Classification d'Images pour la Catégorisation de Produits sur un Site de E-Commerce

Z Li, E Guàrdia-Sebaoun, M. Bevilacqua, C Grauer, M Cornec, B. Goutorbe

Classification d'Images pour la Catégorisation de Produits sur un Site de E-Commerce

Z. LI[1] E. GUÀRDIA-SEBAOUN[2] M. BEVILACQUA[2] C. GRAUER[2] M. CORNEC[2] B. GOUTORBE[2]
[1] pierrezhiwei@gmail.com [2] datascience@cdiscount.com

Domaine principal de recherche : IA
Papier soumis dans le cadre de la journée commune : OUI

Résumé

Avec l'ouverture de ses rayons à des vendeurs externes (via sa place de marché), Cdiscount voit la taille de son catalogue exploser, passant de 10 millions de produits fin 2015 à plus de 30 millions de produits à la fin de l'année 2017. Face à un flux de produits arrivant sur la place de marché toujours plus important, la création manuelle de nouveaux produits est devenue impossible, rendant nécessaire l'automatisation de certaines étapes, notamment la catégorisation.
Dans un premier temps, nous avons attaqué le problème sous un angle sémantique. Si ces méthodes nous ont permis de traiter une bonne partie du flux d'entrée, il reste cependant une part non négligeable de produits non catégorisés. Le but de ces travaux est d'utiliser les images associées aux produits pour catégoriser ce reliquat.

Mots Clef

Classification déséquilibrée, classification en grand nombre de classes, classification d'images, RNN, CNN, vision par ordinateur.

Abstract

Over the past two years, Cdiscount's product catalog grew from 10 to 30 million products which made it impossible to keep integrating products manually. It became clear we needed to automatize the integration process, starting with categorization.
An approach based on semantic analysis enabled us to categorize the major part of the integration flow. However, our models reached a plateau, leaving a non negligeable part of the products uncategorized. In this paper, we propose to use image classification to categorize the remaining products.

Keywords

Unbalanced classification, extreme classification, image classification, RNN, CNN, computer vision.

1 Introduction

Cdiscount a généré plus de 3 milliards d'euros de volume d'affaires en 2016, ce qui en fait le deuxième acteur e-commerce en France. Avec l'ouverture de ses rayons à des vendeurs externes (via sa place de marché), Cdiscount voit la taille de son catalogue exploser, passant de 10 millions de produits fin 2015 à plus de 30 millions de produits à la fin de l'année 2017. Face à un flux de produits arrivant sur la place de marché toujours plus important, la création manuelle de nouveaux produits est devenue impossible, rendant nécessaire l'automatisation de certaines étapes, notamment la catégorisation.

Le catalogue de Cdiscount compte aujourd'hui plus de 7600 catégories, dans lesquelles les produits sont répartis de manière fortement déséquilibrée (80% du catalogue est concentré dans 20% des catégories), rendant la tâche de classification d'autant plus complexe.

Dans un premier temps, nous avons attaqué le problème sous un angle sémantique. Si ces méthodes nous ont permis de traiter une bonne partie du flux d'entrée, il reste cependant une part non négligeable de produits non catégorisés. Le but de ces travaux est d'utiliser les images associées aux produits pour catégoriser ce reliquat. Au cours de ces travaux, nous avons été confrontés à 4 défis majeurs :

- **Très grand nombre de classes** : les données contiennent plus de 7600 catégories, soit près de 8 fois plus que ImageNet.
- **Contraintes de temps et de budget** : le cadre de ces travaux nous a demandé de trouver le meilleur compromis entre les performances, le temps d'apprentissage et le coût de l'infrastructure. De plus, l'arborescence des catégories évolue au cours du temps. Ce qui accentue la contrainte de temps sur l'apprentissage du modèle.
- **Déséquilibre des données** : 80% des données représentent 20% des catégories. Cette répartition doit être prise en compte pour éviter le sur-apprentissage.
- **Données Bruitées** : les données contiennent environ 10% de données mal catégorisées.

Cet article est organisé de la façon suivante : tout d'abord, nous exposons le contexte bibliographique de nos travaux (section 2) ainsi que les données utilisées (section 3). Dans la section 4, nous présentons le modèle utilisé et la méthodologie d'apprentissage. La section 5 est consacrée au protocole d'évaluation et résultats.

2 Travaux Connexes

L'apprentissage profond, en anglais *deep learning*, représente désormais l'état de l'art dans différents problèmes du domaine de la vision par ordinateur. En particulier 2012 a marqué une tournant dans la classification d'images. Le modèle décrit dans Krizhevsky et al. (2012), nommé AlexNet, fut en effet capable de réduire le taux d'erreur d'environ 10% sur le jeu de données du célèbre challenge ImageNet (Russakovsky et al., 2015).

