Projet P8 Triof

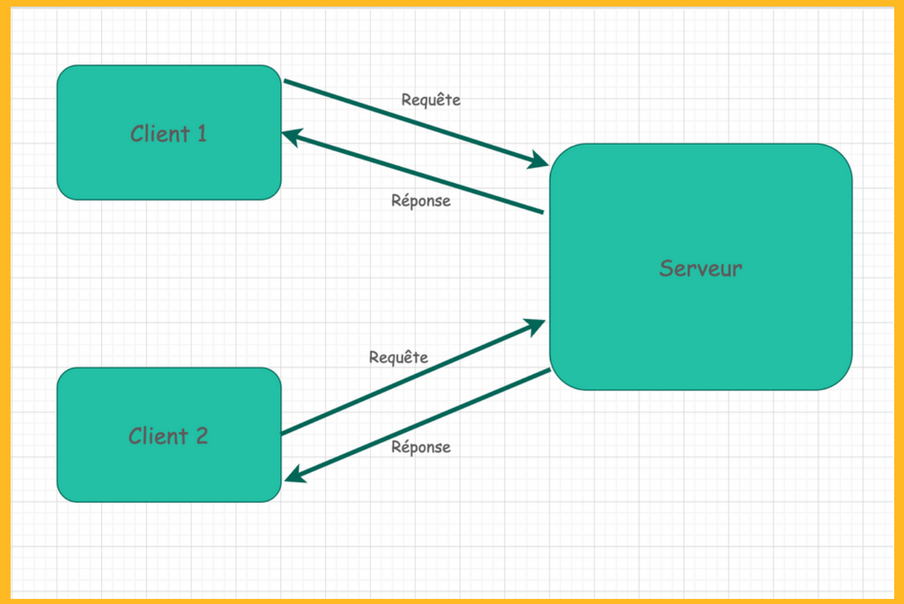
Ophélie Sabanowski

Amélioration grâce à l'IA d'une application existante

Triof est une start-up de tri et de recyclage des déchets spécialisée dans le traitement des déchets plastiques. Elle a développé une machine qu'elle loue aux entreprise : grâce à cette dernière les salariés peuvent déposer leurs déchets plastiques afin qu'ils soient recyclés en fil d'impression 3D.
Pour augmenter la qualité du filament, je vais proposer une solution intégrant de l'IA.

- GÉNÉRALITÉS SUR L'INTÉGRATION D'UNE IA DANS UNE APPLICATION EXISTANTE
- GÉNÉRALITÉS SUR LES SOLUTIONS CLOUD DE COMPUTER VISION
- L'APPLICATION TRIOF : ANALYSE , COMPRÉHENSION DU PROBLÈME
- PROPOSITION D'UNE SOLUTION
- UNE SOLUTION IA CLOUD
- UNE AMÉLIORATION VIA UNE SOLUTION LOCALE

Généralités sur l'intégration d'une IA dans une application existante



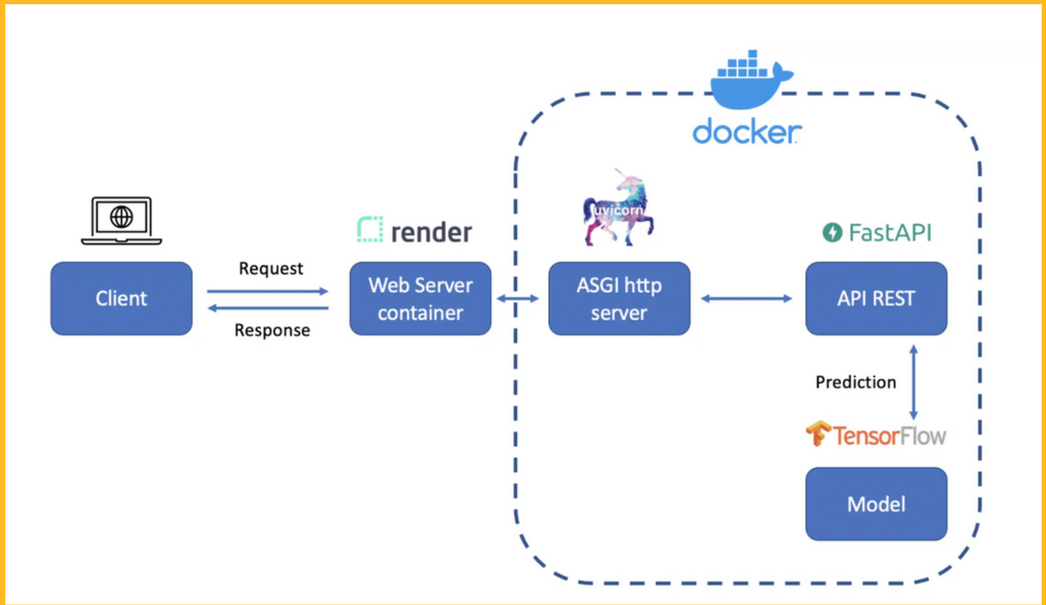
https://app.diagrams.net/#G1mSYeNEW9RVzvzlm4pVMEZnraTSGMsAPo

De nombreuses applications fonctionnent selon un environnement client/serveur, cela signifie que des machines clientes (des machines faisant partie du réseau) contactent un serveur, une machine généralement très puissante en terme de capacités d'entrée-sortie, qui leur fournit des services. Ces services sont des programmes fournissant des données telles que l'heure, des fichiers, une connexion, etc.

