

# Note méthodologique : POC

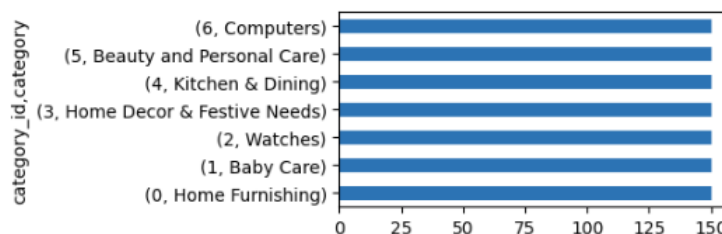
Notre projet utilise la version **V2** d'**Efficient Net** développée par Google disponible dans le module *Python* de *TensorFlow.Keras*.

PI, la V1 de 2019 est moins performante et la V3 de 2023 n'est pas encore disponible.

## Dataset retenu

La **source** des données est celle du projet N° 6 "*Classifiez automatiquement des biens de consommation*" (cf. [fichier ZIP sur AWS](#)). Il s'agit d'un millier d'images et d'un **tableau** avec les informations des images de produits, notamment un **arbre** de catégorisation de produits.

A partir de l'**arbre**, on extrait la catégorie de chaque image avec un identifiant de catégorie. Il y a 7 catégories distribuées équitablement :

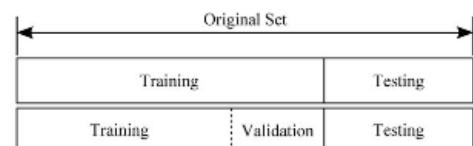


A partir du **tableau**, on extrait un **Pandas** à 2 colonnes

	category_id	image
0	0	55b85ea15a1536d46b7190ad6fff8ce7.jpg
1	1	7b72c92c2f6c40268628ec5f14c6d590.jpg

Le **Pandas** est partitionné en 3 jeux (entraînement, validation et test)

- le jeu complet est divisé en apprentissage et test
  - le jeu d'apprentissage est sous-divisé en entraînement et validation
- les partitions conservent la distribution équilibrée des catégories
- les jeux d'entraînement et validation sont **augmentés** par transformations d'images (miroir horizontal, vertical et rotation de 90°) afin d'incrémenter le volume de données



Le **système de fichiers** est également partitionné. Le modèle *EfficientNet* requiert :

- la distribution des fichiers images dans des répertoires correspondant aux 3 jeux de données et aux 7 catégories
- une résolution d'images spécifique au modèle pré-entraîné (240x240 pour B1)

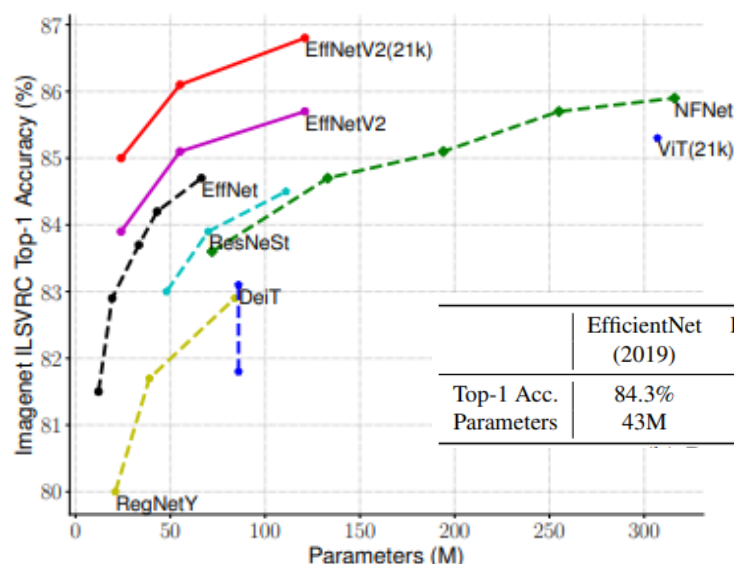
Le **jeu de données** en entrée du modèle pré-entraîné est composé donc de ces images chargées dans 3 **tableaux Numpy** (entraînement, validation et test).

## Les concepts de l'algorithme récent

Nous avons utilisé les modèles pré-entraînés B0 et B1 d'**Efficient Net V2** qui sont plus petits et plus rapides à l'entraînement comme le dit le titre de l'article "[EfficientNetV2: Smaller Models and Faster Training](#)" datant de 2021. PI, cet article a été trouvé grâce à [Google-Scholar](#) et Arxiv.

