2021/2022

esprit

Se former autrement

Rapport de stage ingénieur

Reconnaissance d'images RVM

Réalisé par :

-Abbour Maroua

-Stiti Bacem

Encadré par : Madame Khiari Nefissa

Effectué au sein de : ESPRIT-Tech

ECOLE SUPÉRIEURE PRIVÉE D'INGÉNIERIE ET DE TECHNOLOGIES

EUR-ACE® Délivrée par la Commission des Titres d'Ingénieur

CONFÉRENCE DES GRANDES ÉCOLES

cdio

AUF

Liste des Tableaux

Liste des Figures

Table des matières

Introduction générale

Un Reverse Vending System est une machine qui récupère les contenants de boissons vides que les gens retournent afin de le recycler.

L'objectif de ce projet est de développer une solution automatique de reconnaissance d'images à implanter dans une RVM (Reverse Vending Machine).

Le présent rapport se compose de quatre chapitres :

- Le premier chapitre intitulé « Contexte général du projet » présente donne une vision générale sur le RVM ainsi que la gestion du projet (l'avancement du projet dans le temps ...)
- Le second chapitre qui s'intitule « Outil et Technologies » donne une vision détaillée sur les différents outils et technologies utilisés ainsi que l'installation de l'API de détection d'objets TensorFlow.
- Le troisième chapitre intitulé « Préparation des données » montre la collecte des données, le prétraitement des données pour les modèles de classification ainsi que le prétraitement des données pour les modèles de détection.
- Le dernier chapitre intitulé « Modélisation » est consacré à la description des modèles de classification et des modèles de détections et la présentation est discussion du Résultats des modèles de détection d'objets.

Chapitre 1 : Contexte générale du projet

1. Introduction :

Un Reverse Vending System est une machine qui récupère les contenants de boissons vides que les gens retournent, comme des bouteilles et des canettes, afin de les recycler. La machine habituellement, redonne le montant de la consigne ou du remboursement au dernier utilisateur. C'est une machine « inversée », au lieu d'introduire de l'argent dans la machine pour obtenir le produit désiré comme dans les distributeurs de confiserie, l'usager introduit le produit et récupère sa valeur monétaire en argent.

Figure 1: Reverse Vending System

Les systèmes de récupération automatisée sont une méthode pour récupérer, trier et traiter automatiquement les contenants de boissons vides qui sont retournés. La première machine de récupération entièrement automatisée a été créée par TOMRA en 1972. Ces automates sont surtout présents dans les régions où les dispositifs de retour des contenants permettent de récupérer de l'argent en retournant des contenants spécifiques, ou lorsque la législation impose le recyclage obligatoire.

Elles sont aussi connues comme étant des « machines à déconsigner », « machines de retour et de recyclage », « machines à recycler les canettes et les bouteilles » ou RVS « Reverse Vending System ».

2. Processus du RVM :

Le recyclage de vos contenants vides s'effectue en 3 étapes simples :

- Introduire votre contenant vide dans l'ouverture située sur le devant de l'automate.
- . Lorsque vous avez fini d'insérer tous vos contenants, presser le bouton sur le devant de la machine
- Récupérer le reçu que la machine a imprimé. Là où le dispositif de retour des contenants le prévoit, le reçu indique le montant de la consigne qui peut être remboursé à la caisse.

Figure 2 : Processus du RVM

Lorsque vous introduisez votre contenant, la machine numérise son code-barres, son matériau et sa forme, pour identifier le type d'emballage et vous donner le remboursement exact. La machine va donc trier les contenants en différentes catégories. En fonction des contenants existant dans votre région, ceux qui sont réutilisables vont dans une autre section que ceux qui ne peuvent être réutilisés et qui sont broyés et récupérés dans des bacs différents.

3. La gestion du projet :

Pour répartir les tâches entre les membres du notre groupe ainsi que l'avancement dans le temps on a choisi le digramme de Gantt avec l'outil lucidchart.

Le diagramme de Gantt est un outil utilisé en ordonnancement et en gestion de projet et permettant de visualiser dans le temps les diverses tâches composant un projet. Il s'agit d'une représentation d'un graphe connexe qui permet de représenter graphiquement l'avancement du projet.

