SIMPLON.CO

Microsoft

Projet Chef d’Oeuvre

Cheeser, outil de reconnaissance de fromages

Corantin OGIER

Sommaire

1. Introduction

1.1. Contexte

Avec une consommation annuelle moyenne de 30kg, parmis plus de 1000 variétés de fromages, les français accordent une réelle importance à ce produit laitier. 95% d'entre eux en consomment au moins une fois par semaine et 40% quotidiennement.
Le fromage s'est indéniablement imposé dans le patrimoine culturel et gastronomique français et certaines variétés possèdent un label permettant de mettre en avant et préserver le savoir-faire local. Aujourd'hui la France compte 46 fromages AOP ("Appellation d'Origine Protégée").
C'est dans ce contexte que je me suis lancé sur le sujet du fromage pour mon projet chef d'œuvre, alors que moi-même je n'avais que très peu de connaissances sur les différents fromages, leurs histoires et leurs goûts… Autant dire que je me suis régalé ! Les techniques apprises au cours de la formation m'ont permis de mettre au point une application proposant divers services dont l'outil principal, la reconnaissance d'images de fromages.

1.2. Problématique

Comment mettre en place une application permettant à un utilisateur de prédire un type de fromage en fonction de l'image envoyée ? Comment renseigner l'utilisateur sur les caractéristiques des fromages ainsi que sur leurs histoires ? Est-il possible de proposer un itinéraire vers la fromagerie la plus proche ?

1.3. Démarche de résolution

Pour répondre à ces problématiques, je propose de réaliser une application Flask, codée en Python, qui va permettre à un utilisateur de s'inscrire puis de se connecter afin d'accéder à divers services dont les suivants:

- Prédiction d'image de fromage
- Histoire des fromages
- Localisation de la fromagerie la plus proche

Une fois la prédiction effectuée avec la probabilité associée, l'utilisateur pourra valider ou invalider la classe prédite. Cela me permettra ultérieurement de visualiser les faux positifs afin de mieux comprendre les conditions qui ont permis une mauvaise prédiction, mais aussi de réentraîner le modèle pour le rendre plus robuste.
Un onglet Histoire sera mis en place dans le menu afin de proposer à l'utilisateur de découvrir les différentes variétés de fromages, leurs caractéristiques mais aussi leurs passionnantes histoires.
L'activation de la géolocalisation par l'utilisateur permettra, en utilisant l'API Maps de Google, de lui proposer la fromagerie la plus proche de sa position.

1.4. Analyse des besoins

L’application Cheeser peut être utilisée par les consommateurs de fromages, occasionnels comme réguliers. Elle permet à l’utilisateur d’obtenir le nom du fromage pris en photo mais aussi les caractéristiques de divers fromages pour potentiellement le rediriger vers des fromages à goûter qui se rapprochent le plus de ses goûts, ou tout simplement en apprendre plus pour enrichir ses connaissances. En proposant la fromagerie la plus proche, elle permet de mettre en avant le savoir-faire français et nos petits commerçants.

- “En tant que consommateur, j’obtiens de plus amples informations sur mon fromage en me connectant et en me dirigeant vers l’histoire du fromage en question.”
- “En tant que consommateur, quand je prends en photo mon fromage, j’obtiens le nom correspondant.”
- “En tant que consommateur, l’application me géolocalise et me propose la fromagerie la plus proche.”

2. Gestion du projet

2.1. Méthode

Pour la gestion du projet, j’utilise l’application de management Trello. C’est une application dont j’ai déjà eu recours pour des projets professionnels, simple d’utilisation et qui permet une meilleure organisation des tâches. Le projet Cheeser est une réalisation indépendante, je suis donc le seul développeur. Chaque mois je réalise un compte-rendu de ce qui a été réalisé ainsi que les différentes observations et résultats obtenus (modèles, collecte d’images, nouvelles idées, …), tout en veillant à la bonne réalisation des tâches présentes dans le tableau Trello.
Trello : (images à venir)

2.2. Rétroplanning

Rétro planning: (images à venir)

3. Intelligence artificielle

Le but de l’IA Cheeser est de prédire avec une image de fromage, la classe correspondante parmi plusieurs types de fromage. Il s’agit donc de vision par ordinateur (Computer Vision) réalisant un problème de classification multiclasse.

