PROJET 8

Déployez un modèle dans le cloud

Problématique

Votre start-up souhaite dans un premier temps se faire connaître en mettant à disposition du grand public une application mobile qui permettrait aux utilisateurs de prendre en photo un fruit et d'obtenir des informations sur ce fruit.

Pour la start-up, cette application permettrait de sensibiliser le grand public à la biodiversité des fruits et de mettre en place une première version du moteur de classification des images de fruits.

De plus, le développement de l'application mobile permettra de construire une première version de l'architecture Big Data nécessaire.

Problématique

Les données

Votre collègue Paul vous indique l'existence d'un jeu de données constitué des images de fruits et des labels associés, qui pourra servir de point de départ pour construire une partie de la chaîne de traitement des données.

Votre mission

Vous êtes donc chargé de développer dans un environnement Big Data une première chaîne de traitement des données qui comprendra le preprocessing et une étape de réduction de dimension.

Contraintes

Vous devrez tenir compte dans vos développements du fait que le volume de données va augmenter très rapidement après la livraison de ce projet.

Vous développerez donc des scripts en Pyspark et utiliserez par exemple le cloud AWS pour profiter d'une architecture Big Data (EC2, S3, IAM), basée sur un serveur EC2 Linux.

La mise en œuvre d'une architecture Big Data sous (par exemple) AWS peut nécessiter une configuration serveur plus puissante que celle proposée gratuitement (EC2 = t2.micro, 1 Go RAM, 8 Go disque serveur).

GITHUB

Les deux notenooks Jupyter, les jeux de données, le fichiers après traitement et les fichiers csv sont visibles sur mon GITHUB :

https://github.com/eleplanois/P8

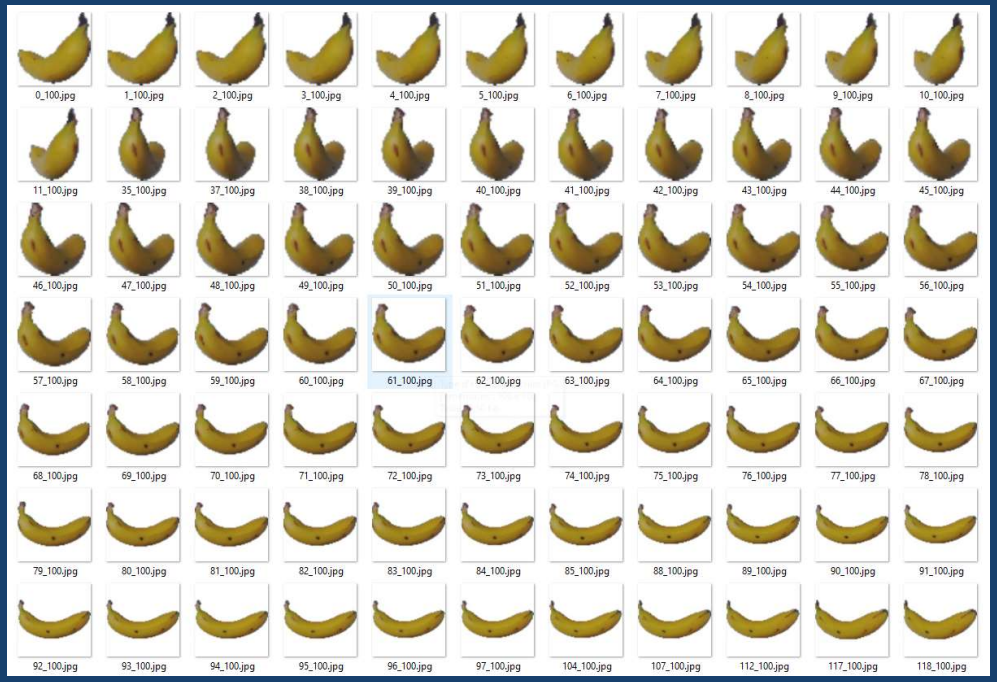
Le Jeu de Données

- 131 Types de fruits avec un repertoire par fruit
- au minimum 450 photos du fruit

Pour des raisons de cout (taille mémoire)

on va se limiter à deux jeux de données:

- 20 fruits, 25 images par fruit (500 images)
- 131 fruits, 9 images par fruit (1179 images)



Principe du traitement

- chargement des images

- débruitage et redimensionnement des images pour entrée Reseaux Neurones

- Transfer Learning : utilisation reseau pré-entraine ResNET50 pour extraction features

