Analyse par Composantes Principales

A- L'exploration des données de la data set breast_cancer

In[1]:
```
from sklearn.datasets import load_breast_cancer
```

➔ **On a importé seulement la data set breast_cancer du module sklearn.datasets (pour des raisons de mémoire et de complexité).**

In[2]:
```
breast = load_breast_cancer()
```

In[3]:
```
breast_data = breast.data
```

In[4]:
```
breast_data.shape
```

Out[4]: (569, 30)

➔ **Notre data set a 569 échantillons et 30 attributs.**

In[5]:
```
breast_labels = breast.target
breast_labels.shape
```

Out[5]: (569,)

In[6]:
```
import numpy as np
labels=np.reshape(breast_labels,(569,1))
```

➔ **On a importé la librairie numpy pour la manipulation des matrices et tableaux multidimensionnels.**

In [7]:
```
final_breast_data=np.concatenate([breast_data,labels],axis=1)
final_breast_data.shape
```

Out[7]: (569, 31)

➔ **On ajoute le label à notre data set. Elle contient maintenant 569 échantillons et 31 attributs.**

In[8]:
```
import pandas as pd
breast_dataset=pd.DataFrame(final_breast_data)
```

➔ **On a importé la bibliothèque pandas pour la manipulation et l'analyse des données.**

In [9]:
```
features = breast.feature_names
features
```

➔ **Affichage des attributs de notre dataframe.**

Out[9]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'meanconcavepoints','meansymmetry','meanfractaldimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavityerror',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

In[10]:
```
features_labels=np.append(features,'label')
```

In[11]:
```
breast_dataset.columns = features_labels
```

➔ **Ajout du labels aux colonnes de la dataset.**

In[12]:
```
breast_dataset.head()
```

Out[12]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	di
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 31 columns

➔ **La fonction head() permet d'afficher les 5 premières lignes de notre data frame.**

In[13]:

```
breast_dataset['label'].replace(0,'Benign',inplace=True)
breast_dataset['label'].replace(1,'Malignant',inplace=True)
```

➔ **Nous avons remplacé la valeur 0 par Benign et la valeur 1 par Malignant. L'option inplace=True precise que les modications se fronts sur la meme data frame sans en créer une autre.**

In[14]:

```
breast_dataset.tail()
```

Out[14]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587

5 rows × 31 columns

➔ **La fonction tail() permet d'afficher les 5 dernières lignes de notre data frame.**

B- Exploration des données de la data set cifar10

In[15]:
```
from keras.datasets import cifar10
```

➔ **Nous avons importé la data set cifar 10 du module keras.datasets.**

In[17]:
```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

➔ **Importation du module SSL (Secure Sockets Layer) pour créer une connexion sécurisée entre le client et le serveur.**

In[18]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-
python.tar.gz170500096/170498071 [==============================] - 99s1us/step
170508288/170498071 [==============================] - 99s1us/step
```

➔ **Division des données entre des données pour l'apprentissage automatique (x_train, y_train) et des données pour le test(x_test, y_test)**

X: attributs
Y: label

In [19]:
```
print('Traning data shape:', x_train.shape)
print('Testing data shape:',x_test.shape)
```

Out [19]:
```
Traning data shape: (50000, 32, 32,3)
Testingdatashape:(10000,32,32,3)
```

➔ **Voici comment lire la forme : (nombre d'échantillons, hauteur, largeur, canaux de couleur). Dans ce cas, nous avons 3 canaux de couleur qui représentent les valeurs RVB.**

In [20]:
```
y_train.shape,y_test.shape
```

Out[20]: ((50000, 1), (10000, 1))

➔ **Nous avons 50000 échantillons pour l'apprentissage et 10000 pour le test. Nous avons un seul label.**

In[21]:

```
# Find the unique numbers from the train labels
classes=np.unique(y_train) nClasses
= len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)
```

```
Total number ofoutputs:    10
Outputclasses:    [0123456789]
```

