Screening and Diagnosis of esophageal cancerfrom in-vivo microscopy images

Youssef AIT SI
Ecole Centrale de Nantes
youssef.amce@gmail.com

Mehdi El Haylali
Ecole Centrale de Casablanca
mehdi.el.haylali@gmail.com

Abstract

La classification des images médicales joue un rôle important dans la médecine d'aujourd'hui. A cause de la taille croissante des données cliniques, il est primordiale de proposer des algorithmes pour assisster les medecins dans le traitement de ces données. En particulier, dans ce projet nous nous intéressons à l'assistance des médecins lors du dépistage de l'oesophage de Barrett. Au cours de cette procédure, les images bougent rapidement et les médecins doivent traiter beaucoup d'informations en même temps, ce qui rend parfois difficile de ne manquer aucune information de valeur. Dans ce cadre, un classificateur d'images pour aider les médecins dans le dépistage et le diagnostic du cancer de l'œsophage est proposé. Il consiste à classifier des images microscopiques afin de détecter celles qui contienent le cancer. Dans ce contexte, on propose d'utiliser des réseaux de neurones convolutionnel afin de construire un classificateur d'images pour aider les médecins dans le dépistage et le diagnostic du cancer de l'œsophage. Notre méthodes consiste à utiliser un réseau de neurones (e.g. ResNet [5], VGG [11], SqueezeNet [6]) pré-entraîné sur ImageNet [3] comme brique de départ pour construire le classificateur, ensuite le modèle est entraîné en utilisant une version équilibrée de la perte "cross-entropie". Finalement, on propose un post-processing qui prend en compte le fait que les images provenant d'un même individu ont en général la même classe. Notre méthode a démontré son utilité sur la première partie de l'ensemble du test: on atteint une "accuracy" de 97% qui nous a mis dans la quatrième position dans le classement général du data challenge.

1 Introduction

Au cours des dernières années, en raison de l'apparition des dispositifs informatiques puissants tels que les Graphic Processing Unit (GPU) qui proposent plus de puissance de calcul, la présence des ensembles de données à grande échelle tels que ImageNet [3] ainsi qu'avec le développement de nouveaux modèles et algorithmes tels que les Convolutional Neural Networks (CNN) [8], l'intelligence artificiel et l'apprentissage automatique ont vécu une grande amélioration pour atteindre des performances proche des performances humaine dans plusieurs domaine: AlphaGo [10] a vaincu le champion du monde dans le jeux compliqué de Go, AlphaStar [1] a démontré des performances similaires à celle des joueurs professionnels du StarCraft, ResNet [5] surpasse les humains dans la tâche de classification des images sur le jeux de données ImageNet [3]. L'apprentissage automatique et l'intelligence artificiel proposent aussi des outils d'assistance au êtres humains dans leurs vie quotidienne, tels que les assistants vocaux, les moteurs de recherche, les voitures autonomes et les robots industriels.

Dans ce contexte que le "data challenge" de Mauna Kea à été élaborer. En effet,Ce Défi de données est un concours de pointe pour les étudiants scientifiques enthousiastes qui veulent mettre en valeur leurs compétences analytiques et techniques. Le but c'est de développer un algorithme pour classifier les images médicales afin de détecter celles qui contiennent le cancer.

Dans ce rapport, on va définir premièrement les réseaux de neurones profonds et les réseaux de neurones convolutionnels, ensuite on va visualiser dans la section 3 les images d'entraînement, dans la section 4 on va présenter la démarche de résolution de notre problème en définissant les modèles et leurs hyperparamètres, finalement on va montrer les résultats obtenu pour chaque modèle et proposer des améliorations et finir par une conclusion et perspectives.

2 Réseaux de neurones

2.1 Deep feed forward network

Deep FeedForward Networks, également appelés multilayer perceptrons(MLPs), sont les modèles d'apprentissage en profondeur les plus classiques. Par exemple, pour un classicateur, $y = f(x)$ qui associe une catégorie y à une entrée x. Un réseau feedforward définit une fonction $y = f(x; \theta)$ et apprend la valeur des paramètres θ qui donne la meilleure approximation de la fonction f.

Ces modèles sont appelés feedforward parce que l'information circule à travers la fonction en cours d'évaluation de x, à travers les calculs intermédiaires utilisés pour définir f, et enfin à la sortie y. Il n'y a pas de connexions de retour (récurrentes) dans lesquelles les sorties du modèle sont réintroduites dans lui-même. Lorsque les réseaux de neurones sont étendus pour inclure des connexions de retour, ils sont appelés réseaux de neurones récurrents.

