DataScientest • com

Rapport Technique d'évaluation

Rakuten_py

Promotion DEC 2020

Participants :

Fatoumata BARRY
Eric MARCHAND
Emmanuel BONNET
Edgar HIDALGO LOPEZ

1 Introduction

1.1 Contexte

Ce projet s'inscrit dans le challenge ***Rakuten France Multimodal Product Data Classification***: Il s'agit de prédire le code type de produits (tel que défini dans le catalogue Rakuten France) à partir d'une description texte et d'une image.

Note: plus de détails dans annexe 1

Le chiffre d'affaires du commerce en ligne augmente de façon importante depuis plusieurs années (plus de 100% entre 2013 et 2019). L'arrivée de la COVID-19 a accéléré cette tendance qui va certainement s'inscrire dans le temps (étude Médiamétrie-FEVAD).

Les progrès dans le domaine de classification « image plus texte » avaient été limités ces dernières années en raison du manque de données réelles provenant de catalogues commerciaux.

Ces techniques numériques d'automatisation des tâches sont fortement sollicitées par les entreprises de e-commerce, en particulier pour les raisons suivantes:

- Classification de produit à grande échelle pour mieux gérer l'offre d'un site
- Recommandation de produit (génère en moyenne 35% de revenu supplémentaire)
- Amélioration de l'expérience client

1.2 Modèle de référence et objectifs

Un modèle de référence est indiqué par le site du challenge: l'objectif du projet est de faire mieux.

Cette référence est en réalité composée de deux modèles distincts : un pour les images, un pour le texte (les participants sont encouragés à utiliser ces deux sources lors de la conception d'un classificateur, car elles contiennent des informations complémentaires) :

1. Pour les données images, une version du modèle Residual Networks (ResNet), le ResNet50 pré-entraîné avec un jeu de données Imagenet; 27 couches différentes du haut sont dégelées, dont 8 couches de convolution pour l'entraînement.
2. Pour les données textes, un classificateur RNN simplifié est utilisé. Seuls les champs de désignation sont utilisés dans ce modèle de référence.

Les données appartiennent à 27 classes distinctes. Pour évaluer la qualité de la classification, il est demandé d'utiliser la métrique **weighted-F1-score**. Il s'agit de la moyenne des F1-scores de toutes les classes pondérée par le nombre de représentants dans ces classes. Le F1-score de chaque classe est la moyenne harmonique de la précision et du rappel pour cette classe.

Le modèle de référence obtient les résultats suivants :

- **0.5534** pour le modèle image (ResNet)
- **0.8113** pour le modèle texte (RNN)

1.3 Déroulement du projet

Le projet a suivi un plan en plusieurs étapes:

- Chargement et première exploration des données, visualisation des images
- Analyses statistiques
- Modélisation NLP sur les données textuelles
- Modélisation CNN/Transfer learning sur les données images
- Modélisation joignant à la fois les modèles textuels et les modèles images
- Soumission d'un modèle sur le site du challenge
- Rédaction du rapport
- Démonstration

2 Projet

2.1 Exploration et Visualisation des données

La visualisation des données permet de tirer rapidement des informations grâce aux représentations graphiques. Elle s'incorpore dans diverses phases du workflow d'un projet de Data Science. La data visualisation est utilisée par les Data Scientist pour:

- Explorer les données d'une façon visuelle, rapide et simple à assimiler
- Communiquer ses idées et faire passer un message complexe d'une façon aisée en s'appuyant sur des supports visuels intuitifs et facilement compréhensibles par différents lecteurs ou auditoires

2.1.1 Les données d'entrée

Pour ce challenge Rakuten, l'ensemble de données se compose des champs *désignation*, *description*, *product id*, *image id*, et des 'images de produits

Rakuten fournit les fichiers suivants:

- X_train_update.csv: il s'agit d'un tableau contenant 84916 échantillons d'entraînement, avec la description textuelle ainsi que le référencement du fichier image associé
- X_test_update.csv: ce fichier est un tableau de 13812 échantillons de test

Id	designation	description	productid	imageid

Id : cet identifiant est utilisé pour associer le produit à son code correspondant.
designation : le titre du produit, un texte court le résumant.
description : un texte plus détaillé décrivant le produit. Tous les exposants n'utilisent pas ce champ, donc pour conserver l'originalité des données, le champ de description peut contenir une valeur NaN.
productid : un identifiant unique pour le produit.
imageid : un identifiant unique pour l'image associée au produit.

- y_train_CVw08PX.csv: ce fichier correspond à un tableau contenant les 84916 codes produits (prdtypecode).
- images.zip: cela correspond à l'archivage des fichiers images, d'un nombre total de 84916 images dans le répertoire image_train et de 13812 image dans le répertoire image_test.

Le graphique suivant montre la proportion de produits par catégories:

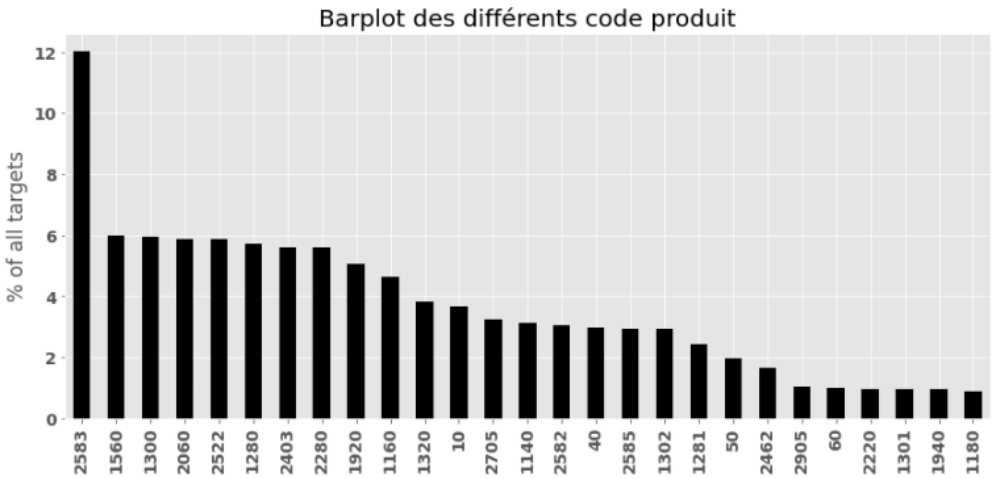


Figure 1 - Répartition par codes produits (en %)

Nous observons que la proportion de produits par catégorie est déséquilibrée, ce qui aura possiblement un impact sur la classification des produits moins représentés.

Les champs *imageid* et *productid* sont utilisés pour récupérer les images du dossier d'images respectif.

```
image_name = 'image'+'_'+str(df['imageid'])+'_'+'product'+'_'+str(df['productid'])+'.jpg'
```

Pour un produit particulier, le nom du fichier image est : ***image_imageid_product_productid.jpg***

Aucune des 27 catégories n'est décrite de manière formelle dans le challenge. L'exploration des images par catégorie nous permet de définir la typologie des produits inclus dans chaque catégorie. Un extrait de quelques images du dataset est affiché ci-après:

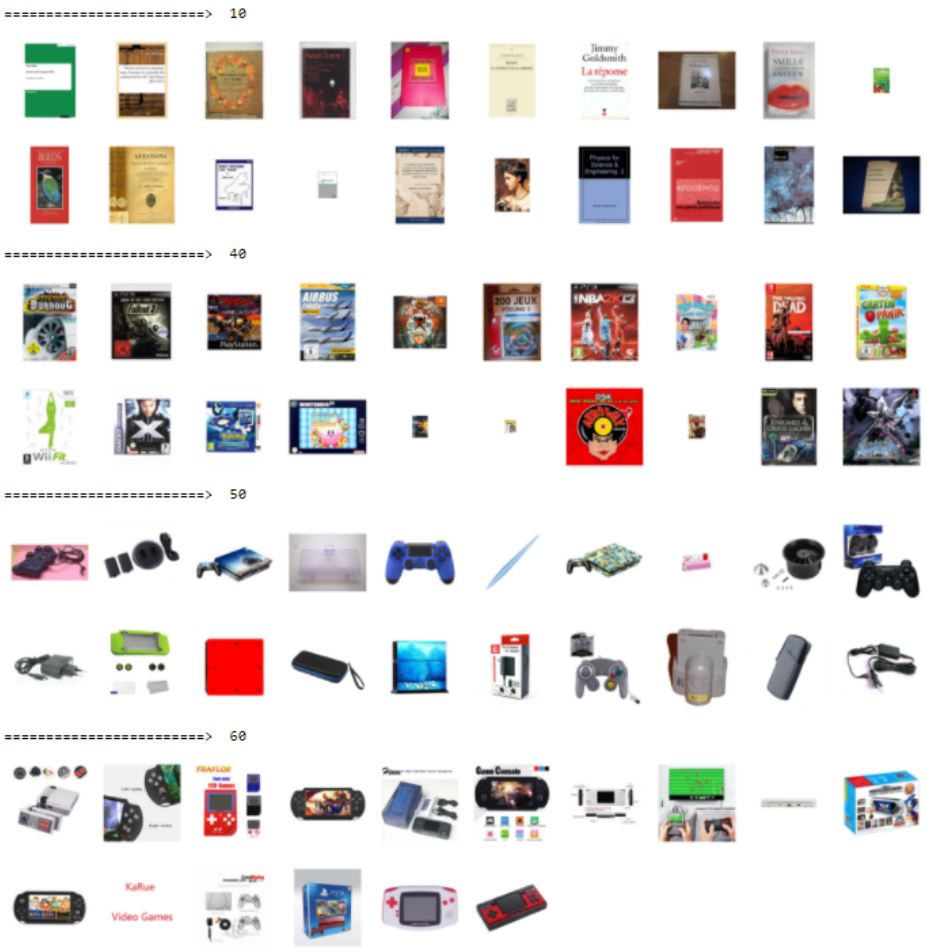


Figure 2 - Exemples d'images par catégorie

Dans l'échantillonnage d'images de la Figure 2,nous observons que les produits de différentes catégories d'images peuvent présenter de fortes similitudes visuelles: cela devrait rendre plus complexe la classification de ces produits en computer vision.

Par exemple : les images de la catégorie 50 et 60 contiennent toutes les deux des consoles de jeux. D'autres similitudes sont observées dans les autres catégories. Cette exploration visuelle nous a permis de nommer les différentes catégories et les grouper par similitudes d'offres produits.

L'ensemble de ces 27 catégories du dataset avec leur description et le nombre d'images associées, est décrite dans le tableau ci après :

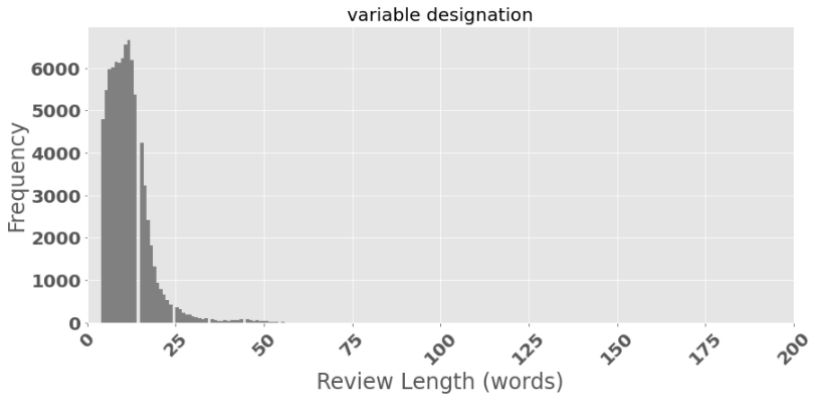
	y	Nombre	Caractéristiques des images
Livres	10	3116	Livres occasion
	2280	4760	Journaux et revues occasion
	2403	4774	Livres, BD et magazines
	2522	4989	Fournitures papeterie et accessoires bureau
	2705	2761	Livres neufs
Jeux	40	2508	Jeux videos, CDs, équipements, câbles, neufs
	50	1681	Accessoires gaming
	60	832	Consoles de jeux
	2462	1421	Jeux vidéos occasion
	2905	872	Jeux vidéos pour PC
Jouets & figurines	1140	2671	Figurines, objets pop culture
	1160	3953	Cartes de jeux
	1180	764	Figurines et jeux de rôles
	1280	4870	Jouets enfants
	1281	2070	Jeux société enfants
	1300	5045	Modélisme
	1302	2491	Jeux de pleins air, Habits
Meubles	1560	5073	Mobilier général : meubles, matelas, canapés lampes, chaises
	2582	2589	Mobilier de jardin : meubles et outils pour le jardin
Equipements divers	1320	3241	Puériculture, accessoire bébé
	2220	824	Animalerie
	2583	10209	Piscine et accessoires
	2585	2496	Outillages de jardin, équipements technique extérieur maison et piscines
Déco	1920	4303	Linge de maison, oreillers, coussins
	2060	4993	Décoration
Autres	1301	807	Chaussettes bébés, petites photos
	1940	803	Confiserie

Figure 3 : Répartition des catégories du dataset Rakuten

Une pré-étude de clustering à partir de l'algorithme Locally Linear Embedding avec K Means clustering, menée sur un échantillonnage d'images du dataset (25000) a montré qu'il ne ressort que quelques clusters (<10), basé sur le score Silhouette. Ce point nous a mis en garde sur le fait que la modélisation des images ne serait pas simple à réaliser car nous avons 27 classes.

