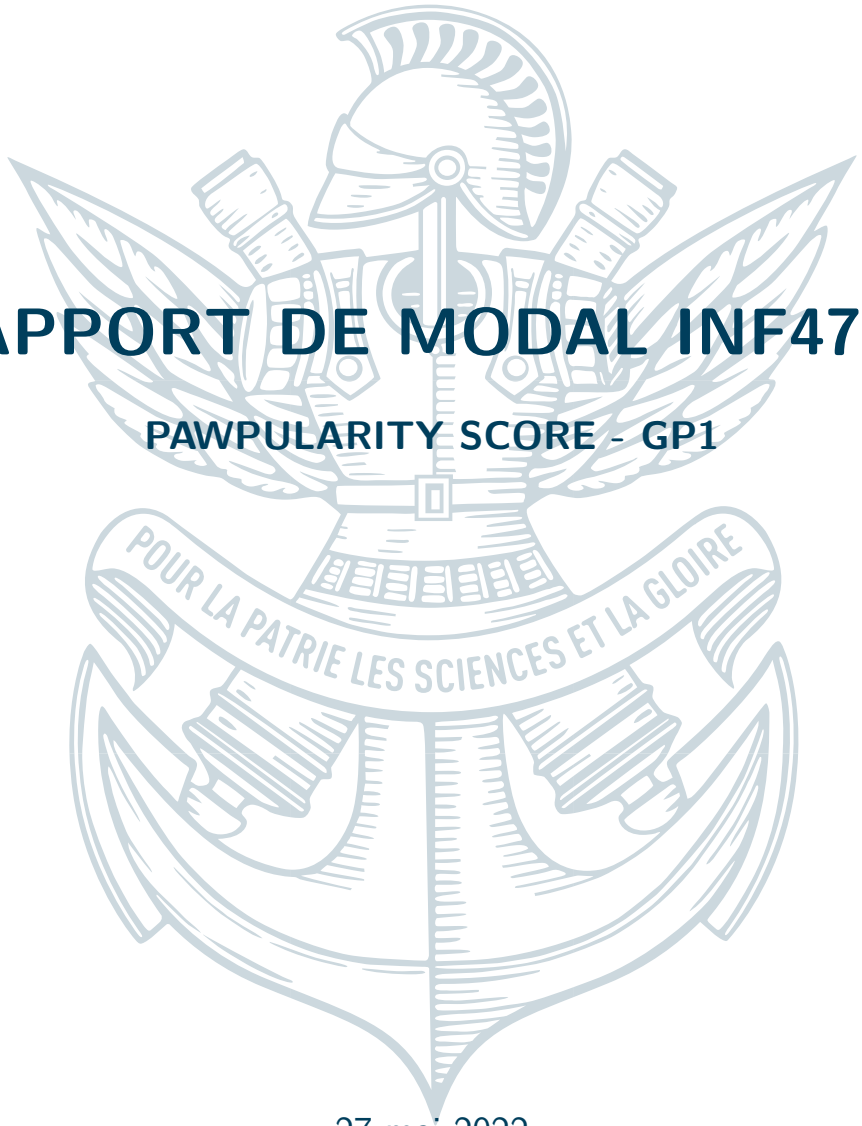

RAPPORT DE MODAL INF473V

PAWPULARITY SCORE - GP1

27 mai 2022

Nicolas PRAT - Vannvatthana NORNG

1 INTRODUCTION

1.1 Description du problème du projet

Petfinder.my est la première plateforme de protection des animaux de Malaisie. Selon cette organisation, un animal abandonné pourra être adopté plus rapidement quand sa photo est plus "mignonne" ("cute"). Pour le moment, Petfinder utilise son outil "Cuteness Meter" pour donner des scores aux photos des animaux. Même si si cet outil se montre déjà utile, ils cherchent encore à améliorer l'algorithme du système qui juge les photos.

Dans ce projet, nous devons donc créer un modèle qui reçoit une photo d'animal de compagnie comme input, et renvoie à l'utilisateur un score appelé "Pawpularity Score", censé quantifier la qualité de la photographie dans un contexte de recherche de nouveau propriétaire pour l'animal. Un objectif secondaire est alors d'aider l'utilisateur à prendre de meilleures photos, notamment en lui donnant des conseils sur la façon d'améliorer ses prises de vues.

