

Déployez un modèle dans le cloud

Projet 8 du parcours Data Scientist

Dernière Màj
28 août 2023

Raquel Sanchez Pellicer



Contexte



Environnement Big Data



Traitement images



Architecture de développement



Conclusions et perspectives



Annexes

"Fruits!", jeune start-up de l'AgriTech, cherche à proposer des **solutions innovantes pour la récolte des fruits**.

La volonté de l'entreprise est de préserver la biodiversité des fruits en permettant des traitements spécifiques pour chaque variété de fruits en développant des robots cueilleurs intelligents.

Nous souhaitons dans un premier temps **nous faire connaître** en mettant à disposition du grand public une **application mobile** qui permettant aux utilisateurs de **prendre en photo un fruit et d'obtenir des informations sur ce fruit**.

Cette application permettra 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**.



Un alternant a formalisé un document dans lequel il teste une première approche dans un environnement Big Data. Le notebook réalisé par l'alternant servira de point de départ pour construire une partie de la chaîne de traitement des données.

Objectif :

Réviser la chaîne de traitement proposée par l'alternant pour le développement du moteur de classification, il n'est **pas nécessaire d'entraîner un modèle** pour le moment, et la **compléter** avec une étape de réduction **de dimensions**.

Mettre en place les premières briques de traitement qui serviront lorsqu'il faudra passer à l'échelle en termes de volume de données.



Données : 90483 images
100x100 pixels
131 fruits

Training dataset : 67692 images

Test dataset : 22688 images

Test multiple fruits : 103 images

Notebook avec première approche





Contexte



Environnement Big Data



Traitement images



Architecture de développement

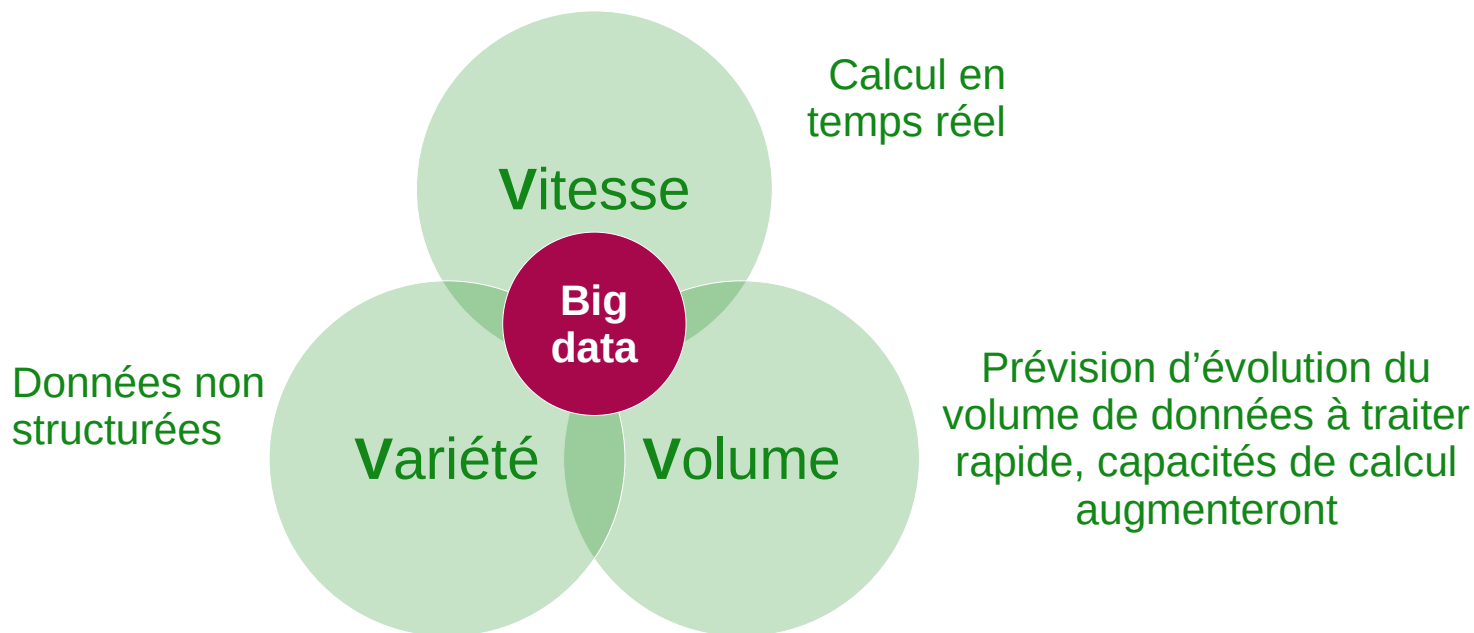


Conclusions



Annexes

Enjeux



La **tolérance aux pannes** : loi de Murphy stipule, la probabilité qu'un composant tombe en panne tend vers 1 avec le temps

Une **bonne maintenabilité** : architecture facile à maintenir et à modifier.

Un **coût faible** : déployer des composants simples, ajustés aux besoins pour minimiser ces coûts.

Services Cloud

























SaaS Software as a service				
FaaS Function as a service				
DBaaS Database as a service				
PaaS Platform as a service				
STaaS Storage as a service				
IaaS Infrastructure as a service				

Image du cours découvrez la cloud avec AWS d'OpenClassrooms

Simple Storage Service

- Données
- Code chaîne de traitement des images
- Informations configuration EMR, bootstrapping
- Résultats



S3

EMR



Elastic Map Reduce

- Gestion instances EC2 (Master & slaves)
- Installation packages complémentaires (Spark, Tensorflow, JupyterHub)

IAM



Identity and Access Manager

- Service de sécurité
- Gestion des accès

Avantages

- Essai gratuit + *Pay as you go*
- Régulation automatique puissance loué
- Résolution des problèmes techniques par AWS



