

Journal de bord Hackathon IA Équipe 3

6milarité - <https://github.com/plfavreau/ia-data-hack>



Samy Hadj-said
Virgile Hermant
Angelo Eap
Pierre-Louis Favreau
Gautier Gally

24 Avril 2024

Sommaire

1 Journal de bord - Jour 1	1
1.1 Objectif du Hackathon	1
1.2 Étapes réalisées	1
1.3 Difficultés rencontrées	2
1.4 Échanges avec les coachs	3
1.5 Prochaines étapes	3
 2 Journal de bord - Jour 2	 4
2.1 Recherche de biais dans la base de données	4
2.2 Matrice de confusion	4
2.3 Modification de la répartition des données	5
2.4 Développement du code	5
2.5 Exploration des hyperparamètres	5
2.6 Augmentation de la précision	6
2.7 Visualisation des zones d'intérêt avec Grad-CAM	6
2.8 Interprétation des Heatmaps de nos différents modèles	7
2.9 Amélioration de l'analyse des erreurs du modèle	8
2.10 Mise en commun	8
2.11 Bilan du jour 2	9

1 Journal de bord - Jour 1

1.1 Objectif du Hackathon

L'objectif de ce Hackathon est d'optimiser les performances d'un modèle de classification basé sur le modèle ResNet-18. Nous travaillerons sur le jeu de données [stanford-car-dataset-by-classes-folder](#) composé d'images de voitures. Notre modèle doit pouvoir déterminer la marque et le modèle d'une voiture pour une image donnée en entrée, à partir de l'architecture du modèle ResNet18.

1.2 Étapes réalisées

- **État de l'art :** Nous avons tout d'abord entrepris des recherches sur le modèle ResNet-18 afin de l'adapter à notre ensemble de données. Venant tous d'EPITA, nous avions déjà acquis des connaissances sur la classification d'images à l'aide de CNN (Convolutional Neural Networks) lors de nos cours, et ayant également réalisé un projet de classification de bateaux à l'aide d'un CNN conçu à la main "from scratch", nous étions déjà familiers avec les principes fondamentaux de ces modèles. Nous avons donc commencé à nous attaquer aux principes de mise en place de ResNet18 en cherchant tous d'abord si nous allions utiliser la bibliothèque pytorch ou tensorflow.
- **Choix de la bibliothèque :** Ayant déjà utilisé Keras de TensorFlow et PyTorch, nous nous sommes mis d'accord sur la bibliothèque à utiliser pour avoir un réseau ResNet-18. Cependant, nous n'avons trouvé qu'un modèle ResNet18 pré-entraîné sur ImageNet disponible sur PyTorch, alors que sur Keras, seul ResNet-50 était disponible, il aurait donc fallu implémenter le modèle de zéro comprenant aussi la partie de l'entraînement depuis le début, ce qui aurait pris trop de temps pour un hackathon d'une journée et demi. Nous avons donc décidé d'opter pour PyTorch.
- **Entraînement initial du modèle :** Pour sélectionner notre modèle de base, nous avons réalisé un entraînement avec le dataset des voitures, sur chacun des modèles issus de nos recherches respectives. Nous avons décidé de retenir le modèle qui offrait la meilleure accuracy sans pré-traitement additionnel, et c'est ainsi que nous avons retenu un modèle d'une accuracy de 89%. Celui-ci nous servira de base pour nos futures expérimentations. La prochaine étape est de comprendre ce pourcentage et d'explorer les moyens d'améliorer les performances du modèle en faisant varier les hyperparamètres, le prétraitement des images, la répartition des données que nous donnons à notre modèle durant l'entraînement, les tests et la validation.
- **Prétraitement des données :** Nous avons également exploré différentes transformations à appliquer sur nos images afin d'augmenter la diversité de notre jeu de données et d'améliorer les performances du modèle. Cela permet d'exposer notre modèle face à des nouvelles images qui ne seraient pas capturées dans les meilleures conditions. Les indicateurs que nous avons mis en place pour accepter une transformation dans notre modèle final sont la loss et l'accuracy de la dernière époque d'un entraînement de 10 itérations. Voici celles que nous avons testées :