- PCA pour réduction dimension des features

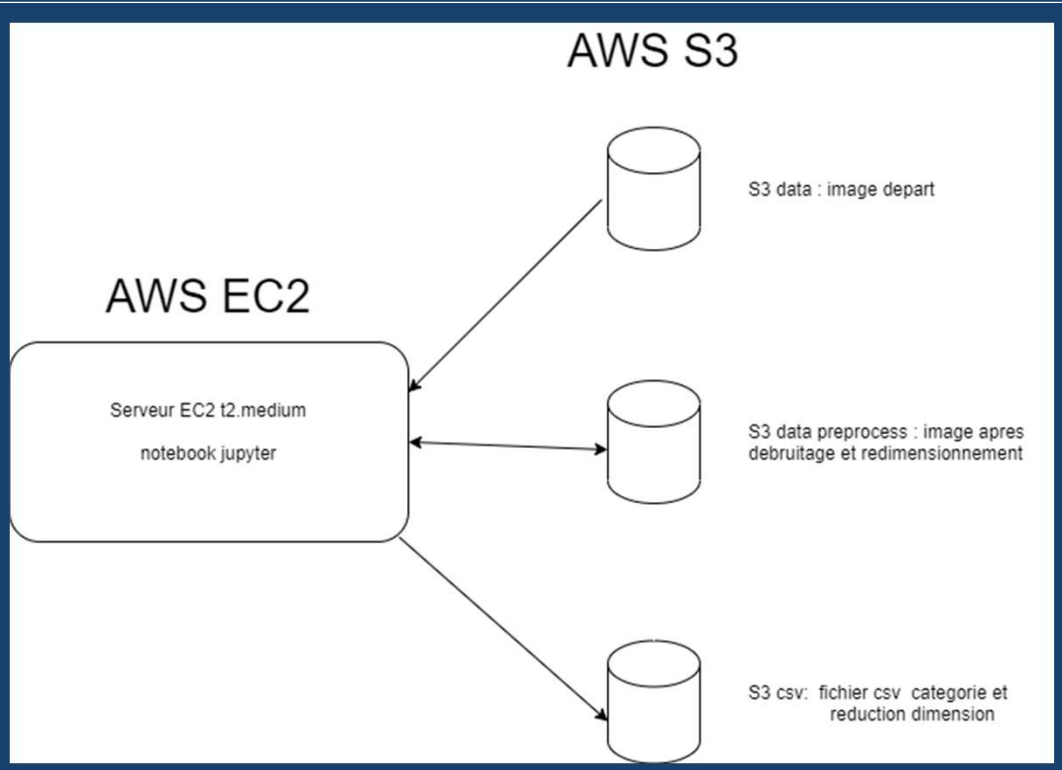
- Kmeans pour catégoriser les images

Architecture

Architecture hébergée sur Amazon Web Services :

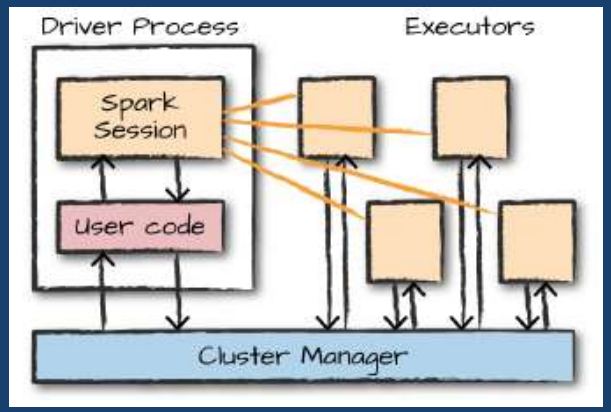
- - serveur EC2 de type t2.xlarge (4 VCPU, 16 GB mémoire)
- - stockage de fichier dans bucket S3
- - utilisation Anaconda, Keras, Spark, OpenCV, Boto3