Modèle	Paramètres (millions)	FLOPs (milliards)	Résolution (pixels)
EfficientNetV2-B0	7.1	0.8	224 x 224
EfficientNetV2-B1	8.1	1.2	240 x 240
EfficientNetV2-B2	9.2	1.9	260 x 260
EfficientNetV2-B3	12.0	3.2	300 x 300

**Efficient Net V2** est une nouvelle famille de réseaux convolutifs qui optimise : l'exactitude (*accuracy*), le temps d'entraînement et le nombre de paramètres.



## Recherche automatique d'architecture neuronale ([NAS](#))

- NAS a été utilisée pour concevoir le modèle pour optimiser l'efficacité des paramètres et l'entraînement : vitesse, stabilité, mise à l'échelle (p.ex. le traitement de gros volumes) et utilisation de ressources.
- NAS a identifié automatiquement les blocs classiques de convolution qu'il convient de remplacer par les plus récents *Fused-MVConv*
- Le score mesuré par NAS dépend de l'exactitude, le temps d'entraînement, et le nombre de paramètres :  $E \times T^w \times P^v$ 
  - les valeurs  $w=-0.07$  et  $v=-0.05$  ont été trouvées de manière empirique

## Bloc Fused-MVConv

- Bloc de convolution introduit en 2019 et qui exploite de façon optimale les accélérateurs de serveur (*GPU, TPU, FPGA, ASIC*).

## Apprentissage progressif

Combine l'incrément progressif de la résolution des images avec la *régularisation adaptative* (p. ex. celui de la robustesse de l'augmentation de données). Ces ajustements se font de manière dynamique.

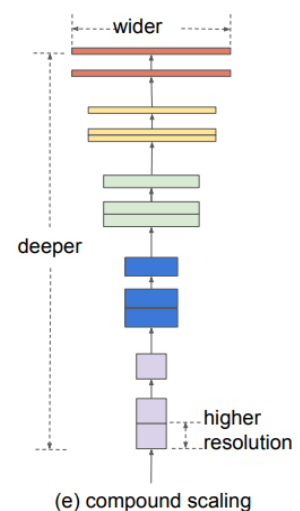
- Redimensionnement progressif des images
  - lors des premières époques d'entraînement on traite des images de basse-résolution pour apprendre les caractéristiques basiques
  - lors des dernières époques, celles de haute-résolution (480 pixels max.) pour apprendre des détails plus fins
  - pour traiter des images de haute-résolution, il faut que le modèle soit large, donc, assez puissant pour apprendre trop bien
  - d'où le risque de sur-apprentissage et de perte d'exactitude (*accuracy*)
- Régularisation adaptative
  - Augmentation aléatoire d'image
    - compense cet effet pervers en appliquant
      - des transformations légères sur les images de faible résolution lors des premières époques
      - des transformations plus importantes sur les images de haute-résolution
    - introduit de la variété ce qui empêche le sur-apprentissage.
  - Augmentation par la combinaison d'attributs et de cibles de 2 images
$$X_i = \lambda X_j + (1 - \lambda) X_i$$
$$y_i = \lambda y_j + (1 - \lambda) y_i$$
  - Désactivation aléatoire de canaux
    - coupe temporairement des connexions entre neurones trop dépendantes
    - force le modèle à apprendre des caractéristiques générales au lieu de mémoriser des motifs spécifiques des images d'entraînement avec risque de sur-apprentissage

## Mise à l'échelle combinée du réseau

Optimise la performance du réseau de neurones en ajustant trois dimensions :

1. profondeur ou nombre de couches pour capturer des caractéristiques complexes
2. largeur ou nombre de canaux par couche pour capturer plus de caractéristiques
3. résolution ou taille des images pour capturer plus d'informations

Ceci assure que la croissance en profondeur, largeur et résolution est équilibrée et ne dépasse pas la capacité de calcul disponible



# La modélisation

## Méthodologie de modélisation

### Comparaison homogène

On comparera le nouveau modèle *EfficientNet V2* avec le meilleur modèle du projet N° 6 : *Random Forest*.

- on alimente les modèles avec le même jeu d'images
  - les attributs texte sont enlevés du jeu de données issu du projet N° 6
  - le jeu d'images original est augmenté par transformations
  - on adapte les images à la résolution requise par le nouveau modèle
- on mesure avec la même métrique les performances des modèles à comparer

