



PROJET CHEF D'OEUVRE

DUMPIN

—
Avril 2021
—

SEVERINO NICOLAS



ECOLE IA MICROSOFT POWERED BY SIMPLON

TABLE DES MATIÈRES

INTRODUCTION	2
PROBLÉMATIQUE	2
OBJECTIFS	2
CONTEXTE	3
ÉTAT DE L'ART	3
SPÉCIFICATIONS FONCTIONNELLES	3
GESTION DE PROJET	4
INTELLIGENCE ARTIFICIELLE	5
DATASETS	5
Collecte	5
Analyse Exploratoire des Données	6
Stockage en base de données	7
MÉTHODE	7
Baseline de Machine Learning	7
Feature engineering	8
Baseline de Deep Learning	9
SOLUTION DE DEEP LEARNING RETENUE	10
APPLICATION	16
BASE DE DONNÉES	16
Conception	16
Sauvegarde de la base de données	18
Population et utilisation	18
Optimisation	19
FRONT	19
Librairies utilisées et versions	19
Parcours utilisateur	19
BACKEND	21
QUALITÉ ET MONITORING	21
Monitoring	21

CONCLUSION

22

INTRODUCTION

PROBLÉMATIQUE

La gestion efficace de nos déchets est l'un des plus grands défis de notre époque. Les nouvelles technologies numériques peuvent faciliter les process des collectivités, résidents et entreprises, en leur permettant de trouver de nouvelles réponses aux enjeux de la transition écologique et de la compétitivité mais également en améliorant, dans le même temps, leur gestion des coûts.

La quantité de déchets a doublé en 40 ans en France. Chaque Français en produit 573 kg par an selon l'Agence de l'Environnement et de la Maîtrise de l'Energie (Ademe). S'il faut avant tout veiller à en produire moins, notamment pour préserver les matières premières, il est aussi primordial de savoir où les jeter pour qu'ils ne finissent pas trop facilement incinérés ou stockés.

Bien que de nombreux néophytes opposent d'instinct progrès et nature, science et environnement, la révolution industrielle menée avec la digitalisation a dans le même temps permis l'essor de nouveaux business models. L'IA, également.

OBJECTIFS

Les applications qui vous aident à prendre de bonnes résolutions ont le vent en poupe, même dans le domaine du développement durable et du recyclage.

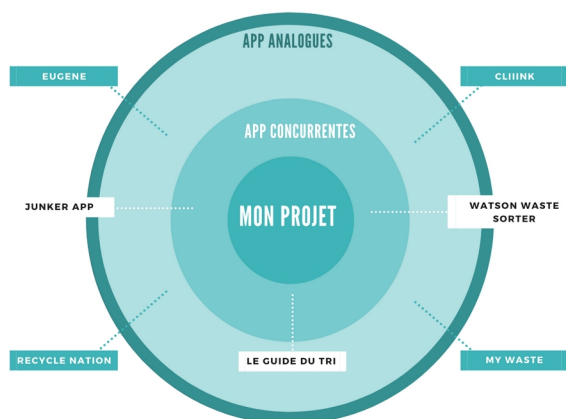
Il en existe actuellement des centaines notamment celles qui vous permettent de connaître les consignes locales, de mieux manger en recyclant, de relever des défis écolos, vous récompense quand vous trie, etc. mais actuellement, aucune ne permet d'apporter une réponse concrète à cette problématique éco-responsable, base de notre quotidien. L'IA, tout autant en vogue, n'y est d'ailleurs pas plus présente.

La distribution des poubelles de couleurs différentes (jaunes, vertes, grises, parfois bleues) qui sont distribuées aux particuliers pour jeter chaque déchet par famille afin qu'il reçoive le traitement qui lui est adapté et, si possible, qu'il soit recyclé, n'a pas arrangé l'affaire. Mais il existe tellement de cas pour lesquels un doute persiste, si le déchet est sale ou n'est pas entièrement recyclable par exemple. Dans quelle poubelle jeter ? Est-ce recyclable ? Et si je déménage, en sera-t-il de même ? Autant de questions qui alourdissent la charge mentale tant par leur redondance que par leur caractère systématique. D'où l'intérêt de l'application "Dumpin", qui propose de répondre à toutes ces questions par la création d'une application web (et mobile) avec une IA (computer vision avec classification multi-classes) pour détecter la matière de l'objet désiré et lui proposer le container adéquat en fonction de sa zone géographique.

CONTEXTE

Comment s'insérer dans le paysage des applications existantes et proposer une solution, à base d'IA, qui soit au cœur même de la vie de tous les jours et au plus près des réalités de tous les foyers dans leur impact environnemental : le tri des déchets domestiques.

ÉTAT DE L'ART



Application analogues (même domaine mais scope différent), par exemple :

- EUGENE : passe à la loupe les produits alimentaires et cosmétiques et indique comment les recycler.
- CLIINK : identifie les conteneurs environnants possibles pour jeter les emballages.

Application concurrentes

- JUNKER APP : Une application avec géolocalisation et lecteur de code-barres des produits pour les recycler selon les

règles propres aux différentes communes

- LE GUIDE DU TRI : L'utilisateur renseigne sa ville et le type d'emballage qu'il souhaite trier et l'application lui le conteneur approprié. L'application n'utilise pas d'IA car l'utilisateur renseigne tout lui-même.

SPÉCIFICATIONS FONCTIONNELLES

Je souhaite que mon utilisateur puisse accéder à mon application sur différents supports et scanner (sur mobile) ou uploader (sur ordinateur) une photo de la matière à détecter pour l'IA puisse l'identifier et le conseiller sur le container dans lequel la jeter.

Aussi, je souhaite que mon utilisateur puisse modifier la prédiction faite par l'IA. Une fois corrigée, je veux sauvegarder l'image et la prédiction (originale si correcte ou celle corrigée par l'utilisateur si incorrecte) pour enrichir le dataset d'entraînement.

TRADUCTION TECHNIQUE DU PROJET

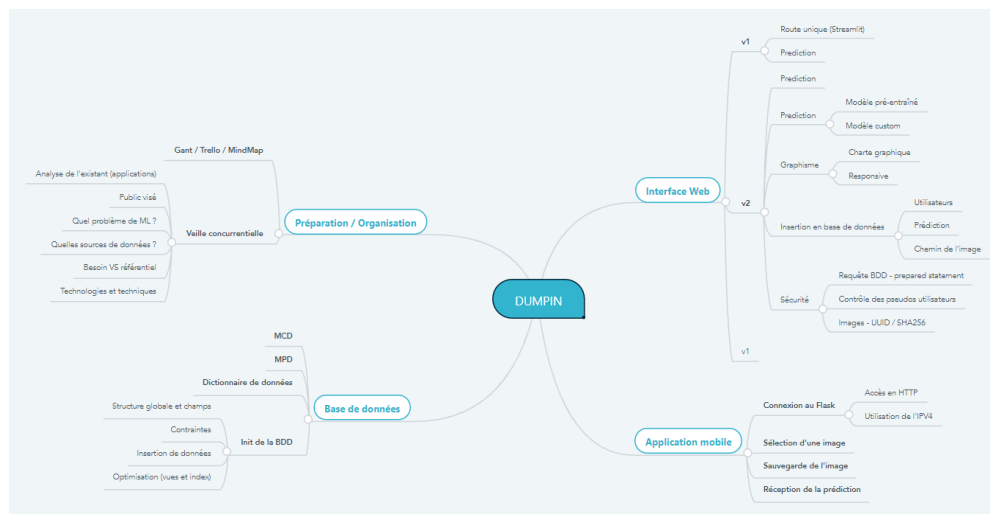
- Le besoin d'historisation formulée par le client, le réentraînement programmable du modèle de Deep Learning et la pluralité de politiques de gestion de déchets justifient le recours à une base de données de type relationnelle (stockage des

utilisateurs, des photos uploadées, des nomenclatures et des données d'entraînement) ;

- Le besoin d'accès permanent de l'utilisateur à l'application doit se faire à travers une interface web ainsi qu'une application mobile native (soit accès au stockage des photos depuis le téléphone soit scan du produit ou de son étiquette) ;
- Docker (avec son gestionnaire de containers Docker Desktop) est également utilisé pour être plus facilement déployé dans le Cloud (actuellement sur la plateforme HEROKU).
- Cron et mysql-cron-backup sont utilisés pour l'automatisation (de la sauvegarde de la base de données, de son alimentation en continu et du réentraînement du modèle). Jenkins sera également incorporé pour l'intégration continue.

