

Project Deepnet

Realized at the HETIC school

Master I DATA & AI

OPEN FOOD FACTS

Team: 3

Data Visualisation

Product finder

Supervised by

- Rodolphe LEFEVRE

Realised by

- Louis CHARLES
- Quentin CHALOPIN
- Maxime PRINCE

2021/2022

Contents

1	Introduction	3
1.1	Open Food Facts	3
1.1.1	Images	3
1.1.2	Product caractreristics	4
1.1.3	Ingredients	4
1.1.4	Nutritional Information	5
1.2	Goals	5
1.3	Protocole	6
1.4	Import	6
2	Cleaning	8
2.1	Columns	8
2.2	Creator	9
2.3	Verification of values	11
2.4	Images	12
2.5	Labelisation	12
3	Machine Learning	14
3.1	Prototype	14
3.1.1	Resizing & Normalisation	14
3.1.2	Binding picture with name	15
3.1.3	Visualisation	17
3.1.4	Model	17
3.2	Training	19
3.2.1	Epoch	19
3.3	Prediction	20

3.3.1	K-Nearest Neighbors - KNN	20
3.4	Conclusion	23

Introduction

1.1 Open Food Facts

OFF (Open Food Facts) is a large "open source" database that allows users to add products related to the food industry in order to enrich it.

We can see through the site what users have been able to add and therefore directly inform the different descriptions are presented as follows :

1.1.1 Images



We can find on the site images of very good quality but many of them are just not usable or even out of subject.

1.1.2 Product characteristics

Caractéristiques du produit

Dénomination générique : Soda au cola

Quantité : 33cl

Conditionnement : Métal, en:Recyclable Metals, Aluminium, en:Aluminium-can, en:Canette, en:Drink can aluminium, en:Green Dot



Marques : Coca-Cola

Catégories : Boissons, Boissons gazeuses, Sodas, en:Boissons, en:Boissons gazeuses, en:Cola sucré, en:Sodas au cola

Labels, certifications, récompenses : en:Point Vert, en:Végétalien, en:Végétarien

Magasins : Carrefour

Pays de vente : France

1.1.3 Ingredients

Ingrédients

→ Les ingrédients sont listés par ordre d'importance (quantité).

Liste des ingrédients:

Eau gazéifiée, sucre, colorant : E150d, acidifiant : acide phosphorique, arômes naturels (extraits végétaux), dont caféine.

Traces éventuelles : Fruits à coque

Analyse des ingrédients :



Sans huile de palme



Peut-être végétalien



Peut-être végétarien

→ L'analyse est basée uniquement sur les ingrédients listés et ne prend pas en compte les méthodes de fabrication.

[Détail de l'analyse des ingrédients »](#)

Additifs :

- E150d - Caramel au sulfite d'ammonium
- E338 - Acide orthophosphorique  Risque élevé de sur-exposition

Groupe NOVA 


NOVA



4 - Produits alimentaires et boissons ultra-transformés

1.1.4 Nutritional Information


Informations nutritionnelles

Note nutritionnelle de couleur NutriScore 

NUTRI-SCORE



[Détail du calcul du Nutri-Score »](#)

 **Avertissement** : Le taux de fruits, légumes et noix n'est pas indiqué sur l'étiquette, il a été estimé en fonction de la liste des ingrédients : 0%

Taille d'une portion : 330ml

Comparaison avec les valeurs moyennes des produits de même catégorie :

☒ en:boissons-gazeuses ([35 produits](#))

☐ en:boissons ([236 produits](#))

☐ Sodas ([2724 produits](#))

☐ Boissons gazeuses ([3557 produits](#))

☐ Boissons ([51214 produits](#))

We have listed here one of the best data that we can have in the database below, all though many of them are found incomplete or wrong.

1.2 Goals

Using this database and the Python programming language, we wish to develop an AI capable of identifying a product according to its image. Thus a user could, thanks to a hypothetical application, "just take a picture" of a product and it will lead him to the data associated with the product. It would also be easier for a user to identify all the characteristics that a consumer might need to study a product.