1.2 Données disponibles

Dans la compétitition Kaggle de Petfinder.my, 9912 photos de chiens et de chats sont données dans le folder "train". Les 8 exemples dans le dossier "test" ne sont pas utilisables : il s'agit seulement d'indications sur le format type des données de test (et d'utilisation). On dispose également d'un fichier au format csv qui donne des informations sur chaque photographie :

- Des métadonnées consistant en 12 features (Focus, Eyes, Face, Near, Action, Accessory, Group, Collage, Human, Occlusion, Info, Blur), ayant pour valeur 0 ou 1
- Le "Pawpularity Score", entre 0 et 100

1.3 Approche de résolution

Notre première priorité est, bien entendu, la prédiction du score de "PawPularity". En effet, avoir déjà une appréciation automatique de la qualité d'une photographie est utile en soi, en ce que cela permet à l'utilisateur de "se situer", et éventuellement, par la soumission de plusieurs photographies, de comprendre (par lui-même) comment s'améliorer. Ainsi nous avons commencé par construire des réseaux neuronaux relativement classiques de régression afin de prédire le "PawPularity Score" à partir des images, sans ignorer les améliorations de performances que peut apporter le *transfer learning*, autrement dit la reprise d'une partie d'architectures classiques déjà entraînées.

Néanmoins, cela n'est pas suffisant : l'objectif est justement de proposer à l'utilisateur un service, en l'aidant à améliorer ses photos en profitant de l'apprentissage statistique... pour qu'il n'ait pas à apprendre par lui-même. Ainsi, l'usage des features, critères humainement interprétables et qui constituent potentiellement des critères de beauté des photographies dans une évaluation manuelle,

pourront permettre de donner des indications à l'utilisateur sur la ou les manières d'améliorer ses photographies en prenant en compte leur influence sur le calcul du score.

Puisque les données d'entrée de l'utilisateur consistent uniquement en une image - les features sont elles issues d'un étiquetage manuel des images de l'ensemble d'entraînement - il faut que la prédiction finale ne repose pas sur l'apport de features, alors même que ceux-ci favorisent l'interprétabilité du modèle et du score ! Autrement dit, on a ici un double apprentissage : il faut apprendre à indiquer les features appropriées pour une image donnée, afin de donner des conseils d'amélioration du score final en relation avec ces features.

2 MÉTHODES

2.1 Data augmentation

En comparaison avec des ensembles de données célèbres tels qu'ImageNet ou même, à plus petite échelle, CIFAR-10, les 9912 photographies annotées disponibles semblent un nombre relativement faible. Par conséquent, nous avons eu recours à de la *data augmentation* en faisant un flip vertical pour chaque photographie, puis on les mélange avec les données originales pour construire le dataset. Ce besoin d'augmentation de la taille de la base de données est renforcé par la réduction des données d'entraînement qui intervient de fait puisque nous avons utilisé 20% des données pour servir d'ensemble de validation.

Nous avons cependant limité à cela la *data augmentation* réalisée, ce qui nous limite mécaniquement à un facteur d'augmentation de 2. En effet, nous avons estimé que c'était la seule transformation qui donnait non seulement une image encore "plausible" dans le contexte de la photographie animalière, mais surtout qui ne risquait pas trop de trahir les *features* et le *PawPularity Score*, que nous avons naturellement copiés de l'originale à l'image retournée. Par exemple, une symétrie par rapport à un axe autre que vertical semble malvenue pour le réalisme et la qualité de l'image, donc pour le *Pawpularity Score* (un chien pris en photo à l'envers risquerait de moins attirer l'oeil), tandis qu'un changement de luminosité pourrait être corrélé à un changement du *PawPularity Score*, dans un sens ou dans l'autre.

2.2 Comment réaliser plus simplement des réseaux de neurones d'architecture semblable ?

Afin de renforcer la modularité et de faciliter la création de nouveaux réseaux de neurones d'architectures plus ou moins semblables (sans avoir à écrire une nouvelle classe à chaque fois, notamment), nous avons décidé de créer des classes capables de représenter différentes architectures, et avec des constructeurs suffisamment développés pour permettre une certaine modularité par la création de réseaux assez différents en tant qu'instances d'une même classe. Certaines classes sont associées à des fonctions réalisant un pré-traitement des arguments en vue d'appeler le constructeur avec tous ses arguments sans que l'utilisateur n'ait nécessairement à tous les connaître.

Une première fonction permet, à partir d'un nom de modèle classique parmi AlexNet et les ResNets, et de dimensions souhaitées en sortie de couches convolutives, de réutiliser la partie convolutive de ce modèle, ainsi qu'éventuellement sa première couche entièrement connectée. On peut également spécifier si l'on souhaite utiliser le modèle pré-entraîné (dont on ne conservera donc qu'une partie) dans une optique de *transfer learning*, et dans ce cas spécifier si l'on souhaite ou non continuer à entraîner ce modèle par la suite. Plus précisément, on importe un modèle de `pytorch`, puis, selon si l'on a demandé un ResNet ou AlexNet, on remplace le module de couches entièrement connectées par la première de ces couches ou par un module vide (`nn.Sequential()` en l'occurence) suivant le choix exprimé quant à la conservation de cette couche ou non. La couche d'*adaptive pooling* est remplacée par une couche donnant les dimensions souhaitées (exprimées en argument de la fonction). Pour finir, si l'on a choisi de ne pas continuer à entraîner ce modèle, on désactive la backpropagation des gradients sur ses paramètres.