Les services sont exploités par des programmes, appelés programmes clients, s'exécutant sur les machines clientes. On parle ainsi de client (client FTP, client de messagerie, etc.) lorsque l'on désigne un programme tournant sur une machine cliente, capable de traiter des informations qu'il récupère auprès d'un serveur (dans le cas du client FTP il s'agit de fichiers, tandis que pour le client de messagerie il s'agit de courrier électronique).

Avantages de l'architecture client/serveur
Le modèle client/serveur est particulièrement recommandé pour des réseaux nécessitant un grand niveau de fiabilité, ses principaux atouts sont : des ressources centralisées, une meilleure sécurité, une administration au niveau serveur et un réseau évolutif.

Web service et modèle packagé en production



Un Web Service est une application qui permet d'échanger des données avec d'autres applications web. Même si ces dernières sont construites dans des langages de programmation différents. Parmi les Web Services les plus connus on peut citer SOAP, REST ou HTTP.

APIs et Web Services servent de “moyen de communication” entre plusieurs sites ou applications. La seule différence est qu'un service Web facilite l'interaction entre deux machines sur un réseau alors qu'une API sert d'interface entre deux applications différentes afin qu'elles puissent communiquer entre elles.
Tout service web est également une API puisqu’il expose les données ou les fonctionnalités d’une application. Mais toutes les API ne sont pas en même temps des services web.

Il empaquette toutes les ressources nécessaires pour héberger un modèle en tant que service web, et vous permet de télécharger une image Docker entièrement générée ou les fichiers nécessaires pour en générer une.
juste à faire un request, pas besoin de stocker et réentrainer le modèle.

Généralités sur les solutions Cloud de Computer Vision

Caractéristiques :
instance gratuit ou standard.
gratuit : Transactions de chargement, d'apprentissage et de prédiction
Jusqu'à 2 projets
Jusqu'à 1 heure d'apprentissage par mois
5 000 images d'apprentissage gratuites par projet
10 000 prédictions par mois
standard : Transactions de prédiction
Jusqu'à 100 projets
1,802 € toutes les 1 000 transactions
Formation 9,007 € par heure de calcul
Stockage des images
Jusqu'à 6 Mo chacun 0,631 € toutes les 1 000 images
tarification au mois pour une meilleure gestion

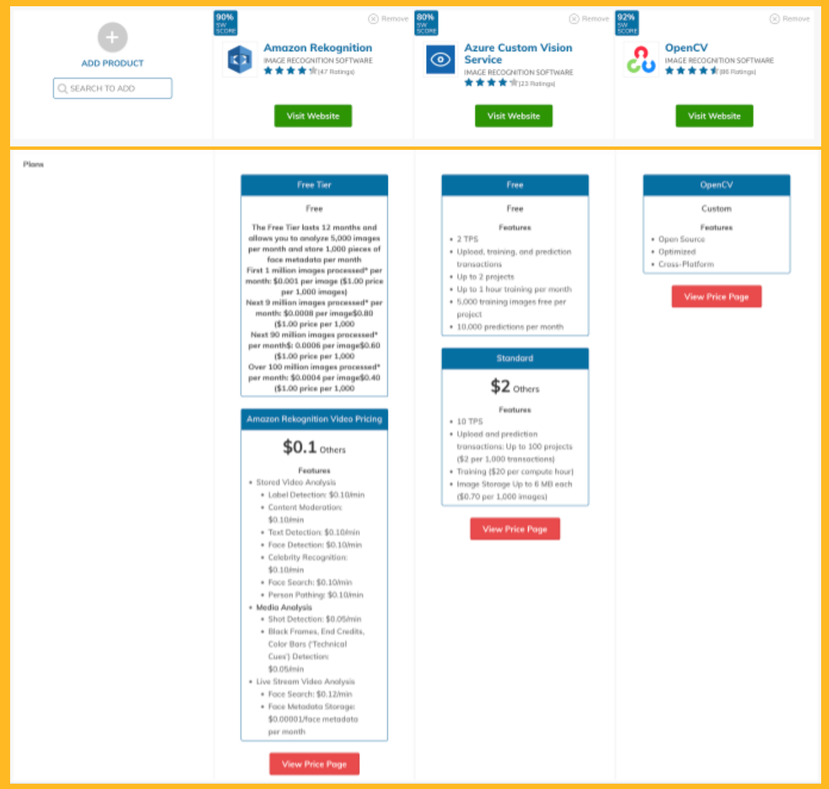
Le service Vision personnalisée facilite la création et le perfectionnement de classificateurs d'images personnalisés pour reconnaître le contenu spécifique des images.

Grâce au Machine Learning à la pointe de la technologie, on peut apprendre au classificateur à reconnaître ce qui importe, comme catégoriser les images des produits ou filtrer le contenu de site web. Il vous suffit de charger des images étiquetées et de laisser le service Vision personnalisée faire le gros du travail.

On peut acheter soit directement via microsoft soit via un partenaire azure.