A la base de la plupart des méthodes d'apprentissage profond, on trouve des réseaux de neurones. Un réseau de neurones consiste en plusieurs unités, appelées neurones. Chaque neurone est associé à une fonction de transfert du type :

$$f(\mathbf{x}; \mathbf{w}, \mathbf{b}, \sigma) = \sigma\left(\mathbf{w}^T\mathbf{x} + \mathbf{b}\right) \ , \tag{1}$$

où $\mathbf{x}$ est le vecteur d'entrée, $\mathbf{w}$ et $\mathbf{b}$ sont, respectivement, les vecteurs de poids et de biais, et σ est une fonction non-linéaire, par exemple la fonction sigmoïde, appelée *fonction d'activation*. La fonction de transfert du neurone consiste donc en l'application d'une non-linéarité à une combinaison linéaire des caractéristiques d'entrée. Plusieurs neurones en cascade forment un *perceptron multicouche* ("multilayer perceptron", MLP, en anglais), qui est le réseau de neurones traditionnel. Les couches entre l'entrée et la sortie sont souvent appelées les couches cachées (*hidden layers*). Quand un réseau contient de nombreuses couches cachées, il est typiquement considéré comme "profond", d'où le terme d'apprentissage profond.

Aujourd'hui les architectures de réseaux de neurones profonds les plus communément appliquées aux images sont les réseaux de neurones convolutifs (*convolutions neural networks*, ou CNNs). La différence principale entre les MLPs et les CNNs est que dans les seconds, les poids du réseau sont partagés entre les différents pixels de l'image. Ainsi, l'opération de pondération de l'entrée est faite via une convolution par un même ensemble de noyaux. De cette manière, le nombre de paramètres à déterminer est considérablement réduit, parce qu'il dépend des tailles des noyaux utilisés et non plus de la dimension des données d'entrée. Pour une couche l donnée on définit un ensemble de K noyaux $\mathcal{W} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_K\}$ et K biais relatifs $\mathcal{B} = \{b_1, b_2, \ldots b_K\}$. Pour chaque couple noyau-biais de la couche l, après application d'une certaine fonction d'activation $\sigma(\cdot)$, on obtient donc une image intermédiaire selon l'expression :

$$\mathbf{X}_k^l = \sigma\left(\mathbf{W}_k^{l-1} \star \mathbf{X}^{l-1} + b_k^{l-1}\right) \ . \tag{2}$$

Une autre différence est la présence dans les CNNs d'un nouveau type de couche, nommée couche de *pooling* (mise en commun), qui consiste en l'agrégation de l'information de plusieurs pixels voisins en appliquant une fonction invariante aux permutations (comme la fonction $\max$). Le *pooling* peut être vu comme une forme de sous-échantillonnage de l'image.

AlexNet est un réseau de neurones convolutif relativement peu profond, constitué de 5 couches convolutives, dans lesquelles les supports des noyaux deviennent de plus en plus petits à mesure que la couche relative s'approche de la couche de sortie. La fonction d'activation utilisée dans le réseau est appelée Unité de Rectification Linéaire (ReLU). Suite au succès d'AlexNet, de nouvelles architectures de réseau ont été explorées à partir de 2012, la tendance générale étant d'utiliser plusieurs couches avec des noyaux à petit support au lieu d'une seule couche avec des noyaux à grand support. Les premiers à utiliser des réseaux profonds de ce type, avec de noyaux de taille fixe dans chaque couche, sont Simonyan and Zisserman (2014). Leur modèle, nommé VGG19 et constitué de 19 couches, a remporté le challenge ImageNet en 2014.

Au-delà de l'utilisation d'un grand nombre de couches, de nouveaux blocs ont été introduits au cours des dernières années pour à la fois améliorer l'efficacité de la procédure d'apprentissage et réduire le nombre de paramètres. Szegedy et al. (2015) introduisent un nouveau réseau à 22 couches appelé GoogLeNet ou plus communément Inception. Cette dénomination est liée à l'utilisation par le réseau du module "inception" introduit par Lin et al. (2013). Ce module remplace l'opération définie par l'équation 2 par un ensemble de convolutions de tailles différentes, avec toujours pour objectif de concevoir une fonction qui requiert moins de paramètres. L'architecture ResNet, proposée par (He et al., 2016) et caractérisée par le module éponyme, a remporté le challenge ImageNet de 2015. Le module ResNet, au lieu d'apprendre des fonctions entre entrée et sortie, cherche à caractériser uniquement le signal résidu. Cela permet un apprentissage plus efficace, avec des poids à déterminer plus faibles. Depuis 2016, les performances des CNNs sur les jeux de données de référence comme ImageNet semblent avoir atteint un plateau, et nombre d'architectures proposées au cours des dernières années sont essentiellement des variantes des modèles Inception et ResNet. Dans ce travail nous utilisons une version de l'architecture Inception, appelée *Inception v3* et décrite par Szegedy et al. (2016).