Les réseaux neuronaux Feedforward sont appelés réseaux car ils sont généralement représentés par la composition de plusieurs fonctions différentes. Le modèle est associé à un graphe acyclique dirigé décrivant la composition des fonctions. Par exemple, nous pourrions avoir trois fonctions $f^{(1)}$, $f^{(2)}$ et $f^{(3)}$ connectées dans une chaîne, pour former $f(x) = f^{(3)}\left(f^{(2)}\left(f^{(1)}\left(X\right)\right)\right)$. Ces structures de chaîne sont les structures les plus couramment utilisées des réseaux de neurones. Dans ce cas, $f^{(1)}$ est appelée la première couche du réseau, $f^{(2)}$ est appelée la deuxième couche, etc.

Pendant l'entraînement du réseau de neurones, nous poussant $f(x)$ pour qu'elle corresponde à $f^*(x)$. Les données d'entraînement nous fournissent des exemples approximatifs de $f(x)$ évalués à différents points d'entraînement. Chaque exemple x est accompagné d'une étiquette $y = f(x)$. Les exemples d'apprentissage spécifient directement ce que la couche en sortie doit faire à chaque point x: elle doit produire une valeur proche de y. Le comportement des autres couches n'est pas directement spécifié par les données d'apprentissage. L'algorithme d'apprentissage doit décider comment utiliser ces couches pour produire le résultat souhaité, mais les données d'apprentissage ne disent pas ce que chaque couche doit faire. Au lieu de cela, l'algorithme d'apprentissage doit décider comment utiliser ces couches pour mieux implémenter une approximation de f^*.

2.2 Convolutional nural network (CNN)

Réseau de neurones convolutionel (CNN ou convNet) (**figure**1) partent du même principe des réseaux de neurones feedforward définit précédemment, sauf qu'ils sont caractérisés par l'utilisation des couches de convolutions qui servent à appliquer des transformations à son entrée (dans notre cas une image) à l'aide des filtres, appelés souvent kernels, le pooling et le padding.

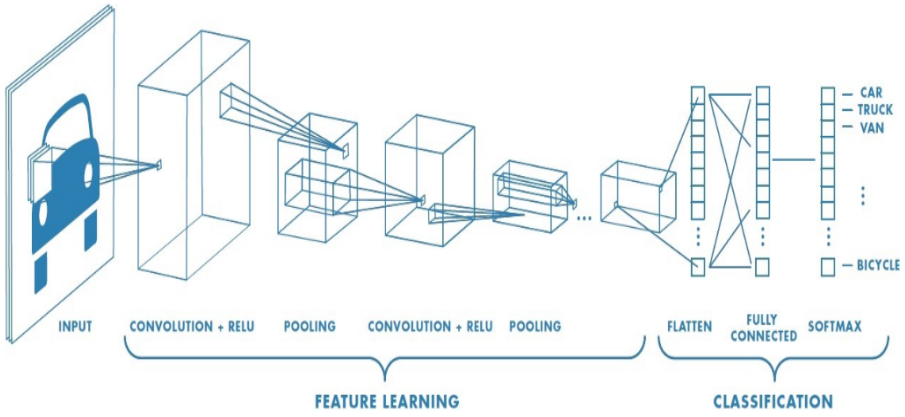

Figure 1: Architecture du CNN

L'idée c'est d'appliquer des convolutions à des images à l'aide des filtres dont les paramètres sont ajustés lors de l'entraînement du réseau, ces filtres nous permettent de détecter des paternes qui caractérisent les images de chaque classe. Ces convolutions sont appliquées avec des filtres avec un pas s donné appelé « strides » : si s=1 le filtre se décale d'un pixel.

