Rapport technique

Participants : Olga TOLSTOLUTSKA, Mohamed BACHKAT, Charly LAGRESLE
Mentor : Manu POTREL
Promotion : DST Bootcamp DEC22
Date du document : 2023/03/06

Introduction

Contexte

Rakuten est un site de e-commerce, regroupant près de 1.3 milliards d'utilisateurs. Afin de faciliter la gestion des suggestions de leurs recherches ou des recommandations qui leurs seront proposées, il est important de classifier les produits. Du fait de la quantité de produits et de leurs catégories associées, ce classement ne peut ni être fait manuellement, ni être régi par des règles. Une approche plus flexible, efficace et reproductible est donc à mettre en place.

L'objectif est de prédire le code de chaque produit, c'est-à-dire sa catégorie, sur la base d'une analyse de données textuelles et d'images.

Étapes du projet

Le déroulement du projet suit la statégie suivante :

- Analyse exploratoire
  - Prise en main des données
  - Analyse statistique et *feature engineering*

    - Choix des métriques
- Conception de modèles de référence en *machine learning*
- Conception de modèles *deep learning*
    - Traitement des images et du texte
    - Fusion de modèles

Le rapport ici présent se compose de parties similaires à ce plan.

Analyse exploratoire

L'exploration des données est la première étape de ce projet. Cette dernière permet de se familiariser avec le jeu de données, d'en comprendre la structure, les particularités et d'anticiper les actions à mettre en place pour extraire, transformer et traiter ces données.

Prise en main des données

Les données fournies pour ce challenge sont les suivantes :

- `X_train_update.csv` : données explicatives et textuelles pour l'entraînement
- `X_test_update.csv` : données explicatives et textuelles pour les tests
- `Y_train_CVw08PX.csv` : variable cible pour l'entraînement
- `images/image_train/*.jpg` : images pour l'entraînement
- `images/image_test/*.jpg` : images pour les tests

> Nous notons que la variable cible de test n'est pas fournie : les observations de tests ne seront donc pas exploitées.

Le jeu d'entrainement se compose a priori de 84 916 observations : des données textuelles ainsi que des images.

Données d'entrées

Deux types de données d'entrées sont à notre disposition : des textes et des images. Nous analysons chacun de ces types de données séparemment.

Textes

Les données `X_train_update.csv` se composent de plusieurs champs :

- `designation` : titre du produit dans différentes langues (0% de NaNs)
- `description` : description détaillée du produit dans différentes langues (35% de NaNs)
- `productid` : identifiant unique du produit (0% NaNs)
- `imageid` identifiant unique de l'image du produit (0% NaNs)

> Les champs `productid` et `imageid` ne servent qu'à construire le nom du fichier image associé au produit. Ils ne seront donc pas étudiés statistiquement parlant.

Près de 35% des observations de la colonne `description` est manquant : cette colonne sera par la suite fusionnée avec la colonne `designation`, ce qui rendra nul le nombre de NaNs de cette nouvelle colonne.

Images

Chaque produit est associé à une image qui le représente. L'image comporte le produit, généralement photographié sur un fond blanc, mais peut également être complété par une mise scène du produit (jardin, décoration) ou par un support (main portant le produit par exemple). Les images sont en couleur, ont une taille de 500x500 pixels et nous sont fournies au format `.jpg`.

Données de sorties

Les cibles sont fournies dans le fichier `Y_train_CVw08PX.csv`. À ce stade, nous nous contentons de noter que 27 catégories différentes seront à déterminer.

Analyse du problème

Type de problème

Les observations textuelles et graphiques doivent permettre au modèle d'affecter à chaque observation dont les features sont actuellement (`designation`, `description`, et une image) à une catégorie de produit `prdtypecode`. Cette cible est disponible et se présente sous la forme d'un numéro associé à une catégorie (par exemple `livre`, `tech`, `décoration`, etc.).

Nous avons donc affaire à un problème de catégorisation supervisé. Pour la partie texte, un analyse automatique du langage naturel (*NLP : Natural Langage Processing*) sera à mettre en place. Pour la partie image, l'analyse des pixels et de leur organisation ou *pattern* sera à réaliser via des modèles de type CNN par exemple.

Particularité du problème

La catégorisation se base sur l'apprentissage fait sur les données à notre disposition. Ces données sont de deux types (textuel et graphique). De ce fait :

- les pré-traitements seront différents
- plusieurs modèles seront à concevoir, à entraîner et à faire dialoguer
  - un modèle pour catégoriser le produit en fonction du texte fourni
  - un modèle pour catégoriser le produit en fonction de son image associée
  - une étape de regroupement des prédictions sera à réaliser afin d'obtenir une unique catégorie en fin de process

Analayse statistique et feature-engineering

Maintenant que la structure des données est connue, l'étape suivante est d'analyser plus en profondeur les données, d'en créer de nouvelle et de préparer le dataset à son entraînement. Ces étapes de feature-engineering sont détaillées dans les prochains paragraphes.

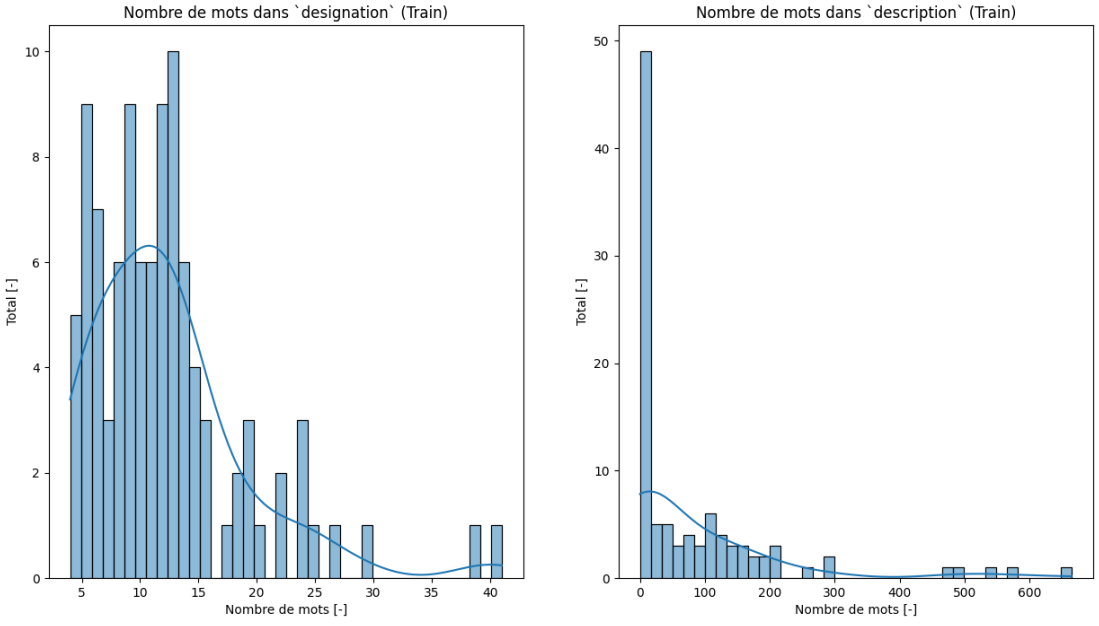
Données d'entrée : textes

L'analyse exploratoire et statistiques de données textuelles nécessite que des données complémentaires soient calculées pour caractériser numériquement nos textes.

Nombre de mots

Pour `designation`, les textes sont composés de 4 à 54 mots. La répartition est de type gaussienne : la moyenne se situe à 11.5 mots et l'écart-type est de 6.4 mots. 75% des textes de `designation` ont moins de 14 mots.

Pour `description`, les textes sont plus longs et comportent entre 0 (certaines descriptions sont vides) et 2 068 mots. En moyenne, la description se compose d'un texte de 78 mots et 75% des produits sont décrits avec moins de 124 mots. La répartition du nombre de mots suit ici la densité de probabilité d'une loi exponentielle. Nous n'oublions pas que 35% de ces valeurs sont des NaNs.



Plusieurs actions seront mises en place suite à cette étude

- Les colonnes `designation` et `description` seront fusionnées en une seule colonne `text`
- La colonne `text` sera limitée à 500 mots.

