Ocular Disease Recognition

https://github.com/adblackx/ODR

Rapport de Projet

Alan Adamiak, Antoine Barbannaud
Sara Droussi, Maya Gawinowski
Ramdane Mouloua, Romain Mussard

Avril 2021

Table des matières

1 Introduction

Le projet que nous avons choisi consiste en une classification d'images de fond d'oeil humain en 8 classes. En effet, nous disposons d'images ainsi que de leur label correspondant respectivement à des maladies. Nous avons voulu entraîner un modèle afin qu'il arrive de lui-même à poser un diagnostic à partir d'une image.

Pour ce faire nous avons choisi de nous servir de CNN : un réseau de neurones ayant la particularité de contenir une couche de convolution permettant le repérage de « features ». Le choix d'utiliser un CNN vient de notre volonté de découvrir et mettre en oeuvre des techniques de Deep Learning qui nous étaient jusqu'à là inconnues.

Nous avons choisi d'utiliser le framework PyTorch ainsi que Torchvision. Nous avons utilisé des techniques de preprocessing, de la data augmentation et du transfer learning afin de maximiser la justesse (lire l'« accuracy ») de notre modèle. Pour l'architecture de notre projet nous avons repris un template que nous avons trouvé sur github [1] et qui nous a permis d'avoir le contrôle sur de nombreux aspects de notre projet, d'entraîner et d'enregistrer facilement nos modèles.

2 Présentation du dataset

Nous avons basé nos recherches sur la base de donnée ODIR[2], qui est un dataset de données ophtalmologiques contenant plus de 5000 patients. Sont renseignés dans celui-ci leur âge, sexe, des photographie de leurs yeux ainsi que des diagnostics et classifications. Les patients ont été recueillis dans différents centres médicaux en Chine avec des appareils photos variés, ce qui donne des images avec plusieurs résolutions.

2.1 Format des données

Le dataset contient tout d'abord un fichier CSV regroupant les informations sur les patients. L'en-tête de celui-ci se présente comme suit.

	ID	Patient Age	Patient Sex	Left-Fundus	Right-Fundus	Left-Diagnostic Keywords	Right-Diagnostic Keywords	N	D	G	C	A	H	M	O	filepath	labels	target	filename
0	0	69	Female	0_left.jpg	0_right.jpg	cataract	normal fundus	0	0	0	1	0	0	0	0	../input/ocular-disease-recognition-odir5k/ODI...	['N']	[1, 0, 0, 0, 0, 0, 0, 0]	0_right.jpg
1	1	57	Male	1_left.jpg	1_right.jpg	normal fundus	normal fundus	1	0	0	0	0	0	0	0	../input/ocular-disease-recognition-odir5k/ODI...	['N']	[1, 0, 0, 0, 0, 0, 0, 0]	1_right.jpg
2	2	42	Male	2_left.jpg	2_right.jpg	laser spot，moderate non proliferative retinopathy	moderate non proliferative retinopathy	0	1	0	0	0	0	0	1	../input/ocular-disease-recognition-odir5k/ODI...	['D']	[0, 1, 0, 0, 0, 0, 0, 0]	2_right.jpg
3	4	53	Male	4_left.jpg	4_right.jpg	macular epiretinal membrane	mild nonproliferative retinopathy	0	1	0	0	0	0	0	1	../input/ocular-disease-recognition-odir5k/ODI...	['D']	[0, 1, 0, 0, 0, 0, 0, 0]	4_right.jpg

FIGURE 1 – En-tête et premières valeurs du CSV

Les colonnes N D G C A H M O correspondent aux maladies considérées, respectivement : Normal (N), Diabète (D), Glaucome (G),Cataracte (C), Dégénérescence maculaire liée à l'âge (A), Hypertension (H), Myopie pathologique (M), Autres maladies/anomalies (O).

Le dataset comprends également deux dossiers. L'un contient 6392 images préprocessées et l'autre 1000 images destinées au training et 7000 à la validation. Les images préprocessées ont été redimensionnées afin d'avoir une taille constante et on y retrouve principalement des paires d'yeux.