- *RandomRotation* : Rotation aléatoire des images pour augmenter la robustesse du modèle face à la rotation des objets dans les images. Résultat : Aucune amélioration notable, perte d'accuracy de 0.5%.
- *ElasticTransform* : Transformation élastique des images pour simuler la déformation des objets dans les images. Résultat : perte de 0.01 sur la précision. Augmentation significative du temps d'entraînement. La transformation n'apporte rien au résultat du modèle nous avons donc décidé de ne pas la garder.
- *GaussianBlur* : Application d'un flou gaussien sur les images pour réduire le bruit et les détails superflus. Résultat : le modèle devient significativement moins performant : une loss instable, une accuracy moins bonne et un temps d'entraînement plus long dû au prétraitement des images.
- *ColorJitter* : Modification aléatoire de la luminosité, du contraste, de la saturation et de la teinte des images pour augmenter la diversité des données. (Résultat : 90% d'accuracy). Nous avons remarqué une légère augmentation de la précision du modèle avec l'utilisation de cette transformation, mais de manière globale, il n'était pas intéressant de l'utiliser, car cela n'a pas apporté les améliorations attendues et a même entraîné une diminution des performances du modèle.
- *RandomAdjustSharpness* : Ajustement aléatoire de la netteté des images pour augmenter les détails. (Résultat : 90% d'accuracy)
- *RandomAutoContrast* : Application aléatoire du contraste automatique sur les images pour améliorer la visibilité des détails. (Résultat : 90% d'accuracy). Malgré l'amélioration de l'accuracy, le temps perdu par les calculs nécessaires à l'application du filtre ne nous a pas semblé satisfaisant et nous ne l'avons donc pas conservé.

1.3 Difficultés rencontrées

- Nous avons rencontré un problème avec le service Onyxia Datalab. Nous avons manqué d'espace sur notre service PyTorch-GPU, et bien que nous ayons tenté de recréer un service avec plus de 10 Go d'espace, le site a continué de tourner indéfiniment car aucun GPU n'était disponible du côté d'Onixya. Nous nous sommes rapidement adapté à la situation en utilisant temporairement la plateforme Kaggle pour poursuivre nos expériences.
- Nous souhaitions tester différents hyperparamètres dans notre processus d'entraînement. Cependant chaque entraînement durait entre 20 et 30 minutes pour 10 époques de notre modèle, nous avons donc décidé de le faire en parallèle. Chaque membre du groupe se concentre sur un hyperparamètre spécifique et travaille à le comprendre. Nous avons ainsi entraîné notre modèle plusieurs fois sur cinq ordinateurs simultanément sur Kaggle. Cependant, lors de ces manipulations, nous avions reçu un bannissement de l'adresse IP du réseau wifi d'Albert-School, un problème que nous avions déjà eu à l'EPITA nous connaissions donc déjà la solution pour y avoir accès à nouveau, il a fallut nous mettre en partage de connexion depuis notre téléphone, ce qui n'était pas possible dans les salles du bas nous avons donc été obligé de changer de salle.

Après avoir récupéré l'accès à la plateforme, nous avons continué au mieux nos expérimentations en alternant nos entraînements entre Onyxia et Kaggle lorsque les services se rendaient disponibles.

1.4 Échanges avec les coachs

Les coachs nous ont aidés à y voir plus clair. Par exemple, bien que nous ayons obtenu une précision de 89%, nous nous sommes demandés si ce pourcentage était significatif. Il est possible que le jeu de données de test soit structuré de la même manière que celui d'entraînement où qu'il ne soit pas assez représentatif de chaque classe. Un coach nous a également conseillé de créer une matrice de confusion pour mieux évaluer les performances de notre modèle. L'échanges avec les coachs nous a permis de dresser une liste de tache pour la journée du lendemain.

1.5 Prochaines étapes

- Analyser plus en détail les résultats du modèle
- Explorer d'autres approches pour améliorer les performances du modèle, notamment en faisant varier les hyperparamètres.
- Créer une matrice de confusion pour évaluer plus finement les performances du modèle.
- Résoudre les problèmes rencontrés avec le service Onyxia Datalab.
- Modifier la distribution de notre jeu de données entre les parties d'entraînement, de test et de validation

Nous avons bien progressé pour cette première journée malgré les quelques obstacles rencontrés. A la fin de cette journée, nous avons constaté qu'ajouter des transformations supplémentaires aux images n'a pas apporté d'améliorations significatives. Certains pré-traitements ont entraîné une diminution des performances du modèle. Nous avons donc décidé de nous tourner vers d'autres méthodes d'optimisation pour améliorer nos résultats.