3.1. Dataset

Pour réaliser un modèle de computer vision qualitatif, il est nécessaire d’obtenir un grand nombre d’images. Pour cela, j’ai tout d’abord créé plusieurs dossiers correspondants aux différentes classes de fromages que le modèle sera capable de prédire, puis j’ai collecté des images sur internet :

- Google Images

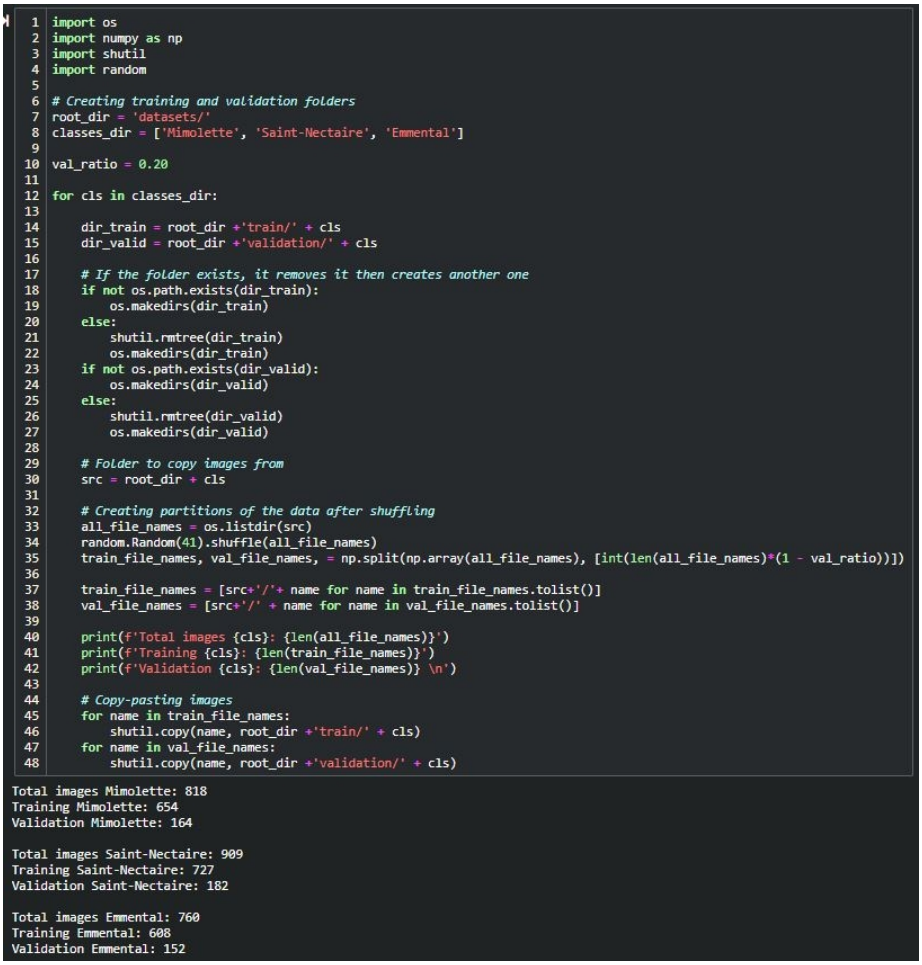
- Réseaux sociaux (Facebook, Instagram, Pinterest, …)
- Banques d'images

Ayant fait le tour des images disponibles et souhaitant plus de variété, je me suis par la suite dirigé vers des fromagerie. Ainsi, j'ai pu acheter les fromages nécessaires pour réaliser mes propres photos. Cela m'a permis d'obtenir un grand nombre d'images, d'autant plus que je pouvais facilement faire varier les paramètres permettant au modèle de bien généraliser : l'angle de vue, la luminosité, le morceau en lui-même (taille, forme, …), arrière plan (assiettes de différentes couleurs et formes, avec des couverts ou encore des accompagnements, …), créer un léger flou, changer la qualité de la photo, etc !
Pour encore plus de diversité, mais aussi pour avoir une première approche du comportement de l'utilisateur final de Cheeser, j'ai mis en place un formulaire AirTable qui permet aux participants de soumettre une ou plusieurs photos de fromages. Grâce à cela j'ai pu collecter de nouvelles inputs jamais vues par le modèle, ni en entraînement ni en test. J'ai aussi la possibilité de créer de nouvelles photos en prenant exemple sur les retours obtenus.