Une classe d'"assemblage" permet alors, à partir d'un modèle censé comporter des couches convolutives, de la taille de sa sortie, et d'une liste de tailles de couches ainsi que de la taille de la sortie globale souhaitée, de créer un modèle intégrant : d'abord le premier modèle à couches convolutives, débouchant sur des couches entièrement connectées décrites par la liste, donnant une sortie comme spécifié. À partir de la taille de la sortie du modèle à couches convolutives, prise comme taille d'entrée pour la suite, on parcourt alors la liste des tailles de couches entièrement connectées : à chaque élément, on ajoute à une liste de couches linéaires en formation la couche reliant la taille d'entrée précédemment notée à la taille de sortie donnée par l'élément actuel de la liste parcourue, élément que l'on affecte ensuite comme prochaine taille d'entrée ; autrement dit la sortie est l'entrée de la prochaine couche. À la fin, cette liste est *unpacked* dans un module séquentiel, et la taille d'entrée restante sert de taille d'entrée de la dernière couche, qui débouche sur la taille de sortie finale décidée.

Associée à cette classe et pour faciliter son utilisation, une fonction a pour objet de réaliser la construction complète d'un modèle comportant un morceau d'architecture classique et terminant sur des couches entièrement connectées. Dans l'objectif d'être modulable, elle prend comme paramètre une "spécification" d'un modèle à couches convolutives qui peut être une chaîne de caractères (le nom d'un modèle classique), auquel cas la fonction d'extraction de couches convolutives d'un modèle classique est appelée, ou bien un modèle déjà existant. Dans le cas des modèles AlexNet et ResNet (dans ses différentes versions), la taille de la sortie de couches convolutives et de l'éventuelle première couche entièrement connectée est connue. Dans le cas contraire, la fonction fait passer un tenseur-test dans le modèle et considère simplement sa forme en sortie, pour déterminer la taille de sortie du modèle. Pour finir, la fonction appelle le constructeur de la classe d'assemblage précédemment présentée pour renvoyer le modèle produit.

Afin de permettre la création de réseaux entièrement connectés (et sans partie convolutive) facilement, nous avons également créé une classe qui, inspirée de la classe d'assemblage, reprend simplement le principe de création et de juxtaposition de couches entièrement connectées "à la demande".

2.3 Modèle 1 : prédiction directe de PawPularity score

Nous avons commencé par rechercher ce qui, dans nos connaissances et les architectures les plus "habituelles", pourrait être adapté à la situation. Le problème principal est le suivant : à partir d'une

image, il faut prédire un score, en se basant sur la manière dont les images d'entraînement sont notées. Autrement dit, c'est (en première approche) un simple problème de régression à partir d'images. Étant donnée la place des réseaux de neurones convolutifs en vision depuis les travaux de Yann Le Cun, il semble évident qu'il faut utiliser une telle architecture. Nous avons donc envisagé plusieurs architectures de ce type :

Tout d'abord, nous avons commencé par entraîner un réseau aux couches convolutives issues d'AlexNet, combiné à 2 couches entièrement connectées. Néanmoins, il semble évident qu'il est assez irréaliste d'espérer bien entraîner un réseau AlexNet, qui possède des millions de paramètres, avec un ensemble d'entraînement comportant moins de 20000 échantillons.

Le *transfer learning* peut alors apporter une réponse à ce problème d'un entraînement trop compliqué avec si peu de données. C'est pourquoi nous avons également utilisé un AlexNet pré-entraîné dont nous avons extrait les couches convolutives. La question est alors de savoir s'il vaut mieux continuer à entraîner lesdites couches convolutives pendant la phase d'entraînement, ou bien entraîner seulement les couches entièrement connectées qui les suivent.

Ne pas ré-entraîner ces couches convolutives présente des avantages : puisque la backpropagation et les étapes d'évolution des paramètres ne se produisent pas dans les couches convolutives, le temps d'entraînement est réduit - cela se vérifiera de manière générale par la suite entre les modèles dont les couches convolutives sont ré-entraînés et ceux dont on a arrêté l'entraînement de ces couches, avec des facteurs d'environ 3/5, ce qui n'est pas négligeable. De plus, une raison plus "métaphysique" serait de penser qu'en arrêtant leur entraînement, on ne "trahit" pas leur pré-entraînement (sur ImageNet), qui est réputé être très bon et présenter des features particulièrement utiles à l'analyse de photographies. En revanche, en ré-entraînant ces couches, on peut espérer une forme de spécialisation dans les tâches d'analyse pour lesquelles le modèle sera utilisé.