Contexte



Environnement Big Data



Traitement images



Architecture de développement



Conclusions

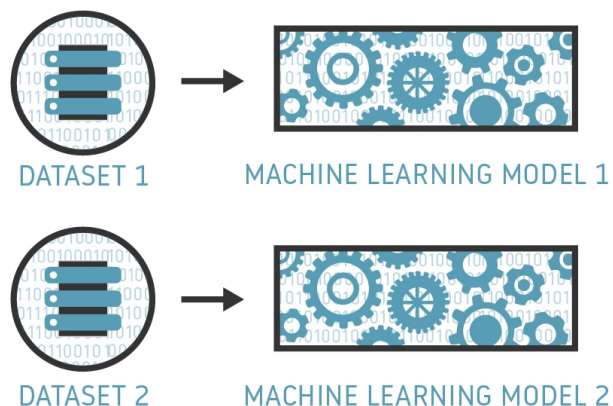


Annexes

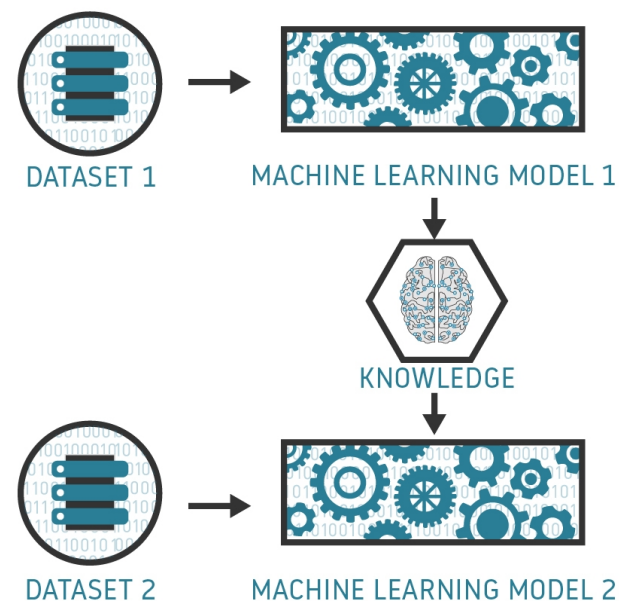
Objectif : Moteur classification données

Solution : Transfer learning >> **MobileNetV2**

TRADITIONAL MACHINE LEARNING



TRANSFER LEARNING



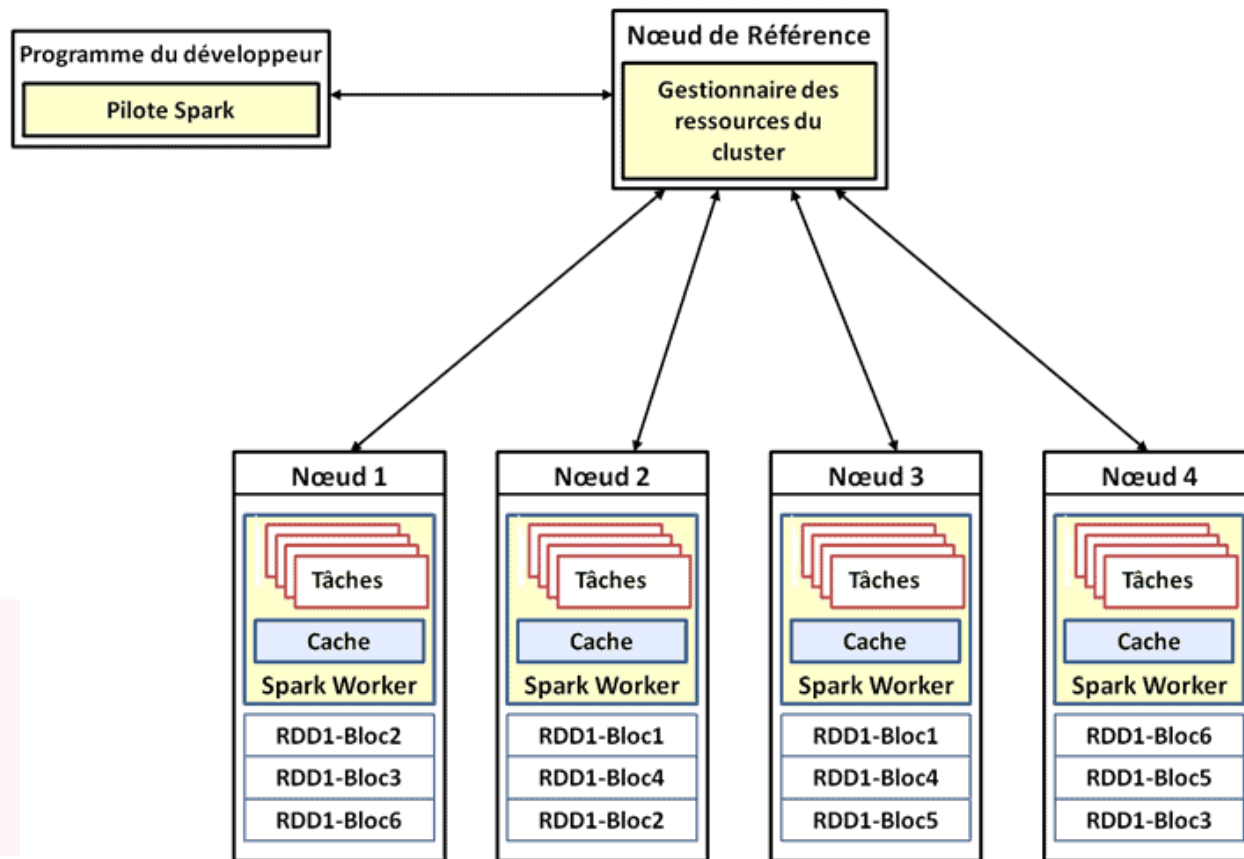
Objectif : Moteur classification données

Solution : Transfer learning >> **MobileNetV2**

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

Architecture du modèle MobileNetV2. Avec t : facteur d'expansion, c : nombre de canaux de sortie, n : nombre de répétitions, s : stride. La convolution spatiale utilise kernels 3×3 .

Calcul distribué



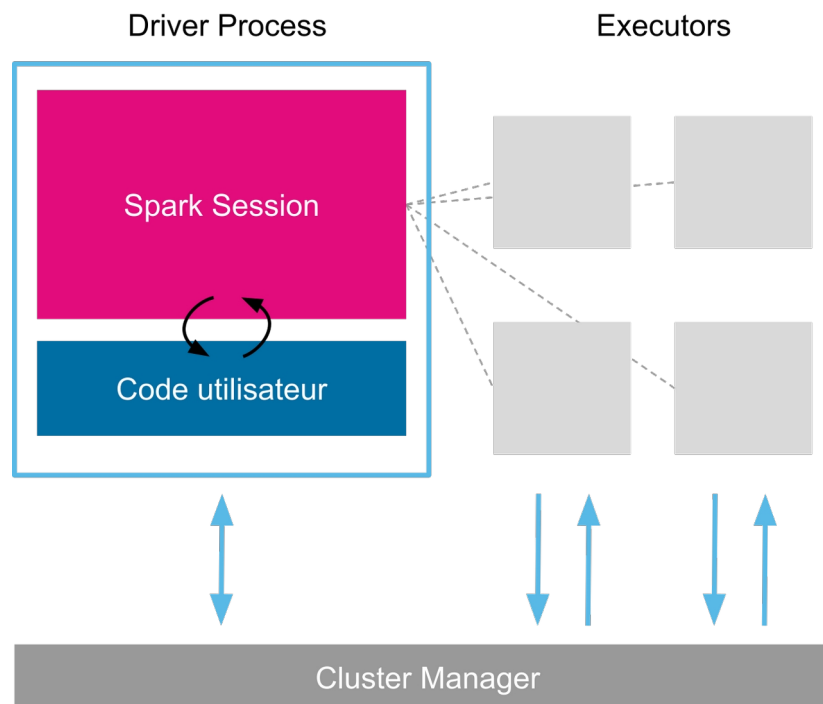
Avantages :