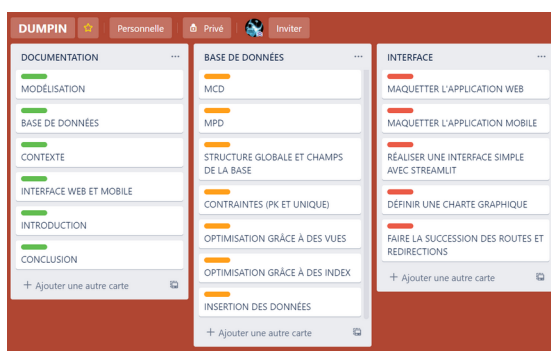
GESTION DE PROJET

MINDMAP



Un MindMap permettant de catégoriser et lister exhaustivement l'ensemble des fonctionnalités et tâches de mon application a été réalisé.

TRELLO





Un Kanban sur Trello a été initié avec un WIP limite de 5 (un pour chaque catégorie : back / front / BDD / documentation / IA)

GITHUB

Le projet a également été sauvegardé sur un repo GitHub dédié : <https://github.com/nicolasseverino/dumpin>

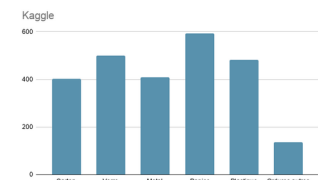

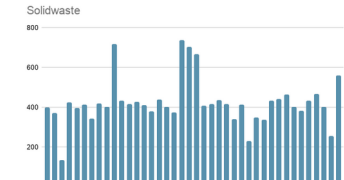
INTELLIGENCE ARTIFICIELLE

Le besoin client fait ressortir un problème de Machine Learning relevant de la classification d'images multi-classes couplée à de la vision par ordinateur (Computer Vision) pour l'identification des objets et leur matière.

DATASETS

Collecte

Les 3 datasets suivants ont été collectés en open data et utilisés pour le projet.

	KAGGLE	FLIKR MATERIAL DATABASE	SOLIDWASTE
Lien	https://www.kaggle.com/asdasdasdas/garbage-classification	https://people.csail.mit.edu/lavanya/fmd.html	https://github.com/qiuyeqiang/solid-waste-dataset
Nombre d'images	2527	1003	16979
Nombre de classes	6	10	40
Répartition des classes			



Caractéristiques des images	Photos avec fond uni	Données utilisateur sur le terrain	Mélange de photos prises en condition normale et photos "propres"
------------------------------------	----------------------	------------------------------------	---

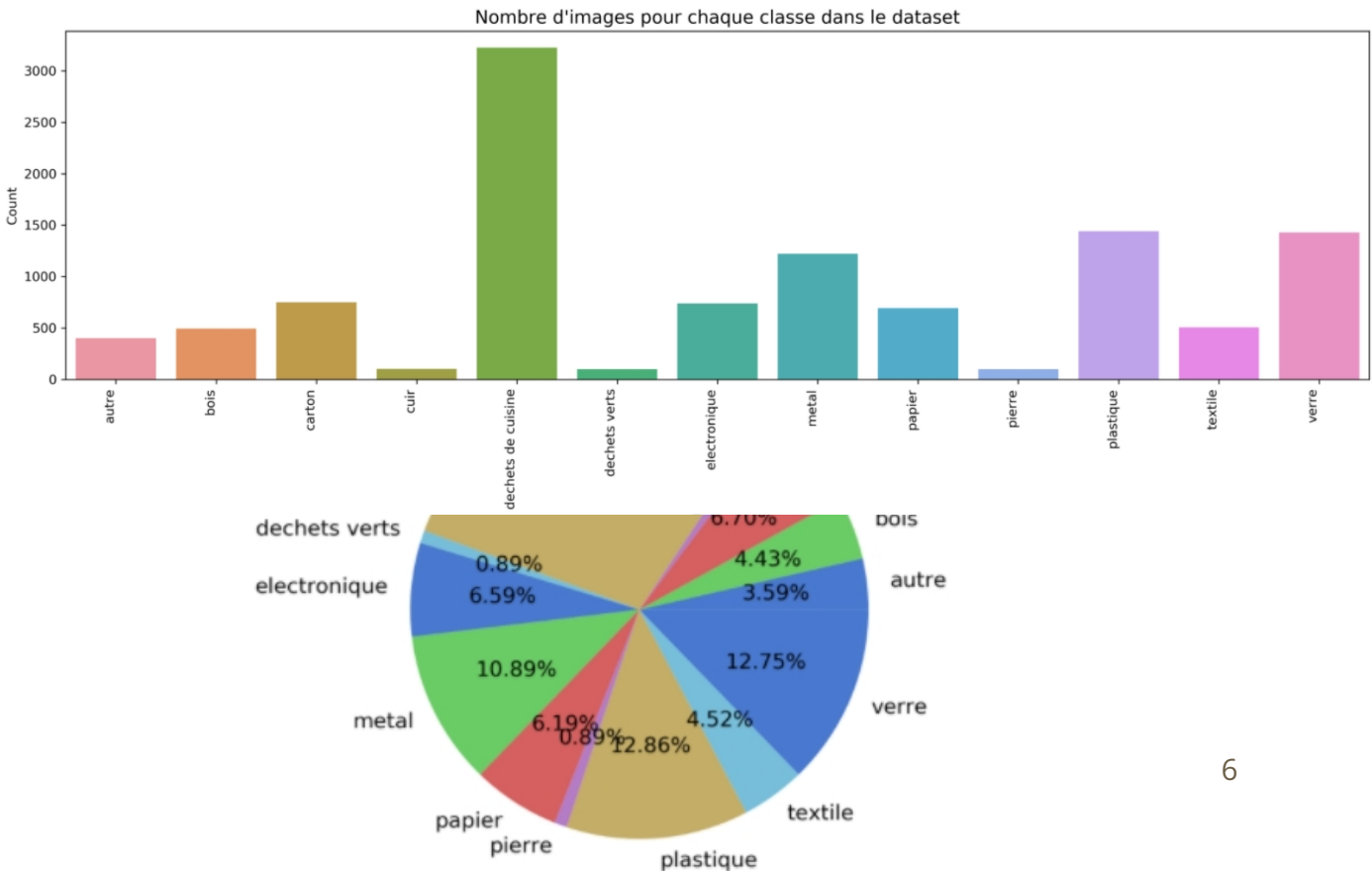
- Un nouveau dataset a donc été constitué à partir des trois précédents, afin de :
- garder un mélange d'images en condition réelle et d'images sous de belles conditions, et ainsi permettre un meilleur entraînement du modèle ;
 - augmenter la taille des classes principales ainsi que celle du dataset en général.

Celui-ci sera divisé en ensemble de train, validation et test pour l'entraînement de l'IA.

Caractéristiques du nouveau dataset :

- 12 000 images et 13 classes :
 - Bois
 - Carton
 - Cuir
 - Textile
 - Alimentaire
 - Déchets verts
 - Electronique
 - Metal
 - Papier
 - Pierre
 - Plastique
 - Verre
 - Autres (câbles, ...)
- Classes déséquilibrées: la classe la moins représentée (cuir / déchets verts ou pierre) comporte 100 éléments contre 3224 éléments pour les déchets de cuisine.