Ces différents modèles, semblables par leur traitement linéaire, de l'image à la prédiction du score, sont des modèles de régression relativement simples à imaginer à partir d'architectures classiques... mais ils font l'impasse sur les métadonnées fournies avec les données d'entraînement ! Néanmoins, au-delà de leur fonction de prédiction en tant que telle, ils nous ont permis, ne serait-ce que dans l'énonciation du type d'architecture classique envisagé, de commencer à voir ce qui pouvait être amélioré pour prendre en compte l'attendu de génération de recommandations pour améliorer la qualité des clichés.

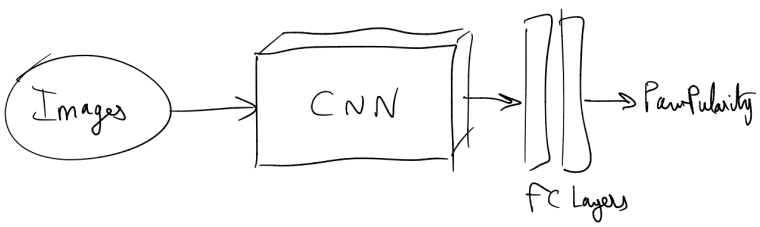

Figure 1 – Schéma du premier modèle

2.4 Modèle 2 : prédiction de features + prédiction du score en fonction des features

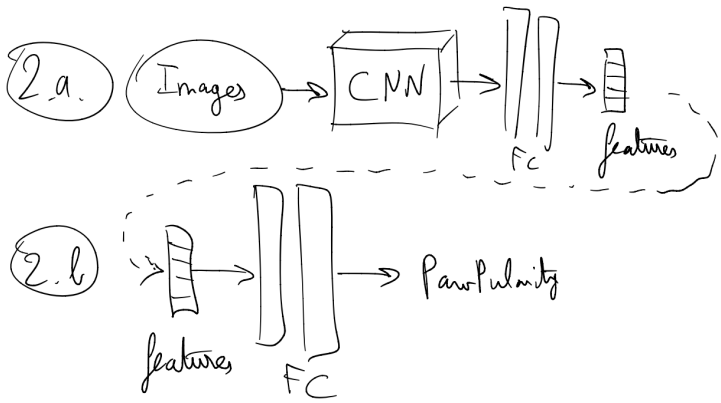

Figure 2 – Schéma du deuxième modèle

Dans ce (type de) modèle, on voit le processus d'évaluation d'une photographie en deux temps : d'abord, on estime les *features* de la photographie (par rapport à la liste de catégories donnée), puis on cherche à prédire le *PawPularity Score* en fonction de ces features.

Dans la partie "A" du modèle, autrement dit celle consacrée à la prédiction de features à partir d'une image, on reprend une architecture très sembable à celle utilisée précédemment.

Néanmoins, il ne s'agit plus là de régression pour prédire le scalaire qu'est le *PawPularity Score*, mais d'une forme de classification binaire multi-dimensionnelle (multi-label classification). La fonction de perte de moyenne de l'erreur quadratique, qui était tout indiquée pour la prédiction du score puisque c'est l'erreur qui était mentionnée dans l'énoncé pour évaluer les prédictions, ne semble alors plus très adaptée, notamment du fait qu'ici des valeurs en-dehors du segment [0,1] n'auraient pas vraiment de sens, alors que le réseau de neurones n'a pas *a priori* de raison de ne pas prédire en-dehors de ce segment. On peut alors le laisser prédire un réel quelconque et appliquer une sigmoïde si l'on veut une valeur dans [0,1] : tout réel positif donnera une image supérieure à 0,5, tandis que tout réel négatif donnera une image inférieure à 0,5. Ce cas de figure correspond à la fonction de perte d'entropie croisée binaire (Binary Cross Entropy), lorsqu'on a précédemment appliqué la sigmoïde, ou à une fonction qui réalise cette opération du même coup (ce qui permet de gagner en stabilité numérique d'après la documentation de `pytorch`. Comme on peut le voir sur la figure ci-dessous, la fonction pénalise quasi-linéairement les *outputs* du réseau qui ne sont pas du bon signe par rapport à la catégorie véritable de l'échantillon d'entrainement, et donne rapidement des valeurs proches de 0 lorsque l'*output* est du bon signe.

FIGURE 3 – La fonction de perte BinaryCrossEntropy (en rouge) pour une catégorie-label valant en réalité 1

On envisage ici, pour la partie A, des architectures semblables à celles envisagées précédemment, en modifiant évidemment la dimension de la sortie. On envisage notamment l'utilisation de couches convolutives pré-entraînées de ResNet, avec et sans continuation de leur entraînement.

