Projet Machine Learning
IRM Brain Tumor Dataset

Roc de Larouzière, Mouna Ahamdy

May 2024

1 Introduction

Pour réaliser notre projet, nous avons choisi de travailler sur le dataset « Brain Tumor MRI », mis en ligne sur Kaggle par Masoud Nickparvar en 2021. Le choix de ce dataset s'est donc deroulé en plusieurs étapes. Nous voulions choisir un problème d'apprentissage supervisé, puisque le programme de cette option se concentre majoritairement sur cette partie. De plus, pour les mêmes raisons, nous voulions traiter une tâche de classification. Plusieurs options se sont alors proposées entre de la classification de nombres avec un dataset permettant de créer un modèle jugeant quelles exoplanètes sont habitables en fonction de leurs caractéristiques ou de la classification d'image avec deux datasets nous plaisant. Le premier concernait la présence de danger en fonction d'éléments anormaux sur une route et le second à de la classification multiclasse de tumeurs de cerveaux sur des IRM. Nous avons finalement opté pour ce dernier sujet car l'analyse de données à des fins médicales représente un domaine d'application qui selon nous donne du sens, et sur lequel Mouna aimerait sûrement travailler plus tard.

2 Le Jeu de Données

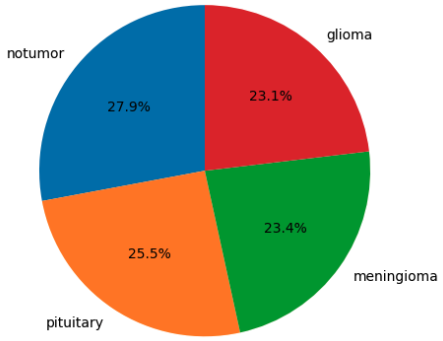
Notre problème consiste donc en un problème de classification où l'on doit prédire si pour une irm de cerveau donnée, celle-ci est saine (absence de tumeur) ou si elle possède une tumeur de type Glioma, Meningioma ou Pituitary.
Pour l'importation du Dataset, nous avons d'abord téléchargé le Dataset complet sur Google Drive pour traiter les données depuis Google Drive. Le temps de recuperation des données de Google Drive à Google Colab était important ce qui a entrainé un embedding (transformation des images en vecteurs exploitables) de l'ensemble de nos images tournant pendant plus de 2 heures. Nous avons finalement compris qu'en téléchargeant nos données directement sur Google Colab, la démarche serait bien plus efficace et notre temps d'embedding est passé de 2 heures à environ 5 minutes.
Notre Dataset se présente comme deux sous dossiers, un Training, et un Testing, contenant chacun 4 sous-dossiers : Notumor, Glioma, Meningioma, Pituitary. Chacun de ces sous-dossiers contient ainsi des images de tumeurs de cerveaux avec présence ou non de tumeur en fonction du sous dossier dans lequel elles sont.

Pour la répartition des images en fonction de la classe à laquelle elles appartiennent, on a :

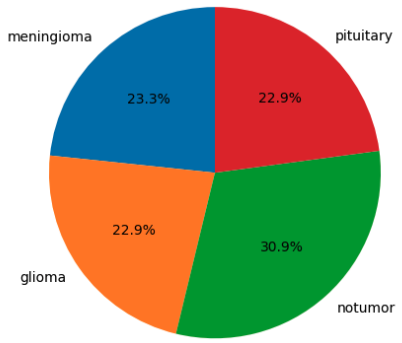
Un total de 5742 images d'entrainement dont :

— 1605 sont des images de cerveau sans tumeur
— 1321 sont des images de cerveau avec une tumeur glioma
— 1349 sont des images de cerveau avec une tumeur meningioma
— 1467 sont des images de cerveau avec une tumeur pituitary

Un total de 1311 images de test dont :
— 405 sont des images de cerveau sans tumeur
— 300 ont des images de cerveau avec une tumeur glioma
— 306 sont des images de cerveau avec une tumeur meningioma
— 300 sont des images de cerveau avec une tumeur pituitary



Reparition des données de la partie train du Dataset



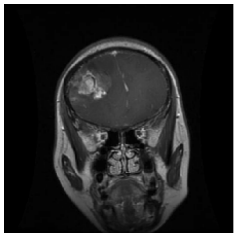
Repartition des données de la parie test du dataset

Les données sont ainsi équilibrées, chaque classe possède un nombre environ équivalent de données par rapport aux autres classes ce qui évite des problèmes de surapprentissage des caractéristiques d'une classe majoritaire.

