Reconnaissance d'Images de Jeux de Seconde Main

Projet de Fin d'Etudes

Corentin Seutin
Iban Oyharcabal
Anessa Diallo

Mars 2025

GitHub du projet : `https://github.com/Argitoon/pfe`
GitHub remis au client : `https://github.com/Argitoon/Cajou`
Etablissement / Formation : Université de Bordeaux - Master 2 informatique
Client : Baptiste Hasbrouk (Deuzio)
Encadrante : Marie Beurton-Aimar

Table des matières

Introduction

La classification automatique d'images est une tâche cruciale dans de nombreux domaines. Elle permet en outre d'attribuer une catégorie à une image en fonction de ses caractéristiques visuelles. Cette tâche est notamment réalisée à l'aide de réseaux de neurones profonds (DNNs) et plus précisemment à l'aide des réseaux de neurones convolutifs (CNNs). Cela est dû au fait qu'un CNN peut apprendre et sélectionner de manière efficace des caractéristiques visuelles. Les avancées technologiques des CNNs ont permis de développer des architectures de plus en plus complexes et performantes pour la classification d'images. L'utilisation des CNNs se généralise de plus en plus et trouve des applications dans de nombreux domaines, notamment en médecine[8], en particulier pour le diagnostic médical[12][20], mais aussi dans l'industrie automobile pour la conduite autonome[1] ou encore pour la reconnaissance d'espèces animales[4]. Dans le domaine de l'industrie, l'utilisation des CNNs pour la classification d'objets permet de réaliser des tâches de tri automatique et représente un gain de temps, d'argent et évite une classification manuelle redondante et laborieuse. Cette application permet à l'utilisateur de classer automatiquement à l'aide d'un CNN des images de jouets soit à partir des fichiers de l'ordinateur utilisé soit en prenant en temps réel une photo via la webcam de celui-ci. Le rapport est structuré comme suit. L'analyse du sujet est présenté dans la première partie. La deuxième partie introduit la conception du projet. La réalisation est décrite en troisième partie. La quatrième partie conclut le rapport.

Analyse

1 Contexte

Dans le cadre de l'Unité d'Enseignement Projets de fin d'études, l'entreprise **deuzio** nous a demandé de réaliser une application de reconnaissance d'images de jeux de seconde main. L'automatisation de cette tâche permet notamment de classifier rapidement et efficacement les images de jouets et de les trier en fonction de leur catégorie. Pour ce projet, le client nous a fourni une liste de catégories de jouets à classifier, qui sont les suivantes : les véhicules, comprenant les petites voitures, les trottinettes ou autre véhicules jouets ; les jouets de la marque **Playmobil** ; les poupées ; les jeux d'imitation, c'est-à-dire des jouets visant à imiter des actions ou métiers courants de la vie (par exemple, une cuisine, un aspirateur ou encore une caisse à outils) ; et enfin, les jeux d'éveil, destinés aux enfants en bas âge. L'application, que nous avons nommée **Cajou** (pour Classification Automatique de Jouets), doit afficher la catégorie de jouet à laquelle appartient l'image fournie et être simple d'utilisation. Aussi, l'utilisateur doit pouvoir corriger la catégorie prédite par le modèle si celle-ci est incorrecte. La catégorie corrigée doit être enregistrée dans le cloud Dropbox, selon les demandes du client et l'utilisateur doit pouvoir ré-entraîner le modèle en utilisant les images du cloud. Enfin, le client souhaite pouvoir ajouter facilement de nouvelles catégories de jouets à classifier.

Pour répondre à ces besoins, nous avons décidé d'utiliser des réseaux de neurones convolutifs (CNNs) pour la classification des images. Les CNNs comme le mentionnent Yadav et al.[20] sont des modèles de réseaux de neurones qui ont été conçus pour traiter des données structurées en grille comme des images. Les DNNs prodiguent en moyenne une meilleure précision ou **accuracy** que les méthodes plus traditionnelles comme Oriented FAST and Rotated BRIEF (ORB) ou Support Vector Machine (SVM). Ils ont permis d'obtenir de nombreux résultats prometteurs et de nos jours très utilisés comme la famille de modèles Residual Networks proposée par He et al[5].

2 Dataset

Dans le cadre de ce projet, nous avons utilisé des images fournies par **deuzio**. La quantité d'images étant limitée ($\sim$ 30 images par catégorie), nous avons téléchargé manuellement d'autres images issues de différentes sources web. Notre usage des images se limite strictement au contexte académique de ce projet, en conformité avec les exigences de notre université. Nous avons aussi utilisé des images issues du dataset Toy Cars annotated on YOLO format[13] proposé par Tuba Siddiqui. Dans l'optique d'une exploitation commerciale du modèle, nous recommandons à l'entreprise de constituer son propre dataset en collectant des images sous licences adaptées ou en générant ses propres données.

Le manque de données pour l'entraînement du modèle est un problème majeur qui peut entraîner un surapprentissage et, par conséquent, une baisse de performance. Pour pallier ce problème, nous avons suggéré au client d'implémenter un apprentissage supervisé permettant à l'utilisateur de corriger les prédictions du modèle et ainsi d'augmenter la quantité de données d'entraînement à chaque utilisation de l'application.

3 Modèles

La convolution est une opération mathématique utilisée en traitement d'image. Elle permet d'appliquer un filtre à une image. Ce filtre, représenté traditionnellement par une matrice de taille $n \times m$, est appliqué en effectuant un produit scalaire entre les valeurs de l'image et celles du filtre. Cette opération permet de détecter des caractéristiques spécifiques telles que les contours, les textures ou encore les variations d'intensité en prenant en compte les valeurs des pixels voisins. E.g. le filtre de Sobel est utilisé pour la détection des contours. Il est défini comme suit :

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Lorsque ce filtre est appliqué à une image en niveaux de gris, il met en évidence les variations d'intensités horizontales, permettant ainsi de détecter les contours verticaux. De la même manière, un filtre de Sobel vertical peut être utilisé pour détecter les contours horizontaux. Il est important de noter que les convolutions spatiales classiques en traitement d'image et celles utilisées dans les CNNs diffèrent. En effet, en traitement d'image traditionnel, la convolution est généralement appliquée sur un unique canal, de couleur ou de niveaux de gris. En revanche, dans les CNNs, la convolution est effectuée sur plusieurs canaux, et les filtres utilisés sont optimisés automatiquement par l'apprentissage du réseau de neurones pour extraire les caractéristiques des images. Les CNNs utilisent ainsi des convolutions 3D et permettent une détection plus riche et adaptative des motifs

dans une image. Il existe de nombreux modèles CNNs pour la classification d'images. Pour notre projet, par gain de temps et de performances nous avons choisi de ne pas être exhaustifs dans le choix des modèles testés et nous nous sommes ainsi limités aux modèles disponibles depuis certaines bibliothèques Python comme **PyTorch** et **PyTorch Image Models (timm)** qui regroupent un grand nombre de modèles, ainsi qu'un modèle de la famille des Residual Networks[5] que nous avons implémenté nous-même. Le projet a donc été réalisé en utilisant le langage Python. Les modèles utilisés, à l'exception du nôtre, sont tous pré-entraînés, testés et possèdent un nombre de sorties différent et des catégories différentes de ce dont nous avons besoin pour la classification d'images de jeux de seconde main. Ceci implique d'utiliser dans le cadre du projet la technique d'apprentissage par transfert ou **transfert learning**. Cette technique consiste, appliquée à notre utilisation, à adapter à l'aide d'une couche entièrement connectée ou **fully connected** (FC) les sorties d'un modèle au nombre de sorties dont nous avons besoin, i.e. le nombre de catégories demandées par le client. Les points suivants détaillent l'ensemble des modèles utilisés dans le cadre de ce projet.

◆ Inception Networks

François Chollet propose dans son article intitulé *Xception : Deep learning with depthwise separable convolutions*[2] le modèle Xception. L'architecture de ce modèle a été inspirée par le module Inception de l'architecture Inception V3[14] illustré sur la Figure 1.

