DataScientest • com

Rakuten

Rapport Technique d'Évaluation

Projet : eShopPye

Equipe :

Nada STAOUITE
Bastien PIQUEREAU
Lucas GANDY

Mentor :

Chloé GUIGA

Promotion :

Bootcamp Décembre 2020

1 TABLE DES MATIERES

2 CONTEXTE

2.1 PLACE DANS L'ACTUALITE

Il s'agit d'un projet réalisé dans le cadre du challenge Rakuten France Multimodal Product Data Classification, organisé par *Rakuten Institute of Technology Paris*.

Du point de vue technique, le projet était ambitieux au vu du temps disponible. Il fait appel aux technologies du *Deep Learning*, aussi bien pour le *Natural Language Processing (NLP*) que de *Computer Vision (CV)* : réseaux de neurones artificiels récurrents (RNN) et convolutifs (CNN).

Du point de vue économique, c'est un sujet très actuel, au vu du développement en perpétuelle croissance du e-commerce, et qui, en 2021 d'où nous écrivons ces lignes, a certainement encore de beaux jours devant lui en cette période crise sanitaire dus au COVID-19 et ses variants.

Du point de vue scientifique, ce projet trouve sa place parmi les recherches actuelles sur les problèmes de classification multi-classe (ou multi-label) à grande échelle, et de classification bimodale texte et images.

2.2 PLACE DANS LA FORMATION

Lucas et Bastien étaient tous deux novices en Data Science au début du projet.

Nada en revanche, issue d'un master en intelligence artificielle, avait déjà des connaissances théoriques en Deep Learning et quelques applications pratiques avec le module Keras, sans utiliser toutes les fonctionnalités de Tensorflow.

Ce projet était sans aucun doute un levier puissant de montée en compétence, comme cela a été confirmé par la suite.

3 OBJECTIFS

Le problème consiste, à partir des informations textuelles et de l'image associées à chaque article du catalogue, à classifier automatiquement chacun d'eux dans l'une des catégories issues de la taxonomie produit de Rakuten, en commettant le moins d'erreurs possible.

En situation de production, la taxonomie comprend plus de 1000 catégories, mais dans le cadre de ce challenge, on se limite à seulement 27 d'entre elles.

4 DONNEES ET CADRE

4.1 DESCRIPTION DES DONNEES

Les données utilisées sont la propriété de *Rakuten Institute of Technology* et sont fournies lors de l'inscription au challenge. Elles présentent les caractéristiques suivantes :

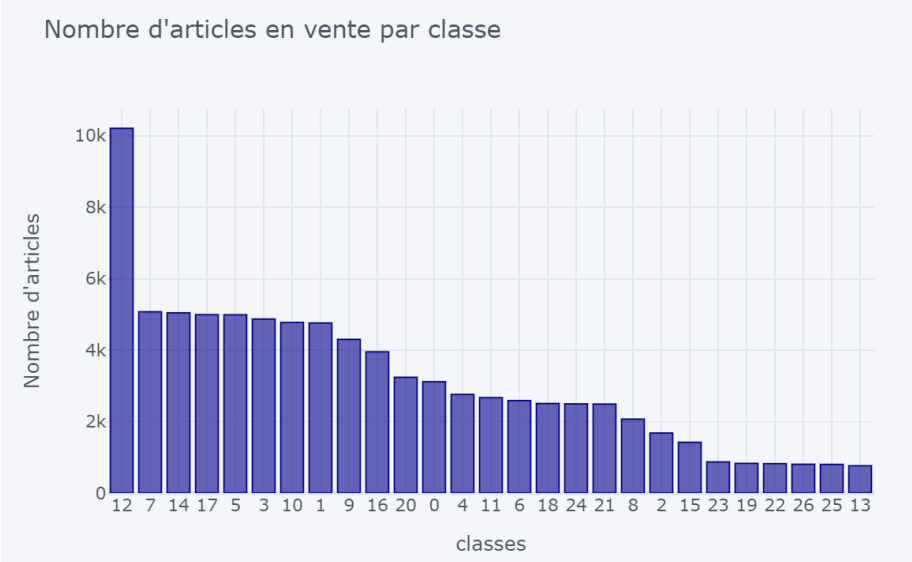
- Le jeu de données disponible pour l'entraînement (et la validation) est constitué de 84916 observations. Censément, chaque observation représente un article du catalogue, mis en vente sur la plateforme de e-commerce de Rakuten France,
- Chaque observation comporte nécessairement une image et une désignation textuelle, et éventuellement une description textuelle plus détaillée,
- L'ensemble des deux champs de texte contiennent en moyenne 11 mots par observation, encodés au format Html et leur volume total est d'environ 50 Mo,
- Les images associées aux articles sont en couleur, redimensionnées sur 500x500 pixels, et encodées au format JPG. Leur volume total est d'environ 2,5 Go.

4.2 DESCRIPTION DES VARIABLES

Au cours de l'exploration des données, nous avons établi que :

- Le jeu de données ne présente aucun doublon,
- Chaque article est identifié par un code "productid" unique,
- Tous les articles ont un code "imageid" unique, faisant référence à l'image associée,
- Toutes les images JPG fournies sont valides et correctement identifiées,
- Tous les articles ont un champ "designation" renseigné,
- En revanche, tous les articles n'ont pas leur champ "description" renseigné. Le pourcentage de descriptions manquantes est d'environ 35.09 %.

La distribution des classes est déséquilibrée comme en témoigne le graphique suivant.



4.3 SELECTION DES VARIABLES

Tout d'abord, disons qu'il était impératif de traiter à la fois le texte et les images.

Cependant, l'information textuelle étant répartie entre les deux champs "designation" et "description", et ce dernier ayant des valeurs manquantes, il fallait décider de la marche à suivre :

1. N'utiliser que le champ "designation"
2. Fusionner les champs "designation" et "description"
3. Utiliser une méthode d'imputation pour combler les champs "description" manquants

Après l'analyse visuelle des données et l'étude des unigrammes/bigrammes, nous avons constaté que la colonne description contenait beaucoup de bruits et de données inutiles qui sont réparties sur toutes les classes. Nous avons décidé d'utiliser uniquement la colonne champ "designation". Cela nous permet de ne pas introduire de biais ni de bruit dans les données, et de réduire le temps de calcul, compte tenu du volume important des données.

4.4 VARIABLE CIBLE

Nous avons encodé le champ "prdtypecode" pour obtenir un champ "target" contenant des numéros de classe pour labels, pour faciliter le traitement. La table suivante donne la correspondance entre les deux champs.

prdtypecode	10	2280	50	1280	2705	2522	2582	1560	1281	1920	2403	1140	2583	1180
target	0	1	2	3	4	5	6	7	8	9	10	11	12	13
prdtypecode	1300	2462	1160	2060	40	60	1320	1302	2220	2905	2585	1940	1301	
target	14	15	16	17	18	19	20	21	22	23	24	25	26	

4.5 PRETRAITEMENT DONNEES TEXTES

4.5.1 Nettoyage

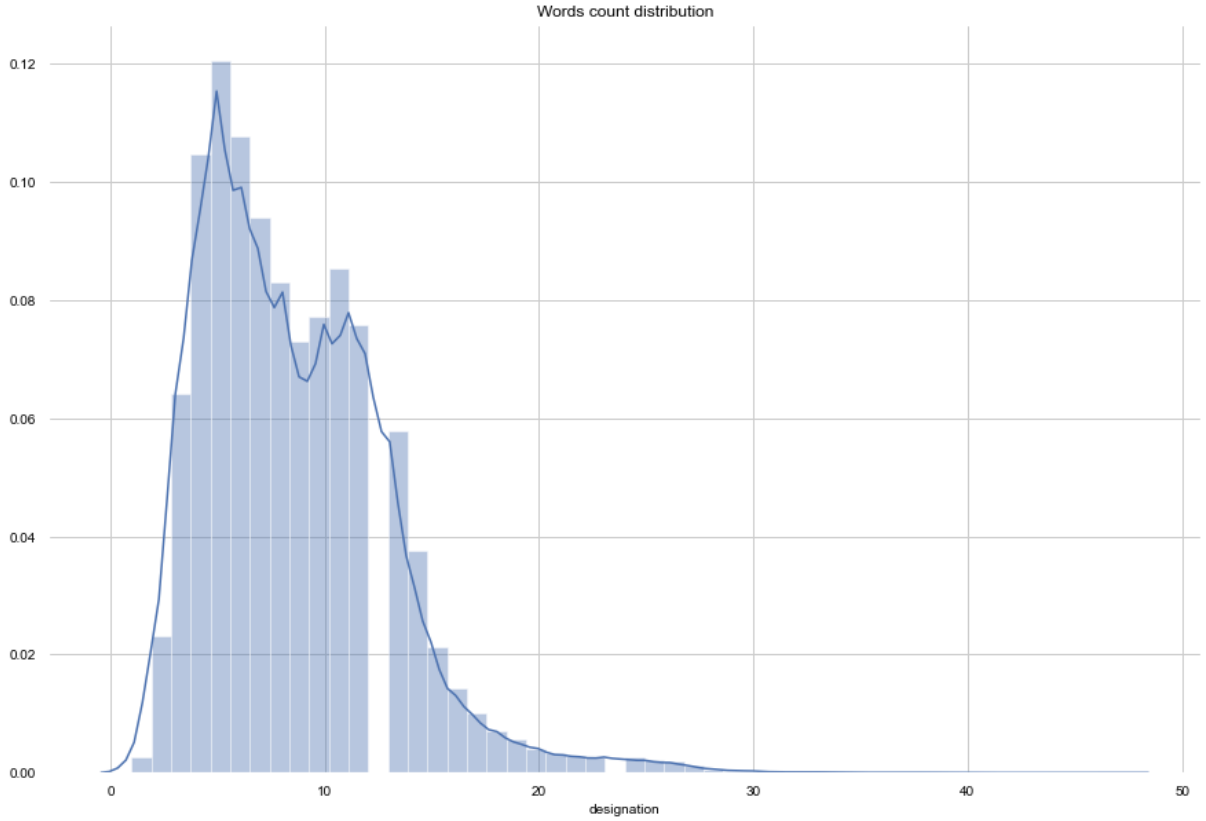
Par suite des étapes data visualisation et data analyse, nous avons pu constater différents prétraitements à réaliser sur les données :

- Remplacement des lettres avec des accents encodés en HTML comme "&eacute"
- Suppression des balises HTML
- Suppression de la ponctuation
- Tous les mots en lettres minuscules
- Suppressions des stopwords classiques des mots en français, mais également des mots présents en grandes quantités dans toutes les classes.

4.5.2 Représentation vectorielle

Nous avons ensuite séparé le texte en mots grâce à la classe Tokenizer de « tf.keras.preprocessing.text » avec un nombre maximum de mots 46000 (nombre de mots total de 46010).

Nous avons ensuite défini la longueur maximum d'une séquence en nous basant sur la distribution du nombre de mots par ligne. Comme 96% des séquence ont une longueur de 17 mots, nous avons ainsi fixé la taille du padding à 17.