Comparatif de différentes solutions Cloud

Azure Custom Vision, Amazon Rekognition, OpenCV



L'application Triof : analyse et compréhension du problème

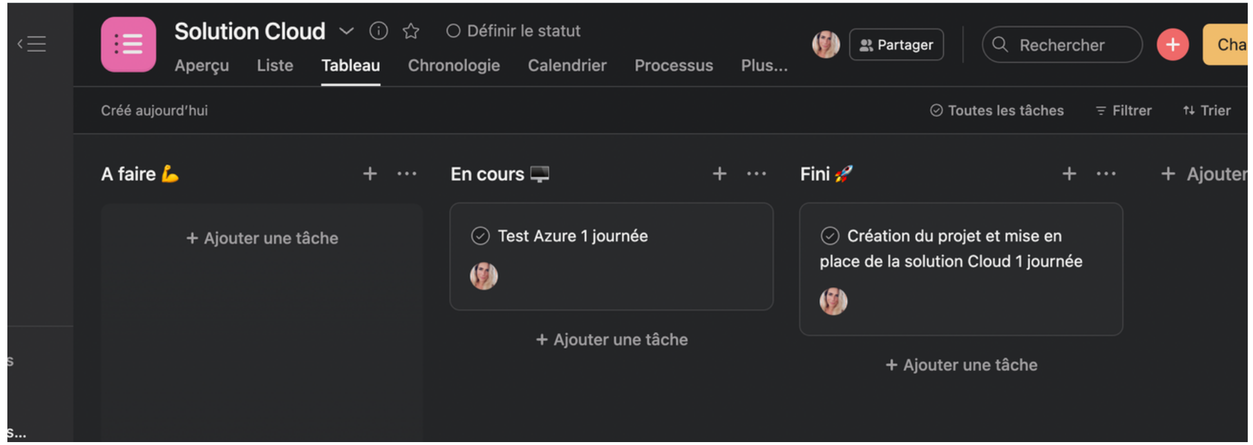
L'application est composée de:
-un dossier src avec les fichiers utils.py(indique les def des actions du site, dans src -> renvoie à la prise de photos de la poubelle), scrap.py(récupération des images sur Google),
- un dossier static avec les feuilles de styles(images, css, front), des fichiers photos -> camera
- 5 fichiers html des templates pour les input photos, type,..
-le fichier triof_app.py indique les routes des requêtes

Proposition d'une solution :

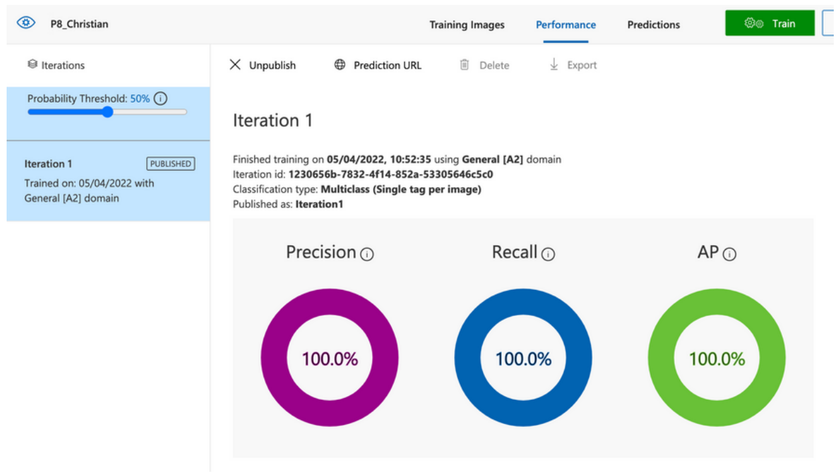
Je vais utiliser la solution Cloud d'Azure Custom Vision pour prédire le type de déchets.

Mise en place des tâches en méthodes agiles :

