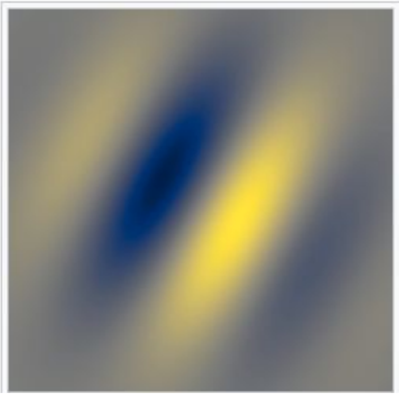
OBJECTIFS

Renforcer les connaissances générales en traitement des images (computer vision)

Balayer les principales bibliothèques de traitements d’image Python (OpenCV & SKIMAGE)
mieux appréhender la notion de kernel dans le traitement d’images

Comprendre l'influence des paramètres mathématiques sur la filtration des kernels et
donc la sélection de caractéristiques d’images (features)

Évaluer le potentiel d’utilisation de ces notions “un peu théoriques” en machine learning & deep learning, dans la cadre du projet RAKUTEN



Pour le traitement d'images et la vision par ordinateur, les filtres de Gabor sont généralement utilisé dans l'analyse de texture, la détection de contours, l'extraction de caractéristiques, etc. **Les filtres de Gabor sont des classes spéciales de filtres passe-bande**, c'est-à-dire qu'ils permettent le filtrage d'une certaine « bande » de fréquences et rejeter les autres.

Un filtre Gabor , nommé d'après Dennis Gabor , est un filtre linéaire utilisé pour l'analyse de texture , ce qui signifie essentiellement qu'il analyse s'il y a un contenu de fréquence spécifique dans l'image dans des directions spécifiques dans une région localisée autour du point ou de la zone d'analyse. Les représentations de fréquence et d'orientation des filtres de Gabor sont revendiquées par de nombreux scientifiques contemporains de la vision comme étant similaires à celles du système visuel humain . Ils se sont avérés particulièrement appropriés pour la représentation et la discrimination des textures. Dans le domaine spatial, un filtre de Gabor 2-D est une fonction à noyau gaussien modulée par une onde plane sinusoïdale

$$g(x,y;\lambda,\theta,\psi,\sigma,\gamma) = \exp\left(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right)\exp\left(i\left(2\pi\frac{x'}{\lambda}+\psi\right)\right)$$

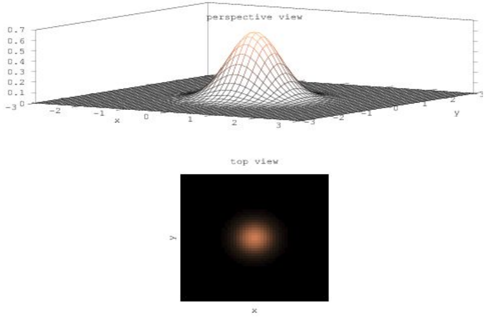


fig 3, 2d gaussian, σ_x=0.5,σ_y=0.5

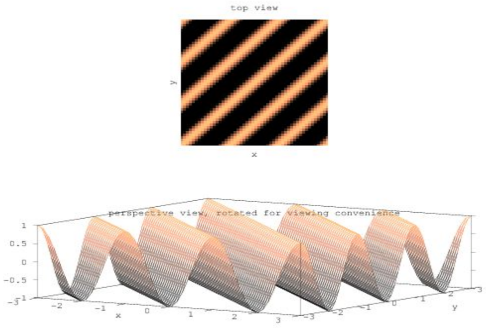


fig 4, 2d cos function, cos(2 π (x u_1 + y v_1)), u_1=0.5, v_1=0.5

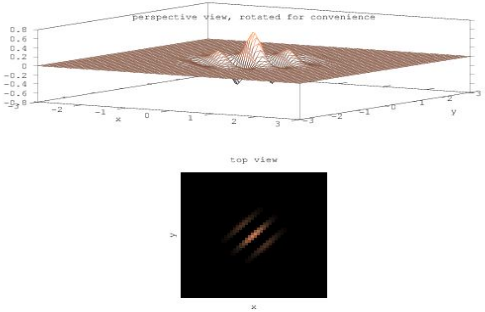


fig 5, 2d gaussian times cos function, σ_x=0.5,σ_y=0.5, u_1=1.0, v_1=1.0

http://kgeorge.github.io/2016/02/04/understanding-gabor-filter
https://en.wikipedia.org/wiki/Gabor_filter#cite_note-1