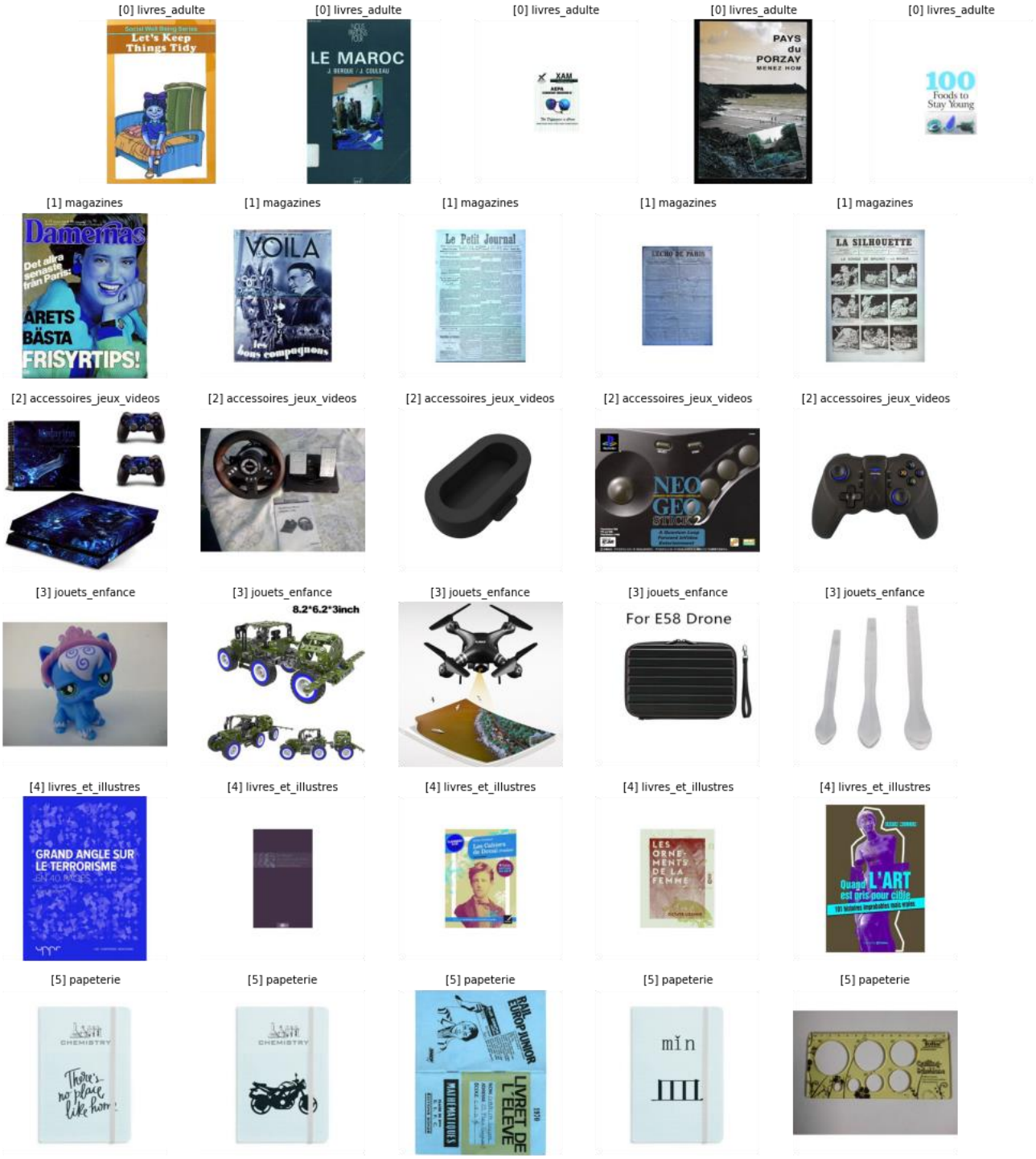


4.6 EXPLORATION ET ANALYSE DES IMAGES

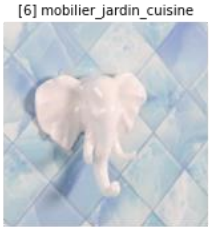
4.6.1 Affichage d'échantillons

Afin d'avoir un aperçu des catégories de produits, nous avons affiché 5 images pour chacune de ces classes.

Les étiquettes associées ici aux images sont constituées du label de la classe et du type de produit (champ "prdtype") que nous avons créé lors de l'exploration.



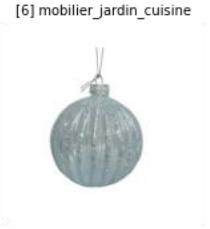
[6] mobilier_jardin_cuisine



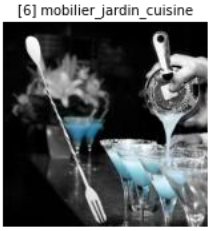
[6] mobilier_jardin_cuisine

[6] mobilier_jardin_cuisine

[6] mobilier_jardin_cuisine



[6] mobilier_jardin_cuisine



[7] mobilier_interieur_litterie

[7] mobilier_interieur_litterie

[7] mobilier_interieur_litterie

[7] mobilier_interieur_litterie

[7] mobilier_interieur_litterie

[8] jeux_societe

[8] jeux_societe

[8] jeux_societe

[8] jeux_societe

[8] jeux_societe

[9] accessoires_interieur

[9] accessoires_interieur

[9] accessoires_interieur

[9] accessoires_interieur

[9] accessoires_interieur

[10] livres_jeunesse

[10] livres_jeunesse

[10] livres_jeunesse

[10] livres_jeunesse

[10] livres_jeunesse

[11] goodies_geek

[11] goodies_geek

[11] goodies_geek

[11] goodies_geek

[11] goodies_geek

[12] piscine_spa

[12] piscine_spa

[12] piscine_spa

[12] piscine_spa

[12] piscine_spa

[13] figurines_wargames

[13] figurines_wargames

[13] figurines_wargames

[13] figurines_wargames

[13] figurines_wargames

[14] modeles_reduits_ou_telecommandes

[14] modeles_reduits_ou_telecommandes

[14] modeles_reduits_ou_telecommandes

[14] modeles_reduits_ou_telecommandes

[14] modeles_reduits_ou_telecommandes

[15] jeux_geek

[15] jeux_geek

[15] jeux_geek

[15] jeux_geek

[15] jeux_geek

[16] cartes_a_jouer

[16] cartes_a_jouer

[16] cartes_a_jouer

[16] cartes_a_jouer

[16] cartes_a_jouer

[17] decoration_interieur

[17] decoration_interieur

[17] decoration_interieur

[17] decoration_interieur

[17] decoration_interieur

[18] jeux_videos_import

[18] jeux_videos_import

[18] jeux_videos_import

[18] jeux_videos_import

[18] jeux_videos_import

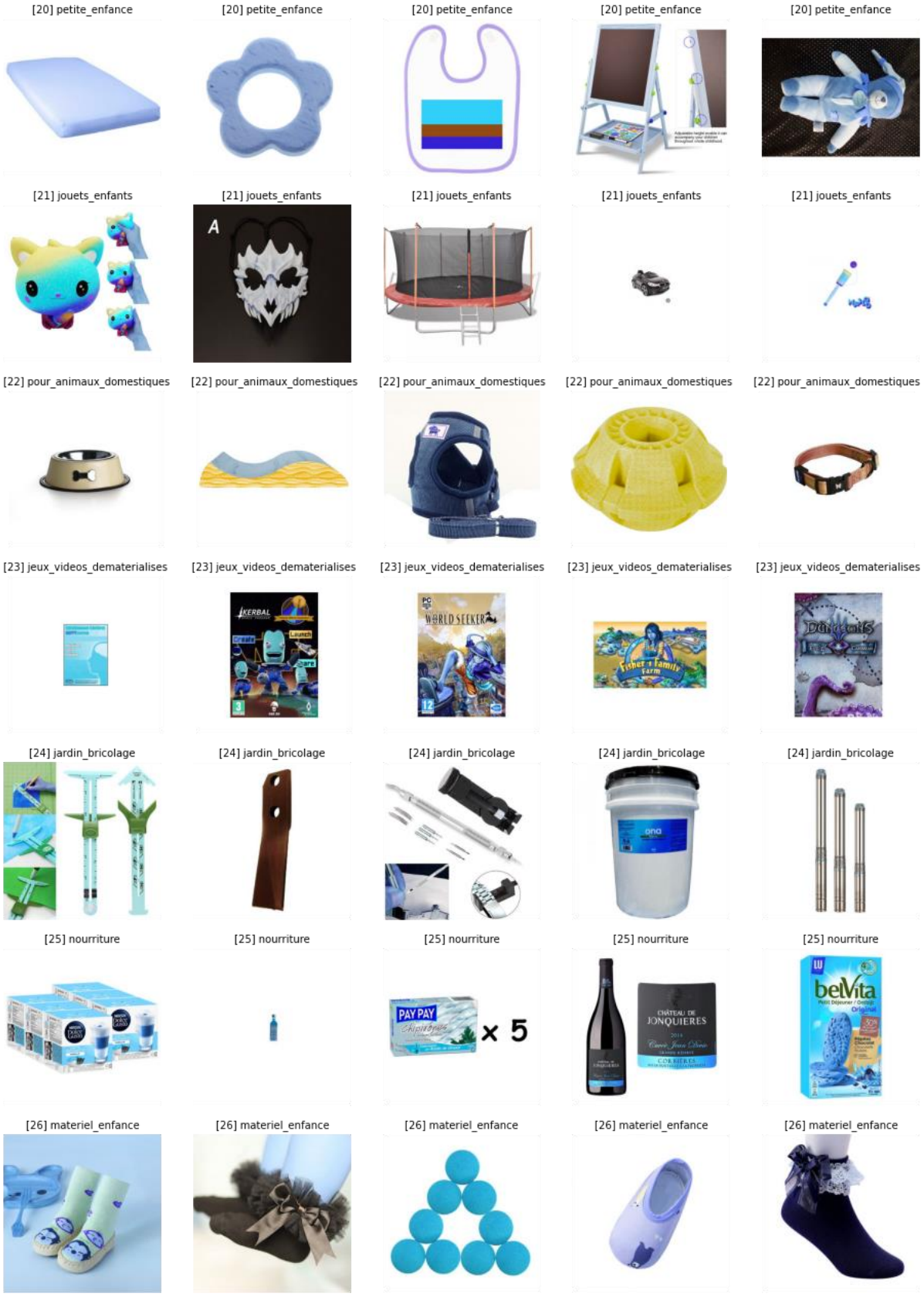
[19] jeux_et_consoles_retro

[19] jeux_et_consoles_retro

[19] jeux_et_consoles_retro

[19] jeux_et_consoles_retro

[19] jeux_et_consoles_retro



Bien que cet échantillon soit de taille très modeste, il permet déjà d'illustrer les faits suivants.

Premièrement, certaines classes comportent des images similaires (e.g. classes 0, 1, 4, 16, 19, 23), Manifestement, ces classes concernent un type de produit très spécifique et au packaging standardisé (livres, magazines, cartes à jouer, jeux vidéo, etc.), et sont relativement minoritaires.

Deuxièmement, pour la majorité des classes, on observe en revanche une forte dispersion intra-classe des images (notamment classes 3, 7, 17, 20, 21, 24). Ceci peut s'expliquer en partie par le fait que certaines de ces classes ont vocation à regrouper des objets assez divers (notamment bricolage, jardinage, etc.). Cela illustre en tous cas la difficulté à extraire des caractéristiques classifiantes des images d'une classe.

Troisièmement, bien que certaines classes se ressemblent (e.g. ensembles de classes {0, 1, 4} et {18, 23}), dans la majorité des cas, la dispersion inter-classe des images est forte, ce qui devrait faciliter la classification.

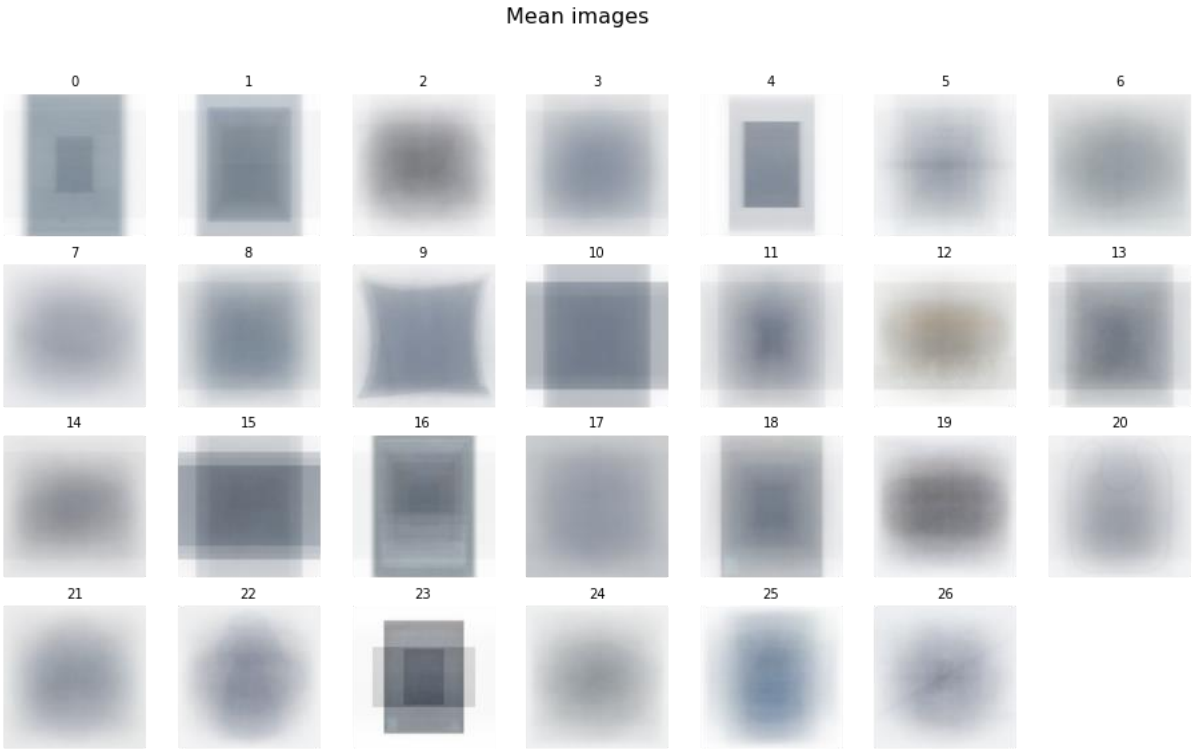
4.6.2 Dispersion des images

Pour visualiser les dispersions inter-classe et intra-classe, il est courant et généralement pertinent de calculer l'image moyenne, et éventuellement l'image variance pour chaque classe.