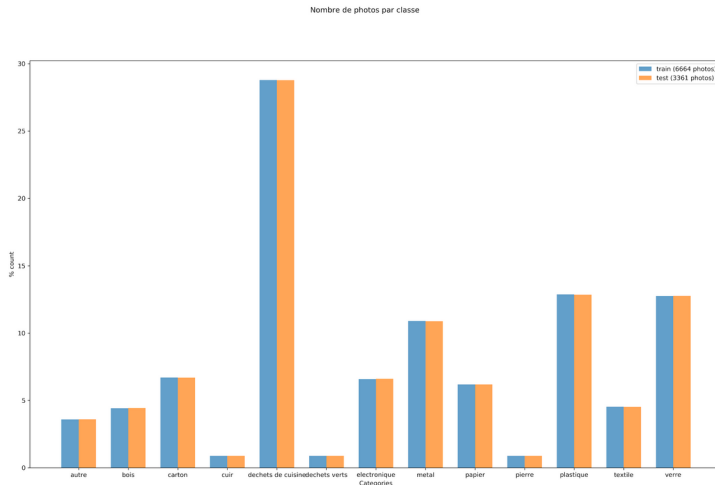
Analyse Exploratoire des Données





Stockage en base de données

Un split des données a été effectué (55% pour le train set, 15% pour l'ensemble de validation et 30% pour l'ensemble de test), en veillant à respecter la répartition des classes dans les ensembles de train et test (grâce à un stratify_split dans le train_test_split) :



Puis, les données sont transposées en DataFrame :

```
train_df = pd.DataFrame({'img_path':x_train,'img_class':y_train})  
train_df['subset'] = 'training'  
train_df
```

puis injectées grâce à PyMySQL et Pandas dans les tables "training_dataset" et "test_dataset".

```
train_df.to_sql(name='training_dataset', con=engine, if_exists = 'append', index=False)  
val_df.to_sql(name='training_dataset', con=engine, if_exists = 'append', index=False)  
test_df.to_sql(name='test_dataset', con=engine, if_exists = 'append', index=False)
```

MÉTHODE

Avant de me lancer dans la fastidieuse recherche d'optimisation des performances, j'ai d'abord établi deux baselines :

- La première utilisant le Machine Learning en confrontant une partie de mon dataset à divers classifieurs réputés en classification d'images multi-classes de la librairie Scikit Learn ;
- Une seconde utilisant le Deep Learning en confrontant ce même échantillon à différentes méthodes d'apprentissage automatique profond avec les librairies Keras et Tensorflow.

Baseline de Machine Learning

Nous avons vu précédemment que le dataset Kaggle a été incorporé dans le dataset final. Mais il a également permis d'établir les conclusions, en Machine Learning comme en Deep Learning, sur lesquelles ont reposés les choix pour le dataset final.

En effet, celui-ci est plus réduit en terme de taille (2527 éléments au lieu des 11202 éléments retenus) et plus accessible (Kaggle étant une plateforme bien connue de Data Science mettant à disposition en accès libre des datasets pour les compétitions qu'ils encadrent). Classifiées par classes, les photos étaient directement labellisées.

Étapes de la baseline :

- Après le chargement des photos, utilisation de la fonction `train_test_split` (70% pour le train, soit 1768 éléments, et 30% pour le test, soit 759 éléments) avec un stratify pour respecter la répartition des classes entre les ensembles de train et de test ;
- Standardisation sur le train et test ;
- Puis sélection de 10 classifieurs (dont notamment `SGDClassifier`, `SVC`, `XGBClassifier`, `RandomForestClassifier`)
- Entraînement de chacun des modèles d'apprentissage automatique et vérification des résultats de 3 validations croisées. Cette évaluation, configurée avec la même `Random seed`, permet de garantir que les mêmes fractionnements aux données d'apprentissage sont effectués et que chaque algorithme est évalué exactement de la même manière ;
- Visualisation des performances des différents classifieurs ;
- Au vue des métriques, le modèle le plus performant était le `RandomForest` (accuracy de 67%) ;
- Souhaitant obtenir une meilleure précision, 3 recherches d'hyper-paramètres ont été entreprises :
 - Par le biais des validation curves : accuracy de 68%
 - Par le biais d'un `RandomizedSearch`: accuracy de 69%
 - Par le biais d'un `GridSearch` : accuracy de 70%.

Même si les résultats semblent plutôt bons, une baseline basée sur le Deep Learning en a résulté pour essayer d'obtenir plus de précision.

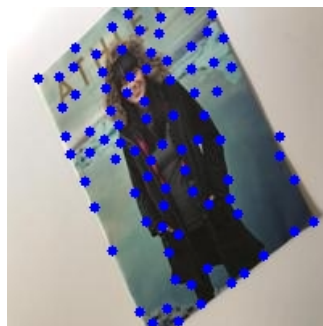
Feature engineering

Grâce à OpenCV, des techniques de vision par ordinateur pour la détection des caractéristiques ont été entreprises, notamment pour la détection des coins :

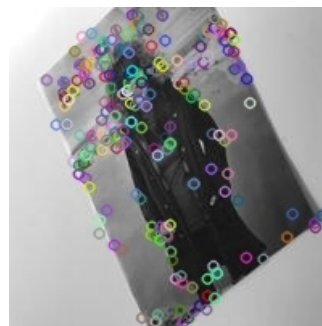
Harris



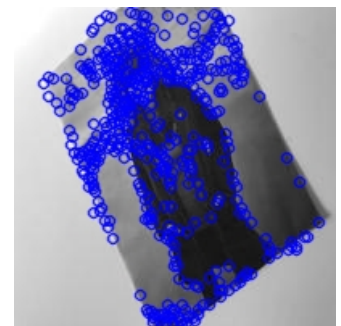
Shi-Tomasi



SIFT



FAST



Harris : utiliser fonction de fenêtre gaussienne pour détecter les coins ;
Shi-Tomasi : Même procédé que la détection de coins de Harris mais plus poussée ;
Scale-Invariant Feature Transform (SIFT) : Détection invariante à l'échelle contrairement aux 2 précédentes ;
Features from Accelerated Segment Test (FAST) : Il s'agit d'une technique de détection de coin beaucoup plus rapide que SURF, qui est elle même plus poussée que SIFT.

Baseline de Deep Learning

Basée sur ce même dataset, le baseline était l'occasion de tester et analyser différentes techniques d'IA : mon propre CNN, simple puis plus complexe, du transfer learning, ...

- 1) Split du dataset : train (70%), validation (15%) et test (15%) et stockage dans des dossiers dédiés et séparés également ;
- 2) Preprocessing des images : Notre ensemble d'apprentissage comprend 1766 images et l'ensemble de validation en comprend 111. Chaque image a une taille de 224 x 224 et a 3 dimensions (Rouge, Vert et Bleu - RGB). Nous allons échelonner chaque image de valeurs par pixel entre 0 et 255 à des valeurs entre 0 et 1 parce que les modèles de Deep Learning marchent mieux avec des petites valeurs d'entrée.
- 3) Traitement des labels : application d'un LabelEncoder (transformation des labels en chiffres): 0 pour bois, 1 pour carton,...
- 4) Création de différents modèles : CNN (**C**onvolutional **N**eural **N**etwork), simple puis complexe (avec ou sans data augmentation) / Transfer learning (avec ou sans data augmentation) / Fine-tuning (avec ou sans data augmentation)
- 5) Évaluation : Voici les conclusions des différentes combinaisons :

TYPE DE MODÈLE	SANS DATA AUGMENTATION	AVEC DATA AUGMENTATION	ÉVALUATION	COMMENTAIRES
CNN SIMPLE	X		OVERFIT	3 couches de convolution, avec des MaxPooling / 1 couche Flatten et 2 couches Dense.
CNN PLUS COMPLEXE	X		LÉGER OVERFIT	1 couche de convolution en plus / 1 autre couche cachée Dense / 1 dropout de 0.3 après chaque couche cachée Dense
CNN PLUS		X	PAS D'OVERFIT	Précision de 79%

COMPLEXE				
TRANSFER LEARNING	X		PAS D'OVERFIT	Précision de 65%
TRANSFER LEARNING		X	PAS D'OVERFIT	Précision de 76%
FINE TUNING		X	PAS D'OVERFIT	Précision de 86%

Suite à ce pipeline, la solution du Transfer Learning avec fine-tuning était donc à privilégier mais également à approfondir.