3 Données

Lors de ces travaux, nous avons utilisé le jeu de données que nous avons mis à disposition lors du concours Kaggle [1] que nous avons organisé en 2017 (Bellétoile, en cours). Ce jeu de données se compose de plus 8.8 millions de produits, représentés par plus de 20 millions d'images (chaque produit étant associé à une à quatre images). Nous avons construit nos ensembles de données de la façon suivante :

- Nous ne gardons que les catégories contenant au moins 20 produits.
- Si la catégorie contient moins de 100 produits, nous mettons 10 produits dans le test, et répartissons le reste entre validation et apprentissage.

1. https://www.kaggle.com/c/cdiscount-image-classification-challenge

— Si la catégorie contient plus de 100 produits, nous faisons une répartition 80% - 10% - 10% des données entre apprentissage, validation et test.

Nous obtenons alors les jeux de données présentés dans la Table.1 et quelques exemples d'images tirées aléatoirement sont disponibles dans la Figure.1.

Nom	Images	Catégories
Apprentissage	16M	8K
Validation	2M	8K
Test	2M	8K

TABLE 1 – Tailles des jeux de données utilisés.

4 Modèle et Méthodologie d'Apprentissage

4.1 Modèle

Dans le cadre de cet article, nous avons considéré plusieurs modèles avant de décider d'utiliser inceptionV3 Szegedy et al. (2016). Après comparaison des différents modèles disponibles, c'est inceptionV3 qui semblait présenter le meilleur compromis entre la consommation de ressources, le temps d'apprentissage et les performances.

Nous avons retenu deux approches, la première consiste en un apprentissage total (*full tuning*) du réseau sur les données d'apprentissage. La seconde consiste en l'utilisation d'un réseau pré-entraîné. Dans ce cas, nous extrayons les caractéristiques du *bottleneck* que nous envoyons dans un classifieur plus simple (réseau de neurones ou machine à vecteurs de support).

Dans le cadre applicatif de ces travaux, nous avions une contrainte de temps sur l'apprentissage. En effet, les catégories ne sont pas figées et évoluent au cours du temps. Il était donc important pour nous de pouvoir apprendre notre modèle sur une période d'une semaine maximum. Cette contrainte a fortement influencé nos choix quant à la solution retenue. Comme on peut le voir dans la Table. 3, le *full tuning* est beaucoup trop chronophage pour permettre une mise en production. Tous les temps de calcul ont été estimés sur une machine virtuelle Microsoft Azure NC12 décrite en Table 2.

	Modèle	Cœurs	RAM
CPU	Intel E5-2690v3	12	112 Gio
GPU	Nvidia K80	4992	24 Gio

TABLE 2 – spécifications de la machine virtuelle Microsoft Azure NC 12.

Ainsi, nous proposons l'utilisation du modèle inceptionV3 pré-entraîné sur les données ImageNet (Deng et al., 2009), dont nous avons ôté la couche de sortie pour récupérer les *bottleneck* (*i.e.* la pénultième couche du réseau). Nous passons alors ces vecteurs de caractéristiques dans un classifieur entraîné sur des caractéristiques extraites à partir de nos données. Nous avons comparé les performances d'un réseau de neurones à deux couches cachées et d'un SVM.

Dans un second temps, nous nous sommes intéressés à l'impact d'un ré-apprentissage (*fine-tuning*) du modèle sur 5% des données d'apprentissage.

Pour évaluer nos approches, nous avons aussi utilisé un modèle plus traditionnel. Plus précisément, nous avons extrait des marqueurs HOG auxquelles nous avons appliqué un SVM. Les résultats de chacun de ces modèles sont disponibles dans la section 5.2.

Mode	Epoch	20M10	20M50
full tuning	33000	70 jours	350 jours
Bottlenecks 2NN	500	27,7 heures	5,8 jours

TABLE 3 – Récapitulatif des temps d'apprentissage pour le *full training* et l'utilisation des *bottlenecks* avec un réseau de neurones à deux couches cachées (2NN). Epoch correspond au temps en secondes nécessaire à chaque passage sur l'intégralité du jeu d'apprentissage. 20M10 correspond au temps nécessaire à 10 exécutions de la routine d'apprentissage sur les 20 millions d'images. 20M50 correspond au temps nécessaire à 50 exécutions de la routine d'apprentissage sur les 20 millions d'images.