4.6.2.1 Images moyenne

Pour obtenir l'image moyenne, on calcule, pour chaque pixel et pour chaque canal de couleur, la moyenne des valeurs sur l'ensemble des observations appartenant à une classe donnée. On obtient donc une image en couleur, de mêmes dimensions que les autres.

Les étiquettes associées ici aux images sont les labels des classes.



On peut remarquer plusieurs caractéristiques notables.

Premièrement, considérons la luminance des images moyennes. Pour simplifier, on peut dire que la luminance d'un pixel représente le niveau de clarté (perçu par l'oeil humain) de ce pixel sur une échelle contenant toutes les nuances de gris du blanc au noir.

Pour quelques classes, la luminance s'approche de celle d'un bruit gaussien 2D centré (e.g. classes 6, 7, 21, 24), ce qui révèle une très grande dispersion intra-classe.

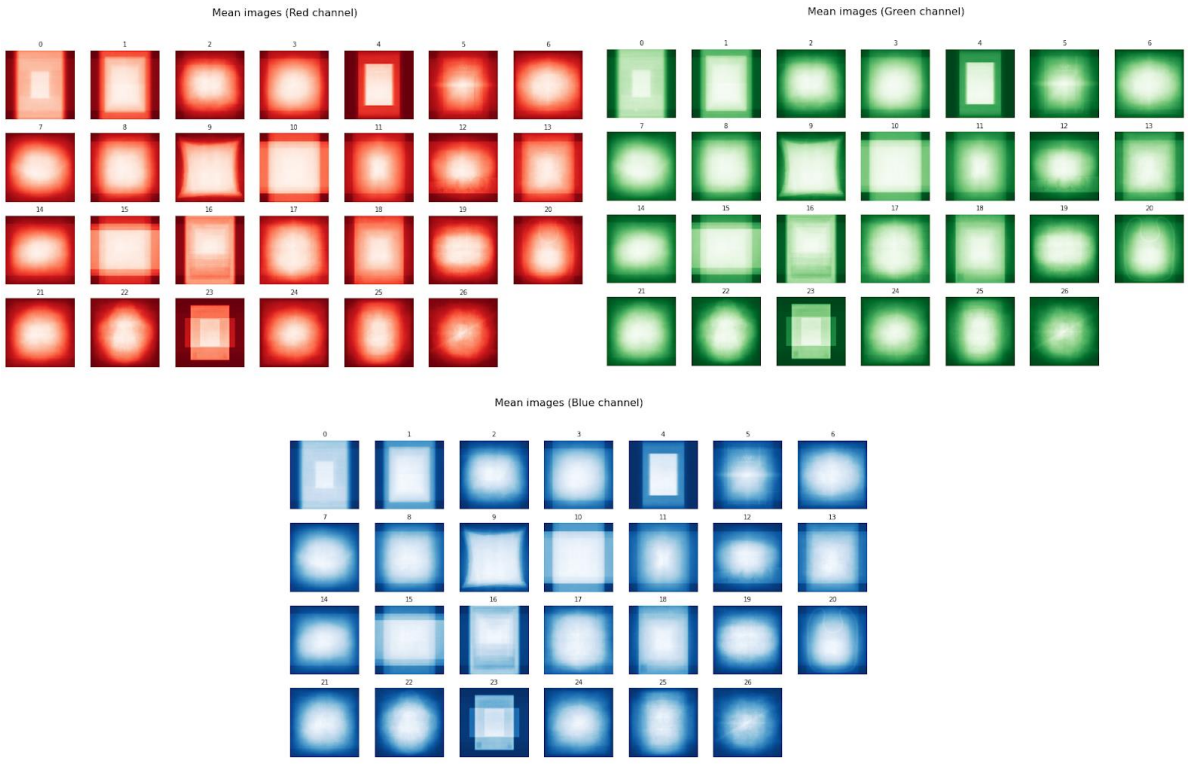
Pour d'autres classes, en revanche, les images moyenne tendent à adopter des géométries rectangulaires (e.g classes 0, 1, 4, 10, 13, 15, 16, 23). On pourrait être tenté d'en conclure que les objets des classes concernées sont majoritairement de forme rectangulaire, et placés soit en portrait soit en paysage sur les images. C'est très certainement vrai pour certaines classes (e.g. [0]livres, [1]magazines, [16] cartes à jouer, etc.), cependant il faut rester prudent, car les images ont été redimensionnées sur 500x500 pixels, et cela a pu créer artificiellement des discontinuités (e.g. cadres ou bandeaux blancs) sur les bords.

Deuxièmement, considérons les teintes de couleurs des images.

Pour chaque classe, à quelques exceptions près (e.g. classes 12 et 25), la teinte dominante sur l'ensemble des pixels de l'image est toujours dans des tons gris-brun, ce qui indique que la répartition globale des trois couleurs principales {Rouge, Vert, Bleu} est sensiblement la même pour chaque classe.

De plus, chaque classe, la distribution de la teinte en fonction des pixels de l'image moyenne semble relativement homogène, ce qui semble indiquer une forte dispersion spatiale de chaque couleur primaire au sein des images d'une même classe.

L'observation des images moyenne pour chaque canal de couleur pris séparément permet de confirmer les points précédents.



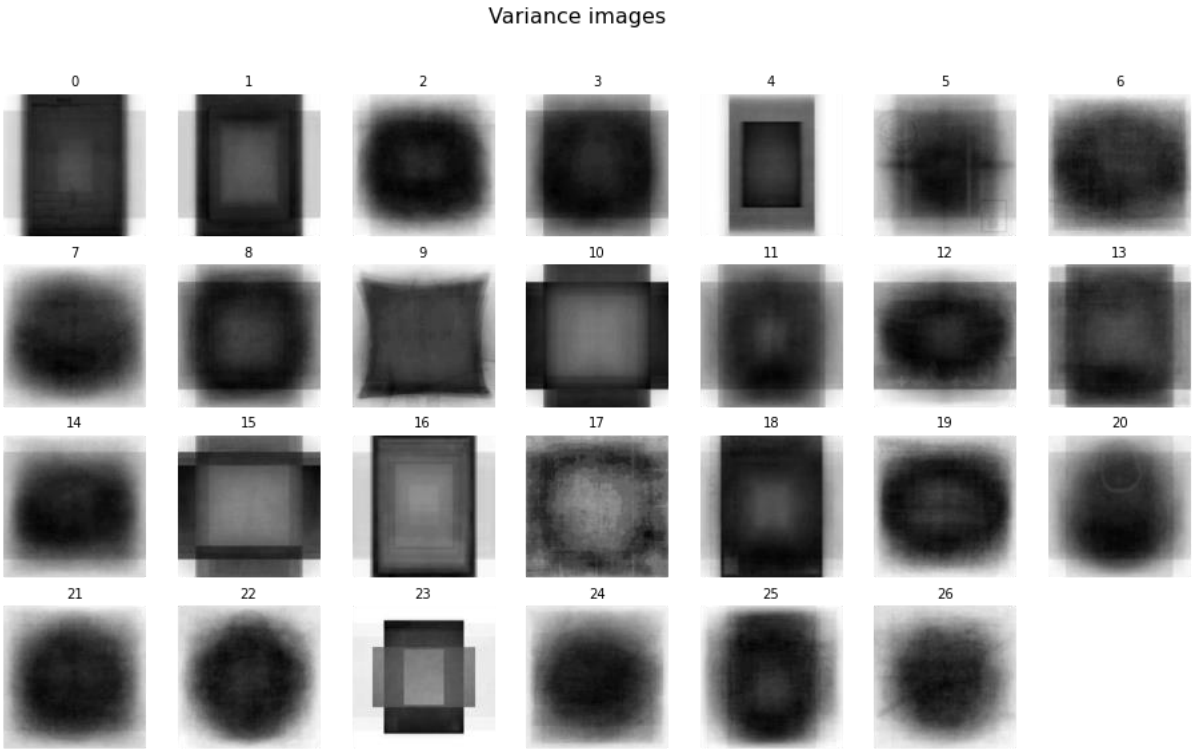
On n'observe pas de différence sensible entre les canaux de couleur. La teinte de couleur de l'image ne semble donc pas être un critère de classification pertinent.

4.6.2.2 Images variance

Pour mieux faire apparaître les contrastes, nous avons converti les images en noir et blanc (*angl.: grayscale*) en calculant simplement la moyenne sur les trois canaux de couleur. Notons que, rétrospectivement, il nous apparaît qu'il aurait été plus juste d'utiliser la technique de conversion pondérée dite colorimétrique.

Pour obtenir l'image variance, on calcule, chaque pixel, la variance des valeurs sur l'ensemble des observations appartenant à une classe donnée. On obtient donc une image en noir et blanc, de mêmes dimensions que les autres.

Les étiquettes associées ici aux images sont les labels des classes. Avec la *colormap* utilisée (*'binary'* de *matplotlib*), plus on se rapproche du noir, plus la variance est élevée et plus on se rapproche du blanc, plus la variance est faible.



Ce que l'on observe est similaire à ce que l'on voyait pour la moyenne, mais avec un contraste supérieur. Par exemple pour la classe 16 ('cartes_a_jouer'), on voit mieux les multiples différences de zoom entre les photographies des cartes. Pour la classe 17 ('decoration_interieur'), on observe de la granularité au centre de l'image, ce qui montre que les objets photographiés ont des formes et dimensions très différentes.

De manière générale, on observe assez peu de variance sur les bords des images, ce qui est assez naturel, car les objets sont le plus souvent au centre des images, et de plus les images ont été reformatées.

4.6.3 Prétraitement des données images

Comme il est fréquent de le faire dans ce contexte, nous avons choisi d'utiliser la classe *ImageDataGenerator* (du module *tensorflow.keras.preprocessing.image*). Celle-ci présente plusieurs fonctionnalités très intéressantes :

- Permet de réaliser de l'augmentation de données (*angl.: data augmentation*). En appliquant aux images d'entraînement des transformations affines d'amplitude modérée, on aide le modèle à apprendre des caractéristiques qui se généralisent mieux, et on diminue ainsi le risque de surapprentissage (*angl. overfitting*).
- Permet, par l'intermédiaire de plusieurs méthodes, de ne pas avoir à charger l'ensemble des images dans la mémoire vive pendant l'entraînement et l'évaluation, mais de les transmettre sous forme de flux (*angl.: stream*) et par lots (*angl.: batches*).
- Permet de redimensionner les images et de leur appliquer une fonction de prétraitement (*angl.: preprocessing*) à la volée.

Afin d'avoir accès aux labels de la variable cible de notre *DataFrame* (classe du module *Pandas*), nous avons préféré utiliser la méthode *flow_from_dataframe()* de la classe *ImageDataGenerator*. Celle-ci va chercher les chemins des images et les labels dans des colonnes à spécifier, et renvoie un objet *DataFrameIterator* qui fournit, entre autres, les images transformées selon les paramètres spécifiés lors de la création de l'objet *ImageDataGenerator* appelant.

Nos images sont redimensionnées en 256x256 pixels (format par défaut proposé), taille présentant un bon compromis entre fidélité et complexité dans l'état de l'art.

La fonction de prétraitement est la fonction *preprocess_input()* issue du module correspondant au modèle sélectionné (e.g. *resnet, vgg16, xception* du package *tensorflow.keras.applications*). L'effet de cette fonction varie selon le modèle. Pour *resnet* et *vgg16*, les canaux de couleurs sont inversés (RGB vers BGR), puis chaque canal est centré par rapport à la moyenne du jeu de données ImageNet, tandis que pour *xception*, il s'agit d'une normalisation (*angl.: scaling*) des valeurs des pixels dans l'intervalle [-1;1] indépendante pour chaque image.

5 PROJET

5.1 DESCRIPTION DU PROBLEME

Il s'agit d'un problème de classification taxonomique à large échelle (*angl.: Large Scale Taxonomy Classification*), qui est un cas particulier de la classification multi-classes fréquemment rencontré dans le domaine de e-commerce.

5.2 CHOIX DES METRIQUES

Dans un problème de classification multi-classes tel que celui-ci, il est important de surveiller les *precision* et *recall* de chaque classe.