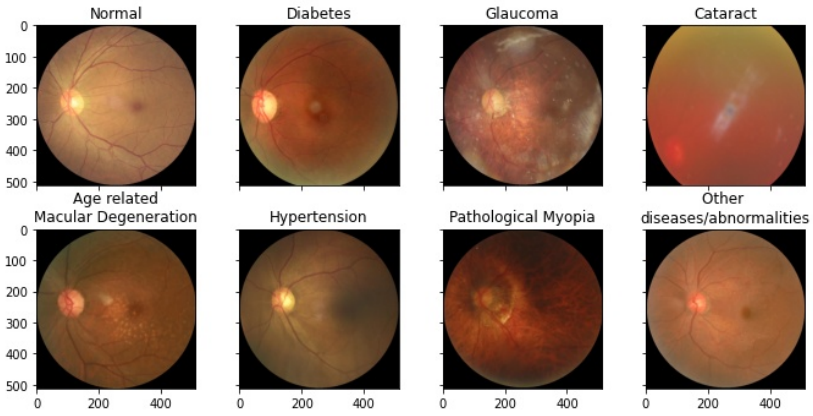

FIGURE 2 – Maladies oculaires représentées par le dataset

2.2 Analyse du dataset

Une première analyse des données du dataset nous permet de mieux comprendre ce que l'on manipule. Nous avons commencé par tracer la répartition des classes, afin de déterminer si nous avions une représentation assez conséquente de chacune pour entraîner le modèle avec une précision suffisante. Malheureusement le dataset représentait une distribution très inégale, avec la plupart des classes étant sous-représentées.

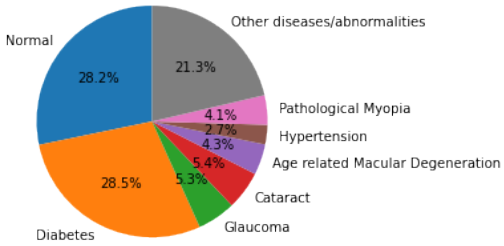

FIGURE 3 – Répartition des classes

Une analyse plus poussée nous montre également la répartition des maladies oculaires chez l'homme et la femme. En plus de visualiser le dataset de manière plus parlante, ces représentations nous permettent de chercher des corrélations entre les données que nous aurions autrement eu du mal à trouver.

3 Preprocessing

Le dataset proposait déjà un ensemble d'images préprocessées (suppressions des images inutilisables, simple recadrage et uniformisation de la taille des images) accompagnées d'un fichier CSV. Ce dernier indiquait pour chaque patient le diagnostic de chaque oeil, un vecteur contenant le type de maladie (sans distinguer s'il s'agissait de l'œil gauche ou de l'œil droit) et d'autres informations basiques comme l'âge du patient, son sexe, le chemin vers les images, etc. Cependant, ces informations ne nous convenaient pas et nous avions remarqué dans cette analyse que nous gagnerions à augmenter leur nombre.

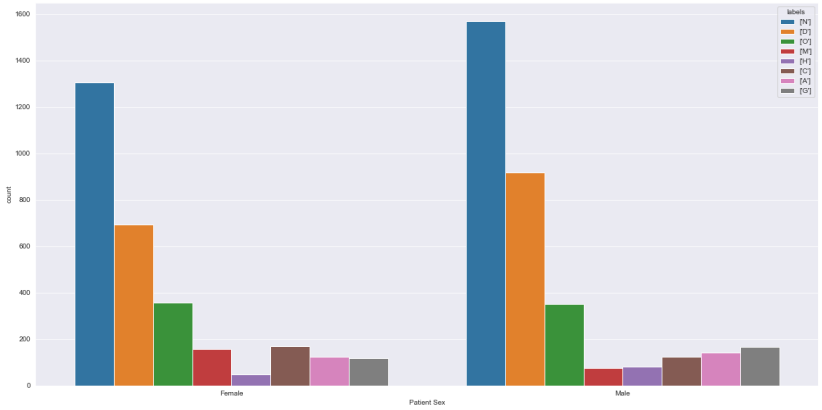

FIGURE 4 – Répartition et relation entre classes et sexe

3.1 Création d'un dataset plus personnalisé

Nous avons décidé dans un premier temps de remanier le fichier CSV en ne gardant que l'âge et le sexe du patient, une catégorie de maladie et l'image correspondante. Nous avons également décidé de ne faire qu'une image par ligne pour simplifier la lecture. En partant du tableau Excel contenant ces données, nous avons catégorisé les diagnostics en fonction du sujet du problème et nous en avons profité pour supprimer les entrées ayant plus d'une catégorie de maladie pour éviter de faire un modèle multiclasses. Nous avons aussi remarqué et supprimé les images annotées *low quality* par le médecin car elles n'étaient pas exploitables pour poser un diagnostic.