4.2 Apprentissage des *Bottlenecks*

Pour chaque itération, nous avons fixé la taille des lots à 256 images. Nous avons alors entraîné notre modèle par descente de gradient stochastique avec la librairie Keras (Chollet et al., 2015) et défini un pas d'apprentissage dégressif suivant la fonction disponible en Équation.3 et illustrée dans la Figure. 2. Nous avons fixé empiriquement les variables lr et $decay$ comme il suit : $lr(0) = 0.1$ et $decay = 7 * 10^5$

$$lr(t) = lr(0)\frac{1}{1 + t * decay} \tag{3}$$

Pour accélérer l'apprentissage, nous avons implémenté une normalisation des lots (*batch normalization*), comme proposée dans (Ioffe and Szegedy, 2015) . Cette normalisation prend la forme d'une couche ajoutée après le calcul des logits, permettant de les normaliser suivant une distribution gaussienne. Comme on peut le voir dans la Figure. 3, si l'ajout d'une couche de normalisation permet en effet d'accélérer l'apprentissage, il ne sert à rien de les multiplier.

4.3 *Fine-Tuning*

Pour chaque itération, nous avons fixé la taille des lots à 128 images. Nous avons alors entraîné notre modèle par descente de gradient stochastique avec un pas d'apprentissage dégressif (cf. Équation 3 et Figure 2) via Keras (Chollet et al., 2015). Nous avons fixé empiriquement les variables lr et $decay$ comme il suit : $lr(0) = 0.05$ et $decay = 10^4$.

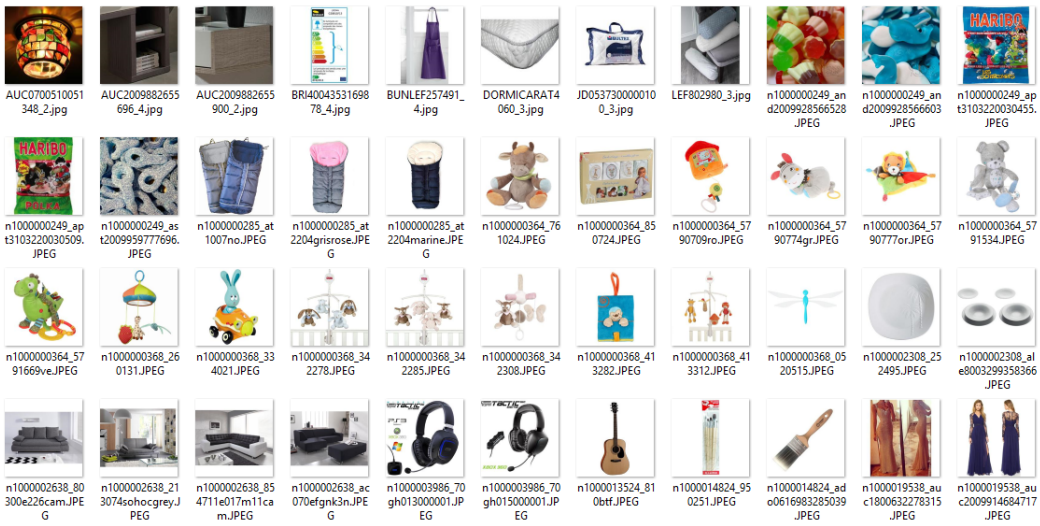

FIGURE 1 – Exemples d'images tirées aléatoirement dans le jeu de données d'apprentissage

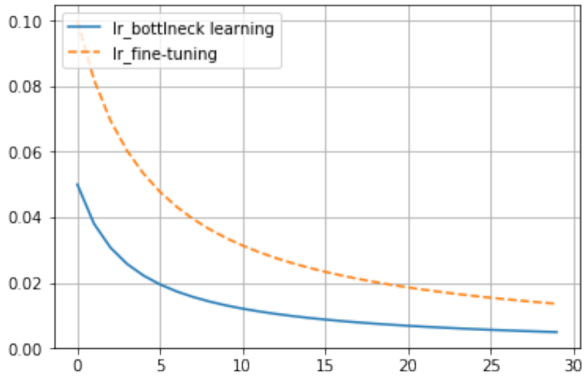

FIGURE 2 – Evolution du pas d'apprentissage au cours du temps pour l'apprentissage des *bottlenecks* (bleu) et le *fine-tuning* (orange)

4.4 *Gestion du Sur-Apprentissage*

Nous avons implémenté deux méthodes pour limiter le sur-apprentissage :

- Nous avons décidé de fixer les poids des premières couches (40%) du modèle durant le *fine-tuning*. Comme expliqué par Yosinski et al. (2014), cette approche ne pénalise pas le modèle, les premières couches étant les plus facilement transférables.
- Nous avons implémenté un *dropout* (désactivation aléatoire de certains neurones à chaque itération), comme préconisé par Krizhevsky et al. (2012). Nous l'avons intégré directement après la première couche cachée.