- Équilibrage de la charge
- Optimisation des transferts
- Scalabilité
- Tolérance aux pannes

Objectif : **solution performante et scalable**, capable de traiter un gros volume de données

Utilisation de la technologie **Spark** pour distribuer les calculs sur plusieurs cœurs / machines

Les traitements sont donc développés en **pyspark**



Les étapes suivantes ont été implémentées :

1. Création d'une **SparkSession** (driver process) et la variable **sparkContext** (connexion à un cluster Spark)

2. **Charger** les images et les associer aux images leur **label**

3. Préparation du modèle. Importer le modèle **MobileNetV2**

Créer un **nouveau modèle dépourvu de la dernière couche** de MobileNetV2

Broadcast des "weights" du modèle

4. Définition du processus de chargement des images et application de leur featurisation

Redimensionner les images pour qu'elles soient compatibles avec le modèle

Extraction de features à travers l'utilisation de pandas UDF

5. Réduction des dimensions via un **PCA**

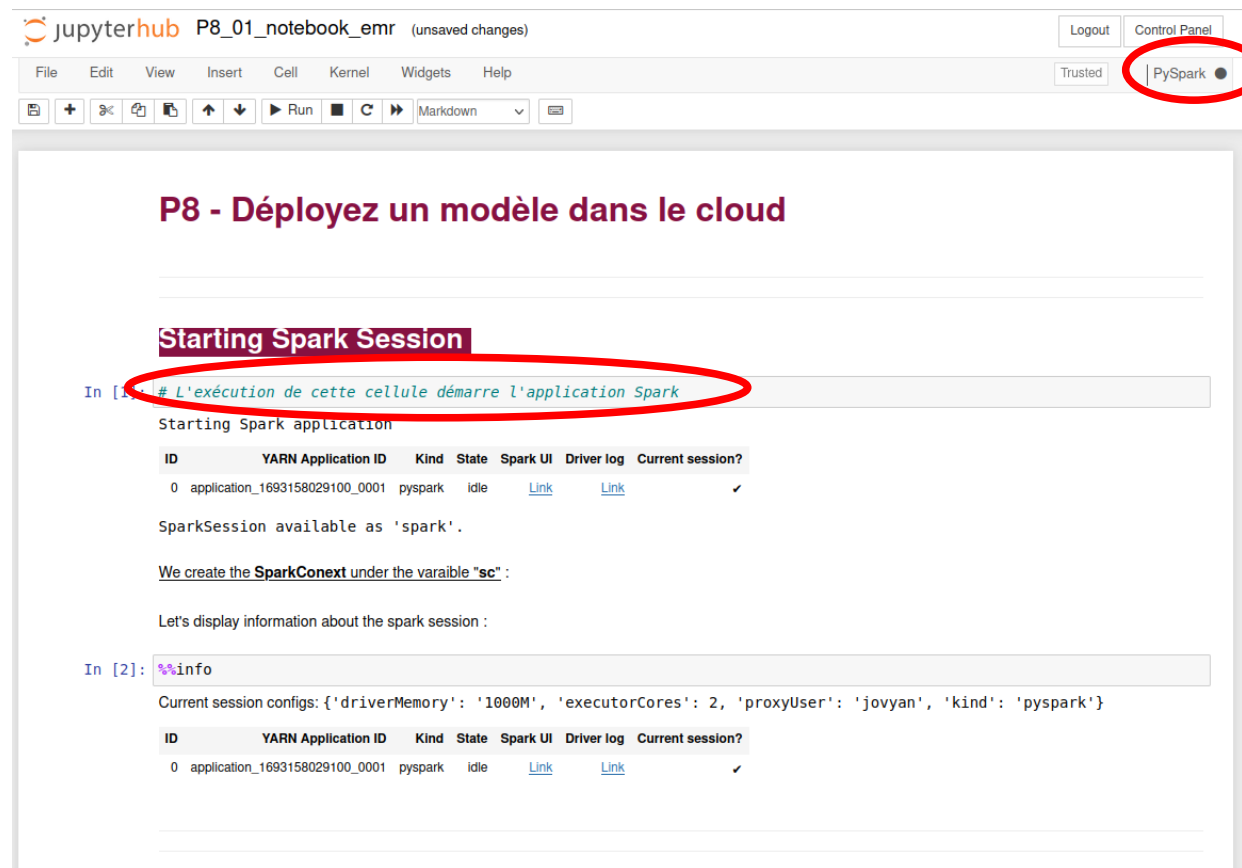
Recherche nombre optimal composantes expliquant 95 % variance

6. Sauvegarde du résultat de nos actions

Les étapes suivantes ont été implémentées :

1. Création d'une **SparkSession** (driver process) et la variable **sparkContext**

EMR



jupyterhub P8_01_notebook_emr (unsaved changes) Logout Control Panel

File Edit View Insert Cell Kernel Widgets Help Trusted PySpark

P8 - Déployez un modèle dans le cloud

Starting Spark Session

In [1]: `# L'exécution de cette cellule démarre l'application Spark`

Starting Spark application

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1693158029100_0001	pyspark	idle	Link	Link	✓

SparkSession available as 'spark'.