SOLUTION DE DEEP LEARNING RETENUE

Le dataset Kaggle ayant permis de relever l'intérêt d'explorer la solution du Transfer learning avec fine-tuning, un autre pipeline devait donc être mis en place. Compte tenu de la lourdeur de l'entraînement des modèles pré-entraînés et de la taille du dataset final, cette fois cela devait être effectué sur un échantillon du dataset final : trouver le modèle pré-entraîné et essayer de l'optimiser en recherchant certains hyper paramètres afin d'améliorer ses performances.

Pré-processing

Sur les 12 000 images qui composent le dataset final, 100 images par classes (13 classes - donc 1300 images au total) ont servies comme ensemble d'entraînement pour le pipeline de deep learning final et 20 images par classe (soit 260 images) ont servies pour l'ensemble de test.

```
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    zoom_range = 0.3,
    rotation_range = 50,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    horizontal_flip = True,
    vertical_flip = True,
    fill_mode = 'nearest')

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    classes=classes,
    class_mode='categorical')
```

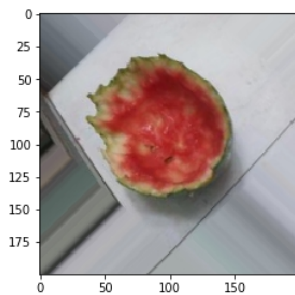
Une Data Augmentation a également été opérée afin d'améliorer la variabilité des entrées pour nos modèles (symétrie horizontale et verticale, rotation, zoom , ...) et une normalisation de l'intensité des pixels entre 0 et 1.

Le ImageDataGenerator va procéder à un LabelEncoding et à un One-Hot-Encoding. Le paramètre "class" ([bois,

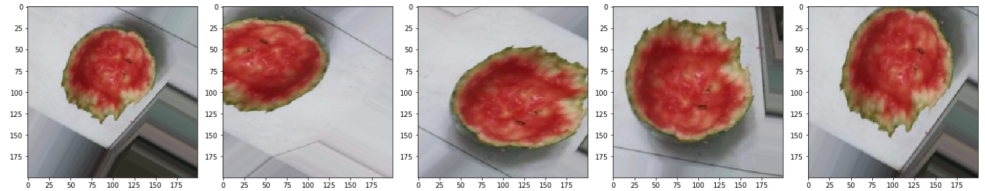


carton, ...]) permet de définir la correspondance entre l'ID ([0,1,...]) produit par le LabelEncoder et le label sous forme de chaîne de caractères ("bois" : 0, "carton" : 1, ...)

Originale



Avec Data Augmentation



Evaluation des modèles de transfer learning

Afin de déterminer le modèle qui sera sélectionné puis optimisé, nous avons entraînés tous les modèles du module "Applications" de Keras, soit 18 modèles pré-entraînés, divisés en 2 catégories en fonction des tailles d'images acceptées en guise d'input: les modèles qui acceptent à minima des images de taille 150x150 et ceux qui acceptent à minima des images de taille 200x200. Les 1300 images initiales ont donc été processées 2 fois : une fois pour être dimensionnées en 150x150 et une fois pour être en taille 200x200.

**Top 5 en terme de temps d'inférence
par epoch (sec)**

	Optimizers	Inference
9	mobilenetv2	66.9
14	inceptionresnetv2	150.1
13	xception	152.3
15	inceptionv3	154.8
10	densenet121	182.1

Top 5 en terme d'accuracy

	Optimizers	Accuracy
10	densenet121	0.6093
11	densenet169	0.6068
12	densenet201	0.5918
15	inceptionv3	0.4527
13	xception	0.4409

Dans un premier temps j'ai comparé plus en détails les meilleurs modèles, notamment Xception et DenseNet121 : 30 epochs au lieu de 10, modèle simple ou plus complexe, différents paramètres d'entraînement, ... Finalement, le modèle sélectionné a été Xception car il est en 3e position (sur 18 modèles) en temps d'inférence par epoch tout en conservant d'honnêtes performances d'accuracy (5e sur 18).

Étapes Transfer Learning / Fine Tuning vers le modèle actuel

- Instanciation du modèle de base avec des poids pré-entraînés ;
- On va ajouter des couches au modèle de base :

```
# Couches à ajouter au Model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
x = Dense(128, activation = 'relu')(x)
x = Dropout(0.5)(x)
x = Dense(64, activation = 'relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(nb_classes, activation='softmax')(x)

# Ajout du bloc de couches au modèle de base
model = Model(base_model.input, predictions)
print(model.summary())
```

Exemple avec l'un
de mes modèles

- Data augmentation appliquées sur les ensembles de train, validation, test ;
- Les couches sont figées au départ ;
- On compile le modèle :

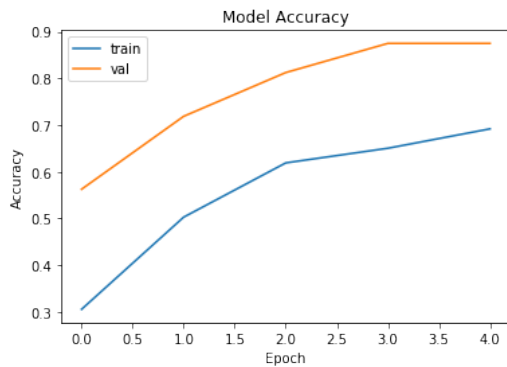
```
model.compile(optimizer=Nadam(learning_rate=2e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

- Utilisation de callbacks : EarlyStopping, ModelCheckpoint, ReduceLROnPlateau et CSVlogger ;
- Entraînement du modèle sur 5 epochs pour le moment (car chaque epoch est très longue mais à terme l'entraînement se fera sur bien plus d'epochs) et sauvegarde des poids de la meilleure epoch.
- Puis dégel des couches possible et entraînement des couches inférieures avec les mêmes poids et paramètres d'entraînement.

Performances

Différents paramètres d'entraînements ont été testés afin d'en sélectionner le meilleur.

- Modèle avec une couche de Fine-Tuning :

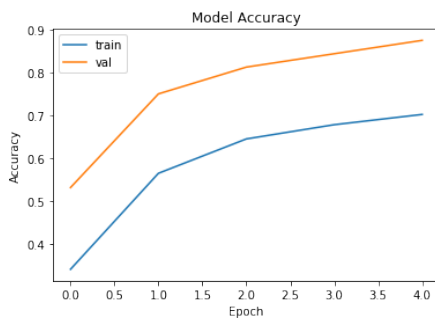


Version minimaliste : modèle de base avec une couche Flatten, une couche de Dropout (0.25) et une couche Dense comprenant en sortie le nombre de classes de mon projet, c'est-à-dire 13.

On y voit un modèle qui n'apprend plus et qui en 3 epochs atteint les 90% d'accuracy.

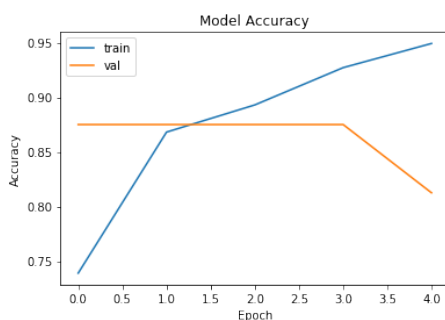
Léger overfitting au delà de 3 epochs.

- **Modèle avec 2 couches de Fine-Tuning** : Une combinaison de paramètres de régularisation plus forts (par exemple, un Dropout plus élevé : 0.3) pour compenser la taille du réseau de neurones.



On peut noter que ce modèle continue d'apprendre au-delà de 4 epochs.

Au bout de 4 epochs, ses performances sont plus élevées que le modèle précédent.



- Sur le modèle précédent, le dernier bloc de couches de convolution a été dégelé, on note un rapide overfitting (performance sur le train très élevée et chute sur l'ensemble de validation).

- Modèle avec pondération des classes

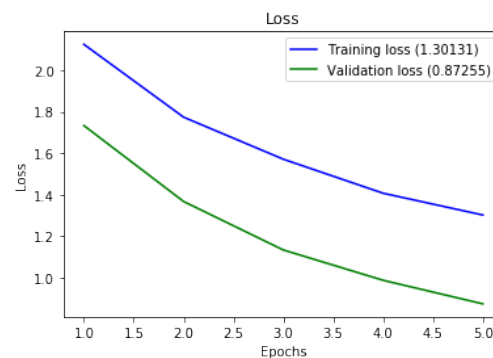
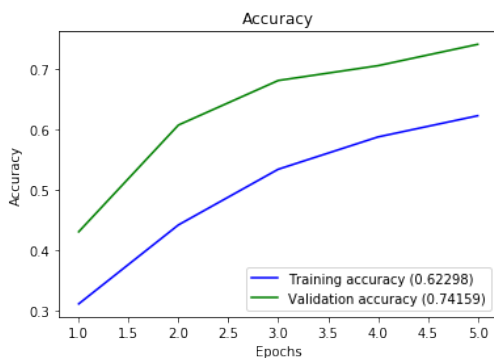
Pour ce modèle, des poids de classe ont été calculés. Ces poids seront utilisés pour pondérer les loss et ainsi obtenir de meilleurs scores sur les classes minoritaires.


