MINISTÈRE
DES ARMÉES
Liberté
Égalité
Fraternité

HACKATHON IA DATA HACK - 6milarité

Groupe 4

23/04/2024 - 25/04/2024

Florine Kieraga

Eliana Junker

Eithan Nakache

Sacha Hibon

Martin Natale

Github: https://github.com/chuklee/army

Table des matières

Introduction

Le challenge 6milarité consistait à reconnaître différents modèles de voitures, parfois particulièrement similaires, grâce à l'utilisation d'un réseau neuronal (Resnet18). L'utilisation de l'explicabilité était également demandée.

Ce journal de bord retrace les différentes étapes de notre réflexion durant ce hackathon ainsi que les résultats que nous avons obtenus pour ce challenge.

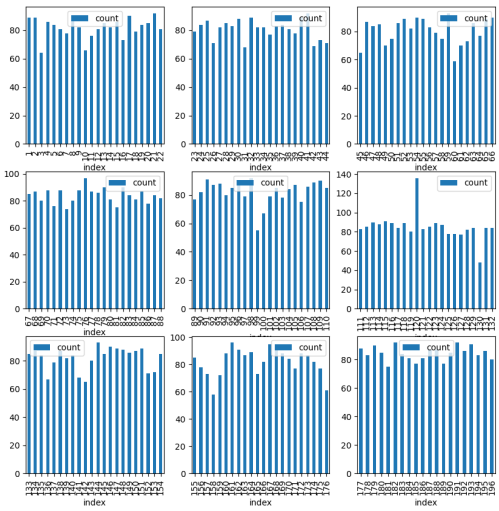
Jour 1

1 Statistiques sur les données

1.1 Répartition des classes

Toutes classes confondues

Afin de connaître la répartition des classes de notre jeu de données, nous avons décidé de regarder la distribution de nos classes afin d'observer si la répartition était équilibrée.

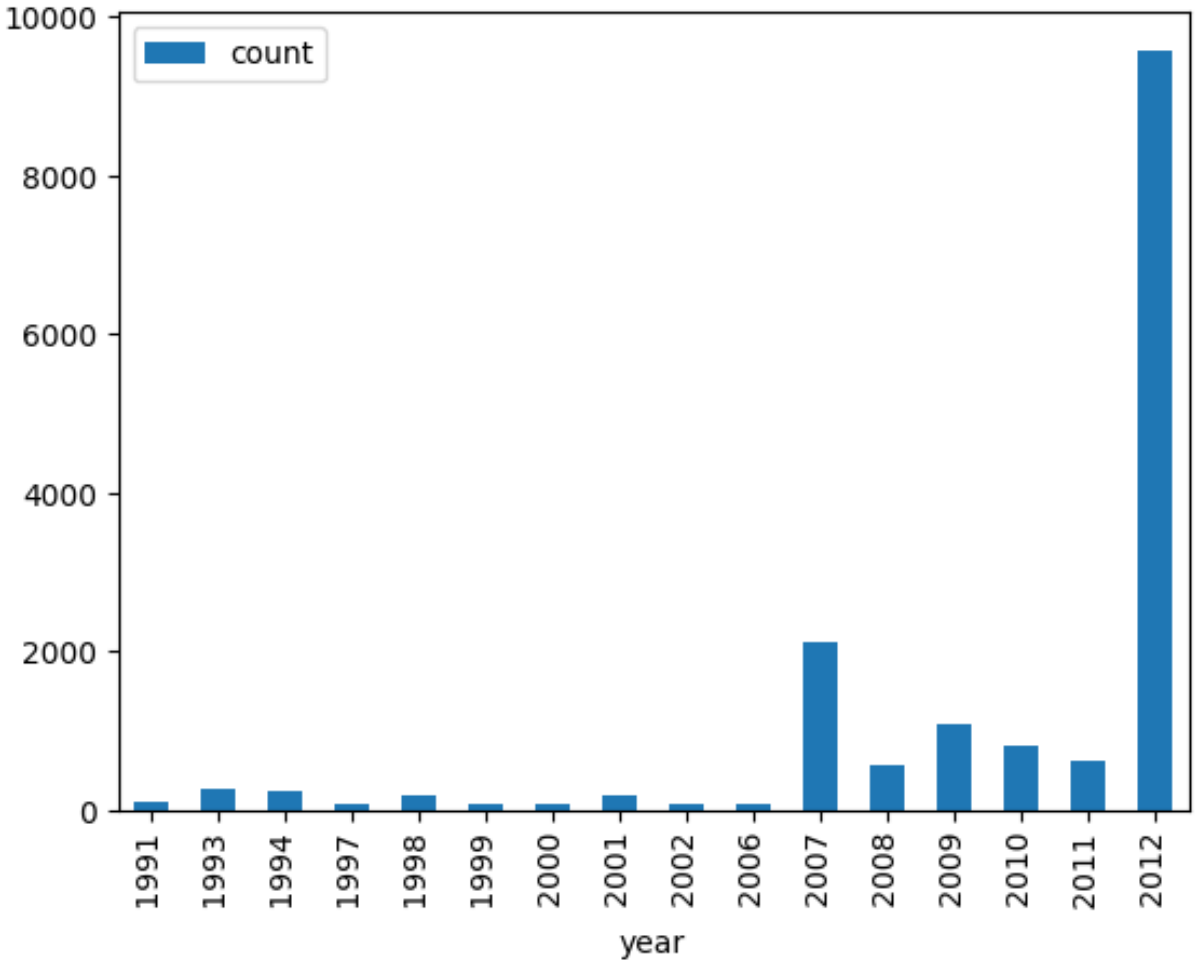


	number of samples per classes
mean	82.566327
std	8.693097
min	48.000000
max	136.000000

Ainsi, nous constatons que les données sont plutôt bien réparties parmi les classes, avec en moyenne 82 images par classe.