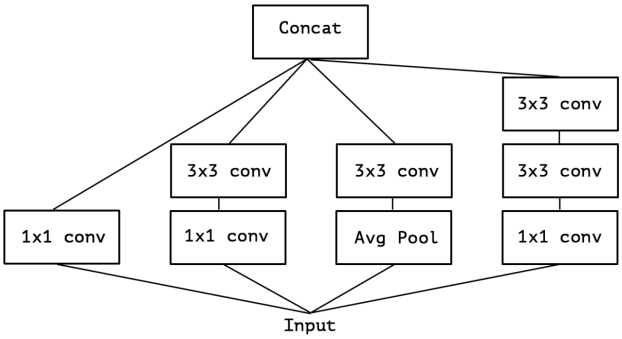

FIGURE 1 – Module Inception, source : [2].

En effet, l'auteur propse dans un premier temps de simplifier ce module en séparant les convolutions en deux étapes distinctes, i.e. une étape comportant une convolution 1x1 dite *pointwise* et une autre comportant une convolution 3x3 dite *depthwise* puis de réduire le nombre de convolution 1x1 à une seule comme illustré sur la Figure 2. L'auteur propose alors de généraliser l'idée de séparer les convolutions sur chaque canal de manière à séparer le traitement spatial (*depthwise*) et le traitement des canaux (*pointwise*) ce qui constitue le module Separable Convolution [1] comme nous pouvons le constater sur la Figure 2.

1. Le module final effectue d'abord les convolutions spatiales puis la convolution inter-canaux contrairement à ce qui est montré sur les différentes figures.

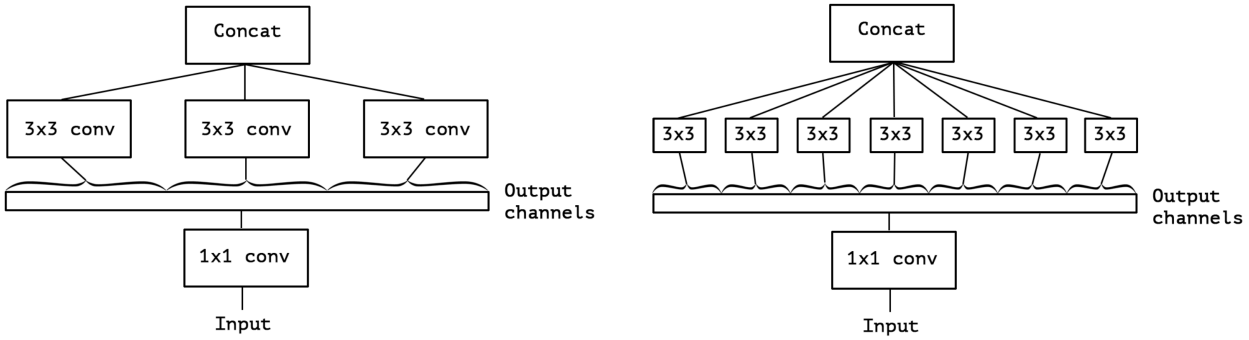

FIGURE 2 – De gauche à droite : module Inception simplifié, module Inception modifié avec une convolution par canal, source : [2].

La Figure 3 détaille l'architecture du modèle Xception utilisant ce type de module qui reste, en dehors du module proposé dans l'article, classique.

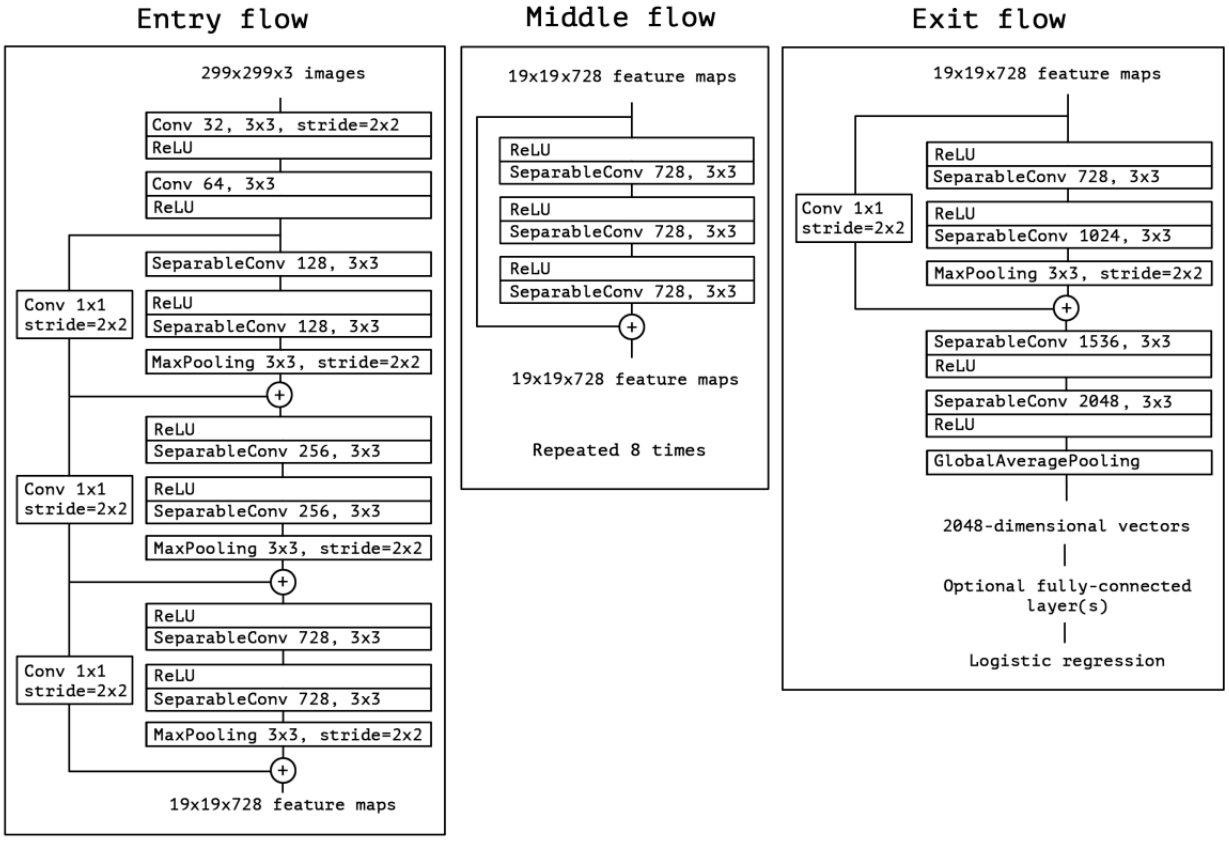

FIGURE 3 – Architecture du modèle Xception, source : [2].

◆ Dense Networks

Huang et al.[7] proposent dans leur article le bloc de convolution **Dense Block** qui permet de

connecter toutes les couches entre elles au sein d'un même bloc comme illustré sur la Figure 4.

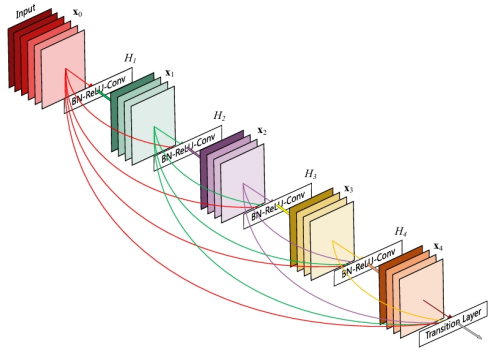

FIGURE 4 – Dense Block, source : [7]

Cette connexion dense permet de faciliter la propagation du gradient et en outre de réduire l'impact du problème de *vanishing gradient*[2]. Les auteurs introduisent aussi le modèle DenseNet-121 qui est composé de 121 couches utilisant ce type de bloc.

