

Présentation du projet et état des lieux :

L'application déjà présente s'appelle Triof, il s'agit d'un code réalisé en [Python](#), un langage populaire de programmation.

Elle a pour objectif de décomposer les étapes d'une machine à recycler et d'y intégrer une notion numérique et domotique.

Le code fourni présente une interface graphique qui fonctionne via la bibliothèque de python : [Flask](#).

Flask a la particularité de simuler un serveur, celui-ci peut être local ou mis dans un container.

Dans cette architecture, l'application à un usage local, le serveur serait lié au fonctionnement du recycleur, et le client aurait en interface tactile les moyens de communication.

En l'espèce, il existe un formulaire d'utilisateur présent dans l'application en développement, il y aura donc des échanges de requête entre le serveur et le client grâce à des requêtes PUSH ou GET sur la gestion des identifiants.

Le reste du fonctionnement de l'application prévoit l'affichage d'une image issue du dossier "Caméra", cette image devra être reconnue par l'utilisateur qui validera sa réponse, mais la réponse ne sera pas stockée.

L'objectif attendu du client est que la machine devienne capable de reconnaître les objets attendus selon 3 familles d'objets en plastique qui sont :

Couvert – Gobelet – Bouteille.

Pour la seconde mission, reconnaître un objet sale permettrait d'optimiser la qualité du fil du recycleur.

La reconnaissance d'image

Cette technologie s'appelle la computer vision, il s'agit de faire appliquer notre vision à un ordinateur.

Le principe serait, comme le ferait un humain, de voir des objets et qu'il assigne le terme commun attendu. Pour notre situation, il s'agira de la labellisation de gobelets, bouteilles et couverts en plastique.

Pour faire comme un humain, l'ordinateur a besoin d'une manière de classifier (Machine learning) ou d'un [réseau de neurone](#) (Deep learning) dans les 2 cas le sets de données doit être labellisé, le résultat étant attendu il s'agit de donnée discriminante.

Il existe 2 moyens pour procéder à l'établissement de ce type de réseau :

- Une solution Local
- Une solution No code

Les deux solutions disposent d'avantages et d'inconvénients, elles dépendent des objectifs à court terme et des moyens techniques et / ou financiers dont le client dispose.

Les solutions no code ont un avantage d'être facilement mis en place, être user friendly car elles ne nécessitent pas forcément un développeur et sont efficace sous réserves d'un abonnement de service.

Les principaux prestataires pour ce genre de service sont :

- Amazon avec son service AWS, module : Computer vision
- Google avec son service Vision, module : IA VISION
- Microsoft avec son service AZURE, module : Computer vision

Les solutions Local impliquent un développeur qui devra éventuellement obtenir des données et connaître un langage de programmation qui dispose de module / bibliothèque permettant de gérer l'entraînement du réseau de neurone.

Pour le langage de python il s'agit des bibliothèques Tensorflow.Keras et de Pytorch pour les plus populaires.

Utilisation d'une API

Prélude : Pour des raisons purement scolaires et de partenariat, Azure est retenue comme solution pour la reconnaissance des 3 familles de produits.

La création d'un modèle de computer Vision :

Tout d'abord il faut se créer un compte sous Azure en renseignant une carte bancaire, celle-ci servira à financer les consommations effectuées sur la plateforme.

Dans le cadre de la mission, la création du groupe de ressource s'effectuera en mode "Free" et non "Pro" celui-ci répond aux critères concernant le nombre de personnes simultanément en ligne et sur le nombre de projets.

Une fois les étapes préliminaires effectuées, l'accès aux instances de la computer vision permet de se créer un projet.

2 méthodes sont possibles, la graphique en interface web ou celle du SDK, pour des raisons de simplicité j'ai opté pour l'interface graphique.

Jusqu'à présent, aucun frais n'est engagé pour le client.

Pour la partie utilisateurs, j'ai donc commencé mes travaux en tant qu'utilisateur et non développeur, Azure fonctionne avec une très bonne précision avec un nombre réduit d'éléments.

En conséquence, j'ai labellisé 5 photos de chaque élément à reconnaître à savoir des couverts, des gobelets et des bouteilles en plastique.