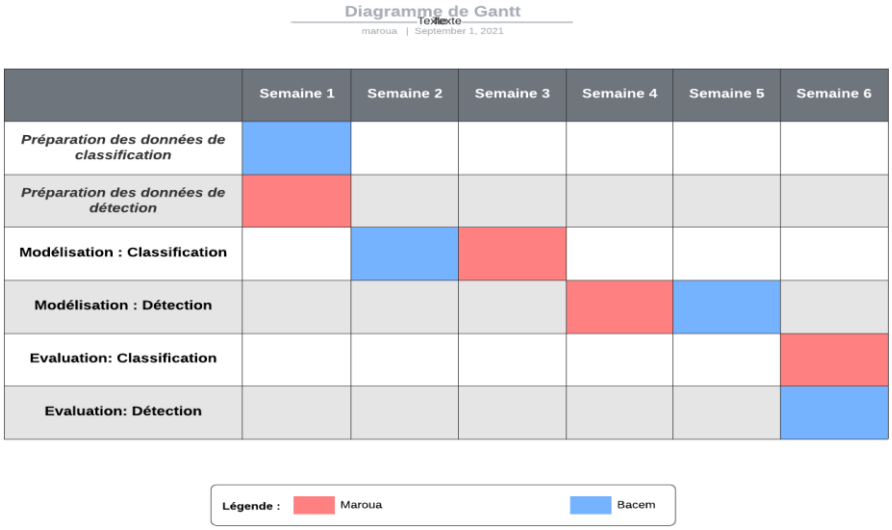

Figure 3: Diagramme de gantt

4. Conclusion :

Ce premier chapitre a présenté le contexte général du projet, notamment un bref descriptif de fonctionnement de RVM (Reverse Vending Machine) ainsi que la démarche adoptée pour la conduite du projet et son planning prévisionnel de réalisation.

Chapitre 2 : Outil et Technologies

1. Introduction :

Ce chapitre porte sur les outils et les technologies que nous avons utilisés dans le cadre de ce projet.

2. Technologies :

A. Google Colab :

Google Colab est un environnement de notebook Jupyter gratuit de Google dont le runtime est hébergé sur des machines virtuelles sur le cloud.

Les environnements d'exécution GPU et TPU gratuits, et le bloc-notes est préinstallé avec des modules de machine et d'apprentissage en profondeur tels que Scikit-learn et Tensorflow.

Figure 4: Google Colab

B. Python :

Python est le langage de programmation open source le plus employé par les informaticiens. Ce langage s'est propulsé en tête de la gestion d'infrastructure, d'analyse de données ou dans le domaine du développement de logiciels. En effet, parmi ses qualités, Python permet notamment aux développeurs de se concentrer sur ce qu'ils font plutôt que sur la manière dont ils le font. Il a libéré les développeurs des contraintes de formes qui occupaient leur temps avec les langages plus anciens. Ainsi, développer du code avec Python est plus rapide qu'avec d'autres langages.

Il reste aussi accessible pour les débutants, à condition de lui consacrer un peu de temps pour la prise en main. De nombreux tutoriels sont d'ailleurs disponibles pour l'étudier sur des sites Internet spécialisés ou sur des comptes Youtube. Sur les forums d'informatique, il est toujours possible de trouver des réponses à ses questions, puisque beaucoup de professionnels l'utilisent.

Figure 5: Python

C. Tensorflow :

TensorFlow est une bibliothèque open source de Machine Learning, créée par Google, permettant de développer et d'exécuter des applications de Machine Learning et de Deep Learning.

Figure 6: Tensorflow

D. Keras :

Keras est une bibliothèque open source de composants de réseaux neuronaux écrits en Python. Composée d'une bibliothèque de composants d'apprentissage automatique couramment utilisés, notamment des objectifs, des fonctions d'activation et des optimiseurs, la plate-forme open source de Keras prend également en charge les réseaux de neurones récurrents et convolutifs.