TRAITEMENT IMAGE - PRESENTATION DES FILTRES GABOR DANS LA BIBLIOTHEQUE OPEN CV

```
cv2.getGaborKernel(ksize, sigma, theta, lambda, gamma, psi, ktype)
```

-> **ksize** : Taille du filtre renvoyé

-> **sigma** : Écart-type de l'enveloppe gaussienne

-> **theta** : Orientation de la normale aux bandes // d'une fonction de Gabor

-> **lambda** : Longueur d'onde du facteur sinusoïdal

-> **gamma** : Aspect ratio spatial

-> **psi** : Décalage de phase

-> **ktype** Type de coefficients de filtre. Il peut s'agir de CV_32F ou CV_64F :indique le type et la plage de valeurs que chaque pixel du noyau Gabor peut contenir: float32 ou float64

◆ getGaborKernel()

```
Mat cv::getGaborKernel ( Size    ksize,
                         double  sigma,
                         double  theta,
                         double  lambd,
                         double  gamma,
                         double  psi = CV_PI *0.5 ,
                         int     ktype = CV_64F
                       )
```

Python:

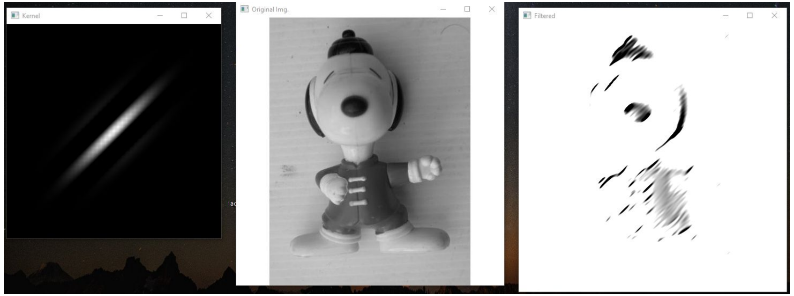
```
cv.getGaborKernel( ksize, sigma, theta, lambd, gamma[, psi[, ktype]] ) -> retval
```

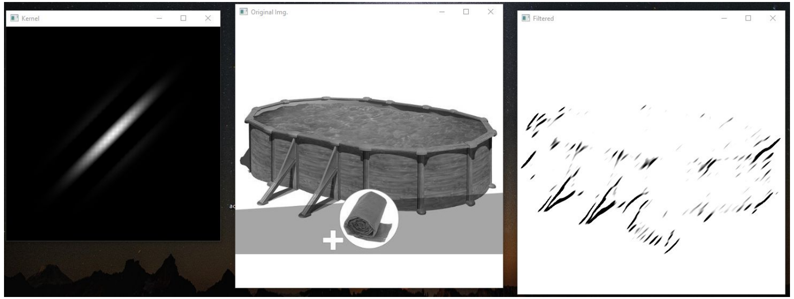
TESTS UTILISATION DES FILTRES GABOR OPEN CV - IMAGES RAKUTEN

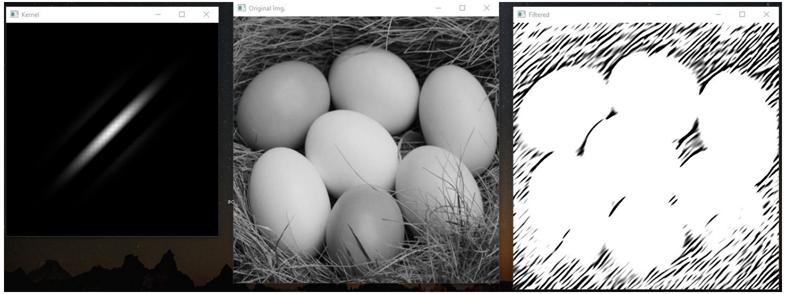
CODE ((ML0_img_rak.py)