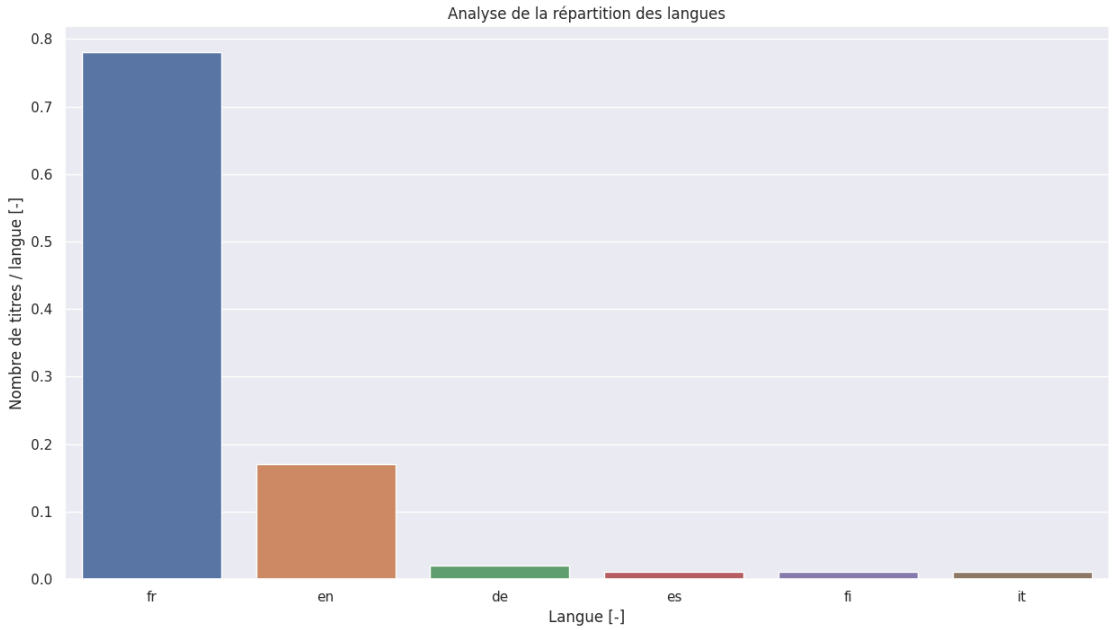
Langues

Intéressons-nous maintenant à la langue. Les colonnes `designation` et `description` contiennent du texte dont la langue peut varier. Le graphique ci-dessous montre la répartition des langues suite à la

fusion des ces deux colonnes en une colonne `text`. La répartition des langues est très hétérogène :

- 81% en français
- 14% en anglais
- 1.5% en allemand
- inférieurs à 1.5% : nl, ca, it, ro, pt, etc...

Suite à la détection de la langue, nous notons que 19% du dataset est dans une autre langue que le français. Le choix a été fait de traduire les textes étrangers vers le français : le travail de preprocessing sera de ce fait simplifié car la gestion de la langue ne sera plus à faire, ce qui aurait probablement posé problème pour l'*embedding*.

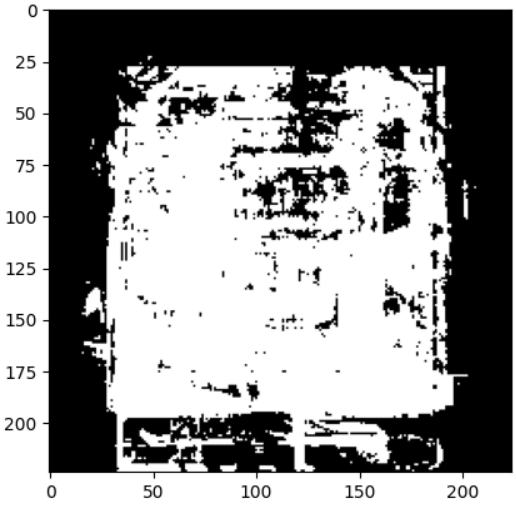


Données d'entrée : images

Réduction de dimension

Analyse de la variance des images

Nous avons noté que beaucoup d'images sont photographiées sur un fond blanc. Ce fond n'apporte en soi aucune information et peut mener à un biais dans notre modèle de classification des images.



En faisant une analyse de la variance des pixels sur un échantillon d'images, nous notons que les bordures des images présentent une variance très basses.

Les images pourront être rognées de 20% pour supprimer ces marges porteuses de peu d'information avant d'être redimensionnées à la dimension souhaitée.

Analyse des couleurs des images

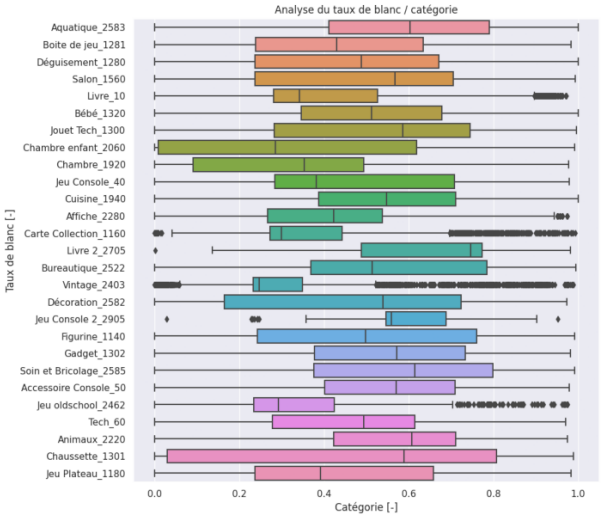
En regroupant les taux de blanc dans les images par catégories, il est possible de distinguer quelques tendances sur le graphique de gauche:

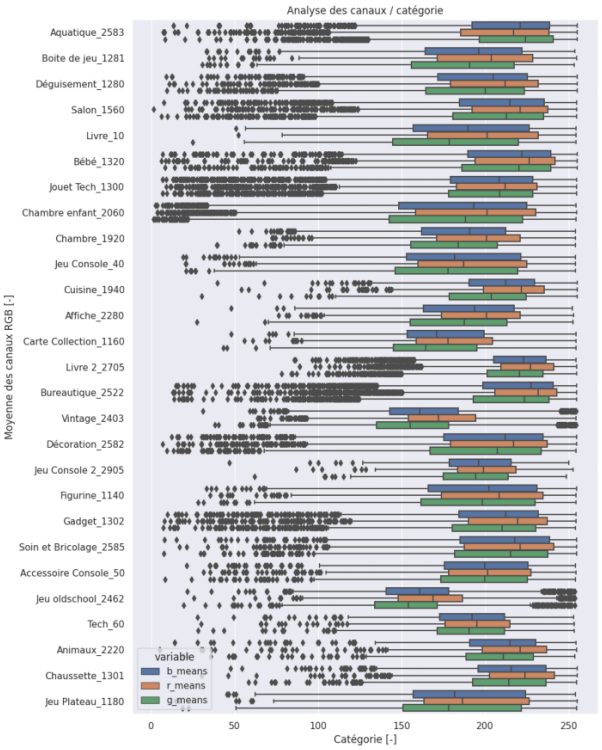
- `Chaussette` et `Chambre enfant` : étendue très large
- `Vintage`, `Jeu oldschool`, `Carte Collection`, `Jeu Console 2` : étendue faible

Le taux de blanc est différent selon les catégories et pourrait permettre d'aider le modèle à catégoriser les produits. Il faudra vérifier que ce taux n'introduise pas un biais dans notre modèle, c'est-à-dire que le modèle ne reconnaisse pas une image pour ses formes mais plutôt pour son taux de blanc.

Sur le graphique de droite, pour les trois canaux RGB de chaque image, sa moyenne a été extraite.

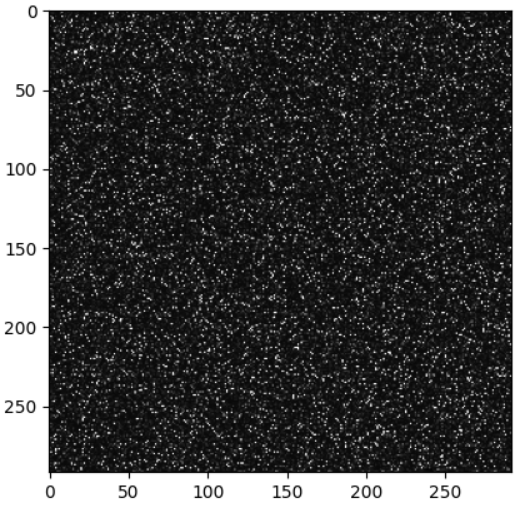
- Les répartition des couleurs entre canaux sont globalement proches : aucune catégorie ne présente donc des images totalement rougeâtres ou bleutées.
- Le moyennes RGB des `Jeu Oldschool`, `Vintage` sont plus basses : les photos sont probablement plus sombres.





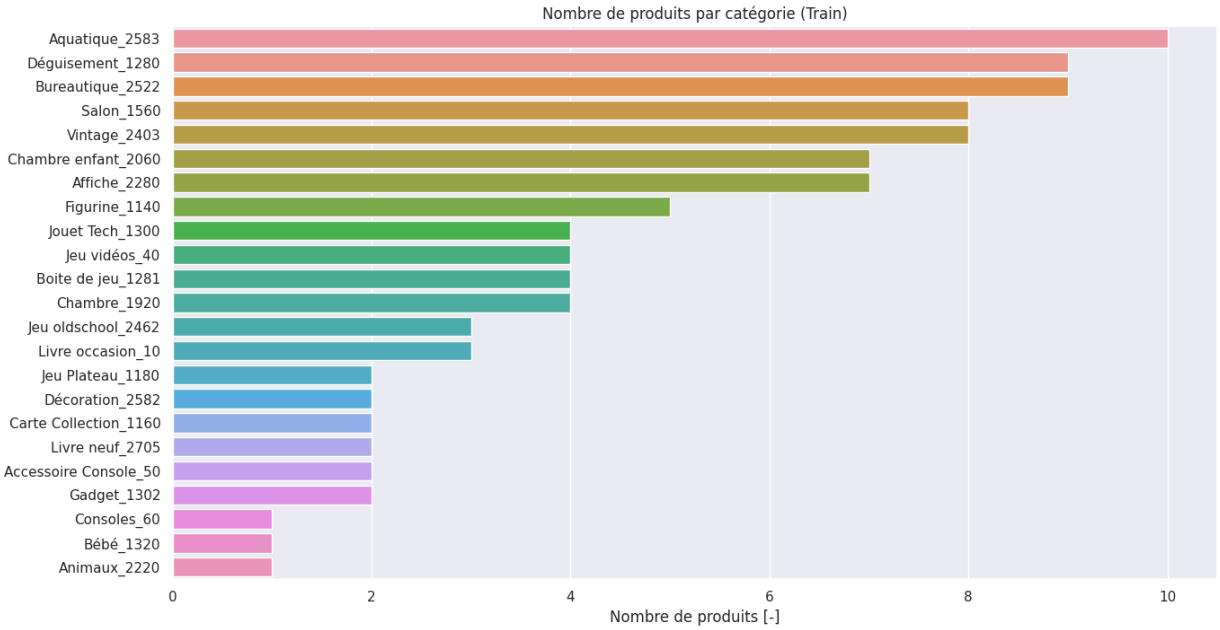
Données de sorties

Comme expliqué précédemment, 27 catégories sont présentes dans notre dataset. La figure ci-dessous présente sur 84 916 pixels les catégories en nuance de gris : aucun patern n'émerge visuellement, le dataset d'origine semble être déjà randomisé.Si cela n'avait pas été le cas, nous aurions observé des rectangles distincts.