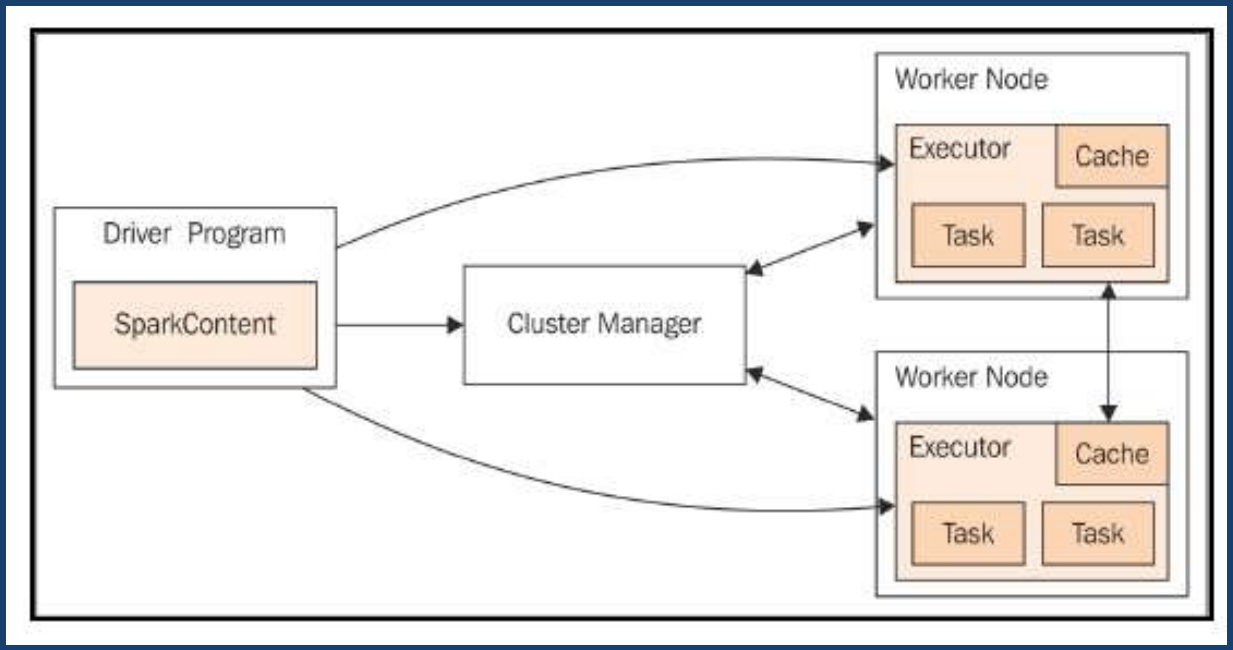
Architecture



Architecture

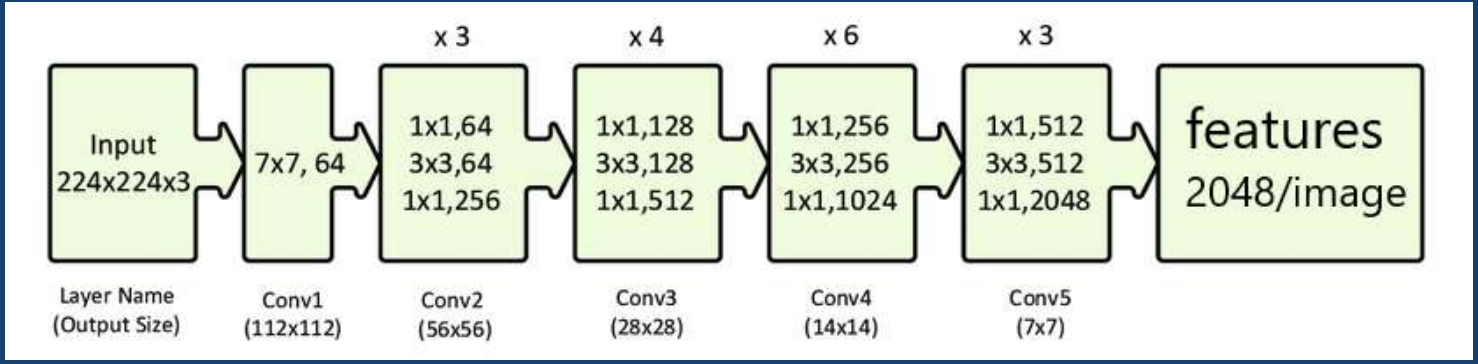
PYSPARK standAlone mode : basic cluster manager Apache Stark

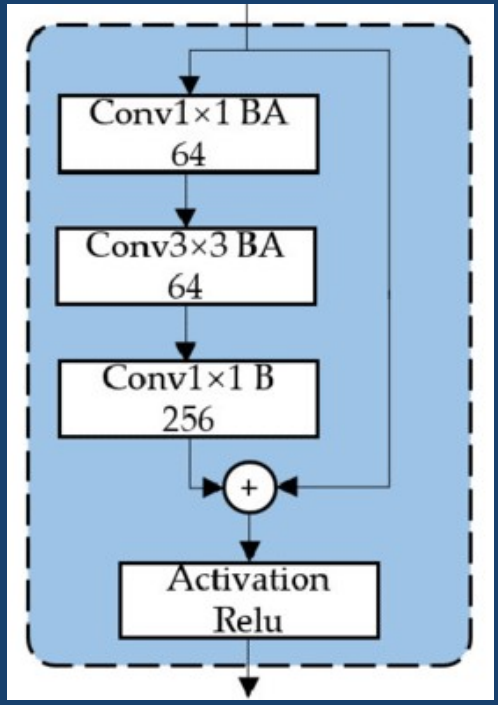




Architecture

REsNet50





Chaine de traitement

1 - Chargement des images à partir du bucket S3 dans un dataFrame Spark

```
bucket_str_data = 'elp-oc-p8-data'
bucket_str_data_upload = 'elp-oc-p8-data-preprocess'
bucket_str_data_csv = 'elp-oc-p8-data-csv'
echantillon = "data_sample_009/"

uri_data_S3 = "s3a://"+bucket_str_data+"/"+echantillon
uri_data_upload_S3 = "s3a://"+bucket_str_data_upload+"/"+echantillon

bucket = s3.Bucket(bucket_str_data)
bucket_upload = s3.Bucket(bucket_str_data_upload)
```

```
img_df = spark.read.format("image").load(uri_data_S3+"*/*.jpg")
```

```
img_df.printSchema()
```

```
root
 |-- image: struct (nullable = true)
 |    |-- origin: string (nullable = true)
 |    |-- height: integer (nullable = true)
 |    |-- width: integer (nullable = true)
 |    |-- nChannels: integer (nullable = true)
 |    |-- mode: integer (nullable = true)
 |    |-- data: binary (nullable = true)
```