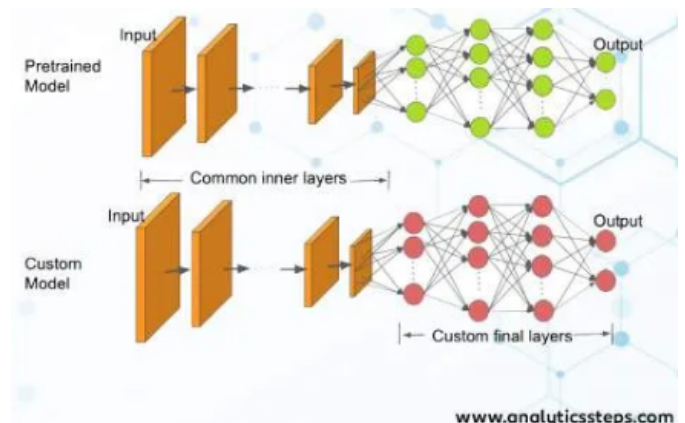
### Prévention de fuites

On vérifie qu'il n'y a pas de fuite d'information entre les différentes partitions

- l'entraînement par *fit()* se fait seulement sur le jeu d'apprentissage divisé en entraînement et validation
- la prédiction par *evaluate()* se fait seulement sur le jeu de test

### Transfer Learning

- la dernière couche des modèles pré-entraînés *EfficientNetV2* renvoie un vecteur de mille catégories correspondant aux classes des données ImageNet
- on les adapte à notre projet avec 7 catégories en remplaçant les 4 dernières couches
- on compile le modèle avec l'ensemble complet de couches



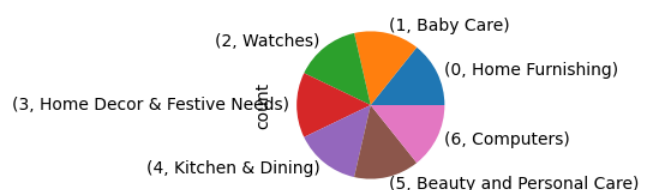
## Métrique d'évaluation retenue

### Exactitude


On utilisera la métrique choisie pour le projet N° 6 qui était l'exactitude ou "*accuracy*" pour être en cohérence avec les scores relevés pour ce projet.

$$\text{Exactitude} = \frac{\text{Nb prédictions correctes}}{\text{Nb total de prédictions}}$$

Elle donne une bonne idée de la performance globale du modèle, étant donné que les observations de ce jeu de données sont équitablement réparties par classe,



- Ces images illustrent le calcul de l'exactitude pour une classification multi-classe.

Actual						Actual			<div>Predicted</div>	Ground truth				
	Cat	Dog	Tiger	Wolf			Dog	Not Dog			+	-		
Cat	6	2	3	1		Dog	4	7		+	True positive (TP)	False positive (FP)	Precision = TP / (TP + FP)	
Dog	0	4	3	4		Not Dog	6	20		- <td>False negative (FN)</td> <td>True negative (TN)</td> <td></td>	False negative (FN)	True negative (TN)		
Tiger	3	0	3	1								Recall = TP / (TP + FN)		
Wolf	1	4	0	2								Accuracy = (TP + TN) / (TP + FP + TN + FN)		

## Fonction de perte

Lors de la recherche de poids optimales du modèle par la méthode "*fit()*", la métrique minimisée est la fonction de perte ou "*loss*". Elle indique à quel point les prédictions d'un modèle s'écartent des valeurs réelles.

$$Loss = - \sum_{i=1}^N y_i \cdot \log(\hat{y}_i) , y_i: \text{vraie cible (0 ou 1) pour la classe } i, \hat{y}_i: \text{probabilité prédite, } N: \text{nb. classes}$$

## Démarche d'optimisation

Pour l'optimiser le modèle, on évaluera ses versions de configuration B0, B1 et B2. Chacune de ces versions comporte une combinaison différente des valeurs des paramètres suivants :

Modèle	Profondeur	Largeur	Résolution	Nombre de poids	Dropout Rate
EfficientNetV2-B0	20	1.0x	224x224	~7.1M	0.2
EfficientNetV2-B1	22	1.1x	240x240	~8.1M	0.2
EfficientNetV2-B2	24	1.2x	260x260	~9.2M	0.3

N.B. : Le traitement de chacune de ces versions prend des heures et quelques fois provoquent des pannes du Kernel Python. Alors on a préféré ces choix prédéfinis des valeurs des hyper-paramètres au lieu de ceux qu'une grille de recherche par validation croisée pourrait fournir après une longue attente.

Lors de l'entraînement sur les jeux d'entraînement et de validation

- on utilise des fonctions de rappel
  - *Early Stopping* pour interrompre le long traitement (au bout de 2 époques où la perte augmente)
  - *Model Checkpoint* pour sauvegarder le meilleur modèle trouvé dans un répertoire
- le meilleur modèle est trouvé en fonction des scores sur le jeu de validation
- on applique l'optimiseur *Adam (Adaptive Moment Estimation)* sur 10 époques avec une taille de blocs de 16

## Une synthèse des résultats

### Résultats du précédent projet

- Le meilleur modèle du projet N° 6 était *Random Forest* avec un score d'exactitude de 83,81% sur le jeu de test était plus élevé avec un temps de traitement plus court comparé notamment au CNN.
- L'augmentation par transformations appliquées au jeu de données fait incrémenter légèrement le score.