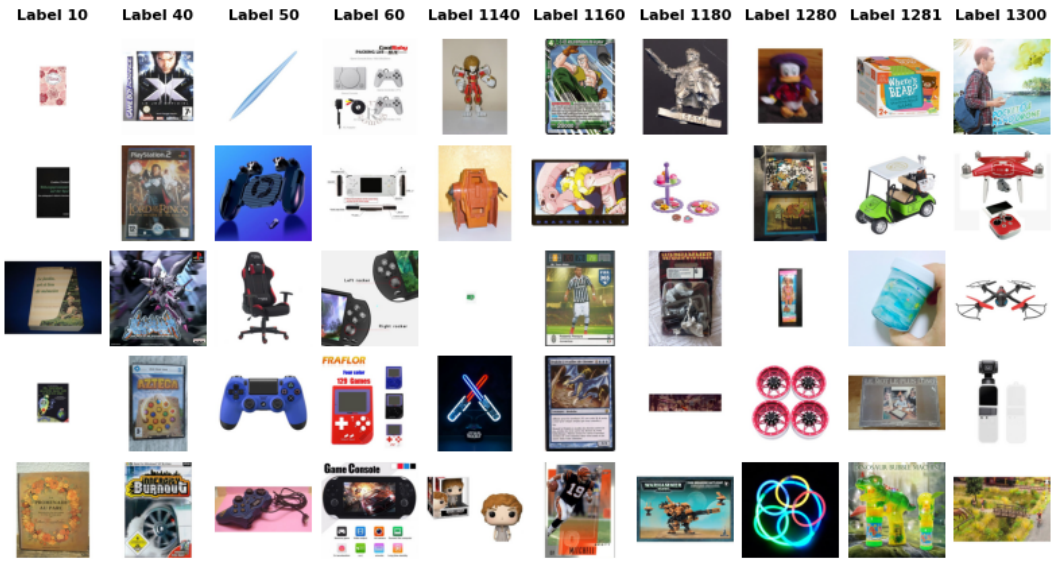
Déséquilibre

Le graphique ci-dessous montre le nombre de produits pour chacune des 27 catégories. On remarque facilement que la catégorie `Aquatique` est sur-représentée : elle comporte le double de produits par rapport à la catégorie suivante. On observe également une baisse linéaire du nombre de produits de `Affiche` vers `Accessoire console`. Les catégories de `Cuisine` à `Jeu PC` ne présentent que peu d'individus.



Catégories

L'analyse visuelle de quelques produits pour chaque catégorie nous permet d'une part de traduire le numéro de catégorie en une variable plus compréhensible et d'autre part d'étudier la constitution de chacune d'entre elles. Cette analyse a été faite sur toutes les catégories, voici ci-dessous un exemple pour les 10 premières d'entre elles.



Les catégories et leurs descriptions sont proposées dans le tableau suivant.

Numéro de catégorie	Description	Numéro de catégorie	Description	Numéro de catégorie	Description
10	Livre d'occasion	1301	Chaussette	2462	Jeu oldschool
40	Jeu Console	1302	Gadget	2522	Bureautique
50	Accessoire Console	1320	Bébé	2582	Décoration
60	Tech	1560	Salon	2583	Aquatique
1140	Figurine	1920	Chambre	2585	Soin et Bricolage
1160	Carte Collection	1940	Cuisine	2705	Livre neuf
1180	Jeu Plateau	2060	Chambre enfant	2905	Jeu PC
1280	Déguisement	2220	Animaux		
1281	Boite de jeu	2280	Affiche		
1300	Jouet Tech	2403	Revue		

Certains catégories ont été délicates à définir, car présentant de très fortes similitudes avec d'autres. Des catégories plus larges pourraient être identifiées. C'est par exemple le cas pour :

- Gaming : jeu PC et jeu Console
- Lecture : livre neuf et livre d'occasion
- Jeu : figurine, jeu oldschool et jeu manuel

Les algorithmes de classification éprouveront probablement les mêmes difficultés que notre oeil humain pour distinguer les produits à l'intérieur de ces catégories plus étendues.

Choix des métriques

Afin de pouvoir qualifier la performance de nos modèles, le choix de la métrique est important. Les notions importantes avant de choisir notre métrique sont les suivantes :

- connaissance métier : ici, nous catégorisons des produits sur un site de vente et faire une erreur de catégorie n'est pas fatidique, l'erreur peut être corrigée.
- jeu de données déséquilibré : nous n'avons pas d'informations sur l'origine de ce déséquilibre. Une catégorie sur-représentée présente-elle les produits les plus vendus ou les produits les plus souvent mal classés?
- forte tendance à l'*overfitting*

Notre choix se porte sur le *weighted f1-score*, qui considère le déséquilibrage du dataset et fournit un bon compromis entre le *recall* et l'*accuracy*. Si nous avions davantage de connaissances des besoins de Rakuten, une nouvelle métrique aurait pu être développée.

Enfin, deux informations complémentaires relatives à la comparaison des modèles:

- 27 catégories sont à notre disposition. Cela signifie qu'un score de plus de 3.7% est d'ores et déjà supérieur à un modèle purement aléatoire.
- afin de garantir une reproductibilité des résultats, les échantillons d'entraînement et de tests sont pour tous les modèles similaires, avec un *random_state* fixée à 123.

Prévention de l'overfitting

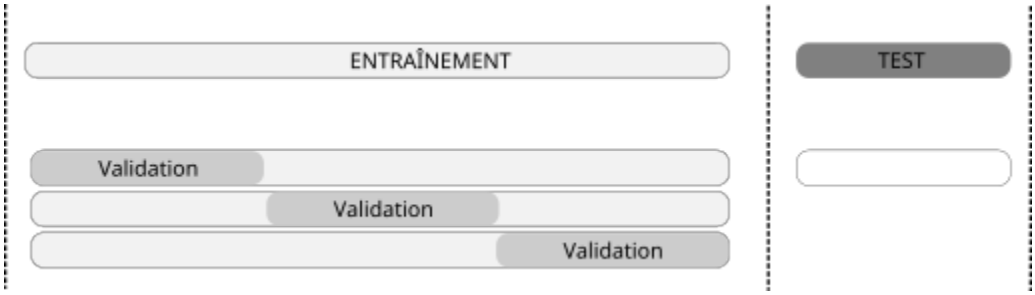
Les modèles implémentés sont sensibles à l'*overfitting*. Afin de minimiser au maximum ce risque, plusieurs actions ont été mises en place :

- suivi de l'évolution des métriques d'apprentissage via *Tensorboard*
- ajout des couches `Dropout` dans nos architectures
- augmentation nos données images
- simplification l'architecture des modèles
- utilisation de batchs de petite taille
- application d'une cross-validation à 3 échantillons :
  - le modèle est entraîné sur les données *train*
  - à chaque époque, son évalution est faite sur les données *validation*
  - une fois l'entraînement terminé, le modèle est ensuite globalement étudié sur les données *test*
- appel à des callbacks

Les callbacks sont utiles pour contrôler le déroulement de l'apprentissage. Nous avons mis en place deux méthodes de gestion de l'apprentissage :

- `EarlyStopping` : met fin à l'apprentissage si `val_loss` augmente pendant plus de 5 périodes à partir de la 8ème période
- `ReduceLROnPlateau` : réduit le taux d'apprentissage si `val_loss` stagne sur un plateau pendant plus de 5 périodes

Finalement, le découpage du dataset pour l'entraînement et la prédiction est le suivant :



Modèles de références

L'analyse exploratoire nous a permis de définir une stratégie pour le traitement du texte et des images. Des modèles de références simples sont mis en place pour catégoriser les produits en fonction des données textuelles et des images à notre disposition. Ces modèles simples se basent sur des techniques de *machine learning* et sur l'ensemble des données fournies.

Textes

Les modèles de texte nécessitent un travail de *preprocessing* important. Une fois la méthodologie mise en place, des modèles de classification simples sont entraînés pour définir la catégorie du produit en fonction du texte dont nous disposons.