Figure 7: Keras

3. Installation de l'API de détection d'objets TensorFlow :

A. Téléchargement du TensorFlow Model Garden :

L'installation de l'API de détection d'objets est réalisée en installant l'object-détection package.

```
!git clone https://github.com/tensorflow/models.git
```

B. Installation/Compilation de Protobuf :

L'API de détection d'objets Tensorflow utilise Protobufs (Protocol Buffers) pour configurer les paramètres de modèle et d'entraînement. Avant que le framework puisse être utilisé, les bibliothèques Protobuf doivent être téléchargées et compilées.

Cela devrait être fait comme suit :

```
!protoc object_detection/protos/*.proto --python_out=.
```

C. Installation de l'API COCO :

API COCO (Common Objects in Context) est un ensemble de données de détection et de segmentation d'objets.

```
!git clone https://github.com/cocodataset/cocoapi.git
```

```
cd cocoapi/PythonAPI
```

```
/content/drive/My Drive/train_demo/models/research/cocoapi/PythonAPI
```

```
!make
```

D. Installer l'API de détection d'objets :

L'installation de l'API de détection d'objets est réalisée en installant le object_detection package.

Cela se fait en exécutant les commandes suivantes depuis l'intérieur Tensorflow\models\research :

```
cp object_detection/packages/tf2/setup.py .
```

```
#!python -m pip install .
!python -m pip install --use-feature=2020-resolver .
```

E. Tester l'installation :

Pour tester l'installation, on a exécuté la commande suivante depuis l'intérieur Tensorflow\models\research :

```
!python object_detection/builders/model_builder_tf2_test.py
```

4. Conclusion :

Ce chapitre a été consacré aux outils utilisés, ainsi que les détails de leurs installations.

Chapitre 3 : Préparation des données

1. Introduction :

Dans ce chapitre, nous passerons à la préparation des données en les diverses étapes prévues pour assurer la réalisation de ce projet.

2. Collecte des données :

Pour la collecte de nos données, nous avons une base de données qui contient des images des bouteilles et des cannettes qui sont bien organisées par types et par contenances.

Nous avons huit classes :

- Bouteille en plastique de 30cl
- Bouteille Plastique de 50cl
- Bouteille Plastique de 1.5l
- Bouteille Plastique de 1l
- Bouteille Plastique de 2l
- Canettes en aluminium
- Bouteilles en verre

	PLASTIC 0.3L	**PLASTIC 0.5L**	**PLASTIC 1L**	**PLASTIC 1.5L**	**PLASTIC 2L**	**CAN**	**GLASS**
TRAIN	32	62	126	102	59	165	36
TEST	8	24	29	21	12	42	8

Tableau 1 Nombre d'images par classe

3. Description des données et prétraitement :

Le preprocessing ou le prétraitement des données est une étape indispensable pour assurer le bon fonctionnement de modèle prédictif.

A. Prétraitement des données pour les modèles de classification :

Pour commencer on a séparé les images que l'on dispose en images qui seront utilisées pour entrainer les modèles et en images qui seront utilisées pour évaluer les performances des modèles qu'on a déjà entrainé, ensuite dans les dossiers 'test' et 'train' on a créé d'autres dossiers qui portent les noms des classes de nos images où chaque image sera déplacée dans le dossier portant le même nom que sa classe.

Les noms des dossiers qu'on vient de citer vont nous servir pour labeliser nos données.