Original Img RGB **Kernel** **Original Img Gray** **Filtered Img**

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import IPython.display as display

ksize = 50  #Use size that makes sense to the image and fetaure size. Large may not be good.
#On the synthetic image it is clear how ksize affects imgae (try 5 and 50)
sigma = 3 #Large sigma on small features will fully miss the features.
theta = 1*np.pi/4  #/4 shows horizontal 3/4 shows other horizontal. Try other contributions
lamda = 1*np.pi/4  #1/4 works best for angled.
gamma=0.4  #Value of 1 defines spherical. Calue close to 0 has high aspect ratio
#Value of 1, spherical may not be ideal as it picks up features from other regions.
phi = 0  #Phase offset. I leave it to 0.


kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lamda, gamma, phi, ktype=cv2.CV_64F)
plt.imshow(kernel)

img = cv2.imread('image_688575878_product_57497717.jpg')
#img = cv2.imread('BSE_Image.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
fimg = cv2.filter2D(img, cv2.CV_8UC3, kernel)

kernel_resized = cv2.resize(kernel, (400, 400))
cv2.imshow('Kernel', kernel_resized)
cv2.imshow('Original Img.', img)
cv2.imshow('Filtered', fimg)
cv2.waitKey(30000) #milliseconds
cv2.destroyAllWindows()
```







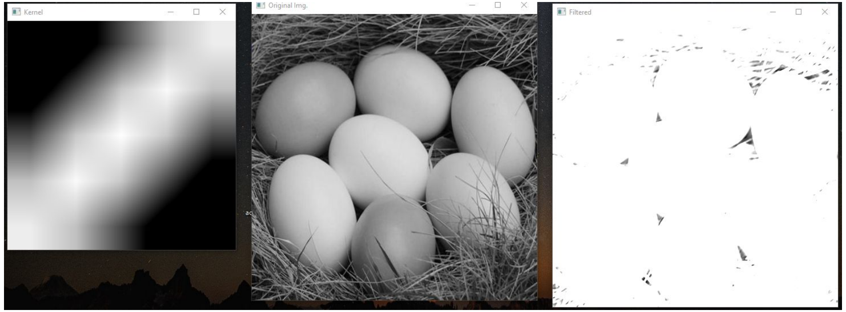
https://corpocrat.com/2015/03/25/applying-gabor-filter-on-faces-using-opencv/

TESTS UTILISATION DES FILTRES GABOR OPEN CV - IMAGES RAKUTEN

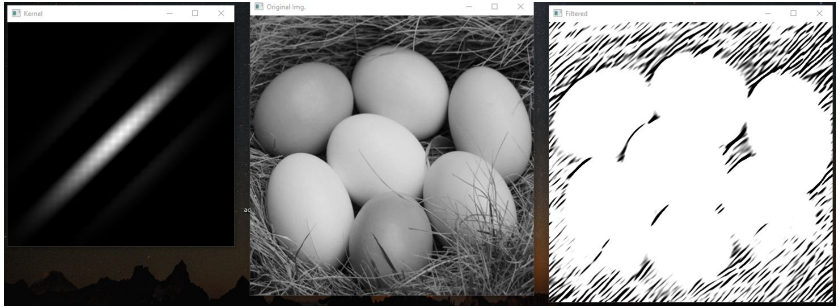
Original Img RGB | Kernel | Original Img Gray | Filtered Img

PARAMÈTRE MODIFIÉ

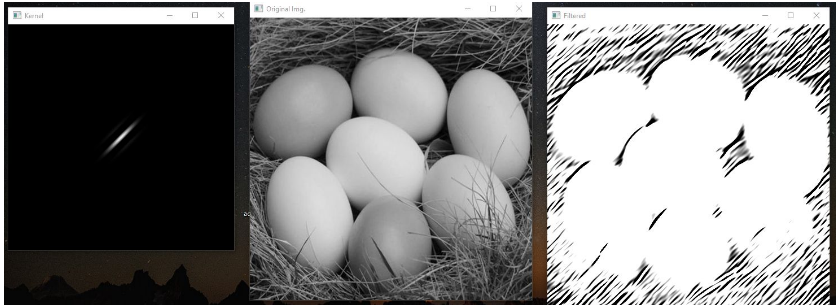
```
ksize = 5 #Use size that makes sense to the image and fetaure size. Large may not be good.
#On the synthetic image it is clear how ksize affects imgae (try 5 and 50)
sigma = 3 # Large sigma on small features will fully miss the features.
theta = 1*np.pi/4  # 1/4 shows horizontal
lamda = 1*np.pi/4  # 1/4 works good
gamma=0.4  #Value of 1 defines spherical. Close to 0 has high aspect ratio
phi = 0  #Phase offset
```