Il est cependant plus pratique, et usuel de se contenter d'une part du seul *F1-score* (moyenne harmonique de *precision* et *recall*), et d'autre part d'agréger celui-ci par la méthode *macro* (moyenne non pondérée sur les classes) ou *weighted* (moyenne sur les classes, pondérée par leurs cardinaux).

Dans le cadre du challenge, la méthode *weighted* était préconisée au départ, mais les leaderboards montrent le score *macro.* Nous avons donc jugé utile de noter les deux, d'autant que la première méthode est pessimiste et la seconde optimiste.

Cela étant dit le jeu de données n'est pas si déséquilibré que la métrique *accuracy* en perde toute pertinence, nous l'avons donc également relevée.

5.3 CLASSIFICATION A PARTIR DES IMAGES

5.3.1 Type d'architecture

Lors de la phase d'exploration des données, nous avons pu constater que nos images sont complexes et très diverses. Nous en avons déduit que des méthodes traditionnelles d'apprentissage machine (*angl.: machine learning*) ne nous permettraient pas d'obtenir de très bons résultats, et qu'il était préférable de s'orienter vers des modèles basés sur les réseaux de neurones artificiels (*angl.: artificial neural networks*) capables d'apprentissage profond (*angl.: deep learning*).

Les travaux de ces dernières années ont montré que les réseaux de neurones les plus adaptés au traitement d'images sont les réseaux de neurones convolutifs (*angl.: Convolutional Neural Networks* ; abrév.: *CNN*).

5.3.2 Apprentissage par transfert

Cependant, l'entraînement des CNN modernes est coûteux en temps et en puissance de calcul, c'est pourquoi nous avons réalisé un apprentissage par transfert (*angl.: transfer*

learning). Le principe de cette méthode est d'extraire les couches convolutives d'un modèle CNN pré-entraîné (*angl.: backbone*) et ayant fait ses preuves le jeu de données ImageNet (qui est un incontournable de l'état de l'art), et d'y ajouter en aval nos propres couches neuronales denses (*angl.: dense layers / fully-connected layers*).

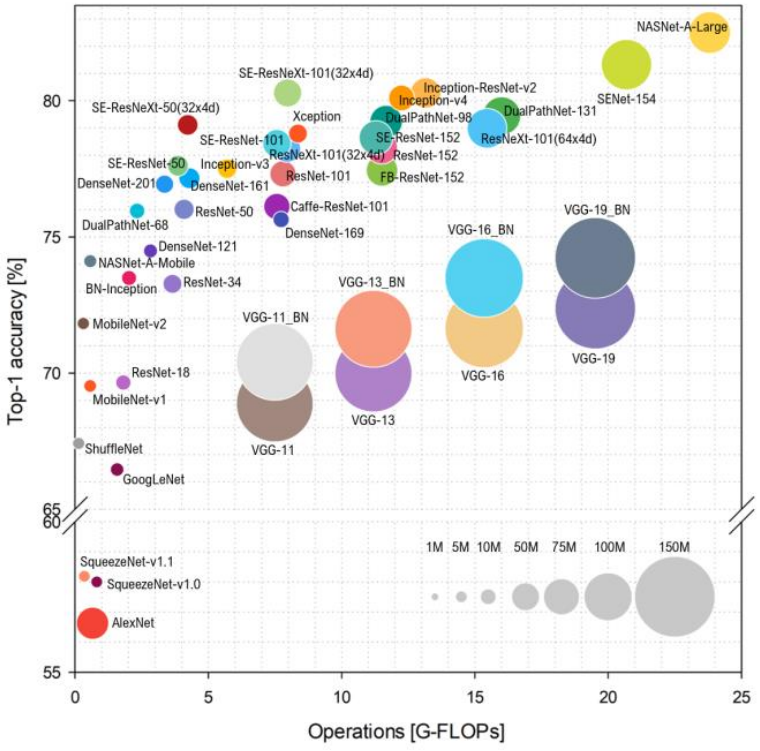
Pour profiter au maximum du pré-entraînement, il est important de geler (*angl.: freeze*) les poids associés aux blocs convolutifs, qui serviront uniquement à l'extraction de caractéristiques (*angl.: feature extraction*), lors de la première phase de l'entraînement.

5.3.3 Choix du CNN pré-entraîné

Pour choisir nos modèles CNN pré-entrainés, nos critères étaient les suivants :

1. Notoriété du modèle : pour débuter il est souhaitable d'utiliser des technologies bien connues et éprouvées,
2. Complexité du modèle : nos moyens étant limités, nous devions si possible choisir des modèles peu coûteux à entraîner,
3. Performance du modèle : dans la mesure où les autres critères sont satisfait, il faut évidemment choisir le meilleur modèle.

Pour faire une première sélection informée, nous nous sommes appuyés sur cet article, ainsi que [2018, arXiv:1810.00736v2]. De cette dernière publication est extraite la figure suivante, qui donne un aperçu de la performance (top-1 accuracy) en fonction de la complexité (nombre d'opérations par epoch d'entraînement et nombre de paramètres à entraîner).



Nous avons retenu les modèles suivants :

- ResNet50 [2015, arXiv:1512.03385v1] : un bon choix du point de vue de nos critères, et de plus c'était ce modèle qui avait été utilisé pour réaliser le benchmark du challenge, il devait donc naturellement être notre modèle de référence,
- Xception [2016, arXiv:1610.02357v3] : la famille Inception bénéficie d'une certaine popularité et cette variante bénéficiait de performances presque aussi bonnes que les dernières versions, tout en conservant une complexité faible,
- VGG16 [2014, arXiv:1409.1556v6] : un modèle bien connu, dont la complexité est en partie due à ses couches denses massives (4096 x 2), mais qui dans notre cas pouvaient être fortement réduites.

Afin de gagner du temps, nous avons décidé de tester les différents CNN pré-entraînés retenus sur un sous-ensemble équilibré contenant 750 observations par classes, tirées aléatoirement, en partant de l'hypothèse que les performances relatives de ces modèles auraient été similaires si on avait considéré l'ensemble des observations.

5.3.4 Couches denses cachées

Concernant nos blocs prédictifs, si la taille de la couche dense finale était contrainte par le nombre de classes à prédire (dans notre cas 27), il fallait aussi définir le nombre de couches denses cachées (*angl.: hidden layers*) et leurs tailles respectives (en nombre de neurones).

Premièrement, d'après un des théorèmes d'approximation universelle, un réseau de neurones composé de deux couches cachées de taille finie permet d'approximer pratiquement n'importe quelle fonction.

Deuxièmement, il semblerait que dans l'état de l'art actuel, le choix de la taille des couches denses cachées reste un domaine empirique. Cependant, certains principes sont couramment applicables :

1. Une architecture dont l'efficacité a été prouvée pour une certaine application devrait avoir des performances satisfaisantes pour une application similaire,
2. La taille optimale de chaque couche dépend de l'ensemble des couches précédentes et suivantes, mais en première approximation, on peut considérer les couches qui lui sont immédiatement adjacentes,
3. Diminuer la taille d'une couche dense donnée contribue à diminuer la capacité du réseau, ce qui à l'excès peut engendrer du sous-apprentissage (*angl.: underfitting*),
4. Augmenter la taille d'une couche dense donnée contribue à augmenter la capacité du réseau, ce qui à l'excès, en plus d'augmenter le coût de l'entraînement, peut engendrer du sur-apprentissage (*angl.: overfitting*),
5. Diminuer successivement les tailles des couches tend à compresser l'information, tandis que l'augmenter tend à la développer.

Dans le contexte du challenge ImageNet, le nombre de classes à prédire est de 1000, tandis que dans le cadre de notre étude, il est de 27. Il est donc évident que notre problème

ne nécessite pas un modèle d'une telle complexité, et nous avons donc pris le parti de réduire la taille de nos couches denses cachées par rapport à celles des modèles originaux afin de réduire le coût de l'entraînement. Nous avons ensuite ajusté ces tailles de manière empirique en fonction des résultats obtenus, tout en suivant les principes évoqués plus haut.

5.3.5 Régularisation

Le premier modèle basé sur Resnet50 n'ayant pas été très performant, et ayant apparemment tendance au surapprentissage, nous avons ajouté une couche de BatchNormalisation et une couche de Dropout après chaque couche dense cachée.
Suite à ce changement, les performances se sont améliorées, indépendamment du backbone choisi d'ailleurs.

5.3.6 Choix des hyperparamètres

Nous avons considéré les hyperparamètres suivants :

- Couches convolutives gelées : dans une seconde phase d'entraînement, il aurait pu être utile de dégeler les dernières couches convolutives, mais nous avons dû y renoncer.
- Taux de Dropout : le dropout exerce une forte régularisation, et il aurait été intéressant de le faire varier, mais nous avons privilégié d'autres paramètres,
- Fonction de perte : la *Categorical Cross-Entropy* est bien adaptée aux problèmes de classification multi-classes, nous avons donc privilégié d'autres paramètres,
- Optimiseur : Adam est un optimiseur adaptatif éprouvé et très populaire ces temps-ci, nous avons privilégié d'autres paramètres,
- Batch size et learning rate (LR) : le choix de ces hyperparamètres est très lié aux données traitées. Les augmenter peut accélérer sensiblement l'entraînement, mais risque cependant de nuire à l'optimisation. Un guide pratique complet est fourni dans cet article.

5.3.7 Comparaison des CNN

5.3.7.1 Résultats

Le tableau suivant résume les performances pour les types de CNN testés avec quelques variantes d'architecture et des hyperparamètres, et entraînes sur le même sous-ensemble équilibré (750 observations par classe).

CNN backbone	*Hidden layers*	*Batch size*	*LR*	*Batch norm.*	*Drop. rate*	*Epochs*	*Valid. acc.*	*Valid. F1 macro*	*Valid. F1 weighted*
Resnet50	[512, 256]	32	0.001	No	0	50	45.00%	42.00%	46.00%

Resnet50	[512, 256]	32	0.010	Yes	0.2	15	54.00%	50.00%	55.00%
VGG16	[256, 128]	32	0.001	Yes	0.2	20	49.00%	45.00%	50.00%
VGG16	[1024, 512]	32	0.001	Yes	0.2	15	52.00%	48.00%	53.00%
Xception	[2048, 1024]	128	0.001	Yes	0.375	20	53%	49.00%	53.00%

Nous sommes conscients que changer plusieurs hyperparamètres entre chaque essai ne constitue pas un plan d'expérience idéal (pour décorréler les influences respectives des paramètres), cependant c'était un bon compromis compte tenu du temps et des moyens dont nous disposions.

Ce tableau permet malgré tout plusieurs constats intéressants.

Premièrement, en comparant les deux essais basés sur *ResNet50*, on constate clairement l'effet bénéfique de la régularisation par *BatchNormalisation* et *Dropout*.

Deuxièmement, dans le cas du VGG16, avec des couches cachées de taille très modeste, le premier modèle semblait souffrir d'un déficit de capacité, et converge rapidement vers un état de sous-apprentissage. L'augmentation de la taille des couches cachées, et donc de la capacité, a permis d'améliorer sensiblement la performance.

5.3.7.2 Cheminement

Bien que sa piètre performance fût davantage imputable à l'absence de régularisation des couches denses qu'à l'architecture, le premier essai avec un ResNet50 a semblé décevant, ce qui nous a incités à chercher une alternative. Rétrospectivement, lors du second essai, qui s'est déroulé bien plus tard, il est apparu que c'était en fait un bon candidat.

Après avoir ajouté de la régularisation, les essais avec VGG16 donnaient des résultats prometteurs, mais nous avons voulu tout de même tester autre chose.

Le modèle Xception a fourni des performances comparables à VGG16 mais au prix d'un entraînement plus long (du moins avec les couches denses et hyperparamètres testés).
Finalement, poussés par la nécessité d'avancer, nous nous sommes concentrés sur le modèle VGG16, qui nous paraissait le plus fiable.

5.3.8 Entraînement du modèle

L'entraînement a été réalisé à partir des *DataFrameIterator* d'entraînement et de validation, qui envoient les images et les labels associés vers la méthode *fit()* (de la classe *Model* de *tensorflow.keras.models*), en configurant pour la fin de chaque epoch l'appel de fonctions dites *callbacks* (héritant de la classe *Callback* de *tensorflow.keras.callbacks)*. Ces dernières sont détaillées dans ce qui suit.

Les données contenues dans ces *DataFrameIterator* étaient issues soit pour la phase de choix du CNN, du sous-ensemble équilibré contenant 750 observations par classe, soit pour la phase d'optimisation, de l'ensemble des observations.