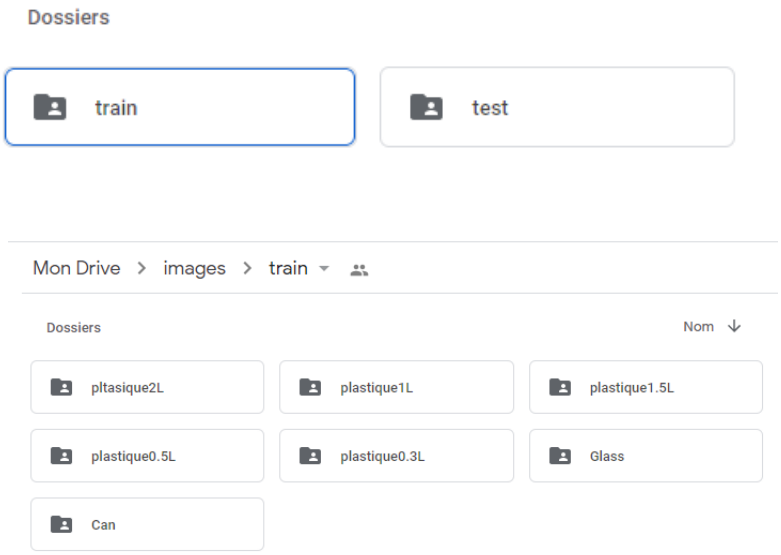

Figure 8: Labels

Les images seront chargées dans deux listes une première liste qui contient les images et une deuxième liste qui contient les labels, un label n'est rien d'autre que le nom du dossier à partir duquel on a récupéré l'image.

L'étape suivante consiste à redimensionner toutes nos images pour qu'elles aient toutes la même taille.

La dernière étape à ce niveau serait de convertir chaque image en trois tableaux RGB.

```
[ ]  #Capture training data and labels into respective lists
     train_images = []
     train_labels = []
```

```
for directory_path in glob.glob("/content/drive/MyDrive/images/train/*"):
    label = directory_path.split("/")[-1]
    print(label)
    for img_path in glob.glob(os.path.join(directory_path, "*.jpeg")):
        print(img_path)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
        train_labels.append(label)
```

Figure 9: Code de prétraitement des données

B. Prétraitement des données pour les modèles de détection d'objets :

Pour le prétraitement des données relatif à la détection d'objets on a utilisé une bibliothèque de Python qu'on appelle 'labelimg' qu'on peut manipuler à l'aide d'une interface graphique, grâce à 'labelimg' on peut repérer l'objet qu'on souhaite détecter en l'encadrant et en lui attribuant un label.

Un fichier xml va être automatiquement généré, ce fichier contient les coordonnées de l'objet qu'on a encadré ainsi que le nom de l'image et le label de l'objet.

Figure 10: Labélisation des images avec labelimg

02_03_2020_14_33_29.xml

C: > Users > Bacem > Desktop > images > train > 02_03_2020_14_33_29.xml

```xml
<annotation>
    <folder>img</folder>
    <filename>02_03_2020_14_33_29.jpeg</filename>
    <path>C:\Users\Bacem\Desktop\img\02_03_2020_14_33_29.jpeg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>350</width>
        <height>768</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>canette</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>92</xmin>
            <ymin>437</ymin>
            <xmax>283</xmax>
            <ymax>704</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 11: Fichier XML

4. Conclusion :

Ce chapitre a été consacré aux prétraitements des données pour les modèles de détection d'objets et les modèles de classification d'objets.

Chapitre 4 : Modélisation

1. Introduction :

Après la phase de prétraitement vient la phase de modélisation, dans cette partie nous allons présenter les modèles de classification et de détection d'objets.

2. Modèles de classification :

On a entrainé trois modèles de classification d'images qui sont : XGboost, Random Forest et Logistic regression, en ce qui concerne la partie de 'feature engineering ' vu qu'on travaille sur des images c'est trop compliqué de pouvoir déterminer les caractéristiques manuellement c'est la raison pour laquelle on a fait appel au Deep learning et à la bibliothèque 'Tensorflow '.

L'idée est d'utiliser des caractéristiques d'un modèle qui a déjà était entrainé sur une autre dataset, c'est ce qu'on appelle le 'transfer learning', et d'extraire ces mêmes caractéristiques sur nos images et les utiliser pour l'entrainement des modèles qu'on a mentionné.

Pour ce faire on a choisi d'utiliser le modèle 'VGG16' qui est entrainé sur la fameuse base de données 'imagenet'.