```
ksize = 30 #Use size that makes sense to the image and feature size. Large may not be good.
#On the synthetic image it is clear how ksize affects imgae (try 5 and 50)
sigma = 3 # Large sigma on small features will fully miss the features.
theta = 1*np.pi/4  # 1/4 shows horizontal
lamda = 1*np.pi/4  # 1/4 works good
gamma=0.4  #Value of 1 defines spherical. Close to 0 has high aspect ratio
phi = 0  #Phase offset
```



```
ksize = 150  #Use size that makes sense to the image and feature size. Large may not be good.
#On the synthetic image it is clear how ksize affects imgae (try 5 and 50)
sigma = 3 # Large sigma on small features will fully miss the features.
theta = 1*np.pi/4  # 1/4 shows horizontal
lamda = 1*np.pi/4  # 1/4 works good
gamma=0.4  #Value of 1 defines spherical. Close to 0 has high aspect ratio
phi = 0  #Phase offset
```



TESTS UTILISATION DES FILTRES GABOR OPEN CV - IMAGES RAKUTEN

PARAMÈTRE MODIFIÉ

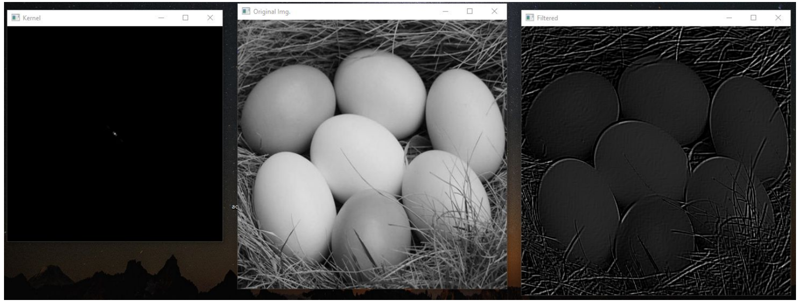
```
ksize = 150  #Use size that makes sense to the image and feature size. Large may not be good.
#On the synthetic image it is clear how ksize affects imgae (try 5 and 50)
sigma = 3 # Large sigma on small features will fully miss the features.
theta = 1*np.pi/4  # 1/4 shows horizontal
lamda = 1*np.pi/4  # 1/4 works good
gamma=10  #Value of 1 defines spherical. Close to 0 has high aspect ratio
phi = 0  #Phase offset
```

Original Img RGB

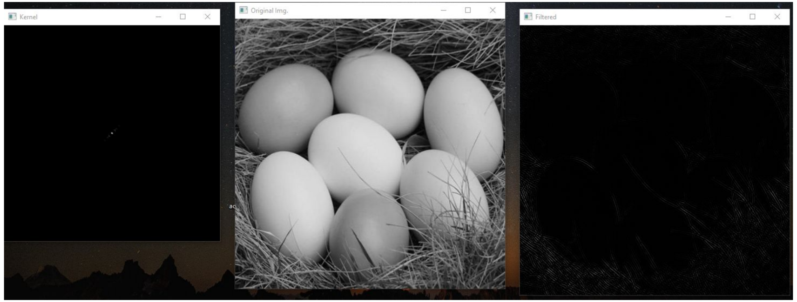
Kernel

Original Img Gray

Filtered Img