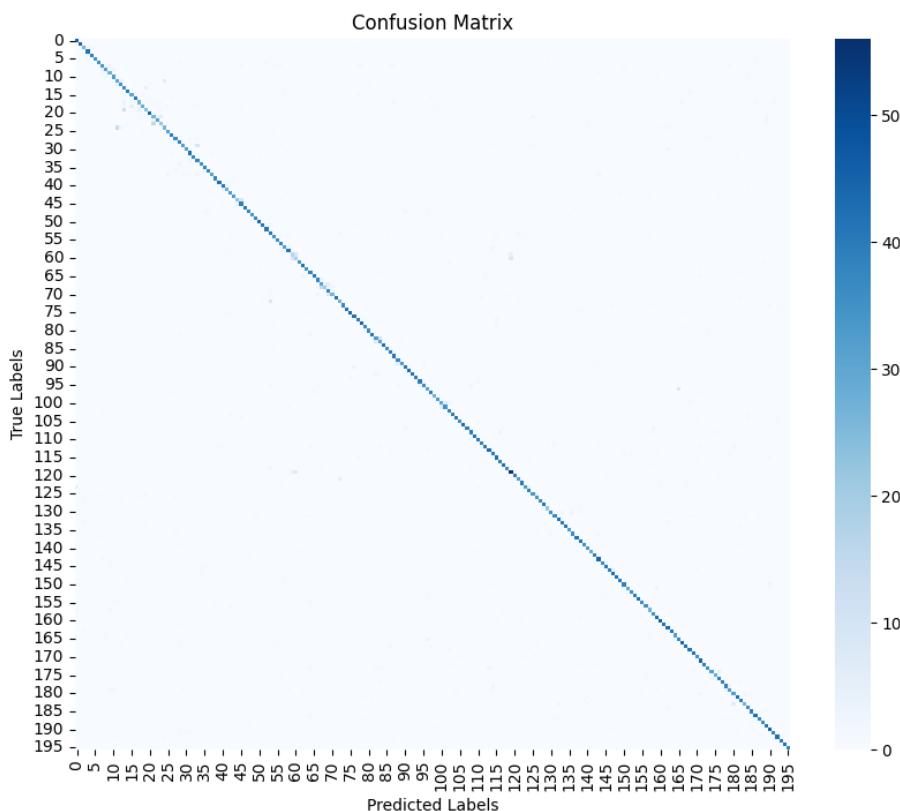
2 Journal de bord - Jour 2

2.1 Recherche de biais dans la base de données

Nous avons exploré plus en détail notre base de données pour nous assurer qu'elle n'était pas biaisée. Nous avons remarqué la présence d'éléments intéressants dans la base de données : notamment, une image de voiture avec plusieurs marques aléatoires écrites au-dessus (pour tester si le modèle se basait uniquement sur la marque et se trompait). De plus, nous avons identifié des images où seuls les pneus ou le bas de la voiture étaient visibles. La répartition des classes est aussi assez bien respecté. Il n'y a pas de différences impactant l'entraînement du modèle.

2.2 Matrice de confusion

Nous avons mis en place une matrice de confusion pour évaluer les performances de notre modèle. Nous avons constaté que la matrice était principalement diagonale, ce qui indique que notre modèle associe correctement la plupart des images au modèle de voiture correspondant.



La diagonale est un indicateur positif concernant les performances de notre modèle. Il était maintenant temps de tenter d'expliquer la cause des mauvaises prédictions qui ne suivent pas cette diagonale.

2.3 Modification de la répartition des données

Nous avons constaté que les tests prenaient beaucoup de temps dû à la quantité de donnée dans le jeu de test. Pour résoudre ce problème, nous avons décidé de modifier la répartition de nos données. Initialement, nous avions une répartition 50/50 entre les données d'entraînement et de test. Nous avons décidé de passer à une répartition 70/30 pour pouvoir tester plus rapidement, mais aussi pour que le modèle s'entraîne sur un plus large échantillon de données. Une plus grande proportion de données d'entraînement permettra également à notre IA de se frotter à des données différentes, ce qui devrait améliorer ses performances et son adaptabilité. En premier lieu nous assignions nos données en fonction de l'architecture du jeu de données : chaque dossier représente une classe et nous avons deux grands dossier de train / test.