➔ **Nous avons 10 classes.**

In[24]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

➔ **Nous avons importé la bibliothèque matplotlib pour pouvoir tracer et visualiser des données sous formes de graphiques.**

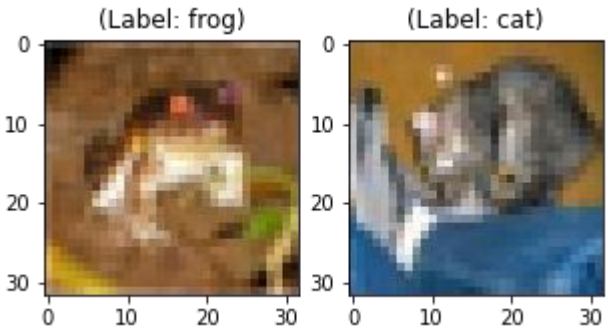
➔ **%matplotlib inline est une fonction qui rend la figure dans un Notebook. Avec ce backend, la sortie des commandes de traçage est affichée en ligne dans les frontends comme le Notebook Jupyter, directement sous la cellule de code qui l'a produite.**

In[25]:

```
label_dict = {
 0:'airplane',
 1:'automobile',
 2:'bird',
 3:'cat',
 4:'deer',
 5:'dog',
 6:'frog',
 7:'horse',
 8:'ship',
 9:'truck',
}
plt.figure(figsize=[5,5])

# Display the first image in training data
plt.subplot(121)
curr_img=np.reshape(x_train[0],(32,32,3))
plt.imshow(curr_img)
print(plt.title("(Label:"+str(label_dict[y_train[0][0]])+")"))

# Display the first image in testing data
plt.subplot(122)
curr_img=np.reshape(x_test[0],(32,32,3))
plt.imshow(curr_img)
print(plt.title("(Label:"+str(label_dict[y_test[0][0]])+")"))
```

➔ **Nous avons attribué à chaque label une valeur qualitative. Ensuite nous avons afficher les images à l'aide de la fonction subplot() de matplotlib. Sublot() prend trois paramètres. Par exemple dans notre cas nous avons plt.subplot(121). Le 1 represente le nombre de lignes que nous allons afficher dans la grille. Le 2 signifie que nous allons afficher deux colonnes. Et le 1 désigne la première image de la grille.**

Text(0.5, 1.0, '(Label: frog)')
Text(0.5, 1.0, '(Label: cat)')



-C: Analyse en Composantes Pricipales avec la dataset Breast cancer

In [26]:

```
from sklearn.preprocessing import StandardScaler
x = breast_dataset.loc[:, features].values
x = StandardScaler().fit_transform(x) # normalizing the features
x.shape
```

➔ **Pour appliquer la normalisation, nous avons importé le module StandardScaler à partir de la bibliothèque sklearn et sélectionné uniquement les fonctionnalités du breast_dataset que nous avons créées à l'étape d'exploration des données. Nous avons appliqué la mise à l'échelle en effectuant fit_transform sur les données. Lors de l'application de StandardScaler, toutes les données doivent être normalement distribuées.**

Out[26]: (569, 30)

In[27]:
```
np.mean(x),np.std(x)
```

Out[27]: (-6.826538293184326e-17, 1.0)

In[28]:
```
feat_cols=['feature'+str(i)foriinrange(x.shape[1])]
normalised_breast=pd.DataFrame(x,columns=feat_cols)
normalised_breast.tail()
```

➔ **Nous avons converti les données normalisées dans un format tabulaire à l'aide de DataFrame.**

Out[28]:

	feature0	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8
564	2.110995	0.721473	2.060786	2.343856	1.041842	0.219060	1.947285	2.320965	-0.312589
565	1.704854	2.085134	1.615931	1.723842	0.102458	-0.017833	0.693043	1.263669	-0.217664
566	0.702284	2.045574	0.672676	0.577953	-0.840484	-0.038680	0.046588	0.105777	-0.809117
567	1.838341	2.336457	1.982524	1.735218	1.525767	3.272144	3.296944	2.658866	2.137194
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085	-1.150752	-1.114873	-1.261820	-0.820070