```

counter = Counter(train_generator.classes)
max_val = float(max(counter.values()))
class_weights = {class_id : max_val/num_images for class_id, num_images in counter.items()}

```

Ce modèle a une accuracy plus faible. En revanche, il donne de bien meilleures performances sur les classes minoritaires. Il sera intéressant de définir une nouvelle métrique qui permette de prendre en compte les classes minoritaires d'une meilleure façon. Il sera alors nécessaire de réentraîner les modèles précédents et de les comparer au regard de cette nouvelle métrique.



Recherche d'hyper paramètres

J'ai cherché à améliorer le deuxième modèle présenté ci-dessus. Pour ce faire, j'ai mis en place différentes méthodes d'optimisation.

● GridSearch - Recherche de l'optimizer :

```
optimizer=hp.Choice('optimizer', ['SGD', 'RMSprop', 'Adagrad', 'Adadelata', 'Adam', 'Adamax', 'Nadam']),
```

Voici les résultats obtenus

```

Results summary
Results in ./dumpin
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
optimizer: Adadelata
Score: 0.17692308127880096
Trial summary
Hyperparameters:
optimizer: Adamax
Score: 0.1269230842590332
Trial summary
Hyperparameters:
optimizer: RMSprop
Score: 0.08076923340559006
Trial summary
Hyperparameters:
optimizer: Adam
Score: 0.08076923340559006
Trial summary
Hyperparameters:
optimizer: SGD
Score: 0.07692307978868484

```

La meilleure combinaison, notamment l'optimizer, est conservé dans un dictionnaire qui est ensuite transmis

pour l'entraînement du modèle.

```

# Construction du modèle avec les hyperparamètres optimaux et entraînement sur 5 epochs
model = tuner.hypermodel.build(best_hps)
history = model.fit_generator(
    train_generator,
    steps_per_epoch=train_generator.n//train_generator.batch_size,
    epochs=5,
    verbose=1,
    validation_data=validation_generator,
    validation_steps=validation_generator.n//validation_generator.batch_size
    callbacks=callbacks_list)

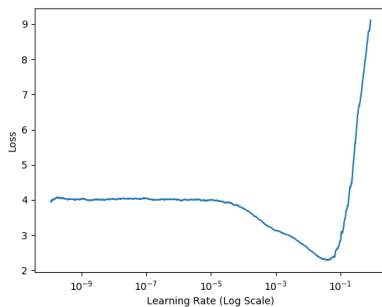
```

- Optimisation Bayésienne - Recherche de la valeur du Dropout et du nombre de neurones de la couche Dense :

```
waste_model.add(layers.Dense(units=hp.Int('units',
                                         min_value=32,
                                         max_value=1024,
                                         step=32),
                             activation='relu'))
waste_model.add(layers.Dropout(
    hp.Float('dropout', 0, 0.9, step=0.1, default=0.5)))
waste_model.add(layers.Dense(units=hp.Int('units',
                                         min_value=32,
                                         max_value=1024,
                                         step=32),
                             activation='relu'))
waste_model.add(layers.Dense(13, activation='softmax'))
```