2.1.3 Données texte

Les illustrations visuelles de longueur des mots pour les variables *designation* et *description* sont les suivantes :



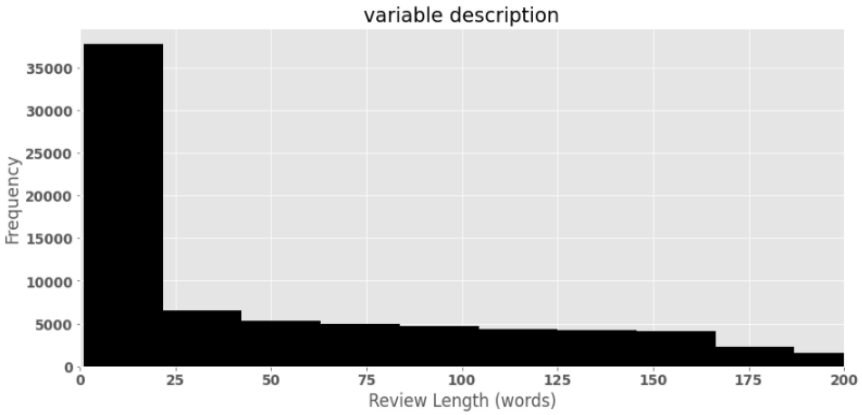


Figure 4 : Taille des mots (tokens) pour les variables *designation* et *description*

Le champ *designation* est plus condensé en information que le champ *description* : en effet, la longueur moyenne des mots pour la variable *designation* est inférieure à celle de la variable *description*. De plus, la variable *description* contient une distribution moyenne de longueur de mots beaucoup plus large que la variable *designation*.

Une illustration de type 'Word Cloud' permet de représenter les mots les plus utilisés communément aux données *designation* et *description* :

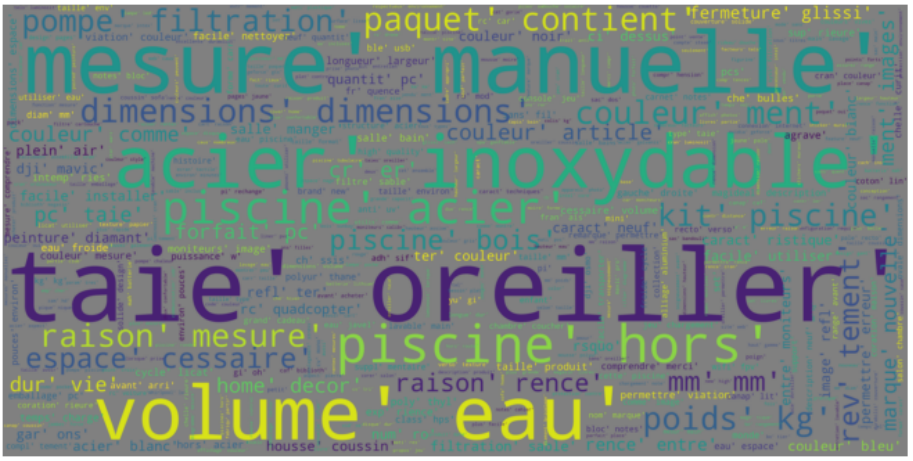


Figure 5 : Word cloud illustratif des variables *designation* et *description* fusionnées

Note: plus de détails dans annexe 2.1

2.2 Modélisation Machine Learning

L'objectif de ce projet est de classifier différents produits dans différentes classes déjà définies. Nous avons développé des modèles de classification supervisés qui ont permis au modèle de classifier les produits en fonction d'une image ou de sa description textuelle.

Dans ce paragraphe nous allons résumer les différents modèles de classification essayés présents dans la librairie scikit learn.

2.2.1 Modèle Images

Dans le cadre du projet nous avons entraîné des modèles afin de classifier les produits avec des données image.

Les données images comportant un grand nombre de caractéristiques (features), l'utilisation d'algorithmes de réduction de dimensions pour la machine learning nous a paru intéressante à étudier dans une première approche:

- Nous sommes partis des images d'entrée 500x500 pixels en RGB (250000 variables)
- Nous avons réduit les images à 50x50 pixels dans l'échelle de gris (2500 variables)
- L'algorithme PCA a réduit le nombre de variables à 99 avec 90% de la variance.

Nous avons comparé les résultats des prédictions avec et sans PCA des classes obtenues avec l'algorithme Random Forest Classifier du jeu de données complet et réduit de 99 variables. Les f1-weighted scores pour chaque dataset est le suivant:

- Pour le dataset complet (2500 variables): 0,44
- Pour le dataset réduit (99 variables): 0,38

En conclusion, bien que les résultats ne soient pas au niveau attendu, car notre objectif est d'obtenir un f1-score supérieur à 55% (score d'un ResNet dans le modèle de référence), il n'en demeure pas moins que cette première approche est intéressante compte tenu de la relative simplicité des algorithmes type ACP ou Random Forest.

Note: plus de détails dans annexes 2.2

2.2.2 Modèle Texte

2.2.2.1 Choix du traitement de texte

L'objectif du projet est aussi de classifier les produits grâce à la *désignation* et la *description* donnée par le vendeur. Pour cet objectif nous avons appliqué différentes techniques de text mining, NLP, tokenisation et la bibliothèque NLTK qui ont permis de transformer les données texte en vecteurs.

Afin de trouver le meilleur traitement possible, différentes combinaisons de traitement de textes (lemmatizer, steeming, stopwords…) ont été intégrées dans un modèle LightSVM comportant les mêmes paramètres par défaut. Les scores obtenus étaient équivalents. Il apparaît donc que

concernant le modèle LightSVM, les méthodes de pré-traitement n'ont pas d'impact sur les scores finaux.

2.2.2.2 Choix meilleur modèle

Pour notre problème de classification supervisée, différents modèles de la librairie scikit-learn ont été essayés afin d'obtenir le meilleur score possible (weighted F1 score). Chaque observation du jeu de données est associée à un texte, provenant de la fusion des champs *designation* et *description*. Le texte est ensuite vectorisé à l'aide de la classe TfidfVectorizer qui a été priorisée par rapport à la classe CountVectoriser. En effet, la classe tf-idf consiste à réduire les poids des mots qui apparaissent dans de nombreux documents du corpus et sont donc moins informatifs que ceux qui n'apparaissent que dans une plus petite partie du corpus.

Afin de raccourcir les temps d'exécution, nos premiers essais ont été effectués avec un jeu de données réduit.

Des essais supplémentaires ont été réalisés avec les classificateurs identifiés comme les meilleurs, en changeant les paramètres des classificateurs eux-mêmes mais aussi de tfidfVectorizer. Le tokenizer utilisé étant toujours celui de Spacy.

Classifier	score	Durée(s)	Principaux paramètres
SVC	0.7929	184	kernel=linear decision_function_shape=ovo
LinearSVC	0.7863	10	penalty=l1 dual=False
MLPClassifier	0.7792	1466	layer_sizes=40, max_iter=40
LogisticRegression	0.7708	109	class_weight=balanced max_iter=200
KNeighborClassifier	0.6459	4	n_neighbors=5 metric=minkowski

Figure 6 : Scoring f1-weighted de nos modèles machine learning partie NLP

En conclusion, les modèles SVC(linear) et Linear SVC, qui sont similaires, donnent les meilleurs scores. Le modèle XG Boost a également donné des résultats de scoring intéressants. Enfin, nous avons constaté que le modèle Linear SVC présente la particularité d'être beaucoup plus rapide à l'entraînement, sur notre jeu de données texte.

La fonction GridSearchCV a été utilisée pour faire une sélection des meilleurs paramètres du modèle LinearSVC. Les meilleurs paramètres trouvés sont penalty="l2", dual=True, C=0.8, tol=1e-5, max_iter=4000, et permettent d'obtenir un **weighted F1 score de 0.85**. Ce modèle est donc très intéressant car il est plus performant que le RNN proposé par l'état de l'art du challenge, avec un weighted score de 0.81.

Note: Nous avons aussi essayé un modèle XGBoost qui a obtenu un score de 0,83.
Note: plus de détails dans annexe 2.2

2.2.2.3 Conclusion Machine Learning

Les modèles machine learning ont prouvé être très rapides et performants pour la classification de texte mais pas pour la classification des images. Nous sommes donc partis sur la modélisation en Deep Learning afin d'essayer d'améliorer ces scores.

2.3 Modelisation Deep Learning

Cette section traite des modèles de classification des données images et textuelles testés puis ceux finalement retenus afin d'intégrer le modèle concaténé de classification de ces deux types de données.

2.3.1 Modèles Images

Afin de résoudre le problème de classification basée sur des données images, les réseaux de neurones convolutifs *(ConvNet)* qui sont une famille de modèles d'apprentissage profond ont été utilisés car ils sont populaires et réputés efficaces pour la vision par ordinateur.

En effet, ces derniers présentent de nombreux avantages comparativement aux modèles contenant uniquement des couches entièrement connectés :

- une capacité de généralisation améliorée grâce à leur capacité à repérer un motif apprit quel que soit sa position dans une nouvelle image, les formes apprises étant invariantes par translation,

- une aptitude à l'apprentissage des hiérarchies spatiales des motifs, de par leur habilité à reconnaître en premier lieu les bords hyperlocalisés des objets, puis les bords localisés de l'objet (reconstitués à partir des bords hyperlocalisés) et enfin les concepts de haut niveaux (générés par un assemblage des bords localisés). Cette habilité permet de détecter méthodiquement les formes usuelles constitutives des éléments des classes.

Après avoir déterminé le type de réseau le plus adéquat pour répondre au problème de classification, un focus a été porté sur la stratégie à employer dans l'application de la méthode.

2.3.1.1 Stratégie

La stratégie employée correspond à celle proposée par François Chollet dans son livre « **L'apprentissage profond avec python - Les meilleures pratiques** ».

Dans un premier temps, les données ont été divisées en trois échantillons distincts : d'entraînement (68% des données), de validation (12% des données) et de test (20% des données). Ceci afin d'évaluer et d'améliorer la robustesse et la capacité de généralisation du modèle.

Par la suite, un réseau de neurones convolutifs classique et simple *(ConvNet)* a été entraîné et différentes méthodes d'optimisation ont été testées. La stratégie finalement retenue a consisté à l'utilisation d'un générateur d'images pour augmenter le nombre d'images de l'échantillon d'entraînement et à l'extraction des caractéristiques d'un réseau pré-entraîné appartenant à la famille des *ConvNet*.

2.3.1.2 Choix du meilleur modèle

En vue de parvenir à la meilleure modélisation possible, différents modèles ont été testés par chacun avec différentes tentatives d'optimisation des paramètres et des hyperparamètres, différents nombre de couches et types de couches et sur un nombre de données réduits (30 à 50% des données). Cette démarche a permis à chacun de monter en compétence individuellement et de tester une pluralité d'éléments.