Par année

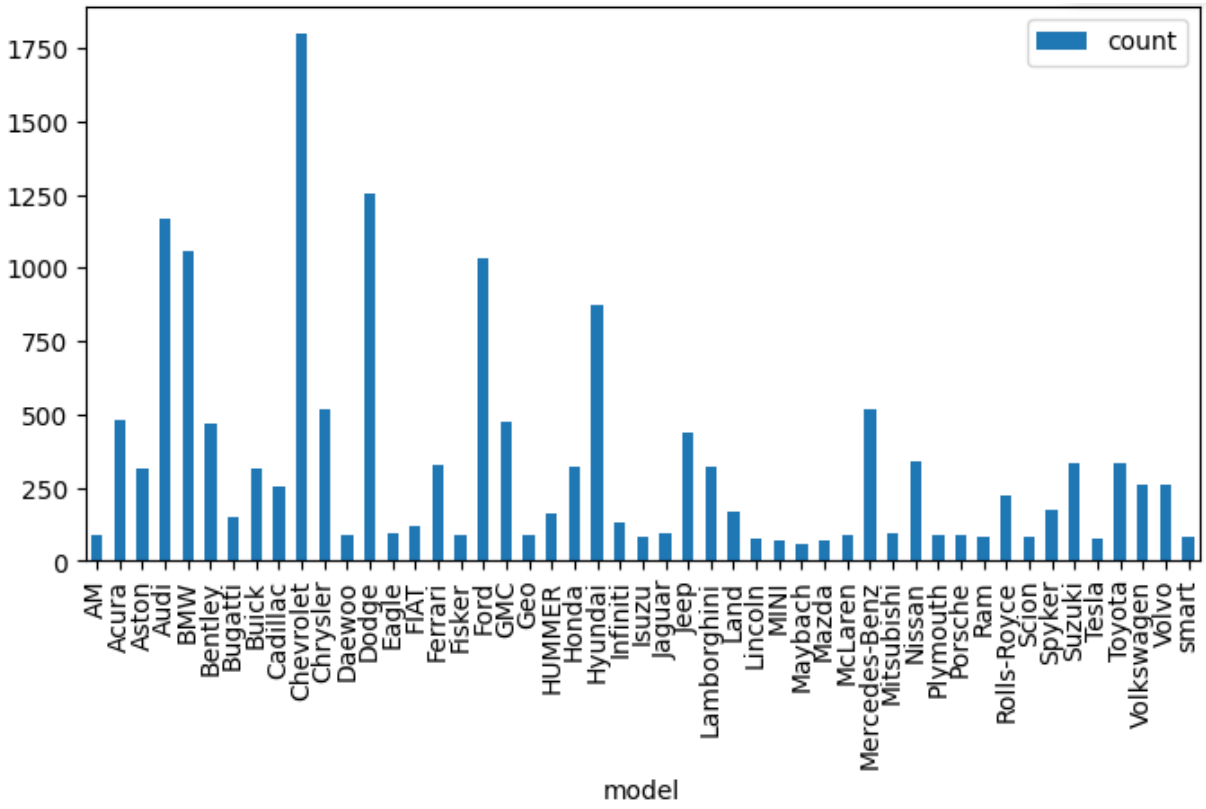
Nous nous sommes par la suite intéressés à l'année de fabrication des modèles. Pour ce faire, nous avons utilisé le fichier 'names.csv' qui contenait les noms des véhicules ainsi que leurs années de fabrication. Voici ci-dessous la distribution des véhicules en fonction des années.



Ainsi, nous pouvons observer que les données des différentes classes ne sont pas réparties équitablement par années.

Par marque (premier mot de chaque classe)

Bien que notre ensemble de données présente une répartition équilibrée des classes, nous avons observé une disparité significative au niveau des marques individuelles. En effet, en examinant la distribution, nous constatons que certaines marques ont seulement 30 images tandis que d'autres en ont 1750.