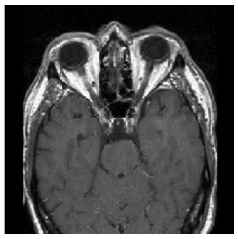
3 Preprocessing et embedding

Pour le traitement de nos données, nous nous sommes concentrés dans un premier temps sur le preprocessing des images. En effet, les images étaient, dans

le dataset original, de tailles différentes. Nous avons alors créé une fonction qui transformait toutes les images en dimensions (224,224,3) puis les normalisait. Cependant, en cherchant, nous avons réalisé qu'il existait des fonctions de preprocessing exprès pour les modèles pré-entrainés que nous utilisons : preprocess_input des librairies vgg16 et resnet50 de keras. Ainsi, afin de convertir nos images dans un format propice à l'execution du modèle vgg16 et resnet50, nous effectuons un preprocessing grâce à la classe ImagedataGenerator de Keras et les `preprocess_input` fonctions de vgg16 et resnet50. En effet, ces dernières effectuent des tâches particulières sur nos images comme l'inversion des canaux de couleurs car les modèles vgg on été entrainés avec des images utilisant le format BGR et non RGB (R pour Rouge, G for Vert et B pour Bleu). La fonction preprocess réalise aussi un centrage de couleur, où chaque canal de couleur est centré autour de 0 en soustrayant la moyenne de ce canal calculée grâce aux données imageNet (données d'entrainement de vgg16 et resnet50). Il s'agit donc d'une forme de standardisation de nos données. Nous allons maintenant présenter l'évolution de trois exemples d'image de chaque classe, avant le preprocess et après. La présence de tumeur pituitary après le preprocessing se voit notamment très bien sur la dernière image.



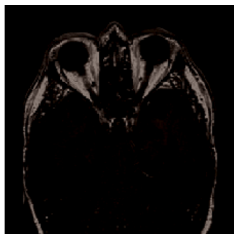
IRM tumeur Glioma avant preprocessing



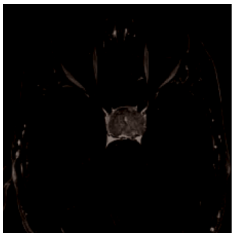
IRM sans tumeur avant preprocessing

IRM tumeur pituitary avant preprocessing

IRM tumeur Glioma après preprocessing

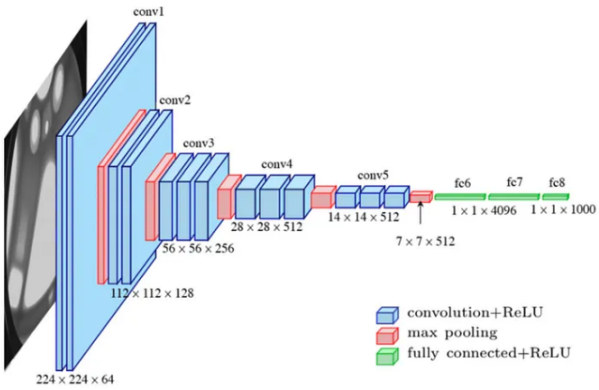


IRM sans tumeur après preprocessing



IRM tumeur pituitary après preprocessing

Afin de traiter nos données transformées (preprocessing), nous avons décidé d'utiliser le modèle de reseau de neurone convolutionnel, modèle adapté dans le but de transformer nos images en vecteurs afin de pouvoir les exploiter.
Nous avons commencé par nous intéresser au fonctionnement des réseaux de neurones convolutionnels. Nous avons dans un premier temps cherché à réaliser notre propre réseau de neurone convolutionnel, en essayant différentes combinaisons, de manière plutôt aléatoire, de couches convolutionnelles, de couches pooling et couches denses. Nos essais ont présenté un excellent apprentissage sur notre base de données train (0.90), nous en étions ravis, jusqu'à tester notre modèle sur les données test avec un retour à la réalité et une accuracy de 0.12... la définition de l'overfitting. Nous pensons que cela était dû à la phase d'entrainement sur un ensemble de couches trop complexes par rapport à la quantité trop faible d'images que nous avons (5147 images d'entrainement). Nous avons alors opté pour réaliser notre transformation d'images en vecteurs grâce à des modèles pré-entrainés.
Pour cela, nous avons travaillé sur deux reseaux de neurones pré-entrainés, vgg16 et resnet50. Ces deux modèles, entrainés sur la base de données ImageNet, contenant plus de 14 millions d'images annotées, sont deux architectures de réseaux de neurones profonds très influentes et robustes dans le domaine de la vision par ordinateur. Nous avons donc tiré parti de ces deux architectures dont l'une est présentée ci-dessous :