Comme on le voit dans la Figure 4, l'ajout du *dropout* à notre méthodologie d'apprentissage a permis d'efficacement limiter le sur-apprentissage.

5 Expériences et Résultats

5.1 Protocole d'Évaluation

Dans le cadre d'une application industrielle, nous avions besoin de pouvoir garantir une précision minimale de 0.9. Nous nous retrouvons donc avec une implémentation de seuils de confiance dans notre modèle, ce que nous avons du prendre en compte dans l'évaluation.
Soient Ω l'ensemble des n produits à évaluer, et $k \in \Omega$ un produit. Soient $cat_k, \bar{cat}_k$ respectivement, la catégorie de k (vérité terrain), la catégorie prédite pour k, on définit la précision $prec(\Omega)$ de la manière suivante :

$$\begin{aligned} \mathbb{1}(k) &= \begin{cases} 1 & \text{si } cat = \bar{cat} \\ 0 & \text{sinon} \end{cases} \\ prec(\Omega) &= \frac{1}{n} \sum_{k=1}^{n} \mathbb{1}(k) \end{aligned} \tag{4}$$

On définit alors le taux d'acceptation à $n \in [0, 1]$, noté acc_n tel que suit :

$$\begin{aligned} acc_n(\Omega) &= max \left(\frac{Card(\omega)}{Card(\Omega)} \ , \ \omega \subset \Omega \right) \\ &\text{avec } prec(\omega) \geq n \end{aligned} \tag{5}$$

Dans cette partie, nous présentons nos résultats et analyses. Pour des raisons de confidentialité, nous devons présenter des résultats partiels, ainsi ils ne sont dévoilés que sur 1000 catégories tirées aléatoirement.

5.2 Optimisation des Meta-Paramètres

Pour évaluer nos modèles, nous les comparons à un modèle simple, basé sur l'extraction de caractéristiques HOG (Dalal and Triggs, 2005), jointe à une catégorisation par SVM.
Nous avons comparés différents modèles et initialisations comme décrit en Table. 4. Nous avons considéré les axes de variation suivants :

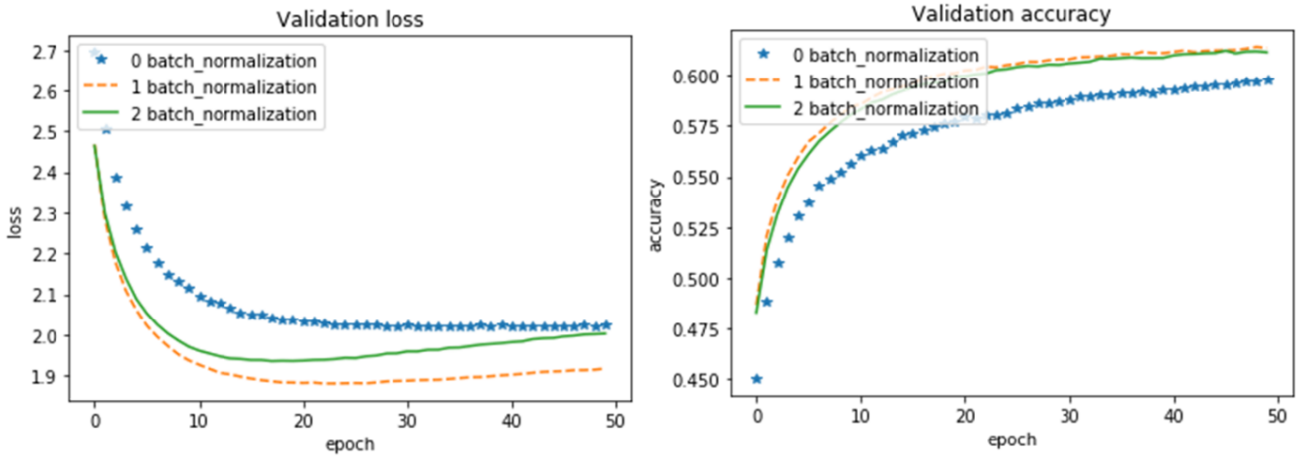

FIGURE 3 – Impact de la *batch normalization* sur la fonction de coût (à gauche) et la précision (à droite) en validation.