7 jours de travail prévus pour ce projet

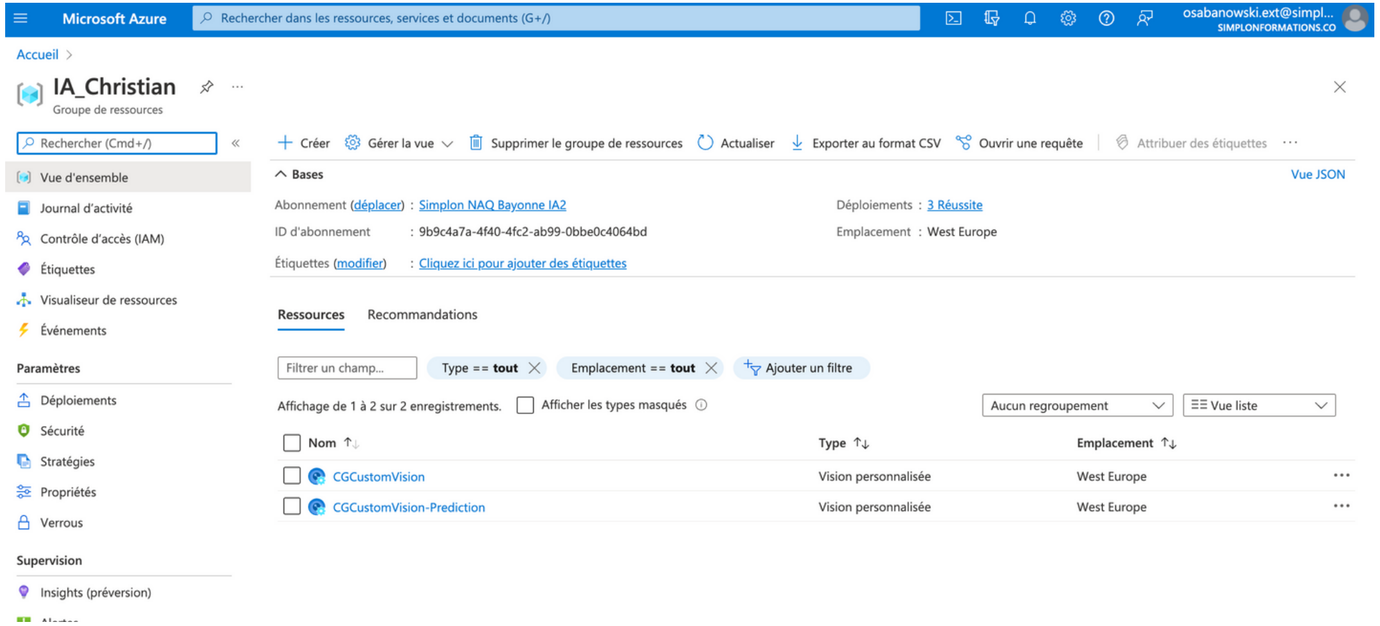


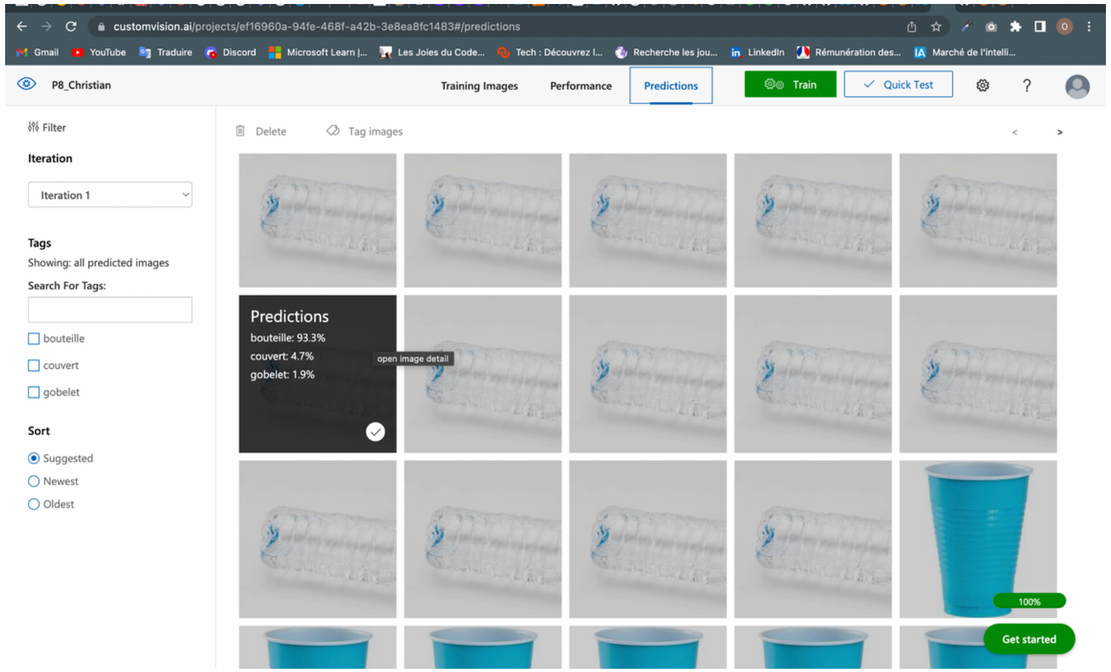
- Création du groupe de ressources, ressources et projet
- Déploiement
- Importation des images
- Entrainement
- Test



Performance Per Tag

Tag	Precision	Recall	A.P.	Image count
gobelet	100.0%	100.0%	100.0%	6
couvert	100.0%	100.0%	100.0%	6
bouteille	100.0%	100.0%	100.0%	6





Intégration du type de déchet dans l'application avec précochage de la prédiction

Welcome | E2_triof.md | type.html | triof_app.py 9+

Users > opheliesabanowski > Desktop > exercices > ProjetP8Triof > triof > templates > type.html > html > body.is-preload > div#wrapper > section#main > form > div.field

```
26                  </header>
27
28                  <div class="proba">
29                      <p>
30                          The probability that you inserted a {{prediction}} is PROBA % !
31                      </p>
32                  </div>
33
34                  <img src = {{ url_for('static', filename = "images/camera/" + pic )}}/>
35
36                  <form method="post" action="/confirmation">
37
38
39
40
41
42                      <div class="field">
43
44                          <input type="radio" name="type" id="bouteille" {% if prediction=="bouteille" %}
45                              checked="checked" {% endif %} value="Bouteille plastique">
46
47                          <label for="bouteille">Bottle</label>
48
49
50
51                          <input type="radio" name="type" id="gobelet" {% if prediction=="gobelet" %} checked="checked" {%
52                              endif %} value="gobelet">
53                          <label for="gobelet">Cup</label>
54
55
56
57                          <input type="radio" name="type" id="couvert" {% if prediction=="couvert" %} checked="checked" {%
58                              endif %} value="couvert">
59                          <label for="couvert">Cutlery</label>
60
61
62                      </div>
63
```