◆ Efficient Networks

La famille des modèles Efficient Networks[16] est une famille de modèles qui a été conçu pour être plus performante que les modèles existants en terme de précision et de temps d'entraînement notamment en utilisant l'architecture Mobile Neural Architecture Search (MNAS)[15]. Cette architecture permet en outre de chercher de manière automatique les architectures de réseaux de neurones ayant le meilleur compromis *accuracy* - latence. Le modèle EfficientNet-B0 est le plus petit modèle de la famille des Efficient Networks. Le Tableau 1 illustre l'architecture de ce modèle.

Stage i	**Operator** $\hat{F}_i$	**Resolution** $\hat{H}_i \times \hat{W}_i$	**#Channels** $\hat{C}_i$	**#Layers** $\hat{L}_i$
1	Conv3×3	224 × 224	32	1
2	MBConv1, k3×3	112 × 112	16	1
3	MBConv6, k3×3	112 × 112	24	2
4	MBConv6, k5×5	56 × 56	40	2
5	MBConv6, k3×3	28 × 28	80	3
6	MBConv6, k5×5	14 × 14	112	3
7	MBConv6, k5×5	14 × 14	192	4
8	MBConv6, k3×3	7 × 7	320	1
9	Conv1×1 & Pooling & FC	7 × 7	1280	1

TABLE 1 – Architecture du modèle EfficientNet-B0, source : [16].

◆ Convolutional Neural Networks et Transformers

2. Les gradients des poids des couches les plus proches des données d'entrées sont exponentiellement plus petits que les gradients des poids des couches les plus proches des sorties. Cela est dû à la nature multiplicative de la valeur des gradients lors de la rétropropagation[18].

ConvNeXt[10] est un modèle qui combine la simplicité et l'efficacité des modèles de la famille des Convolutional Neural Networks (ConvNets) avec la précision des Transformers (ViTs)[17] et notamment des Swin Transformers[9]. Ceci peut être fait en modifiant le bloc caractéristique des Residual Networks en reprenant certaines opérations utilisées pour le bloc d'un Swin Transformer et en ajoutant une convolution *depthwise* comme montré sur la Figure 5.

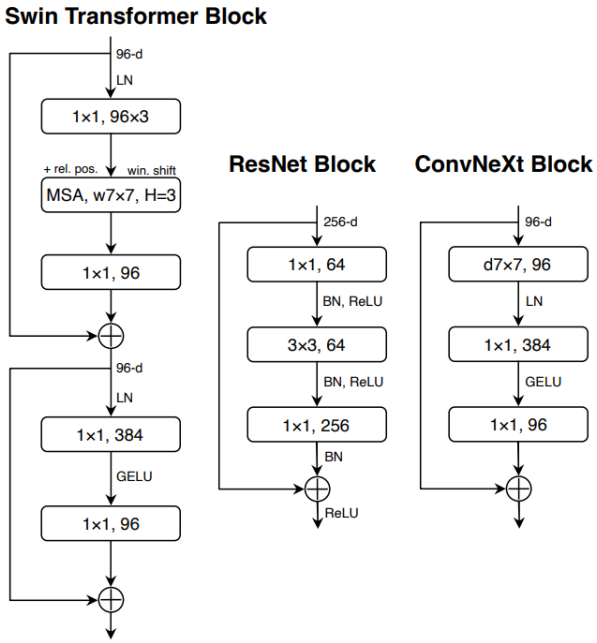

FIGURE 5 – Design des blocs Swin Transformer, ResNet et ConvNeXt, source : [10].

Ces modifications permettent en outre d'obtenir un modèle plus performant que les modèles de la famille des Residual Networks et des Swin Transformers.

◆ Residual Networks

Les Residual Networks[5] est une famille de modèles de réseau de neurones convolutifs qui a été introduit par He et al. en 2015. Les modèles de cette famille sont composés de blocs résiduels (Figure 6) qui introduisent le concept de *skip connection* ou connexion résiduelle.

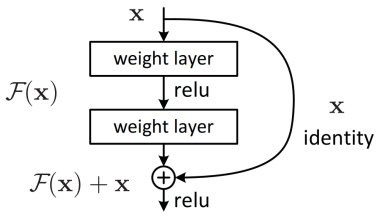

FIGURE 6 – Block résiduel, source : [5]

Cette connexion permet de passer les données d'entrée directement à la sortie du bloc. Elle permet aussi de palier aux problèmes de *vanishing gradient* et de dégradation énoncés par les auteurs.

Le modèle ResNet-50 est donc un modèle de la famille des Residual Networks qui est composé de 50 couches convolutives utilisant des connexions résiduelles. En raison de sa simplicité et de sa performance, nous avons décidé d'implémenter un modèle de la famille des Residual Networks. Ce modèle est composé de quatre couches (blocs Basic, Figure 7).

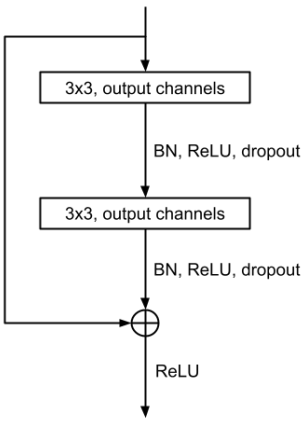

FIGURE 7 – Bloc Basic utilisé pour le modèle implémenté ResNet.

Chaque couche est composée de deux convolutions 3x3 combinées à des normalisations de *batch* (BN), des activations ReLU, des *dropouts* et une connexion résiduelle. La Figure 8 présente le modèle décrit.

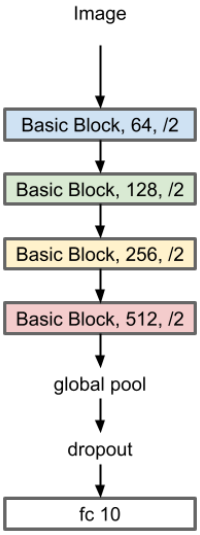

FIGURE 8 – Modèle ResNet proposé. Le modèle est composé de quatre blocs Basic, d'un *global pooling*, d'un *dropout* et d'une couche *fully connected*.

Des *batch normalizations* et des *dropouts* sont présents entre chaque couche pour améliorer les performances du modèle, notamment en permettant de traiter en partie le problème de généralisation que nous avons rencontré lors de ce projet et qui sera défini plus en détail dans les prochaines parties. De la même manière que pour le modèle EfficientNet-B0, Radosavovic et al.[11] ont proposé une méthode de recherche automatique de familles de réseaux de neurones en définissant un **Design Space**, i.e. un espace de recherche. Ils ont ainsi introduit le **design space** AnyNet définit par trois blocs distincts, i.e. le bloc **stem**, le bloc **body** et le bloc **head** comme montré sur la Figure 9.

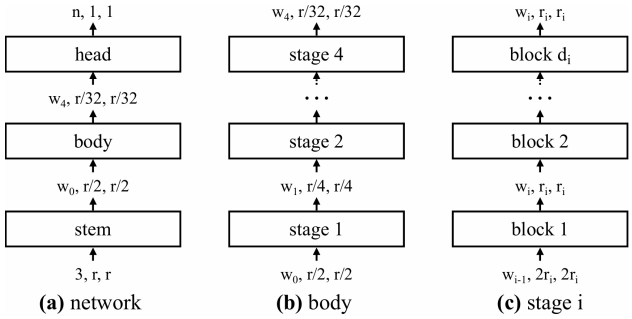

FIGURE 9 – AnyNet **design space**, source : [11].

Comme énoncé par les auteurs, ce **design space** est structurellement simple mais est vaste en terme de nombre de modèles possibles. Pour les blocs, des blocs X, comme illustré sur la Figure 10, sont utilisés.