Ensuite les paramètres utilisés pour le modèle présentant le meilleur f1-score pondéré ont été généralisés aux autres modèles afin d'effectuer une comparaison de leur performance. Les scores finalement obtenus (cf. Figure 7) étaient compris dans l'intervalle [0.5, 0.67].

Modèle	VGG16	ResNet50	InceptionV3	MobileNetV2	DenseNet201	NASNetMobile	EfficientNetB0	EfficientNetB1	EfficientNetB5
f1-score weighted	0,51	0,62	0,50	0,61	0,54	0,60	0,63	0,66	0,67
10	0,39	0,53	0,43	0,53	0,49	0,49	0,53	0,55	0,56
1140	0,52	0,63	0,5	0,58	0,52	0,62	0,66	0,66	0,69
1160	0,85	0,89	0,75	0,88	0,86	0,9	0,89	0,92	0,93
1180	0,09	0,4	0,23	0,39	0,27	0,38	0,44	0,39	0,52
1280	0,32	0,45	0,38	0,44	0,4	0,44	0,44	0,47	0,47
1281	0,16	0,29	0,13	0,27	0,25	0,22	0,27	0,36	0,31
1300	0,54	0,68	0,54	0,69	0,59	0,64	0,73	0,77	0,75
1301	0,36	0,67	0,47	0,63	0,53	0,66	0,68	0,71	0,75
1302	0,27	0,48	0,33	0,47	0,29	0,45	0,49	0,49	0,55
1320	0,34	0,49	0,36	0,47	0,4	0,48	**0,5**	0,49	0,57
1560	0,5	0,56	0,49	**0,59**	0,53	0,58	0,58	0,58	0,61
1920	0,75	0,8	0,73	0,79	0,75	0,81	0,79	0,83	0,83
1940	0,42	0,66	0,52	0,62	0,51	0,64	0,7	0,8	0,75
2060	0,42	0,5	0,42	0,52	0,46	0,52	0,53	0,56	0,59
2220	0,26	0,45	0,23	0,32	0,27	0,4	0,45	0,59	0,52
2280	0,65	0,69	0,59	0,66	0,61	0,67	0,7	0,75	0,71
2403	0,58	0,66	0,55	0,64	0,6	0,65	**0,68**	0,67	0,67
2462	0,42	0,55	0,37	0,49	0,46	0,51	0,6	0,61	0,62
2522	0,58	0,69	0,57	0,66	0,59	0,67	0,69	0,74	0,74
2582	0,29	**0,46**	0,33	0,42	0,35	0,4	0,4	0,43	0,46
2583	0,73	0,82	0,7	0,82	0,75	0,78	0,8	0,85	0,86
2585	0,24	0,45	0,29	0,44	0,34	0,41	0,52	0,6	0,53
2705	0,57	0,67	0,57	0,65	0,61	0,63	0,71	0,72	0,75
2905	0,48	0,56	0,38	0,56	0,36	0,51	0,64	0,72	0,55
40	0,46	0,54	0,37	0,51	0,38	0,49	0,55	0,55	0,6
50	0,27	0,4	0,21	0,4	0,32	0,36	0,44	0,47	0,49
60	0,6	0,67	0,47	0,56	0,62	0,61	0,71	0,72	0,71

Figure 7 : Scoring f1-weighted de nos modèles Computer Vision, incluant le détail par classes

Concernant les résultats par classe, il apparaît que certaines catégories ont été complexes à classer (score f1 < 0.5 pour tous les modèles). Il s'agit plus précisément des labels : **1280** (jouets enfants), **1281** (jeux de société enfant), **2582** (Piscines et accessoires), **50** (Accessoires gaming) et dans une moindre mesure, avec un seul score dépassant ce seuil, : les labels **10** (livres d'occasion) et **1180** (figurines et jeux de rôles).

Au niveau des résultats globaux, plusieurs réseaux ont dépassé le score de référence de *ResNet50* à 0.55, parmi lesquels : *ResNet50* - 0.62, *MobileNetV2* - 0.61, *NasNetMobile* - 0.6, *EfficientNetB0* - 0.63, *EfficientNetB1* - 0.66 , *EfficientNetB5* - 0.67.

Bien que *EfficientNetB5* présentait le meilleur résultat, le score obtenu dépassait légèrement le second modèle plus performant (de 0.01 point) pour un temps d'exécution additionnel d'environ 75%. Compte tenu des contraintes opérationnelles relatives au processus de concaténation final des modèles, l'*EfficientNetB1* a été retenu en tant que réseau pré-entraîné favori pour la classification des images avec un **weighted f1-score** de **0.66**.

Par ailleurs, des tentatives d'optimisation du réseau ont été réalisées. Celles-ci visaient à la concaténation de modèles images afin d'optimiser les résultats obtenus sur certaines classes. Ainsi, le réseau *Resnet50* a été concaténé à l'*EfficientNetB1* car il présentait des meilleurs résultats au niveau de la classe 2582 (+0.03). Cependant, le *modèle concaténé de classification d'images* n'était pas plus performant qu'*EfficientNetB1*. D'autres réseaux présentaient

également de légères améliorations (+0.01), notamment *MobileNetV2* et *EfficientNetB0*, mais ils n'ont pas été intégrés au modèle concaténé compte tenu du faible gain attendu.

Ainsi, seul le modèle avec le réseau pré-entraîné *EfficientNetB1* a été utilisé.

2.3.1.3 Résultats de la modélisation

Le modèle de classification d'images finale a, durant sa seconde phase, utilisé 2 *callbacks (ReduceLROnPlateau et EarlyStopping)*. Ainsi, le taux d'apprentissage a été ré-ajusté à 2 reprises, lors de la 10ème et de la 17ème époque finissant à 8e-6. De plus, l'entraînement du modèle a été interrompu à la 18ème époque.

Au final, le réseau a atteint une précision de 0.7244 et une précision de validation de 0.6580. Le F1-score pondéré a finalement été évalué à 0.6614.

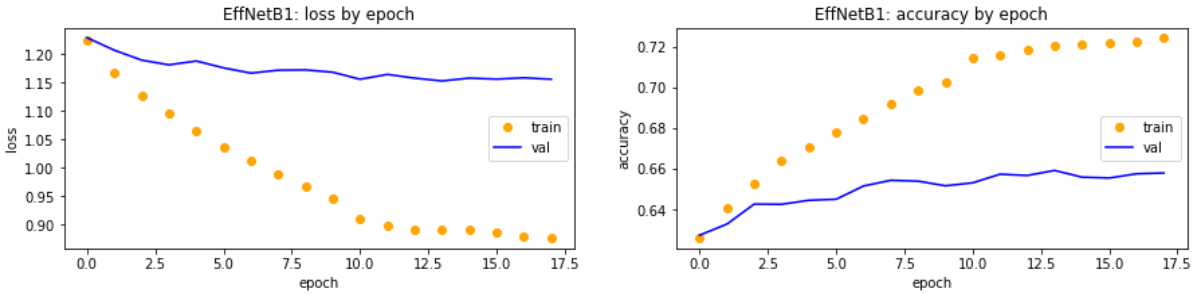


Figure 8 : Résultat détaillé de l'entraînement du modèle EffNetB1

Note: plus de détails dans annexe 2.3

En parallèle au développement d'un modèle de classification d'image, des modèles de classification des données textuelles ont été réalisés afin de répondre au même problème posé.

2.3.2 Modèles Texte

2.3.2.1 Strategie

Concernant les données textuelles, comme pour la partie NLP text, la stratégie a consisté à regrouper les deux variables textuelles *description* et *designation* en une variable unique, ceci afin d'exploiter un maximum d'informations pour le problème de classification.

Il existe plusieurs façons d'associer un vecteur à un token : le one hot encoding ainsi que le token Embedding, sans oublier les techniques de vectorisation issues du machine learning (CountVectorizer, TfidfVectorizer). Les vecteurs obtenus par encodage one hot sont binaire (composés de 0 et de 1), clairsemés et de très haute dimension (même dimensionnalité que le nombre de mots dans le vocabulaire), les word Embeddings sont des vecteurs de faibles dimensions (c'est-à-dire des vecteurs denses). D'autre part, les mots d'encodage one hot conduisent généralement à des vecteurs de dimension 20 000 ou plus (capturant un vocabulaire de 20 000 tokens, dans ce cas). **Les word Embeddings permettent le stockage de davantage d'informations dans beaucoup moins de dimensions :** ils sont destinées à cartographier le langage humain dans un espace géométrique.

2.3.2.2 Choix du meilleur modèle

Comme pour la partie computer vision, une analyse basée sur la performance des f1-scores pondérés par classes a été menée. La synthèse des f1-scores pondérés de l'ensemble des modèles testés en intégralité (une ou plusieurs fois), c'est à dire jusqu'à une évaluation classe par classe, est la suivante :

	TextOneHot 1	Multilingual-large 1	Glove	Autre modele Embedding	Multilingual-large 2	Multilingual-large 3	Multilingual-large 4	TextOneHot 2	EmbedRNN
f1-score weighted	**0,8406**	**0,7763**	**0,75**	**0,77**	**0,78**	**0,797**	**0,7953**	**0,8471**	**0,8054**
10	0,6	0,59	0,43	0,43	0,6	0,61	0,6	0,59	0,54
1140	0,8	0,74	0,57	0,57	0,73	0,72	0,73	0,81	0,74
1160	0,95	0,89	0,71	0,75	0,91	0,92	0,91	0,95	0,92
1180	0,69	0,54	0,84	0,83	0,54	0,63	0,54	0,73	0,58
1280	0,71	0,58	0,72	0,71	0,61	0,62	0,61	0,74	0,7
1281	0,62	0,57	0,8	0,83	0,55	0,55	0,55	0,65	0,48
1300	0,93	0,83	0,54	0,46	0,86	0,88	0,86	0,95	0,94
1301	0,93	0,85	0,58	0,6	0,87	0,88	0,87	0,94	0,9
1302	0,83	0,72	0,47	0,54	0,72	0,75	0,72	0,85	0,77
1320	0,86	0,72	0,83	0,86	0,72	0,74	0,72	0,86	0,76
1560	0,85	0,76	0,89	0,89	0,76	0,79	0,76	0,85	0,83
1920	0,92	0,86	0,71	0,75	0,87	0,89	0,87	0,92	0,9
1940	0,92	0,84	0,7	0,75	0,92	0,89	0,92	0,93	0,74
2060	0,81	0,71	0,77	0,82	0,76	0,78	0,76	0,82	0,78
2220	0,86	0,74	0,86	0,91	0,85	0,83	0,85	0,86	0,76
2280	0,77	0,85	0,79	0,8	0,84	0,85	0,84	0,79	0,85
2403	0,79	0,74	0,74	0,79	0,78	0,79	0,78	0,78	0,76
2462	0,81	0,81	0,71	0,77	0,8	0,81	0,8	0,8	0,73
2522	0,92	0,86	0,66	0,7	0,88	0,88	0,88	0,93	0,9
2582	0,78	0,65	0,69	0,74	0,69	0,72	0,69	0,77	0,73
2583	0,98	0,95	0,67	0,68	0,96	0,97	0,96	0,98	0,97
2585	0,88	0,72	0,87	0,9	0,78	0,8	0,78	0,87	0,76
2705	0,74	0,76	0,69	0,7	0,73	0,76	0,73	0,74	0,72
2905	0,97	0,98	0,96	0,96	0,98	0,98	0,98	0,99	0,98
40	0,74	0,67	0,7	0,74	0,71	0,71	0,71	0,73	0,64
50	0,85	0,76	0,7	0,68	0,82	0,84	0,82	0,85	0,77
60	0,9	0,81	0,97	0,97	0,88	0,92	0,88	0,92	0,84

Figure 9 : Scoring f1-weighted de nos modèles Deep NLP, incluant le détail par classes

De cette synthèse Deep NLP, nous avons observé qu'en dépit de la bonne performance globale des modèles, certaines classes restent compliquées à classer (score f1-weighted < 0.7 pour tous les modèles) il s'agit principalement des **classes 10** (livres d'occasion) , **1281** (jeux de société enfant) , même si nous pourrions inclure dans une moindre mesure les classes **1180** (figurines et jeux de rôles), **1280** (jouets enfants). A noter que ces 4 classes ont également posé problèmes pour la partie classification en computer vision.