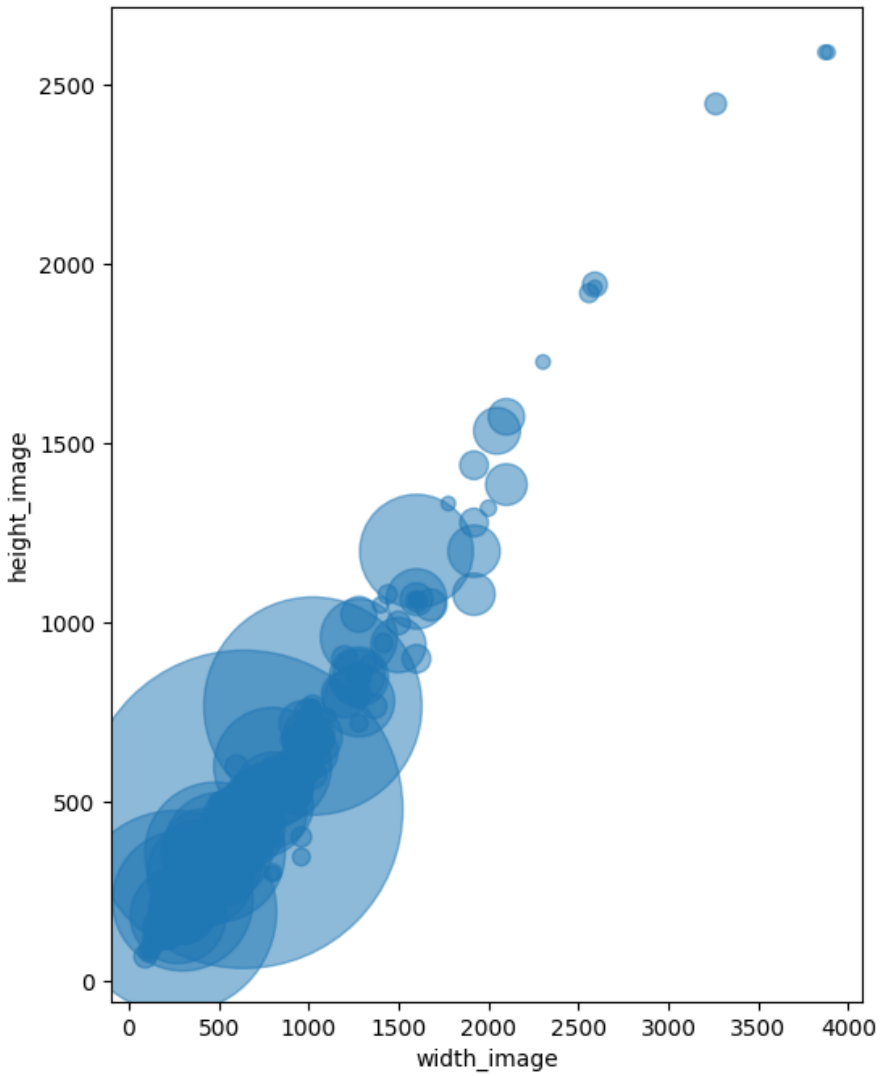


En conclusion, les données ne sont pas équitablement réparties entre les marques, ce qui pourrait influencer les performances de nos modèles d'apprentissage automatique.

1.2 Analyse sur la taille des images

Graphe

En examinant la taille des images dans notre ensemble de données, nous constatons que la moyenne des dimensions est de (700, 483). Cela indique une certaine variabilité dans les tailles d'image, avec des images plus petites et plus grandes que la moyenne.



Table

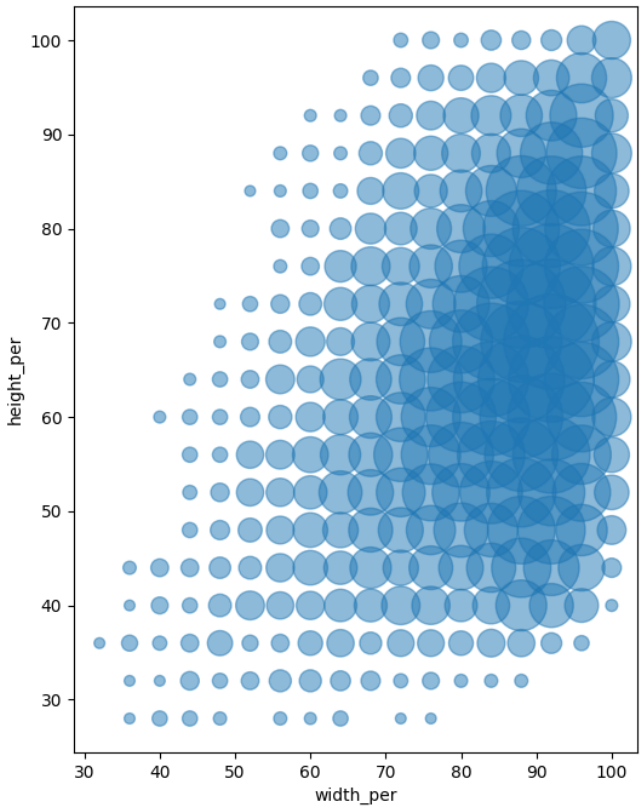
	mean	std	min	max
width_image	700.513131	453.25475	78.0	7800.0
height_image	483.281468	318.32877	41.0	5400.0

Conclusion : La taille des images peut avoir un impact significatif sur la performance des modèles d'apprentissage automatique. Des images trop grandes peuvent nécessiter plus de ressources informatiques pour le traitement, tandis que des images trop petites peuvent perdre des détails importants pour la tâche de classification ou de reconnaissance. Nous nous rendons compte qu'il va falloir prendre en compte cette variabilité lors de la préparation des données, en ajustant éventuellement la taille des images ou en utilisant des techniques de redimensionnement appropriées pour garantir des performances optimales du modèle.

1.3 Analyse sur la taille des voitures comparées aux images

Ici, nous utilisons les informations sur les positions des voitures fournies dans le jeu de données pour comparer la taille réelle des voitures par rapport aux dimensions des images. Cela nous permet d'évaluer si les voitures sont correctement représentées en termes de taille dans nos images, ou s'il existe des distorsions significatives qui pourraient affecter la qualité de nos analyses ou de nos modèles.

Graphe



Table

	mean	std	min	max
width_per	82.951492	12.627833	16.0	116.0
height_per	65.053451	15.530102	20.0	100.0

Conclusion : il n'y a pas vraiment de conclusions intéressantes concernant cette analyse.