1.3 Protocole

To do this we must "clean" the database in order to keep only products with all the necessary information as we saw in 1.1. We will thus be able, in the ideal, to train a model on the "best" values of the dataset. Once the selection has been made, we need to find the best python library for image recognition. We will be able to obtain predictions of the name of the product according to the presented image.

1.4 Import

Here is the list of libraries used for the realization of this project in python:

```
1  # Basic import for computer science
2  import numpy as np
3  import pandas as pd
4  # Google colab
5  import os
6  # Regex
7  import re
8  #Add delay
9  import time
10 # Scraper
11 from bs4 import BeautifulSoup
12 import requests
13 import urllib.request
14 # Picture Download
15 from PIL import Image
16
17 #Visualisation
18 import matplotlib.pyplot as plt
19 import seaborn as sns
```

```
1 #Machine Learning
2 from keras.backend_config import image_data_format
3 from keras.preprocessing.image import img_to_array
4 import cv2
5 import tensorflow as tf
6 import pathlib
7
8 from sklearn.neighbors import KNeighborsClassifier
9 from sklearn.metrics import
```


Cleaning

After getting lost more than 3/4 of the time in this huge database we came to the conclusion that out of more than 2 million lines only a handful of data was really efficient.

2.1 Columns

The DataFrame is made up of 191 columns and 2,148,801 rows, so we have to filter by deleting all the columns with more than 80% of "NaN", in fact we estimate that these columns are too empty to be exploitable

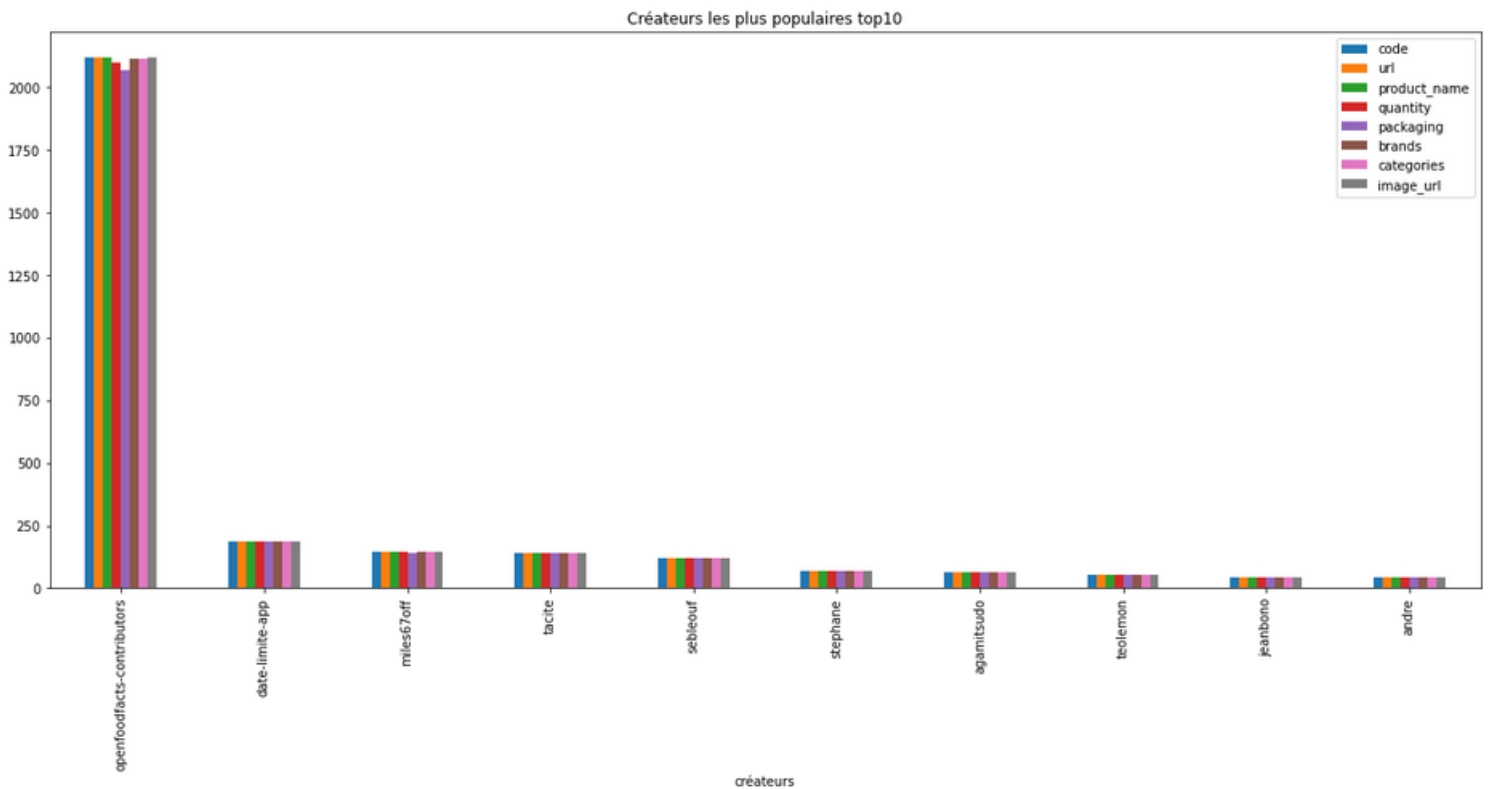
```
1  #Drop Column > 79% NaN
2
3  column_with_nan = df.columns[df.isnull().any()]
4  for col in column_with_nan:
5      if df[col].isnull().sum()*100.0/df.shape[0] > 79:
6          df.drop(col,1, inplace=True)
7
8  #Drop Row with NaN
9
10 index_with_nan = df.index[df.isnull().any(axis=1)]
11 df.drop(index_with_nan,0,inplace=True)
```