triof_app.py

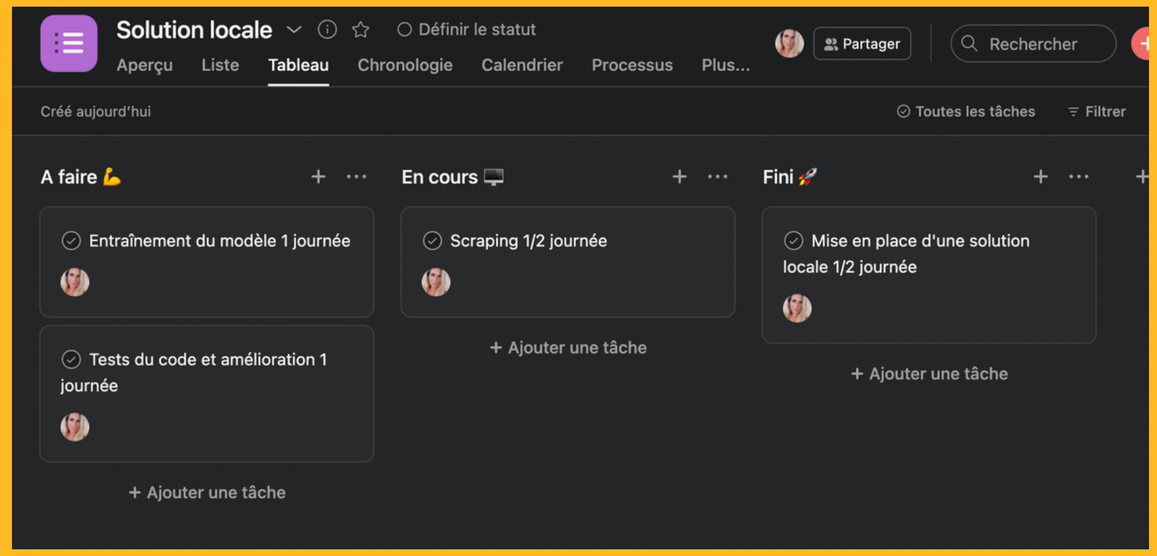
Welcome | E2_triof.md | type.html | triof_app.py 9+

Users > opheliesabanowski > Desktop > exercices > ProjetP8Triof > triof > triof_app.py

```
4
5
6   app = Flask(__name__)
7
8   @app.route('/')
9   def home():
10      return render_template('home.html')
11
12
13  @app.route('/start')
14  def insert():
15      open_waste_slot()
16
17      return render_template('insert.html')
18
19
20  @app.route('/waste/pick-type')
21  def pick_type():
22      close_waste_slot()
23      pic = take_trash_picture()
24      print(pic)
25      prediction = pred_func(pic)
26      print(prediction)
27      return render_template('type.html', prediction=prediction)
28
29
30  @app.route('/confirmation', methods=['POST'])
31  def confirmation():
32      waste_type = request.form['type']
33
34      process_waste(waste_type)
35      return render_template('confirmation.html')
36
37
38  if __name__ == "__main__":
39      app.run(debug=True)
40
```

Amélioration avec un modèle local

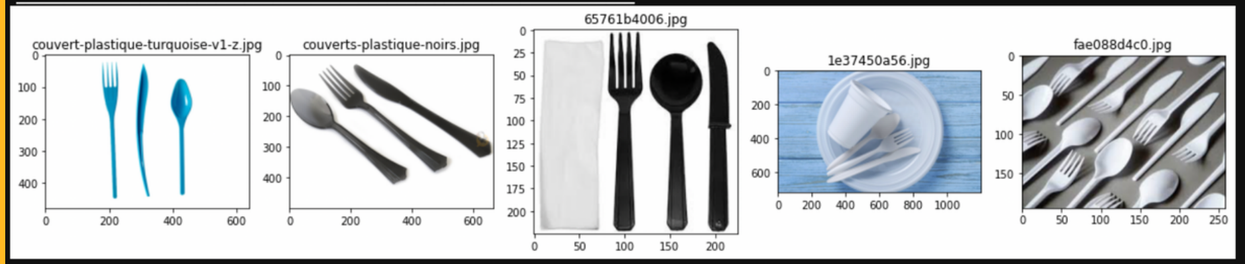
L'insertion par les salariés de plastiques sales entraîne un fil d'impression de moins bonne qualité.
Je vais créer un modèle en local permettant de trier les plastiques propres et les plastiques sales.
Premièrement, je mets de nouveau en place un planning agile pour cette solution.
Afin d'améliorer la solution, je vais scraper Google, utiliser la Data Augmentation puis interpréter les résultats.