Actuellement, le modèle est entraîné à effectuer une prédiction sur trois classes :

- Emmental
- Mimolette
- Saint-Nectaire

J'ai réalisé un script qui me permet de créer un dossier train et un dossier validation avec au sein de chacun, trois sous-dossiers pour chacune des classes de fromages. Sont copiées ensuite les images de chacune des classes depuis les dossiers originaux, datasets entiers, puis distribuées aléatoirement dans chaque sous-dossier correspondant, pour l'entraînement et pour la validation, en respectant le ratio indiqué train-validation (0.8-0.2).



```python
import os
import numpy as np
import shutil
import random

# Creating training and validation folders
root_dir = 'datasets/'
classes_dir = ['Mimolette', 'Saint-Nectaire', 'Emmental']

val_ratio = 0.20

for cls in classes_dir:

    dir_train = root_dir +'train/' + cls
    dir_valid = root_dir +'validation/' + cls

    # If the folder exists, it removes it then creates another one
    if not os.path.exists(dir_train):
        os.makedirs(dir_train)
    else:
        shutil.rmtree(dir_train)
        os.makedirs(dir_train)
    if not os.path.exists(dir_valid):
        os.makedirs(dir_valid)
    else:
        shutil.rmtree(dir_valid)
        os.makedirs(dir_valid)

    # Folder to copy images from
    src = root_dir + cls

    # Creating partitions of the data after shuffling
    all_file_names = os.listdir(src)
    random.Random(41).shuffle(all_file_names)
    train_file_names, val_file_names, = np.split(np.array(all_file_names), [int(len(all_file_names)*(1 - val_ratio))])

    train_file_names = [src+'/'+ name for name in train_file_names.tolist()]
    val_file_names = [src+'/' + name for name in val_file_names.tolist()]

    print(f'Total images {cls}: {len(all_file_names)}')
    print(f'Training {cls}: {len(train_file_names)}')
    print(f'Validation {cls}: {len(val_file_names)} \n')

    # Copy-pasting images
    for name in train_file_names:
        shutil.copy(name, root_dir +'train/' + cls)
    for name in val_file_names:
        shutil.copy(name, root_dir +'validation/' + cls)
```

```
Total images Mimolette: 818
Training Mimolette: 654
Validation Mimolette: 164

Total images Saint-Nectaire: 909
Training Saint-Nectaire: 727
Validation Saint-Nectaire: 182

Total images Emmental: 760
Training Emmental: 608
Validation Emmental: 152
```

Fig. Script de création des datasets de train et validation

Le dataset est composé de :

Total images Mimolette: 818
Training Mimolette: 654
Validation Mimolette: 164

Total images Saint-Nectaire: 909
Training Saint-Nectaire: 727
Validation Saint-Nectaire: 182