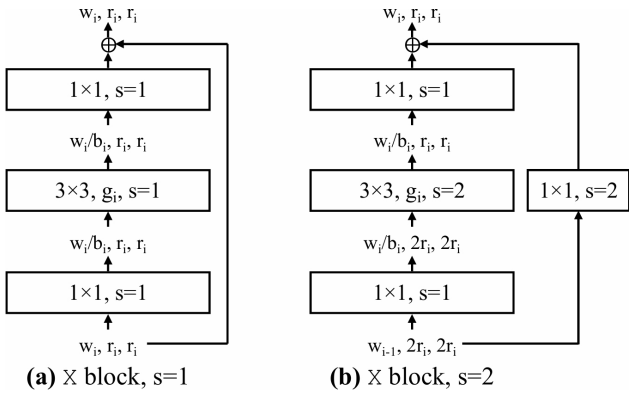

FIGURE 10 – Bloc X **design space**, source : [11].

Ces blocs constituent des gouleaux d'étranglement résiduels avec convolution de groupe[19] et permettent d'améliorer les performances d'un modèle par rapport aux méthodes classiques qui se limitent à augmenter la profondeur et la largeur des réseaux de neurones comme l'évoquent Xie et al[19]. Ces blocs possèdent trois paramètres modifiables dont la largeur w_i, le ratio du gouleau d'étranglement b_i et la largeur du groupe g_i. Les auteurs ont ainsi pu affiner le **design space** AnyNet en analysant les résultats obtenus pour chaque combinaison de ces paramètres et notamment en conservant la simplicité et la qualité de la structure du **design space**. Ce **design space** affiné a été nommé **Regular Network** et se caractérise donc comme étant une famille de modèles. En outre les auteurs ont pu à nouveau affiner ce **design space** grâce à une attention particulière portée sur

les performances, notamment au niveau de l'**accuracy** et de la complexité[3]. La nouvelle famille émergente est appelée Regular Networks X. En ajoutant des blocs Squeeze-and-Excitation (SE)[6], servant à améliorer les performances en introduisant des interdépendances entre les canaux, les performances de cette famille peuvent être améliorées, ce qui constitue la famille variante Regular Networks Y. Ainsi, le modèle RegNetY-400MF que nous utiliserons par la suite est un modèle de cette famille qui possède une complexité de 400 millions de FLOPs.

4 Choix logiciels

Pour implémenter notre projet, nous avons utilisé le langage de programmation Python et les bibliothèques **PyTorch** et **timm** pour les modèles de classification d'images. Ces bibliothèques permettent d'implémenter simplement et intuitivement des modèles de réseaux de neurones convolutifs. Pour entraîner et tester les modèles choisis nous avons décidé d'utiliser les bibliothèques **scikit-learn**, **tqdm**, **Pillow**, **torchvision**, **matplotlib**, **NumPy** et **pandas**. Pour simplifier l'implémentation de l'application nous avons décidé de l'implémenter aussi en Python et notamment à l'aide des bibliothèques **PyQt6** et **OpenCV** pour l'interface et l'utilisation de la webcam en temps réel et de la bibliothèque **Dropbox** pour la gestion des fichiers et des images en ligne. Ce dernier choix a été apporté par le client. En effet, de nombreuses autres solutions alternatives existent pour le stockage de données en ligne comme l'**API Google Drive** ou la bibliothèque **AWS SDK for Python (Boto3)** (**Amazon S3**).

Conception

5 Choix du modèle

◆ Prétraitement des images

Les classes d'images que nous avons utilisées pour entraîner notre modèle n'étant pas exclusives, i.e. une image peut contenir plusieurs classes comme montré sur la Figure 11, nous avons dû apporter une attention particulière à la bonne généralisation du modèle.

3. Les auteurs mesurent la complexité en Floating Point Operations per Second (FLOPs). Cette complexité représente la complexité computationnelle d'un modèle.

FIGURE 11 – Photo d'un véhicule **Playmobil**, source : **Pixabay**

Aussi, le fait que certaines des classes soient très "larges" comme les véhicules, les poupées, les jeux d'imitation ou encore les jeux d'éveil rend la tâche de classification et notamment l'extraction de features en commun plus complexe. Pour limiter ce problème, nous avons décidé d'effectuer des transformations aléatoires sur les images comme le zoom, le retournement horizontal, la rotation, etc. Ces transformations, faites à partir de la bibliothèque **torchvision**, permettent de générer des images uniques et à la volée lors de l'entraînement à partir des images originales. Cela permet d'améliorer en conséquence la généralisation des modèles. Nous avons aussi décidé d'effectuer une augmentation des données en appliquant certaines transformations aux images d'origine que nous sauvegardons dans un cache. Ces transformations permettent notamment d'appliquant un filtre Gaussien, de tourner une image à 45°, de diminuer et augmenter la luminosité ou encore diminuer ou augmenter le contraste à l'aide de la bilbiothèque **Pillow**. L'augmentation de données permet d'augmenter de manière conséquente la taille du dataset de sept fois sa taille d'origine, i.e. du nombre de transformations qui sont utiliées pour l'augmentation, et donc d'améliorer grandement la généralisation du modèle. En effet et comme nous pouvons le constater sur le Tableau 2, l'utilisation de ces deux méthodes permettent d'améliorer l'*accuracy* du modèle RegNetY-400MF de près de 20%.

Modèle	**Sans augmentation**	**Avec augmentation**
RegNetY-400MF	0.49	0.67

TABLE 2 – Comparaison de l'*accuracy* du modèle RegNetY-400MF avec et sans augmentation de données et transformations à la volée pour environ 100 images par classe.

◆ Evaluation

Le Tableau 3 présente la définition de la matrice de confusion.

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

TABLE 3 – Matrice de confusion

La matrice de confusion est notamment utilisée pour calculer l'*accuracy* du modèle qui sert de métrique d'évaluation. L'accuracy est calculée comme suit :

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Nous calculerons aussi la probabilité moyenne de la vraie classe ou Mean True Class Probability (Mean TCP)[3] qui est calculée comme suit :

$$\text{Mean TCP} = \frac{1}{N} \sum_{i=1}^{N} P(Y = y_i^* \mid x_i)$$

Avec Y la sortie du modèle, y_i^* la vraie classe, N le nombre d'images et x_i les entrées du modèle. Cette métrique d'évaluation, différente de l'*accuracy*, permet de mesurer la confiance du modèle dans ses prédictions.

◆ Matériel

Le matériel a son importance, i.e. il est nécessaire de prendre en compte le fait que le client ne possède pas nécessairement un matériel récent et donc performant notamment pour l'entraînement des modèles présentés plus haut. Nous prendrons comme modèle de référence le modèle RegNetY-400MF. En effet ce modèle est le plus performant des modèles présentés dans l'état de l'art comme nous le verrons plus bas. Le matériel présenté sur le Tableau 4 permet d'entraîner ce modèle sur le GPU en un temps raisonnable, i.e. en 38 minutes pour 51 *epochs* sur un dataset d'environ 100 images par classe et en environ deux heures lorsque le modèle est entraîné sur le CPU.

Matériel	**Paramètres**
CPU	12th Gen Intel(R) Core(TM) i7-12700F
RAM	32 G
GPU	NVIDIA GeForce GTX 1660 SUPER
Language de programmation	Python 3.12.0
Deep Learning Framework	PyTorch 2.4.1
CUDA	11.8

Table 4

◆ Comparaison des modèles

Le nombre d'*epochs* que nous avons défini est 51 car le modèle ResNet-50 commence à *overfit* à partir de ce nombre-là comme nous pouvons le constater sur la Figure 12. C'est en effet à partir de ce modèle que nous avons commencé à étudier le problème et nous nous sommes donc basés sur les résultats obtenus pour celui-ci.