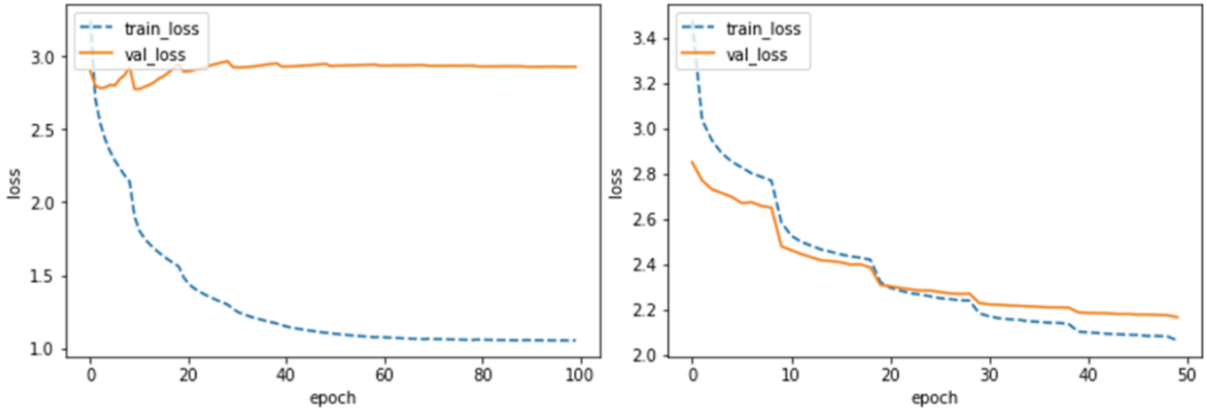

FIGURE 4 – Impact du *dropout* sur le sur-apprentissage : évolution des fonctions de coût en apprentissage et validation avec (à droite) et sans (à gauche) *dropout*.

— *Fine-tuning* du modèle sur les données Cdiscount.
— Classifieur : SVM ou réseau de neurones à deux couches cachées.
— Nombre de neurones du réseau : 512, 2048 ou 4096.

Feature	Classifieur	Fine Tuning	Préc. App.	Préc. Val.
HOG	SVM	Non	0.266	0.1
Bottlenecks	SVM	Non	0.608	0.533
Bottlenecks	2NN - 512	Non	0.544	0.551
Bottlenecks	2NN - 4096	Non	0.716	0.598
Bottlenecks	2NN - 4096	Oui	0.725	0.614
Bottlenecks	2NN - 2048	Oui	0.773	0.726

TABLE 4 – Résultat obtenus en validation lors du choix des méta-paramètres (exprimés en précision).

Aux vues de ces résultats, nous avons donc choisi le setup suivant : *Bottlenecks*, réseau de neurones à deux couches de 1024 neurones chacune (2048 neurones au total) et *fine-tuning*.

On remarque que l'augmentation du nombre de neurones finit par pénaliser le modèle. Ceci peut venir d'une trop forte spécialisation de certaines régions du réseau, pénalisant ainsi la prédiction générale.

5.3 Résultats

Dans cette partie, nous évaluons les performances du modèle utilisant les *Bottlenecks* un réseau de neurones à deux couches de 1024 neurones chacune (2048 neurones au total) et du *fine-tuning* dans des conditions iso-production. Comme nous l'avons expliqué précédemment, pour des questions de confidentialité, nous présentons nos résultats sur un sous-ensemble de 1000 catégories tirées aléatoirement.

Dans la Figure. 5, sont présentés les résultats en terme de taux d'acceptation fonction de la précision. Comme vu dans la partie 5.1, le taux d'acceptation à n correspond au taux de produits catégorisés, sachant que l'on accepte