Total images Emmental: 760
Training Emmental: 608
Validation Emmental: 152

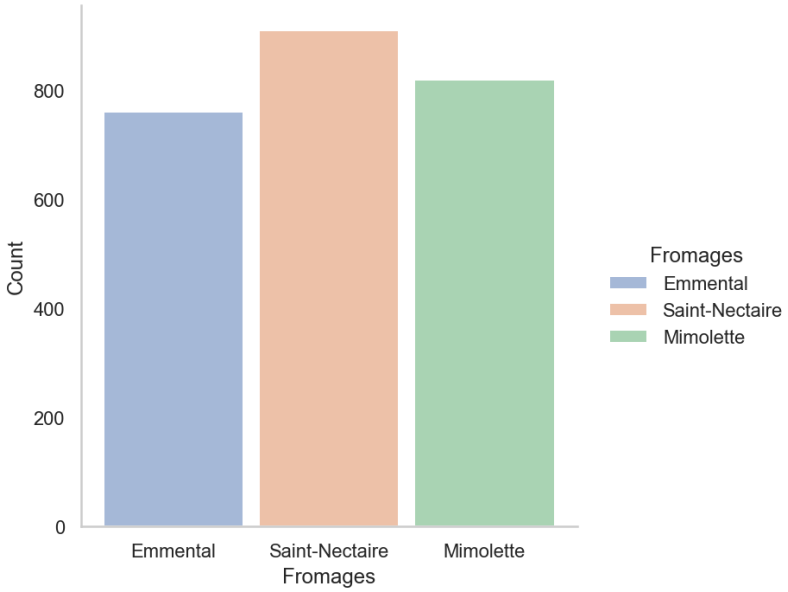


Fig. Distribution des classes de fromages

Le nombre d'images est assez proche pour chaque classe même si on remarque que celle du Saint-Nectaire est plus représentée que les deux autres, cela ne pose pas de problème de déséquilibre pour la généralisation du modèle.

Fig. Quelques exemples d'images de fromages qui serviront à l'entraînement

Les images de tests sont quant à elles stockées dans un sous dossier test_folder, lui-même dans le dossier test. Elles ont été sélectionnées :

- Via AirTable pour obtenir des images en “conditions réelles” qui se rapprochent le plus de celles utilisées par l’utilisateur final.
- Via une recherche sur Internet, en faisant bien attention à ce que celles-ci ne soient pas déjà présentes dans le dataset qui sert à créer ceux de train et de validation.
- Via des photos que j’ai prises moi-même en respectant bien les variétés d’exécutions.

Il y a 31 images de test.
Ces images sont alors inconnues du modèle. Elles permettent d’effectuer un test cohérent, d’observer et d’évaluer le comportement de celui-ci avec des images variées des trois classes de fromages.

La disposition des dossiers et sous-dossiers est importante puisque j’utilise de la Data Augmentation Online (aussi dite On-the-fly) avec ImageDataGenerator de Keras. Cela permet, lors de l’entraînement, que le modèle puisse voir de “nouvelles” images afin d’augmenter sa généralisation, ces images sont en fait celles du training dataset, certaines ayant subi aléatoirement diverses modifications.
La méthode flow_from_directory() va permettre de lire les images au sein des différents dossiers et nécessite donc cette architecture spécifique.