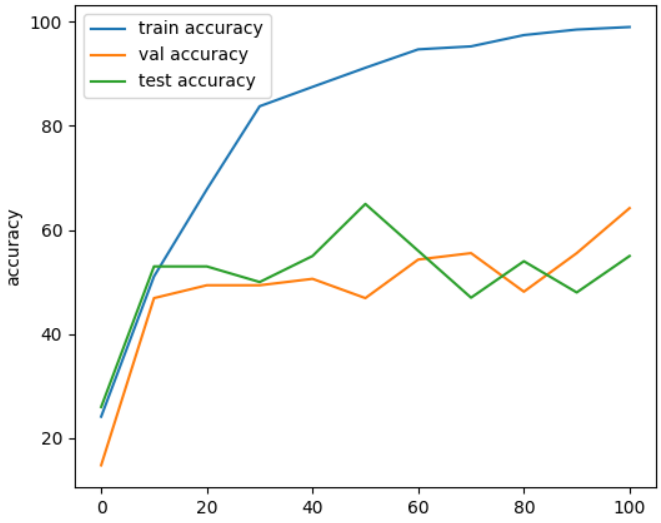

FIGURE 12 – Evolution de l'*accuracy* (%) en fonction du nombre d'*epochs* pour le modèle ResNet-50.

Aussi, par contrainte de temps nous avons décidé de nous restreindre à étudier les différents modèles après un entraînement de 51 *epochs*. De plus, limiter le temps d'entraînement des modèles permet d'assurer un temps d'entraînement plus rapide sur une machine moins performante. Il aurait toutefois été plus intéressant d'étudier les modèles sur un nombre d'*epochs* variable en fonction de la convergence des modèles car rien ne garantit que les modèles commencent tous à *overfit* au bout de 51 *epochs*. Ainsi, nous avons décidé d'entraîner les modèles présentés dans l'analyse avec les transformations à la volée et l'augmentation de données sur 51 *epochs*. Le Tableau 5 montre les résultats obtenus pour les différents modèles utilisés.

Model	Class	Mean TCP (per class)	Mean TCP	Mean Accuracy
ConvNeXt	Jouet Eveil	35.54	52.82	53.75
	Jouet Imitation	29.94		
	Playmobil	**78.70**		
	Poupee	65.79		
	Vehicule	54.13		
ResNet (from scratch)	Jouet Eveil	42.93	39.79	54.91
	Jouet Imitation	29.89		
	Playmobil	33.58		
	Poupee	36.91		
	Vehicule	55.66		
DenseNet-121	Jouet Eveil	**63.10**	60.66	62.45
	Jouet Imitation	48.12		
	Playmobil	50.38		
	Poupee	61.53		
	Vehicule	80.16		
EfficientNet-B0	Jouet Eveil	57.49	65.45	66.86
	Jouet Imitation	52.64		
	Playmobil	71.06		
	Poupee	77.40		
	Vehicule	68.65		
RegNetY-400MF	Jouet Eveil	34.45	65.72	**67.45**
	Jouet Imitation	**72.09**		
	Playmobil	53.98		
	Poupee	81.02		
	Vehicule	**87.08**		
ResNet-50	Jouet Eveil	48.66	55.58	59.06
	Jouet Imitation	22.97		
	Playmobil	69.48		
	Poupee	63.87		
	Vehicule	72.94		
Xception	Jouet Eveil	45.61	**66.02**	67.09
	Jouet Imitation	54.62		
	Playmobil	76.63		
	Poupee	**82.02**		
	Vehicule	71.24		

TABLE 5 – Mean TCP par classe (%), Mean TCP (%) et Mean *accuracy* (%) par modèle.

Les résultats du tableau sont une moyenne des résultats obtenus sur dix itérations et donc sur dix entraînements différents, ce qui permet d'avoir une idée des performances moyennes de chaque modèle. Les résultats convergeant vers une valeur stable, ils auraient pu être plus précis avec un plus grand nombre d'itérations. Nous aurions aussi pu, pour chaque modèle, noter la variance ce qui aurait été utile pour comparer la stabilité des performances des différents modèles. Dans ce tableau, nous remarquons tout d'abord que la TCP moyenne par classe diffère grandement d'un modèle à l'autre. Cependant, nous pouvons voir que la TCP de la classe "véhicule" est généralement plus élevée que celle des autres classes, tandis que les classes "jouet d'imitation" et "jouet d'éveil" ont une TCP moyenne plus faible. Cela n'est pas surprenant, car ces deux classes sont très similaires. En effet, les jouets d'imitation sont définis par le fait qu'ils représentent des objets réels en version jouet, ce qui, en termes de forme et de couleur, peut les rapprocher des jouets d'éveil. Les véhicules, quant à eux, sont plus facilement identifiables grâce à leurs formes distinctes. Les poupées et les Playmobil sont également facilement identifiables, mais peuvent parfois être confondus entre eux, notamment lorsqu'un seul personnage Playmobil est présent sur l'image. Cela explique donc la disparité des résultats des TCP moyennes par classe. Concernant le choix d'un modèle, nous remarquons que trois modèles se démarquent des autres par leur *accuracy* moyenne : RegNetY-400MF, EfficientNet-B0 et Xception. Ces trois modèles affichent une *accuracy* et une TCP moyennes supérieures à 65%, tandis que les autres modèles ont une *accuracy* moyenne comprise entre 50% et 60% et une TCP moyenne entre 40% et 60%. Nous avons donc choisi d'utiliser le modèle RegNetY-400MF pour la suite de notre étude. Ce choix est justifié par le fait que ce modèle présente une *accuracy* moyenne de 67,45% et une TCP moyenne de 65,72%, ce qui en fait le modèle ayant la meilleure *accuracy* moyenne et la deuxième meilleure TCP moyenne. Des tests supplémentaires pourraient être effectués pour confirmer ce choix. Toutefois, l'objectif de ce projet n'étant pas d'identifier le modèle le plus performant, mais plutôt de sélectionner un modèle performant et de développer une application autour de celui-ci, nous avons décidé de nous arrêter sur ce modèle.

◆ Performances du modèle RegNetY-400MF

La Figure 13 présente l'évolution de l'*accuracy* en fonction du nombre d'*epochs* et montre alors qu'entraîner le modèle sur plus de 51 *epochs* n'est pas nécessaire.

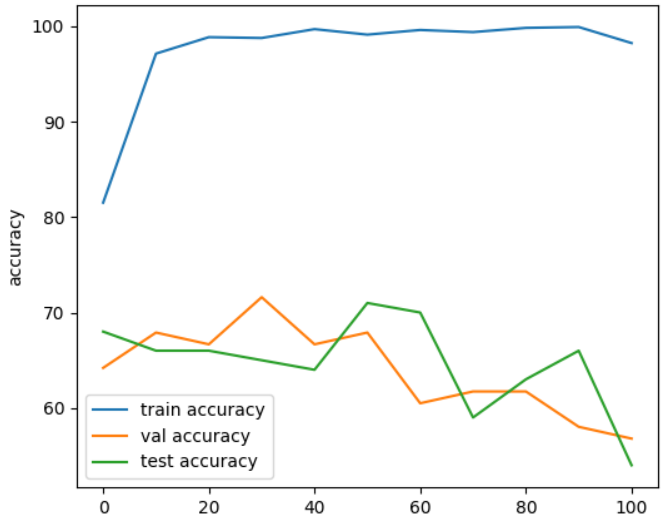

FIGURE 13 – Evolution de l'*accuracy* (%) en fonction du nombre d'*epochs* pour le modèle RegNetY-400MF.

Les différences d'*accuracy* entre celles calculées et celles données en résultats des différents articles scientifiques présentant les modèles utilisés sont dûes au fait que peu d'images soient utilisées pour la classification ce qui défavorise la bonne généralisation des modèles. Aussi et comme mentionné plus haut, les classes utilisées pour la tâche de classification ne sont pas exclusives et larges, ce qui rend l'identification de similitudes pour chaque classe particulièrement complexe. La Figure 14 montre la matrice de confusion du modèle utilisé.