De cette synthèse, nous avons également conclus que les deux meilleurs modèles obtenus pour la partie Deep Learning Text, toutes classes confondues ont été les suivants :

Modèle EmbedRNN (f1-weighted score : 0,8054) : le preprocessing des données texte est effectué via l'API Keras (tf.keras.preprocessing.text.Tokenizer). Le text 'préprocessé' vient ensuite alimenter un modèle relativement simple, basé sur l'utilisation d'un Embedding et RNN (GRU).

Modèle TextOneHot (f1-weighted score : 0,8471). Il s'agit d'un modèle plus sophistiqué alliant un mélange des techniques de machine learning et de deep learning.
Le texte est tout d'abord 'tokenizé' et lemmatisé en utilisant la bibliothèque Spacy (fonction get_X_text_spacy_lemma()), puis retourné pour alimenter un pipeline sklearn de preprocessing de type TfidfVectorizer | SelectFromModel(LinearSVC). Les données de sortie de ce pipeline de machine learning viennent enfin alimenter un modèle simple de deep learning pour effectuer la classification finale.

Important : Le modèle EmbedRNN n'est pas mieux que le Modèle TextOneHot de type LinearSVC qui utilise TF-IDF. Nous pouvons en déduire que l'ordre des mots (séquentialité) n'est pas une caractéristique importante pour la classification des produits de ce dataset.

2.3.2.3 Résultats de la modélisation

Pour notre meilleur modèle Deep Texte (**Modèle TextOneHot - f1 score : 0,8471)**, les résultats de la modélisation ont été les suivants :

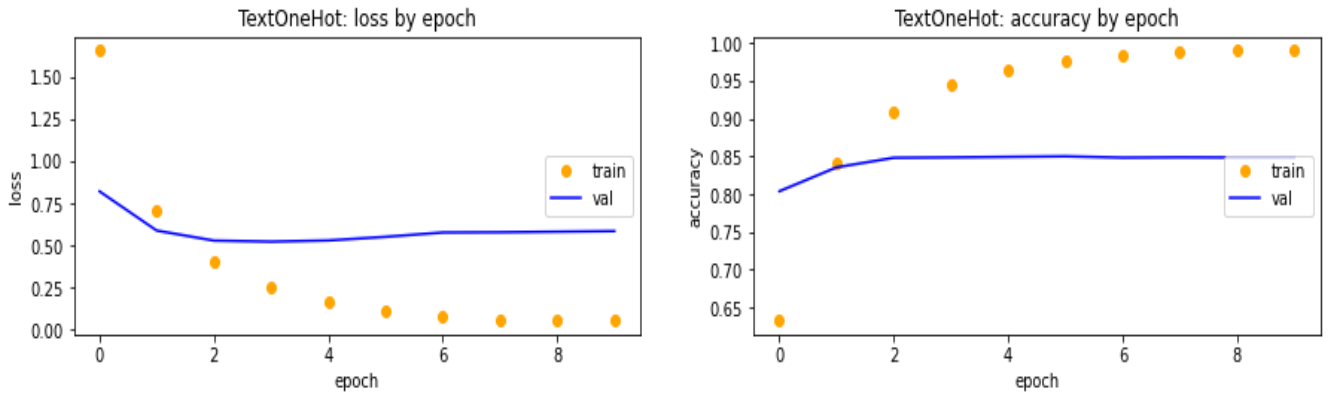


Figure 10 : Résultat détaillé de l'entraînement du modèleTextOneHot

Au cours de la phase d'entraînement de ce modèle, le taux d'apprentissage s'est ré-ajusté 1 fois, lors de la 7ème époque finissant à 0.00001. L'entraînement du modèle a été interrompu dès la 10ème époque par le callback Early Stopping.

A noter que ces courbes d'entraînement sont moins représentatives que celles de la partie computer vision, car la grande majorité du traitement des données texte a été effectuée sur un modèle machine learning (Linear SVC), et la partie Deep learning se limitant à un modèle minimal d'une seule couche dense, suivie d'une étape de dropout puis de batch normalization.

Note: plus de détails dans annexe 2.3

En parallèle au développement des modèles de classification texte et image, nous avons travaillé à la concaténation de nos modèles afin de réaliser un modèle global Concatenate.

2.3.3 Modèles concatenate

2.3.3.1 Strategie et implémentation

Un des points centraux du projet a été de combiner des modèles sur des données de formes très différentes pour obtenir un modèle qui fasse mieux que chacun des modèles de base. Lors de l'exploration des techniques Machine Learning, des méthodes de type ensemble ont été essayées telles que le Voting ou le Stacking. Le passage aux réseaux de neurones nous a conduit à étudier les méthodes de combinaison de sous modèles proposées par Keras, opérantes au choix par addition, multiplication, dot product, moyenne, minimum ou maximum, mais aussi par concaténation, à l'aide de la couche concatenate.

Une fois que chaque modèle de base avait été exécuté (entraîné puis sauvegardé sur disque), le modèle concaténant pouvait être lui-même exécuté.

Il doit être préalablement configuré avec une liste de modèles de base (au minimum deux). Il effectue alors les traitements suivants pour son propre entraînement :

1. Les objets associés aux modèles de base, déjà exécutés, sont chargés en mémoire

2. Les données d'entrée (texte ou image) sont transformées avec les méthodes de preprocessing de chacun des modèles de base. Un générateur est instancié afin d'alimenter l'entraînement en lots de données (l'implémentation est basée sur la classe Sequence de Keras)

3. Le modèle concaténant est alors construit à partir des couches suivantes :

    - Les couches des modèles de base (obtenues par appel à une méthode générique exportée par chacun des objets de base)
    - Des couches éventuelles ajoutées à chaque sortie des modèles de base et avant la couche Concatenate
    - La couche Concatenate
    - Des couches éventuelles après la couche Concatenate
    - La couche de classification Dense(NB_CLASSES, activation= softmax)

4. Les poids des couches des modèles de base (déjà entraînés) sont copiés dans les couches correspondantes du modèle concaténant.

5. Le modèle complet est compilé et entrainé en une ou deux étapes, après congélation et décongélation de certaines couches

2.3.3.2 Choix du meilleur modèle

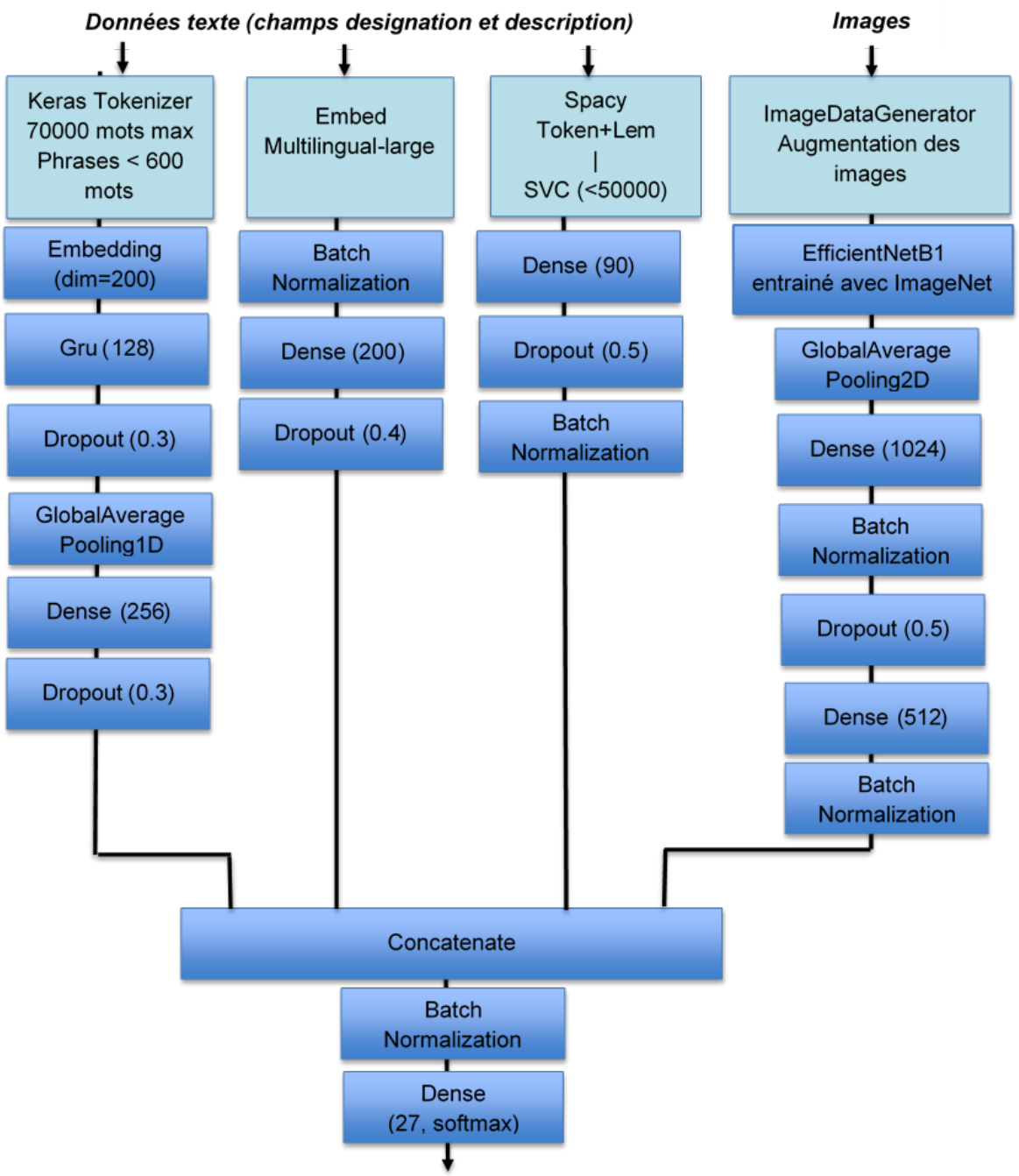


Figure 11 : Architecture de notre modèle Concatenate

Note: plus de détails dans annexe 2.3bis

2.3.3.3. Choix du meilleur modèle

Comme pour les sous modèles Computer Vision et Deep NLP, une analyse basée sur la performance des f1-scores pondérés par classes a été menée. La synthèse des f1-scores pondérés pour les différents tests menés sur le modèle final Concatenate EmbedRNN, OneHot, Multilingu, EfficientNetB1**,** est la suivante **:**

	f1 weighted																												
Models	weighted	10	1140	1160	1180	1280	1281	1300	1301	1302	1320	1560	1920	1940	2060	2220	2280	2403	2462	2522	2582	2583	2585	2705	2905	40	50	60
EmbedRNN, OneHot, Multilingu, EfficientNetB1 **TEST 1**	**0,8776**	0,78	0,86	0,98	0,7	0,76	0,66	0,95	0,92	0,83	0,84	0,85	0,91	0,92	0,81	0,87	0,89	0,87	0,86	0,93	0,77	0,98	0,86	0,91	0,99	0,83	0,89	0,93
EmbedRNN, OneHot, Multilingu, EfficientNetB1 **TEST 2**	**0,878**	0,76	0,84	0,98	0,68	0,76	0,67	0,97	0,94	0,84	0,86	0,86	0,93	0,95	0,82	0,88	0,89	0,85	0,84	0,94	0,77	0,98	0,86	0,9	0,99	0,82	0,88	0,92
EmbedRNN, OneHot, Multilingu, EfficientNetB1 **TEST 3**	**0,8783**	0,72	0,84	0,98	0,72	0,77	0,67	0,96	0,94	0,86	0,87	0,86	0,93	0,97	0,83	0,88	0,88	0,84	0,85	0,94	0,78	0,98	0,88	0,88	0,99	0,81	0,87	0,93

Figure 12 : Scoring f1-weighted de notre modèle Concatenate incluant le détail par classes