Image URL

REGUREBERAREQf//Z →

or

Browse local files

File formats accepted: [jpg](#), [png](#), [bmp](#)
File size should not exceed: [4mb](#)

Using model trained in

Iteration

Iteration 1

Predictions

Tag	Probability
Plastic + Couvert	100%
Plastic + Bottle	0%
Plastic + gobelet	0%

A ce stade le modèle de vision prévoit une grille de [tarification](#):

J'ai donc utilisé 25 images pour cet entraînement sur 5000 et environ 10 minutes sur une limite de 1 heure mensuel..

Je peux effectuer 10 000 prédictions sur le modèle avant d'avoir des frais, et le cas échéant cela coûtera 1,69 € les 1000 prédictions supplémentaires.

Le résultat du test en ligne donne une constatation, l'entraînement à bien fonctionner, mais le modèle ne reconnaît que les formes et non le matériau.




Image URL

→

or

Browse local files

File formats accepted: jpg, png, [bmp](#)
File size should not exceed: 4mb

Using model trained in

Iteration

Iteration 2

Predictions

Tag	Probability
Plastic + Couvert	79.9%
metal - couvert	20%
Plastic + Bottle	0%
Plastic + gobelet	0%

Dans l'absolu, cela ne pose pas de problème car l'application dépend essentiellement de l'utilisateur et non d'un traitement automatique.

Une fois que le modèle est en place, on peut soit en extraire les liens de données SDK ou API pour préparer son intégration à notre application.

De retour sur Flask, j'ai commencé par corriger la caméra.

Pour des raisons de sécurité, et même si l'application se veut locale, j'ai transféré le dossier caméra en static pour que les fichiers html puissent y accéder sans déroger aux règles de sécurité.

Avec cette procédure j'y lie les balises html et cela me permet d'afficher une image du dossier camera que ma fonction python renverra.

```
path = "static/camera/"  
  
list_of_files = glob.glob(path+'/*')  
latest_file = max(list_of_files, key=os.path.getctime)
```

La création de la fonction prévoit déjà de récupérer pour la prédiction le fichier le plus récent dans camera.

Je suis partie sur le fonctionnement d'une API, après la tentative de le faire en direct avec le SDK d'azure, j'ai fais cela en reprenant les paramètres d'adresse appeler Endpoint et en communiquant les informations headers dans un module request pour interroger l'api directement..

```
import requests
image = open(latest_file, 'rb').read()
endpoint = ('https://westeurope.api.cognitive.microsoft.com/customvision/v3.0/Prediction/5d1b58cb-1ba7-4d46-b40d-51a1d9625d89/classify/iterations/Iteration1/image')
headers = { "Prediction-Key" : '25fddb2027c1489f8544b4d67bf7a6aa' , "Content-Type" : "application/octet-stream" }

request = requests.post(endpoint, headers = headers, data = image)
pred = request.json()
```

Azure nous renvoie la prédiction, celle-ci est laissée en suggestion utilisateurs à travers un message dans sa balise

Ceci clôt la première partie du projet.

Le modèle a réussi 4 des 5 tests unitaires sur ses différentes possibilités de sortie.

Le résultat dépend d'une connexion internet et de la stabilité du service Azure.

Pour le cas actuel notre compte à été clôturé entre temps, une fonction indiquant un message comme quoi Azure était injoignable.

Utilisation d'un modèle local

Prélude : Pour des raisons purement scolaires la solution pour d'optimisation se réalisera en local.

La solution attendue est un pronostic complet ou complémentaire qui vient qualifier la qualité du plastique par le terme propre ou sale.

Je me suis donc posé la question qu'est ce que je considère sale ou propre, c'est un contexte de recyclage, les objets seront déjà utilisés donc forcément "sale".

A la différence de service clé en main, le nombre d'images doit être plus conséquent, j'ai donc mis au point un scrapper d'image google.

Celui-ci stocke les résultats et augmente la base d'image sur des termes anglais ou français, à travers une fonction de scrapping pouvant rechercher le terme dans en natif ou dans la langue la plus commune.