```
ksize = 150  #Use size that makes sense to the image and feature size. Large may not be good.
#On the synthetic image it is clear how ksize affects imgae (try 5 and 50)
sigma = 3 # Large sigma on small features will fully miss the features.
theta = 3*np.pi/4 # 1/4 shows horizontal
lamda = 3*np.pi/4 # 1/4 works good
gamma=10  #Value of 1 defines spherical. Close to 0 has high aspect ratio
phi = 0  #Phase offset
```

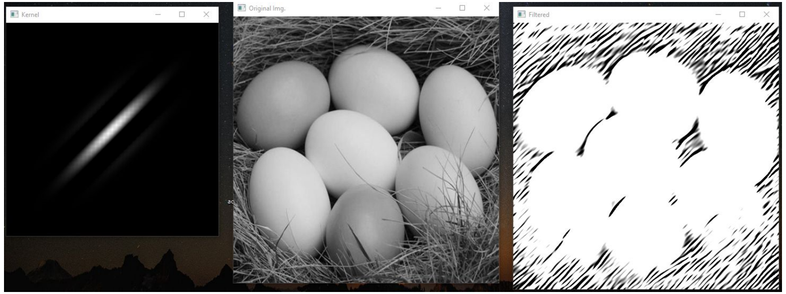


TESTS UTILISATION DES FILTRES GABOR OPEN CV - IMAGES RAKUTEN

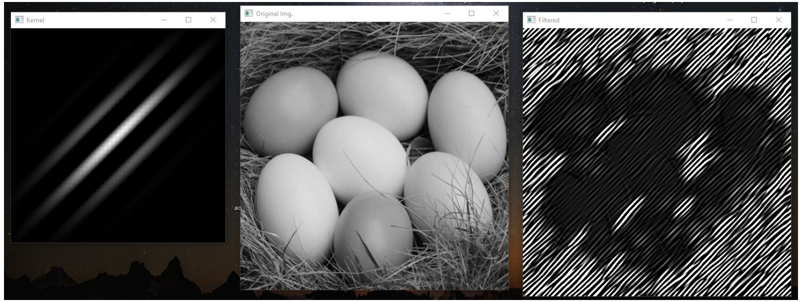
PARAMÈTRE MODIFIÉ

Original Img RGB **Kernel** **Original Img Gray** **Filtered Img**

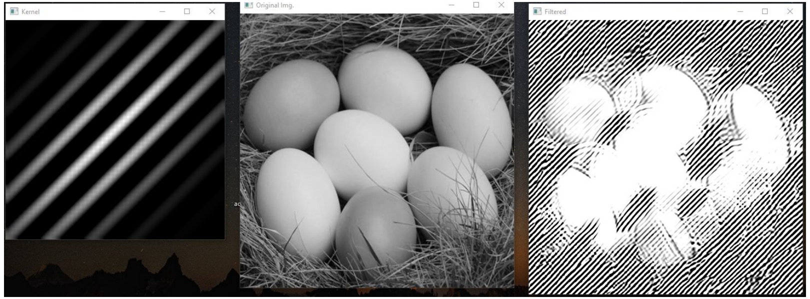
```
ksize = 50  #Use size that makes sense to the image and fetaure size. Large may not be good.
#On the synthetic image it is clear how ksize affects imgae (try 5 and 50)
sigma = 3 # Large sigma on small features will fully miss the features.
theta = 1*np.pi/4  # 1/4 shows horizontal
lamda = 1*np.pi/4  # 1/4 works good
gamma=0.4  #Value of 1 defines spherical. Close to 0 has high aspect ratio
phi = 0  #Phase offset
```