Étapes de *preprocessing*

Les étapes suivantes sont exécutées via des pipelines afin de transformer les données textuelles :

- calculer les longueurs des textes pour `designation` et `description`
- calculer le nombre de mots pour `designation` et `description`
- fusionner ces deux colonnes dans `text`
- détecter la langue dans `text`
- traduire les textes vers le français
- préparer la colonne `text` :
  - supprimer la ponctuation
  - supprimer les caractères spéciaux
  - supprimer les valeurs numériques
  - supprimer les majuscules
  - supprimer les balises html
  - supprimer les *stopwords*
  - extraire la racine des mots
  - vectorisation

Extraction de la racine des mots permet de fusionner des mots partageant la même racine et de regrouper leurs poids. Par exemple, les mots `marée`, `marin`, `maritime` sont basés sur le même

radical `mar` (ou `mer`).

Suppression des stopwords qui sont des mots couramment utilisés dans la langue choisie. En français, les mots `afin`, `elle`, `est`, `sur` en font par exemple partie : ils n'apportent peu ou pas d'informations. Cette étape a été maintenue en place bien que le `vectorizer` `tf-idf` aurait pénalisé automatiquement ce genre de mots (car fréquents dans tous les documents).

Vectorisation du texte via un `Tokenizer`. Ici l'idée est de calculer l'importance du mot dans un texte. Nous n'utilisons pas le *tokenizer* en mode `counter` (le mot le plus important est le mot le plus fréquemment présent) mais en mode `tf-idf` pour *Term Frequency - Inverse Document Frequency*. La partie `TF` calcule la fréquence d'apparition d'un mot dans un texte alors que la partie `IDF` corrige ce premier terme en analysant la distribution d'un mot dans l'ensemble des documents. Ainsi, un mot apparaissant fréquemment dans quelques documents seulement aura d'avantage de poids qu'un mot très fréquent dans tous les documents.

À la fin de cette étape de *preprocessing*, nous disposons d'un texte nettoyé. L'étape de vectorisation varie en fonction du modèle utilisé par la suite.

Modèles simples et résultats

Suite à l'application du preprocessing, à la séparation du dataset de texte, plusieurs algorithmes de machine learning ont été entraînés via cross-validation et testés sur le dataset de test :

- LogisticRegressionClassifier
- RandomForestClassifier
- DecisionTreeClassifier
- KNeighborsClassifier

La comparaison des résultats sur la métrique *weighted f1-score* est proposée ci-dessous:

	logistic_regression	random_forest	kneighbours	decision_tree
10	0.449	0.472	0.260	0.376
40	0.543	0.595	0.320	0.499
50	0.730	0.770	0.518	0.618
60	0.875	0.891	0.814	0.823
1140	0.670	0.685	0.530	0.586
1160	0.877	0.865	0.731	0.836
1180	0.391	0.475	0.381	0.420
1280	0.638	0.603	0.449	0.534
1281	0.507	0.482	0.322	0.410
1300	0.899	0.866	0.729	0.890
1301	0.880	0.849	0.847	0.793
1302	0.754	0.758	0.603	0.671
1320	0.699	0.674	0.579	0.576
1560	0.793	0.746	0.636	0.660
1920	0.893	0.905	0.856	0.828
1940	0.821	0.791	0.648	0.669
2060	0.731	0.739	0.620	0.664
2220	0.744	0.751	0.488	0.632
2280	0.790	0.812	0.577	0.764
2403	0.720	0.734	0.604	0.671
2462	0.722	0.768	0.604	0.709
2522	0.891	0.846	0.763	0.755
2582	0.697	0.671	0.525	0.571
2583	0.964	0.926	0.900	0.919
2585	0.728	0.676	0.522	0.540
2705	0.643	0.644	0.321	0.546
2905	0.950	0.977	0.045	0.963
weighted F1-Score	0.771	0.760	0.619	0.695

Pour les algorithmes de machine learning simples, sans recherche d'hyperparamètres idéaux, c'est le modèle *LogisticRegressionClassifier* qui obtient le meilleur *weighted f1-score* avec une valeur de 77.1%, juste avant le *RandomForestClassifier* avec son score de 76.0%. Nous notons que de manière générale, certaines classes sont mal catégorisées. En utilisant un crosstab, nous pouvons trouver les produits pour lesquels il y a confusion. Nous analysons ci-dessous les résultats de *LogisticRegressionClassifier* :

- Catégorie `10` (Livre d'occasion) souvent confondue avec `2705` (Livre neuf) et `2403` (Revue)
- Catégorie `40` (Jeu console) souvent confondue avec `10` (Livre occasion) et `2462` (Jeu oldschool)
- Catégorie `1280` (Déguisement) souvent confondue avec `1281` (Boîte de jeu) et `1140` (Figurine)

La confusion apparaît souvent entre des produits du même type. Ce risque avait été évoqué lors de l'analyse exploratoire de nos données cibles : les produits se décrivent dans un vocabulaire similaire.

Realité	10	40	50	60	1140	1160	1180	1280	1281	1300	1301	1302	1320	1560	1920	1940	2060	2220	2280	2403	2462	252
10	0.52	0.05	0.00	0.00	0.01	0.01	0.04	0.01	0.02	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.07	0.11	0.00	0.0
40	0.10	0.55	0.05	0.00	0.02	0.05	0.01	0.01	0.04	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.02	0.01	0.06	0.0
50	0.00	0.05	0.73	0.02	0.01	0.00	0.00	0.02	0.01	0.01	0.00	0.00	0.01	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.07	0.0
60	0.00	0.02	0.03	0.86	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.0
1140	0.03	0.05	0.00	0.00	0.68	0.03	0.03	0.04	0.02	0.01	0.00	0.01	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.03	0.01	0.0
1160	0.03	0.01	0.00	0.00	0.01	0.89	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.01	0.00	0.0
1180	0.10	0.04	0.02	0.00	0.07	0.02	0.50	0.03	0.06	0.00	0.01	0.02	0.01	0.00	0.00	0.01	0.01	0.00	0.01	0.03	0.01	0.0
1280	0.02	0.02	0.01	0.00	0.05	0.01	0.01	0.61	0.09	0.04	0.00	0.03	0.03	0.01	0.00	0.00	0.02	0.01	0.01	0.00	0.00	0.0
1281	0.03	0.03	0.01	0.01	0.03	0.02	0.02	0.17	0.51	0.00	0.01	0.04	0.01	0.00	0.00	0.00	0.02	0.00	0.01	0.01	0.01	0.0
1300	0.01	0.01	0.00	0.00	0.01	0.01	0.01	0.03	0.00	0.88	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.0
1301	0.00	0.02	0.00	0.01	0.01	0.00	0.01	0.01	0.03	0.00	0.89	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.02	0.0
1302	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.05	0.03	0.01	0.00	0.76	0.03	0.01	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.0
1320	0.03	0.01	0.01	0.00	0.02	0.00	0.01	0.05	0.01	0.00	0.00	0.02	0.69	0.03	0.02	0.01	0.02	0.00	0.01	0.01	0.01	0.0
1560	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.02	0.79	0.03	0.00	0.05	0.00	0.00	0.00	0.00	0.0
1920	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.02	0.90	0.00	0.02	0.00	0.00	0.00	0.00	0.0
1940	0.01	0.01	0.00	0.00	0.02	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.01	0.02	0.00	0.83	0.03	0.00	0.00	0.01	0.00	0.0
2060	0.01	0.01	0.01	0.00	0.01	0.00	0.00	0.02	0.01	0.00	0.00	0.01	0.02	0.06	0.04	0.00	0.71	0.00	0.00	0.00	0.00	0.0
2220	0.00	0.02	0.00	0.00	0.01	0.00	0.01	0.03	0.00	0.01	0.00	0.02	0.05	0.02	0.00	0.01	0.03	0.73	0.00	0.01	0.00	0.0
2280	0.07	0.01	0.00	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.80	0.06	0.00	0.0
2403	0.11	0.02	0.00	0.00	0.00	0.01	0.02	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.69	0.01	0.0
2462	0.03	0.06	0.05	0.00	0.02	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.77	0.0
2522	0.01	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.01	0.01	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.8
2582	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.01	0.07	0.01	0.00	0.09	0.01	0.00	0.00	0.00	0.0
2583	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
2585	0.01	0.00	0.01	0.00	0.00	0.00	0.01	0.02	0.00	0.01	0.00	0.02	0.01	0.03	0.00	0.00	0.04	0.01	0.00	0.00	0.01	0.0
2705	0.18	0.03	0.00	0.00	0.00	0.00	0.02	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.04	0.00	0.0
2905	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0