Je choisis de créer mon propre modèle CNN pour la classification du type de déchet et la propreté.

```
IMG_WIDTH=64
IMG_HEIGHT=64


def create_dataset(img_folder):

    img_data_array=[]
    class_name=[]

    for dir1 in os.listdir(img_folder):
        for file in os.listdir(os.path.join(img_folder, dir1)):

            image_path= os.path.join(img_folder, dir1,  file)
            image= cv2.imread( image_path, cv2.COLOR_BGR2RGB)
            image=cv2.resize(image, (IMG_HEIGHT, IMG_WIDTH),interpolation = cv2.INTER_AREA)
            image=np.array(image)
            image = image.astype('float32')
            image /= 255
            img_data_array.append(image)
            class_name.append(dir1)
    return img_data_array, class_name
# extract the image array and class name
img_data, class_name =create_dataset(r'/Users/opheliesabanowski/Desktop/exercices/ProjetP8Triof/triof/static/images/test')

target_dict={k: v for v, k in enumerate(np.unique(class_name))}
target_dict

target_val=  [target_dict[class_name[i]] for i in range(len(class_name))]

model=tf.keras.Sequential(
        [
            tf.keras.layers.InputLayer(input_shape=(IMG_HEIGHT,IMG_WIDTH, 3)),
            tf.keras.layers.Conv2D(filters=64, kernel_size=3, strides=(2, 2), activation='relu'),
            tf.keras.layers.Conv2D(filters=64, kernel_size=3, strides=(2, 2), activation='relu'),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(2)
        ])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 31, 31, 64)        1792
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 15, 15, 64)        36928
_________________________________________________________________
flatten_4 (Flatten)          (None, 14400)             0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 28802
=================================================================
Total params: 67,522
Trainable params: 67,522
Non-trainable params: 0
_________________________________________________________________
```




```
model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
[68]: %%time
      model.fit( train_generator, validation_data = validation_generator ,epochs=10)

      Epoch 1/10
      5/5 [==============================] - 3s 320ms/step - loss: 5.1206 - accuracy: 0.5829 - val_loss: 7.0101 - val_accuracy: 0.4769
      Epoch 2/10
      5/5 [==============================] - 1s 215ms/step - loss: 6.4318 - accuracy: 0.4344 - val_loss: 3.6928 - val_accuracy: 0.4769
      Epoch 3/10
      5/5 [==============================] - 1s 237ms/step - loss: 3.2146 - accuracy: 0.4250 - val_loss: 0.8099 - val_accuracy: 0.5538
      Epoch 4/10
      5/5 [==============================] - 1s 233ms/step - loss: 0.7476 - accuracy: 0.6321 - val_loss: 0.6877 - val_accuracy: 0.5385
      Epoch 5/10
      5/5 [==============================] - 1s 238ms/step - loss: 0.6836 - accuracy: 0.6813 - val_loss: 0.6710 - val_accuracy: 0.7846
      Epoch 6/10
      5/5 [==============================] - 1s 230ms/step - loss: 0.6496 - accuracy: 0.7936 - val_loss: 0.5820 - val_accuracy: 0.8000
      Epoch 7/10
      5/5 [==============================] - 1s 242ms/step - loss: 0.5830 - accuracy: 0.7439 - val_loss: 0.5365 - val_accuracy: 0.8154
      Epoch 8/10
      5/5 [==============================] - 1s 236ms/step - loss: 0.5432 - accuracy: 0.7238 - val_loss: 0.5184 - val_accuracy: 0.8000
      Epoch 9/10
      5/5 [==============================] - 2s 431ms/step - loss: 0.4802 - accuracy: 0.8057 - val_loss: 0.6282 - val_accuracy: 0.8615
      Epoch 10/10
      5/5 [==============================] - 1s 260ms/step - loss: 0.4641 - accuracy: 0.8458 - val_loss: 0.8006 - val_accuracy: 0.8615
      CPU times: user 19.6 s, sys: 789 ms, total: 20.4 s
      Wall time: 13.8 s
[68]: <tensorflow.python.keras.callbacks.History at 0x18b635ca0>
```

Avec 154 images en train, 65 en test, 2 couches de convolution et 10 epochs.
J'obtiens une accuracy de 0,84 et une validation de 0,86.

```
[65]: gen = ImageDataGenerator()

[66]: !pwd

      /Users/opheliesabanowski/Desktop/exercices/ProjetP8Triof/triof/src

[67]: train_gen = ImageDataGenerator(rescale = 1./255)
      test_gen = ImageDataGenerator(rescale = 1./255)
      train_generator = train_gen.flow_from_directory('../static/images/camera/train',
                                                      target_size=(64,64))
      validation_generator = test_gen.flow_from_directory('../static/images/camera/test',
                                                          target_size=(64,64))

      Found 154 images belonging to 2 classes.
      Found 65 images belonging to 2 classes.
```