```
ksize = 50  #Use size that makes sense to the image and fetaure size. Large may not be good.
#On the synthetic image it is clear how ksize affects imgae (try 5 and 50)
sigma = 5 # Large sigma on small features will fully miss the features.
theta = 1*np.pi/4  # 1/4 shows horizontal
lamda = 1*np.pi/4  # 1/4 works good
gamma=0.4  #Value of 1 defines spherical. Close to 0 has high aspect ratio
phi = 0  #Phase offset
```



```
ksize = 50  #Use size that makes sense to the image and fetaure size. Large may not be good.
#On the synthetic image it is clear how ksize affects imgae (try 5 and 50)
sigma = 10 # Large sigma on small features will fully miss the features.
theta = 1*np.pi/4  # 1/4 shows horizontal
lamda = 1*np.pi/4  # 1/4 works good
gamma=0.4  #Value of 1 defines spherical. Close to 0 has high aspect ratio
phi = 0  #Phase offset
```



TRAITEMENT IMAGE - PRÉSENTATION DE QUELQUES FILTRES COMPLÉMENTAIRES A PARTIR DE OPENCV

◆ GaussianBlur()

```
void cv::GaussianBlur ( InputArray   src,
                        OutputArray  dst,
                        Size         ksize,
                        double       sigmaX,
                        double       sigmaY = 0 ,
                        int          borderType = BORDER_DEFAULT
                      )
```

Python:

```
cv.GaussianBlur( src, ksize, sigmaX[, dst[, sigmaY[, borderType]]] ) -> dst
```

`#include <opencv2/imgproc.hpp>`

Blurs an image using a Gaussian filter.

The function convolves the source image with the specified Gaussian kernel. In-place filtering is supported.

◆ Scharr()

```
void cv::Scharr ( InputArray   src,
                  OutputArray  dst,
                  int          ddepth,
                  int          dx,
                  int          dy,
                  double       scale = 1 ,
                  double       delta = 0 ,
                  int          borderType = BORDER_DEFAULT
                )
```

Python:

```
cv.Scharr( src, ddepth, dx, dy[, dst[, scale[, delta[, borderType]]]] ) -> dst
```

`#include <opencv2/imgproc.hpp>`

Calculates the first x- or y- image derivative using Scharr operator.

◆ Sobel()

```
void cv::Sobel ( InputArray   src,
                 OutputArray  dst,
                 int          ddepth,
                 int          dx,
                 int          dy,
                 int          ksize = 3 ,
                 double       scale = 1 ,
                 double       delta = 0 ,
                 int          borderType = BORDER_DEFAULT
               )
```

Python:

```
cv.Sobel( src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]] ) -> dst
```

`#include <opencv2/imgproc.hpp>`

Calculates the first, second, third, or mixed image derivatives using an extended Sobel operator.

In all cases except one, the $\mathbf{ksize} \times \mathbf{ksize}$ separable kernel is used to calculate the derivative. When $\mathbf{ksize} = 1$, the 3×1 or 1×3 kernel is used (that is, no Gaussian smoothing is done). `ksize = 1` can only be used for the first or the second x- or y- derivatives.

There is also the special value `ksize = CV_SCHARR (-1)` that corresponds to the 3×3 Scharr filter that may give more accurate results than the 3×3 Sobel. The Scharr aperture is

$$\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

for the x-derivative, or transposed for the y-derivative.

◆ getStructuringElement()

```
Mat cv::getStructuringElement ( int    shape,
                                Size   ksize,
                                Point  anchor = Point(-1,-1)
                              )
```

Python:

```
cv.getStructuringElement( shape, ksize[, anchor] ) -> retval
```

`#include <opencv2/imgproc.hpp>`

Returns a structuring element of the specified size and shape for morphological operations.

The function constructs and returns the structuring element that can be further passed to **erode**, **dilate** or **morphologyEx**. But you can also construct an arbitrary binary mask yourself and use it as the structuring element.

https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html

TRAITEMENT IMAGE - PRÉSENTATION DE QUELQUES FILTRES COMPLÉMENTAIRES A PARTIR DE SKIMAGE

roberts

`skimage.filters.roberts(image, mask=None)`

Find the edge magnitude using Roberts' cross operator.

prewitt_v

`skimage.filters.prewitt_v(image, mask=None)`

Find the vertical edges of an image using the Prewitt transform.

gaussian

`skimage.filters.gaussian(image, sigma=1, output=None, mode='nearest', cval=0, multichannel=None, preserve_range=False, truncate=4.0)` [source]

Multi-dimensional Gaussian filter.

scharr

`skimage.filters.scharr(image, mask=None, *, axis=None, mode='reflect', cval=0.0)`

Find the edge magnitude using the Scharr transform.

sobel

`skimage.filters.sobel(image, mask=None, *, axis=None, mode='reflect', cval=0.0)`

Find edges in an image using the Sobel filter.

https://scikit-image.org/docs/stable/api/skimage.filters.html?highlight=roberts#skimage.filters.roberts

TESTS UTILISATION DE PLUSIEURS FILTRES COMBINES POUR UN 1er MODÈLE DE MACHINE LEARNING - IMAGES RAKUTEN

STEP 1 -> création d'un dataframe des features d'une image (ML3_img_rak.py)

```
import numpy as np
import cv2
import pandas as pd
from skimage.filters import roberts, sobel, scharr, prewitt
from scipy import ndimage as nd

img = cv2.imread('image_688575878_product_57497717.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#### Multiple images can be used for training. For that, you need to concatenate the data
img2 = img.reshape(-1)
df = pd.DataFrame()
df['Original Image'] = img2
#Generate Gabor features
num = 1  #To count numbers up in order to give Gabor features a label in dataframe
kernels = []
for theta in range(2):   #Define number of thetas
    theta = theta / 4. * np.pi
    for sigma in (1, 3):  #Sigma with 1 and 3
        for lamda in np.arange(0, np.pi, np.pi / 4):
            for gamma in (0.05, 0.5):
                gabor_label = 'Gabor' + str(num)  #Label Gabor columns as Gabor1, Gabor2...
                ksize=9
                kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lamda, gamma, 0, ktype=cv2.CV_32F)
                kernels.append(kernel)
                #Now filter the image and add values to a new column
                fimg = cv2.filter2D(img2, cv2.CV_8UC3, kernel)
                filtered_img = fimg.reshape(-1)
                df[gabor_label] = filtered_img  #Labels columns as Gabor1, Gabor2, etc.
                print(gabor_label, ': theta=', theta, ': sigma=', sigma, ': lamda=', lamda, ': gamma=', gamma)
                num += 1  #Increment for gabor column label

#COMPLEMENTARY FILTERS
#CANNY EDGE
edges = cv2.Canny(img, 100,200).reshape(-1)
df['Canny'] = edges
#ROBERTS EDGE
edge_roberts = roberts(img).reshape(-1)
df['Roberts'] = edge_roberts
#SOBEL
edge_sobel = sobel(img).reshape(-1)
df['Sobel'] = edge_sobel
#SCHARR
edge_scharr = scharr(img).reshape(-1)
df['Scharr'] = edge_scharr
#PREWITT
edge_prewitt = prewitt(img).reshape(-1)
df['Prewitt'] = edge_prewitt
#GAUSSIAN with sigma=3
gaussian_img = nd.gaussian_filter(img, sigma=3).reshape(-1)
df['Gaussian_sig3'] = gaussian_img
#GAUSSIAN with sigma=6
gaussian_img2 = nd.gaussian_filter(img, sigma=6).reshape(-1)
df['Gaussian_sig6'] = gaussian_img2
#MEDIAN with sigma=3
median_img = nd.median_filter(img, size=3).reshape(-1)
df['Median s3'] = median_img

#Now, add a column in the data frame for the Labels
labeled_img = cv2.imread('image_688575878_product_57497717.jpg')
labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_BGR2GRAY).reshape(-1)
df['Labels'] = labeled_img
print(df.head())
df.to_csv("Gabor3.csv")
```

TESTS UTILISATION DE PLUSIEURS FILTRES COMBINES POUR UN 1er MODÈLE DE MACHINE LEARNING - IMAGES RAKUTEN

STEP 1 -> création d'un 1er mdèle simple de ML pour en sortir les feature importances (ML4_img_rak.py)