En ce qui concerne la partie B du modèle - tâchant de prédire le score à partir des features -, elle est *a priori* bien plus simple. On construit pour cette partie des modèles entièrement connectés à partir de la fonction modulable dédiée. On envisage des structures [200, 200], [120, 120, 120] ou encore [200, 200, 200, 200] pour tester les effets de la profondeur du réseau sur la précision des prédictions (à nombre de paramètres similaire pour les deux premières architectures). Il n'y a pas de doute à avoir ici sur la fonction de perte, qui est l'erreur quadratique moyenne puisque cette perte est associée au score.

L'utilisation du résultat intermédiaire que constituent les features permet, lors de l'assemblage des deux parties du prédicteur, de donner à ces features calculées un statut de variable (en activant le calcul du gradient pour elles). Ainsi, en étudiant le signe de la dérivée partielle du score (prédit) par rapport à chaque feature (prédite), on est en mesure de formuler des conseils : si la dérivée partielle est positive, cela signifie qu'augmenter la feature en question devrait améliorer le score, et inversement si la dérivée partielle est négative.

Néanmoins, on se convainc aisément que le modèle 2 dispose en fait d'informations assez appauvries : les images, dans la première partie du modèle, sont résumées à 12 variables booléennes. Cela constitue un goulot d'étranglement dans le traitement des données. Notamment, 2 images ayant exactement les mêmes features pourront avoir des notes différentes, du fait d'autres critères non-explicités et éventuellement inconscients. Et étant donné le nombre d'images, par principe des tiroirs, il y aura, même avant *data augmentation*, nécessairement des images ayant les mêmes features.

2.5 Modèle 3 : prédiction de PawPularity par features + image réintroduite

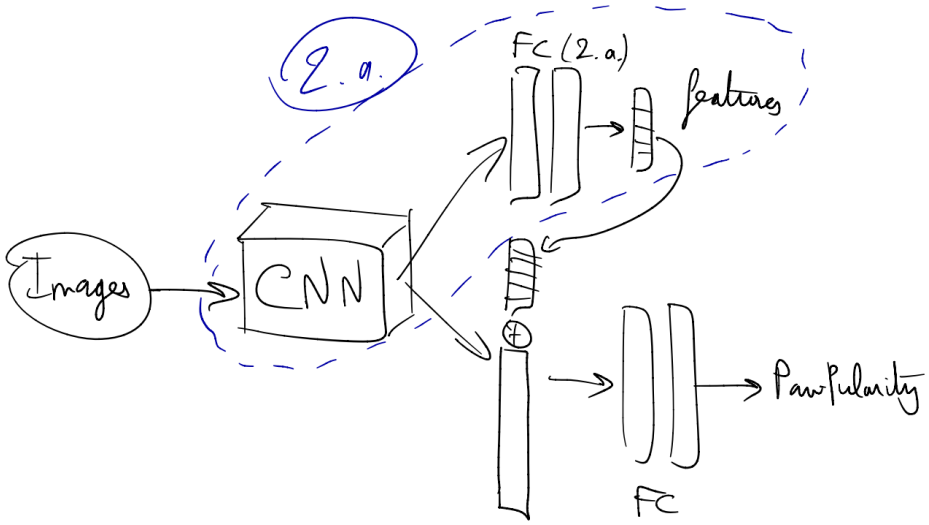

Figure 4 – Schéma du troisième modèle

La structure du modèle 3 est créée afin d'éviter le goulot d'étranglement du type de modèle 2. En prenant une image comme entrée, les données empruntent deux chemins différents. L'image est d'abord analysée par des couches convolutives, puis leur sortie, vectorialisée, est doublement utilisée : d'abord, elle est traitée comme dans le modèle 2.A, pour en tirer des *features*, tandis que cette sortie vectorialisée est conservée à part. Ensuite, la sortie vectorialisée et les features prédites sont concaténées avant de passer des couches entièrement connectées (FC) aboutissant à la prédiction du score.

En réalité, on utilise un modèle 2.A déjà entraîné pour reconnaître les features. En particulier, on utilise ses couches convolutives pour la partie convolutive du modèle 3.

On envisage alors deux types d'utilisation des features : sous une forme "binaire" ou "hard", où l'on effectue réellement une classification stricte, ainsi que sous une forme "soft" où l'on conserve une sorte de nuance, permettant notamment des conseils d'amélioration plus progressifs.

3 RESULTATS

3.1 Modèle 1

Nous avons essayé à entraîner le modèle 1 avec 5 CNN différents pour trouver lequel donne le meilleur résultat.