After going through all the remaining columns we decided to keep the most relevant ones.

```
1 keep_column = ['code', 'url', 'creator', 'product_name',
2 'quantity', 'packaging', 'brands', 'categories', 'image_url']
```

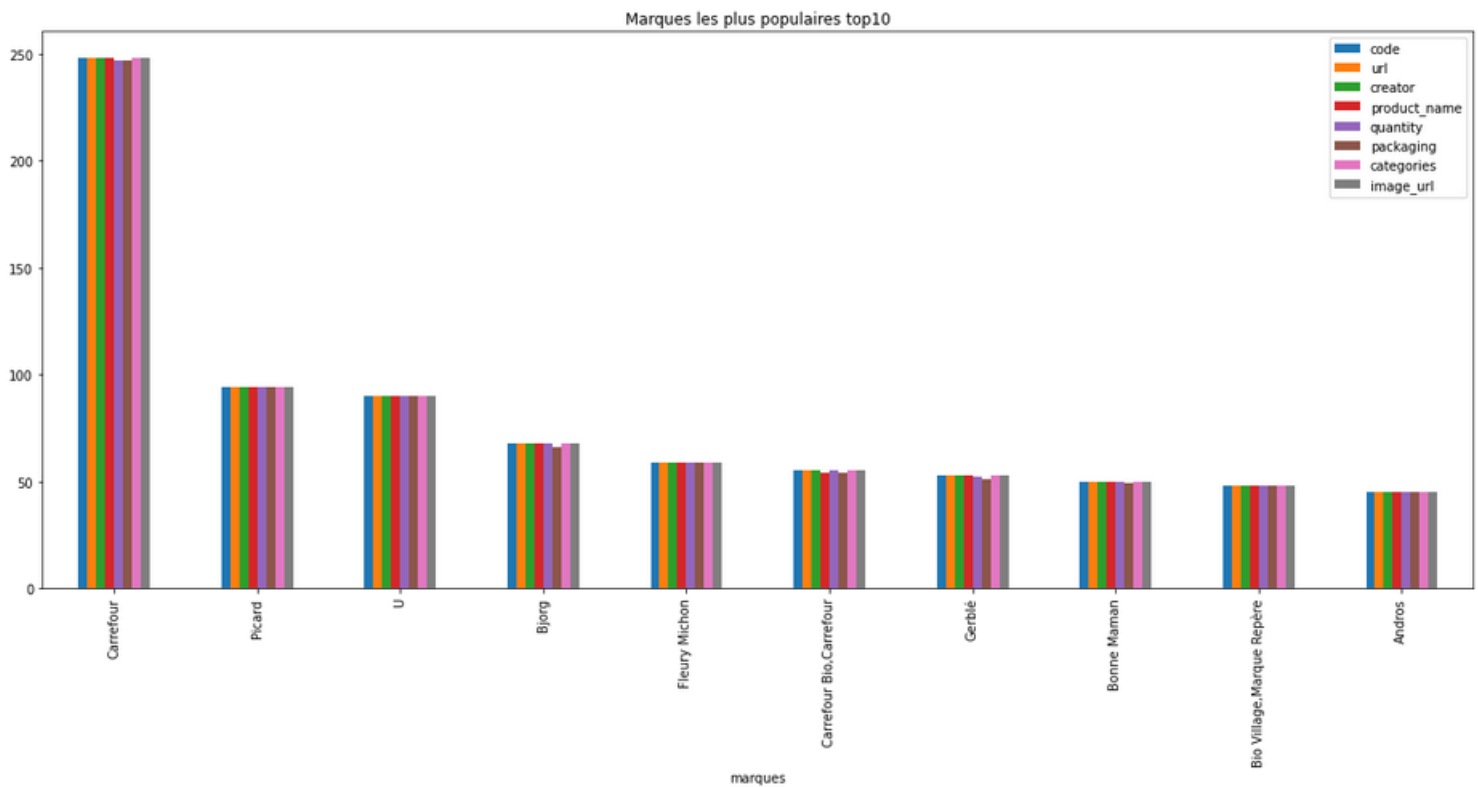
2.2 Creator

Among the creators populating this database we noticed that some did not completely fill all the columns of the DataSet, so we chose that no matter how popular and how many products a creator entered we will focus on those who filled all the columns.



```
1 df_creator_pop = df_populaire.groupby(['creator'],
2 as_index=True).count()
3 df_creator_pop = df_creator_pop.sort_values(['code'],
4 ascending=False)
5
6 plt.rcParams["figure.figsize"] = (20,8)
7 (df_creator_pop.head(10)).plot(kind='bar')
8 plt.title('Cr ateurs les plus populaires top10')
9 plt.xlabel('cr ateurs')
10 plt.show()
```

We also noticed that some brands are more common in this database so we find a way to show on the graphic below the top 10 of brands that are almost completely f.



```

1 df_creator_pop = df_populaire.groupby(['creator'],
2 as_index=True).count()
3 df_creator_pop = df_creator_pop.sort_values(['code'],
4 ascending=False)
5
6 plt.rcParams["figure.figsize"] = (20,8)
7 (df_creator_pop.head(10)).plot(kind='bar')
8 plt.title('Cr ateurs les plus populaires top10')
9 plt.xlabel('cr ateurs')