Le jeu de donnée a aussi des annotations contenant les labels et les noms de chaque image/donnée. Cependant jusqu'à lors nous n'utilisions pas directement ces fichiers d'annotations. Nous avons donc modifié notre code pour regrouper toutes nos données d'annotations dans un seul DataFrame pandas pour pouvoir ensuite le diviser à notre bon vouloir.

Les difficultés de cette fusion étaient qu'il fut difficiles de bien reproduire le chemin d'accès exacte des images. Les annotations ne fournissent "que" le nom de l'image et pas son chemin relatif à l'emplacement du dataset. Il y a eu également des coquilles dans le nom des fichiers certains avaient des "/" dans les annotations remplacés par des "-" dans l'architecture du jeu de données.

Nous avons réussi à répartir nos données en 70% pour l'entraînement et 30% pour les test / validation. Nous avons gagné 0.01 sur la précision après entraînement. L'augmentation de données pour l'entraînement n'a pas ralenti ce dernier. Et en augmentant certains paramètres des classes au chargements de données nous avons même réussi à accélérer l'entraînement. Nous envisageons d'essayer une répartition 80/20 et même 90/10.

2.4 Développement du code

Malgré les difficultés rencontrées concernant la disponibilité des services Onyxia et Kaggle, nous avons réussi à développer un code fonctionnel. Nous avons implémenté beaucoup de fonctionnalités en échangeant constamment entre nous pour déterminer si ces modifications étaient bénéfiques à la performances de notre modèle.

2.5 Exploration des hyperparamètres

Nous continuons à tester différents hyperparamètres (optimizer, learning rate, nombre d'époques, batch_size ..) et à faire varier leurs valeurs. Jusqu'à présent, le pourcentage de précision le plus élevé que nous avons obtenu est de 85%. Ce score est le résultat de la combinaison précise de ces différents hyper-paramètres déterminés.

2.6 Augmentation de la précision

Cela est encourageant de constater que les ajustements que nous avons effectués ont eu un impact significatif sur les performances du modèle. Atteindre une précision de 85% est une amélioration notable. En modifiant le nombre de workers et la batch size, ainsi que le ratio d'entraînement/validation, nous avons pu optimiser les performances du modèle.

De plus, en augmentant le nombre d'échantillons dans l'ensemble de validation, nous avons probablement obtenu une meilleure estimation des performances réelles du modèle. Ces résultats montrent l'importance de l'expérimentation et de l'ajustement des hyperparamètres dans le processus d'entraînement des modèles d'apprentissage automatique.

2.7 Visualisation des zones d'intérêt avec Grad-CAM

L'une des méthodes les plus utilisées pour visualiser les zones d'intérêt dans une image est la méthode Grad-CAM (*Gradient-weighted Class Activation Mapping*). C'est une technique populaire qui permet de comprendre quelles parties de l'image sont les plus importantes pour la classification.



Nous avons initialement essayé d'utiliser OpenCV (cv2) pour implémenter Grad-CAM from scratch. Bien que l'implémentation soit réussie, nous avons constaté des problèmes avec OpenCV sur Onyxia Datalab. Sachant que les coachs testeraient notre code sur Onyxia, nous avons dû trouver une solution alternative. Nous avons finalement opté pour l'utilisation de `torchcam`, une bibliothèque PyTorch qui offre une implémentation flexible de Grad-CAM compatible avec Onyxia Datalab et Kaggle.

L'image ci-dessous montre un exemple de la superposition de la carte de chaleur sur une image de voiture



FIGURE 1 – Version sans cv2 avec Heatmap, 0.919 val accuracy, 0.7 train 0.3 test

2.8 Interprétation des Heatmaps de nos différents modèles

Cette deuxième heatmap nous a démontré que la modification de l'équilibrage de notre set d'entraînement / de test avait un impact sur la prise de décision de notre modèle lors de la prédiction. En effet, la première heatmap avait été relevée après un entraînement sur une répartition 50/50. Cette version se concentrerait sur le logo et la forme des phares. Avec une répartition 70/30, nous constatons que le modèle tend à généraliser sa prise de décision à partir de la forme des phares mais aussi des roues du véhicules.