5 rows × 30 columns

In[29]:
```
from sklearn.decomposition import PCA
pca_breast=PCA(n_components=2)
principalComponents_breast = pca_breast.fit_transform(x)
```

In[30]:
```
principal_breast_Df=pd.DataFrame(data=principalComponents_breast
                 , columns = ['principal component 1', 'principal component 2'])
principal_breast_Df.tail()
```

➔ **Nous avons utilisé la bibliothèque sklearn pour importer le module PCA, et dans la méthode PCA, nous avons passé le nombre de composants (n_components=2) et enfin nous avons appelé fit_transform sur les données agrégées.**

Out[30]:

	principal component 1	principal component 2
564	6.439315	-3.576817
565	3.793382	-3.584048
566	1.256179	-1.902297
567	10.374794	1.672010
568	-5.475243	-0.670637

In[31]:

```
print('Explainedvariationperprincipalcomponent:{}'.format(pca_breast.explained_
```

Explained variation per principal component: [0.44272026 0.18971182]

➔ La première composante contient 44% de l'information et la deuxième composante en contient 18%. 38% de l'information ont été perdus.

In [32]:

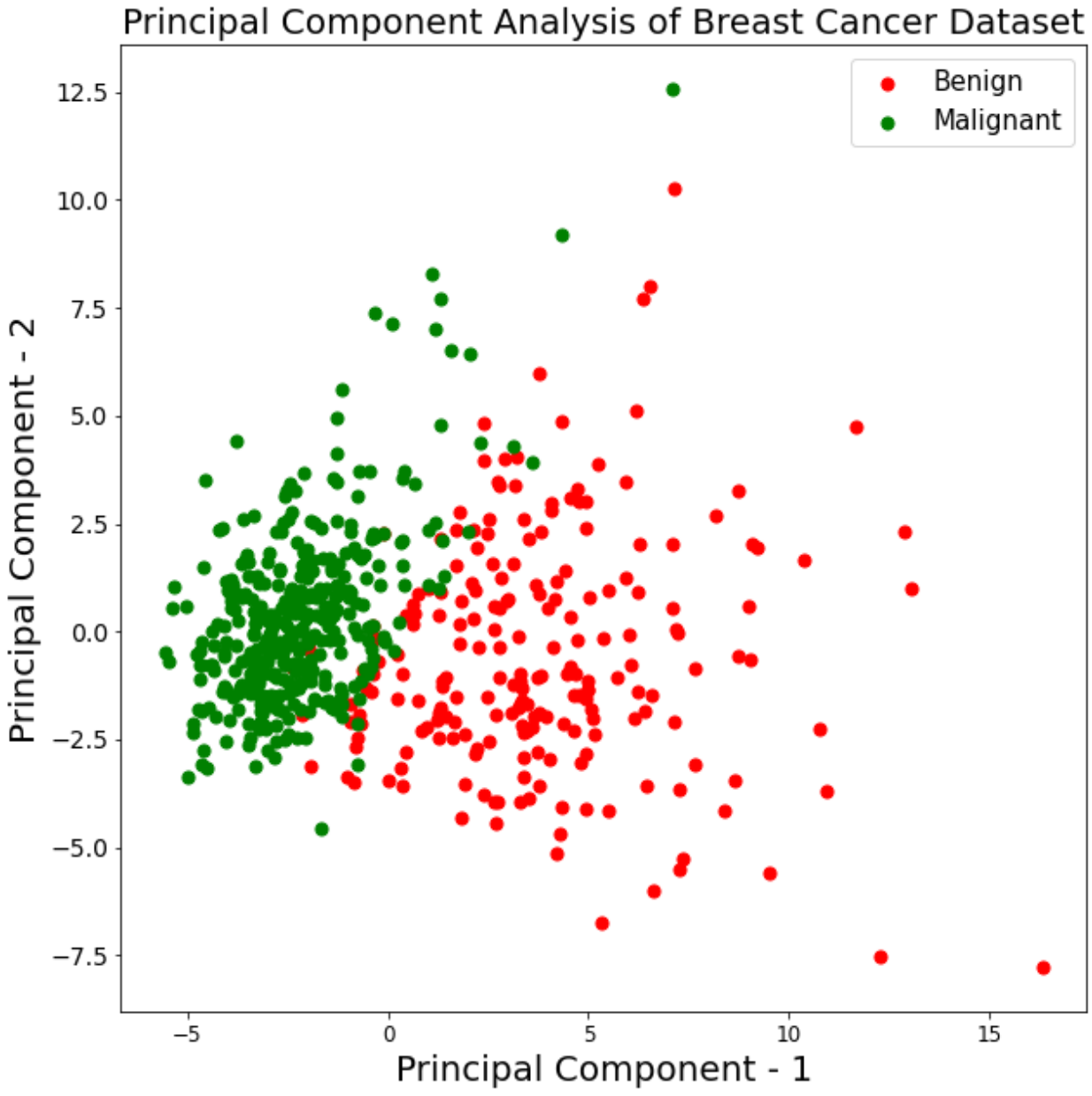
```
plt.figure()
plt.figure(figsize=(10,10))
plt.xticks(fontsize=12)
plt.yticks(fontsize=14)
plt.xlabel('PrincipalComponent-1',fontsize=20)
plt.ylabel('PrincipalComponent-2',fontsize=20)
plt.title("PrincipalComponentAnalysisofBreastCancerDataset",fontsize=20)
targets = ['Benign', 'Malignant']
colors = ['r','g']
```

```
for target, color inzip(targets,colors):
    indicesToKeep = breast_dataset['label'] == target
    plt.scatter(principal_breast_Df.loc[indicesToKeep, 'principal component 1']
                , principal_breast_Df.loc[indicesToKeep, 'principal component2'], c

plt.legend(targets,prop={'size':15})
```