Images

Modèles Machine Learning appliqués

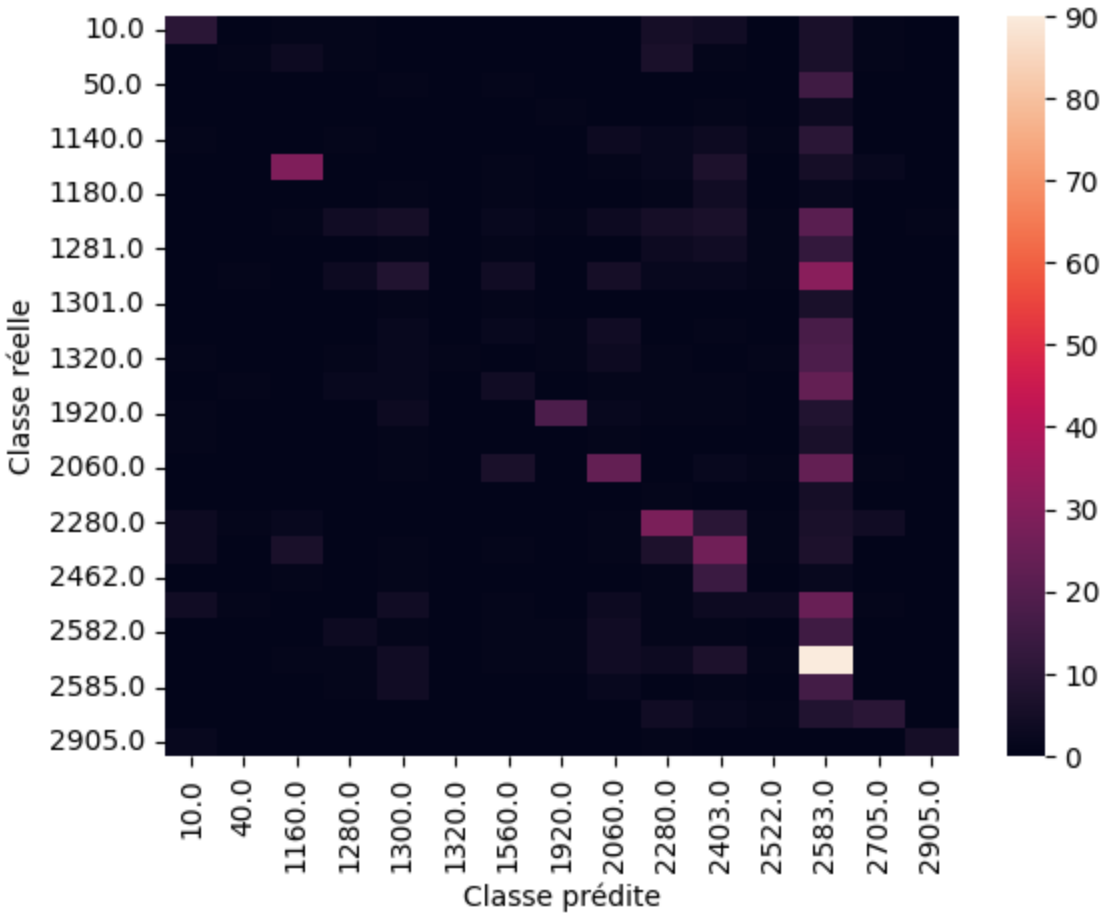
Sur la base de 50 176 variables (224x224), nous avons entraîné et testé différents modèles de *Machine Learning* en utilisant la validation *StratifiedKFold* avec 3 *folds*. Pour ces modèles de machine learning, l'entraînement a été fait sur 1000 images. Nous avons également appliqué la réduction des dimensions PCA pour optimiser l'entraînement.

Le tableau ci-dessus contient la comparaison de modèles testés :

Classifier	Configuration	Accuracy	Precision weighted	Recall weighted	F1 weighted
LogReg	max_iter=10000, tol=0.1	0.18	0.16	0.18	0.16
RF		0.12	0.04	0.12	0.04
KNN		0.18	0.16	0.18	0.16
SVC	C=1, kernel='linear'	0.18	0.17	0.18	0.17
GradBoost		0.09	0.08	0.09	0.06

Figure : Tableau de comparaison de classifiers

Le modèle SVC est plus perfomant selon la métrique *weighted f1-score*. Et sa matrice de confusion heatmap est présentée ci-dessous :



En testant plusieurs types de modèles de classification de *machine learning*, nous avons constaté qu'ils n'arrivent pas à gérer la complexité de notre jeu de données d'où le niveau assez bas de leur perfomance.

Ce constat nous pousse vers la recherche de modèles plus perfomants et plus complexes et donc vers le *deep learning*.

Modèles de deep-learning

Dans la précédente partie, des techniques simples de modélisation ont été mises en place. Afin d'améliorer notre métrique *weighted f1-score*, des modèles de *deep-learning*, plus complexes et longs à entraîner sont utilisés. Finalement, ces modèles seront fusionnés pour exploiter au mieux les performances de chacun d'entre eux.

Aussi, les paramètres de compilation pour les modèles de textes et ceux d'images sont identiques :

- **Fonction de perte** : `sparce_categorical_crossentropy` . Cette fonction de perte est utilisée lorsqu'il s'agit de plusieurs classes d'étiquettes.
- **Optimiseur** : `Adam` qui est reconnu pour avoir de bonnes performances, une vitesse de convergence rapide et est adapté aux problématiques de classification.
- **Batch** : 64 (les valeurs conseillées 16, 32, 64 et etc.)

Textes

Les textes subissent la même étape de *preprocessing* que pour les modèles de *machine learning*.

Modèles testés

Modèle neuronal dense

Le modèle Neural_Simple se base sur la tokenisation *tf-idf* de nos textes. Les entrées sur modèle sont donc une matrice représentant le score *tf-idf* de chaque mot de chaque texte, si ce mot est dans le vocabulaire de notre tokenizer.

Layer (type)	Output Shape	Param
text_dense_1 (Dense)	(None, 54)	270054
text_drop_1 (Dropout)	(None, 54)	0
text_output (Dense)	(None, 27)	1485

Seulement 271,539 paramètres sont à entraîner pour ce modèle.

Modèle neuronal embedding

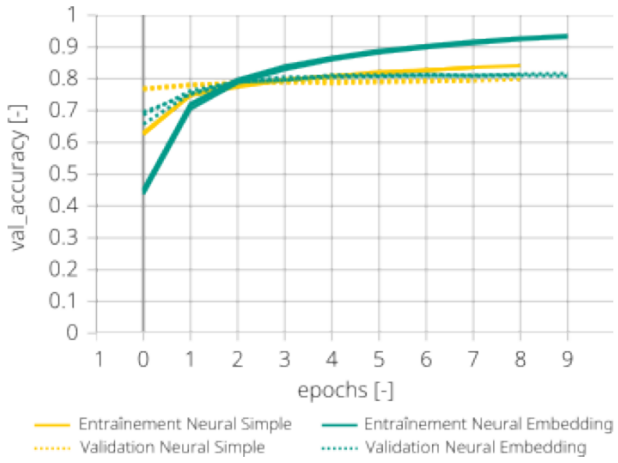
Le modèle *Neural_Embedder*, a la particularité de présenter une couche d'embedding qui nous permet de fournir directement un ensemble de mots à ce réseau de neurones. La vectorisation pour ce réseau gère 50 000 mots et seuls les 500 plus fréquents sont conservés.

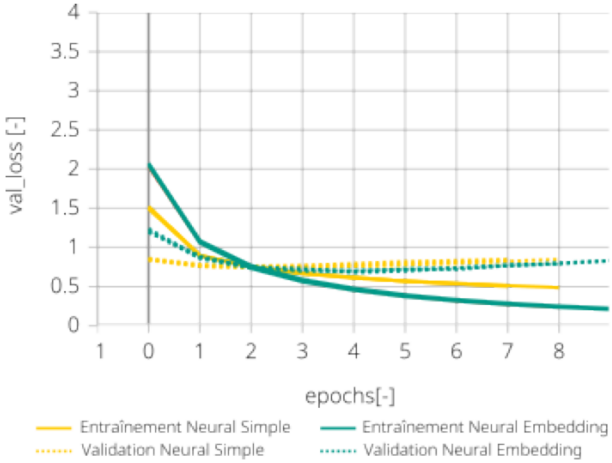
Layer (type)	Output Shape	Param
text_input (Embedding)	(None, None, 100)	5000000
text_average (GlobalAverage Pooling1D)	(None, 100)	0
text_drop_1 (Dropout)	(None, 100)	0
text_dense_1 (Dense)	(None, 54)	5454
text_drop_2 (Dropout)	(None, 54)	0
text_output (Dense)	(None, 27)	1485

Pour ce modèle, cette couche est conçue via un le tokenizer de keras. Le nombre de paramètres à entraîner est ici bien plus important, avec près de 5 millions de paramètres.

Résultats des modèles textes

Avant de présenter les scores de ces deux modèles, intéressons-nous à leurs entraînements afin de vérifier que nous n'avons pas un sur-apprentissage trop important lors de cette phase.





Les courbes sont celles de l'apprentissage (traits pleins) et de la validation (traits pointillés) pour le modèle *neural_simple* (en jaune) et le modèle *neural_embedding* (en vert). Il y a un un léger sur-apprentissage, mais les *callbacks* ont permis de réduire le taux d'apprentissage dans le plateau puis de stopper l'entraînement après une certaine période de patience.