We create the **SparkContext** under the variable "sc" :

Let's display information about the spark session :

In [2]: `%info`

Current session configs: {'driverMemory': '1000M', 'executorCores': 2, 'proxyUser': 'jovyan', 'kind': 'pyspark'}

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1693158029100_0001	pyspark	idle	Link	Link	✓

2. Charger les images et les associer aux images leur label

Exploration des données

Nous accédons directement à nos données sur S3 comme si elles étaient stockées localement.

```
In [4]: # Define the folder containing the files with the project data
s3_P8 = "s3://rsp-oc-p8-fruits/"
PATH_DataTest = s3_P8+'Test/'
PATH_Result = s3_P8 + "Results/"
```

```
print('PATH Project:      '\
      s3_P8+'\n\nPATH_DataTest:  '\
      PATH_DataTest+'\n\nPATH_Result:  '\
      PATH_Result)
```

```
PATH Project:      s3://rsp-oc-p8-fruits/
```

```
PATH_DataTest:     s3://rsp-oc-p8-fruits/Test/
```

```
PATH_Result:       s3://rsp-oc-p8-fruits/Results/
```

- Les images sont chargées au format binaire, ce qui offre, plus de souplesse dans la façon de prétraiter les images.
- Seulement les fichiers dont l'extension est **jpg** seront chargés.
- Les fichiers contenus dans les sous-dossiers du dossier communiqué seront également chargés.

```
In [5]: images = spark.read.format("binaryFile") \
        .option("pathGlobFilter", "*.jpg") \
        .option("recursiveFileLookup", "true") \
        .load(PATH_DataTest)
```

Seulement le **path** de l'image est conservé, une colonne contenant les **labels** de chaque image est ajouté :

```
In [7]: images = images.withColumn('label', element_at(split(images['path'], '/'),-2))
print(images.printSchema())
print(images.select('path','label').show(5,False))
```

3. Préparation du modèle. Importer le modèle **MobileNetV2**

Créer un **nouveau modèle dépourvu de la dernière couche** de MobileNetV2

Broadcast des “weights” du modèle

```
In [8]: model = MobileNetV2(weights='imagenet',
                             include_top=True,
                             input_shape=(224, 224, 3))

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224.h5
14540800/14536120 [=====] - 1s 0us/step
```

```
In [9]: new_model = Model(inputs=model.input,
                          outputs=model.layers[-2].output)
```

```
In [10]: broadcast_weights = sc.broadcast(new_model.get_weights())
```

```
In [11]: new_model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
=====			
input_1 (InputLayer)	[(None, 224, 224, 3)]	0	
Conv1 (Conv2D)	(None, 112, 112, 32)	864	input_1[0][0]
bn_Conv1 (BatchNormalization)	(None, 112, 112, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 112, 112, 32)	0	bn_Conv1[0][0]
expanded_conv_depthwise (Depthwise Conv2D)	(None, 112, 112, 32)	288	Conv1_relu[0][0]
expanded_conv_depthwise_BN (BatchNormalization)	(None, 112, 112, 32)	128	expanded_conv_depthwise[0][0]
expanded_conv_depthwise_relu (ReLU)	(None, 112, 112, 32)	0	expanded_conv_depthwise_BN[0][0]
expanded_conv_project (Conv2D)	(None, 112, 112, 16)	512	expanded_conv_depthwise_relu[0][0]

```
In [11]: new_model.summary()
```

block_16_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_16_depthwise_BN[0][0]
block_16_project (Conv2D)	(None, 7, 7, 320)	307200	block_16_depthwise_relu[0][0]
block_16_project_BN (BatchNormalization)	(None, 7, 7, 320)	1280	block_16_project[0][0]
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409600	block_16_project_BN[0][0]
Conv_1_bn (BatchNormalization)	(None, 7, 7, 1280)	5120	Conv_1[0][0]
out_relu (ReLU)	(None, 7, 7, 1280)	0	Conv_1_bn[0][0]
global_average_pooling2d (Global Average Pooling)	(None, 1280)	0	out_relu[0][0]
=====			
Total params: 2,257,984			
Trainable params: 2,223,872			
Non-trainable params: 34,112			

4. Définition du processus de chargement des images et application de leur featurisation

Redimensionner les images pour qu'elles soient compatibles avec le modèle

Extraction de features à travers l'utilisation de pandas UDF

```
In [23]: # Resize images to 224x224
def preprocess(content):
    """
    Preprocesses raw image bytes for prediction.
    """
    img = Image.open(io.BytesIO(content)).resize([224, 224])
    arr = img_to_array(img)
    return preprocess_input(arr)

# Featurize images and return a series of vectors (flattened tensors)
def featurize_series(model, content_series):
    """
    Featurize a pd.Series of raw images using the input model.
    :return: a pd.Series of image features
    """
    input = np.stack(content_series.map(preprocess))
    preds = model.predict(input)
    # For some layers, output features will be multi-dimensional tensors.
    # We flatten the feature tensors to vectors for easier storage in Spark DataFrames.
    output = [p.flatten() for p in preds]
    return pd.Series(output)

@pandas_udf('array<float>', PandasUDFType.SCALAR_ITER)
def featurize_udf(content_series_iter):
    """
    This method is a Scalar Iterator pandas UDF wrapping our featurization function.
    The decorator specifies that this returns a Spark DataFrame column of type ArrayType(FloatType).

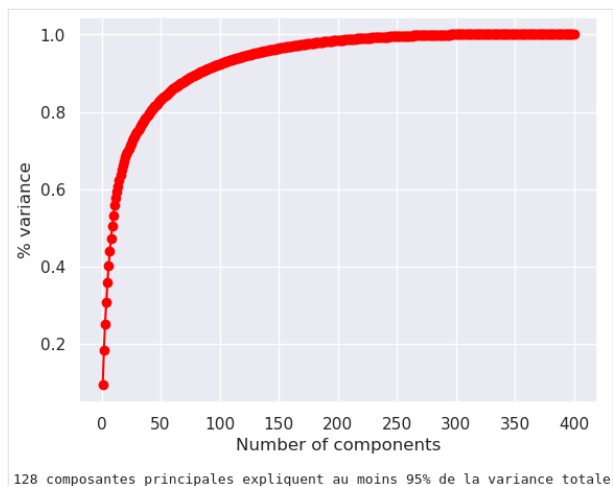
    :param: content_series_iter, This argument is an iterator over batches of data, where each batch
           is a pandas Series of image data.
    """
    # With Scalar Iterator pandas UDFs, we can load the model once and then re-use it
    # for multiple data batches. This amortizes the overhead of loading big models.
    model = model_fn()
    for content_series in content_series_iter:
        yield featurize_series(model, content_series)
```

path	label	features
file:/home/raquel...	Raspberry	[0.29254088, 1.06...
file:/home/raquel...	Strawberry	[2.20994, 0.09814...
file:/home/raquel...	Strawberry	[1.6356167, 0.0, ...]
file:/home/raquel...	Peach	[0.13757613, 0.0, ...]
file:/home/raquel...	Tomato not Ripened	[0.0, 0.4384215, ...]

only showing top 5 rows

5. Réduction des dimensions via un PCA

Recherche nombre optimal composantes expliquant 95 % variance



Réduction des dimensions via PCA

Préparation des données