La problématique était la même que ma question d'origine, sur un échantillon de 200 bouteilles sales , 180 des images ne répondait pas aux attentes, et le reste pouvait représenter le même produit bouteille / marque sous un léger angle différent.

En conséquence, j'ai arrêté le scraper et ai décidé de prendre une quinzaine d'images propres et le même volume pour les images sales pour chaque produit.

Il existe deux solutions pour compenser un faible volume de données.

- La première est la data augmentation, il s'agit de jouer avec les données composants l'image, cela peut être du rognage, un décalage, une rotation, ajuster les contrastes, la luminosité ou les couleurs.

Keras dispose déjà d'une fonction permettant de le faire, il s'agit de `ImageDataGenerator`, il propose les inversions horizontal, vertical, et une randomisation du centrage de l'élément.

En conséquent, pour mieux la comprendre j'ai créé une fonction locale une équivalence de l'image data generator pour directement influencer le nombre de degrés et le contraste.

```
def data_augmentation(path,size,dest_path,degrees,step_degrees, rotate = True, extra_generator = False, photo_change = False):
    from PIL import Image, ImageEnhance
    import glob, os

    for infile in glob.glob(path + "*"):
        file, ext = os.path.splitext(infile)
        filename = os.path.splitext(os.path.basename(infile))[0]
        im = Image.open(infile)
        im.thumbnail(size)

        if not os.path.exists(dest_path):
            os.mkdir(dest_path)

        steps_choose = int(step_degrees)
        for steps in range(steps_choose) :
            if rotate is True:
                pictures = im.rotate(steps * (degrees / step_degrees))

                pictures.save(dest_path+ filename+"-{nb_img}.JPG".format(nb_img = steps))
                print(dest_path+ filename+"-{nb_img}.JPG".format(nb_img = steps))

            if extra_generator is True :
                ratio = 0.75
                pictures_bright = ImageEnhance.Brightness(pictures).enhance(ratio)
                pictures_bright.save(dest_path+ filename+"-{nb_img}-Bright-{ratio}.JPG".format(nb_img = steps, ratio = ratio))
                pictures_Contrast = ImageEnhance.Contrast(pictures).enhance(ratio)
                pictures_Contrast.save(dest_path+ filename+"-{nb_img}-Contrast-{ratio}.JPG".format(nb_img = steps, ratio = ratio))
                pictures_Sharpness = ImageEnhance.Sharpness(pictures).enhance(ratio)
                pictures_Sharpness.save(dest_path+ filename+"-{nb_img}-Sharp-{ratio}.JPG".format(nb_img = steps, ratio = ratio))
                pictures_Color = ImageEnhance.Color(pictures).enhance(1-ratio)
                pictures_Color.save(dest_path+ filename+"-{nb_img}-Color-{ratio}.JPG".format(nb_img = steps, ratio = 1-ratio))

            if photo_change is True :
                ratio = 0.50
                pictures_bright = ImageEnhance.Brightness(pictures).enhance(ratio)
                pictures_bright.save(dest_path+ filename+"-{nb_img}-Bright-{ratio}.JPG".format(nb_img = steps, ratio = ratio))
                pictures_Contrast = ImageEnhance.Contrast(pictures).enhance(ratio)
                pictures_Contrast.save(dest_path+ filename+"-{nb_img}-Contrast-{ratio}.JPG".format(nb_img = steps, ratio = ratio))
                pictures_Sharpness = ImageEnhance.Sharpness(pictures).enhance(ratio)
                pictures_Sharpness.save(dest_path+ filename+"-{nb_img}-Sharp-{ratio}.JPG".format(nb_img = steps, ratio = ratio))
                pictures_Color = ImageEnhance.Color(pictures).enhance(1-ratio)
                pictures_Color.save(dest_path+ filename+"-{nb_img}-Color-{ratio}.JPG".format(nb_img = steps, ratio = 1-ratio))
```

Cet argument de base me permet de donner une dimension et un format commun à la base d'image, cette résolution inférieure est optimum sur le stockage.