Les *weigthed f1-scores* des modèles sont ceux des données de tests, totalement indépendantes des données d'entraînement et de validation.

	neural_simple	neural_embedding
10	0.486	0.535
40	0.575	0.661
50	0.795	0.822
60	0.872	0.878
1140	0.707	0.731
1160	0.892	0.938
1180	0.445	0.577
1280	0.648	0.708
1281	0.549	0.582
1300	0.906	0.951
1301	0.888	0.957
1302	0.794	0.816
1320	0.729	0.783
1560	0.803	0.819
1920	0.899	0.908
1940	0.815	0.900
2060	0.764	0.785
2220	0.701	0.809
2280	0.829	0.818
2403	0.737	0.750
2462	0.735	0.764
2522	0.910	0.928
2582	0.730	0.742
2583	0.964	0.974
2585	0.749	0.795
2705	0.678	0.691
2905	0.957	0.941
weighted F1-Score	0.790	0.819

Les *weighted f1-score* obtenus par ces modèles de *deep learning* sont meilleurs que ceux de *machine learning*. Ici, le modèle avec la couche *embedding* est de presque 5 points plus performant que le modèle *LogisticRegressionClassifier*. Les produits propices aux erreurs de classification sont en revanche similaires et ce, pour les mêmes raisons. Les modèles de textes testés se trompent sur les mêmes catégories, il est probable qu'en les combinant, les erreurs restent identiques. La concatenation du modèle de texte avec le prochain modèle d'analyse des images pourrait pallier ce problème.

Images

Sélection du modèle

Pour effectuer une première sélection, nous nous sommes appuyés sur les données d'étude Benchmark Analysis of Representative Deep Neural Network Architectures ainsi que sur la documentation de keras affichant sous forme de tableau comparatif les performances de modèles de traitement d'images.

Pour garantir la conformité de la comparaison de nos modèles, les traitements des images doivent être similaires. Compte-tenu du temps imparti à ce projet et en accord avec les paramètres d'entrée des différents modèles utilisés, les images sont réduites d'une taille 500x500 pixels en des images de taille

de 224x224 pixels. La réduction est importante, mais les images restent lisibles, comme le montre l'image ci-dessous :

Les modèles de traitement des images suivant ont été retenus:

- VGG16
- LeNet
- ResNet
- MobileNet

L'entraînement complet de modèles de *deep-learning*, notamment les CNN, est coûteux en temps et en ressources : c'est pourquoi nous avons choisi d'exploiter des modèles pré-entraînés en appliquant des techniques d'apprentissage par transfert.

Étapes de *preprocessing*

Lors de premiers entraînement des modéles présentés avec notre jeu de données, nous avons remarqué leurs grandes sensibilités à l'*overfitting*.

Afin de minimiser au maximum ce risque, plusieurs actions ont été mises en place :

- évolution de couches
- ajout des couches `Dropout` pour réduire les connexions entre les neurones
- augmentation nos données via l'`ImageDataGenerator`
- utilisation de batchs de plus petite taille
- une fois l'entraînement terminé, le modèle est ensuite globalement étudié sur les données test
- suivi de `val_accuracy` et `val_loss` via `tensorboard`

L'utilisation d'un `ImageDataGenerator` (du module `tensorflow.keras.preprocessing.image`) a permis d'effectuer :

- une augmentation de données grâce à l'application des transformations aléatoires paramétrées
- une optimisation de l'utilisation de la mémoire RAM. Les images sont transmises sous de batchs ce qui évite de traiter l'ensemble des données d'un coup. De plus, il s'agit d'un générateur : les images ne sont pas chargées tant qu'elles ne sont pas utilisées.
- un redimensionnement des images rapide

En complément de ces générateurs, le traitement des images doit correspondre aux besoins des modèles pré-entrainés. Pour ajuster notre jeux de données à leurs besoins, nous appliquons un `preprocess_input` spécifique à chaque modèle. Chaque modèle dispose de son propre `preprocess_input`, ce qui en fait une spécificité du modèle : nous estimons que cette étape est liée au modèle et ne fait pas partie du *preprocessing*.

Différentes achitectures

Sont présentées ci-dessous les différentes couches des modèles utilisés.

Architecture VGG16

Le pré-traitement des données est effectué via `preprocess_input` et applique les transformations suivantes :

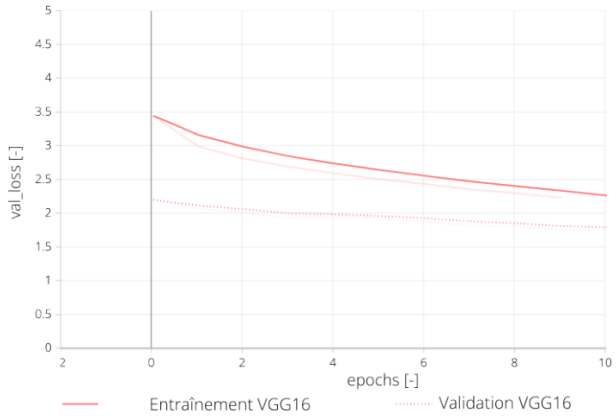
- les canaux de couleurs sont inversés (RGB vers BGR)
- chaque canal est centré autour de la moyenne

Le tableau suivant affiche une structure des couches :

Layer (type)	Output Shape	Param
vgg16 (Functional)	(None, None, None, 512)	14714688
global_average_pooling2d (G lobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 1024)	525312
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524800
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 27)	13851

Total params: 15,778,651
Trainable params: 1,063,963
Non-trainable params: 14,714,688

L'entraînement est présenté ci-dessous, avec en trait plein les scores sur les données d'entraînement et en pointillés, ceux sur les données de validation.



Architecture ResNet

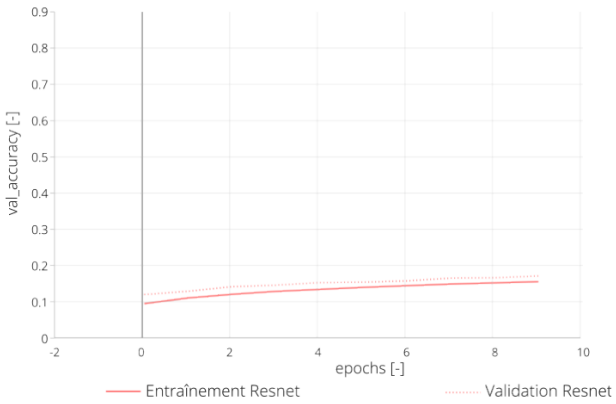
Pour ResNet, le `preprocess_input` a pour objectif de :

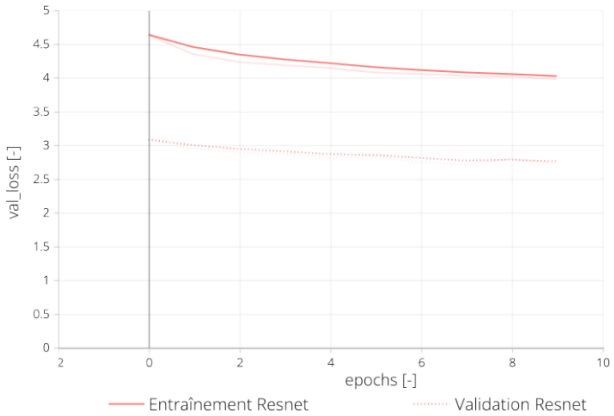
- inverser les canaux de couleurs (RGB vers BGR)
- centrer chaque canal de couleur autour de zéro par rapport au jeu de données ImageNet, sans mise à l'échelle.

Layer (type)	Output Shape	Param
resnet50 (Functional)	(None, None, None, 2048)	23587712
global_average_pooling2d (G	(None, 2048)	0
lobalAveragePooling2D)		
dense (Dense)	(None, 1024)	2098176
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524800
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 27)	13851

Total params: 26,224,539
Trainable params: 2,636,827
Non-trainable params: 23,587,712

Les courbes d'accuracy et de pertes pour l'entraînement et la validation de ce modèle sont visibles sur le graphique suivant.





Architecture MobileNet

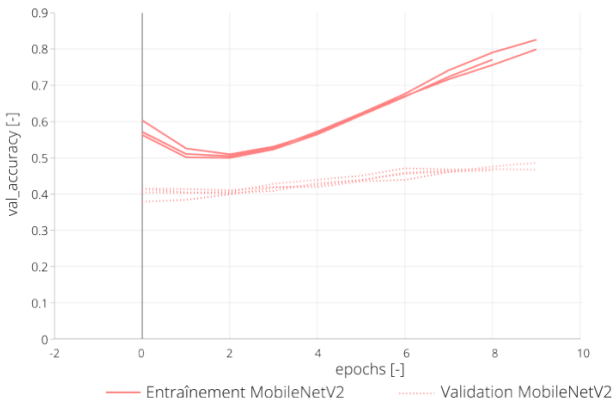
La recommandation principale pour MobileNet en terme de `preprocess_input` est de :

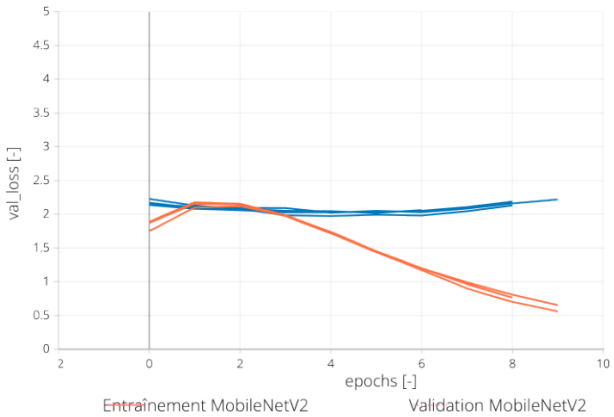
- mettre à l'échelle les valeurs des pixels d'entrée entre -1 et 1.