Out[32]:

```
<matplotlib.legend.Legend at 0x25cfc230d60>

<Figure size 432x288 with 0 Axes>
```

➔ **À partir du graphique ci-dessous, nous pouvons observer que les deux classes bénigne et maligne, lorsqu'elles sont projetées dans un espace à deux dimensions, peuvent être linéairement séparables jusqu'à un certain point. D'autres observations peuvent être que la classe bénigne est étalée par rapport à la classe maligne.**



-D: Analyse en Composantes Pricipales avec la datasetCIFAR10

```
np.min(x_train),np.max(x_train)
```

(0, 255)

In[33]:

Out[33]:

In[34]:

```
x_train = x_train/255.0
```

➔ **on divise par 255 parceque Pour tout pixel donné de l'image, la valeur attribuée à ce pixel peut être comprise entre 0 et 255.**

In [35]:

```
np.min(x_train),np.max(x_train)
```

Out[35]: (0.0, 1.0)

In[36]:

```
x_train.shape
```

Out[36]: (50000, 32, 32, 3)

In[37]:
```
x_train_flat=x_train.reshape(-1,3072)
feat_cols=['pixel'+str(i)foriinrange(x_train_flat.shape[1])]
df_cifar=pd.DataFrame(x_train_flat,columns=feat_cols)
df_cifar['label'] = y_train
print('Sizeofthedataframe:{}".format(df_cifar.shape))
```

Size of the dataframe: (50000, 3073)

In [38]:
```
df_cifar.head()
```

Out[38]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9
0	0.231373	0.243137	0.247059	0.168627	0.180392	0.176471	0.196078	0.188235	0.168627	0.266667
1	0.603922	0.694118	0.733333	0.494118	0.537255	0.533333	0.411765	0.407843	0.372549	0.400000
2	1.000000	1.000000	1.000000	0.992157	0.992157	0.992157	0.992157	0.992157	0.992157	0.992157
3	0.109804	0.098039	0.039216	0.145098	0.133333	0.074510	0.149020	0.137255	0.078431	0.164706
4	0.666667	0.705882	0.776471	0.658824	0.698039	0.768627	0.694118	0.725490	0.796078	0.717647