```
[ ]  #Load a trained model to use its feautures
     VGG_model = VGG16(weights='imagenet', include_top=False, input_shape=(SIZE, SIZE, 3))

     Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
     58892288/58889256 [==============================] - 1s 0us/step
     58900480/58889256 [==============================] - 1s 0us/step

[ ]  #Make loaded layers as non-trainable. This is important as we want to work with pre-trained weights
     for layer in VGG_model.layers:
       layer.trainable = False

[ ]

[ ]  #Now, let us use features from convolutional network
     feature_extractor=VGG_model.predict(x_train)
```

Figure 12: VGG16

A. Random Forest :

L'algorithme des « forêts aléatoires » (ou Random Forest parfois aussi traduit par forêt d'arbres décisionnels) est un algorithme de classification qui réduit la variance des prévisions d'un arbre de décision seul, améliorant ainsi leurs performances. Pour cela, il combine de nombreux arbres de décisions dans une approche de type bagging.

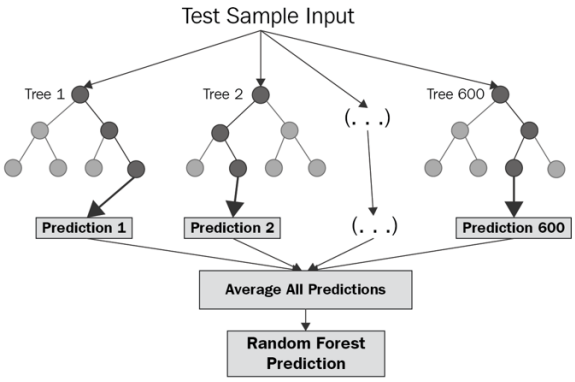

Figure 13: Random Forest

- **Réglage des hyperparameters:**

Pour trouver les meilleurs hyperparamètres, on a utilisé le 'Grid search' , l'idée est simple on doit créer un dictionnaire ayant en clef le nom de l'hyperparamètre à ajuster et en valeur une liste de ces valeurs possibles.

Pour chaque combinaison on va entraine le modèle et calculer son score pour choisir à la fin la meilleure combinaison.

Les images ci-dessous montrent les hyperparamètres à ajuster.

```
[ ]
    param_grid = {
        'n_estimators': [200, 500],
        'max_features': ['auto', 'sqrt', 'log2'],
        'max_depth' : [4,5,6,7,8],
        'criterion' :['gini', 'entropy']
    }
```

Apres l'entrainement du modèle en utilisant toutes les combinaisons possibles on a eu les valeurs qui ont donné la meilleure performance.

```
[ ]  rfc=RandomForestClassifier()
     CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
     CV_rfc.fit(X_for_training, y_train)
```

En entrainant cet algorithme sur nos données on a eu une 'accuracy ' de 0.9 sur les données de test.

```
#Print overall accuracy
from sklearn import metrics
print ("Accuracy = ", metrics.accuracy_score(test_labels, prediction_RF))
```

```
Accuracy =  0.9027777777777778
```

Figure 14: Random Forest's accuracy

B. XGBoost :

XGBoost est un algorithme d'apprentissage machine basé sur un arbre de décision d'ensemble et utilise un cadre de renforcement de gradient. Les réseaux neuronaux artificiels sont généralement plus performants que d'autres cadres ou algorithmes lorsqu'il s'agit de prédire des problèmes avec le texte, les images et d'autres données non structurées.

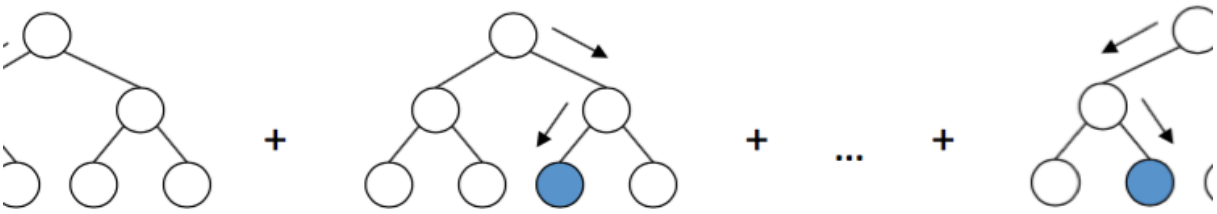

Figure 15: XGBoost

❖ **Réglage des hyperparamètres :**

La capture d'écran suivante montre les hyperparamètres à ajuster :

```
# A parameter grid for XGBoost
params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }
```

Après les itérations faites grâce à 'GridSearchCV' on a obtenu les meilleures valeurs des hyperparamètres qu'on vient de lister.

