databrickstest-s3-access

Prétraitement des Images de Fruits

L'entreprise *Fruits!* souhaite créer un modèle de reconnaissance de fruits. Il met à disposition un jeu de données d'images (https://www.kaggle.com/moltean/fruits) de fruits pour entraîner un modèle. On effectue dans ce notebook l'étape de prétraitement des images en aval de l'entraînement du modèle. Cette étape se décompose en deux parties :

- une featurisation des images,
- une réduction de dimensions de l'espace de plongement.

La featurisation des images est effectué avec le réseau de neuronnes convolutif (CNN) pré-entrainé Resnet50, développé sur TensorFlow. La réduction des images est effectué avec l'algorithme PCA.

Le code est écrit en Pyspark et est exécuté sur DataBricks. On utilse les ressources AWS EC2 et S3 pour effectuer cette tâche.

```
# synchronisation S3 et DBFS
#mount_name = "fruit-360"
#mount_name = "fruit-360/apple_6"
#mount_name = "fruit-360/selection"
mount_name = "fruit-360/test-compute"
display(dbutils.fs.ls("/mnt/%s" % mount_name))
```

	path	name	size
1	dbfs:/mnt/fruit-360/test-compute/apple_6/	apple_6/	0
2	dbfs:/mnt/fruit-360/test-compute/apple_braeburn_1/	apple_braeburn_1/	0
3	dbfs:/mnt/fruit-360/test-compute/apple_crimson_snow_1/	apple_crimson_snow_1/	0

4	dbfs:/mnt/fruit-360/test-compute/apple_golden_1/	apple_golden_1/	0
5	dbfs:/mnt/fruit-360/test-compute/apple_golden_2/	apple_golden_2/	0
6	dbfs:/mnt/fruit-360/test-compute/apple_golden_3/	apple_golden_3/	0
7	dbfs:/mnt/fruit-360/test-compute/apple_granny_smith_1/	apple_granny_smith_1/	0
8	dbfs:/mnt/fruit-360/test-compute/apple_hit_1/	apple_hit_1/	0
9	dbfs:/mnt/fruit-360/test-compute/apple_pink_lady_1/	apple_pink_lady_1/	0
10	dbfs:/mnt/fruit-360/test-compute/apple_red_1/	apple_red_1/	0
11	dbfs:/mnt/fruit-360/test-compute/apple_red_2/	apple_red_2/	0
12	dbfs:/mnt/fruit-360/test-compute/apple_red_3/	apple_red_3/	0
13	dbfs:/mnt/fruit-360/test-compute/apple_red_delicios_1/	apple_red_delicios_1/	0
14	dbfs:/mnt/fruit-360/test-compute/apple_red_yellow_1/	apple_red_yellow_1/	0
15	dbfs:/mnt/fruit-360/test-compute/apple_rotten_1/	apple_rotten_1/	0
16	dbfs:/mnt/fruit-360/test-compute/cabbage_white_1/	cabbage_white_1/	0
17	dbfs:/mnt/fruit-360/test-compute/carrot_1/	carrot_1/	0
18	dbfs:/mnt/fruit-360/test-compute/cucumber_1/	cucumber_1/	0
19	dbfs:/mnt/fruit-360/test-compute/cucumber_3/	cucumber_3/	0
20	dbfs:/mnt/fruit-360/test-compute/eggplant_violet_1/	eggplant_violet_1/	0
21	dbfs:/mnt/fruit-360/test-compute/pear_3/	pear_3/	0
22	dbfs:/mnt/fruit-360/test-compute/zucchini_1/	zucchini_1/	0
23	dbfs:/mnt/fruit-360/test-compute/zucchini_dark_1/	zucchini_dark_1/	0

```
dbutils.fs.ls("/mnt/%s" % mount_name)
```

```
Out[2]: [FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_6/', name='apple_6/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_braeburn_1/', name='apple_braeburn_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_crimson_snow_1/', name='apple_crimson_snow_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_golden_1/', name='apple_golden_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_golden_2/', name='apple_golden_2/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_golden_3/', name='apple_golden_3/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_granny_smith_1/', name='apple_granny_smith_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_hit_1/', name='apple_hit_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_pink_lady_1/', name='apple_pink_lady_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_red_1/', name='apple_red_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_red_2/', name='apple_red_2/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_red_3/', name='apple_red_3/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_red_delicios_1/', name='apple_red_delicios_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_red_yellow_1/', name='apple_red_yellow_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/apple_rotten_1/', name='apple_rotten_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/cabbage_white_1/', name='cabbage_white_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/carrot_1/', name='carrot_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/cucumber_1/', name='cucumber_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/cucumber_3/', name='cucumber_3/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/eggplant_violet_1/', name='eggplant_violet_1/', size=0),
 FileInfo(path='dbfs:/mnt/fruit-360/test-compute/pear_3/', name='pear_3/', size=0),
```

```
# chargement des images de fruit
images = spark.read.format("binaryFile") \
  .option("pathGlobFilter", "*.jpg") \
  .option("recursiveFileLookup", "true") \
  .load("/mnt/%s" % mount_name)
```

```
display(images.limit(5))
```