2 Création du modèle en PyTorch

Nous avons fait le choix d'utiliser PyTorch pour l'entrainement de notre modèle. Cette bibliothèque a l'avantage d'être très flexible et permet d'importer facilement le modèle Resnet18.

2.1 Première version du modèle

Comme observé dans l'analyse statistique des données, le jeu de données initial a été divisé en deux sous-ensembles distincts : un ensemble d'entraînement et un ensemble de test, chacun représentant 50% des données totales. Les transformations appliquées se limitent à une normalisation des valeurs et une conversion en tenseurs. Les seules transformations effectuées sont la normalisation et la transformation en Tensor. Après avoir demandé à un coach, il nous a été expliqué que pour des images RGB, la normalisation basique est la division par 255. Nous avons décidé d'utiliser cette méthode de normalisation.

Hyperparamètres

Pour l'optimizer, nous avons utilisé l'optimizer Adam avec un learning rate de 0.001. et une loss function de type CrossEntropyLoss. Voulant tester rapidement la convergence de notre modèle, nous utilisons un batch size de 64 qui privilégie la rapidité d'apprentissage pour délaisser les performances du modèle.

Nous avons constaté que la dernière couche de base de Resnet18 possède une sortie prévoyant 1000 classes. Afin que le modèle puisse prédire nos classes, nous avons changé la couche de sortie en une couche de 196 neurones. Nous avons ensuite entraîné le modèle sur 10 Epochs.

Résultats

Le modèle a atteint une accuracy de 65% sur le test. Nous avons donc décidé de changer les hyperparamètres pour voir si nous pouvions améliorer les performances du modèle.

Explication du résultat

Les performances du modèle sont faibles. Cela peut être dû à plusieurs facteurs :

— Le modèle underfit et n'a pas eu le temps de converger

— Les hyperparamètres ne sont pas adaptés

— Le set de données est trop petit

Nous allons donc essayer de changer les hyperparamètres pour voir si nous pouvons améliorer les performances du modèle.

2.2 Deuxième version du modèle

Hyperparamètres

Afin d'améliorer les performances du modèle, nous avons décidé d'utiliser une méthode de data augmentation sur le jeu de train. Les transformations effectuées sont les suivantes : RandomHorizontalFlip, qui va inverser certaines images et RandomRotation(15) qui va faire une rotation de 15 degrés sur certaines images. Nous ajoutons également un learning rates scheduler de type ReduceLROnPlateau avec un facteur de 0.1 et une patience de 5. Nous avons également décidé d'utiliser un batch size de 32 pour améliorer les performances du modèle. Nous avons ensuite entraîné le modèle sur 10 Epochs.

Résultats

Le modèle a atteint une accuracy de 98% sur le train et de 88% sur le test. Les performances du modèle ont été grandement améliorées. Nous remarquons néanmoins qu'une divergence commence à apparaître entre les performances du train et du test, ce qui peut indiquer un overfit du modèle.

3 Recherches sur l'explicabilité

Nous avons commencé des recherches sur l'explicabilité afin de comprendre en quoi cela consistait et quelles bibliothèques étaient disponibles.

Lors de ces recherches, nous avons trouvé plusieurs bibliothèques disponibles pour les modèles PyTorch :

— SHAP (SHapley Additive exPlanations)
— Captum
— Torch-Cam qui permet une visualisation avec la génération des cartes d'activations.
— Lime

Nous avons découvert différentes techniques d'explicabilité :

— SHAP : basée sur la théorie des jeux, permet de mesurer l'impact sur la prédiction d'ajouter une variable grâce à des permutations de toutes les options possibles.
— Lime (Local Interpretable Model-agnostic Explanations) : explique les prédictions des modèles en ajustant un modèle interprétable local autour de l'instance à expliquer.
— Occlusion : cette méthode implique la modification de parties de l'image d'entrée pour évaluer l'impact sur la sortie du modèle.
— Grad-CAM et CAM (Classe Activation Mapping) : génèrent des cartes de chaleur pour visualiser les régions importantes de l'image qui ont contribué à la prédiction de classe du modèle.

Sources :

— https://medium.com/@mariusvadeika/captum-model-interpretability-for-pytorch-c
— https://github.com/frgfm/torch-cam
— https://github.com/pytorch/captum
— https://indabaxmorocco.github.io/materials/hajji_slides.pdf
— https://aqsone.com/blog/articlescientifique/interpretabilite-et-explicabilite
— https://courses.minnalearn.com/en/courses/trustworthy-ai/preview/explainabili types-of-explainable-ai/

4 Hyperparamètres

4.1 Recherches

Tout au long de la création de nos deux versions de modèle, nous avons décidé de faire les recherches et comparaisons suivantes sur les hyperparamètres :

— Taille de batch : Le batch size représente le nombre d'échantillons qui seront propagés à travers le réseau de neurones. Nous avions décidé d'explorer les batchs size suivants : 16, 32, 64, 128.