Lors de la convolution deux cas se posent dans les bords de l'image :

- On supprime la partie de l'image où le filtre ne peut pas s'appliquer.
- On ajoute un nombre p de zéros autour de l'image pour tenir compte de l'ensemble des pixels qui correspondent réellement à l'images, cette opération est appelée le « padding » **(figure**2**)**

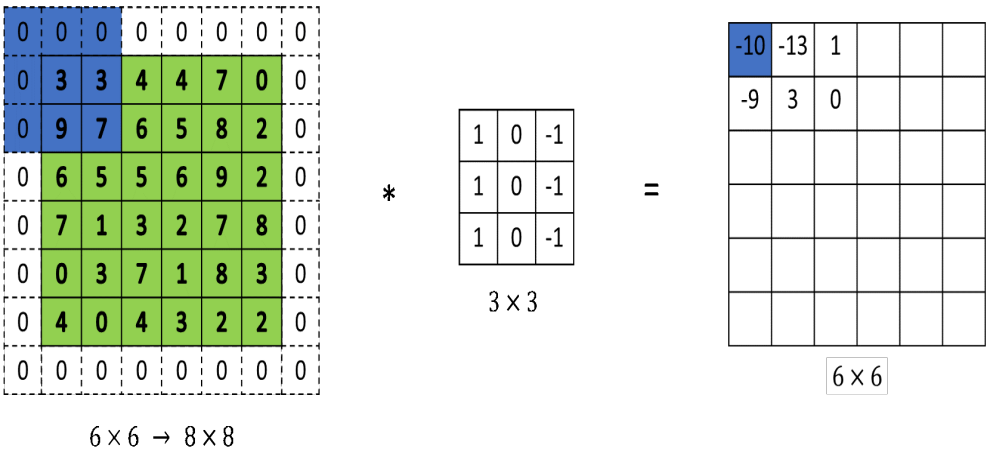

Figure 2: illustration du padding (p=1)

Pour réduire le nombre de pixel de l'image notamment réduire le nombre de paramètre de l'image, on a recours à utiliser des couches de « pooling » **(figure**3**)** on cite 3 types de pooling :

- Le Max Pooling : c'est une technique qui consiste à convoluer un filtre de taille n qui retourne la valeur maximale des pixels de l'image appliquée au filtre.
- Average pooling : le filtre retourne la valeur moyenne des pixels de l'image
- Sum pooling : le filtre retourne la somme des pixels de l'image.

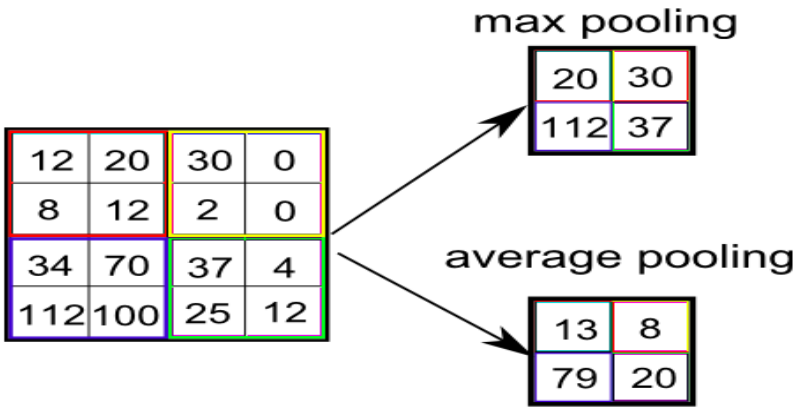

Figure 3: illustration du Max pooling et Average pooling

Beaucoup de travaux de recherche consistent à l'élaboration des modèles et des architectures de réseaux de neurones plus puissantes capable de donner de bons résultats. Ces architectures sont entrainées dans des larges datasets vu le nombre énorme de paramètre du modèle.

2.3 Resnet

Il existe une dépendance entre la profondeur du réseau de neurone et sa performance, plus un réseau est profond (contient plusieurs couches) plus ses performances sont élevées. Sauf qu'il existe un nombre limite de couches qui au-dessus de ce nombre la performance se dégrade, et c'est à cause de l'évanescence du gradient. Lors de la rétropropagation du gradient pour l'ajustement du modèle, le gradient est multiplié par les poids synoptique des couches, ce qui le rend de plus en plus petit ou très grand avec le nombre élevé de couches **(figure**4**)**.