De cette synthèse de notre modèle final, il apparaît qu'en dépit de la bonne performance globale, certaines classes sont restées compliquées à classer (score f1-weighted < 0.8) il s'agit principalement des **classes 10** (livres d'occasion), **1180** (figurines et jeux de rôles), **1280** (jouets enfants) et **1281** (jeux de société enfant) . Ces résultats ne sont pas surprenants car ces 4 classes ont également posé problèmes pour la partie classification en computer vision et Deep NLP : **nous pouvons donc douter de la qualité et/ou uniformité des données textes et images mises à disposition pour ces 4 classes spécifiques.**

2.3.3.2 Résultats de la modélisation

Pour notre meilleur modèle Concatenate (**Modèle EmbedRNN, OneHot, Multilingu, EfficientNetB1 - f1 score : 0,8783)**, le résultats de la modélisation après décongélation de certaines couches est le suivant :

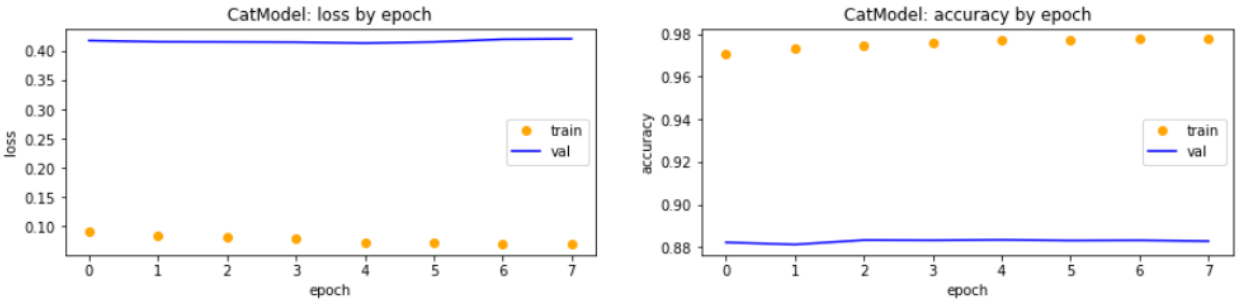


Figure 13 : Résultat détaillé de l'entraînement du modèle final Concatenate

Les résultats de scoring f1 weighted sur les données test de notre modèle final Concatenate sont encourageants (de l'ordre de 0,88). En revanche, les résultats obtenus démontrent la présence d'overfitting résiduel de notre modèle, ce qui reste un point d'amélioration.

2.3.3.3 Matrice de confusion du modèle final Concatenate sur les mauvaises prédictions

Afin d'analyser en détail les mauvaises prédications finales, nous avons réalisé une matrice de confusion du modèle final, sur les données de classifications erronées :

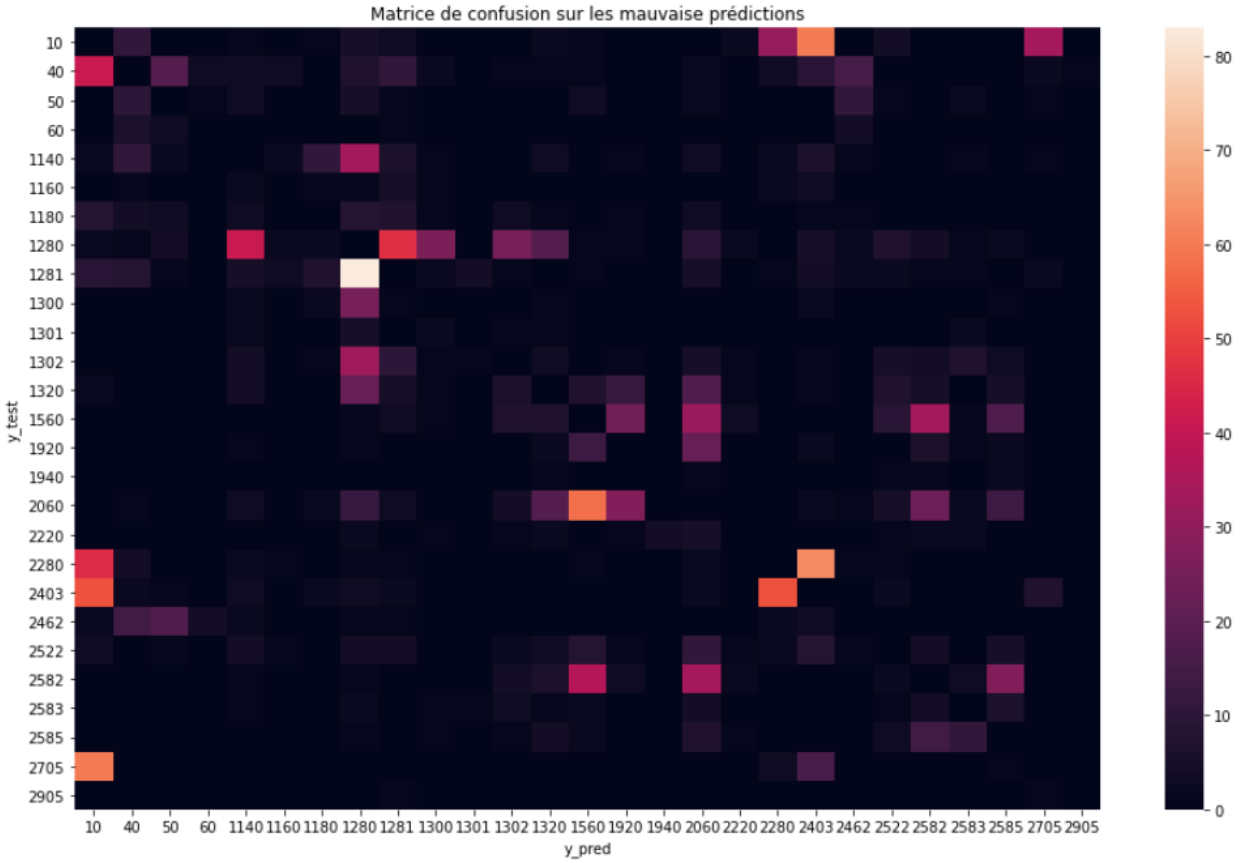


Figure 14 : Matrice de confusion pour les mauvaises prédictions du modèle Concatenate

L'analyse des 20 classes ayant le plus mauvais comportement dans l'algorithme sélectionné a montré que 3 classes semblent plus problématiques car non seulement elles ont beaucoup d'erreur au sein de leurs propres classes mais également, elles génèrent de nombreuses erreurs dans les autres classes de thèmes de produits totalement différents : **1280** (jouets enfants), **1560** (mobilier général) et **2060** (Décoration).

Parmi les autres classes gênantes par leur nombre d'erreurs et le contenu de leur erreurs (sur d'autres thèmes produits), mais générant peu d'erreurs dans les autres classes, nous pouvons citer : **40** (jeux vidéos), **1320** (puériculture), **2582** (mobilier de jardin) et **1920** (Linge de maison).

2.3.4 Conclusions du modèle choisi

Le modèle final est issu de la sélection puis concaténation des meilleurs modèles individuels texte et image. La sélection des modèles individuels est elle même issue d'un processus itératif multi boucles de nos différents modèles machine learning et deep learning texte et images.

Le détail des codes réalisés par l'équipe au cours du projet se trouve via le liens suivant :
https://github.com/Ragdehl/Rakuten_py/tree/main/Livrables

Les résultats de scoring f1 weighted que nous avons obtenus en interne, à la date de diffusion de ce rapport sont de **0.8783**. Par ailleurs, nous avons soumis nos prédictions le 9 août 2021 sur le site du challenge, et notre équipe FEEEScientest a atteint un score de **0.8628**, ce qui nous classe actuellement dans les 10 premières équipes.

Ranking	Date	User(s)	Public score
1	Dec. 10, 2020, 9:35 p.m.	elieS	0.9037
2	April 7, 2020, 1:05 p.m.	Shiro	0.8957
3	Nov. 15, 2020, 10:03 a.m.	kobayashi_shu	0.8940
4	Oct. 18, 2020, 9:51 p.m.	julienC	0.8903
5	March 8, 2020, 3:38 p.m.	Binouze & BlueDrey & JojoFlower	0.8852
6	July 16, 2021, 6:30 p.m.	tbierlaire & meriem_e & vincent__	0.8841
7	Jan. 22, 2021, 3:46 p.m.	NadirEM	0.8714
8	Jan. 25, 2021, 2:08 a.m.	EmmanuelJunior.WafoWembe	0.8708
9	Aug. 9, 2021, 11:01 a.m.	FEEEScientest	0.8628

Nous pensons avoir la possibilité d'améliorer notre score et donc notre classement d'ici la présentation projet, programmée fin Septembre 2021. Pour ce faire, nous allons évaluer l'incorporation d'un nouveau modèle stacking issus des modèles XG_Boost & Linear SVC dans le modèle concatenate. Nous pensons également tester la parallélisation de plusieurs modèles computer vision, par exemple les modèles EfficientNetB1 et B5 , voir d'autres combinaisons si le temps nous le permet.

Une autre approche possible serait de tester des transformeurs viT dans la partie Computer Vision, transformateurs utilisés en NLP, par exemple le modèle BERT, mais nous n'aurons vraisemblablement pas le temps pour étudier ce champ de possibilités d'ici fin Septembre.

De plus, une tentative est en cours afin d'augmenter l'échantillon des classes mal prédites, ceci dans le but d'améliorer leurs scores.

Note: plus de détails dans annexe 2.3ter

D'une manière générale, la définition relativement imprécise des 27 classes, présentant des objets avec de fortes similarités visuelles inter classes, complexifie et limite notablement les possibilités de la phase d'apprentissage en computer vision.

3 Bilan

Ce projet nous a permis de mettre en application les méthodes de machine et deep learning apprises au cours de ce Master Datascientest. Cette mise en pratique grandeur nature, sur ce projet relativement complexe de par la nature diversifiée et hétérogène des données, nous a permis de monter progressivement en savoir-faire, chaque membre de l'équipe apportant au groupe ses idées et ses compétences au fur et à mesure de l'avancée du projet.

Les challenges que nous rencontrées pendant le projet ont été les suivantes :

- La formation en deep learning était assez loin dans la formation. Ce qui ne nous a pas donné assez de temps pour assimiler et appliquer certaines connaissances pratiques au projet,
- Gestion des temps selon les plannings professionnels des uns et des autres,
- Différents backgrounds en informatique entre coéquipiers,
- La correction des bugs qui est assez chronophage et frustrante des fois,
- La taille des données demandait des équipements assez puissants en mémoire et le travail a dû être effectué sans GPU (Google Collab peu performant et carte graphique à GPU peu puissante)
- Les modèles deep learning pour la classification d'images ont des temps d'entraînement très élevés et souvent avec peu d'amélioration en terme de score,
- La différence entre classes n'était pas évidente, même à l'œil humain.

Le bilan technique avec un résultat de scoring dépassant le benchmark, et nous positionnant dans le top 10 est encourageant.

Le bilan projet a montré une très bonne collaboration et entente entre les différents membres de l'équipe, et nous motive à continuer de collaborer ensemble pour continuer à optimiser ce projet mais également travailler ensemble sur des projets futurs afin de parfaire notre formation de Datascientist.

4 Annexes

5 Annexe 1 - détails sur Rakuten

Rakuten, Inc. *est une société japonaise de services internet créée en février 1997 sous le nom de MDM Inc. En juin 1999, MDM, Inc. change de nom pour Rakuten. En 2008, le chiffre d'affaires de Rakuten dépasse 1,1 milliard de dollars avec un bénéfice d'environ 320 millions de dollars. La société est cotée en bourse avec une capitalisation de plus de cinq milliards de dollars et plus de 3 700 employés. Le 17 juin 2010, Rakuten annonce l'acquisition de PriceMinister, premier site de commerce électronique en France10.*

En juillet 2010, Rakuten achète la société américaine Buy.com, spécialisée dans la vente en ligne d'électronique et d'ordinateurs.

En septembre 2011, Rakuten achète la société britannique Play.com, spécialisée dans la vente en ligne de DVD, CD, livres, jeux vidéo, gadgets, mais également de vêtements.