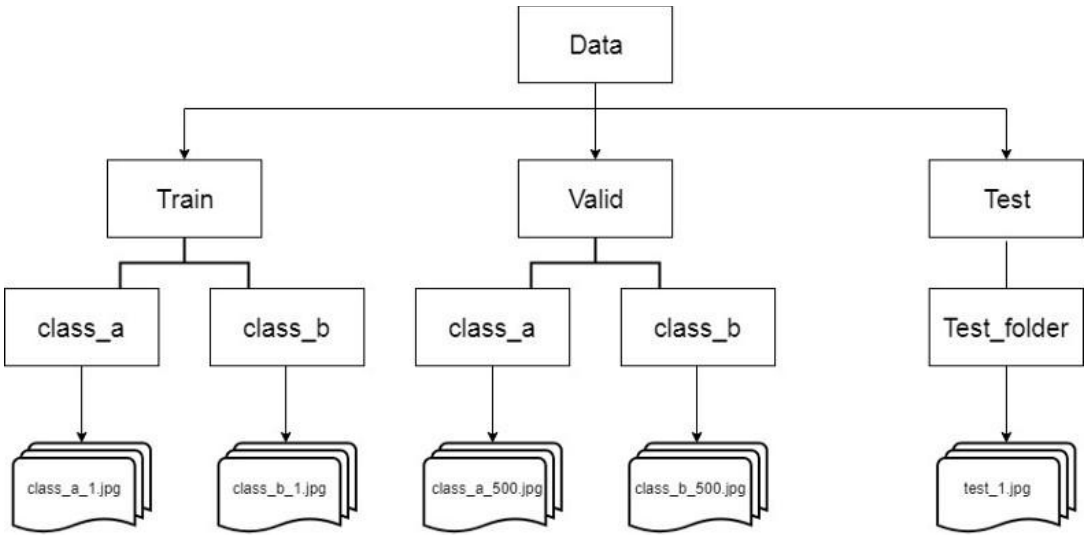


Fig. Architecture de la disposition nécessaire des dossiers/sous-dossiers/images

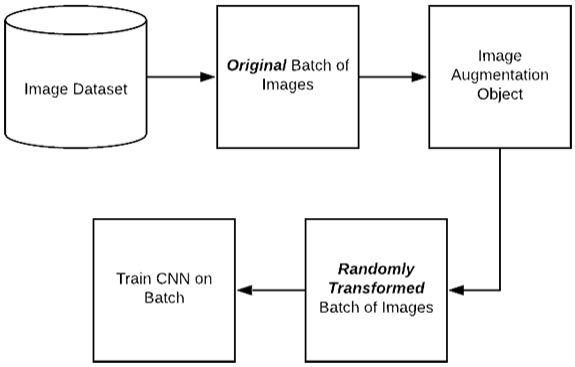


Fig. Schéma de fonctionnement de la Data Augmentation Online

```python
# Testing the imagedatagenerator to see which parameters are the best for the training

# load the image
x = random.randint(1,len(os.listdir('datasets/Emmental')))
img = load_img(f'datasets/Emmental/1 ({x}).jpg')

# convert to numpy array
data = img_to_array(img)

# expand dimension to one sample
samples = np.expand_dims(data, 0)

# create image data augmentation generator
datagen = ImageDataGenerator(brightness_range=[0.7,1.0], rotation_range=180, zoom_range=[0.8,1.0], horizontal_flip=True
                             vertical_flip=True)

# prepare iterator
it = datagen.flow(samples, batch_size=1)

# generate samples and plot
for i in range(1,9,1):
    # define subplot
    sp = plt.subplot(3, 3, i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    sp.axis('Off')
    plt.title(f'Image {x} augmented')
    plt.imshow(image)

plt.show()
```

Fig. Visualisation paramètres ImageDataGenerator

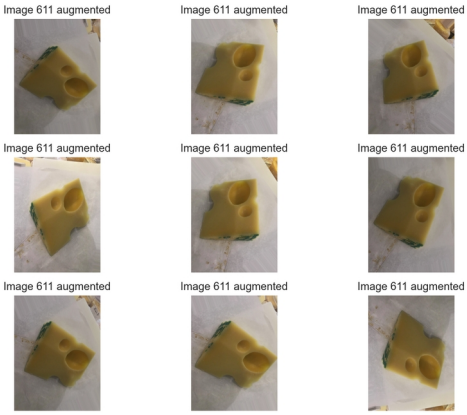


Fig. Images d'Emmental augmentées

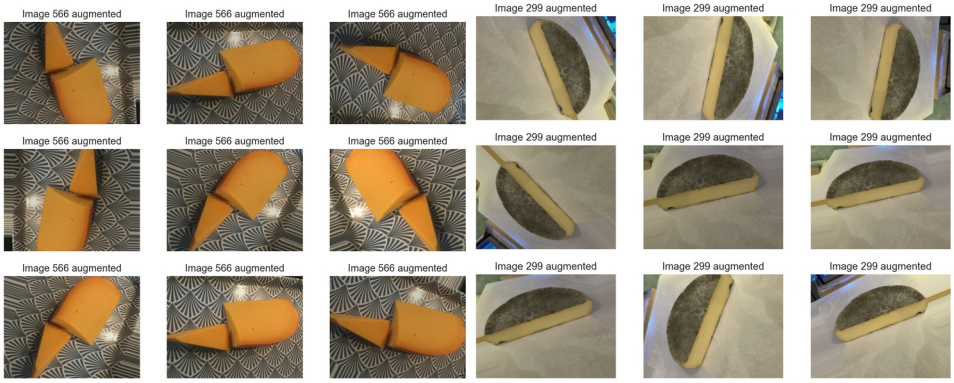


Fig. Images de Mimolette augmentées *Fig. Images de Saint-Nectaire augmentées*

Chaque image au sein du dataset possède sa propre taille mais la méthode flow_from_directory() me permet également de choisir la dimension souhaitée, avec target_size(150,150) j'obtiens donc un redimensionnement des images en 150x150.
Un autre traitement à effectuer mais applicable aussi depuis ImageDataGenerator() est le rescale. Chaque image en couleurs est constituée de coefficients RVB compris entre 0 et 255, mais ces valeurs sont trop élevées pour que notre modèle puisse bien les traiter, on les multiplie donc par un facteur 1./255 pour qu'elles soient comprises entre 0 et 1.