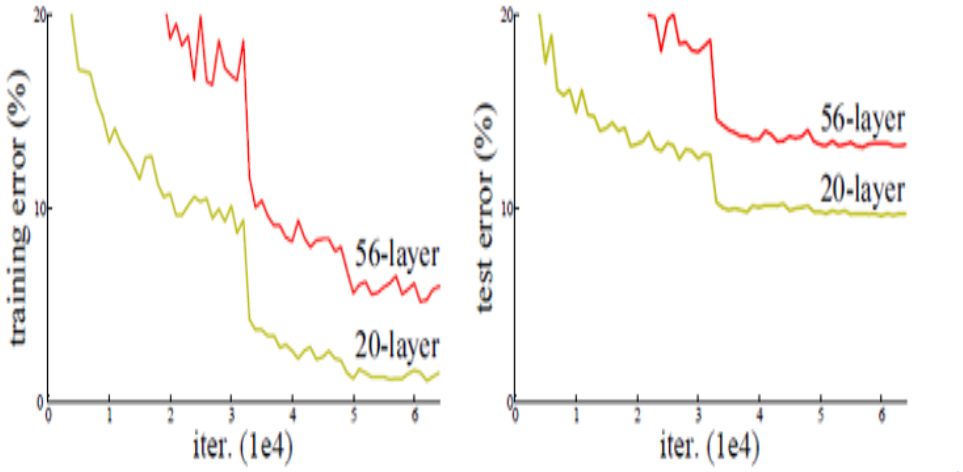

Figure 4: Erreur d'apprentissage (à gauche) et erreur de test (à droite) sur CIFAR-10 avec des réseaux "simples" de 20 et 56 couches. Le réseau le plus profond a une erreur d'apprentissage plus élevée, et donc une erreur de test.

Le réseau RESNET élaboré en 2015 par Microsoft Research réponds à cette problématique en introduisant des couches nommées « couches résiduels » **(figure** 5**)** à feedback intermédiaire, ces

couches sont caractérisées par l'équation suivante :

$$a^{[l+2]} = actv(a^{[l]} + z^{[l+2]})$$

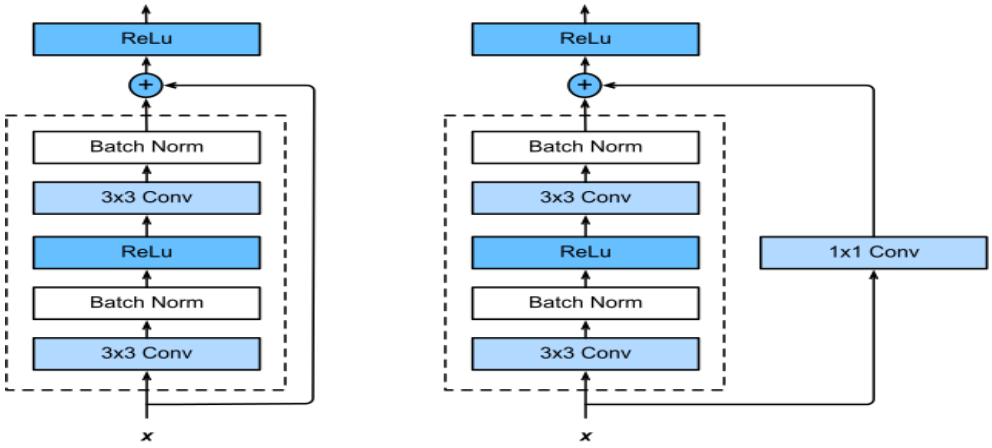

Figure 5: Bloc résiduel: Brique de base de l'architecture de ResNet

3 Analyse des données exploratoire

Les différentes classes des images:

- Épithélium pavimenteux
- Métaplasie intestinale
- Métaplasie gastrique
- Dysplasie/Cancer

3.1 Données d'entraînement

Notre DataSet est composée de 11161 images de taille 521*521 pixels acquises de 61 patient. Chaque image est une matrice 2D X_{i_p} où i est un indice allant de 0 au nombre d'images acquises du patient p. L'inclusion d'un indice des patients s'explique par le fait qu'il pourrait y avoir un certain niveau de corrélation entre les images acquises d'un seul patient, qui est plus grand que celui de deux patients différents.

l'ensemble d'entrainement est composé de 9446 images, acquises auprès de 44 patients répartient comme suit :

classe	Nombre d'images
0	1469
1	3177
2	1206
3	3594

Les deux ensembles de test, $Test_1$ et $Test_2$, sont les suivants:

1. 893 images reçu de 10 patients
2. 822 images reçu de 7 patients

Donc on a 9464 images dans la data de l'entrainement et 1715 images pour tester notre modèle.