10 plt.show()

```

2.3 Verification of values

After removing a large part of the data, we will now check that the remaining data are usable.

So we made a scrapper coded with BeautifulSoup a library that can retrieve values from websites, based on the structure of an HTML page.

```
1 def scrap(x):
2     url = (f"{x}")
3     r = requests.get(url)
4     soup = BeautifulSoup(r.content, 'html5lib')
5     table = soup.findAll('img',
6     attrs = {'class': 'hide-for-xlarge-up'})
7     i=0
8     # url
9     print("\n" + f"{url}" + "\n")
10    # regex
11    barcode = re.findall(r"\d+",url)
12    print(barcode[0])
13
14    for row in table:
15        if i%2 == 0:
16            if 'front_es' in re.findall(r"front_es",row['src']):
17                print(f"img_{i}_{row['src']}")
18            elif 'ingredients_es' in re.findall(r"ingredients_es",
19            row['src']) :
20                print(f"ingre_{i}_{row['src']}")
21            elif 'nutrition_es' in re.findall(r"nutrition_es",
22            row['src']) :
23                print(f"nutri_{i}_{row['src']}")
24        i +=1
```

This function allows us to check if the values present on the site are identical to those found in the database, after analysis we realized that many links are dead and others are not identical to those in the database but gives equivalent results.

