliorer ce réseau en ajoutant d'autres couches de convolution et en augmentant nos données.

V. Conclusion

Dans une première partie, on a appris les deux méthodes d'apprentissage appliqué dans la machine Learning. La première est l'apprentissage supervisé où les données sont étiquetées et la deuxième l'apprentissage non supervisé avec des données non étiquetées en utilisant les méthodes K-means et l'algorithme EM.

Dans une deuxième partie, on s'est focalisé sur le domaine de l'apprentissage profond en découvrant des notions tels que le perceptron, le réseau de neurone et le réseau de neurone convolutif.