Layer (type)	Output Shape	Param
Mobilenet(Functional)	(None, None, None, 1280)	2257984
global_average_pooling2d (G	(None, 1280)	0
lobalAveragePooling2D)		
dense (Dense)	(None, 1024)	1311744
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524800
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 27)	13851

Total params: 4,108,379
Trainable params: 1,850,395
Non-trainable params: 2,257,984

Le suivi des métriques de l'entraînement et de la vlidation de MobileNet est proposé dans les graphiques ci-dessous.





Entraînement des modèles

Le tableau suivant compile les valeurs de `accuracy` et `val_accuracy` pour les modèles entraînés.

Model	Accuracy	Val accuracy
VGG16	0.50	0.49
ResNet	0.16	0.18
MobileNet	0.87	0.47

Nous nous sommes fixés comme objectif d'analyser les performances des modèles sur notre jeu de données dans des conditions similaires :

- utiliser une taille des images 224x224
- appliquer des transformations identiques de pré-traitement
- choisir les même paramètres de compilation

Nous sommes conscients que des améliorations, même simple, sont encore possibles : choisir le nombre d'époques pour trouver le compromis rapidité/performance et trouver les paramètres des couches les plus pertinents. Ce genre d'optimisation est particulièrement long à effectuer compte-tenu de la lenteur de ces modèles sur nos machines, sans GPU.

Cependant les tests effectués ont permis d'effectuer plusieurs constats intéressants:

- le modèles MobileNet est trés rapide en entraînement et donne des résultats satisfaisants. C'est le modèle qui présente le meilleur compromis entre la vitesse d'éxécution et scores.
- l'entraînement de la première époque de VGG16 prend plus de temps que les autres. Avec Google Colab, son temps d'entraînement est compris entre 2 et 5h avec accéleration de GPU et 8h sans GPU.
- il est préférable d'utiliser entre 30 et 50 époques pour bien entraîner le modèle. Cependant, Google Colab interrompt les sessions trop longues. De plus, le temps d'utilisation des GPUs est très limité, tout comme l'accès à cette ressource. Le nombre d'époques a donc été limité à 10.
- l'utilisation de callback `EarlyStopping` aide à optimiser l'utilisation de ressources. Lors de test de modèle `EfficientNetB1` l'entraînement s'est arrêté au bout de 5 époques avec une perfomance finale peu satisfaisante. Ce modèle a été retiré de notre sélection.

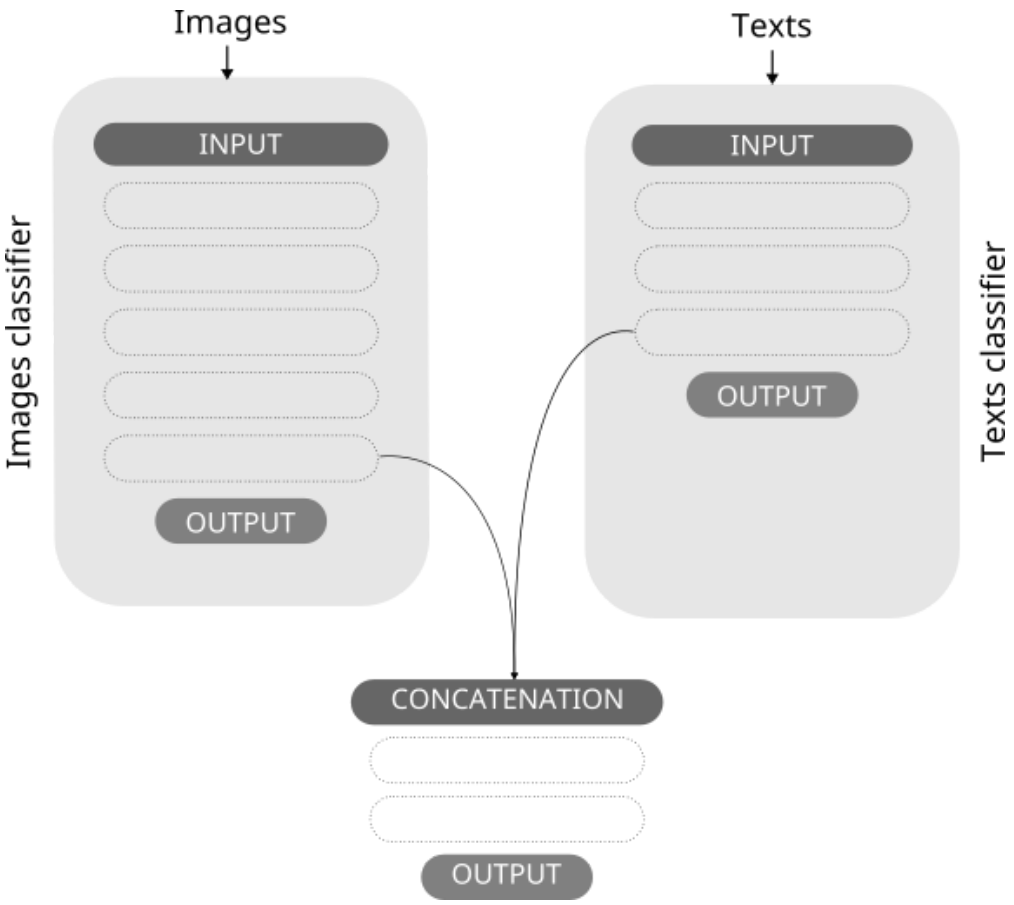
Pour répondre à la problématique, les modèles `VGG16` et `MobileNet` seront à préférer. Ce dernier est par sa légèreté une alternative intéressante face à des modèles plus lourds.

Fusion des modèles images et textes

Les deux modèles peuvent fonctionner indépendemment. Cependant, il est possible de mutualiser leurs capacités en fusionnant ces deux modèles en un unique modèle de classification.

Fusion par concaténation de couches

La fusion est mise en place par concaténation de couches du modèle de texte et du modèle d'image. Cette concaténation est faite sur les avant-dernières couches de nos modèles : ces couches comportent un maximum d'informations. Les autres couches des modèles pré-entrainés sont quant à elles *freezées*. Des couches denses sont ajoutées suite à la concaténation pour obtenir une classification sur 27 classes. Ci-dessous, un graphique simplifié du fonctionnement de cette concaténation.



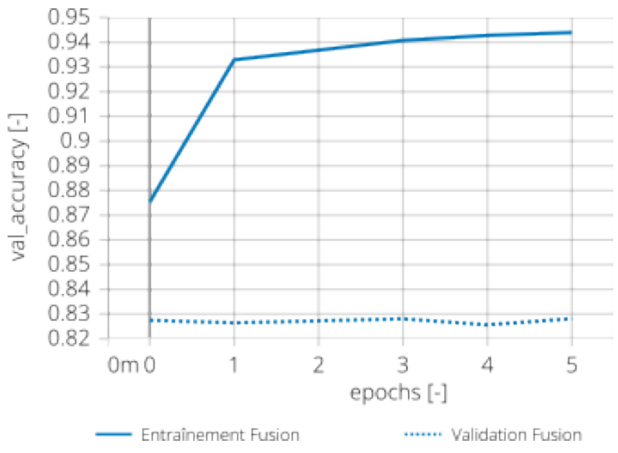
En appliquant cette méthode à nos modèles `text_neural_embedding` et `image_vgg16`, près de 15 millions de paramètres ne sont plus à entraîner. La concaténation a bien lieu au niveau des couches précédant les couches denses de classification.

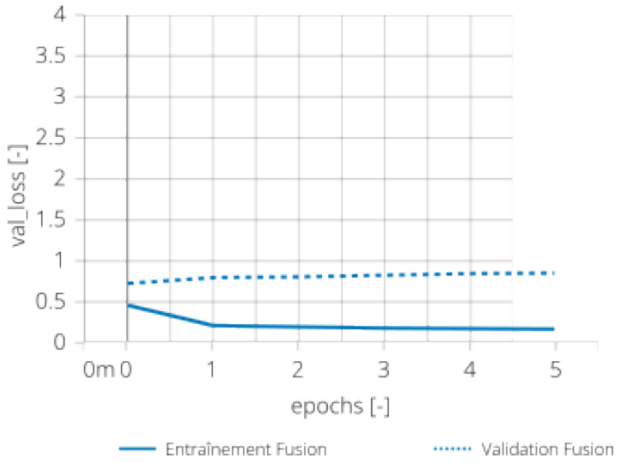
Layer (type)	Output Shape	Param #	Connected to
vgg16_input	[(None, None, None,3)]	0	[]
vgg16_image_CNN_Lenet_1	(None, None, None,512)	14714688	['vgg16_input[0][0]']
global_average_pooling2d_image_CNN_Lenet_1	(None, 512)	0	['vgg16_image_CNN_Lenet_1[0][0]']
flatten_image_CNN_Lenet_1	(None, 512)	0	['global_average_pooling2d_image_CNN_Lenet [0]']
dropout_image_CNN_Lenet_1	(None, 512)	0	['flatten_image_CNN_Lenet_1[0][0]']