5 rows × 3073 columns

In[39]:
```
pca_cifar=PCA(n_components=2)
principalComponents_cifar=pca_cifar.fit_transform(df_cifar.iloc[:,:-1])
```

➔ **Nous avons créé la méthode PCA et transmis le nombre de composants à deux et appliqué fit_transform sur les données d'entraînement**

In[40]:
```
principal_cifar_Df=pd.DataFrame(data=principalComponents_cifar
                , columns = ['principal component 1', 'principal component 2'])
principal_cifar_Df['y'] = y_train
principal_cifar_Df.head()
```

➔ **Ensuite, nous avons converti les principaux composants de chacune des 50 000 images d'un tableau numpy en un DataFrame pandas.**

➔

Out[40]:

	principal component 1	principal component 2	y
0	-6.401018	2.729039	6
1	0.829783	-0.949943	9
2	7.730200	-11.522102	9
3	-10.347817	0.010738	4
4	-2.625651	-4.969240	1

In[41]:

```
print('Explainedvariationperprincipalcomponent:{}'.format(pca_cifar.explained_v
```

```
Explained variation per principalcomponent:[0.2907663      0.11253144]
```

➔ La première composante principale contient 29% de l'information et la deuxième en contient 11%. 60% de l'information ont été perdus.

In [42]:

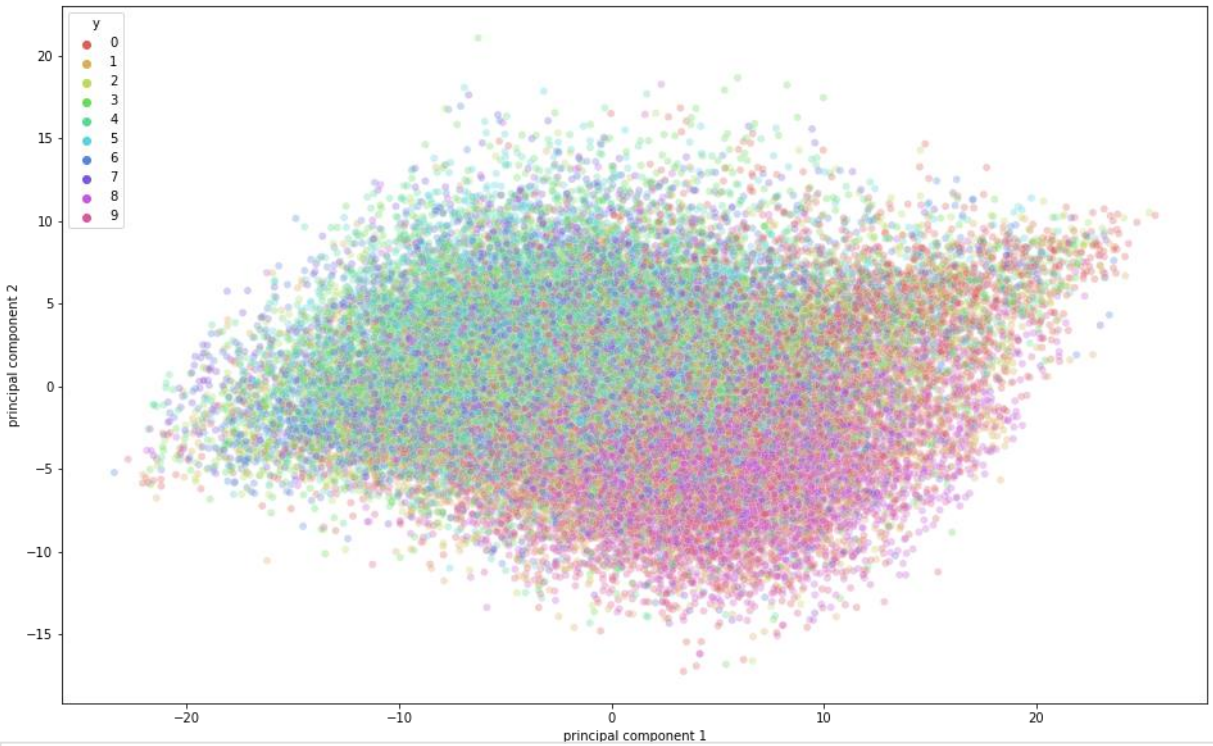
```
import seaborn as sns
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="principal component 1", y="principal component 2",
    hue="y",
    palette=sns.color_palette("hls",10),
    data=principal_cifar_Df,
    legend="full",
    alpha=0.3
)
```

```
    legend="full",
    alpha=0.3
)
```