Chaine de traitement

2 - création des catégories et key par fonction pyspark.sql.functions

```
[16]: img_df2 = img_df.withColumn("categ", split("image.origin","/").getItem(3))

[17]: img_df3 = img_df2.withColumn("file", split("image.origin","/").getItem(4))

[18]: img_df3.select("image.origin", "categ", "file").take(1)

[18]: [Row(origin='s3a://elp-oc-p8-data/Raspberry/8_100.jpg', categ='Raspberry', file='8_100.jpg')]

[19]: def def_key(categ, file):
          return(categ+"/"+file)

      def_key_UDF = udf(def_key, StringType())

[20]: img_df4 = img_df3.withColumn("key", def_key_UDF("categ","file"))

[21]: img_df4.printSchema()

      root
       |-- image: struct (nullable = true)
       |    |-- origin: string (nullable = true)
       |    |-- height: integer (nullable = true)
       |    |-- width: integer (nullable = true)
       |    |-- nChannels: integer (nullable = true)
       |    |-- mode: integer (nullable = true)
       |    |-- data: binary (nullable = true)
       |-- categ: string (nullable = true)
       |-- file: string (nullable = true)
       |-- key: string (nullable = true)
```

Chaine de traitement

3 - preprocessing des images

Debruitage et redimensionnement pour entrée réseau neurones

[22]:
```
def preprocess_image(data):
    img_rgb = np.array(data).reshape(100,100,3)
    image_blur = cv2.GaussianBlur(img_rgb, (3, 3), 0)
    image_resize = cv2.resize(image_blur, (224, 224))
    image_retour = cv2.cvtColor(image_resize, cv2.COLOR_BGR2RGB)
    image_flatten = image_retour.flatten()
    image_bytes = image_flatten.tobytes()
    return image_bytes

preprocess_imageUDF = udf(preprocess_image, BinaryType())

spark_preproces = img_df4.withColumn("preprocess_data", preprocess_imageUDF("image.data"))
```

[23]:
```
img_preprocess = spark_preproces.select("preprocess_data").take(1)
```

Chaine de traitement

4 - sauvegarde des images dans bucket S3

```
def upload_S3(vgg_image, image_key):
    """
        input : ( image a sauver, KEY dans bucket S3 pour sauver )
    """
#    print("envoi : ",image_key)
    image_np = np.array(vgg_image).reshape(224,224,3)

    img_data = BytesIO()
    plt.figure(figsize=(2.24, 2.24), dpi=100, clear=True)
    plt.imshow(image_np)
    plt.axis('off')
    plt.savefig(img_data, format='jpeg', dpi=100)
    img_data.seek(0)
    bucket_upload.put_object(Body=img_data, ContentType='image/jpeg', Key=image_key)
    plt.close()

    return "OK"
```

```
for i,elt in enumerate(spark_save):
    if i%20 == 0 : print(i)
    upload_S3(elt['preprocess_data'], elt['key'])
```

```
0
20
40
60
80
100
120
140
160
```

Chaine de traitement

5 - transfer learning pour extraction features

```
[35]: #utilisation transfer learning pour extraction features
      resnet = ResNet50()
      # remove the output layer
      model = Model(inputs=resnet.inputs, outputs=resnet.layers[-2].output)
      #model.summary()

[36]: bc_model_weights = spark.sparkContext.broadcast(model.get_weights())

[37]: def model_fn():
          modelvgg = ResNet50(weights=None)
          model = Model(inputs=modelvgg.inputs, outputs=modelvgg.layers[-2].output)
          model.set_weights(bc_model_weights.value)
          return model
```

Chaine de traitement

5 - transfer learning pour extraction features

```
@pandas_udf("array<float>")
def func(ser: pd.Series) -> pd.Series:
    mod = model_fn()
    lis = []
    for i in ser:
        image_RGB = mpimg.imread(BytesIO(i), 'jpg')
        image = img_to_array(image_RGB)
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        features = mod.predict(image)
        lis.append(features[0])
    return pd.Series(lis)
```

```
spark_df_vgg = spark_df_files3_upload.withColumn("vgg_feat", func("image"))
```