Layer (type)	Output Shape	Param #	Connected to
text_dense_1_input	[(None, None)]	0	[]
dense_image_CNN_Lenet_1	(None, 32)	16416	['dropout_image_CNN_Lenet_1[0][0]']
text_dense_1_text_neural_simple_0	(None, 54)	270054	['text_dense_1_input[0][0]']
dropout_1_image_CNN_Lenet_1	(None, 32)	0	['dense_image_CNN_Lenet_1[0][0]']
text_drop_1_text_neural_simple_0	(None, 54)	0	['text_dense_1_text_neural_simple_0[0][0]']
dense_1_image_CNN_Lenet_1	(None, 54)	1782	['dropout_1_image_CNN_Lenet_1[0][0]']
fusion_concat	(None, 108)	0	['text_drop_1_text_neural_simple_0[0][0]','dense_1_image_CNN_Lenet_1[0][0]']
fusion_dense1	(None, 128)	13952	['fusion_concat[0][0]']
fusion_drop1	(None, 128)	0	['fusion_dense1[0][0]']
fusion_output	(None, 27)	3483	['fusion_drop1[0][0]']

Total params: 15,020,375
Trainable params: 19,217
Non-trainable params: 15,001,158

Lors de l'entraînement des avant-dernières et dernieres couches des modèles de textes et d'image ainsi que les couches denses du modèle concaténé, on remarque que le modèle est rapidement en overfitting.





Suite à la concaténation des modèles d'images et de textes, les résultats du modèle sont encourageants avec un *weighted f1-score* supérieur de 0.8%, soit 82.7%.

	fusion_concat_embedding_50
10	0.624
40	0.702
50	0.824
60	0.898
1140	0.741
1160	0.941
1180	0.545
1280	0.680
1281	0.566
1300	0.949
1301	0.943
1302	0.812
1320	0.817
1560	0.820
1920	0.896
1940	0.891
2060	0.781
2220	0.827
2280	0.832
2403	0.783
2462	0.806
2522	0.923
2582	0.735
2583	0.971
2585	0.771
2705	0.805
2905	0.925
accuracy	0.827
macro avg	0.808
weighted avg	0.827

Plusieurs observations sont à faire à ce stade :

- Le modèle concaténé n'est pas impacté par les performances réduites du modèle d'image.
  - Toutes les catégories dépassent le score de 54%
  - Une catégorie sur trois dépasse le score de 90%
  - Un catégorie atteint un score de 99%
- Le modèle concaténé s'aide du modèle d'image pour catégoriser les produits où le modèle de texte sous-performait :
  - La catégorie 1080 (Jeu Plateau) gagne 25 points
  - La catégorie 2705 (Livre neuf) gagne 23 points

Realité	10	40	50	60	1140	1160	1180	1280	1281	1300	1301	1302	1320	1560	1920	1940	2060	2220	2280	2403	2462	252
10	0.60	0.04	0.00	0.00	0.01	0.01	0.02	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.08	0.00	0.0
40	0.06	0.72	0.04	0.01	0.02	0.02	0.02	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.02	0.02	0.02	0.0
50	0.00	0.03	0.84	0.02	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.06	0.0
60	0.00	0.01	0.02	0.90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.0
1140	0.02	0.03	0.00	0.00	0.79	0.01	0.04	0.02	0.02	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.01	0.0
1160	0.00	0.01	0.00	0.00	0.01	0.95	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.0
1180	0.02	0.02	0.01	0.00	0.05	0.00	0.76	0.01	0.03	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.01	0.01	0.02	0.03	0.00	0.0
1280	0.01	0.02	0.01	0.00	0.09	0.01	0.01	0.62	0.11	0.05	0.00	0.02	0.02	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.0
1281	0.03	0.04	0.00	0.00	0.02	0.03	0.05	0.18	0.54	0.00	0.01	0.01	0.01	0.01	0.00	0.00	0.01	0.00	0.01	0.01	0.01	0.0
1300	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.00	0.95	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
1301	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
1302	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.07	0.03	0.00	0.00	0.77	0.02	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.0
1320	0.01	0.00	0.00	0.00	0.01	0.00	0.01	0.03	0.01	0.00	0.00	0.01	0.81	0.02	0.02	0.01	0.02	0.01	0.00	0.01	0.00	0.0
1560	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.81	0.04	0.00	0.04	0.00	0.00	0.00	0.00	0.0
1920	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.94	0.00	0.01	0.00	0.00	0.00	0.00	0.0
1940	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.04	0.00	0.01	0.86	0.01	0.01	0.01	0.01	0.00	0.0
2060	0.01	0.00	0.01	0.00	0.01	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.02	0.05	0.06	0.00	0.73	0.00	0.00	0.01	0.00	0.0
2220	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.03	0.02	0.00	0.01	0.01	0.87	0.01	0.00	0.00	0.0
2280	0.03	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.87	0.06	0.00	0.0
2403	0.06	0.01	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.76	0.01	0.0
2462	0.00	0.08	0.04	0.02	0.02	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.81	0.0
2522	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.01	0.00	0.9
2582	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.06	0.02	0.00	0.05	0.01	0.00	0.00	0.00	0.0
2583	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
2585	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.03	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.0
2705	0.02	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.02	0.00	0.0
2905	0.00	0.02	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0

Conclusion

Après une phase d'exploration, une phase de la création de modèles de classification simple, nous aboutissons sur un modèle complexe issu de la fusion de modèles de *deep learning*. Le *weighted f1-score* atteint pour ce dernier modèle est de 82.9%. Pour rappel, un modèle de classification aléatoire atteindrait seulement un score de 3.7%.

En premier temps, avoir davantage d'informations sur la racine du problème de classification pourrait nous aider à adapter nos modèles. Voici quelques questions que nous nous sommes posées :

- Pourquoi ces produits et ces catégories en particulier? Sont-ils les plus vendus ou les moins bien triés? Nous pourrions ainsi créer une métrique plus appropriée au problème.
- Comment la classification des targets a-t-elle été faite? La labélisation a-t-elle été semi-supervisée ou entièrement réalisée par un oeil humain? La question est légitime, car nous avons nous-mêmes éprouvé des difficultés à trouver une dénomination pour certaines catégories.
- Quel est le prix de vente? Le produit est-il d'occasion ou neuf? Ces informations supplémentaires pourraient nous aider à affiner nos modèles.

Le projet est un projet mêlant de l'analyse de texte et du traitement d'images : des notions poussées de deep-learning sont nécessaires à la compréhension et l'implémentation de telles techniques. De nombreux challenges sont apparus tout au long de ce projet :

- L'accès à des ressources de calcul de type GPU ou TPU nous a été quasi impossible, notamment via Google Collab. Tous les calculs ont été réalisés sans GPU.
- L'accès aux 84 916 images, stockées dans un Google Drive et nécessaires à l'entraînement du modèle d'images, était érattique : de nombreuses coupures de ce lien entre Google Drive et Google Collab ont entraîné ici aussi une grande perte de temps et une grande frustration.
- Des notions relatives aux générateurs, au *deep learning* sont rapidement nécessaires pour la réalisation de ce projet. Il a été délicat d'assimiler rapidement ces méthodes pour les mettre en pratique.
- Le traitement des 84916 images nécessite d'utilisation de générateurs. Ces derniers sont à customiser manuellement afin de permettre une gestion en batch des données textuelles et d'images pour le modèle de fusion.
- La création d'un modèle de fusion a été une tâche ardue, principalement pour la gestion des entrées sous forme de générateurs.
- Les implications et les attentes de chaque membre de l'équipe par rapport à ce projet sont variées. La répartition du travail au sein de l'équipe a été de ce fait relativement déséquilibrée.

L'état de l'art évolue rapidement : des améliorations sur nos modèles pourraient être entreprises et de nouvelles techniques pourraient être mises en place.

- Ajout d'autres modèles au modèle de fusion.
- Uniformisation des données dans le code. Actuellement, des dataframes Pandas, des tableaux Numpy, des générateurs d'images fonctionnent ensemble. Tout pourrait être géré autour d'un seul type de données, comme les tf.data.DataSet.
- Changement de la couche d'embedding ou création d'un modèle parallèle. Le modèle de texte par exemple pourrait être doté d'une couche d'embedding pré-entrainée, par exemple celle issue de CamemBERT : son entraînement a été effectué sur 138GB de données en français. Sur un tel modèle, l'entraînement des 110 millions de paramètres serait extrêmement coûteux en termes de temps et d'infrastructure. Cette couche remplacerait alors notre couche seulement entrainée sur nos données, et liée à une couche dense de classification pourrait améliorer les scores de notre modèle de texte.

Ce projet a été très intéressant, car complexe et faisant appel à des notions avancées mêlant le traitement de textes et le traitement d'images. L'exploration de données, le travail de groupe, les différentes implémentations et sprints ont fait de ce projet un projet répondant, nous l'espérons, aux besoins d'une entreprise.

Enfin, nous souhaiterions remercier chaleureusement toute l'équipe de DataScientest pour cette belle et très enrichissante formation et tout particulièrement notre tuteur, Manu, pour ses conseils, la clareté de ses explications, sa bienveillance et sa patience et pour toutes ses anecdotes de vie d'un Data Scientiste. Merci !