Out[42]:
```
<AxesSubplot:xlabel='principal component 1', ylabel='principal component 2'>
```

➔ **À partir de la figure ci-dessous, nous pouvons observer qu'une certaine variation a été capturée par les composantes principales puisqu'il existe une certaine structure dans les points lorsqu'ils sont projetés le long des deux axes des composantes principales. Les points appartenant à une même classe sont proches les uns des autres, et les points ou images sémantiquement très différents sont plus éloignés les uns des autres.**

In[43]:



In[44]:
```
x_test = x_test/255.0
x_test=x_test.reshape(-1,32,32,3)
```

In[45]:
```
x_test_flat=x_test.reshape(-1,3072)
```

In[46]:
```
pca=PCA(0.9)
```

➔ **Plus tôt, nous avons passé n_components en tant que paramètre et nous pouvions alors savoir combien de variance était capturée par ces deux composants. Mais ici, nous mentionnons explicitement la quantité de variance que nous aimerions que l'ACP capture et, par conséquent, les n_composants varieront en fonction du paramètre de variance.**

In[46]:
```
pca.fit(x_train_flat)
```

Out[46]:

In[47]:

```
PCA(n_components=0.9)
pca.n_components_
```

Out[47]: 99

➔ **Nous pouvons observer que pour atteindre 90 % de variance, la dimension a été réduite à 99 composantes principales sur les 3072 dimensions réelles.**

In[48]:

```
train_img_pca = pca.transform(x_train_flat)
test_img_pca = pca.transform(x_test_flat)
```

In[50]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
from tensorflow.keras.optimizers import RMSprop
```

➔ **Importation des bibliothèques nécéssaires pour le modèle de Deep learning.**

In[51]:

```
batch_size  = 128
num_classes  =10
epochs = 20
```

➔ **Le batch size fait référence au nombre d'échantillons d'entrainement utilisés dans une seule itération.**
➔ **Epochs indique le nombre de passages de l'ensemble de données d'entraînement que l'algorithme d'apprentissage automatique a terminé.**

In[52]:

```
model = Sequential()
model.add(Dense(1024,activation='relu',input_shape=(99,)))
model.add(Dense(1024,activation='relu'))
model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(num_classes,activation='softmax'))
```

➔ **Définition du modèle Sequential.**

In[53]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                 OutputShape              Param#
=================================================================
 dense (Dense)                (None,  1024)            102400

 dense_1 (Dense)              (None,  1024)            1049600

 dense_2 (Dense)              (None,  512)             524800

 dense_3 (Dense)              (None,  256)             131328

 dense_4 (Dense)              (None,  10)              2570

=================================================================
Total params: 1,810,698
Trainable params: 1,810,698
Non-trainable params: 0
_________________________________________________________________
```

In [61]:

```
model = Sequential()
model.add(Dense(1024,activation='relu',input_shape=(3072,)))
model.add(Dense(1024,activation='relu'))
model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(num_classes,activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history=model.fit(x_train_flat,y_train,batch_size=batch_size,epochs=epochs,verbos
                  validation_data=(x_test_flat,y_test))
```

```
Epoch 1/20
```