Les métriques surveillées durant l'entraînement étaient au départ seulement la valeur de la fonction de perte et l'*accuracy*, puis grâce au *callback* personnalisé *GeneratorMetrics également* les métriques *precision*, *recall* et *F1-score* agrégés par la méthode *weighted*.

Seuls deux modèles basés sur l'architecture VGG16 ont bénéficié de la phase d'optimisation.

Pour finir, tous les modèles testés ont été entraînés dans les conditions suivantes :

- Blocs convolutifs gelés : Tous
- Pourcentage de validation : 80 %
- Augmentation des images :
    - Inversion Gauche-Droite : OUI
    - Décalage maximal : +/-10%
    - Rotation maximale : +/-10°
    - Zoom maximal : +/-10%
- Optimiseur : *ADAM*
- Fonction de perte : *Sparse Categorical Cross-Entropy*

5.3.9 Callbacks

Pour accompagner l'entraînement des modèles, nous avons progressivement ajouté les callbacks suivants au cours de l'évolution du projet :

- *ModelCheckpoint* et *CSVLogger* permettent de sauvegarder régulièrement le modèle et les métriques pendant l'entraînement. C'est particulièrement utile quand le code est exécuté à distance dans une plate-forme telle que Google Colaboratory, qui ne garantit pas un accès au service sur une durée prolongée.
- *ReduceLROnPlateau* applique une procédure de régulation du taux d'apprentissage simple et efficace, qui offre un surcroît d''adaptabilité appréciable même avec les optimiseurs récents. Cela nous a permis de fixer le taux d'apprentissage initial à une valeur plus élevée, accélérant ainsi l'entraînement.
- L'utilité de *EarlyStopping* a probablement été masquée ici par la présence de ReduceLROnPlateau, mais c'est un bon moyen d'éviter de gaspiller du temps et de la puissance de calcul quand cela n'apporte plus rien.
- *GeneratorMetrics* est un callback personnalisé inspiré de ce notebook. Il permet de calculer des métriques liées aux classes telles que la précision, le rappel et le F1-score, à la fin de l'epoch plutôt que par lots, comme recommandé par cet article.

Notons que les callbacks *ReduceLROnPlateau* et *EarlyStopping* ont été configurés pour s'activer en cas de stagnation de la valeur de la fonction de perte de validation (paramètre *val_loss*). Cela nous a semblé être un moyen à la fois simple et fiable d'évaluer l'évolution de la performance.

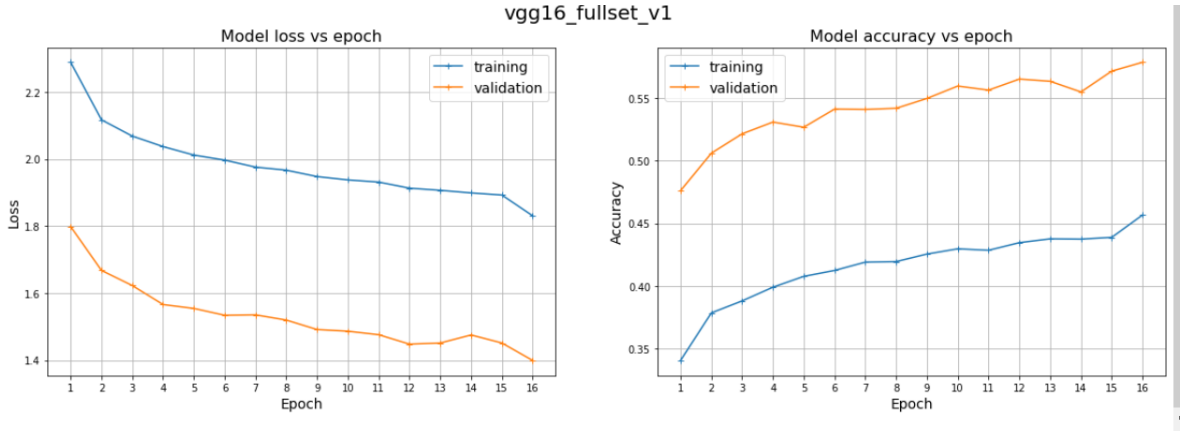
5.3.10 Optimisation du modèle

A l'occasion de la phase d'optimisation, deux modèles basés sur l'architecture *VGG16* ont été entraînés sur le jeu de données complet.

5.3.10.1 Premier essai

Le premier modèle a été entraîné avec une taille de batch et un learning rate initial plus grands (64 et 0.01) que ce que nous avions choisi précédemment (32 et 0.001), afin de garder un temps d'entraînement raisonnable sur le jeu de données complet.

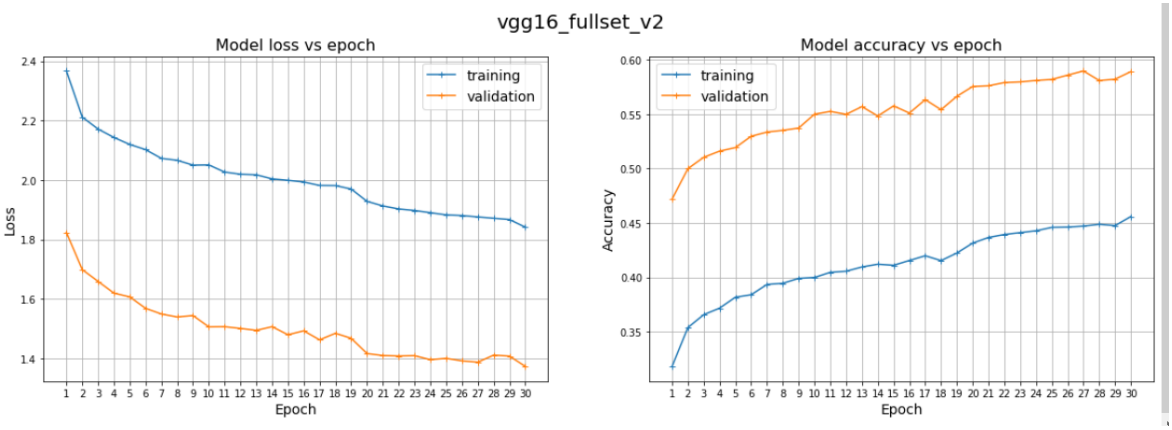
L'entraînement a été interrompu par une déconnexion de Google Colaboratory au bout de 16 epoch, mais les résultats étaient encourageants. L'historique est affiché ci-dessous.



On observe au passage que le modèle présente un meilleur score sur le jeu de données de validation que sur celui d'entraînement. Cela peut être attribué à l'augmentation d'images qui vient perturber les observations d'entraînement, mais pas celles de validation.

5.3.10.2 Deuxième essai

A ce stade, le projet touchait à sa fin, et par précaution, nous avons rétabli la taille de batch (de 32) pour sécuriser l'optimisation. Le learning rate étant monitoré par le *callback ReduceLROnPlateau*, nous n'avons pas changé la valeur initiale (de 0.01).



Au bout d'une trentaine d'epochs, on converge vers un plafond malgré des réductions du learning rate aux epochs 19 et 29 (imprimés dans les logs). Difficile de dire si la diminution de la taille de batch a joué positivement.

Nous avons arrêté là l'entraînement, et sommes passés à l'évaluation détaillée.

5.3.11 Evaluation du modèle

En principe l'évaluation devrait se faire sur un jeu de données de test, différent de celui de validation. Cependant, les labels du jeu de test du challenge ne sont pas fournis aux participants, mais sont utilisés par le jury pour évaluer les productions soumises. Il nous avait échappé que les délais étaient malheureusement trop serrés pour en bénéficier. Nous nous sommes donc contentés ici d'évaluer la performance sur le jeu de données de validation de manière plus détaillée que pendant l'entraînement.

Les caractéristiques et performances des deux modèles retenus sont résumées dans le tableau suivant.

CNN backbone	*Hidden layers*	*Batch size*	*LR*	*Batch norm.*	*Drop. rate*	*Epochs*	*Valid. acc.*	*Valid. F1 macro*	*Valid. F1 weighted*
VGG16	[1024, 512]	64	0.010	Yes	0.2	16	58.00%	52.00%	57.00%
VGG16	[1024, 512]	32	0.010	Yes	0.25	30	59.00%	53.00%	58.00%

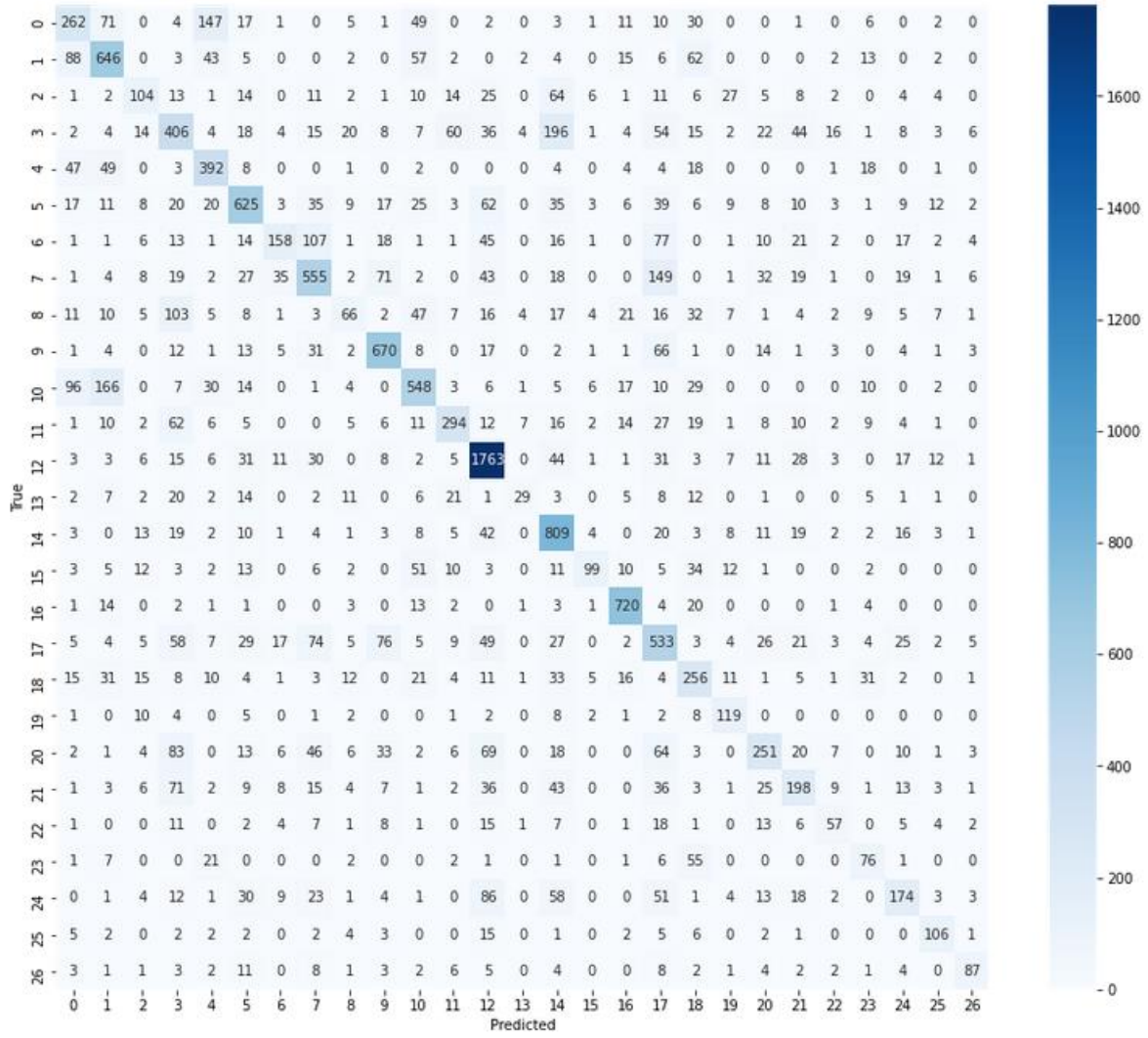
Le rapport de classification présenté ci-dessous permet de constater les écarts de F1-score entre les classes : par exemple 80% pour la classe 12 contre 37% pour la classe 2. Les différences de cardinaux entre ces classes (2042 contre 336 sur le jeu de validation) ne sont certainement pas étrangères à cela. Cela explique notamment la différence de 6% entre les F1-scores *weighted* et *macro*.