— AlexNet non-pré-entraîné
— AlexNet pré-entraîné que nous entraînons encore avec nos donnée
— AlexNet pré-entraîné que nous n'entraînons plus et que nous n'entraînons que la partie FC
— ResNet pré-entraîné avec le même comportement que les 2 AlexNet pré-entraîné

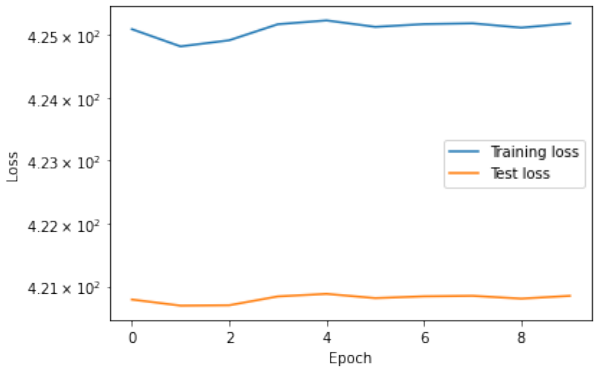

FIGURE 5 – Modèle 1 avec AlexNet non-pré-entraîné

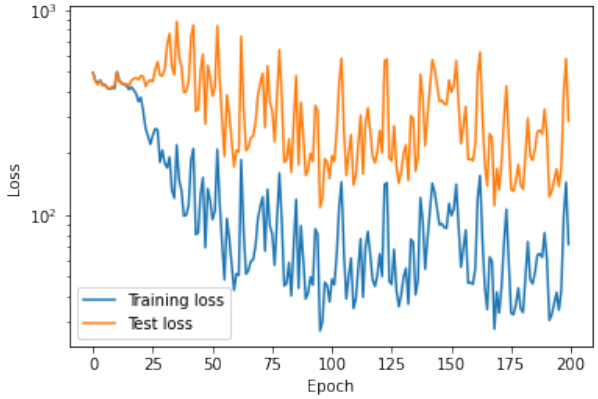

FIGURE 6 – Modèle 1 avec AlexNet pré-entraîné qui continue à s'entraîner

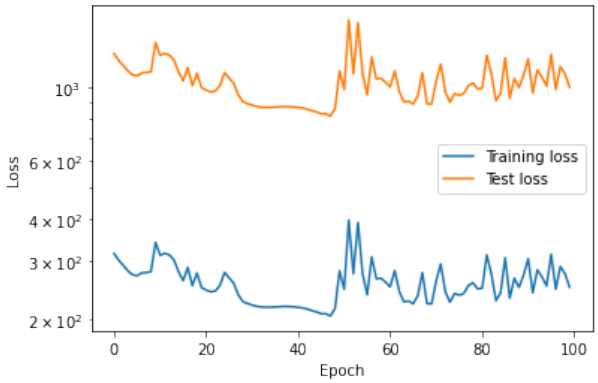

FIGURE 7 – AlexNet pré-entraîné qui arrête à s'entraîner

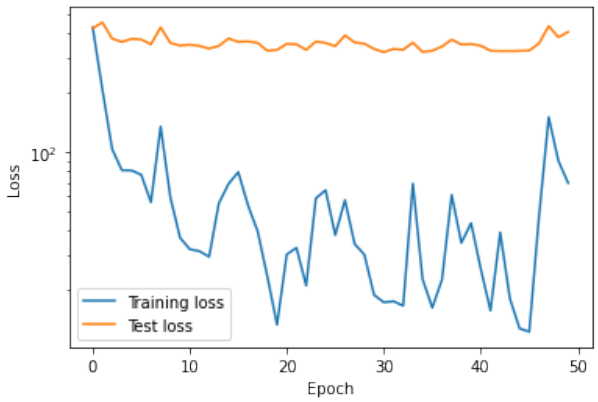

FIGURE 8 – ResNet pré-entraîné qui continue à s'entraîner

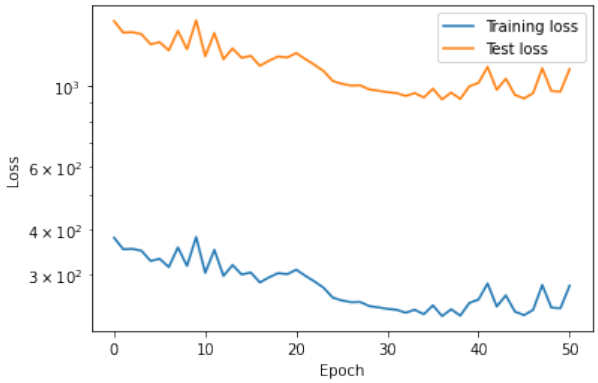

FIGURE 9 – Modèle 1 avec ResNet pré-entraîné qui arrète à s'entraîner

En regardant les graphe ci-dessus, nous choisissons le Modèle qui consiste à ResNet pré-entraîné qui continue à s'entraîner avec nos donnée pour la partie CNN car il donne le meilleur résultat. (Sa courbe de fonction de perte descend plus rapidement que les autre courbe)