2.9 Amélioration de l'analyse des erreurs du modèle

Nous avons aussi implémenté un code permettant de détecter les images sur lesquelles le modèle s'est trompé pendant l'entraînement, en appui de la matrice de confusion discutée précédemment. De cette manière, il est maintenant intéressant d'exécuter Grad-CAM sur ces images pour mieux comprendre les raisons des erreurs du modèle, et savoir si celui-ci généralise certaines d'entre-elles.

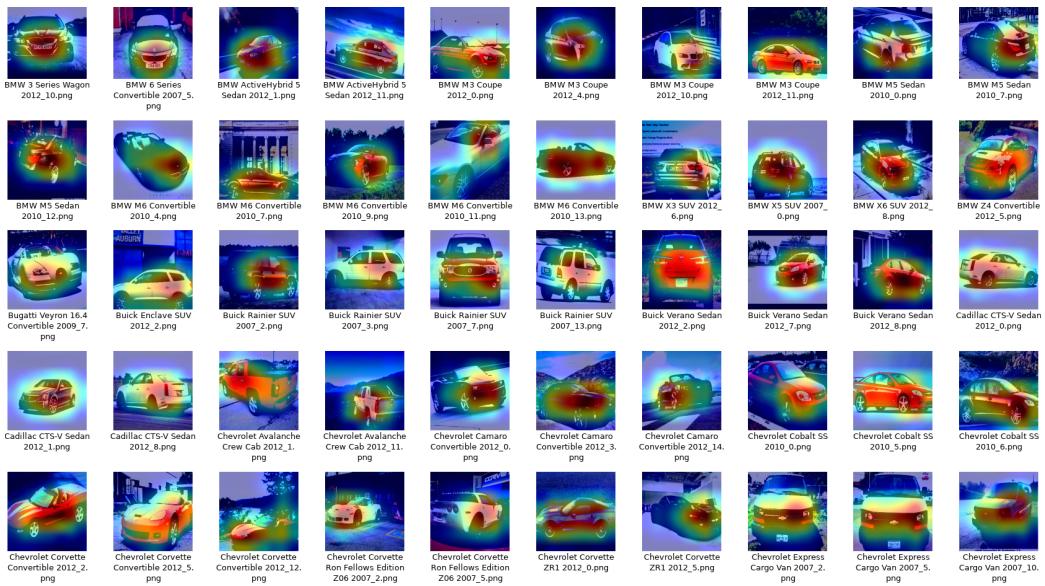


FIGURE 2 – Exemples d’images mal classées par le modèle avec leurs superpositions Grad-CAM correspondantes

2.10 Mise en commun

Pour finir cette deuxième journée, nous avons mis en commun tous nos travaux, et ajouté les meilleurs résultats de nos implémentations respectives dans le notebook qui servira de rendu final. Nous avons également sauvegardé les poids des neurones de notre meilleur modèle. Nous avons aussi ajouté des analyses statistiques sur les résultats et des images modélisant l’explicabilité de notre modèle via les zones d’importance des images.

2.11 Bilan du jour 2

Nous sommes très fiers de notre travail de recherche qui se concrétise en ouvrant des portes sur les différentes optimisations supplémentaires que nous pourrions ajouter à notre modèle pour le rendre encore plus performant.

Par exemple, nous pourrions aller encore plus loin dans l'explicabilité de notre réseau en intégrant dans notre pipeline d'entraînement une correction des poids de neurones en fonction de la GradCAM des prédictions défectueuses. Nous pourrions de même nous recentrer sur le jeu de données en entraînant notre réseau sur beaucoup plus d'itérations, et en constatant un éventuel under-fitting / overfitting.

Il nous reste, pour le jour 3, à faire un travail d'explication et de visualisation de nos résultats. Et aussi nous préparer pour la présentation

Cette compétition a été dans tous les cas très enrichissante. Nous avons appris énormément durant ces 2 jours et demi, tant bien sur le plan technique avec la découverte de techniques de classification avancées pour un CNN, que sur le plan humain avec l'élaboration de stratégies et d'une communication efficace. Nous avons du faire face à de nombreuses difficultés, parfois indépendamment de notre volonté, mais nous nous sommes adaptés en réfléchissant ensemble et de manière organisée.