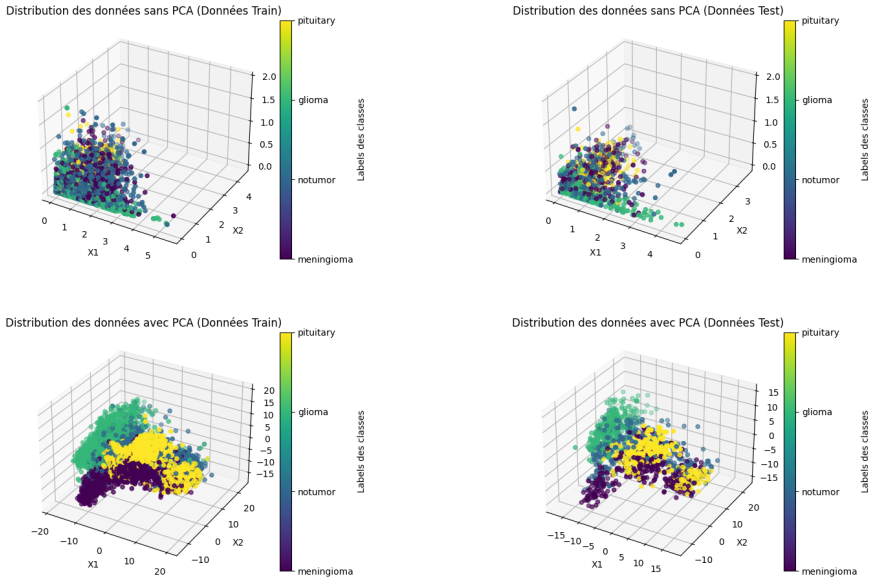
Architecture de vgg16

Le schéma présente ainsi l'achitecture de vgg16, composé de 13 couches convolutionnelles, des couches de maxpooling, et des couches fully connected à la fin du traitement. Pour le projet, nous avons enlever les couches fully connected qui sont typiquement utilisées pour les tâches de classification. De la même manière, nous avons aussi enlevé les couches fully connected de l'architecture de resnet50, afin de pouvoir appliquer les algorithmes de classification vus en cours

sur nos images venant d'être transformées en vecteurs.

Cependant, pour VGG 16, si nous n'utilisons pas de couches supplémentaires, les dimensions des images à la sortie des couches intermédiaires sont de (7,7,512), nous ajoutons donc à notre modèle une couche flatten qui aplatit nos données transformant donc le tensor (7,7,512) en vecteur de 25 088 éléments. Cette dimension étant beaucoup trop grande, il est impératif de réduire la taille. Nous avons donc réfléchi à 2 procédés différents pour la réduction de la taille.

Le premier a été, comme vu dans le cours 6 concernant "Dimensionality Reduction", d'appliquer un PCA. Ce dernier réduit considérablement les dimensions en trouvant les informations les plus utiles dans la qualification de nos objets plutôt que d'utiliser 25 088 éléments. Ceci permet non seulement d'accélérer le processus de traitement des données et de leur classification (puisque nous travaillons avec des vecteurs beaucoup plus petits) mais nous remarquons qu'il permettra également plus tard d'obtenir une meilleur accuracy lors de la classification. Comme il se focalise sur des caractéristiques plus adaptées pour la caractérisation de nos images, il généralise ces dernières également aux images du Test, et rend la délimitation entre les classes plus visible pour l'algorithme que dans le cas où il n'est pas appliqué. Nous avons réussi à illustrer ce phénomène en affichant deux graphes différents pour le Train et le Test, montrant la distribution des données avant et après l'application du PCA.



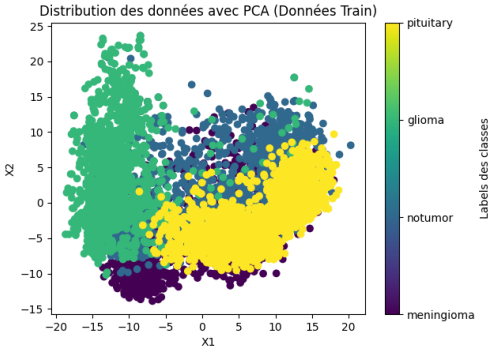
Le deuxième procédé serait de réduire la taille grâce à deux couches denses de paramètre 1024 transformant ainsi notre vecteur de 25 088 éléments en