391/391 [==============================] - 39s 98ms/step - loss: 2.1844 - accuracy: 0.2364 - val_loss: 1.8406 - val_accuracy: 0.3329
Epoch 2/20
391/391 [==============================] - 37s 96ms/step - loss: 1.8424 - accuracy: 0.3338 - val_loss: 1.7884 - val_accuracy: 0.3431
Epoch 3/20
391/391 [==============================] - 38s 98ms/step - loss: 1.7385 - accuracy: 0.3762 - val_loss: 1.7488 - val_accuracy: 0.3897
Epoch 4/20
391/391 [==============================] - 38s 97ms/step - loss: 1.6651 - accuracy: 0.4049 - val_loss: 1.6186 - val_accuracy: 0.4243
Epoch 5/20
391/391 [==============================] - 37s 94ms/step - loss: 1.6086 - accuracy:

0.4224 - val_loss: 1.6163 - val_accuracy: 0.4331
Epoch 6/20
391/391 [==============================] - 37s 96ms/step - loss: 1.5668 - accuracy: 0.4390 - val_loss: 1.6141 - val_accuracy: 0.4187
Epoch 7/20
391/391 [==============================] - 38s 97ms/step - loss: 1.5390 - accuracy: 0.4509 - val_loss: 1.5640 - val_accuracy: 0.4462
Epoch 8/20
391/391 [==============================] - 39s100ms/step - loss: 1.5023 - accuracy: 0.4655-val_loss:1.6333-val_accuracy: 0.4342
Epoch 9/20
391/391 [==============================] - 40s102ms/step - loss: 1.4712 - accuracy: 0.4766-val_loss:1.7436-val_accuracy: 0.4199
Epoch 10/20
391/391 [==============================] –39s 101ms/step – loss: 1.4541 – accuracy: 0.4820 - val_loss: 1.6186 - val_accuracy: 0.4254
Epoch 11/20
391/391 [==============================] – 40s102ms/step – loss: 1.4287 – accuracy: 0.4899 - val_loss: 1.5155 - val_accuracy: 0.4666
Epoch 12/20
391/391 [==============================] - 42s107ms/step - loss: 1.4136 - accuracy: 0.4960-val_loss:1.5709-val_accuracy: 0.4629
Epoch 13/20
391/391 [==============================] - 39s100ms/step - loss: 1.3923 - accuracy: 0.5050-val_loss:1.4787-val_accuracy: 0.4883
Epoch 14/20
391/391 [==============================] - 38s 98ms/step - loss: 1.3744 - accuracy: 0.5118 - val_loss: 1.4799 - val_accuracy: 0.4909
Epoch 15/20
391/391 [==============================] - 39s 99ms/step - loss: 1.3642 - accuracy: 0.5157 - val_loss: 1.5413 - val_accuracy: 0.4659
Epoch 16/20
391/391 [==============================] - 40s 103ms/step - loss: 1.3471 - accuracy: 0.5200 - val_loss: 1.5250 - val_accuracy: 0.4698
Epoch 17/20
391/391 [==============================] - 41s 104ms/step - loss: 1.3321 - accuracy: 0.5260 - val_loss: 1.5021 - val_accuracy: 0.4885
Epoch 18/20
391/391 [==============================] - 38s 96ms/step - loss: 1.3236 - accuracy: 0.5309 - val_loss: 1.4940 - val_accuracy: 0.4948
Epoch 19/20
391/391 [==============================] - 38s 98ms/step - loss: 1.3042 - accuracy: 0.5348 - val_loss: 1.6079 - val_accuracy: 0.4715
Epoch 20/20
391/391 [==============================] - 40s 103ms/step - loss: 1.2885 - accuracy: 0.5409 - val_loss: 1.4989 - val_accuracy: 0.4898

➔ **accuracy faible d'environ 48%. Le sparse categorical cross entropy loss est d'environs 1.2**

Travail réalisé par:
Rahma Ben Saber
Wissal Bouassida
Khalil Essouaid
Hamza Ben Hamza