Chaine de traitement

6 - PCA pour réduction dimension

```
n_components = 20
pca = PCA(
    k = n_components,
    inputCol = 'vgg_vectores',
    outputCol = 'pcaFeatures'
).fit(spark_df_vgg_vector)

df_pca = pca.transform(spark_df_vgg_vector)
print('Explained Variance Ratio', pca.explainedVariance.toArray().sum())
df_pca.show(6)
```

```
Explained Variance Ratio 0.7834938216106161
+--------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|         categ|                 key|               image|            vgg_feat|        vgg_vectores|         pcaFeatures|
+--------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|Apple_Braeburn|data_sample_025_0...|[FF D8 FF E0 00 1...|[1.6289237, 1.030...|[1.62892365455627...|[11.9149547732828...|
|Apple_Braeburn|data_sample_025_0...|[FF D8 FF E0 00 1...|[1.8623443, 0.949...|[1.86234426498413...|[11.0990851646723...|
|Apple_Braeburn|data_sample_025_0...|[FF D8 FF E0 00 1...|[1.1087763, 0.165...|[1.10877633094787...|[11.6145714117175...|
|Apple_Braeburn|data_sample_025_0...|[FF D8 FF E0 00 1...|[1.1596127, 0.285...|[1.15961265563964...|[11.2803308465362...|
|Apple_Braeburn|data_sample_025_0...|[FF D8 FF E0 00 1...|[1.3617629, 0.331...|[1.36176288127899...|[10.7527042475825...|
|Apple_Braeburn|data_sample_025_0...|[FF D8 FF E0 00 1...|[1.119874, 0.4313...|[1.11987400054931...|[12.1570689296424...|
+--------------+--------------------+--------------------+--------------------+--------------------+--------------------+
only showing top 6 rows
```

Chaine de traitement

7

- sauvegarde categorie
  bucket S3 csv

```
df_categ = df_pca.select('categ', 'key').toPandas()
```

```
df_categ
```

	categ	key
0	Apple_Braeburn	data_sample_025_020/data_sample_025_020/Apple_...
1	Apple_Braeburn	data_sample_025_020/data_sample_025_020/Apple_...
2	Apple_Braeburn	data_sample_025_020/data_sample_025_020/Apple_...
3	Apple_Braeburn	data_sample_025_020/data_sample_025_020/Apple_...
4	Apple_Braeburn	data_sample_025_020/data_sample_025_020/Apple_...
...	...	...
495	Watermelon	data_sample_025_020/data_sample_025_020/Waterm...
496	Watermelon	data_sample_025_020/data_sample_025_020/Waterm...
497	Watermelon	data_sample_025_020/data_sample_025_020/Waterm...
498	Watermelon	data_sample_025_020/data_sample_025_020/Waterm...
499	Watermelon	data_sample_025_020/data_sample_025_020/Waterm...

500 rows × 2 columns

```
csv_buffer = StringIO()
df_categ.to_csv(csv_buffer, sep=';')
s3_resource = boto3.resource('s3')
file_csv = echantillon+'df_categ.csv'
s3_resource.Object(bucket_str_data_csv, file_csv).put(Body=csv_buffer.getvalue())
```

```
{'ResponseMetadata': {'RequestId': 'Q84YXDSF3MS88D2S',
  'HostId': 'MAhh7sDwyG9u1z1b/jpi0oc0IuBSpgqtK6NeWOdlvxfFwy/XgAmfIDznvs2RnTru4ix+r53vF1A=',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amz-id-2': 'MAhh7sDwyG9u1z1b/jpi0oc0IuBSpgqtK6NeWOdlvxfFwy/XgAmfIDznvs2RnTru4ix+r53vF1A=',
   'x-amz-request-id': 'Q84YXDSF3MS88D2S',
   'date': 'Tue, 14 Sep 2021 16:02:13 GMT',
   'etag': '"2ec6aab3f98f8ec11228f3f5b89191aa"',
   'server': 'AmazonS3',
   'content-length': '0'},
  'RetryAttempts': 0},
 'ETag': '"2ec6aab3f98f8ec11228f3f5b89191aa"'}
```

Chaine de traitement BONUS

8 - Kmeans pour clustériser

```
km_acp = KMeans(n_clusters=20)
km_acp.fit(pca_vgg)
```

```
KMeans(n_clusters=20)
```