Cette solution ne permet malgré tout d'obtenir un bon entraînement quel que soit les variables du modèle, ce modèle fonctionne avec le module `image_dataset_from_directory`, qui vient construire directement les classes et les sets de données à partir de l'arborescence du dossier précédent remplis avec une fonction local et alternative d'image generator.

Ce modèle a stagner de par le manque de données en lui-même, 50 % soit l'équivalent d'un lancer de pièces.


```

path = "DL_CNN/"

directory = path + 'TRAIN'
Train = image_dataset_from_directory(
    directory,
    labels="inferred",
    label_mode="int",
    class_names=['PROPRE', 'SALE'],
    color_mode="grayscale",
    batch_size=32,
    image_size=(56, 56),
    shuffle=True,
    seed=1,
    validation_split=0.10,
    subset="validation",
    interpolation="bilinear",
    follow_links=False,)

directory = path + 'TEST'
Test = image_dataset_from_directory(
    directory,
    labels="inferred",
    label_mode="int",
    class_names=['PROPRE', 'SALE'],
    color_mode="grayscale",
    batch_size=32,
    image_size=(56, 56),
    shuffle=True,
    seed=1,
    validation_split=0.10,
    subset="validation",
    interpolation="bilinear",
    follow_links=False,)

```

Cet élément réside dans la faible volumétrie initiale de la base de données et dans la diversité qu'il l'a compose aussi bien dans sa feature propres que dans les sous-features qui est la distinction primaire des formes.

Le modèle initial de cet entraînement était en partie similaire à notre exercice de faire référence distinguant les chats et les chiens, les features distinguant les deux étant plus explicites avec une volumétrie de données bien plus élevés font que je suis partie sur une seconde solution.

Le transfert learning est une solution consistant à utiliser un nœud qui a déjà des poids et des reconnaissances de features, le plus commun étant le VGG16 je l'ai utilisé sans sa version de base.

```
from tensorflow.keras.applications.vgg16 import VGG16

model_VGG16 = Sequential([
    VGG16(weights='imagenet', include_top=False, input_shape=(56, 56, 3))
    , Flatten()
    , Dense(128, activation='relu', kernel_initializer='he_uniform')
    , Dense(1, activation='sigmoid')
])
```

Le principe était d'utiliser ce réseau déjà entraîné pour l'appliquer à mon jeu de données, tout en spécifiant les denses qui sont les features attendues, pour du binary / sigmoid on laissera le paramètre 1 et pour du softmax le paramètre 2.

Après un entraînement simple, le modèle était à 70 % de précision, nous obtenons donc déjà un jet plus précis, nous sommes toujours dans le cas d'un 'entraînement avec une carence de donnée, dans le cas où mes jeux de données se ressembleraient trop un drop out pour équilibrer le surentraînement aurait été nécessaire.

```
Epoch 5/5
239/239 [=====] - 458s 2s/step - loss: 1.6205e-04 - accuracy: 1.0000 - val loss: 1.4015 - val accuracy: 0.7125
```

Le modèle fut établi sur notebook car cela permet de constater plus facilement les changements de certains paramètres tels que l'optimisation ou la gestion des couches du nœud.

Ce modèle une fois sauvegardé peut être récupéré dans notre application flask, grâce au chargement du modèle qui est un module keras.

Pour le test de l'image, il faudra par contre tenir compte de la conversation en array de notre image dans un format noir et blanc, cela simplifie le nombre de données tout en se concentrant sur notre feature sale ou propre, il ne s'agit pas d'identifier des couleurs, les taches ou une forme et vient teindre l'objets ou ce qui décrit une détérioration.

Cette couche doit se reproduire quand nous chargeons l'image pour la prédiction de l'image, qui sera toujours la même que sur azure, mais charger du coup avec les paramètres de Azure pour atteindre le format array attendu par la librairie.