2.4 Images

In this part we will simply download the images associated with the database url.

```
1 def Array_Pro(url,code):
2     try :
3         name = (f"{code}.png")
4         urllib.request.urlretrieve(url,name)
5         img = Image.open(name)
6         img_resize = img.resize((128, 128))
7         img_resize.save(name)
8         return name
9     except:
10        return None
11 df.apply(lambda x:
12 (1.append(Array_Pro(x['image_url'],x['code']))),axis=1)
```

2.5 Labelisation

The recovered data it was necessary to make a final cleaning, to begin to train a model we need to choose between 3 modes of models :

- Supervised
- Semi-Supervised
- No Supervised

Having the column of product names "*product_name*" as well as the product images we can theoretically realize a supervised training method.

The image associated with the product name will help the model to learn and assimilate an image to a name. We will go deeper into the subject of machine learning in the next chapter.

The problem is that we are dealing with "open source" data, in other words, the names of the products are likely to be different from one identical product to another.

So we renamed by hand 4800 values to prepare the last part of our project.

Machine Learning

3.1 Prototype

First we decided to use the classifier with basic neural network but we didn't have enough pictures per product, even when we used 10 pictures per product over 100 products we didn't succeed to setup a correct neural network.

The second choice brought us to test a combination between a decoder and an KNN with only one neighbor because we only have 1 picture per product.

3.1.1 Resizing & Normalisation

We decided to use a scale of 128 because we are trying to have enough information over the picture in order to identify brands on the products.

```
1 directory_train = pathlib.Path('dataset')
2 directory_test = pathlib.Path('Image_test')
3
4 image_train_list = list(directory_train.glob('*.png'))
5 image_test_list = list(directory_test.glob('*'))
6
7 #Resizing
8
9 image_size=128
10 img_data = []
```

```

1  for img in image_train_list:
2      img=cv2.imread(str(img), 1)
3      rgb_img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
4      rgb_img=cv2.resize(rgb_img, (image_size,image_size))
5      img_data.append(img_to_array(rgb_img))
6  train_data=np.reshape(img_data, (len(img_data),
7  image_size, image_size, 3))
8  train_data=train_data.astype('float32')/255
9
10 #Normalisation
11
12 test_data=test_data.astype('float32')/255

```


3.1.2 Binding picture with name

Once the picture is transformed into usefull data we can affiliate is class number to his labelised name number.

```

1  #Label
2
3  train_labels = [int(str(label).split('/')[1]
4      .split('.')[0]) for label in image_train_list]
5
6  test_labels = [int(str(label).split('/')[1]
7      .split('.')[0].split()[0]) for label in image_test_list]
8
9  #DataFrame
10
11 df_labels = pd.read_csv('dataset.csv')
12 df_labels.head()

```


	label	image_url	class_number	
0	chocolat noir / poulain	https://images.openfoodfacts.org/images/produ...	0	
1	beurre de cacahuètes / skippy	https://images.openfoodfacts.org/images/produ...	1	
2	guacamole / old el paso	https://images.openfoodfacts.org/images/produ...	2	
3	biscuits au chocolat / cadbury	https://images.openfoodfacts.org/images/produ...	3	
4	céréales bio / bjorg	https://images.openfoodfacts.org/images/produ...	4	

3.1.3 Visualisation

```
1 plt.figure(figsize=[5,5])
2 plt.subplot(121)
3 curr_lbl = train_labels[2]
4 plt.imshow(train_data[2])
5 plt.title("(Label:␣" +
6           str(df_labels.loc[df_labels['class_number'] ==
7           curr_lbl]['label'].values[0]) + " )")
8 plt.axis("off")
```



3.1.4 Model

Initialisation

Here We are fixing all the variables who's gonna be usefull for the training part on the model.

```
1 batch_size = 100
2 epochs = 20
3 inChannel = 3
4 x, y = 128, 128
5 input_img = Input(shape = (x, y, inChannel))
6 num_classes = 1469
```