```
In [18]: # First step is to convert our features arrays into vectors
array_to_vector_udf = udf(lambda l: Vectors.dense(l), VectorUDT())
df_with_vectors = features_df.select(
    features_df["path"],
    features_df["label"],
    features_df["features"],
    array_to_vector_udf(features_df["features"]).alias("vectorFeatures"),
)
# show the 5 first values :
df_with_vectors.show(5)
```

path	label	features	vectorFeatures
s3://rsp-oc-p8-fr...	Watermelon	[0.46832162, 0.22...	[0.46832162141799...
s3://rsp-oc-p8-fr...	Pineapple Mini	[0.0, 4.7068405, ...]	[0.0, 4.7068405151...
s3://rsp-oc-p8-fr...	Watermelon	[0.051970564, 0.1...	[0.05197056382894...
s3://rsp-oc-p8-fr...	Watermelon	[0.11681207, 0.13...	[0.11681207269430...
s3://rsp-oc-p8-fr...	Cauliflower	[0.0, 0.4670273, ...]	[0.0, 0.4670273065...

only showing top 5 rows

```
In [19]: # Second step is to scale the data before applying the PCA process
scaler = StandardScaler(inputCol="vectorFeatures",
    outputCol="scaledFeatures",
    withMean=True, withStd=True
).fit(df_with_vectors)

# when we transform the dataframe, the old feature will still remain in it
df_scaled = scaler.transform(df_with_vectors)
# show the 5 first values :
df_scaled.show(5)
```

path
s3://rsp-oc-p8-fr...

Réduction de dimensions via PCA

```
In [20]: # We will use the optimal number of components for explaining 95% of variance
# identified during the local development
n_components = 128

# Apply the PCA with n_components
pca = PCA(k=n_components, inputCol='scaledFeatures', outputCol='pcaFeatures')

model_pca = pca.fit(df_scaled)
df_pca = model_pca.transform(df_scaled)

# show the 5 first values :
df_pca.show(5)
```

6. Sauvegarde du résultat de nos actions

Enregistrement des résultats

Enregistrement des données traitées au format "parquet" :

```
In [21]: df_results = df_pca.select(["path", "label", "pcaFeatures"])
```

```
In [22]: df_results.write.mode("overwrite").parquet(PATH_Result)
```



Contexte



Environnement Big Data



Traitement images



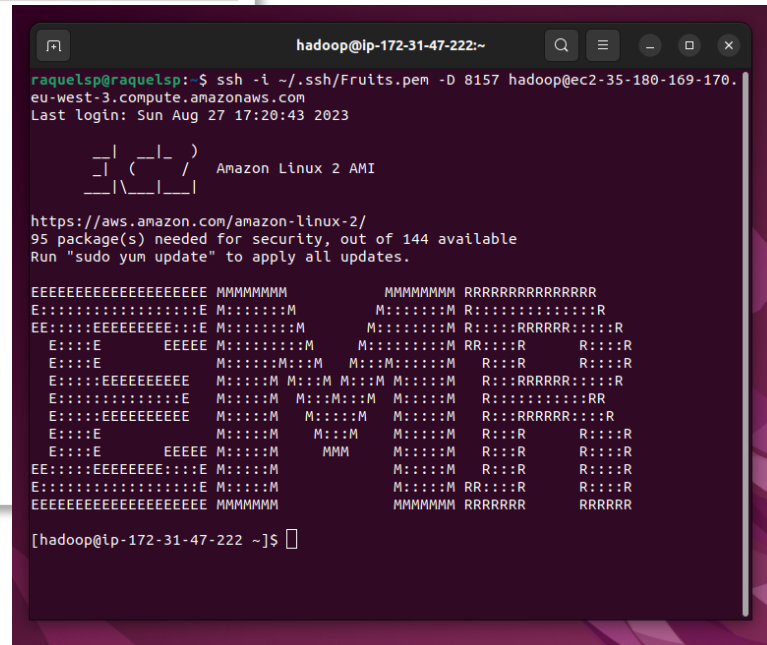
Architecture de développement



Conclusions



Annexes



aws

Services

Rechercher

[Alt+S]

Amazon S3

Compartment

Points d'accès

Points d'accès de l'objet Lambda

Points d'accès multi-région

Opérations par lot

IAM Access Analyzer pour S3

Paramètres de blocage de l'accès public pour ce compte

Storage Lens

Tableaux de bord

Paramètres AWS Organizations

Fonctionnalité spot

AWS Marketplace pour S3

Amazon S3 > Compartiments > rsp-oc-p8-fruits

rsp-oc-p8-fruits

Infos

Objets

Propriétés

Autorisations

Métriques

Gestion

Points d'accès

Objets (4)

Les objets sont les entités fondamentales stockées dans Amazon S3. Vous pouvez utiliser l'[Inventaire Amazon S3](#) pour obtenir une liste de tous les objets de votre compartiment. Pour que d'autres personnes puissent accéder à vos objets, vous devez leur accorder explicitement des autorisations. [En savoir plus](#)

Copier l'URI S3

Copier l'URL

Télécharger

Ouvrir





Supprimer

Actions

Créer un dossier

Charger

Rechercher des objets en fonction du préfixe

	Nom	Type	Dernière modification	Taille	Classe de stockage
<input type="checkbox"/>	 bootstrap-emr.sh	sh	27 Aug 2023 06:54:22 PM CEST	340.0 o	Standard
<input type="checkbox"/>	 jupyter/	Dossier	-	-	-
<input type="checkbox"/>	 Results/	Dossier	-	-	-
<input type="checkbox"/>	 Test/	Dossier	-	-	-

P8 - Déployez un modèle dans le cloud

Starting Spark Session

```
In [1]: # L'exécution de cette cellule démarre l'application Spark
```

Starting Spark application

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1693158029100_0001	pyspark	idle	Link	Link	✓

SparkSession available as 'spark'.

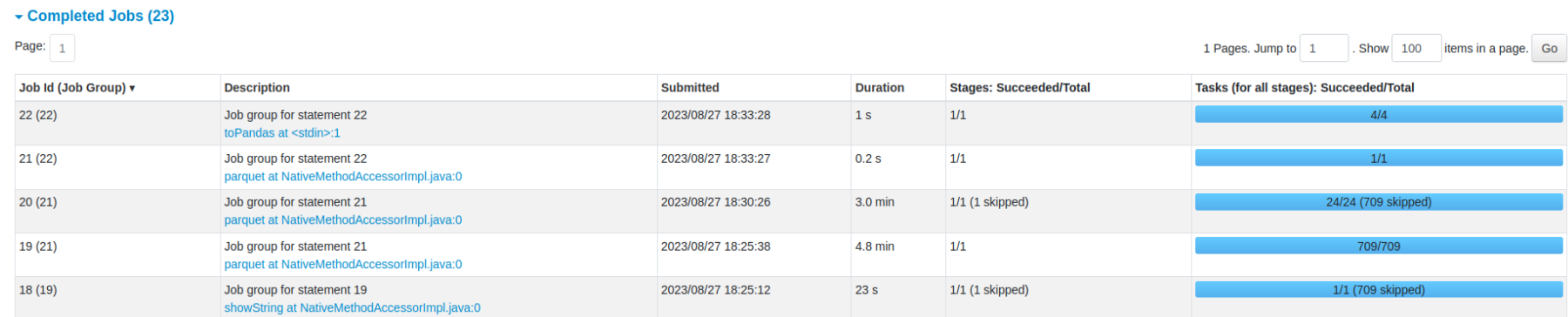
We create the **SparkContext** under the variable "sc" :

Let's display information about the spark session :