Cette première passe nous laisse avec 6200 entrées sur les 7000 de base.Nous avons aussi intégré la possibilité de filtrer les catégories des images pour créer des datasets différents, comme par exemple uniquement yeux saints et yeux malades.

Patient Age	Patient Sex	Image	Label
69	Female	0_left.jpg	1
69	Female	0_right.jpg	0
57	Male	1_left.jpg	0
57	Male	1_right.jpg	0
42	Male	2_left.jpg	1
...	...	...	...

FIGURE 5 – Fichier CSV après preprocessing

3.2 Processing des images

Dans la même lignée de contrôle des données utilisées, nous nous sommes occupés nous même du preprocessing des images. Ceci nous a permis de choisir quelles fonctions appliquer et de faire des tests sur différentes variations du preprocessing. Nous avons donc créé les fonctions suivantes pour appliquer des transformations :

- Resize : redimensionnement des images à la taille désirée. Dans notre cas nous avons choisi 512 x 512 comme pour les images préprocessées originales.
- Crop : retire les bordures noire des images car c'est de l'information inutile et rend l'image carré si elle ne l'est pas deja.
- Graham : Nous nous sommes inspiré d'un article de Ben Graham (Diabetic Retinopathy Detection Competition Report – gagnant de la compétition 2015 sur ce

dataset)[3] pour le traitement des images avec une série de transformation ayant pour objectif d'atténuer les conditions d'éclairage différentes des images : redimensionnement, soustraction de la couleur moyenne et mapping de la couleur moyenne à 50% gris, application flou gaussien pour supprimer les effets de limites. Nous avons implémenté le pseudo-code de son article.

— ToGrayscale : passage en niveau de gris des images pour diviser par 3 le nombre de features des images et donc réduire le temps d'apprentissage.

Ainsi, pour les mêmes images qu'en [Fig 2] nous obtenons les suivantes :

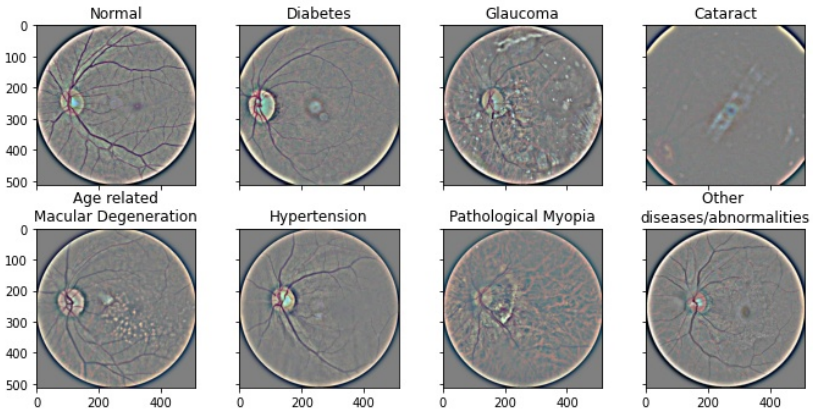

FIGURE 6 – Images après processing

3.3 Réduction du nombre de features

Chaque image propose (512x512) environ 460000 features. Il peut donc être intéressant d'enquêter sur la réduction de dimensions en utilisant Principal Component Analysis afin de réduire le nombre de features à traiter. Cette fonctionnalité qui se base sur la fonction PCA de Scikit-learn a été proposée dans le code du preprocessing mais nous n'avons pas eu le temps de la mettre en oeuvre sur la validation de notre modèle.

3.4 Transformation des données

Nous avons décidé de ne pas appeler nos fonctions de preprocessing dans le Data-Loader pour éviter des calculs lourds à chaque fois que le modèle accède à une image et ainsi réduire le temps d'entrainement. Pour la sélection du modèle nous avons choisi d'appliquer aux images brutes un resize, un crop et l'algorithme de Graham avant de les écrire dans un nouveau fichier sur le disque dur afin de les utiliser sur nos modèles.