```
tuner = BayesianOptimization(
    hypermodel=build_model,
    objective='val_loss',
    max_trials=25,
    num_initial_points=2,
    project_name='dumpin2',
    seed=42)
```

- Recherche du Learning Rate

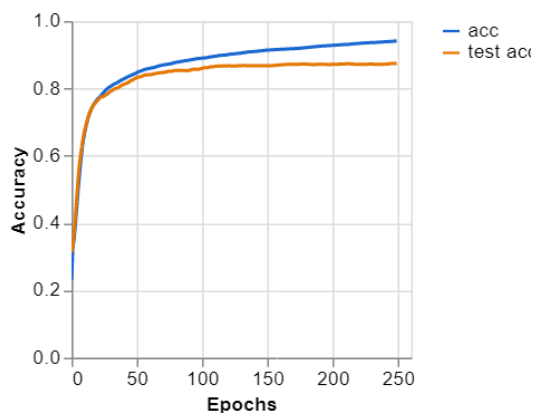


Les résultats peuvent s'expliquer par un nombre d'époques étant trop petit qui empêche de montrer les gains de performance d'un learning rate plus faible.

Ici, le learning rate "optimal" est trop grand, ce qui montre que cela peut amener le modèle à converger trop rapidement vers une solution non optimale. Depuis, le modèle étant actuellement en phase de modification profonde, cette étape devra être générée de nouveau.

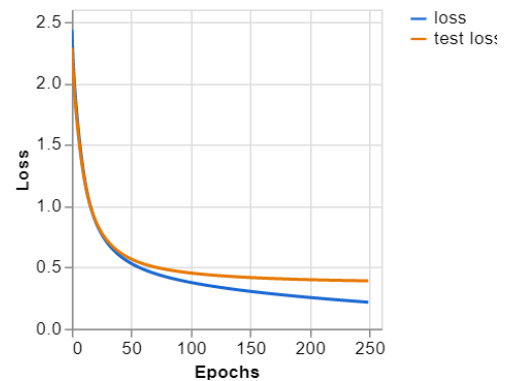
Cloud computing

Les modèles précédents prenant beaucoup de temps à calculer et limitant le nombre d'expériences à réaliser en temps humain sur ma machine personnelle, j'ai testé la plateforme <https://teachablemachine.withgoogle.com/> pour entraîner un modèle de



classification. Entraîné sur 250 epochs, avec un batch size de 128 et un learning rate de 1e-5, celui-ci est compilé avec Tensorflow Lite (extension .tflite) dont voici les métriques :

Accuracy
Loss



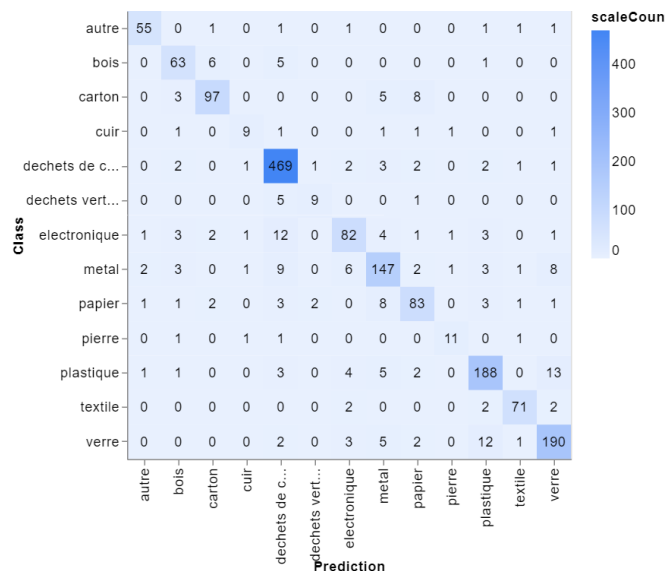
Passage de 32% (epoch 1) à 94% sur le train et 87% sur le test (epoch 250)

Passage de 2.1 (epoch 1) à 0.21 sur le train et 0.39 sur le test (epoch 250)

Accuracy par classe

CLASS	ACCURACY	# SAMPLES
autre	0.90	61
bois	0.84	75
carton	0.86	113
cuir	0.60	15
dechets de cuisine	0.97	484
dechets verts	0.60	15
electronique	0.74	111
metal	0.80	183
papier	0.79	105
pierre	0.73	15
plastique	0.87	217
textile	0.92	77
verre	0.88	215

Matrice de confusion



Même s'il overfit très légèrement , le modèle pourra être amélioré en ajoutant un léger dropout. C'est ce modèle qui est retenu pour mon application web comme mobile.

Réentraînement automatique de modèle d'apprentissage automatique

Cron a été utilisé pour mettre en place la programmation périodique d'un script Python de réentraînement du modèle d'apprentissage profond. Dans un premier temps exécuté de façon régulière et arbitraire, le mercredi de chaque semaine, il pourra être programmé pour être déclenché après l'insertion d'un certain nombre de nouvelles données dans la BDD. Ce processus est packagé dans un container Docker dédié.

APPLICATION

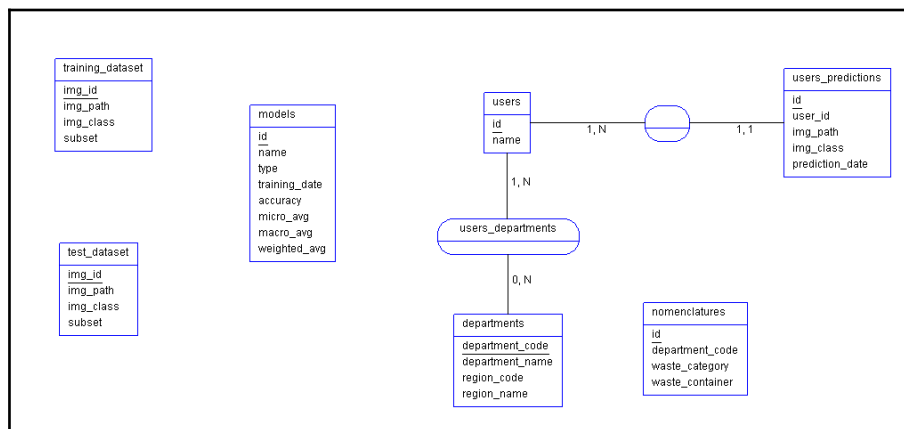
Pour rappel, l'application sert à faire une prédiction à partir des photos uploadées par l'utilisateur, stocker toutes les informations relatives à sa session d'utilisation (pseudo, département, image et prédiction faite) puis plus largement retrouver tout son historique d'upload à des fins statistiques, que ce soit sur l'interface web que l'application mobile.

BASE DE DONNÉES

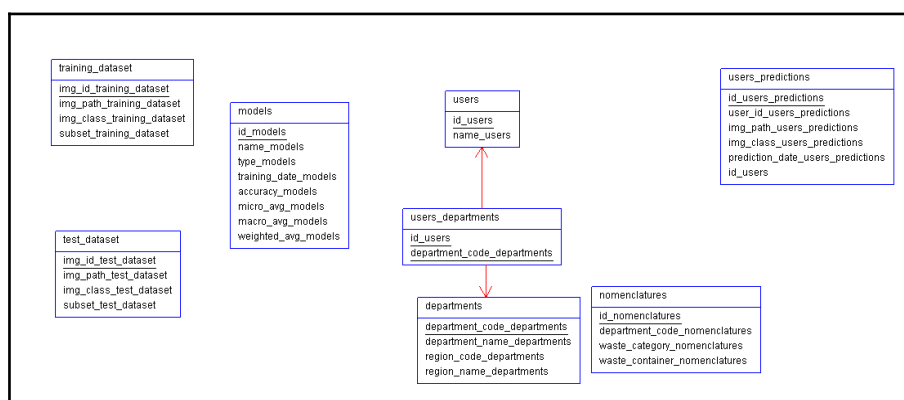
Le SGBD (**S**ystème de **G**estion de **B**ase de **D**onnées) choisi est un SGBDR, c'est-à-dire de type relationnel : MySQL. Pourquoi ? En raison d'un besoin de structurer les données, qui sont liées par des caractéristiques communes et dépendantes.

Conception

L'architecture s'est faite en respectant les principes de la méthode MERISE à travers la création d'un MCD (**M**odèle **C**onceptuel de **D**onnées).



Ce schéma a ensuite été transposé en MPD (**M**odèle **P**hysique de **D**onnées) me permettant d'aboutir au script SQL de création de la BDD. La structure retenue est, au final, très simple et comprend 8 tables :



- stockage du training set et validation sets (dans la même table) ainsi que du test set (table différente) avec les chemins relatifs aux fichiers, leur intitulé et le label correspondant ;
- stockage des différents modèles entraînés et de leurs métriques de performance ;
- stockage des variables de session d'un utilisateur de l'application (le pseudo qu'il a choisi, le département où il se situe actuellement, le nom de l'image qu'il upload et la prédiction qui correspond) ;
- les départements et régions ;
- les nomenclatures concernant la gestion des déchets dans les différentes zones (pour l'instant, seule la région Gironde est incrémentée. Pour les autres, c'est à venir).
- Particularités notables :
 - la table "users_departments" contient une clé primaire composite, c'est-à-dire clé composée de plusieurs champs qui sont en même temps clés étrangères ;
 - la table "nomenclatures" comprend une clé unique sur 2 colonnes ;
 - clé de hachage sur le contenu des fichiers (sha) ;
 - différents types de relations (one to one, one to many ou many to many).
- Un fichier de configuration a été généré et lancé automatiquement lors de la création de la base de données pour l'encodage UTF-8.
- Quatre scripts sont automatiquement exécutés depuis le dossier init lors de la création de la base de données :
 - un script de création de la BDD, des tables, champs et attributs ;
 - un script de modification des tables et attribution des contraintes (clés primaires et clés étrangères) ;
 - un script de peuplement des différentes tables ;
 - un script de création des différentes vues et index.

La structure ainsi obtenue, correspondant au besoin actuel, est bien entendu évolutive et va croître avec l'établissement de nouvelles fonctionnalités.

Exemple du dictionnaire de données

nomenclatures	id	int	PRI	NO	auto_increment
nomenclatures	department_code	varchar(20)	MUL	NO	
nomenclatures	waste_category	varchar(255)		NO	
nomenclatures	waste_container	varchar(255)		NO	
test_dataset	test_img_id	int	PRI	NO	auto_increment
test_dataset	test_img_path	varchar(255)		NO	
test_dataset	test_img_class	varchar(255)		NO	
test_dataset	test_subset	varchar(255)		NO	

Sauvegarde de la base de données

Un mécanisme de sauvegarde automatique, programmé grâce à l'utilisation de cron, pour sauvegarder la base de données de manière hebdomadaire (le dimanche soir à 23h58), a été implanté directement dans le docker-compose. Celui-ci génère un fichier compressé, dans un dossier spécialement dédié “/backups”, qui contient un script de création et peuplement de la base de données et porte le nom suivant : “202103210000.dumpin.sql.gz”.

Population et utilisation

Les tables contiennent des données ayant été peuplées grâce au script “03_data.sql” qui insère, par exemple, 7841 lignes dans la table “training_dataset”, 3361 lignes dans la table “test_dataset”, 13 lignes dans la table “nomenclatures” et 100 lignes dans la table “departments”. Les tables “users”, “users_predictions” ou “models” se peupleront au fur et à mesure de l'utilisation de l'application par les utilisateurs grâce à des requêtes préparées.