Model: 'vgg16_fullset_v2'

	precision	recall	f1-score	support
0	0.46	0.42	0.44	623
1	0.61	0.68	0.64	952
2	0.46	0.31	0.37	336
3	0.42	0.42	0.42	974
4	0.55	0.71	0.62	552
5	0.66	0.63	0.64	998
6	0.60	0.31	0.40	518
7	0.57	0.55	0.56	1015
8	0.38	0.16	0.22	414
9	0.71	0.78	0.74	861
10	0.62	0.57	0.60	955
11	0.64	0.55	0.59	534
12	0.75	0.86	0.80	2042
13	0.58	0.19	0.29	153
14	0.56	0.80	0.66	1009
15	0.72	0.35	0.47	284
16	0.84	0.91	0.88	791
17	0.42	0.53	0.47	998
18	0.41	0.51	0.45	502
19	0.55	0.72	0.62	166
20	0.55	0.39	0.45	648
21	0.45	0.40	0.42	498
22	0.47	0.35	0.40	165
23	0.39	0.44	0.41	174
24	0.51	0.35	0.42	499
25	0.61	0.66	0.63	161
26	0.69	0.54	0.60	161
accuracy			0.59	16983
macro avg	0.56	0.52	0.53	16983
weighted avg	0.59	0.59	0.58	16983

L’analyse de la matrice de confusion présentée ci-dessous permet non seulement d’observer ces différences entre classes mais également d’identifier les paires de classes fréquemment confondues.

Model: 'vgg16_fullset_v2'



On voit que certaines confusions se font en volume considérables. Pour y voir plus clairement, nous avons filtré les plus notables à partir d'un seuil élevé (considérant le nombre de classes).

```
>> Notable confusions between classes (threshold=15.0%)
Class #00 was mistaken for class #04 in about 24% of occurences
Class #02 was mistaken for class #14 in about 19% of occurences
Class #03 was mistaken for class #14 in about 20% of occurences
Class #06 was mistaken for class #07 in about 21% of occurences
Class #08 was mistaken for class #03 in about 25% of occurences
Class #10 was mistaken for class #01 in about 17% of occurences
Class #15 was mistaken for class #10 in about 18% of occurences
Class #23 was mistaken for class #18 in about 32% of occurences
Class #24 was mistaken for class #12 in about 17% of occurences
```

5.3.12 Conclusions

Le modèle retenu présente des formances plutôt satisfaisantes, et même légèrement meilleures qu'attendues puisque nous avons un weighted F1-score de 58% contre 55,34% annoncés dans le benchmark du challenge.

Cependant, il est certainement possible d'améliorer les performances en explorant et en optimisant davantage les points suivants :

- Constitution du jeu de données (rééquilibrage),
- Paramètres d'augmentation des données,
- Type de CNN du backbone,
- Couches convolutives non gelées,
- Tailles des couches denses cachées,
- Taux de Dropout dans les couches denses,
- Fonction de perte (avec éventuellement pénalisations),
- Optimiseur (d'après cet article),
- Learning rate initial,
- Taille de batch.

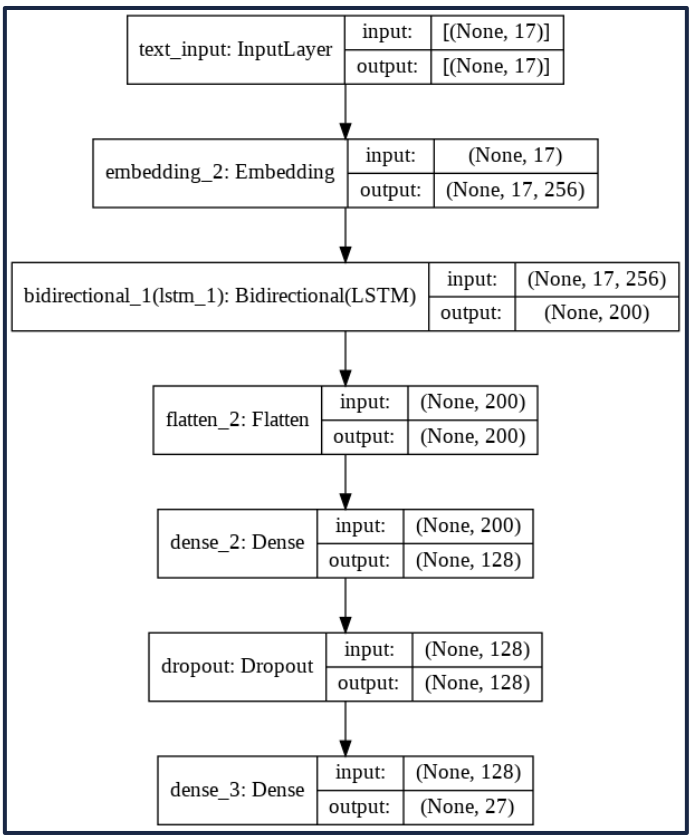
5.4 CLASSIFICATION A PARTIR DU TEXTE

5.4.1 Architecture du modèle de classification

Nous avons utilisé différents modèles de deep learning pour le traitement de texte :

- Conv 1D : Comme la colonne désignation ne représente pas forcément des phrases, un réseau convolutionnel à une dimension nous a paru légitime de le tester dont l'avantage est l'extraction de features avec un entraînement plus rapide que les RNN.
- RNN : Ces modèles sont plus adaptés aux données en séquences mais présentent le problème du vanishing gradient : provoqué par la diminution très rapide des valeurs des gradients pendant la rétropropagation ce qui entraîne l'annulation du gradient et l'arrêt de l'apprentissage.
- LSTM : Ces réseaux sont particulièrement performant pour le vanishing gradient lors de l'entraînement grâce à leur mémoire qui ne garde que ce qui est important pour la prédiction et d'oublier ce qui ne l'est pas.
- GRU : Ces réseaux sont également performants contre le vanishing gradient mais ne sont pas dotés du forget gate, ce qui les rend moins complexes et plus rapides à entraîner.
- Bi_LSTM : ou LSTM BiDirectionnel, dont l'avantage est de parcourir la séquence de mots dans les deux sens, ceci permet de prédire pour chaque mot un output prenant en compte le contexte passé et le contexte futur.

Architecture du meilleur modèle obtenu (cf. comparaison des performances des modèles ci-dessous) :



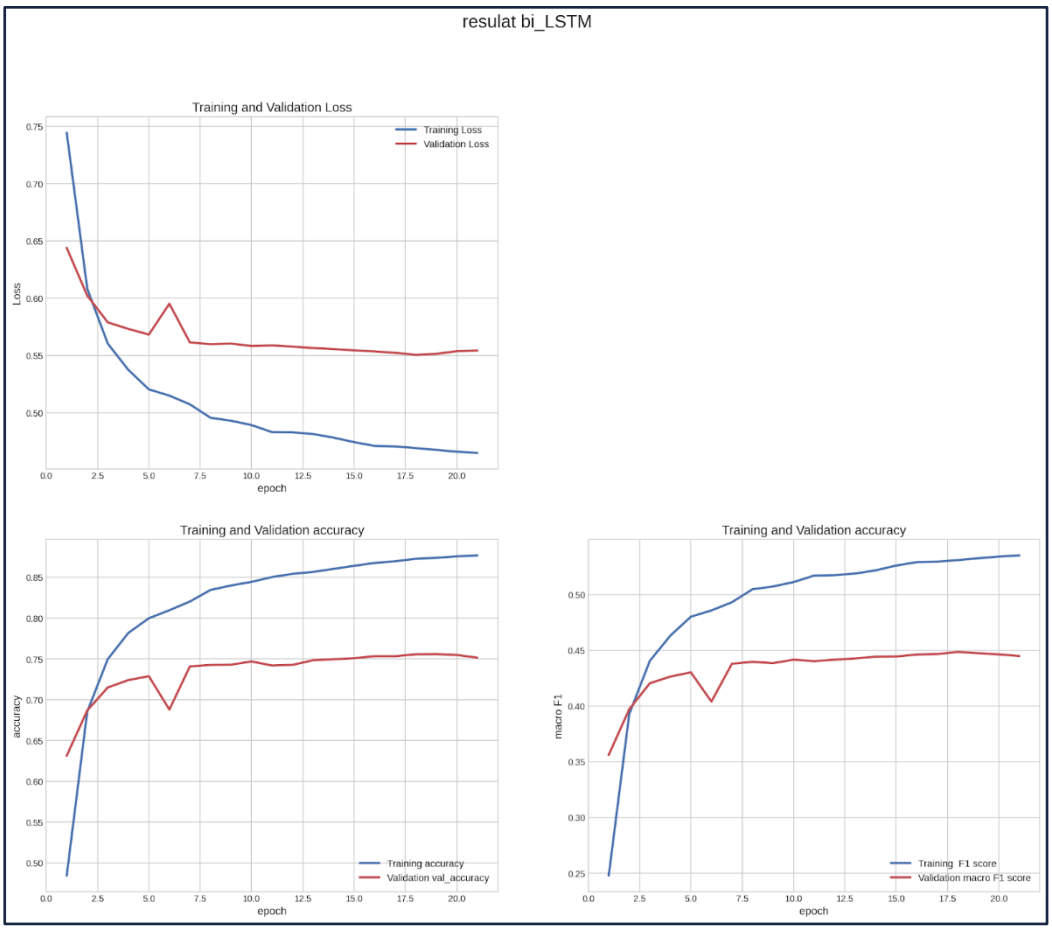
5.4.2 Choix des hyperparamètres :

Nous avons considéré les hyperparamètres suivants :

- Batch size : 1000 (ordre de grandeur du nombre de désignations/nombre d'epochs (20 ou 30 selon les modèles)).
- L'optimizer : Nous avons testé "SDG", "Adam" et "Nadam". Nadam donné de meilleurs résultats, que Adam et SDG, cet optimizer est un adam amélioré puisque :
    - Adam applique multiplie le learning rate par un momentum (qui dépendant du gradient) qui va adapter l'accélération de la descente de gradient selon si l'on est proche ou non du minimum local.
    - Nadam est une descente de gradient de Nesterov qui se déplace de "deux pas" dans la descente de gradient, il calcule d'abord un point intermédiaire grâce au momentum, puis applique la descente de gradient à partir de ce point.
- Learning Rate : avec l'optimizer nadam nous avons décidé de garder un learning rate constant étant donné que cet optimizer est doté de deux momentums. En commençant avec un learning rate de 10^-2
- Utilisation de la loss customisé F1. Comme nous cherchons à maximiser la métrique F1, et qu'il n'existe pas de loss F1 dans la libraire tensorflow. Nous avons créé une loss qui permet de le faire. La métrique F1 n'étant pas différentiable, on ne peut pas l'utiliser comme fonction de perte directement dans le modèle (en calculant 1-F1) . Nous avons donc modifié le score F1 pour le rendre différenciable. Au lieu de calculer le nombre de vrais positifs, faux positifs, faux négatifs sous forme de valeurs entières discrètes, nous les avons calculés comme une somme continue de valeurs de vraisemblance en utilisant des probabilités et une fonction d'activation softmax. (Cf notebooks « classification_text_partie2.ipynb
- Nombre d'epochs : entre 20 et 30

5.4.3 Entraînement du modèle

Voici les graphiques de l'entrainement du modèle :



Tous les réseaux entraînés semblent overfitter et ce malgré de simples réseaux de deep learning avec différents dropouts.