	path	modificationTime	length	content

1	dbfs:/mnt/fruit-360/test-compute/apple_hit_1/r0_116.jpg	2022-01-03T20:03:00.000+0000	125373	
2	dbfs:/mnt/fruit-360/test-compute/apple_hit_1/r0_114.jpg	2022-01-03T20:03:00.000+0000	125088	
3	dbfs:/mnt/fruit-360/test-compute/apple_hit_1/r0_108.jpg	2022-01-03T20:03:00.000+0000	124905	
4	dbfs:/mnt/fruit-360/test-compute/apple_hit_1/r0_118.jpg	2022-01-03T20:03:00.000+0000	124363	

Showing all 5 rows.

Show image preview

```
import pandas as pd
from PIL import Image
import numpy as np
import io

import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
#from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
#from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2, preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array

from pyspark.sql.functions import col, split, pandas_udf, PandasUDFType

from pyspark.ml.functions import vector_to_array
from pyspark.ml.feature import StringIndexer, StandardScaler
from pyspark.ml.linalg import Vectors, VectorUDT
from pyspark.ml.feature import PCA

import seaborn as sns
import matplotlib.pyplot as plt
```

```
images = images.withColumn('label', split(col('path'), '/').getItem(4))
images = images.select('path', 'content', 'label')
images.show()
```

```
+--------------------+--------------------+-----------+
|                path|             content|      label|
+--------------------+--------------------+-----------+
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
```

```
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
|dbfs:/mnt/fruit-3...|[FF D8 FF E0 00 1...|apple_hit_1|
```

```
# chargement du CNN pré-entraîné
model = ResNet50(include_top=False)
#model = VGG16(include_top=False)
#model = InceptionResNetV2(include_top=False)
model.summary()  # verify that the top layer is removed
```

```
Model: "resnet50"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, None, None,  0

__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, None, None, 3 0           input_1[0][0]
__________________________________________________________________________________________________
conv1_conv (Conv2D)             (None, None, None, 6 9472        conv1_pad[0][0]
__________________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, None, None, 6 256         conv1_conv[0][0]
__________________________________________________________________________________________________
```

```
conv1_relu (Activation)         (None, None, None, 6 0           conv1_bn[0][0]
__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, None, None, 6 0           conv1_relu[0][0]
__________________________________________________________________________________________________
pool1_pool (MaxPooling2D)       (None, None, None, 6 0           pool1_pad[0][0]
__________________________________________________________________________________________________
conv2_block1_1_conv (Conv2D)    (None, None, None, 6 4160        pool1_pool[0][0]
__________________________________________________________________________________________________
```

```
bc_model_weights = sc.broadcast(model.get_weights())

def model_fn():
  """
  Returns a ResNet50 model with top layer removed and broadcasted pretrained weights.
  """
  model = ResNet50(weights=None, include_top=False)
  #model = VGG16(weights=None, include_top=False)
  #model = InceptionResNetV2(weights=None, include_top=False)
  model.set_weights(bc_model_weights.value)
  return model
```

```
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)

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)
```

```
/databricks/spark/python/pyspark/sql/pandas/functions.py:386: UserWarning: In Python 3.6+ and Spark 3.0+, it is prefer
red to specify type hints for pandas UDF instead of specifying pandas UDF type which will be deprecated in the future
releases. See SPARK-28264 for more details.
  warnings.warn(
```

```
# étape de featurisation
#features_df = images.repartition(16).select(col("path"), col("label"), featurize_udf("content").alias("features"))
features_df = images.select(col("path"), col("label"), featurize_udf("content").alias("features"))
features_df.show()
```

```
+--------------------+-----------+--------------------+
|                path|      label|            features|
+--------------------+-----------+--------------------+
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
```

```
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
|dbfs:/mnt/fruit-3...|apple_hit_1|[0.0, 0.0, 0.0, 0...|
```

```
list_to_vector_udf = udf(lambda l: Vectors.dense(l), VectorUDT())
features_df = features_df.select(col("path"), col("label"),
list_to_vector_udf(features_df["features"]).alias("features"))
```

```
# normalisation des variables (dimensions de l'espace de plongement)
standardizer = StandardScaler(withMean=True, withStd=True,
                              inputCol='features',
                              outputCol='feats_scaled')
std = standardizer.fit(features_df)
features_df_scaled = std.transform(features_df)
```

```
features_df_scaled_tr = features_df_scaled.sample(fraction = 0.1)
```

```
features_df_scaled_tr.count()
```

```
Out[15]: 626
```

```
#features_df_scaled_tr.show()
```