En novembre 2011, elle annonce l'acquisition de la société canadienne Kobo Inc, spécialisée dans les lecteurs de livres numériques, pour 315 millions de dollars. C'est la société Kobo Inc qui fournit depuis novembre 2011 la liseuse Kobo Touch, qui était vendue exclusivement par la chaîne de magasins Fnac avant d'être également vendue par PriceMinister.

En octobre 2012, Rakuten acquiert Aquafadas grâce au rachat effectué par Kobo Inc. Aquafadas est alors l'un des leaders de l'édition numérique et classé numéro 1 des entreprises technologiques françaises. Rakuten se retrouve en une position de leader de l'édition et la publication numérique.

En novembre 2012, Rakuten acquiert Alpha Direct Services, leader de la logistique fine au service des acteurs du commerce en ligne et du multicanal.

En février 2014, Rakuten acquiert Viber Media, qui édite le logiciel Viber permettant de faire de la VoIP sur smartphone, pour 900 millions de dollars (660 millions d'euros). En septembre 2014, Rakuten acquiert le site de programme de fidélisation américaine Ebates pour 1 milliard de dollars.

En mars 2015, Rakuten acquiert OverDrive, une entreprise spécialisée dans le livre numérique pour 410 millions de dollars.

En juillet 2016, Rakuten annonce l'acquisition de Nextperf, société française spécialisée dans l'optimisation en temps réel de la publicité en ligne.

En novembre 2016, le FC Barcelone annonce qu'un accord de sponsoring a été trouvé avec Rakuten qui devient le principal sponsor du club catalan à partir de juin 2017. Rakuten versera au club de football environ 60 millions d'euros par saison jusqu'en 2021.

En janvier 2017, le groupe engage à la tête de la société Viber média dont il est propriétaire depuis 2014, l'entrepreneur français Djamel Agaoua et ancien vice-président de l'entreprise

chinoise Cheetah Mobile, avec mission de recréer une dynamique du succès des services qu'offre la société.

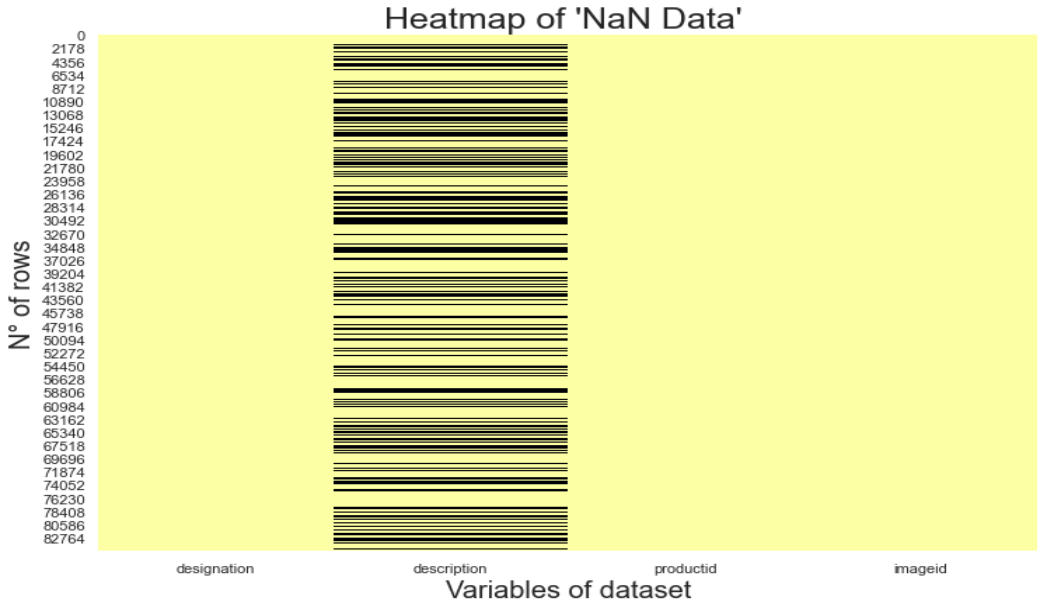
En janvier 2018, Rakuten annonce l'acquisition d'Asahi Fire & Marine Insurance à Nomura Holdings pour 45 milliards de yens (333 millions d'euros)21.

En mars 2018, Rakuten annonce l'abandon de la marque PriceMinister, pour ne garder que le nom Rakuten.

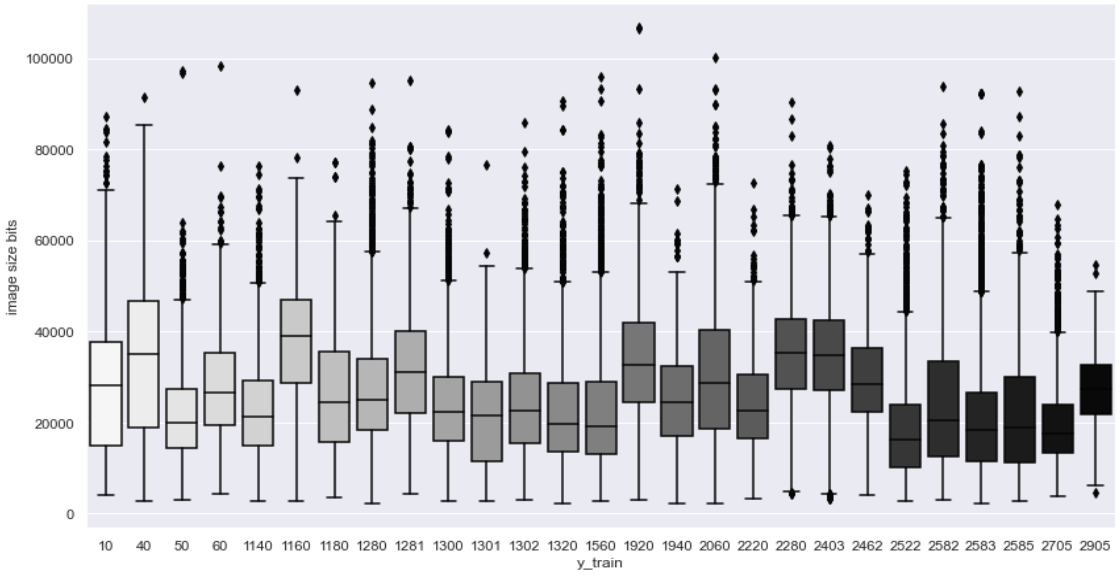
Le chiffre d'affaires de Rakuten France n'est pas publié

5.1 Annexe 2.1 - Exploration et Visualisation des données

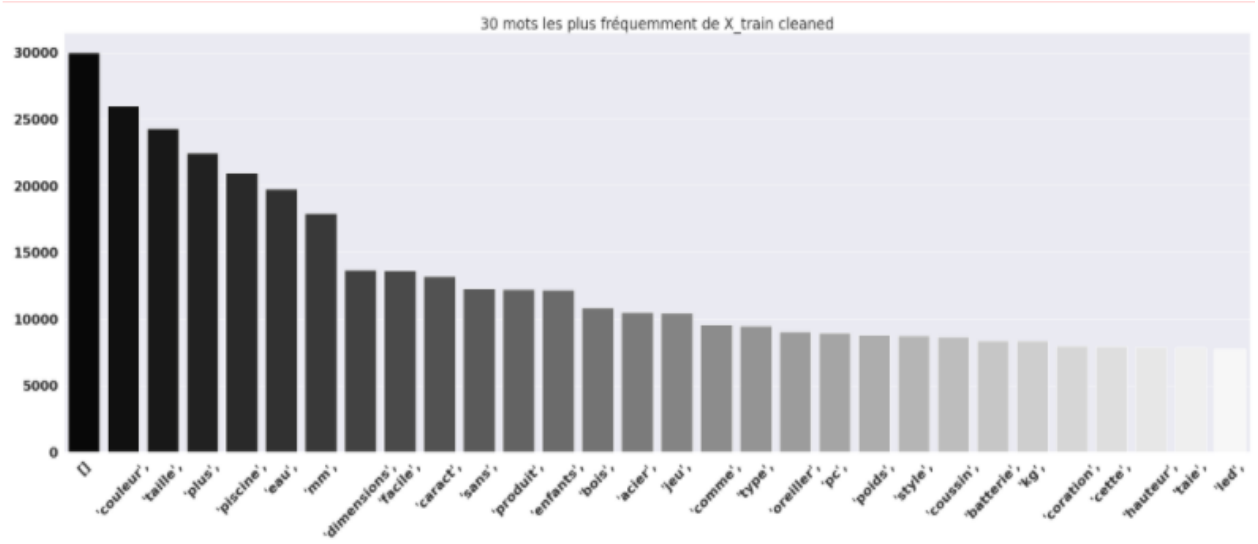
L'observation des données manquantes du dataframe nous montre que celles ci sont concentrées uniquement sur la colonne description



Le boxplot suivant montre la distribution de la taille de stockage des images (en bits, soit ⅛ ème d'octet) par catégorie :



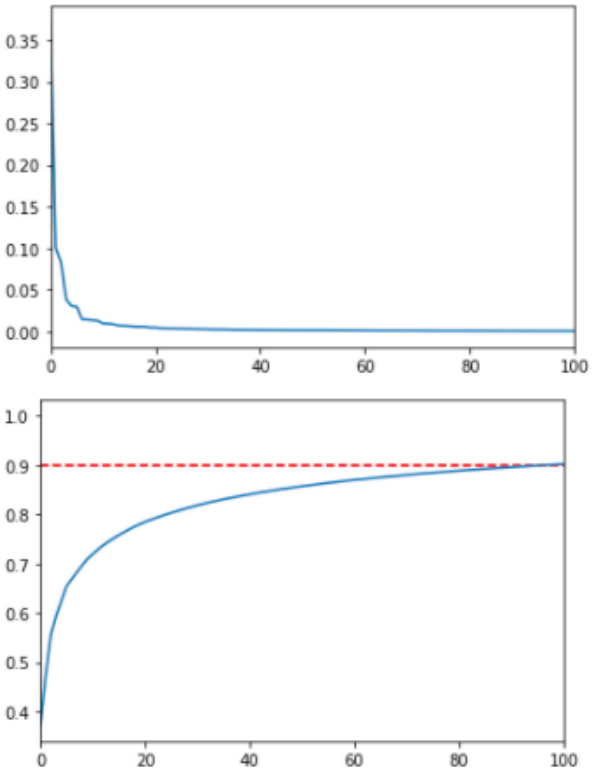
Graphique du nombre d'apparitions des mots les plus fréquents du dataset Rakuten :



5.2 Annexe 2.2 - Modélisation Machine Learning

5.2.1 Modèle Images - Détails de l'analyse PCA

Dans le cadre du projet nous avons entraîné des modèles afin de classifier les produits qu'avec ce type de données. Nous sommes partis des images d'entrée 500x500 pixels en RGB. Afin de faire une première réduction de variables, nous avons réduit les images à 50x50 pixels dans l'échelle de gris, ce qui fait 2500 variables avec des valeurs de 0 à 255 chaque.Ensuite nous avons appliqué l'algorithme PCA de scikit learn qui a réussi à réduire le nombre de variables à 99 tout en gardant 90% de la variabilité des donnés (sans perdre beaucoup d'information).



Ce graphique montre que la part de variance expliquée est majoritaire sur les premières composantes principales puis décroît très rapidement. On constate un phénomène d'épaulement sur ces graphes

5.2.2 Modèle Texte - Définitions des étapes de preprocessing NLP

La tokenisation consiste à segmenter des textes en unités plus petites. On peut segmenter un texte par paragraphe, par mots ou par lettre. Pour notre projet nous avons tokenisé le texte par mots car c'est à ce niveau que l'on trouve l'information nécessaire pour la classification. Pour la tokenisation nous avons utilisé les deux bibliothèques suivantes Natural Language Toolkit (NLTK) ou Spacy dans les différentes étapes du projet.

La partie stop words dans le text mining consiste à enlever les mots très communs, comme par exemple les articles, prépositions ou les signes de ponctuation entre autres. Comme ces mots sont très communs, ils n'apportent aucune information au modèle et souvent sont enlevés de l'analyse. Ces mots sont enlevés grâce aux librairies qui contiennent une liste de mots typiques mais aussi peut être fait par le propre utilisateur en fonction de son besoin.