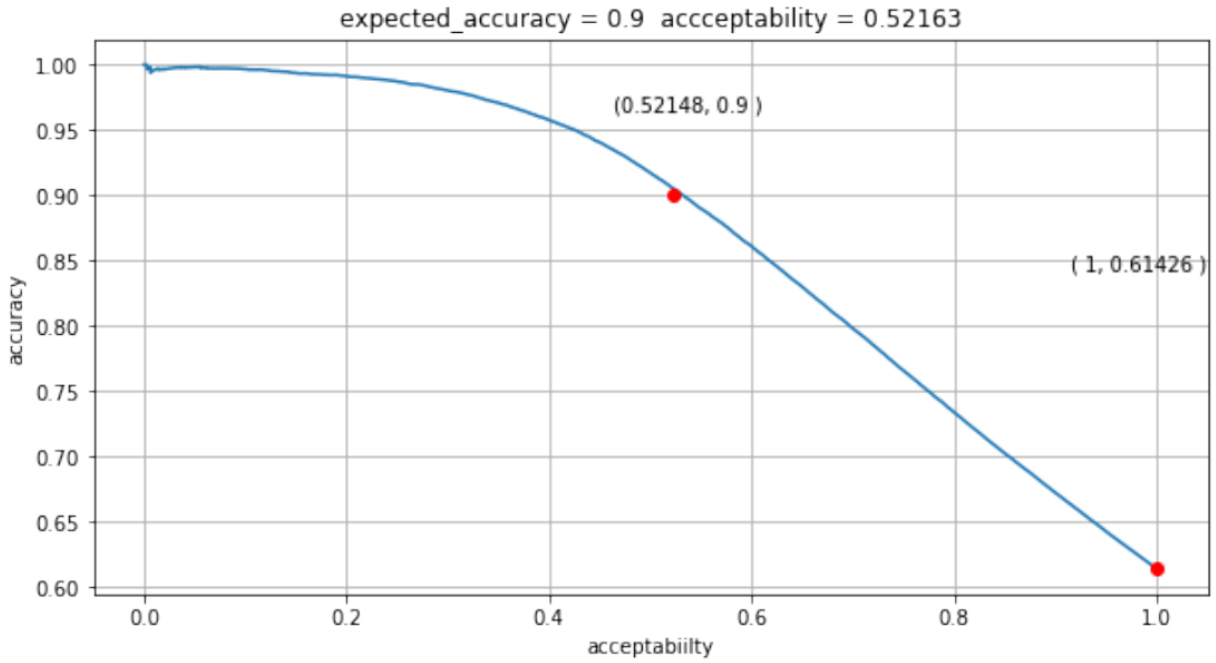

FIGURE 5 – Résultats en terme de *confident top1 accuracy* sur 1000 catégories. Le premier point rouge correspond au taux d'acceptation pour une précision de 90% et le second correspond à la précision pour un taux d'acceptation de 1.

une précision minimale de n. Pour un taux d'acceptation de 100%, on obtient une précision de 61.4%, et pour un taux de précision de 90% (notre contrainte de production) on obtient un taux d'acceptation de 51.5%.
Dans un second temps, nous avons changé la méthode d'agrégation : au lieu de calculer notre précision cumulative au produit, nous l'avons calculée à la catégorie. Les résultats, présentés dans la Figure. 7 montrent qu'en restreignant notre modèle à 80% des catégories, nous pouvons enregistrer un gain de 10.6 points en précision et de 15 points en taux d'acceptation à 0.9. Le choix des catégories s'effectue sur la catégorie prédite, une sélection sur la catégorie réelle n'ayant aucun sens en production.

5.4 Analyse de la Typologie des Produits

Lors d'une analyse en détail du taux de précision par catégorie, nous nous sommes rendus compte que 80% des erreurs provenaient de la librairie. Si notre modèle distingue bien les livres des autres produits, il n'est pas en mesure de les classer efficacement dans les bonnes catégories (par exemple littérature étrangère, livre de cours, livre de cuisine). Le reste des erreurs s'explique entre autres par certaines catégories contenant des produits visuellement trop variés, ou trop proches d'autre catégories, comme les jouets d'éveil, qui se rapprochent des poupées et des peluches (cf. Figure 6). Nous avons pu confirmer cette théorie en nous intéressant aux catégories les mieux traitées. On y retrouve des produits avec une faible variété visuelle, comme le gros électroménager.

6 Conclusion

Dans ces travaux, nous avons proposé une application industrielle aux méthodes de classification d'images par apprentissage profond. Nous avons su prendre en compte les contraintes de budget et de temps d'apprentissage et proposer un modèle capable de passer à l'échelle sur plus de 7600 catégories sans pour autant prendre plus d'une semaine pour l'apprentissage. Le modèle proposé présente des résultats prometteurs, notamment sur les catégories présentant une variabilité visuelle raisonnable (par exemple électroménager, mobilier, pièces auto).

Références

Arnaud Bellétoile. Learning to classify e-commerce products from their images. en cours.

François Chollet et al. Keras. `https://keras.io`, 2015.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2372-2. doi : 10.1109/CVPR.2005.177. URL `http://dx.doi.org/10.1109/CVPR.2005.177`.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei.