— Learning rate : Le learning rate, car il était crucial de déterminer le taux d'apprentissage optimal pour la convergence du modèle. Les valeurs que nous avions décidé d'explorer étaient : 0.0001 et 0.001.
— Optimizer : Le choix de l'optimizer est crucial pour la convergence du modèle. Il permet de mettre à jour les poids du réseau de neurones. Nous avions décidé d'explorer les optimizers suivants : Adam, SGD.
— Scheduler : Le scheduler permet de modifier le learning rate au cours de l'entraînement. Nous avions décidé d'explorer les schedulers suivants : StepLR, ReduceLROnPlateau.
— Poids des classes : Nous avions décidé d'analyser la distribution des classes pour déterminer si nous devions utiliser des poids de classe. Nous avions décidé d'explorer les poids des classes suivants à l'aide de la librairie Sklearn : None, balanced.
— Normalisation : La normalisation des données est une étape importante pour l'entraînement du modèle. Nous avions décidé d'explorer les normalisations suivantes : None, ImageNet, Z-score.
— Augmentation des données : Nous avions décidé de mener une recherche sur la data augmentation, car il était important de trouver celle qui permettrait au modèle de converger. Nous avions choisi d'explorer les data augmentations suivantes : None, HorizontalFlip, VerticalFlip, RandomRotation, AutoAugmentation(ImageNet).
— Dimension des images : Nous avions décidé de mener une recherche sur l'image shape, car il était important de trouver la bonne forme d'image pour que le modèle converge. Nous avions choisi d'explorer les formes d'image suivantes : 224, 256, 416, 512.

Sources :
— https://optuna.readthedocs.io/en/stable/index.html
— https://pytorch.org/docs/stable/optim.html
— https://scikit-learn.org/stable/modules/generated/
— https://pytorch.org/vision/stable/transforms.html
— https://pytorch.org/vision/stable/models.html
— https://pytorch.org/docs/stable/nn.html

4.2 Choix

— Pour l'instant, 32 est conservé car les performances obtenues sont satisfaisantes et permettent un temps d'entraînement acceptable. Ce paramètre sera peut-être changé par la suite avec l'utilisation de Optuna.

— Learning rate : En utilisant 0.0001 avec Adam, nous obtenions de meilleurs résultats comparés à 0.001.
— Optimizer : Nous avons décidé d'employer 'Adam' pour nos premiers modèles, car il s'agissait du plus récent et de celui avec lequel nous étions les plus familiers.
— Poids des classes : l'utilisation des poids n'a finalement pas été adoptée, car nous avons remarqué que les classes étaient équilibrées entre elles. Cela entraînait également un problème au niveau de l'entraînement du modèle.
— Normalisation : nous avons utilisé la normalisation '[0.5,0.5,0.5]' à la suite d'une discussion avec un coach.
— Augmentation des données : nous avons conservé HorizontalFlip, RandomRotation car ils donnent de meilleurs résultats. Cependant, nous n'utiliserons pas AutoAugmentation(ImageNet) car celui-ci détruit les performances du modèle en introduisant beaucoup trop de bruit (baisse l'accuracy sur test à 10%).
— Scheduler : nous avons pour l'instant décidé de conserver ReduceLROnPlateau qui nous fournit des résultats satisfaisants.
— Dimension des images : Nous avons pour l'instant conservé la taille conseillée par la librairie Resnet18 (224*224).

5 Conclusion de la journée

Nous avons réussi à entraîner un modèle avec une accuracy de 88% sur le test et nous avons commencé à explorer les différentes méthodes d'explicabilité pour comprendre comment notre modèle prend ses décisions. Nous avons également commencé à explorer les hyperparamètres pour améliorer les performances de notre modèle. Nous sommes globalement satisfaits de notre journée et nous avons hâte de continuer à explorer les différentes méthodes d'explicabilité et d'optimisation de notre modèle.

Jour 2

1 Troisième version du modèle

1.1 Modifications des hyperparamètres

— Optimizer : nous avons décidé de passer à SGD puisque celui-ci obtient de meilleures performances qu'Adam (90% d'accuracy sur le dataset de test pour SGD contre 88% pour Adam avec les mêmes paramètres puis 92% avec des améliorations)

— Learning rate : avec le passage à SGD, nous avons changé le learning rate et utilisons à présent 0.01, ce qui est plus adapté au nouvel optimizer.

— Dimensions de l'image : Changer la taille de l'image influence nos résultats et passer à une taille supérieure à '224' (valeur conseillée par la librairie de Resnet18) nous permet d'augmenter notre accuracy sur les données de test. Cela peut être justifié par des valeurs plus proches de la taille moyenne des images et donc moins de pertes et de modifications des informations. Après plusieurs tests, le redimensionnement de 512*512 donne les meilleurs résultats.