```
SELECT u.* FROM users u
JOIN users_departments ud ON ud.user_id = u.id
JOIN departments d ON ud.department_code = d.department_code
WHERE d.department_code = "33"
```

id	name
1	vanessa

Exemple
requête
jointure :

Exemple de
requête

sous-requête :

```
SELECT `img_class`, COUNT(*) as count
FROM `users_predictions`
WHERE `user_id` IN
(SELECT `id` FROM `users` WHERE `name` = 'vanessa')
GROUP BY `img_class`
```

img_class	count
carton	1

de
avec

avec

Alimentation automatique des données utilisateurs



L'application sauvegardant les photos uploadées par les utilisateurs dans un dossier en local ("/uploads") et stockant les chemins d'accès vers ces images et leurs prédictions dans la table "users_predictions", un mécanisme de récupération automatique a été créé. Celui-ci permet, via un script Python, de récupérer ces informations directement en base de données si elles sont en quantité suffisante (au moins 10 de chaque classe). Ces données sont splitées en train, validation et test.

Les images sont copiées vers un dossier local. La classe, le chemin et le type de dataset (train, validation ou test) sont ensuite envoyés vers la base de données.

Ces nouvelles données permettront d'entraîner de nouveaux modèles plus performants grâce à des données qui représentent effectivement les données du terrain. Ce script va se lancer tous les dimanches à 23h59.

Optimisation

Selon la taille du dataset et le besoin, il peut parfois paraître intéressant de stocker les images directement en base de données, format Blob (par exemple petit dataset), mais compte tenu de la taille de mon dataset (12 000 entrées), cela rendait le requêtage assez lourd. Du coup, trois optimisations ont été opérées :

- Passage du stockage des blobs en stockage des chemins relatifs ;
- Création de vues, notamment sur les trainings, validation et test sets ;
- Création d'index, dont notamment un index sur l'ID du training set. ou un index unique sur la colonne "name" de la table "users".

Exemple de temps de requêtes :

Sélection par les chemins relatifs

7,841 lignes (0.013 s)

Sélection par la vue

7,841 lignes (0.010 s)

Sélection par l'index

7,841 lignes (0.008 s)

FRONT

Librairies utilisées et versions

Un environnement Anaconda a été entièrement dédié et exporté pour pouvoir reproduire le projet. Le fichier requirements.txt, lancé automatiquement lors de la création des conteneurs Docker, en comprend la liste et les versions.

A titre d'exemple, citons notamment :

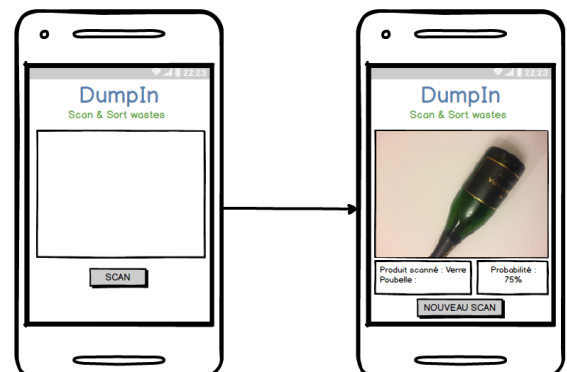
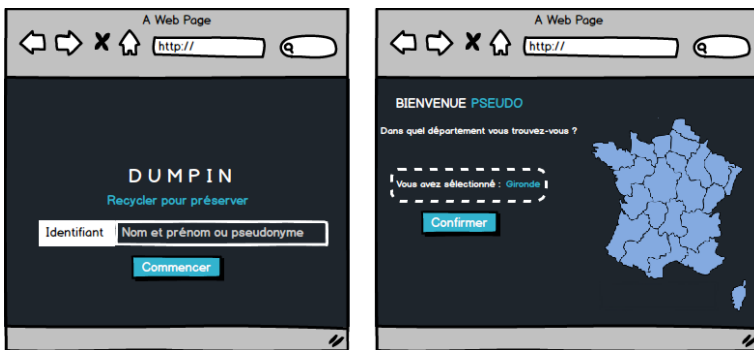
NOM DE LA LIBRAIRIE	VERSION	UTILISATION
Flask	1.1.2	Microframework de développement web
Flask-Session	0.3.2	Stocke des données spécifiques à l'utilisateur entre les demandes
Flask-Testing	0.6.2	Fournit des utilitaires de test unitaire pour Flask
Selenium	3.141.0	Simule le comportement d'un utilisateur (pour les tests)

Parcours utilisateur

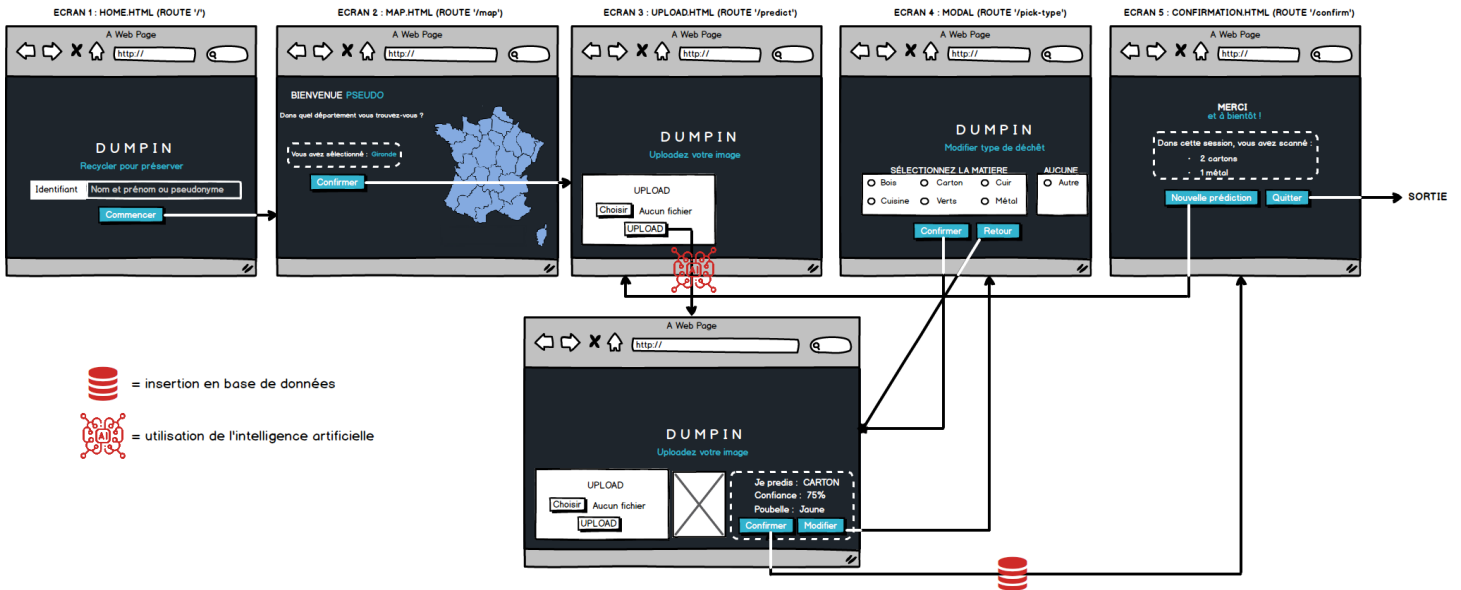
Le parcours utilisateur est un pilier à ne surtout pas négliger de la méthodologie UX lors de la création d'un site web, enchaînement d'étapes qui nous emmène d'un point A vers un point B. On ne peut pas proposer une expérience maîtrisée si le parcours utilisateur n'a pas été correctement modélisé.

MAQUETTAGE AVEC BALSAMIQ MOCKUP

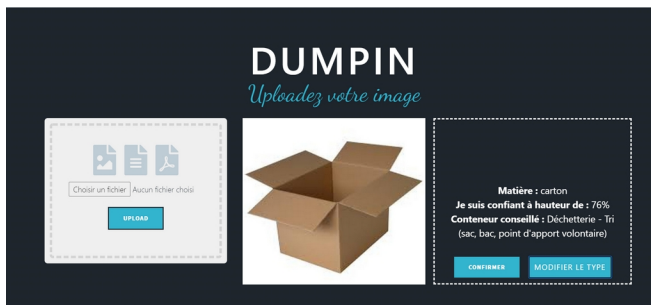
Interface web _____
_____ Application mobile



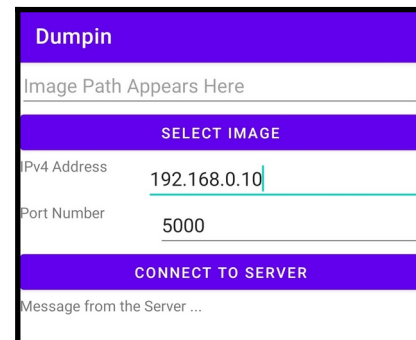
WORKFLOW



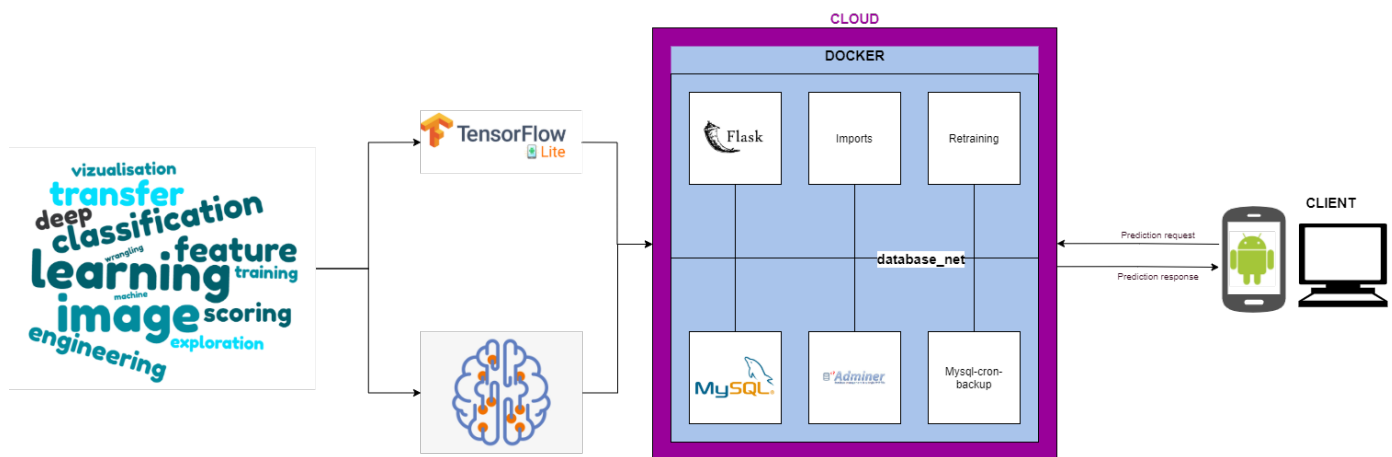
RÉALISATION DE L'INTERFACE WEB



RÉALISATION DE L'APPLICATION MOBILE



BACKEND



Outre toutes les étapes de traitement des données et l'entraînement des 2 IA, un docker avec 6 containers a été créé et déployé dans le Cloud (plateforme HEROKU). Un client va donc faire une requête de prédiction à mon application qui va lui retourner la réponse.

QUALITÉ ET MONITORING

Monitoring

Le monitoring, ou supervision d'une application, est la capacité à avoir une vue globale sur une application à un instant T mais aussi un historique des états passés pour plusieurs éléments : les temps de réponse des différentes ressources du serveur, la vérification que le contenu des pages web ne change pas et la vérification que l'application assure l'intégralité de ses fonctionnalités.

Dans mon application, j'ai utilisé la librairie "Selenium" qui mime le comportement de l'utilisateur et permet d'automatiser l'interaction entre un utilisateur et un navigateur web pour en vérifier la durabilité - Pas encore fonctionnel.

Test

Pour ce projet, j'ai commencé à mettre en place différents types de tests (non encore véritablement implémentés) pour vérifier la qualité et maintenabilité de mon application : Pytest : teste les éléments un à un ou par série grâce à des fixtures qui s'assurent que les tests puissent être exécutés de manière fiable et répétitive.

Pytest-docker : Des montages pytest pour écrire des tests d'intégration avec Docker et docker-compose.

Flask-Testing : Elle permet d'utiliser la puissance de Flask dans les tests.

Exemple de test :