Je décide d'utiliser Augmentor pour avoir plus de données en entraînement et ainsi améliorer mon modèle.

```python
# Generating and saving 5 augmented samples
# using the above defined parameters.
i = 0
for batch in datagen.flow(x, batch_size = 1,
                          save_to_dir ='../static/images/camera/test/propre',
                          save_prefix ='image', save_format ='jpg'):
    i += 1
    if i > 15:
        break


# Importing necessary library
import Augmentor
SOURCE = "/Users/opheliesabanowski/Desktop/exercices/ProjetP8Triof/triof/static/images/camera/propre/"
OUTPUT = "/Users/opheliesabanowski/Desktop/exercices/ProjetP8Triof/triof/static/images/camera/test/propre/"
# Passing the path of the image directory
p = Augmentor.Pipeline(source_directory=SOURCE , output_directory=OUTPUT)

# Defining augmentation parameters and generating 5 samples
p.flip_left_right(0.5)
p.black_and_white(0.1)
p.rotate(0.3, 10, 10)
p.skew(0.4, 0.5)
p.zoom(probability = 0.2, min_factor = 1.1, max_factor = 1.5)
p.sample(5)
```

```
[72]: %%time
      model.fit( train_generator, validation_data = validation_generator ,epochs=10)

      Epoch 1/10
      13/13 [==============================] - 2s 149ms/step - loss: 6.4215 - accuracy: 0.4106 - val_loss: 1.5152 - val_accuracy: 0.86
      81
      Epoch 2/10
      13/13 [==============================] - 2s 143ms/step - loss: 2.1799 - accuracy: 0.8167 - val_loss: 1.1272 - val_accuracy: 0.86
      81
      Epoch 3/10
      13/13 [==============================] - 2s 138ms/step - loss: 1.7018 - accuracy: 0.8467 - val_loss: 1.3105 - val_accuracy: 0.86
      81
      Epoch 4/10
      13/13 [==============================] - 2s 139ms/step - loss: 1.6778 - accuracy: 0.8362 - val_loss: 1.1408 - val_accuracy: 0.86
      81
      Epoch 5/10
      13/13 [==============================] - 2s 143ms/step - loss: 1.5366 - accuracy: 0.8554 - val_loss: 1.9018 - val_accuracy: 0.87
      23
      Epoch 6/10
      13/13 [==============================] - 2s 140ms/step - loss: 2.1976 - accuracy: 0.8674 - val_loss: 2.0232 - val_accuracy: 0.86
      81
      Epoch 7/10
      13/13 [==============================] - 2s 142ms/step - loss: 2.3938 - accuracy: 0.8439 - val_loss: 2.0232 - val_accuracy: 0.86
      81
      Epoch 8/10
      13/13 [==============================] - 2s 141ms/step - loss: 2.4763 - accuracy: 0.8385 - val_loss: 2.0232 - val_accuracy: 0.86
      81
      Epoch 9/10
      13/13 [==============================] - 2s 142ms/step - loss: 2.5922 - accuracy: 0.8310 - val_loss: 2.0232 - val_accuracy: 0.86
      81
      Epoch 10/10
      13/13 [==============================] - 2s 143ms/step - loss: 2.2455 - accuracy: 0.8536 - val_loss: 2.0232 - val_accuracy: 0.86
      81
      CPU times: user 41.5 s, sys: 581 ms, total: 42.1 s
      Wall time: 18.6 s
[72]: <tensorflow.python.keras.callbacks.History at 0x18b6170a0>
```

J'ai légèrement amélioré la validation à 0,86.

J'ai ensuite essayé d'augmenter le nombre d'epochs à 30 puis de modifier le nombre de couches :

```
model=tf.keras.Sequential(
        [
            tf.keras.layers.InputLayer(input_shape=(IMG_HEIGHT,IMG_WIDTH, 3)),
            tf.keras.layers.Conv2D(filters=64, kernel_size=5, strides=(2, 2), activation='relu'),
            tf.keras.layers.Conv2D(filters=64, kernel_size=5, strides=(2, 2), activation='relu'),
            tf.keras.layers.Conv2D(filters=64, kernel_size=5, strides=(2, 2), activation='relu'),
            tf.keras.layers.Conv2D(filters=64, kernel_size=5, strides=(2, 2), activation='relu'),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(units = 50 , activation = 'relu'), Dropout(0.3),
            tf.keras.layers.Dense(units = 2 , activation = 'sigmoid')
        ])
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 62, 62, 64)        4864
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 29, 29, 64)        102464
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 13, 13, 64)        102464
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 5, 5, 64)          102464
_________________________________________________________________
flatten_8 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_8 (Dense)              (None, 50)                80050
_________________________________________________________________
dropout (Dropout)            (None, 50)                0
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 102
=================================================================
Total params: 392,408
Trainable params: 392,408
Non-trainable params: 0
```