```
In [2]: %%info
```

Current session configs: {'driverMemory': '1000M', 'executorCores': 2, 'proxyUser': 'jovyan', 'kind': 'pyspark'}

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1693158029100_0001	pyspark	idle	Link	Link	✓



```
raquelsp@raquelsp: ~  
raquelsp@raquelsp:~$ aws s3 ls s3://rsp-oc-p8-fruits/Results/  
2023-08-26 15:29:12 0 _SUCCESS  
2023-08-26 15:26:44 983807 part-00000-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:26:44 989946 part-00001-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:26:44 981719 part-00002-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:26:44 978612 part-00003-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:27:14 977709 part-00004-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:27:13 981995 part-00005-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:27:12 985869 part-00006-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:27:16 978525 part-00007-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:27:40 976606 part-00008-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:27:40 977645 part-00009-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:27:48 985908 part-00010-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:27:45 985930 part-00011-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:28:09 979726 part-00012-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:28:09 978618 part-00013-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:28:12 980692 part-00014-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:28:16 988048 part-00015-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:28:39 985989 part-00016-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:28:38 978674 part-00017-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:28:40 978878 part-00018-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:28:44 981675 part-00019-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:29:07 988915 part-00020-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:29:09 985865 part-00021-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:29:08 979850 part-00022-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
2023-08-26 15:29:12 979630 part-00023-d5cc9283-6674-4229-9e60-b0a2e9a66ed8-c000.snappy.parquet  
raquelsp@raquelsp:~$
```



Contexte



Environnement Big Data



Traitement images



Architecture de développement



Conclusions



Annexes

- ✓ Premier approche **traitement des images révisé**
- ✓ Implémentation réduction des dimensions via **PCA** (de 1280 à 128)
- ✓ **Chaîne de traitement des images validée**
- ✓ Architecture Big Data : **EMR, S3, IAM**
 - ✓ 15 minutes pour instanciation des clusters
 - ✓ Traitement des images rapide
 - ✓ Les coûts évoluent en parallèle au volume de données
- ✓ Les briques mises en place peuvent évoluer en fonction du volume de données à traiter en augmentant le nombre de workers et / ou leur configuration



Contexte



Environnement Big Data



Traitement images



Architecture de développement



Conclusions



Annexes

AWS

EC2 : Elastic Compute Cloud

Ce service permet de gérer des serveurs sous forme de machines virtuelles dans le cloud. En gros, vous pouvez lancer des serveurs et faire ce que vous voulez avec. Vous avez accès à la ligne de commande, donc vous pouvez les piloter à distance.



Amazon Elastic Compute Cloud (Amazon EC2)

AWS

S3 : Simple Storage Service

Amazon S3 (Simple Storage Service) est un service de stockage et de distribution de fichiers. C'est une sorte d'entrepôt de fichiers à très bas coût qui garantit de ne jamais perdre vos données. Utilisez-le pour faire télécharger des fichiers sur votre site, ou pour y stocker des images.



Amazon Simple Storage Service (Amazon S3)

AWS IAM : Gestion des identités

AWS IAM (Identity and Access Management) est LE service de sécurité par excellence. On y définit les règles d'accès des utilisateurs aux services de la galaxie AWS. Si vous souhaitez autoriser votre comptable à télécharger la facture mais pas à éteindre vos serveurs, c'est là que ça se passe.

Service de sécurité
Gestion des accès



AWS Identity and Access Management (IAM)

Les étapes suivantes ont été implémentées :

1. Création d'une **SparkSession** (driver process) et la variable **sparkContext**

Local

```
1 spark = (SparkSession
2     .builder
3     .appName('P8')
4     .master('local')
5     .config("spark.sql.parquet.writeLegacyFormat", 'true')
6     .getOrCreate()
7 )
```

...

Nous créons également la variable "sc" qui est un **SparkContext** issue de la variable **spark** :

```
1 sc = spark.sparkContext
```

Affichage des informations de Spark en cours d'exécution :