```
def test_homepage_response200(flask_service_url):  
    response = requests.get(flask_service_url)  
  
    assert response.status_code == 200
```

Ce bout de code s'assure que la page d'accueil retourne un code de statut de réponse HTTP 200.

CONCLUSION


Ce projet a été l'occasion de répondre à un besoin réel, à des attentes concrètes d'utilisateurs, à un manque dans ce domaine et à un objectif personnel.

Il a également été l'occasion idéale de parfaire ce qui a été abordé au cours de notre formation, ainsi que d'aller plus loin (fine tuning de modèle pré-entraîné ou hyper tuning par exemple) et de le confronter à une problématique sociétale d'actualité, touchant toute personne dans la vie de tous les jours et qui m'est chère.

Loin d'être arrivé au stade où mon objectif pourrait aboutir, un certain nombre de perspectives d'amélioration ont d'ores et déjà été identifiées dont en voici un court extrait.

Axes d'évolution :

- Stocker toutes les nomenclatures de région (travail de recherche et de scrapping important) ;
- Créer un espace authentification login/mot de passe pour les utilisateurs avec vérifications de saisie et contrôles de sécurité ;
- Compte tenu du **R**èglement **G**énéral sur la **P**rotection des **D**onnées (RGPD), placer des conditions d'utilisation sur le site et respecter le stockage des informations (comme le nom, département, ...) :
 - l'utilisateur détient le copyright de l'image ;
 - il autorise le stockage et l'utilisation pour l'entraînement de modèles ;
 - il autorise des personnes à regarder les photos stockées.
- Prévoir de déployer dans le Cloud (Microsoft Azure à priori) ;
- Mettre en place un réentraînement automatique du modèle en fonction de l'upload d'un certain nombre d'images (en veillant à ce que les classes ne se déséquilibrent pas de façon disproportionnée) - actuellement implanté mais selon une granularité temporelle ;

- 
- Permettre à l'utilisateur d'envoyer plusieurs images en même temps (dans la limite d'un poids maximum toujours).