Pour nous ceci peut s'expliquer par deux facteurs :

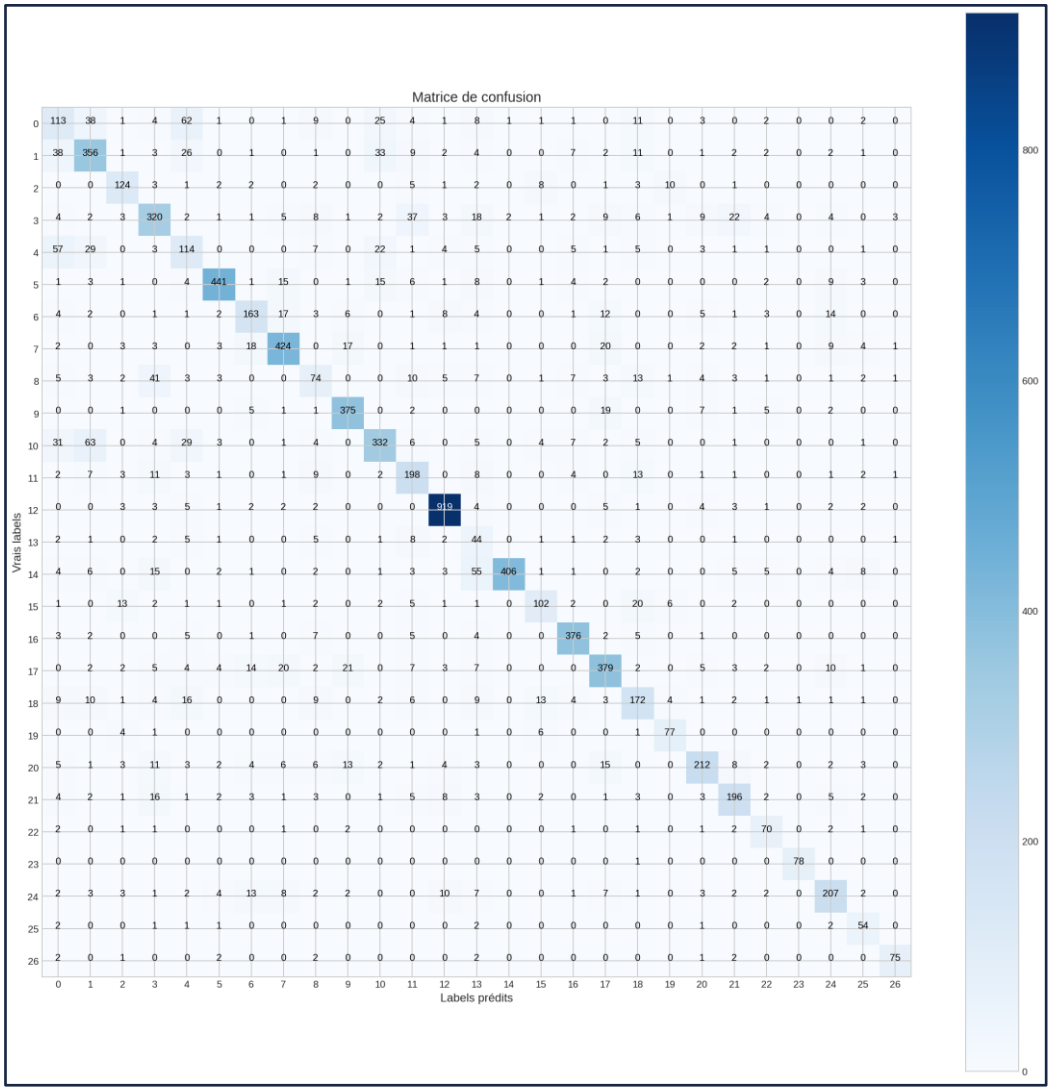
1) Il faut que l'on teste d'autres architectures avec d'autres hyperparamètres (mais les ressources et le temps dont nous disposons ne nous ont pas permis d'explorer plus que les modèles et hyperparamètres listés ci-dessus).
2) La simplicité des données de la colonne « designation », qui ne contient que très peu de mots par rapport à la tache de classification que l'on cherche à effectuer.

5.4.4 Evaluation du modèle

Ci-dessous les résultats des différents modèles, sur les données d'entrainement de validation et de test qui représentent respectivement 70%, 20% et 10% du dataset total.

models	macro_f1 train	w_f1 train	accuracy train	macro_f1 val	w_f1 val	accuracy val	macro_f1 test	w_f1 test	accuracy test
lstm	75,2%	83,1%	82,9%	64,2%	72,4%	71,8%	64,5%	72,8%	72,0%
bi_lstm 200 units	97,7%	97,8%	97,7%	75,8%	77,3%	77,3%	76,5%	77,7%	77,6%
bi_lstm 256 units	86,0%	88,5%	88,0%	72,5%	76,2%	75,6%	72,2%	75,8%	75,4%
gru	81,3%	84,4%	83,4%	69,4%	73,3%	72,0%	69,1%	73,2%	71,8%

Le meilleur modèle est le bi_LSTM à 200 neurones (100 LSTM en bidirectionnel), qui atteint un score Weighted F1 de 78% sur l'ensemble test.



```
La classe 0 a souvent été prise pour la classe 1 , nb d'erreurs = 38
La classe 0 a souvent été prise pour la classe 4 , nb d'erreurs = 62
La classe 0 a souvent été prise pour la classe 10 , nb d'erreurs = 25
La classe 1 a souvent été prise pour la classe 0 , nb d'erreurs = 38
La classe 1 a souvent été prise pour la classe 4 , nb d'erreurs = 26
La classe 1 a souvent été prise pour la classe 10 , nb d'erreurs = 33
La classe 3 a souvent été prise pour la classe 11 , nb d'erreurs = 37
La classe 3 a souvent été prise pour la classe 21 , nb d'erreurs = 22
La classe 4 a souvent été prise pour la classe 0 , nb d'erreurs = 57
La classe 4 a souvent été prise pour la classe 1 , nb d'erreurs = 29
La classe 4 a souvent été prise pour la classe 10 , nb d'erreurs = 22
La classe 8 a souvent été prise pour la classe 3 , nb d'erreurs = 41
La classe 10 a souvent été prise pour la classe 0 , nb d'erreurs = 31
La classe 10 a souvent été prise pour la classe 1 , nb d'erreurs = 63
La classe 10 a souvent été prise pour la classe 4 , nb d'erreurs = 29
La classe 14 a souvent été prise pour la classe 13 , nb d'erreurs = 55
La classe 17 a souvent été prise pour la classe 9 , nb d'erreurs = 21
```

Les résultats sur le test set sont satisfaisants et ce malgré de déséquilibre des données. Le nombre de plus de 20 erreurs de prédictions est également petit : (cf. paragraphe analyses d'erreurs ci-dessous pour plus de détails)

5.5 CLASSIFICATION BIMODALE

L'objectif de cette partie est d'entrainer un modèle sur les données textes et images à la fois et prédire ainsi la classe des produits e-commerce.

5.5.1 Transformation des données

Les données étant volumineuses, et en utilisant Colab, il nous a été impossible de générer les données à partir du drive avec la création d'un générateur de données « Dataset » de tensorflow sur les données brutes.

Nous avons opté pour une transformation des données en fichiers TFRecords. C'est une transformation en format binaire proposé par tensorflow qui diminue de façon importante le volume des données. Ce format binaire est en générale lu beaucoup de façon plus rapide à partir de la mémoire disque, de plus, comme la construction de ces fichiers est réalisé grâce à Tensorflow, ce format a été particulièrement optimisé pour la lecture des données avec cette librairie.

A partir de ces fichiers nous créons un objet tf.data.Dataset qui va permettre de lire les données binaires par batch en mémoire, leur appliquer un preprocessing et transmettre au modèle de deep learning (pour plus de détails voir notebook « classification_multimodale_image_texte »).

5.5.2 Architecture du modèle de classification

Les modèles utilisé ont l'architecture suivante :

- Un modèle pour encoder les données images : transfer learning à partir des réseaux Xception et MobileNet puis entrainement des top layers sur Rakuten
- Un modèle pour encoder les données textes : transfer learning à partir du modèle Bert, mais également un réseau Bi_LSTM comme celui qui a obtenu les meillleurs performances lors de l'entrainement sur les données textes uniquement.
- Puis modèle customisé qui prend en entrée les deux modèles créés précédemment, concatène les représentations textes et images, s'entraine sur ces deux données, et prédit la classe des produits.

Pour des raisons de temps et de ressources nous n'avons pas pu entrainer un modèle VGG16, ni explorer toutes les combinaisons possibles pour avoir des bonnes performances.

Motivation des choix des modèles images : nous nous sommes intéressés aux 3 modèles cité ci-dessus car :

- VGG16 : un réseau de convolution classique à tester, il est utilisé également dans le notebook classification image.
- Xception : utilise plusieurs techniques d'amélioration de performances, comme les "Depthwise Separable Convolution" qui sont des convolutions optimisées pour le calcul (les filtres sont appliqués d'abord par Channel séparément puis une convolution 1*1 appliqué sur tous les channels regroupe les informations des channels), les connexions ResNet qui permettent d'apprendre en profondeur en évitant le vanishing gradient. Ce modèle surperforme VGG16, ResNet et Inception dans l'état de l'art.
- MobileNet : version optimisée de Xception (plus rapide), créé par Google, pour les applications mobiles.

5.5.3 Choix des hyperparamètres

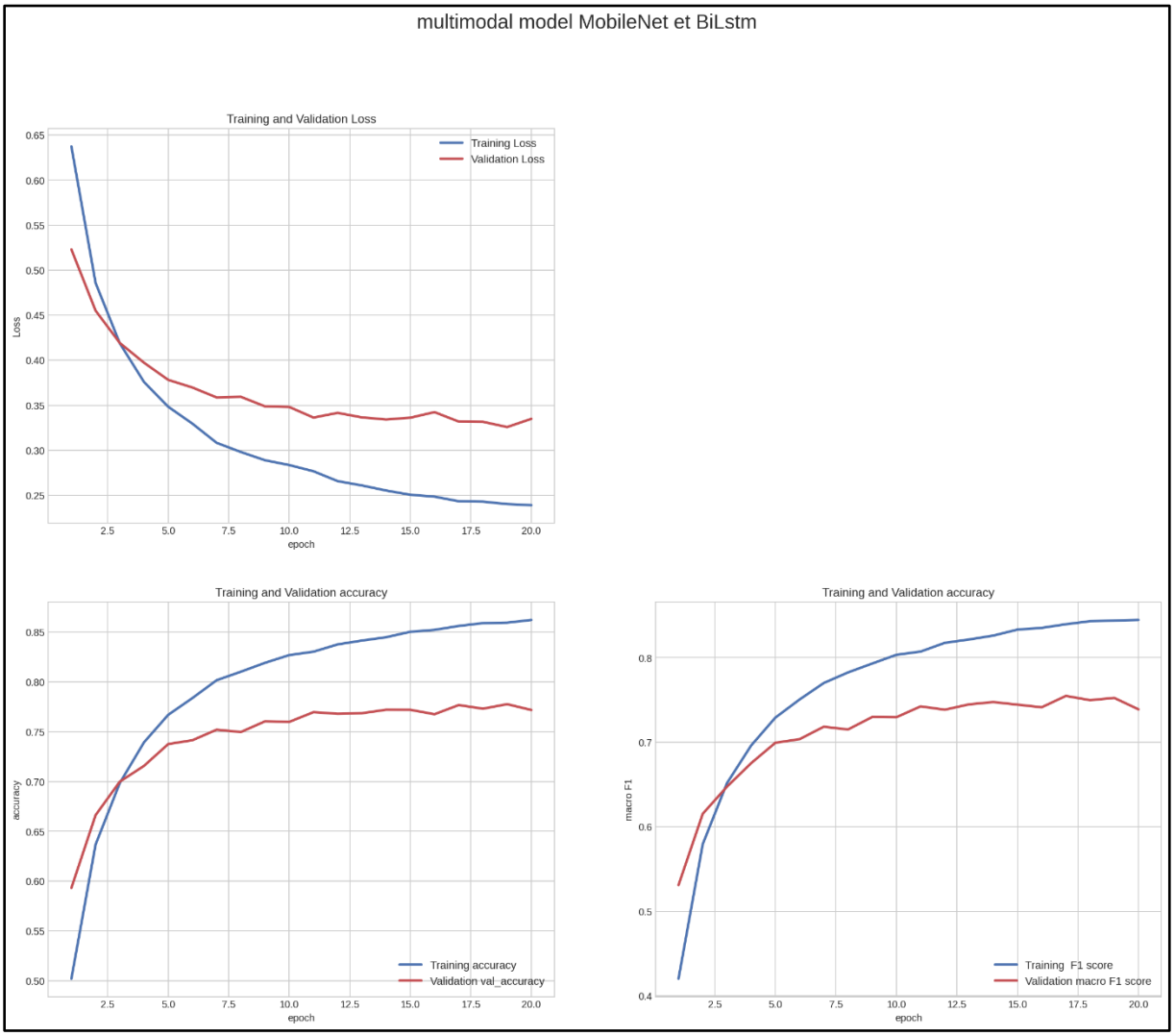
Nous avons considéré les hyperparamètres suivants :

- Utilisation de l'optimizer Nadam comme pour l'entrainement sur les données textes.
- Utilisation des batch size 64 et 128. Pas d'impact notable sur les performances de prédiction, mais un entrainement un peu plus rapide.
- Utilisation de la loss F1 customisée.
- Nombre d'epochs de 20.

5.5.4 Entraînement du modèle

Le modèle décrit par la suite et celui qui s'entraine sur la concaténation des modèles MobileNet pour les images, Bi_LSTM pour le texte. Suite aux problèmes de coupure de connexion sur Colab, nous n'avons pas pu récupérer les poids du modèle [Xception_LSTM].

Le modèle [Xception_Bert] n'a pas eu de bonnes performances (cf. notebook «classification_multimodale_image_text ») .