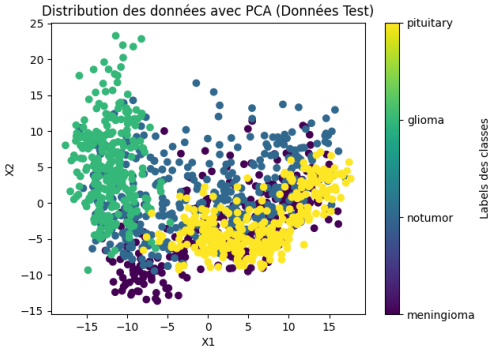
1024, beaucoup plus compact, et permettant d'apprendre des relations plus complexes notamment via la fonction d'activation RELU. Pour les mêmes raisons, nous avons ajouté pour notre modèle utilisant ResNet50, une couche `GlobalAveragePooling`, permettant ainsi de réduire de manière intelligente la dimension de nos données. De plus, nous avons décidé d'appliquer le PCA également sur l'embedding issu de l'utilisation du ResNet50 après avoir vu que le PCA permet d'améliorer les performances de la classification comme indiqué précédemment.

Après ces étapes, notre embedding est près et "peaufiné", et nous pouvons passer à l'étape de la classification.

4 Classification

Afin de pouvoir classifier correctement nos données, il est important de vérifier si nos données sont linéairement séparables ou pas. Ceci nous permettra de choisir le modèle de classification approprié pour notre dataset. Essayons de visualiser à quoi ressemblerait la distribution de nos données en la représentant de façon simplifiée sur un espace 2D.





Comme nous pouvons le voir sur ces 2 graphes, nos données ne sont pas linéairement séparables. Nous avons donc choisis d'utiliser un algorithme vu en cours (Cours 3 "SVM") permettant de classifier exactement ce type de données : Non Linear SVM. Par curiosité, nous nous sommes quand même amusés à utiliser les modèles SVM et Linear Regression vu en cours, bien qu'ils soient conçus pour les données linéairement séparables, afin de voir ce que ça donnerait comme résultat. Nous verrons plus tard que, sans aucune surprise, Non Linear SVM est plus performant sur notre dataset que SVM et Linear Regression le sont. Par ailleurs, pour implémenter la classification multiclasse, nous avons opté pour la stratégie one vs all plutôt que one vs one car celle-ci possède un coût de calcul plus faible.
Comment avons-nous donc implémenter Non Linear SVM ? Nous avons procédé à peu près de la même façon que celle vue dans le TP4, c'est à dire que nous avons mis en place un SVM non linéaire qui a un noyau à fonction de base radiale (RBF), en utilisant la méthode de recherche de grille avec cross-validation pour trouver les meilleurs hyperparamètres C et Gamma.
Bien sûr, l'étape qui est au cœur de ce Non Linear SVM, est de retrouver C et Gamma qui maximisent la performance de notre modèle, i.e l'accuracy sur les images du Test. Comme nous n'avions pas pu trouvé à priori de pistes de réflexion menant à un potentiel bon intervalle dans lequel Gamma pourrait se retrouver pour notre dataset, nous avons décidé qu'il était plus judicieux que Gamma prenne les valeurs 'scale' ou 'auto' plutôt que d'affecter nous même des valeures précises à Gamma. Au moins, nous sommes plus sûrs du potentiel de notre modèle à s'adapter aux caractéristiques de nos images puisque Gamma s'ajustera en fonction de la dispersion de nos données. Pour ce qui est du choix de C, nous avons commencé par un ensemble de valeurs : [0.1, 1, 2, 25, 50], le but n'étant pas de choisir un C excessif qui pourrait nous mener à un overfitting, et de changer l'ensemble des valeurs de C en fonction de celle ayant la meilleure accuracy donnée par la cross validation. Dans le cas où c'est 2 par exemple, nous essayons dans un premier temps des valeurs qui se rapprochent de 2 entre 1 et 2. Si celles-ci n'augmentent pas nos performances, nous nous focalisons sur les valeurs dans l'intervalle [2 ;25[, et ainsi de suite, jusqu'à trouver un C pour lequel l'accuracy est jugée bonne et satisfaisante.

5 Résultats

Dans cette partie nous nous focaliserons plus sur l'analyse des résultats, en fonction des différents paramètres.