Ainsi en utilisant AlexNet sur les données brut et les données ainsi prétraités on observe un gain d'1% d'accuracy. Il faut cependant noter que les CNN ne nécessitent pas beaucoup de preprocessing pour fonctionner correctement. De plus ce gain - bien que négligeable - est surtout accompagné d'une forte diminution du temps de d'entraînement notamment grâce au redimensionnement des images. Ceci explique donc notre choix de faire la sélection du modèle sur les données prétraitées plutôt que les données brutes.

3.5 Conclusion du preprocessing

Les fonctions de transformation images ont été réalisées avec la bibliothèque python OpenCv, les réduction de dimensions (PCA) avec la librairie Sklearn, et la mise en forme

pour le modèle avec Torch. Cela implique une difficulté supplémentaire de faire de multiples mises au format lors du entre différentes librairies (OpenCv, PIL, PCA, Torch). Il y avait sans doute une approche beaucoup plus élégante que nous n'avons pas trouvée par notre manque d'expérience avec les librairies utilisées.

Les fonctions développées dans le preprocessing nécessitent un long travail itératif entre des hypothèses de réglage de données, augmentation de données, réduction de dimension et les résultats de score du réseau neuronal qui doivent eux-même être réglés. Ce jonglage des paramètres nécessite de l'expérience et un temps de calcul conséquent. La durée de projet nous a donc fait défaut, en nous empêchant de réellement proposer une chaîne de preprocessing performante. Néanmoins, nous avons essayé de proposer l'ensemble des fonctions d'une démarche de preprocessing.

4 Modèle

Pour la partie modèle nous avons tenté d'utiliser des CNN pour classer nos images. Pour ce faire nous avons utilisé PyTorch et le module TorchVision qui contient beaucoup de modèles pré-entraînés sur ImageNet, nous permettant ainsi de faire du Transfer Learning.

Le Training Set est composé de 5407 images et celui de validation de 736 images, soit un peu plus de 10% de notre dataset initial. Nous avons voulu avoir une répartition la plus équilibré possible, mais il était impossible d'avoir le même nombres d'images pour chaque classe au risque d'avoir une disparité entre le set de validation et celui d'entraînement. Nous avons donc fait le choix d'appliquer la formule $min(0.15 \times S/8, 1/3 \times S_i)$ avec S la taille du dataset complet et S_i le nombre d'image de la classe i. On obtient la répartition suivante pour le set de validation :

— 115 images pour la classe 0 (Normale)
— 115 images pour la classe 1 (Diabète)
— 115 images pour la classe 2 (Glaucome)
— 87 images pour la classe 3 (Cataracte)
— 79 images pour la classe 4 (Dégénérescence maculaire liée à l'âge)
— 34 images pour la classe 5 (Hypertension)
— 76 images pour la classe 6 (Myopie pathologique)
— 115 images pour la classe 7 (Autres Maladies)

4.1 Les Réseaux de Neurones convolutionnels

Comme pour tout réseaux de neurones, le but d'un CNN consiste à faire des prédictions sur des données grâce à la minimisation d'une fonction de coût. On parle alors de *deep learning*. Les réseaux de neurones sont en fait composés de plusieurs couches : tout d'abord les entrées seront ici les images et leurs labels qui passent par différentes couches qu'on appelle les "hiddens layers". Ceux-ci ont pour rôle d'extraire des informations et chaque couche se sert de la couche précédente pour "apprendre" en profondeur. La dernière couche est la sortie du réseau de neurone. Ce sera dans notre cas un tableau des probabilités où la valeur la plus élevée représente la classe prédite par notre modèle pour une image donnée. Nous nous sommes plus particulièrement intéressés à des réseaux de neurones convolutionnels, les "convolutional neural network" ou "CNN". Ils sont des réseaux de neurones qui comprennent une couche de convolution.

Les couches les plus courantes des CNN sont :

— La **convolution**, qui peut être une matrice (par exemple la matrice d'identité) renvoyant la même image, ou d'autres matrices plus évoluée qui vont alors extraire des informations de l'image (ou de la featuremap).