```
[ ]  print('\n Best hyperparameters:')
     print(random_search.best_params_)
```

```
 Best hyperparameters:
{'subsample': 0.6, 'min_child_weight': 5, 'max_depth': 5, 'gamma': 1, 'colsample_bytree': 1.0}
```

En entrainant cet algorithme sur nos données on a eu une 'accuracy ' de 0.9 sur les données de test.

```
[ ]  #Print overall accuracy
     from sklearn import metrics
     print ("Accuracy = ", metrics.accuracy_score(test_labels, prediction))
```

```
Accuracy =  0.9097222222222222
```

Figure 16: XGBoost's accuracy

C. Logistic regression :

La régression logistique est une technique prédictive. Elle vise à construire un modèle permettant de prédire / expliquer les valeurs prises par une variable cible qualitative (le plus souvent binaire, on parle alors de régression logistique binaire ; si elle possède plus de 2 modalités, on parle de régression logistique polyatomique) à partir d'un ensemble de variables explicatives quantitatives ou qualitatives (un codage est nécessaire dans ce cas).

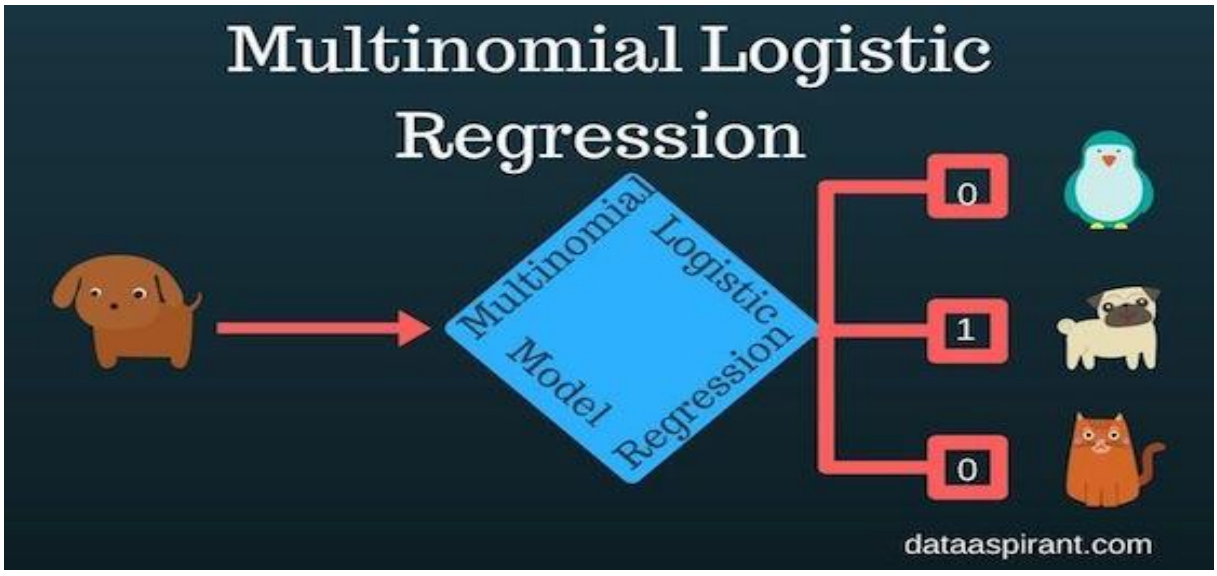

Figure 17: Logistic regression

❖ **Réglage des hyperparamètres :**

La capture d'écran suivante montre les hyperparamètres à ajuster :

```
[ ]  grid={"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}
```

```
[ ]  log_reg=LogisticRegression()
     logreg_cv=GridSearchCV(log_reg,grid)
     logreg_cv.fit(X_for_training, y_train)
```

Après les itérations faites grâce à 'GridSearchCV' on a obtenu les meilleures valeurs des hyperparamètres qu'on vient de lister.

```
[ ]  print('\n Best hyperparameters:')
     print(logreg_cv.best_params_)
```

```
 Best hyperparameters:
{'C': 0.01, 'penalty': 'l2'}
```

En entrainant cet algorithme sur nos données on a eu une 'accuracy ' de 0.89 sur les données de test qui est dans le même ordre de grandeur que celle qu'on a eu avec XGboost.

