Noé Mosca

Rapport final de projet : Projet de IMA05

Sommaire

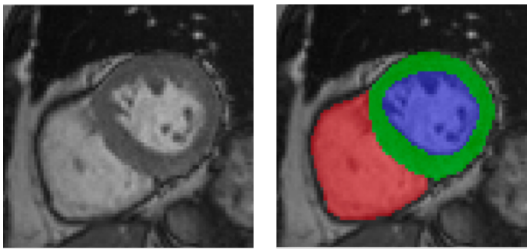
I Présentation du projet

Le projet sur lequel nous travaillons étudie la prédictions de pathologies cardiaques à partir d'IRM du coeur. Les 5 classifications possibles du coeur sont :

1) Contrôles sains (label 0)
2) Infarctus du myocarde (label 1)
3) Cardiomyopathie dilatée (label 2)
4) Cardiomyopathie hypertrophique (label 3)
5) Ventricule droit anormal (label 4)

Pour ce faire nous avons à notre disposition un ensemble de données d'entrainement composé de 100 patients, avec pour chaque patient :

- l'IRM au moment de la fin de la diastole (juste avant la contraction)
- l'IRM au moment de la fin de la diastole segmentée
- l'IRM au moment de la fin de la systole (juste après la contraction)
- l'IRM au moment de la fin de la systole segmentée
- Pour chaque patient le label de sa pathologie



2D slice of CMR image — Left ventricle cavity, myocardium, right ventricle cavity & background

Exemple de coupe extraites des fichiers, et de la segmentation :

Le but de se projet est alors de déterminer, à l'aide de 50 patients test (dont on possède les même donnée que pour les données d'entrainement sauf le label de la pathologie), le label de la pathologie.

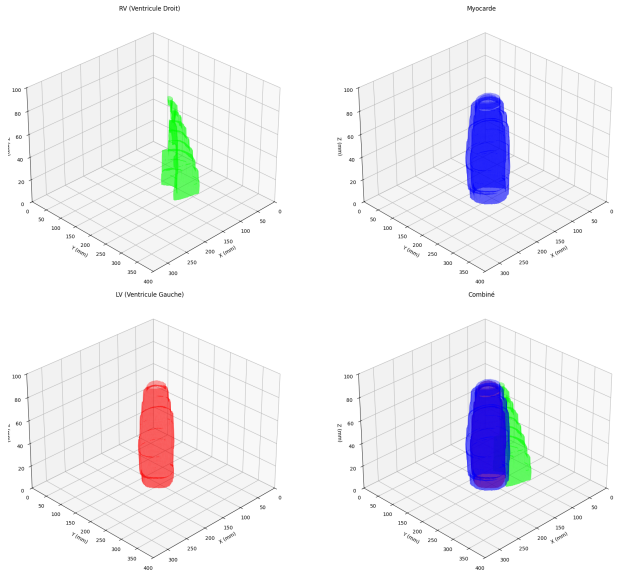
II Pré-process

a) Data visualisation

Pour commencer visualisons le jeu de données que nous avons à disposition.

On observe bien les 4 zones distinctes du coeur.

Pour les données de test, le ventricule gauche n'est pas défini, on doit donc elle rajouter à la main en remplissant l'intérieur du myocarde.



b) Choix des features

Les features ont beaucoup d'importance ici car ils vont servir de base pour les modèles à entrainer.
Initialement nous n'avons à disposition que les images 3D des IRM du coeur avec leur segmentation, la taille et le poids des patients ainsi que leur catégorie de maladie. On doit extraire des features pertinents pour entraîner notre modèle par la suite.

Pour les features on a utilisé ceux-ci, en se basant sur des articles issus de publications *Springer* ainsi que de publications *PubMed*. L'objectif est de récupérer le plus de bio marqueurs pour affiner notre modèle.

Feature	Source	N°
Volume du ventricule gauche en fin de remplissage	article 1	1
Volume du ventricule gauche en fin de contraction	article 1	2
Volume du ventricule droit en fin de remplissage	article 1	3
Volume du ventricule droit en fin de contraction	article 1	4
Masse du muscle cardiaque	article 1	5
Masse du VG normalisée par la surface corporelle	article 1	6
Poids du patient	article 1	7
Taille du patient	article 1	8
Surface corporelle estimée par la formule de Mosteller	article 1	9
Fraction d'éjection du ventricule gauche	article 2	10
Ratio entre le volume du ventricule droit et gauche à l'ED	article 2	11
Fraction d'éjection du ventricule droit	article 3	12
Masse myocardique en ED + volume en ES	article 3	13
Volume d'ejectiion systolique	article 4	14
Volume VG systolique indexé	article 4	15
Volume VD diastolique indexé	article 5	17
Ratio VD/VG en systole	article 5	18
Volume VG diastolique indexé	article 4	19
Volume VD systolique indexé	article 5	20
Rapport masse myocardique / volume VG en diastole	article 5	21
Rapport masse myocardique / volume télédiastolique	article 6	22
Ratio masse en systole / masse en diastole	Pas de mention spécifique	23

Pour le process on va utiliser une méthode ensemble qui vise à combiner plusieurs classifier faibles en ce basant sur l'implémentation de cet article.

c) Data augmentation

Le jeu d'entraînement qui nous est proposé ne contient pas beaucoup de donnée. Pour palier à ce problème on utile une méthode de data-augmentation citée dans cet article.

Le but est de générer plus de données d'entraînement et de rendre notre modèle plus robuste et plus général, en l'exposant à plus de variété, sans collecter plus de données.