Les images de validation et de test doivent refléter la réalité du comportement de l'utilisateur final et ne doivent donc pas subir d'augmentation ou de prétraitements autres que le rescaling.

```python
# Generate batches of tensor image data with real-time data augmentation.
train_datagen = ImageDataGenerator(rescale=1./255,
# All images will be rescaled by 1./255 to transform every pixel value from range [0,255] -> [0,1]
                                   rotation_range=180,
                                   zoom_range=[0.7,1.0],
                                   brightness_range=[0.6,1.0],
                                   horizontal_flip=True,
                                   vertical_flip=True)

# For validation data, we'll apply only rescaling, because our validation set must reflect "real world" performance.
valid_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
    train_dir,  # This is the source directory for training images
    target_size=(150, 150),  # All images will be resized to 150x150
    batch_size=20,
    # Since we use categorical_crossentropy loss, we need categorical labels
    class_mode='categorical',
    shuffle=True,
    seed=41)

# Flow validation images in batches of 20 using valid_datagen generator
valid_generator = valid_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='categorical',
    shuffle=True,
    seed=41)

# Flow test images in batches of 1 using test_datagen generator
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=1,
    class_mode=None,
    shuffle=False)
```

```
Found 1989 images belonging to 3 classes.
Found 498 images belonging to 3 classes.
Found 31 images belonging to 1 classes.
```

Fig. Application d'ImageDataGenerator

Expliquer la manière dont le dataset a été stocké et importé en base de donnée "analytique"

Montrer un bout de code qui fait l'importation

3.2. Méthodes

Justifier le choix des IA considérées et les outils pour la réalisation

Expliquer rapidement avec des schéma les IA utilisées dans le projet IA

Expliquer les features considérées pour votre projet

Pour chaque modèle testé indiquer le choix des hyperparamètres (et architecture si deep learning) : régularisation, learning rate, max_depth, etc.

Pour chaque modèle testé indiquer la méthodologie d'entraînement : nombre de donnée, taille de batch, etc.

Expliquer rapidement les métriques utilisées pour évaluer votre modèle

3.3. Résultats

Tableau de performance avec TRAIN ET TEST pour les métriques

Proposer des explications sur les éventuellements mauvaise performances des modèles

Proposer des axes d'améliorations

4. Application

Rappeler à quoi sert l'application

Mettre un schéma fonctionnel présentant les différents composants et comment ils interagissent

4.1. Base de données

Décrire le choix de votre base de donnée et son utilitée dans l'application

Décrire les outils utilisés pour modéliser la base de données, UML, MCD, Merise, etc.

Mettre le schema de la base de données en le justifiant : justifier quelques choix de type de colonnes, décrire les différents types de relations composant votre base en les justificiant

Lister les requêtes SQL principales de votre application vers votre base de donnée (INSERTION DONNEE ET UTILISATION)

Indiquer la manière dont la base de donnée a été mise en place : mysql workbench, script SQL, bdd managée, etc. etc.

Indiquer une méthode pour faire un backup automatisé

Mesurer la durée d'une requête et proposer une optimisation pour l'accélérer si besoin

4.2. Backend (pas obligatoire)

Décrire les différentes parties de votre backend

Décrire comment l'IA interagit avec votre backend

4.3. Frontend

Décrire le frontend : choix des outils, les différentes pages

Montrer des screenshots de l'application

4.4. Qualité et monitoring (monitoring pas obligatoire)

Expliquer rapidement votre approche pour déboguer votre application

Expliquer rapidement les différents types de tests qui existent pour s'assurer la qualité d'un logiciel

Expliquer comment vous vous y êtes pris pour effectuer les tests de votre application

Faire un ou plusieurs tests unitaires ou fonctionnel avec par exemple pytest

PETIT screenshot de code d'un test

Rappeler le principe du monitoring et de la gestion de logs

4.5. Pour aller plus loin…

- Proposer des recettes avec du fromage
- blob storage (images nombreuses…)
- (toutes x images FP)

- faire un lien entre prédiction à chaque user en fonction de leur connexion ?
- rajouter table d’entrainement - réintégrer données utilisées /
- table fromages avec info sur le fromage ?
- script récupérer les images FP pour réentraîner ?

5. Annexes

5.1. Bibliographie