— Il y a aussi le **MaxPool** qui vise à réduire la dimension de la feature map mais qui génénère une perte d'information au passage
— **Relu** qui s'utilise à la sortie des convolutions en général (en évitant par exemple le problème du gradient qui s'annule)
— les **Drop Out** qui visent à ignorer avec une certaine probabilité à chaque époque différents noeuds du réseau de neurone
— la **Batch normalisation** qui consiste en la normalisation d'un lot de taille *Batch*, qui sont les entrée simultanées d'un réseau de neurone. Ceci permet d'améliorer le taux d'apprentissage.

4.2 Métriques, fonction de coût et optimizer

Nous avons utilisé la Cross Entropy Loss comme fonction de coût. C'est une fonction de coût répandue pour les CNNs qui nous semblait particulièrement adaptée à nos besoins. En effet la Cross Entropy Loss calcule l'écart entre la probabilité d'appartenir à une classe (calculé par le modèle) et sa classe réelle (le label). Cette fonction ne s'intéresse donc pas au fait que le modèle classe bien une image où non, mais au fait que les probabilités calculées par le modèle soient le plus fidèle possible à la *ground truth*. Ici la Cross Entropy Loss aura un rôle important car elle interagira avec le modèle lors de la backpropagation (par descente de gradient) pour modifier les poids du modèle, réduire le coût et par la même occasion augmenter l'accuracy.

Pour la fonction d'optimisation nous avons choisi SGD. Encore une fois, c'est un optimizer très classique dans la classification via CNN. Nous avons aussi pensé à utiliser Adam mais n'avons pas remarqué de différence.

Enfin pour évaluer notre modèle nous avons choisis de calculer l'accuracy, c'est à dire le taux d'images bien classées. En effet ce qui nous intéresse le plus ici c'est que le modèle arrive à bien classer nos images. Qu'il le fasse avec des probabilités élevées serait préférable mais pas indispensable, on préférera un modèle peu confiant mais qui classe bien plutôt que l'inverse.

4.3 Recherche du meilleur modèle

Pour la sélection du modèle nous avons choisit un Learning Rate de 0.01 pour l'ensemble des modèles, sauf AlexNet qui a un Learning Rate de 0.001. Le batch size varie cependant d'un modèle à l'autre, surtout dût aux limites de mémoire de nos cartes graphiques. Chaque modèle à été entraîné sur 40 époques.

Nous avons essayé de sélectionner le meilleur classifieur parmi 4 modèles de CNN réputés : AlexNet, ResNet, Vgg et SqueezeNet. Nous avons entraîné chacun de ces modèles en utilisant une combinaison de diverses méthodes pour mettre en avant le procédé qui marchait le mieux. Ainsi chaque modèle à été entraîné 3 fois : nous avons entraîné le modèle de zéro la première fois, puis nous avons entraîné un modèle pré-entraîné sur ImageNet et enfin nous avons rajouté de la data augmentation. Vous pouvez voir dans le tableau ci-dessous la meilleur accuracy obtenue au cours des 40 époques pour chaque modèle :

Modèles	AlexNet	ResNet	Vgg	SqueezeNet
Raw	NC	37%	43%	35%
Transfert	46%	50%	53%	40%
Transfert + Data Augmentation	41%	54%	57%	29%