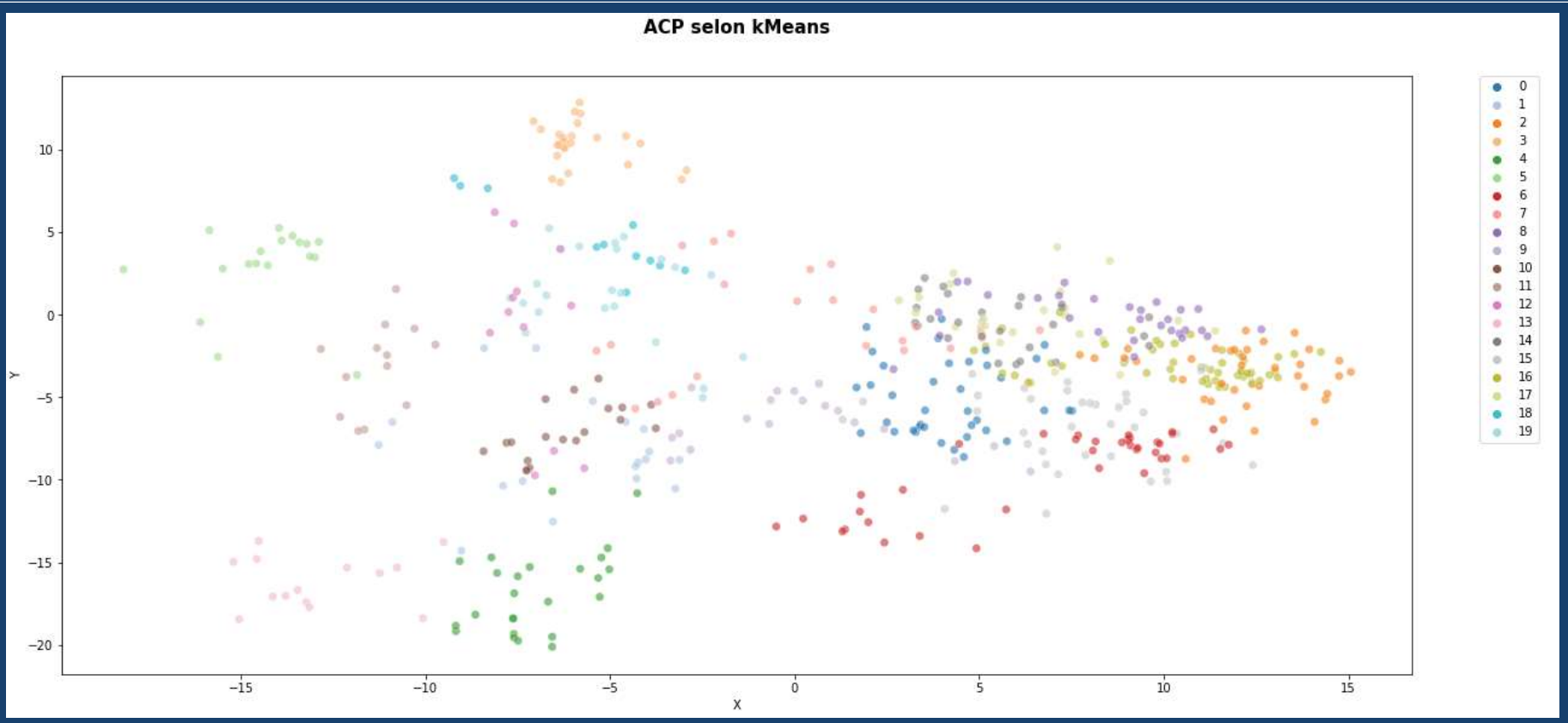
9 - PCA 2D pour affichage

```
n_components = 2
pca = PCA(
    k = n_components,
    inputCol = 'vgg_vectores',
    outputCol = 'pca2D'
).fit(spark_df_vgg_vector)

df_affic = pca.transform(spark_df_vgg_vector)
```