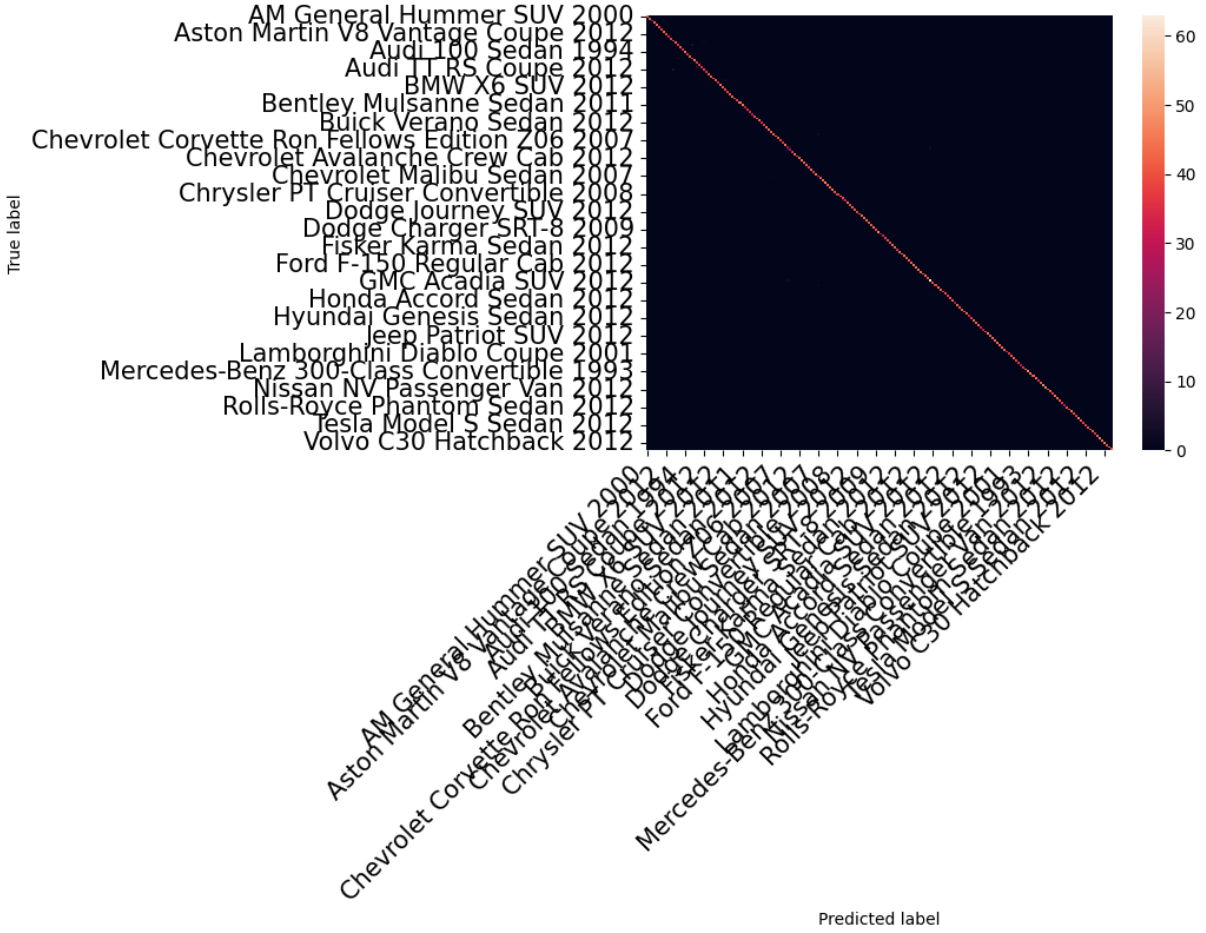
— Taille du dataset de train/ du dataset de test : nous avons décidé de passer à un ratio 80% / 20% plutôt que 50% / 50% pour permettre à notre modèle d'avoir plus de données pour s'entraîner et donc plus de cas différents à voir. Cela nous a permis de passer de 90% d'accuracy à 92% d'accuracy sur le dataset de test.

1.2 Résultats

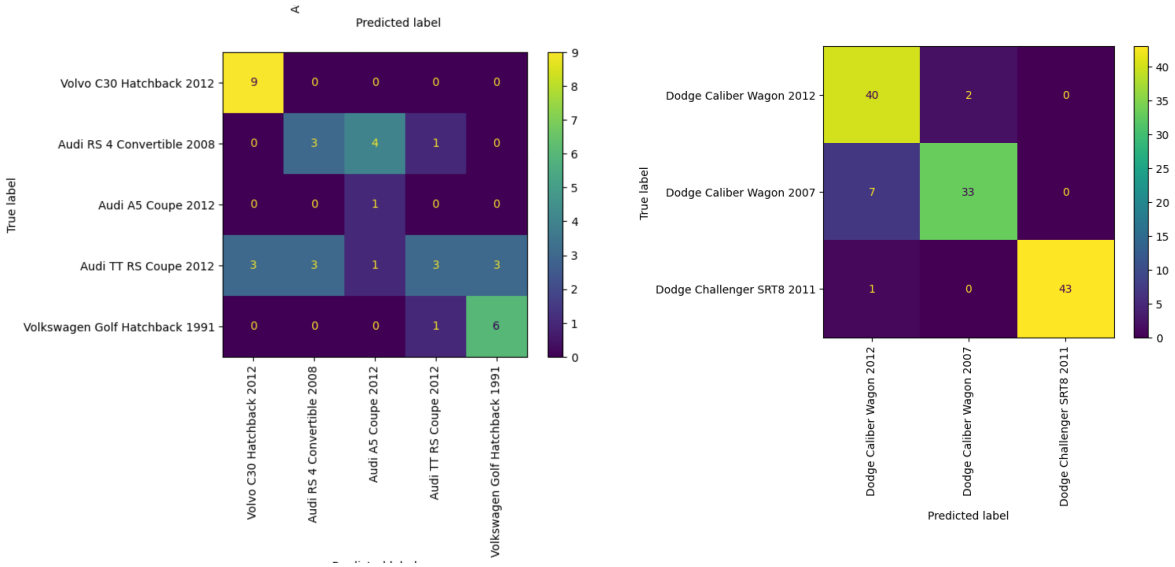
Le modèle a atteint une accuracy de 98% sur les données d'entraînement et une accuracy de 92% sur le test. Les performances du modèle ont été grandement améliorées, mais l'étude des résultats permet de constater des pistes d'amélioration supplémentaires.