FIGURE 7 – Meilleur Validation Accuracy sur 40 epochs

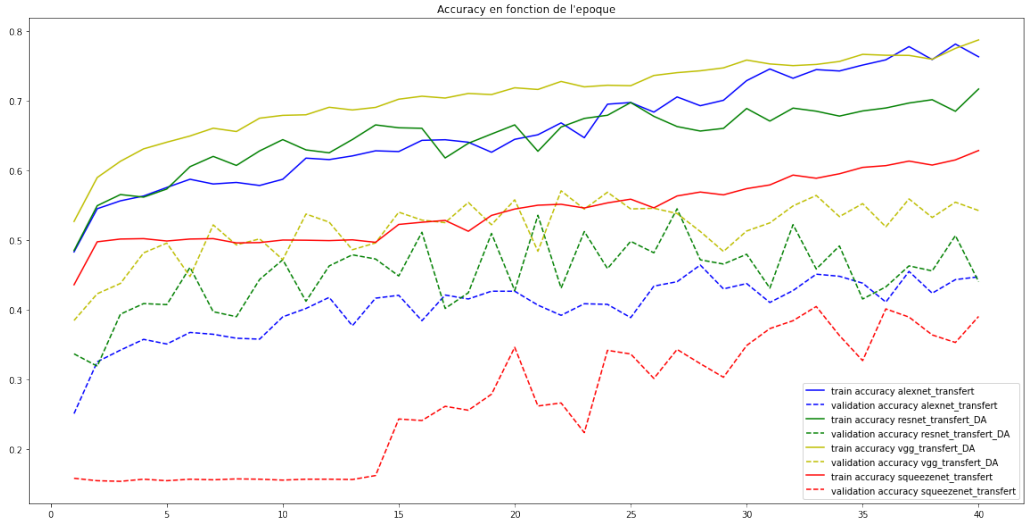

FIGURE 8 – Évolution de l'accuracy en fonction des epochs

Sur la [Fig 8] nous pouvons voir qu'il y a tout même de l'overfitting pour tout les modèles. En effet l'accuracy sur l'ensemble d'entraînement est beaucoup plus haute que pour la validation. On peut aussi bien voir, en accord avec le tableau de la [Fig 7], que Vgg et ResNet ont les meilleurs accuracy dépassant tous les 2 la barre des 50%.

On voit clairement sur la [Fig 7] que pour ces 2 modèles, l'utilisation de modèle pré-entraîné couplé à de la data augmentation permet d'augmenter l'accuracy d'autant plus quand il est. Très souvent utilisé lorsque l'on manque de données, l'apprentissage par transfert consiste à utiliser un modèle pré-entraîné (si possible sur un problème sensiblement proche de celui qu'on souhaite résoudre). Ceci permet de profiter des caractéristiques que le modèle a appris lors de son premier apprentissage pour améliorer l'apprentissage sur le problème et le dataset étudié. Ici les modèles pré-entraînés proviennent de Torchvision et on été entraînés sur ImageNet.

Pour la data augmentation nous sommes restées sur l'utilisation de fonctions PyTorch relativement classiques. Celà nous a permis d'effectuer des rotations d'images aléatoire entre -90 et 90°, des flips horizontaux et verticaux 1 fois sur 2 ou encore des transformations affines. Ces fonctions sont appelées lors du passage d'une image au modèle via le Data Loader. Ainsi le nombre d'images vues en une époque reste la même, mais d'une époque à l'autre certaines images sont modifiées (rotation, flip...), permettant d'augmenter artificiellement la quantité d'images sur laquelle s'entraîne le modèle. Ainsi, nous pouvons mieux généraliser les features mais aussi éviter l'overfitting tout en améliorant le score.

Il apparaît assez clairement que le modèle VGG est celui qui arrive le mieux à résoudre

le problème, son accuracy est de 57%. C'est peu cependant nous travaillons sur un datatset relativement petit avec des maladies sous-représentées et tout de même 8 classes. Il est clair que le modèle fonctionne puisqu'il fait bien mieux que le hasard, mais il est certains que ce score peut être augmenté avec des techniques d'apprentissage plus poussées, peut être en prenant en compte l'âge et le sexe des patients.

4.4 Quelques pistes explorées :

Nous avons créé notre propre modèle pour pouvoir intégrer des informations supplémentaires aux réseaux de neurones. Pour cela, nous avons du créer une nouveau "trainer" et modulé la classe "Dataset" pour pouvoir récupérer en plus de l'image et du label, l'âge ou le sexe. Dans le code, nous avons au final choisi d'intégrer l'âge ou le sexe en fonction de ce qui est commenté dans la classe "Dataset".