La racinisation (steeming en anglais) et la lemmatisation. La racinisation permet de trouver les racines de mots enlevant les préfixes suffixes. La lemmatisation, qui est basé sur la recherche dans tableau lookup qui trouve le lemme ou l'origine du mot. Ces deux méthodes sont implémentés dans les deux libraires utilisés.

La fonction TfidfVectorizer permet aussi d'appliquer le tokeniser souhaité et d'appliquer les les stop words. La matrice np.array obtenue est ensuite passée en entrée d'un classificateur au travers d'un pipeline et le score obtenu est comparé à celui des autres classificateurs.

Pour l'entraînement nous avons divisé le data set en deux avec la fonction train_test_split du module scikit learn. Un échantillon de 80% pour l'entraînement et le 20% restant pour comparer les résultats de la labellisation des modèles avec les vrais labels et obtenir au même temps leur score.

Les premiers essais ont été effectués avec un jeu de données réduit, afin de raccourcir les temps d'exécution. Plusieurs classificateurs ont été testés afin d'identifier les plus prometteurs.

5.2.2.1 Choix meilleur modèle - Détail des Hyperparamètres

score	Classificateur	Hyperparamètres
0.7777	SVC	C=10 kernel=linear
0.7665	LinearSVC	dual=False penalty=l2
0.7578	LogisticRegression	class_weight=balanced max_iter=200
0.7442	SGDClassifier	penaly=l2
0.7378	MLPClassier	hidden_layer_sizes=10 max_iter=40
0.6344	GaussianNB	var_smoothing=1e-08
0.6032	KNeighborClassifier	metric=minkowski n_neighbors=4
0.5476	BernoulliNB	alpha=0.3
0.1702	RandomForest	criterion=entropy max_depth=5

Détail des essais supplémentaires ont été faits avec les classificateurs identifiés comme les meilleurs, en changeant les paramètres des classificateurs eux-mêmes mais aussi de tfidfVectorizer. Le tokenizer utilisé étant toujours celui de Spacy.

score	Durée(s)	Classifier	Hyperparamètres
0.7929	184	SVC	kernel=linear decision_function_shape=ovo
0.7863	10	LinearSVC	penalty=l1 dual=False
0.7792	1466	MLPClassifier	layer_sizes=40, max_iter=40
0.7708	109	LogisticRegression	class_weight=balanced max_iter=200
0.6459	4	KNeighborClassifier	n_neighbors=5 metric=minkowski

Quelques essais ont été faits avec des méthodes ensemble en prenant les meilleurs modèles mais le résultat n'est pas meilleur:

score	Durée(s)	Classifier	Hyperparamètres
0.7885	295	VotingClassifier	estimator=(knn, lr, svc) voting=hard weights=(1,2,2)
0.7835	308	VotingClassifier	estimator=(knn, lr, svc) voting=hard weights=None
0.7628	1630	StackingClassifer	estimator=(knn, lr, svc) final_estimator=LogisticRegression

5.3 Annexe 2.3 - Modélisations Deep Learning

Deep Learning - Détails des meilleurs modèles Image

5.3.0.0.1 -> Description des couches

```
def get_model_body(self):
    self.layer_index = 0
    self.basemodel = tf.keras.applications.EfficientNetB1(
                            input_shape = self.input_shape,
                            include_top = False,
                            # drop_connect_rate=0.4,
                            weights = 'imagenet')
    inp = Input(shape=self.input_shape, name="input_" + self.name)
    x = self.basemodel(inp)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1024, activation="relu", name=self.layer_name("dense"))(x)
    x = tf.keras.layers.BatchNormalization(trainable = True,axis=1, name=self.layer_name("batchnorm"))(x)
    x = tf.keras.layers.Dropout(0.5, name=self.layer_name("dropout"))(x)
    x = tf.keras.layers.Dense(512, activation="relu", name=self.layer_name("dense"))(x)
    x = tf.keras.layers.BatchNormalization(trainable = True, axis=1, name=self.layer_name("batchnorm"))(x)
    return inp, x
```

La version finale modèle de classification des images est constitué de couches diverses parmi lesquelles :

- **la base de convolution d'*EfficientNetB1*, issu de l'apprentissage par transfert**, permettant au modèle d'apprendre les motifs locaux;
- **1 GlobalAveragePooling2D**, présentant l'avantage de réduire le surapprentissage en diminuant le nombre de paramètres;
- **3 Dense,** étudiant les motifs plus globaux et permettant au modèle de s'adapter à son objectif de classification;
- **2 BatchNormalization,** accordant la possibilité d'augmenter la rapidité d'exécution de l'entraînement et de limiter le surajustement;
- **1 *Dropout***, réduisant les risques de surajustement.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_EffNetB1 (InputLayer)  [(None, 240, 240, 3)]     0
_________________________________________________________________
efficientnetb1 (Functional)  (None, 8, 8, 1280)        6575239
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense_1_EffNetB1 (Dense)     (None, 1024)              1311744
_________________________________________________________________
batchnorm_2_EffNetB1 (BatchN (None, 1024)              4096
_________________________________________________________________
dropout_3_EffNetB1 (Dropout) (None, 1024)              0
_________________________________________________________________
dense_4_EffNetB1 (Dense)     (None, 512)               524800
_________________________________________________________________
batchnorm_5_EffNetB1 (BatchN (None, 512)               2048
_________________________________________________________________
dense_EffNetB1 (Dense)       (None, 27)                13851
=================================================================
Total params: 8,431,778
Trainable params: 1,853,467
Non-trainable params: 6,578,311
_________________________________________________________________
```

5.3.0.0.2 -> Choix de paramètres

Le choix des paramètres a reposé sur un arbitrage entre optimisation des résultats de classification et minimisation du surajustement. Cet objectif a principalement conduit à un focus sur la maximisation des métriques de *validation accuracy* (précision mesurée sur l'échantillon de validation) et de *f1-score weighted* avec une analyse visuelle et comparative des courbes d'ajustement de l'*accuracy* (précision mesurée sur l'échantillon d'entraînement) et de la *validation accuracy*. Le visuel recherché était l'optimisation de l'alignement/ la convergence de ces deux courbes.

Afin de parvenir à cela, plusieurs stratégies ont été déployées. Parmi celles-ci :

- L'inclusion de paramètre de réajustement du comportement du processus de modélisation à travers **l'utilisation des *callbacks* pour l'ajustement du nombre d'*époques,* et du taux d'apprentissage**
- L'inclusion et l'augmentation des taux de suppressions des **couches de *Dropout,* pour réduire le surajustement** et inversement pour ne pas qu'il y ait un trop faible ajustement ce qui contraindrait le modèle à ne pas suffisamment apprendre et donc diminuerait ses performances

- **La réduction du nombre de couches et nombre d'unités par couches pour réduire le surajustement avec notamment le recours au *GlobalAveragePooling2D* qui permet de réduire le nombre de paramètres d'apprentissage lors de l'entraînement**

- L'ajustement du nombre de couches non figées.

5.3.0.0.3 Outre la méthode d'optimisation adoptée, l'entraînement du modèle a été effectué sur des images redimensionnées de taille (240, 240) regroupées dans des lots d'entraînement de taille 32

Deep Learning - Détails des meilleurs modèles Texte

Modèle EmbedRNN (f1 score : 0,8054) :

-> Etape 1 de preprocessing des données texte est effectué via l'API Keras

```
def preprocess_X_train(self, off_start, off_end, input_file=None):
    X_train = get_X_text(input_file)[off_start:off_end]
    self.tokenizer.fit_on_texts(X_train)
    self.word2idx = self.tokenizer.word_index
    self.idx2word = self.tokenizer.index_word
    self.vocab_size = self.tokenizer.num_words

    X_train = self.tokenizer.texts_to_sequences(X_train)
    X_train = tf.keras.preprocessing.sequence.pad_sequences(X_train, maxlen=self.maxlen, padding='post')
    return X_train
```

-> Etape 2 ***: modèle simple de Deep Learning (RNN type GRU + Embedding) pour classification finale***

```
self.maxlen = 600
self.num_words = 70000
self.embedding_dim = 200
self.tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=self.num_words)
```

```
def get_model_body(self):
    self.layer_index = 0
    inp = tf.keras.layers.Input(shape=(self.maxlen,),name=self.layer_name("input"))
    x = tf.keras.layers.Embedding(self.num_words, self.embedding_dim,name=self.layer_name("embed"))(inp)
    x = tf.keras.layers.GRU(128, return_sequences=True,name=self.layer_name("gru"))(x)
    x = tf.keras.layers.Dropout(0.3, name=self.layer_name("dropout"))(x)
    x = tf.keras.layers.GlobalAveragePooling1D(name=self.layer_name("batchnorm"))(x)
    x = tf.keras.layers.Dense(256, activation='relu', name=self.layer_name("dense"))(x)
    x = tf.keras.layers.Dropout(0.3, name=self.layer_name("dropout"))(x)
    return inp, x
```

Modèle TextOneHot (f1 score : 0,8471). Les principales étapes de ce modèle sont commentées ci-après.

-> Etape 1 de nettoyage tokenisation, lemmatisation :

```
def get_X_text_spacy_lemma_lower(inputXfile=None):
    """
    Retourne une liste X de tokens créés par le tokenizer.lemma de Spacy
    et transformation en caractères minuscules
    """
    if inputXfile is None:
        inputXfile = os.path.join(OUTDIR, os.path.basename(X_TRAIN_CSV_FILE))
    f = re.sub(r"\.csv$", "_text_spacy_lemma_lower.pkl",
               os.path.basename(inputXfile))
    xtokenfile = os.path.join(OUTDIR, f)
    if os.path.isfile(xtokenfile):
        with open(xtokenfile, 'rb') as fd:
            return pickle.load(fd)
    X_raw = get_X_text_spacy_lemma(inputXfile)
    print(f"Mise en minuscule de {len(X_raw)} listes de tokens")
    X = []
    for sentence in tqdm.tqdm(X_raw):
        tokens = [x.lower() for x in sentence]
        X.append(tokens)
    d = os.path.dirname(xtokenfile)
    if not os.path.isdir(d):
        os.mkdir(d)
    pickle.dump(X, open(xtokenfile, 'wb'))
    return X
```

-> Etape 2 de vectorisation du texte dans un pipeline sklearn de preprocessing de type TfidfVectorizer | SelectFromModel(LinearSVC)

```
TfidfVectorizer(analyzer='word',
  #strip_accents='ascii',
  #stop_words=french_stop_words,
  #    tokenizer=tokenize_spacy, # les données sont déjà tokenizées
  preprocessor=' '.join,
  lowercase=False,
  stop_words=None,
  max_df=0.8,
  min_df=2,
  ngram_range=(1,2),
  use_idf=True,
  smooth_idf=True,
  sublinear_tf=False,
  binary=True,
   max_features=30000,  moins rentable que celui de SelectFromModel
  ),
  SelectFromModel(LinearSVC(penalty="l2", dual=True, C=0.8,
                            tol=1e-5, max_iter=4000),
                  max_features=self.maxfeatures))
```

Le méta-transformateur 'SelectFromModel' de la librairie SKLEARN est utilisé afin de sélectionner les caractéristiques en fonction des poids d'importance, et qui correspondent aux caractéristiques/features les plus pertinentes. Dans notre cas, l'estimateur ayant donné les meilleurs résultats est le Linear SVC avec les hyper paramètres suivants : paramètre de régularisation C=0,8 , penalité Ridge L2, max_iter réglé à 4000.

Le vectorizer TfidfVectorizer permet de convertir les données brutes en une matrice de fonctionnalité TF-IDF (de l'anglais term frequency-inverse document frequency).Cette méthode

statistique permet d'évaluer l'importance d'un terme contenu dans un document, relativement à un corpus de texte. Le poids augmente proportionnellement au nombre d'occurrences du mot dans le document. Il varie également en fonction de la fréquence du mot dans le corpus.Les paramètres de réglages retenus pour notre modèle ont été sélectionnés sur la base de multiples tests. **Les principaux paramètres à retenir sont** :

- le seuil '**max_df**' que nous avons fixé empiriquement à 0,8, permet d'ignorer les termes ayant une fréquence de document strictement supérieure à ce seul donné : nous considérons que les mots au dessus de ce seuil sont des mots 'vides' et donc sans intérêt pour le modèle.

- la valeur **'min_df'** que nous avons fixée empiriquement à 2, a pour but d'ignorer les termes qui ont une fréquence d'apparition unitaire dans le corpus, strictement inférieure à cette valeur, qui est également appelée seuil dans la littérature. En effet, nous considérons que de tels mots sont soit des erreurs ou des mots 'marginaux', et donc sans intérêt pour le modèle.

-le tuple '**ngram_range'** que nous avons fixé à (1,2). Par exemple, un ngram_range de (1, 1) signifie uniquement des unigrammes, (1, 2) signifie des unigrammes et des bigrammes, et (2, 2) signifie uniquement des bigrammes.

-> Etape 3 : modèle simple de Deep Learning (sans RNN) pour classification finale