```
1 spark
```

SparkSession - in-memory

SparkContext

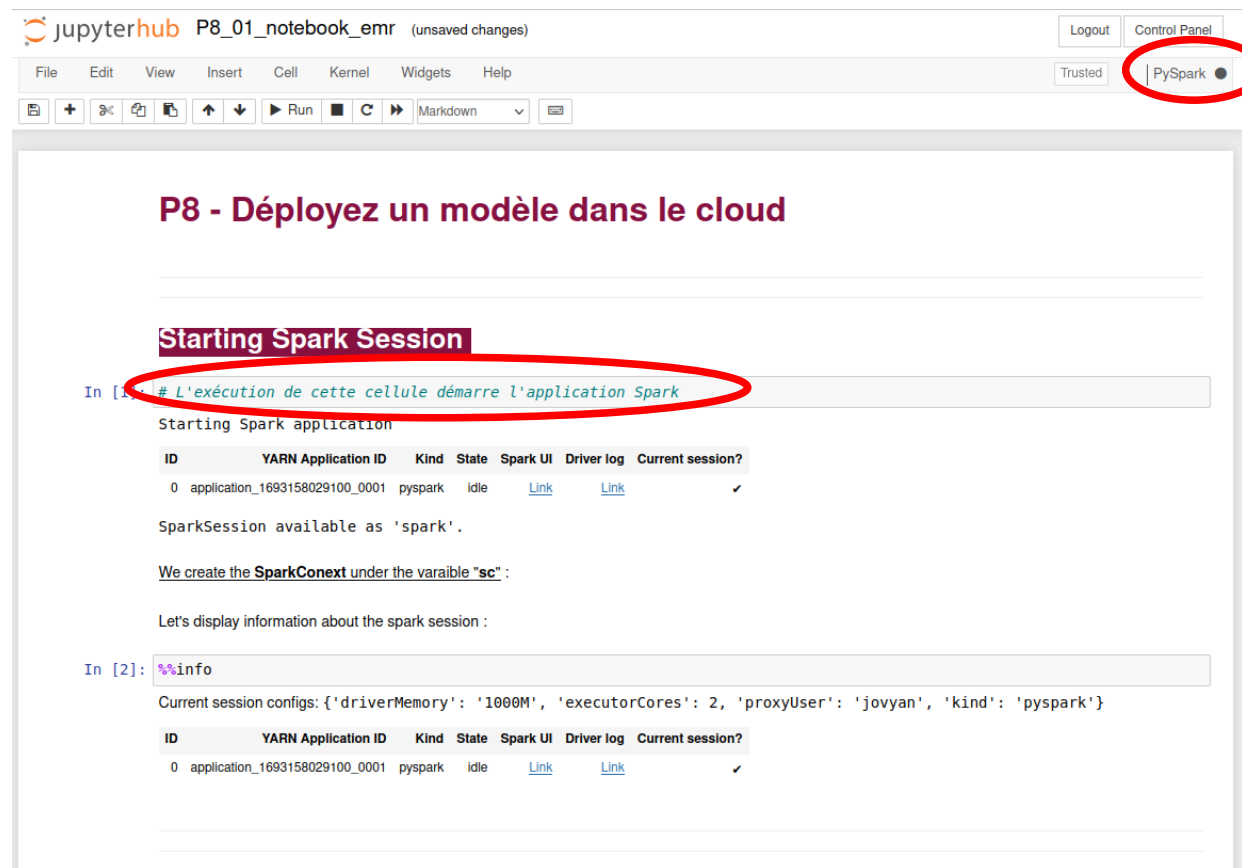
[Spark UI](#)

Version	v3.2.1
Master	local
AppName	P8

Les étapes suivantes ont été implémentées :

1. Création d'une **SparkSession** (driver process) et la variable **sparkContext**

EMR



jupyterhub P8_01_notebook_emr (unsaved changes) Logout Control Panel

File Edit View Insert Cell Kernel Widgets Help Trusted PySpark

P8 - Déployez un modèle dans le cloud

Starting Spark Session

In [1]: `# L'exécution de cette cellule démarre l'application Spark`

Starting Spark application

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1693158029100_0001	pyspark	idle	Link	Link	✓

SparkSession available as 'spark'.

We create the **SparkContext** under the variable "sc" :

Let's display information about the spark session :

In [2]: `%info`

Current session configs: {'driverMemory': '1000M', 'executorCores': 2, 'proxyUser': 'jovyan', 'kind': 'pyspark'}

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1693158029100_0001	pyspark	idle	Link	Link	✓

2. **Charger** les images et les associer aux images leur **label**

Exploration des données

Nous accédons directement à nos données sur S3 comme si elles étaient stockées localement.

In [4]: *# Define the folder containing the files with the project data*

```
s3_P8 = "s3://rsp-oc-p8-fruits/"
PATH_DataTest = s3_P8+'Test/'
PATH_Result = s3_P8 + "Results/"
```

```
print('PATH Project:      '+\
      s3_P8+'\n\nPATH_DataTest:    '+\
      PATH_DataTest+'\n\nPATH_Result:  '+\
      PATH_Result)
```

```
PATH Project:      s3://rsp-oc-p8-fruits/
```

```
PATH_DataTest:    s3://rsp-oc-p8-fruits/Test/
```

```
PATH_Result:      s3://rsp-oc-p8-fruits/Results/
```

2. Charger les images et les associer aux images leur label

Traitement des données

Chargement des données

- Les images sont chargées au format binaire, ce qui offre, plus de souplesse dans la façon de prétraiter les images.
- Seulement les fichiers dont l'extension est **jpg** seront chargés.
- Les fichiers contenus dans les sous-dossiers du dossier communiqué seront également chargés.

```
In [5]: images = spark.read.format("binaryFile") \
        .option("pathGlobFilter", "*.jpg") \
        .option("recursiveFileLookup", "true") \
        .load(PATH_DataTest)
```

Affichage des 5 premières images contenant :

- le path de l'image
- la date et heure de sa dernière modification
- sa longueur
- son contenu encodé en valeur hexadécimal

```
In [6]: images.show(5)
```

```
+-----+-----+-----+-----+
|          path|  modificationTime|length|          content|
+-----+-----+-----+-----+
|s3://rsp-oc-p8-fr...|2023-08-25 07:53:22| 7353|[FF D8 FF E0 00 1...|
|s3://rsp-oc-p8-fr...|2023-08-25 07:53:22| 7350|[FF D8 FF E0 00 1...|
|s3://rsp-oc-p8-fr...|2023-08-25 07:53:22| 7349|[FF D8 FF E0 00 1...|
|s3://rsp-oc-p8-fr...|2023-08-25 07:53:22| 7348|[FF D8 FF E0 00 1...|
|s3://rsp-oc-p8-fr...|2023-08-25 07:53:22| 7328|[FF D8 FF E0 00 1...|
+-----+-----+-----+-----+
```

only showing top 5 rows

2. Charger les images et les associer aux images leur label

Seulement le **path** de l'image est conservé, une colonne contenant les **labels** de chaque image est ajouté :

```
In [7]: images = images.withColumn('label', element_at(split(images['path'], '/'), -2))
print(images.printSchema())
print(images.select('path', 'label').show(5, False))
```

```
root
|-- path: string (nullable = true)
|-- modificationTime: timestamp (nullable = true)
|-- length: long (nullable = true)
|-- content: binary (nullable = true)
|-- label: string (nullable = true)
```

None

path	label
s3://rsp-oc-p8-fruits/Test/Watermelon/r_106_100.jpg	Watermelon
s3://rsp-oc-p8-fruits/Test/Watermelon/r_109_100.jpg	Watermelon
s3://rsp-oc-p8-fruits/Test/Watermelon/r_108_100.jpg	Watermelon
s3://rsp-oc-p8-fruits/Test/Watermelon/r_107_100.jpg	Watermelon
s3://rsp-oc-p8-fruits/Test/Watermelon/r_95_100.jpg	Watermelon

only showing top 5 rows

None

3. Préparation du modèle. Importer le modèle **MobileNetV2**

Créer un **nouveau modèle dépourvu de la dernière couche** de MobileNetV2

Broadcast des “weights” du modèle

Préparation du modèle

On a décidé de travailler avec du transfert learning. Le transfert learning consiste à utiliser la connaissance déjà acquise par un modèle entraîné (ici **MobileNetV2**) pour l'adapter à notre problématique. Nous allons fournir au modèle nos images, et nous allons récupérer l'avant dernière couche du modèle. Cela permettra de réaliser une première version du moteur pour la classification des images des fruits.