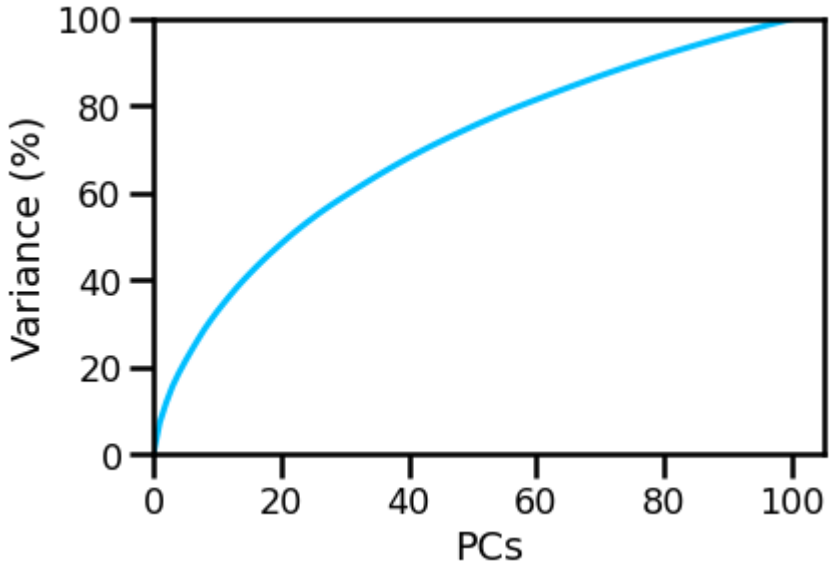
```
n_pc = 100
# entraînement et application d'une PCA
pca = PCA(k=n_pc, inputCol="feats_scaled", outputCol="pca")
#modelpca = pca.fit(features_df_scaled)
modelpca = pca.fit(features_df_scaled_tr)
```

```
transformed = modelpca.transform(features_df_scaled)
```

```
features_df_scaled.count()
```

```
Out[19]: 5905
```

```
# plot du % de variance expliquée en fonction du nombre de PC
var = modelpca.explainedVariance.cumsum()
sns.set_context(context='poster', font_scale=0.8)
sns.lineplot(x=[i for i in range(n_pc + 1)], y=np.insert(var,0,0)*100, color='deepskyblue')
plt.xlabel('PCs')
plt.ylabel('Variance (%)')
plt.ylim(0,100)
plt.xlim(left=0)
plt.show()
```




```
exvar = modelpca.explainedVariance

ipc_cut = 0
threshold = 0.01

for elt in exvar:
    #print(elt)
    if elt < threshold:
        print(elt)
        ipc_cut = np.where(exvar.values == elt)
        break
```

```
0.009653678107931104
```

```
ipc_cut
```

```
Out[37]: (array([28]),)


# sélection des path, label et premieres pc
final = transformed.withColumn("pca", vector_to_array("pca")).select(["path", "label"] + [col("pca")[i] for i in
range(ipc_cut[0][0])])

# écriture des fichiers dans le S3
#final.write.mode("overwrite").parquet("/mnt/reduction.parquet")



# écriture des fichiers dans le S3
#final.write.mode("overwrite").parquet("/mnt/%s/reduction.parquet" % mount_name)
final.write.mode("overwrite").parquet("reduction.parquet")



# conversion dataframe pandas pour visualisation
transformed_pandas = final.toPandas()
transformed_pandas.head()

Out[39]:
```

	path	label	pca[0]	pca[1]	pca[2]	pca[3]	pca[4]	pca[5]	pca[6]	pca[7]	pca[8]	pca[9
0	dbfs:/mnt/fruit-360/test-compute/apple_hit_1/r...	apple_hit_1	-4.244428	-1.297873	-19.306000	1.027980	-12.189369	28.808328	33.051261	4.241512	65.718174	-20.34672
1	dbfs:/mnt/fruit-360/test-compute/apple_hit_1/r...	apple_hit_1	-7.018301	-0.920443	-18.315981	1.381388	-12.419158	26.816941	32.507030	2.377980	62.481883	-16.07186
2	dbfs:/mnt/fruit-360/test-compute/apple_hit_1/r...	apple_hit_1	-1.693223	-3.599726	-22.868818	-0.686607	-15.410438	26.981728	33.973652	1.065745	69.836426	-18.65049
3	dbfs:/mnt/fruit-360/test-compute/apple_hit_1/r...	apple_hit_1	-6.182198	-1.590132	-16.349514	1.237405	-11.328907	24.901114	30.628064	4.463446	58.449516	-15.32839
4	dbfs:/mnt/fruit-360/test-compute/apple_hit_1/r...	apple_hit_1	-4.675239	-3.264141	-18.014367	3.049207	-16.580817	28.431690	36.584868	11.506236	69.417906	-11.86546

```
Out[41]: (5905, 30)
```