2 Étude des résulats

2.1 Matrice de confusion



Pour évaluer les performances de notre modèle, la matrice de confusion peut être un très bon outil. En revanche, le dataset étant composé de 196 classes, il est très difficile de l'exploiter. Cela nous permet cependant de visualiser le bon fonctionnement du modèle sur la majorité des classes grâce à la diagonale.



Pour remédier à ce problème, nous l'avons séparé en multiples sous-groupes en groupant les catégories qui étaient les plus confuses entre elles. Cela permet d'obtenir des sous-matrices de taille 5*5 par exemple et de pouvoir se concentrer sur ces 5 classes qui posent problème. Ce zoom nous permet donc de constater que les classes les plus souvent confondues entre elles sont celles avec des modèles de voitures assez semblables entre eux. Cela comprend les voitures d'une même marque ou d'un même modèle, mais d'années différentes. On peut citer l'ensemble de la marque Audi qui possède des voitures assez similaires entre modèles et au sein d'un même modèle.

Exemple de deux voitures similaires

2.2 Explicabilité

Nous avons pu ensuite utiliser l'explicabilité pour approfondir un peu plus les résultats obtenus par la matrice de confusion. Suite aux recherches réalisées durant le jour 1, nous avons sélectionné deux outils : la librairie Captum pour l'occlusion et la librairie Torch-Cam pour le Grad-Cam.

Nous avons décidé d'utiliser Grad-Cam et non pas SmoothGrad-Cam pour éviter l'impact de la seed présente dans l'utilisation de la deuxième fonction. Cette seed créait bien trop de bruit dans les résultats et impactait donc notre étude de ceux-ci en donnant un résultat parfois totalement différent sur la même image avec le même modèle.

Occlusion

Cette librairie nous permet de constater les zones de l'image qui ont permis d'impacter le plus la décision de notre modèle. Nous pouvons ainsi constater que notre modèle ne semble pas être énormément impacté par le paysage ou par ce qui entoure les voitures et se concentre majoritairement sur la silhouette du véhicule pour faire ses choix.

GradCam

Gradcam nous permet d'obtenir des informations bien plus intéressantes puisque nous pouvons ainsi voir les zones sur lesquelles se concentre notre modèle pour trouver ses caractéristiques.

Si nous reprenons notre exemple des Audi, c'est ainsi que nous pouvons nous rendre compte que le problème avec ces classes vient de la zone observée : notre modèle se concentre principalement sur l'avant de la voiture lorsque celui-ci est visible et, plus particulièrement, sur le logo de la marque. Or, la plupart des voitures de cette marque possèdent un avant et un logo

similaire (celui de la marque). Avec principalement cette caractéristique, il est donc normal d'avoir beaucoup de confusions au sein de ces classes.

3 Quatrième version du modèle

3.1 Amélioration principale

Constater d'où venait le problème pour les classes avec le plus de confusion est intéressant, mais ne change pas grand-chose au problème actuel. Nous pouvons cependant utiliser ces nouvelles connaissances pour améliorer notre modèle.

Pour cette quatrième version, nous allons commencer par entraîner une première fois le modèle avec les données de base puis nous ajouterons quelques images supplémentaires dans notre base de données d'entraînement.

Pour réaliser cette augmentation, nous allons réutiliser les données obtenues à l'aide de GradCam : 20% des images de nos données d'entraînement vont être reprises, la zone désignée comme la plus regardée par le modèle va être floutée et cette nouvelle image sera ajoutée dans les données d'origine. Ainsi, le modèle sera forcé de trouver de nouvelles caractéristiques distinctes parmi ces classes.

Pour réaliser ce flou, nous suivons différentes étapes :

— Une image appartenant aux données d'origine est choisie.

— GradCam est utilisé et nous fourni les points de focus de notre modèle sur cette image.

— Cette image est transformée en un masque en utilisant 10% des pixels les plus utilisés (la zone rouge de la heatmap). Un gaussianBlur est ensuite appliqué sur ce masque.
— Le masque est ensuite appliqué sur l'image pour flouter la zone de focus et l'image est sauvegardée dans nos données d'entraînement.

Le modèle est ensuite entraîné à nouveau avec les nouvelles données.

3.2 Autres améliorations réalisées

Les modifications vues précédemment ne sont pas toujours suffisantes. Certaines des classes possèdent des images de véhicules très similaires et avec très peu de différences. Par conséquent, nous avons décidé d'entraîner notre modèle, mais spécifiquement sur ces cas, pour lui permettre de se concentrer sur la moindre distinction possible.

Les hyperparamètres du modèle sont similaires, mais les données ne sont à présent plus globales, mais spécifique aux classes que nous souhaitons étudier.

Ce modèle spécifique peut se révéler très utile dans le cas où nous ne devrions pas étudier des objets différents, mais bien un seul type d'objet avec des sous-catégories.

Dans notre cas, nous ne voulons plus reconnaître un modèle de voiture parmi tous mais un modèle spécifique d'Audi parmi les classes Audi disponibles. Nous obtenons ainsi beaucoup moins de confusion au sein de cette marque avec ce nouveau modèle spécifique.