3.2 MODEL 2

3.2.1 • Partie a du réseau

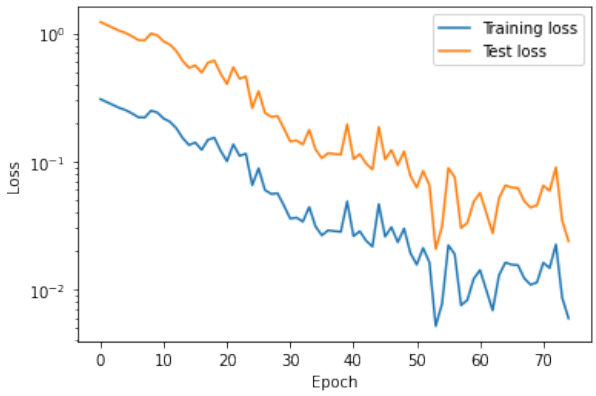

FIGURE 10 – Partie a avec ResNet pré-entraîné qui continue à s'entraîner

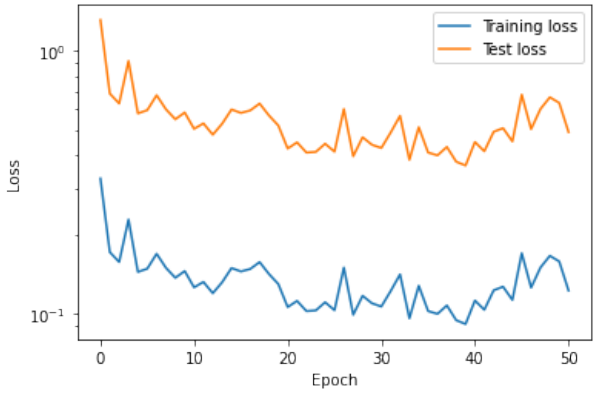

FIGURE 11 – Partie a avec ResNet pré-entraîné qui arrête à s'entraîner

Cette fois, nous avons choisi seulement le ResNet pré-entraîné pour la partie CNN (l'un entraîne encore avec la partie FC, et l'autres n'entraîne pas et seulement la partie FC est entraînée), et nous voyons qu'il la courbe de la fonction de perte pour le ResNet qui continue à s'entraîner donne le meilleur résultat

3.2.2 • Partie b du réseau

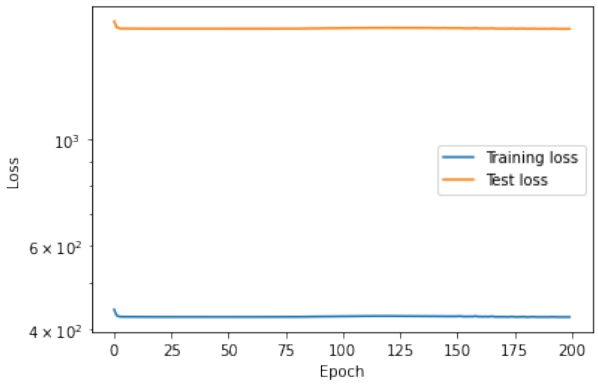

FIGURE 12 – FC de couche [200,200,200,200]

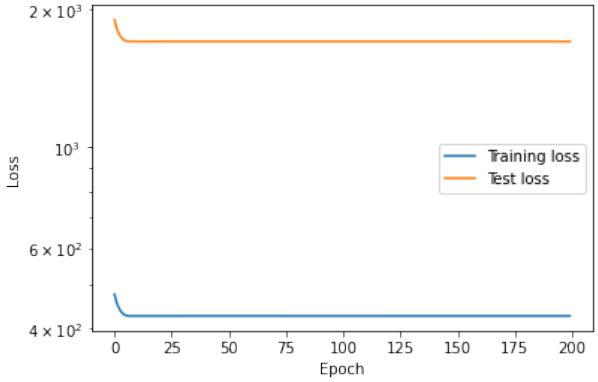

FIGURE 13 – FC de couche [200,200]

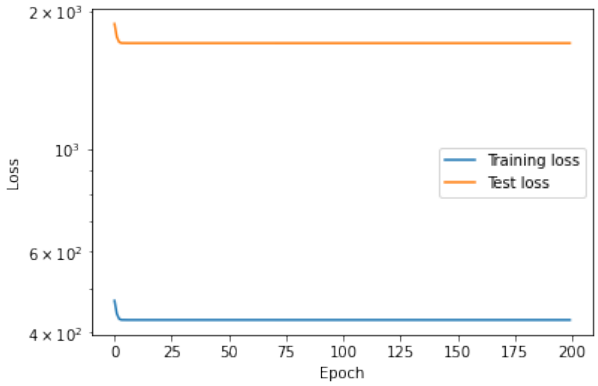

FIGURE 14 – FC de couche [120,120,120]

Ces courbes de fonctions de perte en fonction du nombre d'epoch sont particulièrement plates... Malgré les différentes architectures de réseaux entièrement connectés testées, il n'y a plus de réduction sensible de la perte au bout de quelques epochs. Cela nous conforte dans l'intuition que le goulot d'étranglement que représente la réduction des informations d'une image à son vecteur de 12 features mène à une perte importante de la précision dans la prédiction du score.