```
inp = tf.keras.layers.Input(shape=self.input_shape, name=self.layer_name("input"))
x = inp
x = tf.keras.layers.Dense(90, activation='relu', name=self.layer_name("dense"))(x)
x = tf.keras.layers.Dropout(0.5, name=self.layer_name("dropout"))(x)
x = tf.keras.layers.BatchNormalization(name=self.layer_name("batchnorm"))(x)
```

5.4 Annexe 2.3 bis - Modèle final Concatenate

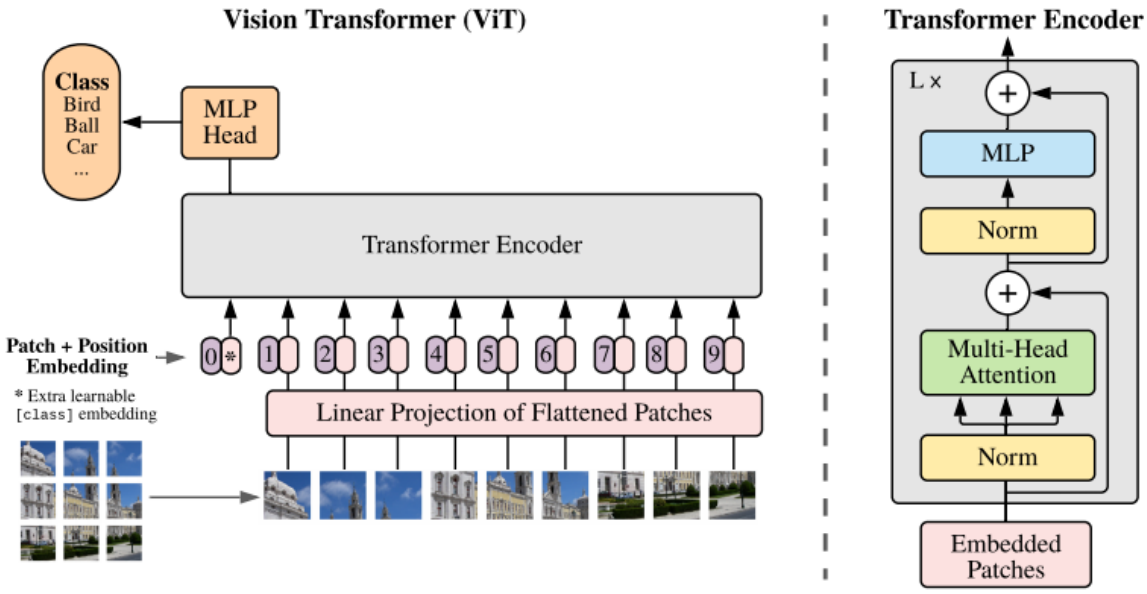
-> Détails du modèle Concatenate (f1 score : 0,8783) :

```
def fit(self, off_start, off_val, off_end, input_file=None):
    self.prt(f'Chargements des modèles référencés "{self.nb}"')
    self.config(self.nb)
    # Générateurs qui alimenteront fit() avec les données train & val
    traingen, valgen = self.create_train_generators(off_start, off_val, off_end, input_file)
            self.prt(f"Création du modèle")
    # Récupération des inputs et des outputs de chaque modèle de base pour construire le modèle concatenate
    inputtensors, outputtensors = [], []
    for obj in self.objs:
        inp, outp = obj.get_model_body()
        inputtensors.append(inp)
        outputtensors.append(outp)
            self.layer_index = 0 # compteur utilisé dans get_model_body()

    tensors = outputtensors
    x = concatenate(outputtensors, axis=-1, name = self.layer_name("concatenate"))
    x = Dense(NB_CLASSES, activation='softmax', name=self.layer_name("dense"))(x)
    self.model = Model(inputtensors, x, name=self.name)

    print(f"  * Layers non entrainables:")
    for layer in self.model.layers:
        if layer.name.find(self.name) < 0:
            layer.trainable = False
            print(f"      - {layer.name}")
    print(f"  * Layers entrainables:")
    for layer in self.model.layers:
        if layer.trainable:
            print(f"      - {layer.name}")

    self.model.compile(optimizer= tf.keras.optimizers.Adam(lr=0.001), loss='sparse_categorical_crossentropy',
                       metrics = ['accuracy'])
    # Initialisation des poids
    self.copy_submodels_weights()
    self.prt("fit(): Début")
    callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True, verbose=1)
                 tf.keras.callbacks.ModelCheckpoint(filepath=self.fbestweights, save_weights_only=True, save_best_only=True,
                 monitor='val_loss', mode='min'),
                 tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', patience=3, factor=0.1,verbose=1)]

    if os.path.isfile(self.fbestweights):
        os.remove(self.fbestweights)
    history = self.model.fit(traingen,epochs=15, validation_data = valgen, callbacks=callbacks)
    if os.path.isfile(self.fbestweights):
        self.model.load_weights(self.fbestweights)
    self.prt("fit(): Fin")
    plot_history(f"{self.name}", history)

    print(f"Décongélation des layers suivants:")
    for layer in self.model.layers:
        if not layer.trainable and layer.name.find("efficient") < 0:
            print(f"      - {layer.name}")

    self.model.compile(optimizer= tf.keras.optimizers.Adam(lr=0.0001), loss='sparse_categorical_crossentropy', metrics = ['accuracy'])
    if os.path.isfile(self.fbestweights):
        os.remove(self.fbestweights)
    history = self.model.fit(traingen, epochs=15,  validation_data = valgen, callbacks=callbacks)
    if os.path.isfile(self.fbestweights):
        self.model.load_weights(self.fbestweights)
    self.prt("fit(): Fin")
    plot_history(f"{self.name}", history)

    return history
```

5.5 Annexe 2.3 ter - alternative viT aux modèles CNN pour Computer Vision (généralités)

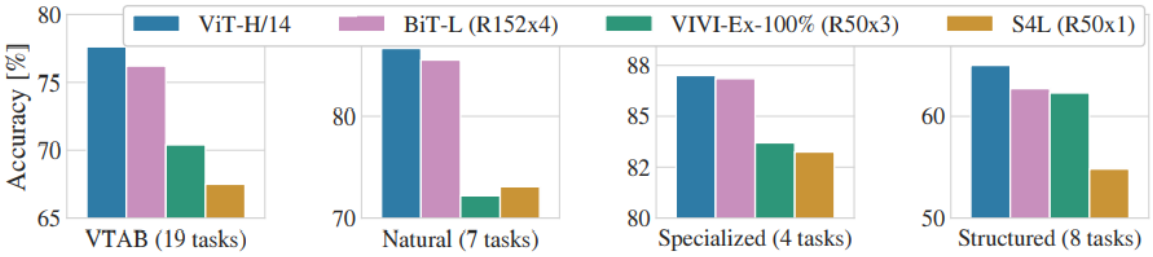
En computer vision, les modèles base CNN commencent à être mis en concurrence et/ou couplés avec des transformers viT traditionnellement utilisés en NLP, par exemple le modèle BERT. Le principe des transformers viT est le suivant : une image est divisée en patchs pour fournir la séquence d'incorporation linéaire de ces patchs en entrée d'un Transformer. Les patchs d'image sont traités de la même manière que les token (mots) dans une application NLP. Nous entraînons le modèle de classification d'images de façon supervisée.

Appliquées à de très grands jeux de données, les performances obtenues par les vision Transformers (viT) sont comparables aux modèles benchmark CNN. Ci dessous une illustration : division d'une image en patchs de taille fixe, intégration linéaire de chacun des patchs, ajout des embeddings de position et alimentation de la séquence de vecteurs résultante dans un transformateur standard encodeur. Afin d'effectuer la classification, nous utilisons l'approche standard consistant à ajouter un apprentissage supplémentaire
« classification token » à la séquence :



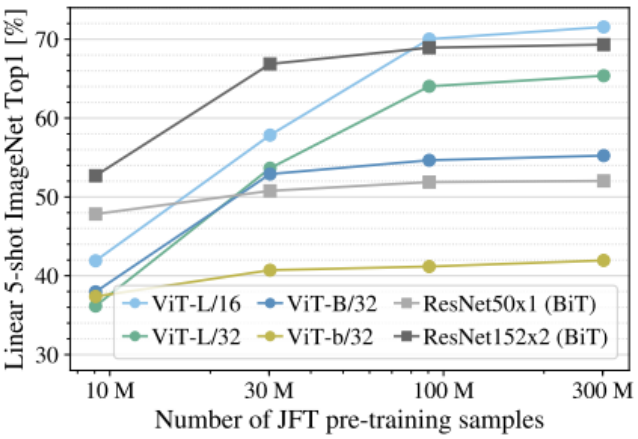
A titre informatif : Un autre modèle connexe récent est l'image GPT (iGPT) (Chen et al., 2020a), qui applique les transformateurs aux pixels de l'image après avoir réduit la résolution de l'image et l'espace colorimétrique. Le modèle est entraîné de manière non supervisée en tant que modèle génératif, et la représentation résultante peut ensuite être affinée linéairement pour les performances de classification, atteignant une précision maximale de 72% sur ImageNet.

Dans l'étude dirigée par Alexey Dosovitskiy (Google Research, Brain Team), les modèles de transformateur pré-entraînés sur l'ensemble de données JFT-300M surpassent les lignes de base basées sur ResNet sur tous ensembles de données, tout en prenant beaucoup moins de ressources de calcul pour le pré-entraînement:



Dans cette même étude, il est observé que la distance d'attention (équivalente aux champs récepteur des CNN) augmente avec la profondeur du réseau. Globalement, il est constaté que les modèles ViT s'occupent des régions de l'image "sémantiquement" pertinentes pour la classification.

Un point capital pour la réussite est le pré entraînement sur des volumes de données maximal type JFT300M : il s'agit d'un ensemble de données interne de Google utilisé pour l'entraînement des modèles de classification d'images. Les images sont étiquetées à l'aide d'un algorithme qui utilise un mélange complexe de signaux Web bruts, de connexions entre les pages Web et de commentaires des utilisateurs. Cela se traduit par plus d'un milliard d'étiquettes pour les 300 millions d'images (une seule image peut avoir plusieurs étiquettes). Sur le milliard d'étiquettes d'images, environ 375 millions sont sélectionnées via un algorithme qui vise à maximiser la précision des étiquettes des images sélectionnées.
(https://paperswithcode.com/dataset/jft-300m)



6 Bibliographie

Francois Chollet - Deep Learning with Python

Francois Chollet - Xception: Deep Learning with Depthwise Separable Convolutions

Aurélien Géron - Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow

Steven Bird, Ewan Klein, Edward Loper - Natural Language Preprocessing with Python

Jan Erik Solem - Programming Computer Vision with Python

Alexey Dosovitskiy - AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Chen Du - Selective Feature Connection Mechanism: Concatenating Multi-layer CNN Features with a Feature Selector

Mingxing Tan - EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Kaiming He - Deep Residual Learning for Image Recognition

https://challengedata.ens.fr/

https://www.fevad.com/barometre-trimestriel-de-laudience-du-e-commerce-en-france-enquete-e-commerce-et-confinement/

https://www.image-net.org/

https://paperswithcode.com/dataset/jft-300m

https://keras.io/api/applications/efficientnet/

https://keras.io/api/layers/merging_layers/concatenate/

https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html