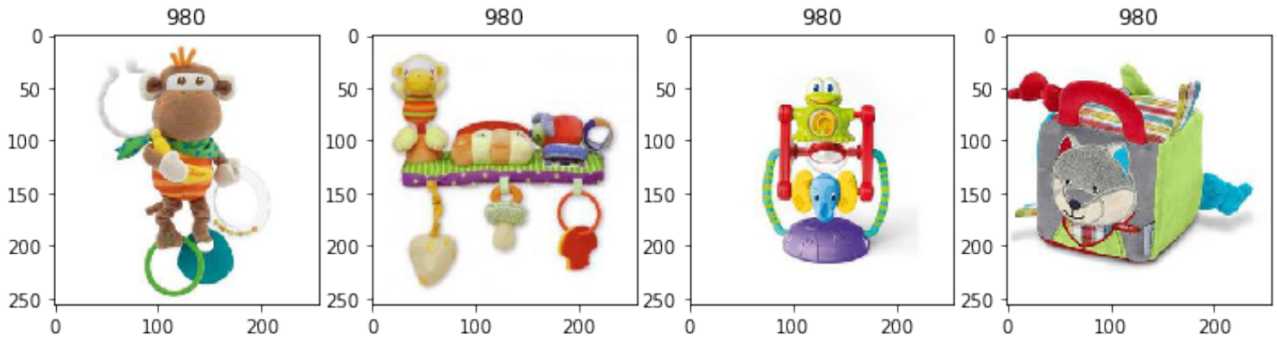

FIGURE 6 – Exemples de produits ambigus. Les jeux d'éveil sont visuellement proches des peluches.

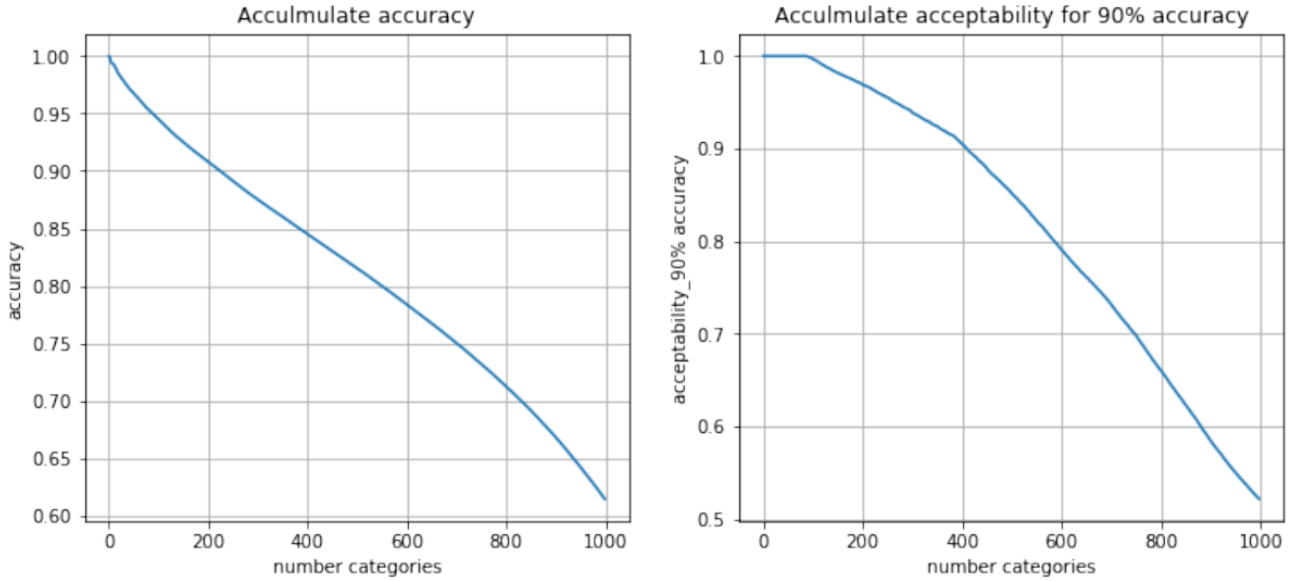

FIGURE 7 – A gauche, la précision cumulée par catégorie, et à droite le taux d'acceptation cumulée par catégorie, pour 90% de précision.

ImageNet : A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv :1312.4400*, 2013.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3) :211–252, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru

Erhan, Vincent Vanhoucke, Andrew Rabinovich, Jen-Hao Rick Chang, et al. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826, 2016. doi : 10.1109/CVPR.2016.308.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks ? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3320–3328, Cambridge, MA, USA, 2014. MIT Press.