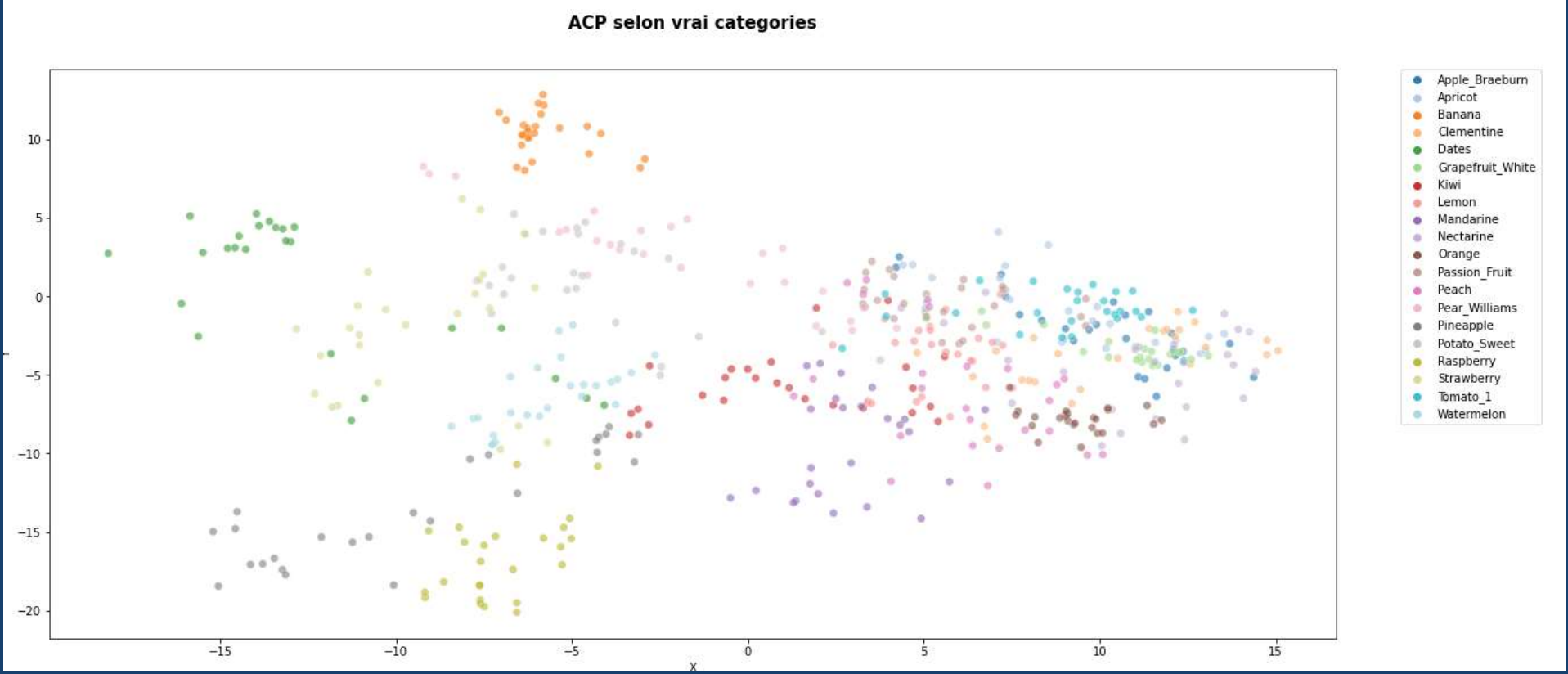
10 - TSNE pour affichage

```
tsne = TSNE(n_components=2, perplexity=30,
                    n_iter=2000, init='random', random_state=1944)

X_tsne_sift = tsne.fit_transform(pca_vgg)
df_tsne_sift = pd.DataFrame(X_tsne_sift[:,0:2], columns=['tsne1', 'tsne2'])
print(X_tsne_sift.shape)
```