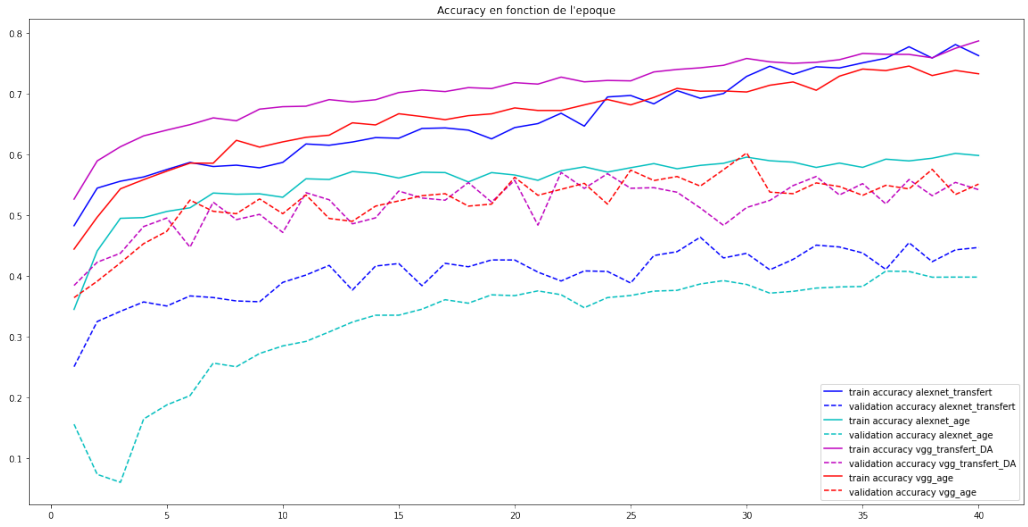

FIGURE 9 – Évolution de l'accuracy en fonction des epochs

La classe "myAlexNet" est une tentative qui reprends le code source de AlexNet (et donc les différentes couches de celui-ci) et y ajoute à la fin des fonctions incorporant l'âge ou le sexe. Nous n'avons toutefois pas eu le temps de le tester. Nous avons vu que le transfert avait l'air de bien marcher avec la data-augmentation. Nous avons également a pu tester deux modèles : "mymodel" et "mymodel2". La différence entre les deux est la linéarisation des sorties du CNN. Ce qu'on fait c'est qu'on récupère l'image dans la fonction forward, notée. Des tests ont été effectués sur le sexe, mais rien de probant n'a été trouvé.

Au final, on a pu tester la classe "model2.py" qui a donné les meilleurs résultats. Dans le graphe, la courbe Alex avec ajout de l'âge fait moins bien que Alexnet (en DA + transfert). Alors que la courbe de VGG avec l'ajout de l'âge fait clairement mieux que VGG sans âge, avec un pic à 0.6.

Nous avions aussi commencé à essayer de combiner nos 2 meilleurs modèles (VGG et ResNet) en prenant leur prédictions pour les donner en entrées à un Random Forest Classifier. Cependant nous n'avons pas eu le temps d'explorer cette piste en profondeur et les résultats préliminaires n'étaient pas très encourageant. A priori il semble primordial d'améliorer le score des modèles CNN avant d'envisager l'utilisation d'un Random Forest pour fusionner plusieurs modèles.

5 Conclusion

Pour conclure, nous avons utilisé de nombreuses techniques pour améliorer le score de nos modèles mais n'avons pas réussi à avoir un score supérieur à 57% (ou encore 60% pour VGG avec ajout de l'âge). Pour arriver à ce résultat nous avons au préalable traité nos images pour les rendre plus homogènes et limiter l'influence de l'appareil. Nous avons également comparé plusieurs modèles de CNN tout en utilisant de la data augmentation et du transfer learning. Bien que le résultat nous paraisse comme étant beaucoup trop faible, il faut rappeler que le dataset n'était pas forcément très facile à traiter par sa taille et la répartition très inégale des classes. Nous pensons néanmoins qu'il est possible d'augmenter le score.

Bien que le résultat n'était pas à la hauteur de nos espérances, nous avons pu à travers ce projet découvrir le Deep Learning et les CNN. Celles-ci nous ont conquis en tant que méthodes fascinantes, quoique pas toujours facile à utiliser à bon escient. Plus d'une fois - notamment lors de l'étape du traitement par modèle - nous avons été confrontés à nos limites en la matière et plus que jamais l'aspect théorique nous parait comme étant fondamental pour mener à bien un projet de la sorte.

Références

[1] Template du projet github. https://github.com/victoresque/pytorch-template, 2015.

[2] Ocular disease recognition, right and left eye fundus photographs of 5000 patients. https://www.kaggle.com/andrewmvd/ocular-disease-recognition-odir5k, 2020 (V2).

[3] Ben Graham. Diabetic retinopathy detection competition report. https://storage.googleapis.com/kaggle-forum-message-attachments/88655/2795/competitionreport.pdf, 2015.