Sans nette amélioration ...

```
5/5 [==============================] - 2s 490ms/step - loss: 0.6741 - accuracy: 0.5754 - val_loss: 0.5562 - val_accuracy: 0.8551
Epoch 4/20
5/5 [==============================] - 2s 476ms/step - loss: 0.6481 - accuracy: 0.6039 - val_loss: 0.6903 - val_accuracy: 0.6963
Epoch 5/20
5/5 [==============================] - 3s 659ms/step - loss: 0.6939 - accuracy: 0.5135 - val_loss: 0.6877 - val_accuracy: 0.8364
Epoch 6/20
5/5 [==============================] - 3s 531ms/step - loss: 0.6913 - accuracy: 0.5486 - val_loss: 0.6571 - val_accuracy: 0.8785
Epoch 7/20
5/5 [==============================] - 2s 465ms/step - loss: 0.6808 - accuracy: 0.5681 - val_loss: 0.5521 - val_accuracy: 0.8785
Epoch 8/20
5/5 [==============================] - 3s 618ms/step - loss: 0.6690 - accuracy: 0.6230 - val_loss: 0.5926 - val_accuracy: 0.9206
Epoch 9/20
5/5 [==============================] - 3s 580ms/step - loss: 0.6661 - accuracy: 0.5995 - val_loss: 0.5271 - val_accuracy: 0.8458
Epoch 10/20
5/5 [==============================] - 2s 491ms/step - loss: 0.6605 - accuracy: 0.6157 - val_loss: 0.5481 - val_accuracy: 0.8925
Epoch 11/20
5/5 [==============================] - 2s 496ms/step - loss: 0.6109 - accuracy: 0.6882 - val_loss: 0.2727 - val_accuracy: 0.8692
Epoch 12/20
5/5 [==============================] - 3s 540ms/step - loss: 0.8359 - accuracy: 0.5438 - val_loss: 0.5279 - val_accuracy: 0.9206
Epoch 13/20
5/5 [==============================] - 2s 484ms/step - loss: 0.6046 - accuracy: 0.7511 - val_loss: 0.4938 - val_accuracy: 0.9252
Epoch 14/20
5/5 [==============================] - 2s 493ms/step - loss: 0.5828 - accuracy: 0.7690 - val_loss: 0.3267 - val_accuracy: 0.9112
Epoch 15/20
5/5 [==============================] - 3s 544ms/step - loss: 0.6593 - accuracy: 0.6760 - val_loss: 0.2978 - val_accuracy: 0.8832
Epoch 16/20
5/5 [==============================] - 3s 568ms/step - loss: 0.7127 - accuracy: 0.6376 - val_loss: 0.7269 - val_accuracy: 0.1495
Epoch 17/20
5/5 [==============================] - 3s 561ms/step - loss: 0.6742 - accuracy: 0.5880 - val_loss: 0.4507 - val_accuracy: 0.8598
Epoch 18/20
5/5 [==============================] - 3s 522ms/step - loss: 0.6638 - accuracy: 0.6035 - val_loss: 0.5281 - val_accuracy: 0.9065
Epoch 19/20
5/5 [==============================] - 2s 499ms/step - loss: 0.5706 - accuracy: 0.7099 - val_loss: 0.7484 - val_accuracy: 0.5888
Epoch 20/20
5/5 [==============================] - 3s 558ms/step - loss: 0.6607 - accuracy: 0.5782 - val_loss: 0.3497 - val_accuracy: 0.9439
CPU times: user 4min 37s, sys: 4.63 s, total: 4min 42s
Wall time: 53.3 s
```

```
[97]: %%time
      model.fit( train_generator, validation_data = validation_generator ,epochs=10)

      Epoch 1/10
      5/5 [==============================] - 3s 520ms/step - loss: 4.8965 - accuracy: 0.4339 - val_loss: 6.8623 - val_accuracy: 0.1449
      Epoch 2/10
      5/5 [==============================] - 2s 369ms/step - loss: 4.7737 - accuracy: 0.4312 - val_loss: 6.9785 - val_accuracy: 0.1449
      Epoch 3/10
      5/5 [==============================] - 2s 355ms/step - loss: 5.0116 - accuracy: 0.3972 - val_loss: 6.9213 - val_accuracy: 0.1449
      Epoch 4/10
      5/5 [==============================] - 2s 342ms/step - loss: 4.9242 - accuracy: 0.4063 - val_loss: 6.9735 - val_accuracy: 0.1449
      Epoch 5/10
      5/5 [==============================] - 2s 348ms/step - loss: 4.9396 - accuracy: 0.4029 - val_loss: 6.8939 - val_accuracy: 0.1449
      Epoch 6/10
      5/5 [==============================] - 2s 346ms/step - loss: 4.6180 - accuracy: 0.4459 - val_loss: 6.8021 - val_accuracy: 0.1449
      Epoch 7/10
      5/5 [==============================] - 2s 361ms/step - loss: 5.1079 - accuracy: 0.3737 - val_loss: 6.8693 - val_accuracy: 0.1449
      Epoch 8/10
      5/5 [==============================] - 2s 327ms/step - loss: 4.6398 - accuracy: 0.4354 - val_loss: 6.8449 - val_accuracy: 0.1449
      Epoch 9/10
      5/5 [==============================] - 2s 360ms/step - loss: 4.4185 - accuracy: 0.4592 - val_loss: 6.7270 - val_accuracy: 0.1449
      Epoch 10/10
      5/5 [==============================] - 2s 350ms/step - loss: 4.4830 - accuracy: 0.4535 - val_loss: 6.7136 - val_accuracy: 0.1449
      CPU times: user 1min 9s, sys: 1.55 s, total: 1min 10s
      Wall time: 18.6 s
[97]: <tensorflow.python.keras.callbacks.History at 0x18add69a0>
```

Conclusion et améliorations :

Mon premier modèle de CNN a me semble-t-il de meilleurs résultats.
Mes principales difficultés ont été dans l'affichage de l'image en front et trouver les meilleurs paramètres pour mon CNN. J'ai passé beaucoup plus de temps sur le scraping que je ne l'avais prévu.

En ce qui concerne les améliorations :
Dans le cas où une API effectue des tâches qui prennent du temps (dans notre cas, la lecture d'image n'est pas aussi rapide que l'envoi d'un nombre comme paramètre), l'asynchrone permet de traiter plusieurs requêtes en même temps en dissociant le traitement de la tâche du thread principal.
Voici quelques pistes pour la suite:
Suivre les performances techniques (usage de la RAM ou du CPU, temps de réponse) mais aussi fonctionnelles : est-ce que les performances du modèle sont bonnes dans le temps ? Y a-t-il une dérive des données ? Pour cela, on peut déjà simplement logger les prédictions dans des des fichiers, et les analyser a posteriori.
Comparer un nouveau modèle à celui actuellement en production. La plupart des plateformes cloud permettent de faire cela grâce au canary release (Un pourcentage des requêtes est dirigé vers le nouveau modèle afin d'évaluer ses performances en production).
Dans la continuité du point précédent, si on modifie notre API, il est parfois nécessaire de la versionner (ex: http://api/v1/predict vs http://api/v2/predict) pour que l'ancienne version reste accessible.
Pour améliorer la sécurité de l'API, on peut ajouter de l'authentification pour s'assurer que les personnes qui requêtent l'API en ont bien le droit.