Encoder & Decoder

```
1 def encoder(input_img):
2     #encoder
3     #input = 28 x 28 x 1 (wide and thin)
4     conv1 = Conv2D(256, (3,3), activation='relu',
5     padding='same')(input_img)
6     pool1 = MaxPooling2D((2,2),
7     padding='same')(conv1)
8     conv2 = Conv2D(64, (3,3), activation='relu',
9     padding='same')(pool1)
10    pool2 = MaxPooling2D((2,2), padding='same')(conv2)
11    #conv3 = Conv2D(256, (3,3), activation='relu',
12    #padding='same')(pool2)
13    #pool3 = MaxPooling2D((2,2), padding='same')(conv3)
14    return pool2
15
16 def decoder(pool3):
17     #decoder
18     conv4 = Conv2D(64, (3,3), activation='relu',
19     padding='same')(pool3)
20     up1 = UpSampling2D((2,2))(conv4)
21     conv5 = Conv2D(256, (3,3), activation='relu',
22     padding='same')(up1)
23     up2 = UpSampling2D((2,2))(conv5)
24     #conv6 = Conv2D(32, (3,3), activation='relu',
25     #padding='same')(up2)
26     #up3 = UpSampling2D((2,2))(conv6)
27     decoded = Conv2D(3, (3,3), activation='relu',
28     #padding='same')(up2)
29     return decoded
```

When all the function are ready we can execute them by using the following lines of codes.

Before compiling we can see that we set up the optimizer and loss function to certain kind of metrics.

```
1  #Set Up
2
3  autoencoder = Model(input_img, decoder(encoder(input_img)))
4  autoencoder.compile(optimizer = "adam",
5  loss="mean_squared_error")
```

3.2 Training

3.2.1 Epoch

To train the model we have to minimise the loss function by modifying certain neurons inside the the neural network. This one is evolving trough epoch to create the best model. When the model is ready we can put the test data to predict results

```
1  autoencoder_train = autoencoder.fit(train_data, train_data,
2  batch_size=batch_size, callbacks=[learning_rate_scheduler,
3  early_stopping], epochs=epochs, verbose=1)
```

```

Epoch 1/20
15/15 [=====] - 27s 759ms/step - loss: 0.2026 - lr: 0.0010
Epoch 2/20
15/15 [=====] - 7s 440ms/step - loss: 0.0523 - lr: 8.1873e-04
Epoch 3/20
15/15 [=====] - 7s 441ms/step - loss: 0.0333 - lr: 6.7032e-04
Epoch 4/20
15/15 [=====] - 7s 445ms/step - loss: 0.0227 - lr: 5.4881e-04
Epoch 5/20
15/15 [=====] - 7s 446ms/step - loss: 0.0192 - lr: 4.4933e-04
Epoch 6/20
15/15 [=====] - 7s 447ms/step - loss: 0.0169 - lr: 3.6788e-04
Epoch 7/20
15/15 [=====] - 7s 447ms/step - loss: 0.0150 - lr: 3.0119e-04
Epoch 8/20
15/15 [=====] - 7s 448ms/step - loss: 0.0139 - lr: 2.4660e-04
Epoch 9/20
15/15 [=====] - 7s 448ms/step - loss: 0.0133 - lr: 2.0190e-04
Epoch 10/20
15/15 [=====] - 7s 446ms/step - loss: 0.0128 - lr: 1.6530e-04
Epoch 11/20
15/15 [=====] - 7s 445ms/step - loss: 0.0125 - lr: 1.3534e-04
Epoch 12/20
15/15 [=====] - 7s 444ms/step - loss: 0.0122 - lr: 1.1080e-04
Epoch 13/20
15/15 [=====] - 7s 445ms/step - loss: 0.0120 - lr: 9.0718e-05
Epoch 14/20
15/15 [=====] - 7s 445ms/step - loss: 0.0118 - lr: 7.4274e-05
Epoch 15/20
15/15 [=====] - 7s 446ms/step - loss: 0.0117 - lr: 6.0810e-05
Epoch 16/20
15/15 [=====] - 7s 446ms/step - loss: 0.0116 - lr: 4.9787e-05
Epoch 17/20
15/15 [=====] - 7s 446ms/step - loss: 0.0114 - lr: 4.0762e-05
Epoch 18/20
15/15 [=====] - 7s 447ms/step - loss: 0.0114 - lr: 3.3373e-05
Epoch 19/20
15/15 [=====] - 7s 447ms/step - loss: 0.0113 - lr: 2.7324e-05
Epoch 20/20
15/15 [=====] - 7s 446ms/step - loss: 0.0112 - lr: 2.2371e-05