Position	Modèle	Augmentation	Attributs	Score
1	<i>Random Forest</i>	<i>augmenté</i>	photos & texte	83.81%
2	<i>Random Forest</i>		photos & texte	82.38%
3	<i>CNN</i>	<i>augmenté</i>	photos	71.29%

### Résultats du projet présent

- La valeur maximale d'exactitude à 79,75% est obtenue par la configuration la plus légère B0
- Les scores décroissent légèrement avec l'incrément de complexité des réseaux neuronaux B1 et B2

Model	B (Hyper-parameter)	Fit Time (s)	Accuracy
EfficientNetV2B0	0	4589	0.7975
EfficientNetV2B1	1	3418	0.7911
EfficientNetV2B2	2	4598	0.6962
Random Forest	N/A	62	0.8286

### Conclusion sur les résultats

Le nouveau modèle neuronal avec les configurations B0 et B1 donne un meilleur score que le réseau de neurones traditionnel du projet précédent.

Cependant il n'atteint pas un score supérieur à celui de *Random Forest*.

# L'analyse de la feature importance globale et locale du nouveau modèle

N/A

Le jeu de données est composé uniquement de pixels d'images. Il n'y a pas d'attributs identifiables dont on puisse évaluer l'importance, soit-elle globale ou locale.

On ne dispose que des tableaux ou *arrays Numpy* multi-dimensionnels (p. ex 158 x 224 x 224 x 3) sans attributs identifiés

## Les limites et les améliorations possibles

### Limites

1. **Complexité du Modèle:**
  - Les modèles *EfficientNet B0 & B1* consomment beaucoup de ressources
2. **Interprétabilité:**
  - Les modèles CNN sont des "boîtes noires", ce qui ne permet pas de déterminer les attributs importants

### Améliorations

1. **Réduction du temps de traitement**
  - Utiliser un ordinateur ou un serveur plus puissant et performant
  - Utiliser le jeu d'images original qui a une taille inférieure à celle du jeu avec augmentation utilisée pour ce *POC* afin de comparer l'efficacité de l'ensemble de modèles sous les mêmes conditions
2. **Réduction du Sur-apprentissage:**
  - Utiliser des techniques de régularisation comme ***Early Stopping*** qui a été utilisée pour ce *POC*

## Recommandations à OCR pour accélérer le traitement

1. Il faudrait pallier à ces problèmes des étudiants
  - Leurs postes PC ont des ressources limitées souvent sans carte graphique. Ce qui pénalise le traitement d'optimisation de modèles CNN qui dure plusieurs heures et ne permet pas de les optimiser au mieux afin d'exploiter leur plein potentiel
  - Les offres gratuites disponibles de serveurs dans le *Cloud* ont une durée limitée qui parfois dépasse la durée d'un projet

2. Des solutions seraient de

- faire reconnaître OpenClassrooms comme organisme éducatif par des plateformes comme AWS, Azure & Google Cloud afin de faire bénéficier ses étudiants d'offres avantageuses incluant de serveurs puissants pour exploiter les CNN, par exemple.
- attribuer aux étudiants d'adresses email *@openclassrooms.com* pour se servir de Heroku grâce au pack étudiant Github



# Annexe

## Compétences évaluées

### 1. Réaliser une veille sur les outils et tendances en data science

CE1 Le candidat a consulté des sources reconnues d'informations, produites récemment, (blogs reconnus, articles de recherche de journaux et conférences reconnues dans le domaine).

CE2 Le candidat a présenté les points clés de chacune des sources bibliographiques, y compris des détails mathématiques.

CE3 Le candidat a mis en place une preuve de concept pour tester le nouvel outil / le nouveau modèle, la nouvelle démarche, comparée à une approche classique.

### 2. Rédiger une note méthodologique contenant notamment le choix des algorithmes testés, les métriques utilisées et l'interprétabilité du modèle proposé, afin de communiquer sa démarche de modélisation.

CE1 Le candidat a présenté la démarche de modélisation de manière synthétique dans une note.

CE2 Le candidat a explicité la métrique d'évaluation retenue et sa démarche d'optimisation.

CE3 Le candidat a explicité l'interprétabilité globale et locale du modèle.

CE4 Le candidat a décrit les limites et les améliorations envisageables pour gagner en performance et en interprétabilité de l'approche de modélisation.

[Tech Behind Google's New CNN, EfficientNetV2](#) 2021

[EfficientNetV2: Smaller Models and Faster Training](#) avril 2021 ([pdf](#))