```
In [8]: model = MobileNetV2(weights='imagenet',
                             include_top=True,
                             input_shape=(224, 224, 3))
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224.h5
14540800/14536120 [=====] - 1s 0us/step

```
In [9]: new_model = Model(inputs=model.input,
                          outputs=model.layers[-2].output)
```

```
In [10]: broadcast_weights = sc.broadcast(new_model.get_weights())
```

```
In [11]: new_model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
=====			
input_1 (InputLayer)	[(None, 224, 224, 3)]	0	
Conv1 (Conv2D)	(None, 112, 112, 32)	864	input_1[0][0]
bn_Conv1 (BatchNormalization)	(None, 112, 112, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 112, 112, 32)	0	bn_Conv1[0][0]
expanded_conv_depthwise (Depthwise Conv2D)	(None, 112, 112, 32)	288	Conv1_relu[0][0]
expanded_conv_depthwise_BN (Batch Normalization)	(None, 112, 112, 32)	128	expanded_conv_depthwise[0][0]
expanded_conv_depthwise_relu (ReLU)	(None, 112, 112, 32)	0	expanded_conv_depthwise_BN[0][0]
expanded_conv_project (Conv2D)	(None, 112, 112, 16)	512	expanded_conv_depthwise_relu[0][0]

3. Préparation du modèle. Importer le modèle **MobileNetV2**

Créer un **nouveau modèle dépourvu de la dernière couche** de MobileNetV2

Broadcast des “weights” du modèle

Préparation du modèle

On a décidé de travailler avec du transfert learning. Le transfert learning consiste à utiliser la connaissance déjà acquise par un modèle entraîné (ici **MobileNetV2**) pour l'adapter à notre problématique. Nous allons fournir au modèle nos images, et nous allons récupérer l'avant dernière couche du modèle. Cela permettra de réaliser une première version du moteur pour la classification des images des fruits.

```
In [8]: model = MobileNetV2(weights='imagenet',
                             include_top=True,
                             input_shape=(224, 224, 3))

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224.h5
14540800/14536120 [=====] - 1s 0us/step

In [9]: new_model = Model(inputs=model.input,
                          outputs=model.layers[-2].output)

In [10]: broadcast_weights = sc.broadcast(new_model.get_weights())

In [11]: new_model.summary()
```

block_16_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_16_depthwise_BN[0][0]
block_16_project (Conv2D)	(None, 7, 7, 320)	307200	block_16_depthwise_relu[0][0]
block_16_project_BN (BatchNormal	(None, 7, 7, 320)	1280	block_16_project[0][0]
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409600	block_16_project_BN[0][0]
Conv_1_bn (BatchNormalization)	(None, 7, 7, 1280)	5120	Conv_1[0][0]
out_relu (ReLU)	(None, 7, 7, 1280)	0	Conv_1_bn[0][0]
global_average_pooling2d (Globa	(None, 1280)	0	out_relu[0][0]

```

Total params: 2,257,984
Trainable params: 2,223,872
Non-trainable params: 34,112

```


5. Réduction des dimensions via un PCA

Recherche nombre optimal composantes expliquant 95 % variance

Réduction des dimensions via PCA

Préparation des données

```
In [18]: # First step is to convert our features arrays into vectors
array_to_vector_udf = udf(lambda l: Vectors.dense(l), VectorUDT())
df_with_vectors = features_df.select(
    features_df["path"],
    features_df["label"],
    features_df["features"],
    array_to_vector_udf(features_df["features"]).alias("vectorFeatures"),
)
# show the 5 first values :
df_with_vectors.show(5)
```

path	label	features	vectorFeatures
s3://rsp-oc-p8-fr...	Watermelon	[0.46832162, 0.22...	[0.46832162141799...
s3://rsp-oc-p8-fr...	Pineapple Mini	[0.0, 4.7068405, ...	[0.0,4.7068405151...
s3://rsp-oc-p8-fr...	Watermelon	[0.051970564, 0.1...	[0.05197056382894...
s3://rsp-oc-p8-fr...	Watermelon	[0.11681207, 0.13...	[0.11681207269430...
s3://rsp-oc-p8-fr...	Cauliflower	[0.0, 0.4670273, ...	[0.0,0.4670273065...

only showing top 5 rows

```
In [19]: # Second step is to scale the data before applying the PCA process
scaler = StandardScaler(inputCol="vectorFeatures",
    outputCol="scaledFeatures",
    withMean=True, withStd=True
).fit(df_with_vectors)

# when we transform the dataframe, the old feature will still remain in it
df_scaled = scaler.transform(df_with_vectors)
# show the 5 first values :
df_scaled.show(5)
```

path	label	features	vectorFeatures	scaledFeatures
s3://rsp-oc-p8-fr...	Watermelon	[0.12482018, 0.02...	[0.12482018023729...	[1.0, 5.261020160438...

5. Réduction des dimensions via un PCA

Recherche nombre optimal composantes expliquant 95 % variance

Réduction de dimensions via PCA

```
In [20]: # We will use the optimal number of components for explaining 95% of variance
# identified during the local development
n_components = 128

# Apply the PCA with n components
pca = PCA(k=n_components, inputCol='scaledFeatures', outputCol='pcaFeatures')

model_pca = pca.fit(df_scaled)
df_pca = model_pca.transform(df_scaled)

# show the 5 first values :
df_pca.show(5)
```

path	label	features	vectorFeatures	scaledFeatures	pcaFeatures
s3://rsp-oc-p8-fr...	Watermelon	[0.46832162, 0.22...	[0.46832162141799...	[0.11042147660729...	[-9.435672178619...
s3://rsp-oc-p8-fr...	Pineapple Mini	[0.0, 4.7068405, ...]	[0.0, 4.7068405151...	[-0.7574006047611...	[-13.55920450180...
s3://rsp-oc-p8-fr...	Watermelon	[0.036369544, 0.0...	[0.03636954352259...	[-0.6900061230316...	[-6.144324392191...
s3://rsp-oc-p8-fr...	Watermelon	[0.013230772, 0.1...	[0.01323077175766...	[-0.7328833598599...	[-11.12438625655...
s3://rsp-oc-p8-fr...	Cauliflower	[0.0, 0.4670273, ...]	[0.0, 0.4670273065...	[-0.7574006047611...	[-6.785150361606...

only showing top 5 rows