```

3.3 Prediction

3.3.1 K-Nearest Neighbors - KNN

Because we used supervised learning techniques, the KNN method gonna help us determines to create classifications between all the datas for a better predictions.

```

1 accuracy_score, confusion_matrix
2 def KNN(label, x_train, y_train, x_test, y_test):
3
4     trainStart = time.time()
5
6     clf = KNeighborsClassifier(n_neighbors = 1,
7     weights='distance')
8
9     flat_img_nb = 65536
10
11     x_train = x_train.reshape((x_train.shape[0],
12     flat_img_nb))
13     x_test = x_test.reshape((x_test.shape[0],
14     flat_img_nb))
15
16     clf.fit(x_train, y_train)
17
18     y_pred = clf.predict(x_test)
19     for index, y in enumerate(y_test):
20         print("prediction: " + str(y_pred[index]) +
21         "/ True label: " + str(y))
22
23     trainEnd = time.time()
24
25     trainTook = round(trainEnd - trainStart, 2)
26
27     testStart = time.time()
28
29     print("Accuracy of {} KNN is {}".format(
30     label, accuracy_score(y_pred=y_pred, y_true=y_test)))

```

```

1  confmatrix = confusion_matrix(y_pred=y_pred,
2  y_true=y_test)
3
4  plt.subplots(figsize=(100,100))
5  sns.heatmap(confmatrix,annot=True,fmt=".1f",
6  linewidths=1.5)
7  plt.show()
8
9  testEnd = time.time()
10
11  testTook = round(testEnd-testStart,2)
12
13  print("Training_{}_KNN_took_{}_seconds".format(
14  label,trainTook))
15  print("Testing_{}_KNN_took_{}_seconds".format(
16  label,testTook))

```

Now we can execute all the functions below.

```

1  encoder = Model(input_img, encoder(input_img))
2  train_codes = encoder.predict(train_data)
3  test_codes = encoder.predict(test_data)
4
5  KNN("With_Encoding",train_codes,train_labels,
6  test_codes,test_labels)

```



```
prediction : 307 / True label : 1
prediction : 488 / True label : 1
prediction : 117 / True label : 2
prediction : 444 / True label : 66
prediction : 66 / True label : 66
prediction : 88 / True label : 29
prediction : 450 / True label : 29
prediction : 1120 / True label : 127
prediction : 73 / True label : 127
prediction : 134 / True label : 198
prediction : 1247 / True label : 198
prediction : 669 / True label : 198
prediction : 1113 / True label : 200
prediction : 1138 / True label : 200
prediction : 1416 / True label : 216
prediction : 1443 / True label : 216
prediction : 32 / True label : 216
prediction : 1437 / True label : 216
prediction : 188 / True label : 312
prediction : 312 / True label : 312
prediction : 40 / True label : 312
prediction : 312 / True label : 312
prediction : 165 / True label : 312
prediction : 1440 / True label : 312
```

3.4 Conclusion

The difficult part 80% of our time was used to clean all the database and create an usefull dataset who is labelled to help the machine learning.

The accuracy of a value of 9% is very low and this can be explained by a lack of images per product, the model does not recognize what is important on a given product. By using 10 images the model could recognize certain patens like particular shapes or colors with different backgrounds.

By using data augmentation we could have better results with only one image.

The data augmentation allows to create copies of an image with different parameters such as colors, contrasts, flipping in short creates different situation of an image.