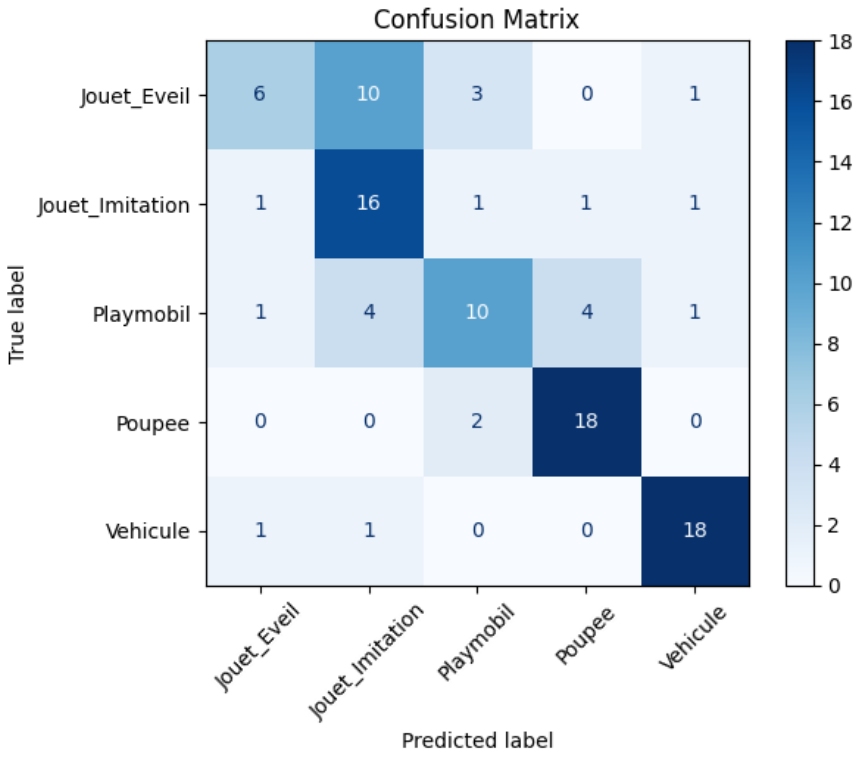

FIGURE 14 – Matrice de confusion du modèle RegNetY-400MF sur l'ensemble de données de test.

On peut notamment observer que le modèle a du mal à distinguer les jouets d'éveil des jouets d'imitation. Cela peut être dû au fait que les jouets d'éveil et les jouets d'imitation ont des formes et des couleurs similaires et que, comme évoqué précédemment, la classe Jouet d'éveil est une classe trop large.

6 Système de requêtes

Afin de permettre à plusieurs utilisateurs d'utiliser en temps réel les mêmes données pour la classification, nous avons décidé de mettre en place une application Dropbox à l'aide de son API dédiée. Cette API permet de créer un espace Dropbox partagé entre plusieurs utilisateurs et d'établir un lien entre l'application et le dossier partagé. Pour cela, il est nécessaire de créer un compte Dropbox et une application sur le site, ce qui génère une clé d'API et un secret d'API. Ces deux éléments doivent être renseignés dans le code de notre application afin de permettre la connexion entre **CAJOU** et Dropbox. Une fois cette configuration effectuée, notre application permet à chaque utilisateur de s'authentifier à l'aide d'un compte Dropbox commun afin d'obtenir un *token* temporaire. Ce *token* est utilisé pour accéder au dossier partagé et récupérer les données qui y sont stockées. Un système de requêtes est mis en place pour *uploader* et *télécharger* les données du dossier partagé. Ainsi, les images catégorisées par l'utilisateur peuvent être sauvegardées dans le dossier partagé à l'aide d'un *upload*. Avec une requête similaire, l'utilisateur peut *uploader* les poids du modèle courant stockés en local. Avant un entraînement, une requête *download* est émise afin de récupérer l'ensemble des images contenues dans le dossier partagé de sorte à lancer un entraînement sur cet ensemble de

données ajouté aux données stockées localement. L'application, dans sa version actuelle, permet uniquement de se connecter à un compte commun unique à l'entreprise pour partager les différents fichiers.

Réalisation

7 Architecture

Notre application suit une architecture modulaire, avec une séparation claire en deux parties : l'application et la classification. Toutes les fonctionnalités du projet sont contenues dans un dossier **src**, qui comprend les sous-dossiers suivants :

- **application** : contient les fichiers relatifs à l'interface graphique de l'application.
- **classifier** : contient les fichiers pour la classification des images et l'entraînement du modèle.
- **tests** : contient les fichiers de tests pour les différents modèles.
- **data** : contient les données d'entraînement, de test ainsi que les poids des modèles.

◆ Application

Pour l'interface graphique de l'application, nous avons utilisé la bibliothèque **PyQt6**. Cette bibliothèque permet de créer des interfaces graphiques intuitives et ergonomiques. L'intégralité de l'application est contenue dans le fichier **appli.py**, qui est lui-même divisé en plusieurs sections correspondant aux différentes pages de l'application.

◆ Classifier

Dans cette partie, nous avons implémenté les fonctionnalités de classification des images et d'entraînement du modèle. Plusieurs fichiers sont présents dans ce dossier :

- **model.py** : contient les modèles de réseaux de neurones décrits dans la partie **Modèles** et utilisés pour la classification des images.
- **train.py** : contient les fonctions pour l'entraînement du modèle.
- **dataloader.py** : contient les fonctions pour le chargement et la préparation des données.
- **tools.py** : contient des fonctions utilisées dans différentes parties du projet.

Pour la classification des images, nous avons utilisé plusieurs modèles de réseaux de neurones convolutifs, comme décrit dans la partie **Modèles**.

◆ Tests

Un seul fichier, **test.py**, est présent dans ce dossier. Ce fichier contient plusieurs tests permettant d'évaluer l'accuracy des différents modèles.

◆ Data

Ce dossier contient les différentes données du dataset ainsi que d'autres fichiers d'images qui sont enregistrés lors de l'utilisation de l'application. C'est également ici, dans un sous-dossier **model**, que sont stockés les poids des modèles après l'entraînement ou le téléchargement.

◆ Main

Un fichier **main.py** est présent à la racine du projet. Ce fichier permet de lancer l'application et d'effectuer différentes actions, telles que l'entraînement du modèle ou le test de performance. Ainsi, avec la commande :

— `python main.py` : l'utilisateur peut lancer l'application.
— `python main.py train` : lance un entraînement du modèle.
— `python main.py test` : teste les performances du modèle (unitest et multitest)

Les fonctions **main** de **train.py** et **test.py** comportent également plusieurs modes d'entraînement et de test. Ainsi :

— `python main.py apptrain` : permet de lancer un entraînement similaire à celui de l'application.
— `python main.py multitest` : permet d'effectuer des tests sur des images contenant plusieurs classes.
— `python main.py unitest` : permet d'effectuer des tests sur des images contenant une seule classe.

De plus, pour faciliter le lancement et l'installation de l'application, un fichier **CAJOU-Windows.bat** et un fichier **CAJOU-Linux-Darwin.sh** sont présents à la racine du projet. Ces fichiers permettent de lancer l'application et son installation sur les systèmes d'exploitation Windows et Linux/MacOS respectivement. Cependant, cette fonctionnalité permet uniquement de lancer l'application, comme avec la commande `python main.py`. Ainsi si l'on souhaite lancer un entraînement ou un test manuellement, il faudra utiliser les commandes citées plus haut.

Beacoup des variables utilisées dans le projet sont définies dans le fichier **variables.yaml**. Ce fichier permet de centraliser les variables utilisées dans le projet et de les modifier facilement. C'est par exemple, dans ce fichier que l'on peut définir les classes des jouets et ainsi en rajouter ou en enlever si nécessaire. Le fichier comporte également les chemins d'accès relatifs des différents dataset, des poids des modèles, des images de test, etc.