```
#Print overall accuracy
from sklearn import metrics
print ("Accuracy = ", metrics.accuracy_score(test_labels, prediction_logreg))
```

```
Accuracy =  0.8958333333333334
```

Figure 18: Logistic regression' accuracy

3. Présentation et discussion du Résultats des modèles de classification :

Les algorithmes qu'on a entrainés ont donné des bons résultats sur les images de test.

La métrique qu'on a utilisée pour les évaluer est l'accuracy, les valeurs de l'accuracy sont très proches pour les trois modèles.

- Random Forest: 0.9
- Logistic regression: 0.89
- XGBoost :0.9

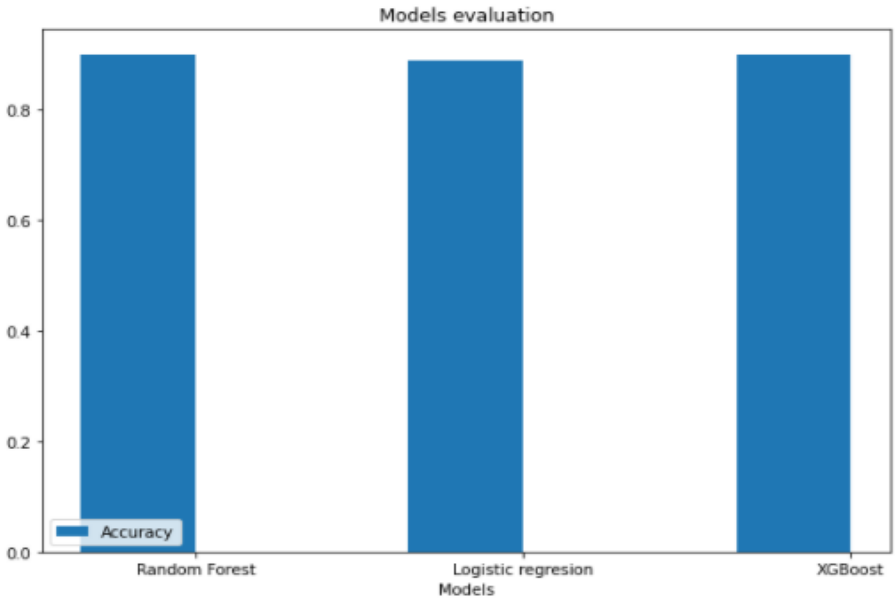

Figure 19: Models Evaluation

Random Forest et XGBoost ont la même valeur d'accuracy ce qui nous a mené à calculer la vitesse d'entrainement pour chaque modelé afin d'en choisir un.

Random Forest a mis uniquement deux secondes pour la phase d'entrainement.

```
[32] # Train the model on training data
     print("'Début de l'entrainement :   ",datetime.now().strftime("%H:%M:%S"))
     RF_model.fit(X_for_training, y_train) #For sklearn no one hot encoding
     print("'Fin de l'entrainement :   ",datetime.now().strftime("%H:%M:%S"))

     'Début de l'entrainement :    16:30:21
     'Fin de l'entrainement :    16:30:23
```

Figure 20: Temps d'entrainement de Random Forest

Alors que XGBoost était beaucoup trop long, il lui faudrait 4minutes et 4 secondes pour la phase d'entrainement.

```
29]
   model = xgb.XGBClassifier(subsample=0.6,min_child_weight=5,gamma=1,colsample_bytree=1)
   print("'Début de l'entrainement :   ",datetime.now().strftime("%H:%M:%S"))
   model.fit(X_for_training, y_train)
   print("'Fin de l'entrainement :   ",datetime.now().strftime("%H:%M:%S"))

   'Début de l'entrainement :    16:23:45
   'Fin de l'entrainement :    16:27:49
```

Figure 21: Temps d'entrainement de XGBoost

En prenant en compte le critère de précision et de vitesse 'Random Forest' est l'algorithme qui a le mieux performé sur nos images.

4. Modèles de détection d'objets :

Pour entrainer des modèles de détection d'objets on a utilisé une API de Tensorflow nommée TFOD (Tensorflow Object Détection), grâce à cette API on peut utiliser des modèles pré-entrainés sur la fameuse dataset 'COCO' qu'on peut télécharger depuis 'Tensorflow model zoo '.

Dans le cadre de ce projet on a téléchargé des modèles et on les a ré-entrainé sur nos propres images.

A. CenterNet Resnet 101 :

Une fois installée, l'utilisation de l'API TFOD est simple la configuration du modele se fait dans deux fichiers.

Le premier fichier nommé 'label_map.pbtxt' c'est le fichier qui va contenir les différentes classes des objets qu'on souhaite détecter sur nos images.