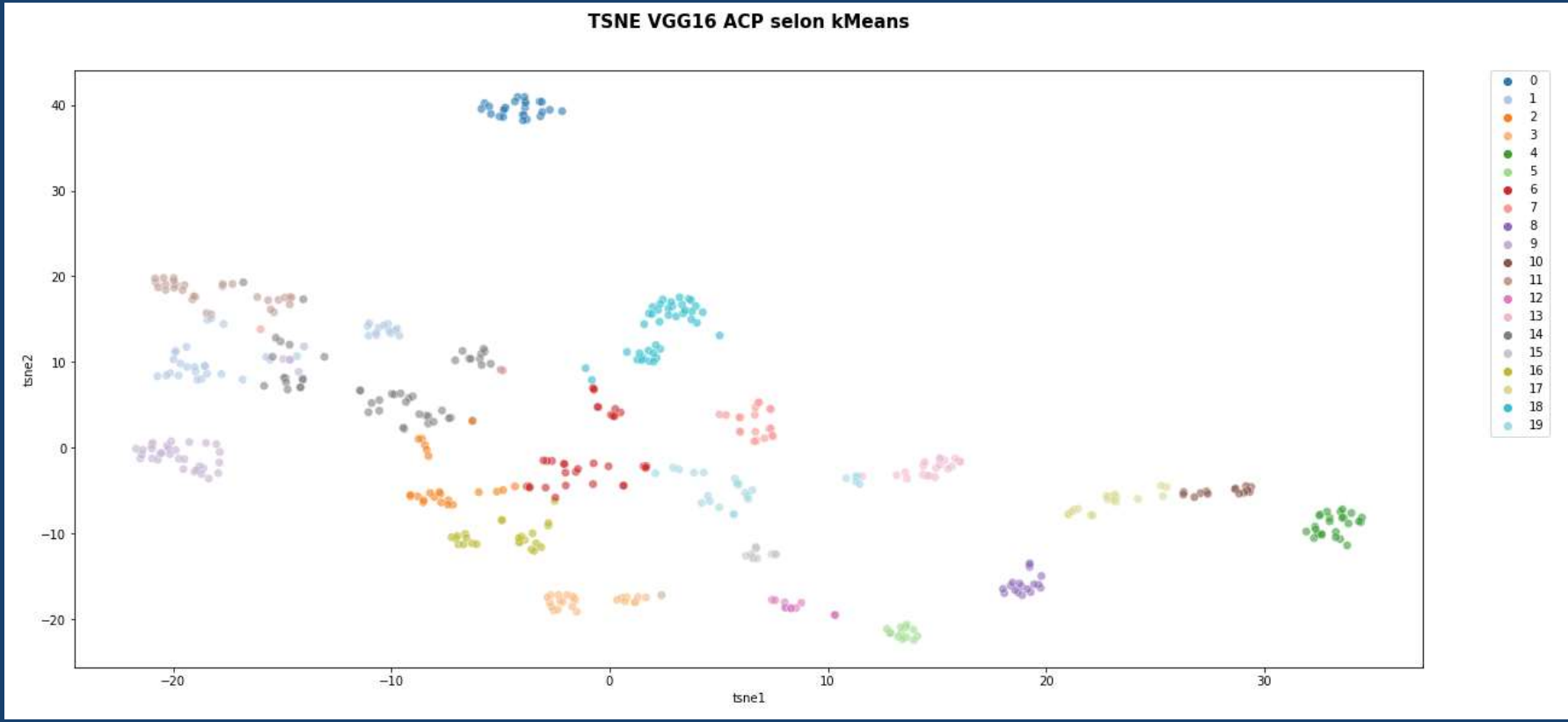
Chaine de traitement BONUS



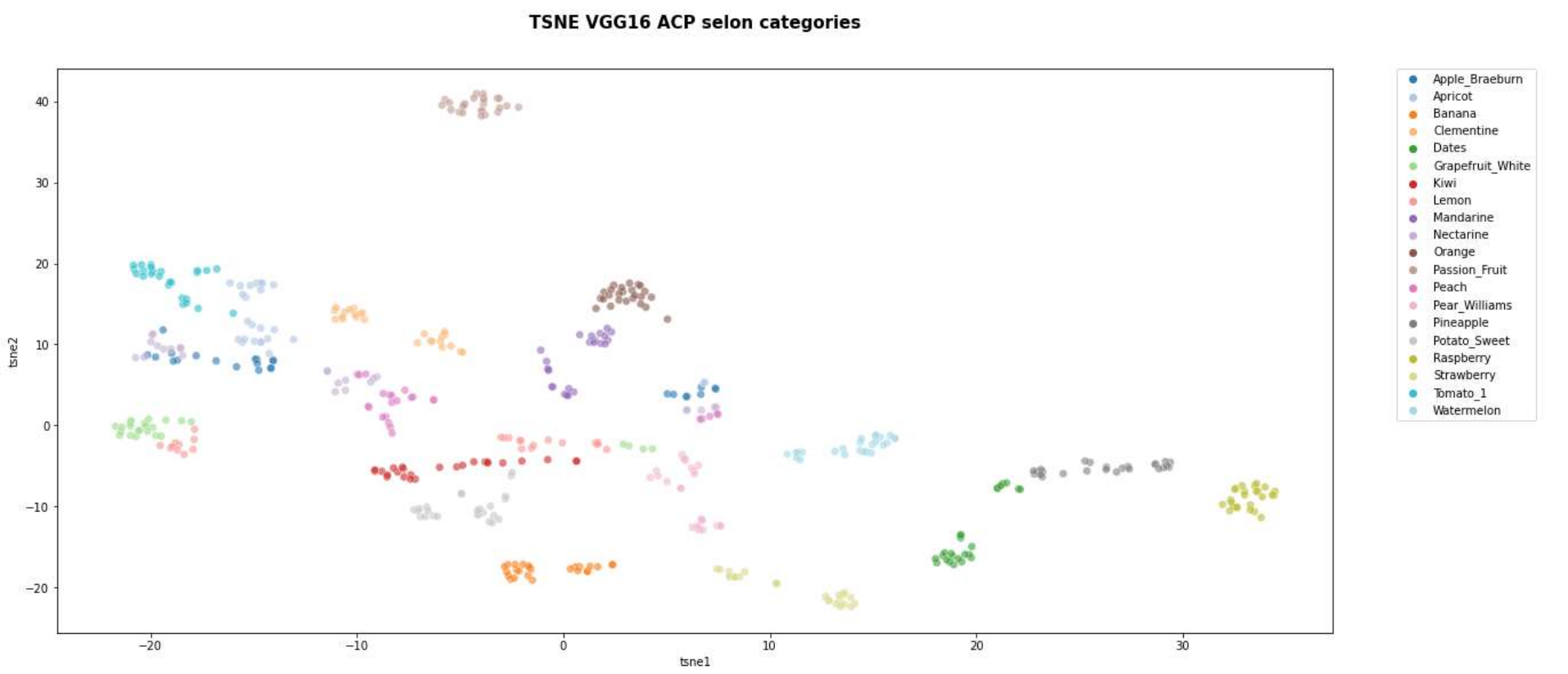
Chaine de traitement BONUS



Chaine de traitement BONUS



Chaine de traitement BONUS



Point d’attention CHARGE

Deux gros process consommateur de CPU et RAM :

- PCA
- passage du dataframe SPARK à PANDAS

MERCI

Questions et Réponses