```python
#Define the dependent variable that needs to be predicted (labels)
Y = df["Labels"].values

#Define the independent variables
X = df.drop(labels = ["Labels"], axis=1)

#Split data into train and test to verify accuracy after fitting the model.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.4, random_state=20)


from sklearn.ensemble import RandomForestClassifier
# Instantiate model with n number of decision trees
model = RandomForestClassifier(n_estimators = 100, random_state = 42)

# Train the model on training data
model.fit(X_train, y_train)

# verify number of trees used. If not defined above.
print('Number of Trees used : ', model.n_estimators)
# TESTING THE MODEL BY PREDICTING ON TEST DATA
prediction_test_train = model.predict(X_train)

#Test prediction on testing data.
prediction_test = model.predict(X_test)
from sklearn import metrics
#First check the accuracy on training data. This will be higher than test data prediction accuracy.
print ("Accuracy on training data = ", metrics.accuracy_score(y_train, prediction_test_train))
#Check accuracy on test dataset. If this is too low compared to train it indicates overfitting on training data
print ("Accuracy = ", metrics.accuracy_score(y_test, prediction_test))

#Get numerical feature importances
importances = list(model.feature_importances_)

feature_list = list(X.columns)
feature_imp = pd.Series(model.feature_importances_,index=feature_list).sort_values(ascending=False)
print(feature_imp)
import pickle
#Save the trained model as pickle string to disk for future use
filename = "rakuten_iml"
pickle.dump(model, open(filename, 'wb'))

#To test the model on future datasets
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.predict(X)

segmented = result.reshape((img.shape))

from matplotlib import pyplot as plt
plt.imshow(segmented, cmap ='jet')
plt.imsave('rakuten_estim.jpg', segmented, cmap ='jet')
```

```
[5 rows x 42 columns]
Number of Trees used :  100
Accuracy on training data =  1.0
Accuracy =  0.96094
Original Image     0.312821
Median s3          0.135375
Gabor24            0.086222
Gabor6             0.071140
Gabor8             0.050236
Gabor7             0.045058
Gaussian_sig3      0.042623
Gabor21            0.030429
Gabor31            0.029950
Gaussian_sig6      0.026675
Scharr             0.021103
Roberts            0.019611
Gabor29            0.019287
Sobel              0.018660
Gabor32            0.017691
Prewitt            0.017249
Gabor5             0.016534
Gabor30            0.011543
Gabor23            0.008515
Gabor4             0.006210
Gabor22            0.004803
Gabor12            0.002736
Gabor11            0.001896
Gabor3             0.001127
Gabor20            0.000956
Canny              0.000639
Gabor28            0.000577
Gabor27            0.000267
Gabor19            0.000070
Gabor25            0.000000
Gabor26            0.000000
Gabor1             0.000000
Gabor18            0.000000
Gabor17            0.000000
Gabor16            0.000000
Gabor15            0.000000
Gabor14            0.000000
Gabor13            0.000000
Gabor10            0.000000
Gabor9             0.000000
Gabor2             0.000000
dtype: float64
```

Filtres les + importants d'un modèle simple ML

a tester

tester modèles complémentaires

https://github.com/tensorflow/models/tree/master/research/slim#Pretrained

MobileNet_v2_1.0_224^*	Code	mobilenet_v2_1.0_224.tgz	71.9	91.0
NASNet-A_Mobile_224#	Code	nasnet-a_mobile_04_10_2017.tar.gz	74.0	91.6
NASNet-A_Large_331#	Code	nasnet-a_large_04_10_2017.tar.gz	82.7	96.2
PNASNet-5_Large_331	Code	pnasnet-5_large_2017_12_13.tar.gz	82.9	96.2
PNASNet-5_Mobile_224	Code	pnasnet-5_mobile_2017_12_13.tar.gz	74.2	91.9