```
item {
    id: 1
    name: 'Bouteille_0.3L'
}

item {
    id: 2
    name: 'Bouteille_0.5L'
}

item {
    id: 3
    name: 'Bouteille_1.5L'
}

item {
    id: 4
    name: 'Bouteille_1L'
}

item {
    id: 5
    name: 'Bouteille_2L'
}

item {
    id: 6
    name: 'Canette'
}
```

Figure 22: Label_map.pbtxt

Le deuxième fichier nommé 'pipeline.config' contient beaucoup de configurations liées au processus d'entrainement du modèle tel que :

- Les chemins vers les données de test et d'entrainement.
- Les paramètres de 'Data augmentation' s'il y'en a besoin.
- Le batch size.
- Le nombre d'étapes lors de l'entrainement.
- Les paramètres utilisés pour évaluer le modèle.

```
eval_input_reader: {
  label_map_path: "/content/drive/MyDrive/train_demo/annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "/content/drive/MyDrive/train_demo/annotations/test.record"
  }
```

```
train_input_reader: {
  label_map_path: "/content/drive/MyDrive/train_demo/annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "/content/drive/MyDrive/train_demo/annotations/train.record"
  }
}

eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  batch_size: 1;
```

B. CenterNet resnet50 v1 :

- **Création du Label Map :**

C'est le fichier 'label_map.pbtxt' comme il est montré ci-dessus qui doivent être placés dans le dossier training_demo/annotations.

- **Création TensorFlow Records :**

Après avoir généré nos annotations, nous avons converti nos annotations dans ce que l'on appelle le TFRecord format.

- **Configuration d'une tâche d'entraînement :**

On a commencé par le téléchargement du modèle pré-entraîné, après on a fait quelques modifications au niveau du fichier pipeline.config, tel que le nombre des classes et les chemins vers les données de test et d'entrainement.

5. Présentation et discussion du Résultats des modèles de détection d'objets :

	AVERAGE RECALL	AVERAGE PRECISION	MAP
CENTERNET RESNET 101	0.894	0.878	0.877
CENTERNET RESNET 50	0.880	0.857	0.856

Tableau 2: Résultats obtenus sur les modèles

Le tableau ci-dessus montre le résultat de chaque modèle. Les résultats obtenus sont exprimés en termes d'average recall, average précision et de MAP. Ceci revient à la grande dimension de la base ce qui nécessite l'utilisation d'un GPU au lieu d'un CPU.

- Les meilleurs résultats présentés sur le modèle Centernet resnet 101.

6. Conclusion :

Dans ce chapitre nous avons détaillé la partie de modélisation en illustrant et détaillant les fichiers utilisés ainsi que les résultats obtenus.

Conclusion générale et perspectives

Notre projet mené au sein d'Esprit-tech est une opportunité de prendre part à un projet innovant qui s'inscrit dans le domaine de Deep Learning.

Notre mission consistait à développer une solution capable de détecter les images dans un RVM en se basant sur des modèles de Deep Learning.

Malgré les difficultés que nous avons rencontrées durant ce projet, qui résident essentiellement dans la nouveauté des outils avec lesquels nous avons travaillé, nous sommes finalement parvenus à nous familiariser avec ces nouvelles techniques.

Bibliographie & Webographie

https://towardsdatascience.com/tagged/google-colab

https://www.lebigdata.fr/tensorflow-definition-tout-savoir

https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445304-python-definition-et-utilisation-de-ce-langage-informatique/

https://deepai.org/machine-learning-glossary-and-terms/keras

https://www.dad3zero.net/201701/renommer-photos-masse-python/