3.3 Résultats

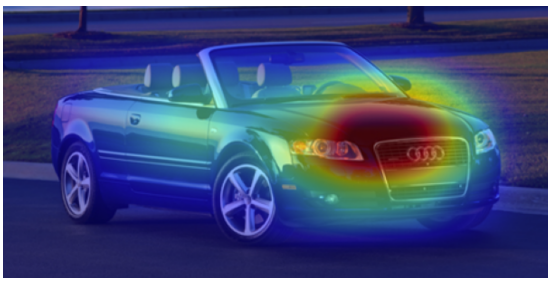


Figure 1 – Heatmap avant le deuxième entraînement du modèle

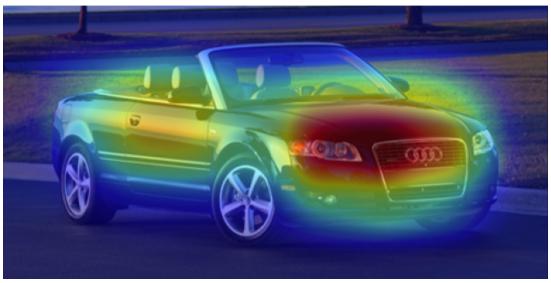


Figure 2 – Heatmap après le deuxième entraînement du modèle

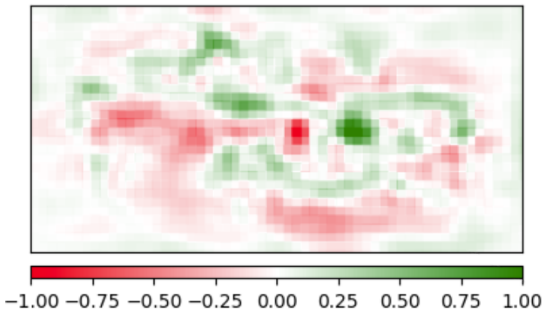


Figure 3 – Occlusion du modèle avant le deuxième entraînement

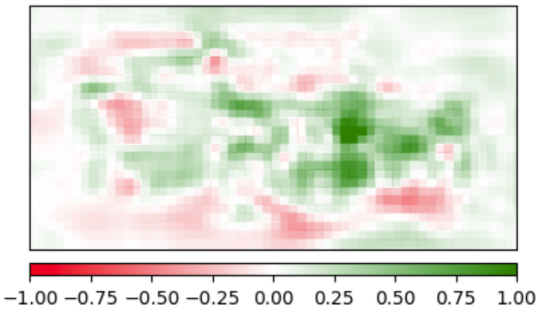


Figure 4 – Occlusion du modèle après le deuxième entraînement

Concernant le deuxième entraînement du modèle, en ajoutant du flou sur certaines caractéristiques, nous pouvons constater que le modèle évolue bien dans ses choix de zone de focus.

Les images ci-dessus permettent de montrer qu'après le deuxième entraînement, le modèle garde sa première caractéristique principale (l'avant de la voiture) mais gagne également un nouveau point de focus (la portière de la voiture). Ces zones sont visibles grâce aux heatmap.

Cette conclusion est également soutenue par l'occlusion : les pixels de la portière qui était considérée comme induisant en erreur notre modèle (diminuant sa certitude concernant la classe qu'il avait choisie comme vraie) sont à présent considérés comme l'aidant à valider sa conclusion.

Les performances sont également un peu meilleures : nous sommes passés de 92.72% d'accuracy à 92.75% sur les données de test.

Concernant le modèle spécifique, nous sommes passés à 95.38% d'accuracy sur les données de test.

4 Conclusion de la journée

Nous avons pu étudier et comparé différentes instances de notre modèle pour trouver des combinaisons d'hyperparamètres nous convenant. Ainsi, nous avons réussi à entraîner un modèle avec une accuracy de 92% sur le test.

L'étude des résultats avec l'explicabilité nous a également permis de comprendre les différentes failles de notre modèle et donc de l'améliorer et de trouver d'autres possibles améliorations à mettre en place ou à expliquer.

À la fin de cette journée, nous finissons donc avec 3 modèles liés les uns aux autres :

— un premier modèle général qui s'entraîne sur les données d'origines
— un deuxième modèle similaire au premier, mais qui s'entraîne sur de nouvelles données obtenues grâce aux résultats du premier modèle
— un troisième modèle qui se spécialise sur les classes avec le plus de confusions entre elles (principalement des Audi dans notre cas)

Jour 3

Le modèle n'a pas évolué durant ce troisième jour.

Cette journée fut principalement consacrée aux livrables. Nous avons donc créé nos diapositives pour les oraux et également fini de rédiger notre journal de bord. Le code a également été nettoyé, commenté et séparé proprement pour permettre une lecture plus simple.

Conclusion

À la fin de ce challenge, nous finissons donc avec trois modèles permettant d'obtenir des résultats de plus en plus précis. Ces trois modèles sont liés entre eux et dépendent les uns des autres. Le fonctionnement général consiste à :

- un modèle général entraîné sur les données de base pour obtenir les résultats d'explicabilité
- le modèle FLOU général entraîné à nouveau sur de nouvelles données (celles de bases + des données floutées). Ces données floutées sont obtenues en utilisant les résultats de GradCam et en floutant les zones de focus du modèle.
- un modèle spécialisé sur les classes qui restent très confuses entre elles après le deuxième entraînement. Le choix de ces classes se base sur la matrice de confusion obtenue après le deuxième entraînement (dans notre cas, sur Audi).

Nous sommes assez satisfaits de notre travail et nous avons pu apprendre de nouveaux concepts tels que l'explicabilité ce qui pourra se révéler utile pour notre futur travail en tant qu'ingénieur en informatique.

Nous tenons également à remercier les coachs pour leur patience et leurs conseils donnés tout au long de ce hackathon.