Dans le fichier d'entraînement, la colonne "image_filename" contient le nom des images composé de deux informations : i m_NuméroImage_NuméroPatient. La figure ci-dessous montre le résultat de la lecture de ce fichier à l'aide du dataframe de Pandas.

	image_filename	class_number
0	im_4_0.png	0
1	im_21_0.png	0
2	im_9_0.png	0
3	im_8_0.png	0
4	im_15_0.png	0

Figure 6: fichier pandas dataframe d'entrainement

Après une visualisation des classes des données d'entraînement et le traçage de leur histogramme voir **Figure**7, on remarque que l'ensemble des données est très déséquilibré, la majorité des échantillons possèdent des labels de classe 3 et de classe 1. Cela nous pousse à penser à des méthodes de pré-traitement afin d'équilibrer ces classes pour prendre en considération le fait qu'il ont la même importance. Pour remédier à ce problème, nous prenons en considération les poids des classes lors du calcul de la fonction de perte.

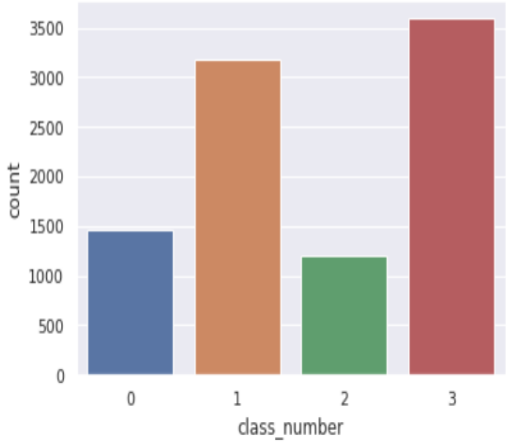

Figure 7: Répartition des images sur les classes

Vu la corrélation éventuelle entre les classes des images et les numéro des patients, nous avons pris cette information en considération lors de la division entre l'ensemble d'entraînement et de validation.

Exemples de micro images:
Nous avons pris aléatoirement une dizaine d'images de notre dataset pour les afficher. Les images ont été pris en utilisant une micro caméra développée par l'entreprise Mauna Kea.

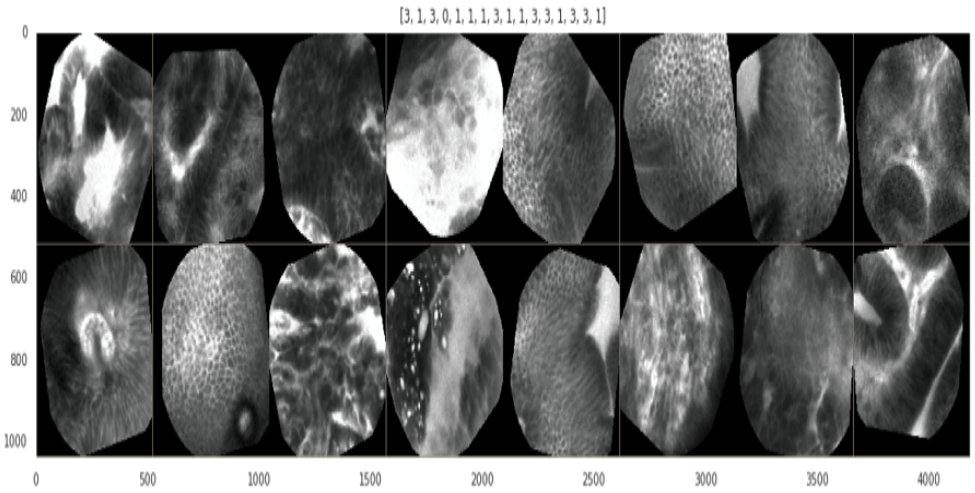

Figure 8: Exemples d'images traitées

4 Notre méthode

Le jeux de donné dont on dispose est relativement petit, ainsi il est difficile d'entraîner un modèle à partir de rien ("from scratsh"). En effet, on a essayer cette stratégie au début de la compétition, mais on a pas pu dépasser 30% en "accuracy". Ainsi, il était primordiale d'utiliser un réseau de neurone pré-entraîné. Heureusement, de telles modèles sont disponible sur torchvision [9], cette bibliothèque propose des implementations des réseaux de neurones les plus célèbre (ResNet [5], VGG [11], SqueezeNet [6]) et met à notre disposition leurs poids après pré-entraînement sur ImageNet [3]). On a utilisé ces modèles comme point de départ pour notre classificateur. En effet, il a était démontré dans plusieurs travaux que les réseaux de neurones convolutionnels ont la capacité d'améliorer l'apprentissage dans une nouvelle tâche par le transfert de connaissances à partir d'une tâche connexe déjà apprise. Ainsi, on utilise les connaissances visuelle apprise par un certain modèles lors de la classification des images provenant d'ImageNet, à la classification des images médical provenant du jeux de données proposé dans ce "data challenge". Il est assez facile de faire cet apprentissage par transfert ("transfer learning") en Torch [2], il suffit de changer la dernière couche de chaque modèle en réduisant le nombre de neurones de sortie du modèle au nombre de classe qu'on a.