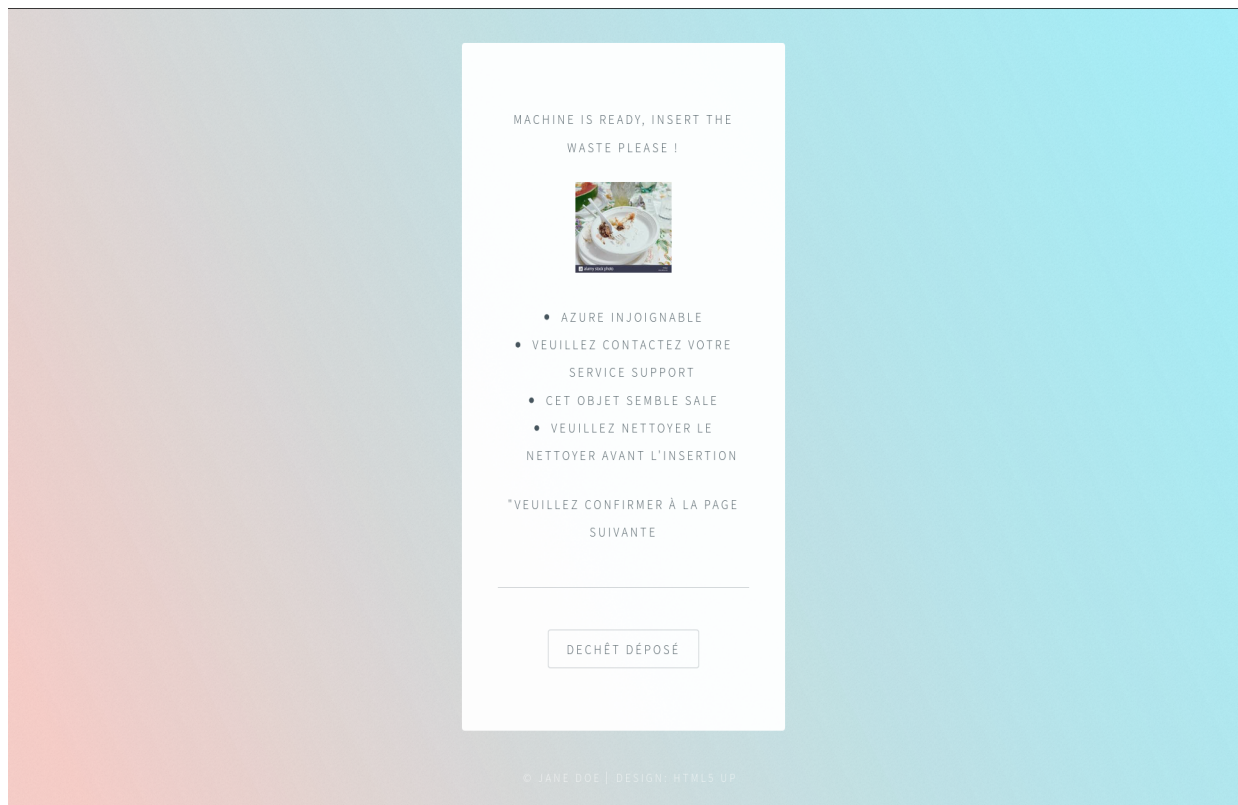
Une fois les modules de pré processing appliquer je lui ajoute une dimension pour répondre aux normes du module et la prédiction tombe sous le format 0 ou 1 dans la version binaire, 0 étant le propre et 1 le sale, ce résultat est directement convertis dans l'application et vient compléter la prédiction azure.

L'objet est un gobelet qui semble sale, ou qui semble propre, au vu de l'absence de précision explicite dans la consigne l'utilisateur dispose toujours du libre arbitre en lui laissant un message de se reporter aux consignes.

L'expérience est concluante 4 des 5 tests unitaires sont positifs, on est donc bien dans une fourchette de prédiction attendue.

L'application ne dispose pas de fonction en conséquent le test de régression est négatif, l'application reste à 100 % de fonctionnalité.

L'amélioration pourrait se faire d'une part avec des images de meilleure qualité et une volumétrie plus importante et dans un second temps dans la gestion des poids.



Annexe

Source Brute :

<https://www.python.org/>

<https://flask.palletsprojects.com/en/1.1.x/>

<https://azure.microsoft.com/en-us/pricing/details/cognitive-services/custom-vision-service/>