8 Interface utilisateur

Nous avons optés à la demande du client, pour notre application de Classification automatique d'images, de l'implémenter comme étant une application de bureau. Nous avons également fait le choix de concevoir une application simple et épurée pour ne pas surcharger les utilisateurs d'informations. Nous avons utilisé **PyQt6** pour développer l'interface utilisateur (UI) de notre application. En effet, PyQt6 offre une intégration complète avec Qt, une bibliothèque qui permet de concevoir des interfaces modernes et ergonomiques. Ensuite, PyQt6 est multiplateforme, ce qui signifie que notre application peut fonctionner sur Windows, MacOS et Linux sans modifications majeures. La bibliothèque fournit également un ensemble riche de *widgets* et composants graphiques qui facilitent le développement d'interfaces interactives et intuitives. Enfin, PyQt6 s'intègre parfaitement avec Python, ce qui nous permet de bénéficier de la simplicité et de la lisibilité du langage tout en exploitant la robustesse de Qt. Cette intégration nous permet en outre d'utiliser simplement les fonctions et les modèles implémentés en Python pour la classification d'images. Nous avons aussi choisi d'utiliser **OpenCV** pour la gestion des images et la capture vidéo. OpenCV (Open Source Computer Vision Library) est une bibliothèque puissante et optimisée pour le traitement d'images

et la vision par ordinateur. L'un des principaux avantages d'OpenCV est sa vitesse d'exécution, ce qui nous permet de traiter en temps réel les images capturées, notamment pour l'utilisation de la webcam. De plus, OpenCV est elle aussi multiplateforme. L'utilisation de la webcam permet en outre d'afficher le flux vidéo aux utilisateurs ainsi que de prendre des photos en temps réel. Ces photos seront par la suite utilisées pour la classification automatique d'images.

Lorsque l'utilisateur lance Cajou, il a dans un premier temps accès à une page d'accueil comme montré sur la Figure 15. Il doit d'abord se connecter à un compte Dropbox pour utiliser l'application.

FIGURE 15 – Page d'accueil de Cajou

La connexion à Dropbox est une demande du client et sert à centraliser les données (images catégorisées et poids du modèle) au même endroit, évitant ainsi une perte de temps considérable quant à la mise en commun des différentes données. Lorsque l'utilisateur clique sur le bouton **"Se connecter à Dropbox"**, un code à usage unique lui est demandé. L'utilisateur récupère ce code sur une page web qui s'ouvre simultanément comme l'illustre la Figure 16.

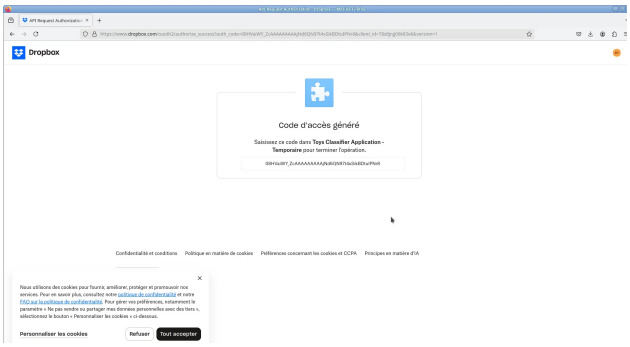

FIGURE 16 – Code d'accès à Dropbox

Une fois la connexion au compte Dropbox faite, l'utilisateur a accès au menu de Cajou (Figure 17).

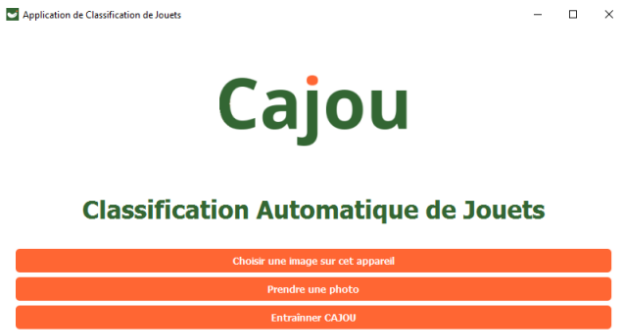

FIGURE 17 – Menu de Cajou

Trois actions lui sont disponibles. Le premier bouton intitulé **"Choisir une image sur cet appareil"** permet, comme son nom l'indique, de selectionner une image déjà présente sur son ordinateur, e.g. dans le cas où le client aurait déjà pris des photos en amont et souhaite les catégoriser. Comme montré sur la Figure 18, l'utilisateur peut ainsi naviguer dans les dossiers de son appareil afin de sélectionner l'image de son choix.

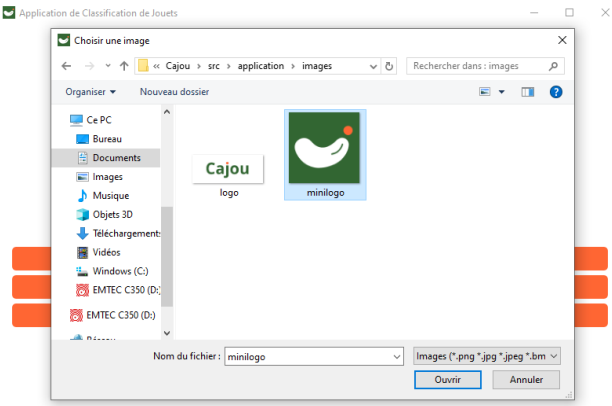

FIGURE 18 – Sélectionner une image sur l'ordinateur

La deuxième action qui lui est disponible est la prise de photo. La Figure 19 montre l'interface de la webcam. L'utilisateur peut ainsi prendre en photo le jouet grâce au bouton **"Capturer l'image"**. De plus, l'image étant capturée en temps réel, un nom de fichier est donné par défaut [4]. Nous laissons cependant le choix à l'utilisateur de mettre un intitulé plus précis dans la zone d'écriture dédiée.

4. Le nom du fichier par défaut est donnée en fonction de la date du jour ainsi que de l'heure de la prise de photo.

FIGURE 19 – Capture de l'image

Une fois l'une de ces deux actions effectuée, l'application utilise le modèle entraîné au préalable pour classifier l'image courante, i.e. l'image sélectionnée ou la photo capturée. Les résultats des probablités associées à chaque classe sont ensuite affichés comme le montre la Figure 20.

FIGURE 20 – Classification

Les différents boutons verts correspondent aux différentes classes possibles pour le jouet. L'utilisateur peut ainsi valider le choix de l'IA ou le modifier en cas d'erreur en cliquant sur l'un des boutons. **"Sauvegarder"** l'image permet de stocker l'image dans le dossier partagé Dropbox correspondant. L'erreur étant humaine, le bouton **"Annuler"** permet au contraire de supprimer l'image du dossier en cas d'erreur. Il permet aussi d'effacer la sélection de la classe précédemment effectuée. Le bouton **"Suivant"** permet quant à lui de passer à l'image suivante et permet notamment de sélectionner à nouveau une image stockée en local si l'utilisateur avait choisi cette action ou de capturer à nouveau une photo s'il avait choisi cette autre possibilité. Nous pouvons observer qu'un bouton **"Hors catégorie"** est présent. Ce bouton peut être utilisé lorsqu'un jouet ne rentre dans aucune des classes présentes. Les images enregistrées dans le dossier partagé Hors Catégorie ne seront en revanche pas utilisées par la suite pour l'entraînement du modèle. En effet, prendre en compte ce type de catégorie pourrait biaiser le modèle et ainsi faire chuter ses performances.

Pour finir, le bouton **Menu** permet de revenir au menu des actions. La troisième action disponible

sur le menu est l'entraînement du modèle qui est caractérisé par le bouton **Entraînement du Modèle**. Comme le montre la Figure 21, ce bouton donne accès à d'autres boutons utiles pour l'entraînement.

FIGURE 21 – Page d'entrainement du modèle

L'utilisateur peut lancer un entraînement, qui s'exécute en tâche de fond, ce qui permet à l'application de télécharger dans un premier temps les images catégorisées sur Dropbox et de les utiliser. Ces images sont téléchargées temporairement en local avant le prétraitement des données. L'utilisateur peut arrêter l'entraînement lorsqu'il le souhaite. De plus les boutons **"Uploader le modèle"** et **"Télécharger le modèle"** permettent respectivement d'envoyer la dernière version des poids du modèle dans le dossier partagé sur Dropbox afin que tous les collaborateurs puissent y avoir accès et de les télécharger s'ils ont été mis à jour.