Vu le déséquilibrage des classes qu'on a dans la dataset, nous sommes amenés à choisir une perte en tenant compte de ce déséquilibrage. La perte « Loss CrossEntropy » avec équilibrage des classes est adéquate avec notre problématique. Ainsi le fonction de perte utiliser est la suivante

$$\mathcal{L}(y, y'(\omega)) = \sum_{i=1}^{N} \sum_{c=1}^{\mathcal{C}} \hat{p}(c) y_{i,c} \log(y'_{i,c}(\omega)) \tag{1}$$

Où $y_{i,c}$ vaut 1 si l'image i appartient à la classe c et vaut 0 sinon, et $y'_{i,c}(\omega)$ représente la probabilité que notre modèle prédit pour que l'image i appartiennent à la classe c, ω représente l'ensemble des poids de notre modèle et $\hat{p}(c)$ est la proportion d'image appartenant à la classe c dans notre jeux de données d'entraînement.

Une pratique courante dans le monde de la vision par ordinateur est l'utilisation de l'augmentation des données ("Data augmentation") pour entraîner des réseaux de neurones convolutionnels avec des jeux de données à des tailles relativement réduites. Cette pratique consiste à appliquer aléatoirement un certains nombres de transformations pré-définis sur chaque image du jeux de données. Il est à noter que à chaque epoch et à chaque mini-batch les transformations sont tirées aléatoirement, c'est à dire que l'image sera transformée dans chaque mini-batch. En pratique, Il est facile d'appliquer ces transformations en utilisant le module "transfomrs" de "torchvision" [9]. Dans notre cas plusieurs expériences ont été réalisées pour décider les transformations à utiliser. Une étude de la convergence du modèle et de sa performance sur le jeux de données de validation a permis de faire ce choix.

Comme indiqué précédemment, l'apprentissage des poids d'un réseaux de neurones pour cette tache de classification peut être considérer comme un problème d'optimisation:

$$\min_{\omega} \mathcal{L}(y, y'(\omega)) = \min_{\omega} \sum_{i=1}^{N} \sum_{c=1}^{\mathcal{C}} \hat{p}(c) y_{i,c} \log(y'_{i,c}(\omega) \tag{2}$$

La pratique courante pour résoudre ce type de problème d'optimisation est d'utiliser une descente de gradient stochastique. La littérature présente plusieurs algorithmes d'optimisation qui permettent de résoudre ce problème (e.g. Adam [7], Adagrad [4], AdaDelta [13], RMSProp [12]). Dans notre cas plusieurs expériences ont été réalisées pour décider l'optimiseur et la valeur du pas de descente ("learning rate") à utiliser. Nos expériences ont montrés que le meilleur choix était d'utiliser Adam avec un "learning rate" de 10^{-5}.

On a également étudié l'influence de la taille du batch ("batch size") sur l'entraînement et les performances du modèle, on a essayé plusieurs valeurs (4, 8, 16, 32) mais on a observé aucune influence significative sur les performances du modèle, on a pourtant utilisé une valeur de batch size de 16 dans la suite de nos expériences, vu quelle réalise un bon compromis entre la vitesse d'entraînement et la consommation de mémoire.

Il est à noter que toutes nos expériences ont été menées en utilisant **google colab**, parcequ'il fournit des sessions gratuites de GPU (**NVIDIA Tesla K80**) de 12h. Ainsi, dans une première partie, nous avons téléchargé la base de données fournie par Mauna Kea sur Google Drive et avons établi une connexion entre le drive et la session de google colab.

Pour avoir plus de flexibilité dans l'entrainement du modèle, raison pour laquelle on a utilisé Torch, nous avons créé une fonction pour entrainer le modèle pour un époch et lors de l'entrainenment on enregistre le modèle dans l'accuraccy est la plus grande. (cf. Colab Notebook)

Comme indiqué lors de l'analyse exploratoire des données, les images provenant d'un même individu partagent dans la plupart des cas le même label, on a proposé d'utiliser cette information pour améliorer la qualité des prédictions de notre modèle. En effet, on assigne au images d'un individu donnée la valeur de la classe majoritaire que prédit notre modèle. Ce "post-processing" a permis de booster la performance de notre modèle d'environ 11 point d'accuracy pour qu'il atteint une valeur de 97% d'*accuracy*, sachant qu'il arrivait seulement à 86% sans ce post-processing.