3.2.3 • La prédiction de score du Modèle 2 entier

```
In [71]: net2(input_a[0].cuda().view((1,3,224,224)))

Here is some advice to improve your photograph, by decreasing order of influence :
        Decrease the criterion : Blur
        Decrease the criterion : Eyes
        Increase the criterion : Face
        Increase the criterion : Accessory
        Decrease the criterion : Info
        Increase the criterion : Group
        Increase the criterion : Collage
        Decrease the criterion : Subject Focus
        Increase the criterion : Near
        Increase the criterion : Human
        Decrease the criterion : Action
        Decrease the criterion : Occlusion
```

FIGURE 15 – Les conseils d'amélioration sur une photo

3.3 Model 3

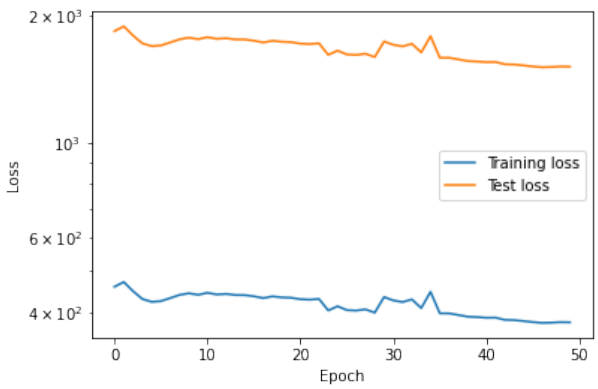

FIGURE 16 – La courbe de fonction de perte du modèle 3 avec softFeatures

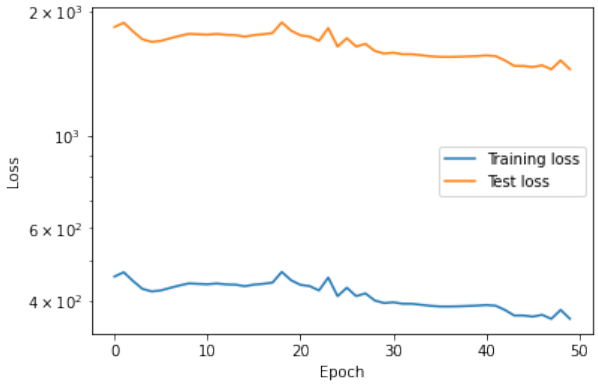

FIGURE 17 – La courbe de fonction de perte du modèle 3 avec hardFeatures

Nous voyons que la performance du modèle 3 avec et sans la classification intermédiaire stricte (softFeatures et hardFeatures respectivement) sont semblable.

4 CONCLUSION

Pour conclure, nous avons procédé en trois temps, afin de prendre croissamment en compte les différentes facettes du problème de PetFinder.my. En commençant par une "simple" régression pour prédire le score de popularité des photographies à partir de celles-ci, nous avons ensuite scindé le processus en deux parties successives : la prédiction de *features* caractérisant les images de manière facilement interprétable, suivie d'une prédiction du score à partir de ces (seuls) critères, afin de permettre des recommandations automatiques d'amélioration des prises de vues. Enfin, pour pallier au problème de goulot d'étranglement posé par ce processus linéaire qui réduit une image à 12 caractéristiques booléennes, nous avons réintroduit la prise en compte de l'image elle-même dans la prédiction du score tout en prenant en considération les *features*.

Il y a encore des points d'amélioration qui peuvent être réaliser dans nos méthodes proposés. D'abord, la luminosité de la photo peut joue un rôle important pour produire une photo de bonne qualité. Nous pouvons prendre en compte "Luminosity" comme une feature dans le Metadata.

Ensuite, le méthode cross-validation peut être appliqué dans la constructions de données train et test. Puisque nous n'avons que le nombre limité de données à utiliser, en appliquant ce méthode, nous pouvons entraîner les modèles avec tous les données disponibles. En plus de cela, nous pouvons également utiliser des technique de normalisation des données (par exemple, Dropout, batch normalisation, etc.).

Par rapport au réseau, la performance peut être améliorer en utilisant un réseau plus grand selon la taille des couche et le profondeur. Néanmoins, il faut faire attention au temps d'entraînement et la taille qu'il prend dans la mémoire. Le réseau plus grand risque aussi de tomber dans la minimisation locale.

Enfin, la solution la plus simple est d'entraîner le modèle avec énermément plus d'époch (par exemple 1000 épochs), mais cela peut risquer à tomber dans le cas de surapprentissage.