9 Limites et prospectives

L'application **Cajou** répond aux exigences du client et offre une interface intuitive pour la classification d'images de jeux de seconde main. Elle permet notamment aux utilisateurs d'importer des images depuis leur ordinateur ou via une capture webcam. Le modèle de classification RegNetY-400MF choisi a été entraîné sur l'ensemble de données que nous avons rassembler. Le modèle est performant en comparaison avec l'*accuracy* donnée par Radosavovic et al.[11], i.e. l'*accuracy* de notre modèle est en moyenne de l'ordre de 67.45% alors que l'*accuracy* attendue est d'environ 74.1% (top-1 *accuracy*). Cette différence peut s'expliquer par la taille de notre dataset qui est plus petit que celui utilisé par Radosavovic et al. et par le fait que les classes de jouets que nous avons utilisé ne sont pas exclusives et larges. Il serait alors intéressant d'entraîner le modèle sur un dataset plus grand et sur des classes plus facilement caractérisables par le modèle. Il aurait été aussi pertinent d'étudier le comportement de divers modèles de la famille des Regular Networks Y comme le modèle RegNetY-800MF ou le modèle RegNetY-16GF qui sont plus performants en terme d'*accuracy* que le modèle RegNetY-400MF. Il aurait notamment été intéressant de comparer les performances d'*accuracy* ainsi que les temps d'entraînement de ces modèles sur notre dataset. Aussi, nous pourrions étudier la vitesse d'entraînement du modèle avec et sans GPU. En effet, le GPU permet d'accélérer le temps d'entraînement des modèles mais la présence et la qualité de ce composant ne sont pas garanties ni égales pour tous les utilisateurs. Une solution serait d'externaliser l'entraî-

nement sur un service cloud afin d'uniformiser les performances pour l'ensemble des utilisateurs. Etendre l'étude aux modèles de classification multi-label d'images permettrait aux utilisateurs de pouvoir donner plusieurs catégories à des jouets. En effet, certains jouets peuvent appartenir à plusieurs classes (Figure 11). L'utilisation de modèles de classification multi-label permettrait ainsi une catégorisation plus flexible. La synchronisation avec Dropbox, quant à elle, permet de centraliser les données entre les différents utilisateurs. Cependant, les utilisateurs doivent tous se connecter à un compte Dropbox unique pour avoir accès aux données centralisées. Il aurait été intéressant de permettre à chaque utilisateur de se connecter à son propre compte Dropbox pour avoir accès ces données. L'interface de gestion du modèle de classification permettant notamment son entraînement, son téléchargement et son envoi à Dropbox est un plus pour la pérennité et l'évolution du modèle. Tous ces éléments permettent une utilisation adaptée et efficace de **Cajou** dans un cadre interne à l'entreprise, simplifiant la mise en commun des données, leur gestion ainsi que leur annotation semi-automatique. Il serait toutefois intéressant d'améliorer la prise de photo des jouets. En effet, nous avons pu remarquer que si la photo est prise avec un arrière plan non uniforme et non monochrome et/ou avec la présence d'un visage humain, la prédiction du modèle est beaucoup moins précise. Une solution serait d'utiliser des filtres de détection de contours comme le filtre de Sobel mentionné plus haut et la suppression de l'arrière plan. Aussi, nous pourrions envisager d'entrainer le modèle sur d'avantage d'images de jouets avec un arrière plan plus complexe. Pour le moment nous recommandons aux utilisateurs d'uniformiser au mieux l'arrière plan et plus généralement le champ de la webcam lors de la capture d'images. De plus, toutes les photos prises en temps réel à l'aide de la webcam sont enregistrées dans un dossier spécifique localement. Un enregistrement récurent d'images surchargerait alors l'espace disque. Une solution serait d'ajouter une fonctionnalité de suppression d'images, e.g. après une période donnée ou en implémentant un bouton permettant de vider le dossier en question. Enfin, il est difficile actuellement d'ajouter une classe sans modifier le code source même si un guide simple et détaillé a été rédigé. Il aurait été préférable d'intégrer cette fonctionnalité directement dans l'application.

Conclusion

Notre Projet de Fin d'Etudes a porté sur le développement d'une application de classification automatique d'images de jouets de seconde main que nous avons nommé **Cajou**. L'objectif était alors de produire un outil permetttant de catégoriser les jouets dans différentes classes données par notre client. Pour produire notre modèle, nous avons étudier différents réseaux de neurones convolutifs existant. Nous avons testé plusieurs modèles pré-entrainés dont les modèles ConvNeXt, ResNets, DenseNet-121, EfficientNet-B0, RegNetY-400MF et Xception. Nous avons optés pour le modèle RegNetY-400MF qui offre la meilleure précision sur notre ensemble de données. Pour diminuer le problème de mauvaise généralisation du modèle engendré par la faible quantité de données et le fait que les classes ne soient pas exclusives nous avons décidé d'intégrer un système d'apprentissage supervisé à l'application. L'interface utilisateur permet d'afficher des prédictions pour une image choisie ou capturée à l'aide de la webcam puis de la catégoriser manuellement. Les images classifiées sont stockées sur Dropbox ce qui permet à l'ensemble des utilisateurs d'y avoir accès. L'utilisateur peut ensuite lancer un entraînement avec ces images et peut aussi stocker le modèle en ligne. L'intégration de Dropbox permet ainsi de centraliser les données sur un espace cloud partagé.

Toutefois, l'application possède quelques limites. Le fait que le temps d'entraînement du modèle ne soit pas uniforme pour tous les utilisateurs peut être un frein. De plus, l'ajout de nouvelles classes nécessitant une modification du code source peut également poser problème. Enfin, la qualité des images capturées par la webcam peut influencer la précision des prédictions. Malgré ces limites, l'application répond aux exigences du client et permet de classer les jouets de seconde main de manière efficace. Ce projet nous a ainsi permis d'explorer les enjeux et les défis de la classification automatique d'images de jouets, tout en étudiant et mettant en œuvre des technologies modernes de vision par ordinateur et d'apprentissage automatique. Grâce à cette approche, nous avons pu proposer une solution évolutive qui répond aux besoins du client, tout en ouvrant des perspectives d'amélioration pour de futurs développements.

Références

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016.

[2] François Chollet. Xception : Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[3] Charles Corbière, Nicolas Thome, Avner Bar-Hen, Matthieu Cord, and Patrick Pérez. Addressing failure prediction by learning model confidence. *Advances in neural information processing systems*, 32, 2019.

[4] Margarita Favorskaya and Andrey Pakhirka. Animal species recognition in the wildlife based on muzzle and shape features using joint cnn. *Procedia Computer Science*, 159 :933–942, 2019.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[7] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[8] Hee E Kim, Alejandro Cosa-Linan, Nandhini Santhanam, Mahboubeh Jannesari, Mate E Maros, and Thomas Ganslandt. Transfer learning for medical image classification : a literature review. *BMC medical imaging*, 22(1) :69, 2022.

[9] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer : Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[10] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.

[11] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces, 2020.

[12] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet : Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv :1711.05225*, 2017.

[13] Tuba Siddiqui. Toy cars annotated on yolo format, 2021.

[14] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[15] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet : Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828, 2019.

[16] Mingxing Tan and Quoc V. Le. Efficientnet : Rethinking model scaling for convolutional neural networks, 2020.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[18] Contributeurs Wikipedia. Vanishing gradient problem, 2025. Accès le : 12/03/2025.

[19] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[20] Samir S Yadav and Shivajirao M Jadhav. Deep convolutional neural network based medical image classification for disease diagnosis. *Journal of Big data*, 6(1) :1–18, 2019.