5 Résultats

Dans cette section, on rapporte les résultats du meilleur modèle qu'on a pu proposer. Ce modèle utilise resnet50 (on a également essayer resnet18, resnet34, vgg16, SqueezeNet, MobileNet), il est entraîné sur 15 epoch en utilisant adam comme méthode d'optimisation avec un "learning rate" de 10^{-5} et avec une taille de mini-batch de 16. On a également utiliser plusieurs augmentation de données (**RandomHorizentalFlip, RandomVerticalFlip, RandomAffine, RandomCrop**). Figure 9 et Figure 10 montre l'évolution de la valeur du loss function et de l'accuracy sur le dataset d'entraînement et de validation lors de l'entraînement de notre modèle.

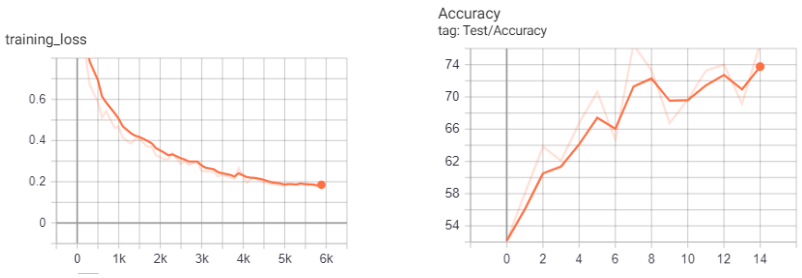

Figure 9: Évaluation de l'entraînement du classificateur. A gauche, la variation de la valeur de la fonctions de perte sur les jeux de données de validation. A droite, l'évolution de l'accuracy sur les jeux de données de validation.

model	val_accuracy	test_accuracy	post-processing accuracy
Resnet18	71%	68%	79%
Resnet50	90%	86%	97%
Squeeznet	77%	73%	82%
VGG16	75%	70%	80%

Table 1: l'accuracy de différents modèles

On remarque clairement que ResNet50 est le prédicateur le plus efficace dans notre cas de figure avec une accuracy de 97%. En outre, notre post-processing nous a permis de gagner 11% en accuracy, chose qui a boosté notre score et nous a permis d'être classé 4^{eme} dans le classement général.

6 Conclusion

Dans ce projet, nous avons utilisés les réseaux de neurones convolutionnels (ResNet,VGG16 et squeezeNet) pré-entraînés pour classifier des images médicales. Ensuite, nous avons fait une augmentation de données, cette augmentation permet d'éviter le sur-apprentissage du modèle. Nous avons aussi testé avec le modèle dont l'accuracy est la plus grande dans l'ensemble de validation. A la fin, nous avons trouvé une précision de 97%, en utilisant le modèle Resnet50 avec un batch size de 16 et un learning rate de 10^{-5}, ce qui nous a mis au 4^{eme} rang du classement public.

Nous avons rencontré des difficultés au cours de ce défi à savoir : la maîtrise de la librairie **Torch**, le long temps de calcul qu'exige l'entraînement du modèle (1h30min en moyenne) et le choix des transformations pour le pré-processing.

Nous pouvons encore améliorer la précision en changeant le pré-processing que nous avons choisi avec un autre bien adéquat. On peut également programmer un post-processing qui ne change que les valeurs des classes des images qui ont une faible probabilité, au lieu d'attribuer la classes majoritaire à tous les images d'un individu. Outre cela, on peut moyenner les prédictions de plusieurs modèles pour avoir un classficateur plus robuste, cette opération est couramment appelé "Model Stacking". Finalement, après avoir fixé le meilleur modèle, on pourrait l'entraîner en utilisant l'intégralité de notre dataset (Training+validation) a fin de parcourir toutes les images.

References

[1] Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective, 2019. cite arxiv:1902.01724.

[2] Ronan Collobert, Samy Bengio, and Johnny Marithoz. Torch: A modular machine learning software library, 2002.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[9] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, pages 1485–1488, New York, NY, USA, 2010. ACM.

[10] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[12] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[13] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.