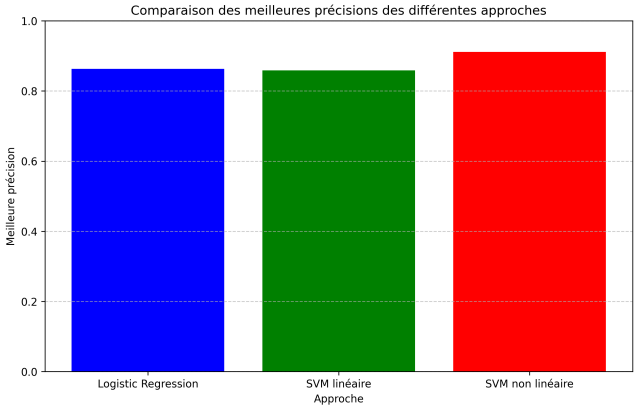
D'abord, il faut savoir que la meilleure stratégie que nous avons pu constaté,

c'est-à-dire, celle menant à une meilleure accuracy, est la suivante : retrouver l'Embedding en utilisant ResNet50, ensuite appliquer le PCA et enfin classifier en utilisant le Non Linear SVM.

Nous allons commencer par illustrer la différence entre les résultats des meilleures accuracy lorsque nous utilisons la même approche mais avec 3 algorithmes différents pour la classification :

— ResNet50, PCA et Non Linear SVM.
— ResNet50, PCA et Linear SVM.
— ResNet50, PCA et Logistic Regression.

```
Meilleure précision de Logistic Regression: 0.8627002288329519
Meilleure précision de SVM linéaire: 0.858886346300534
Meilleure précision de SVM non linéaire: 0.9115179252479023
```



Comme nous nous y attendions, l'approche utilisant Non Linear SVM donne une meilleure accuracy que celle du Linear SVM et Logistic Regression puisque Non Linear SVM est plus adapté au caractère non linéaire de notre dataset. Cependant, cette curiosité de tenter le Linear SVM et Logistic Regression nous a mené à une observation intéressante : ces 2 modèles conçus pour des datasets linéairement séparables ont une accuracy supérieure à 0,85 sur notre dataset non linéairement séparable ce qui est très surprenant ! Comment expliquer cela ? Comme nous l'avons vu dans le cours 3 (diapo 39 sur 57), nous pensons que ce résultat est lié au fait que l'usage d'un embedding peut permettre à un modèle linéaire de trouver la barrière de séparation non linéaire du dataset non linéaire.

Ensuite nous aurions voulu illustrer la différence entre les résultats des meilleures accuracy lorsque nous utilisons la même approche mais avec 2 modèles entraînés différents, VGG16 et ResNet50, mais ça n'aurait pas été cohé-

rent puisque les accuracy de chacun dépends des hyperparamètres différents. Notamment des hyperparamètres qui donnent une bonne accuracy en utilisant ResNet50 ne donneront pas forcément une bonne accuracy en utilisant VGG16 et vice versa, puisque les embeddings sont différents et les hyperparamètres appropriés le seront également. Cependant, nous avons remarqué qu'en général nous parvenons à retrouver de meilleurs accuracies en utilisant ResNet50 que VGG16. Celà est potentiellement lié à la nature de nos images qui sont des IRM. Comme VGG16 est plus entraîné sur des images plus générales qui n'ont pas forcément rapport avec les IRMs, ResNet50 semble être beaucoup plus approprié pour le traitement d'IRM.

Après avoir exploité notre dataset, nous nous sommes demandé comment notre modèle se comporterait sur un autre dataset. Nous avons alors importé le modèle Kaggle Brain Tumor Classification de Sartaj datant de 2020. Celui-ci s'organise de la même façon que notre dataset (même nombre de classes de tumeurs) même si les données ne sont pas bien proportionnées et moins nombreuses (environ deux fois moins de données que le dataset initial). En effet, par exemple, seulement 13% des images d'entrainement sont de types notumeur. Les résultats de notre modèle sur ce dataset sont beaucoup moins convaincants. Nous obtenons une accuracy optimale de 0.74 sur les images d'entrainement et une accuracy de 0.66 sur le test. Une piste d'amélioration serait de faire de l'augmentation de données afin de pouvoir entrainer notre modèle sur un ensemble plus important de données.

6 Conclusion

Pour conclure, ce projet a été particulièrement instructif car il nous a permis de pouvoir véritablement visualiser le rôle du pca, et des CNNs dans les Embeddings. Nous avons pu également comprendre les différences et similarités entre les différents algorithmes de classification. Ces contrastes vont au-delà de la linéarité/non linéarité du dataset traité, car nous avons pu voir les subtilités dans leur utilisation et performance. Notre modèle donne une bonne accuracy pour notre dataset (0,91), mais nous avons vu que lorsque testé sur un dataset moins conséquent, l'accuracy devient faible. Pour essayer de généraliser notre modèle à des datasets de MRI de tumeurs différentes, la Data Augmentation serait une bonne piste pour améliorer l'accuracy notamment dans le cadre des volumes de données faibles.