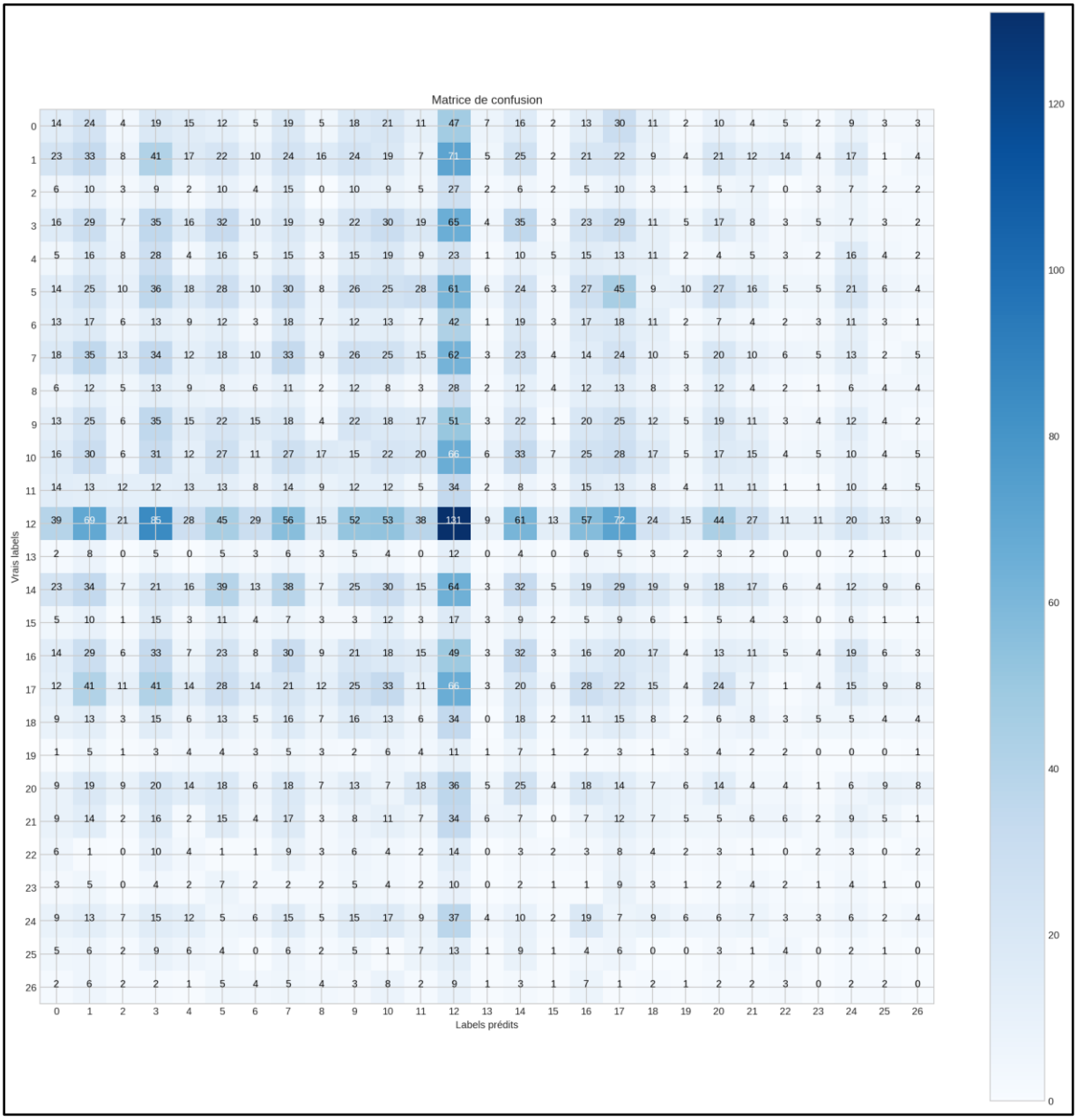
Ce modèle semble moins overfitté que le modèle sur les données textes, il a été entrainé sur 20 epochs seulement, mais aurait pu avoir de meilleurs performance probablement avec plus d'entrainement puisque la val_loss continuait de baisser même à la fin de l'entrainement.

5.5.5 Evaluation du modèle

Voici les scores de ce modèle :

```
Training Accuracy: 0.8690 _ Training F1 macro : 0.8495
Validation Accuracy: 0.7718 _ Validation F1 macro : 0.7389
Testing Accuracy:  0.7687 _ Testing F1 macro : 0.7378
```

La matrice de confusion :



Malgré des scores assez élevés, ce modèle semble avoir appris à prédire uniquement la classe majoritaire, et obtient un taux d'erreurs élevés sur les autres classes. La liste des

erreurs étant élevé, elle est consultable sur le notebook classification multimodale. (cf. analyse des erreurs pour plus d'analyses).

5.5.6 Pistes d'amélioration

Nous avons considéré les possibilités d'améliorations suivantes :

- Entrainement sur les données rééquilibrées
- Utiliser une représentation du texte plus intelligente comme Glove
- Utiliser du transfert Text learning à partir du modèle Camembert version française de Bert.
- Continuer d'entrainer ce modèle comme la loss n'a pas arreté de baisser à la fin des 20 epoch.
- Utiliser d'autres optimizers

5.6 ANALYSE DES ERREURS DE PREDICTION

Tout d'abord nous avons analysé les scores des métriques F1-score et accuracy sur les ensembles de validation et de test. Cependant notre dataset n'est pas très équilibré : nous disposons de 27 classes, donc pour un équilibre parfait chaque classe devrait représenter environ 3.7%, or la classe 12 représente à elle seule 12% des données. Nous avons par conséquent analysé la matrice de confusion ainsi que le nombre d'erreurs supérieur à 20 (nous disposons d'un ensemble de test de 8491 exemples). Cela nous a permis de visualiser et de comprendre sur quelles classes le modèle se trompe, et de vérifier si le modèle n'a pas appris à prédire uniquement la classe majoritaire.

Comme le F1-score de référence est de 55% seulement (le nôtre est à 78% sur les données texte sur l'ensemble de test) nous ne pouvons-nous baser uniquement sur cette métrique pour juger de la qualité du modèle par exemple les modèles multimodaux (cf. notebook multimodal classification) qui obtiennent des scores élevés de F1 prédisent la classe majoritaire relativement bien, et les autres classes sont prédites quasi aléatoirement . Nous avons donc cherché à maximiser les métriques et à diminuer ce nombre d'erreurs.

5.7 DIFFICULTES RENCONTREES

5.7.1 Limitation de la puissance de calcul

Étant donné le nombre et le volume global des fichiers images, il était inconcevable de réaliser l'entraînement des modèles en local sur notre PC avec un simple CPU. L'entraînement des modèles n'aurait donc pu se faire sans la plateforme Google Colaboratory, qui met à disposition, gratuitement des machines virtuelles linux pourvues de GPU destinées à exécuter des notebooks Jupyter pour des applications de calcul scientifique.

Malheureusement, ce service étant gratuit, l'utilisation des GPUs est monitorée et limitée par une politique opaque, menant à des déconnexions de l'environnement non prévisibles avec précision (à notre connaissance). Ces déconnexions peuvent interrompre un entraînement en cours et ont de fait limité notablement nos essais.

5.7.2 Gestion des fichiers images

Dans un premier temps, nous avons copié le dossier contenant les 84916 fichiers images sur Google Drive, monté cet espace comme disque réseau sur la machine virtuelle hôte, puis tenté d'accéder directement aux fichiers depuis le notebook. Cette méthode a deux inconvénients majeurs :

1. Le temps nécessaire pour uploader les fichiers depuis le PC fut très important (une bonne demi-douzaine d'heures). En effet, le nombre de fichiers crée un overhead considérable, et on peut supposer que la connexion n'est pas parfaitement stable et que le débit est bridé,
2. Pendant l'exécution du code, pour accéder aux fichiers l'environnement Python doit, via la machine virtuelle linux hôte, faire une requête à Google Drive à chaque fois qu'on accède à une image. Cela ralentit de manière colossale l'exécution et rend la vie dure à Google Drive, qui en réponse renvoie régulièrement des timeouts.

Une bien meilleure solution consiste à transférer sur Google Drive l'ensemble des fichiers images dans un dossier d'archive compressé (e.g. format ZIP), et au début de l'exécution commander à la machine virtuelle linux hôte de décompresser l'archive dans un dossier temporaire qui sera utilisé par suite. Cette solution ne présente aucun des deux inconvénients précédents et accélère drastiquement l'exécution.

5.7.3 Utilisation de la classe ImageDataGenerator

Pour mémoire, la méthode *flow_from_dataframe()* de la classe *ImageDataGenerator* (du module *tensorflow.keras.preprocessing.image*), va chercher dans un DataFrame fourni les chemins des images et les labels dans des colonnes à spécifier, et renvoie un objet *DataFrameIterator.* Cet objet fournit les images transformées selon les paramètres spécifiés lors de la création de l'objet *ImageDataGenerator* appelant, et les labels associés.

Malgré sa puissance, l'utilisation de cette classe présente un certain nombre d'écueils (*angl.: pitfalls*) pour l'utilisateur néophyte, comme nous l'avons expérimenté à nos dépens. Heureusement, nous avons trouvé ces points couverts dans cet article.

Le premier écueil concerne la manière dont sont générées les images. En effet, par défaut, les images sont mélangées aléatoirement (*shuffle = True*), ce qui, lorsqu'on l'ignore, peut poser deux problèmes :

1. A moins de rendre ce procédé déterministe en définissant une graine (paramètre *seed*), l'ordre des images et leurs augmentations sont inconnues, ce qui rend l'expérience non reproductible,

2. Si on applique ce mélange aléatoire aux données de validation, pour une raison vraisemblablement propre à l'implémentation, la correspondance avec les labels est perdue et les prédictions apparaissent comme fausses.

Le second écueil concerne la manière dont sont réalisées les prédictions catégorielles lorsque la variable cible n'est pas dichotomisée (*class_mode = 'sparse'*). Les prédictions sont les probabilités des différentes classes, mais ordonnés d'une manière, non pas intuitive (numérique ou alphanumérique), mais différente et propre à l'implémentation. La correspondance entre les indices des classes et leurs labels originels est donnée par l'attribut *class_indices* de l'objet *DataFrameIterator* utilisé.

6 CONCLUSION

Nous avons au cours de ce projet développé nos connaissances des domaines du *Natural Language Processing (NLP*), de la *Computer Vision (CV)*, de la classification multi-classe et multimodale, ainsi que nos compétences en *DataViz* et en *Deep Learning* au travers de modèles RNN et CNN.

Par ailleurs ce projet nous a permis d'aborder plusieurs environnements techniques. D'abord l'écosystème Anaconda et ses environnements Python, avec les applications Jupyter (et Spyder) en local sur nos PCs. En parallèle, le système de *versioning* Git / GitHub et les logiciels permettant de faciliter l'interfaçage (e.g. GitHub Desktop). Puis, les notebooks Google Colaboratory hébergés sur des machines virtuelles linux, avec stockage sur Google Drive.

En termes de métier Data Science, nous avons réalisé une étude dans laquelle nous avons conçu, implémenté et évalués des prototypes de modèle de classification dans l'état de l'art.

Comme évoqué dans les parties concernées aux différents modèles, si dans la plupart des cas les performances obtenues peuvent être considéré comme satisfaisantes, nous sommes loin d'avoir pu explorer toutes les pistes d'amélioration. Ce sera là, nous gageons, l'enjeu de nos prochaines années en tant que Data Scientists.

7 ANNEXES

7.1 DESCRIPTION DES FICHIERS FOURNIS

FICHIERS	DESCRIPTION
classification_images.ipynb	Notebook de classification à partir des images.
classification_texte_partie1.ipynb	Notebook de classification à partir du texte avec différents modèles utilisés
classification_texte_partie2.ipynb	
classification_multimodale_image_texte.ipynb	Notebook de classification multimodale à partir des images et textes
data_exploration.ipynb	Notebook d'exploration des données
data_viz_image.ipynb	Notebook de visualisation des images
data_viz_text.ipynb	Notebook de visualisation du texte
imgtools.py	Fonctions personnalisées associées aux notebooks data_exploration.ipynb et data_viz_image.ipynb.
pre_processing.py	Fonctions personnalisées associées au notebook data_viz_text.ipynb.
prdtype_decode.csv	Table de correspondance entre les modalités du champ "producttypecode", et le type de produit "producttype" que nous avons déduit de la phase d'exploration des données.
labels_encoding.csv	Table de correspondance entre les labels et les modalités du champ "producttypecode".
trainset.csv	Jeu de données formaté à l'issue de la phase de visualisation des données.
fullset1.csv	Jeu de données complet formaté pour la classification à partir des images.
subset1.csv	Jeu de données partiel et équilibré, formaté pour la classification à partir des images.