Pour chaque image, on va créer 4 images supplémentaire en appliquant aléatoirement les transformations suivantes :

- rotation avec un angle compris entre -5° e 5° (uniforme)
- Translation selon l'axe x : shift aléatoire entre -5mm et 5mm (uniforme)
- Translation selon l'axe y : shift aléatoire entre -5mm et 5mm (uniforme)
- Changement d'échelle : zoom avec un facteur aléatoire compris entre 0.6 et 1.4 (uniforme)
- Flip horizontal et vertical selon une loi de Bernouilli

On travaille au final avec uns et de 500 images 3D (.nii) au moment de la diastole, et 500 au moment de la systole.

III Process

Pour le process on va combiner plusieurs réseaux de neurones faibles avec un autre classifieur, et on fera ensuite une moyenne pondérée pour déterminer la sortie. On testera plusieurs classifier faibles que l'on peut combiner avec les MLPS : Random Forest, XGBoost, SVM et KNN.

a) Réseau de neurones

Le premier classifier qu'on va combiner est un réseau de neurones avec 4 couches cachées.
Dans chaque couche il y a :

- un **BatchNorm** pour faciliter l'apprentissage, cela de régulariser chaque couche pour éviter que le gradient explose ou disparaisse
- une activation non-linéaire par **LeakyReLU**
- un ajout de **bruit Gaussien** pour rendre le modèle plus robuste.

On entraîne alors 50 MLP comme celui-ci. À chaque-fois, le MLP ne voit que 75% des données d'entraînement. Cela permet de décolérer l'apprentissage, à chaque fois itération le réseau de neurones n'apprend pas le même chose que le précédent.

Il y a 200 epochs, la taille du batch est de 50 à chaque fois, la loss est Adam et le learning rate vaut 10^{-3}.

Résultats

On obtient en résultat une accuracy de 70%, ce qui semble assez faible, et montre surtout que le modèle à besoin d'être amélioré.

Améliorations

Pour améliorer le résultats on va faire un grid search, qui permet d'obtenir les meilleurs paramètres ici. On utilise alors les données suivantes pour le grid-search :

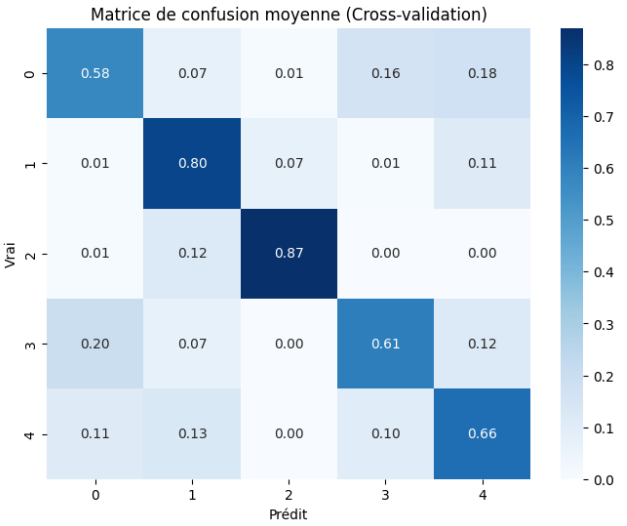
Paramètres	Valeur 1	Valeur 2
Nombre de MLP	50	100
Nombre d'epochs	30	100
Valeur de lr	10^{-3}	5×10^{-4}
Taille du batch	30	60
Taille du subset pris au début	75 %	90 %

Le résultat est obtenu pour :

- **50 MLPs**
- **100 Epochs**
- **learning rate de 1e-3**
- **60 batch**
- **sous set de 90% pris à chaque fois.**

Néanmoins on obtient une accuracy proche de 0.7 080, ce qui reste toujours trop bas.

La matrice de confusion nous montre notamment que seuls les classes 1 et 2 sont correctement classifiées avec une accuracy de 0.80 et 0.87 respectivement. Pour les autres classes l'accuracy est en dessous de 0.7, ce qui est trop faible, on cherche un résultat plus proche de 0.95.



Proposition de changement

Le problème principal semble venir ici du data augmentation. En effet quand la taille du set initial est inchangé, obtient des résultats beaucoup plus convaincants.

Dans notre cas, l'application de transformations aléatoires sur les images (telles que des rotations, des flips ou d'autres modifications géométriques) a conduit à une dégradation significative des performances du modèle, avec une chute de l'accuracy. Cela s'explique principalement par le fait que certaines de ces transformations ont altéré la structure physiologique des images, rendant certaines d'entre elles incohérentes par rapport à leur label d'origine. Par exemple, un retournement horizontal peut inverser la latéralité et rendre l'étiquette incorrecte sans ajustement correspondant. De plus, un ajout excessif de données transformées peut déséquilibrer le jeu d'entraînement en introduisant un bruit artificiel qui dilue le signal utile. Enfin, le modèle, confronté à des exemples parfois irréalistes ou ambigus, apprend des associations erronées qui pénalisent sa capacité à généraliser. Ces effets cumulés expliquent pourquoi l'utilisation non maîtrisée de l'augmentation des données peut parfois détériorer les performances au lieu de les améliorer. Il est donc crucial d'adapter soigneusement les stratégies d'augmentation aux caractéristiques spécifiques des données et de la tâche cible.

Par la suite on ne travaillera donc plus avec le jeu de données transformé, car les transformations appliquées manquent de précaution et faussent nos résultats.

Avec les données initiales

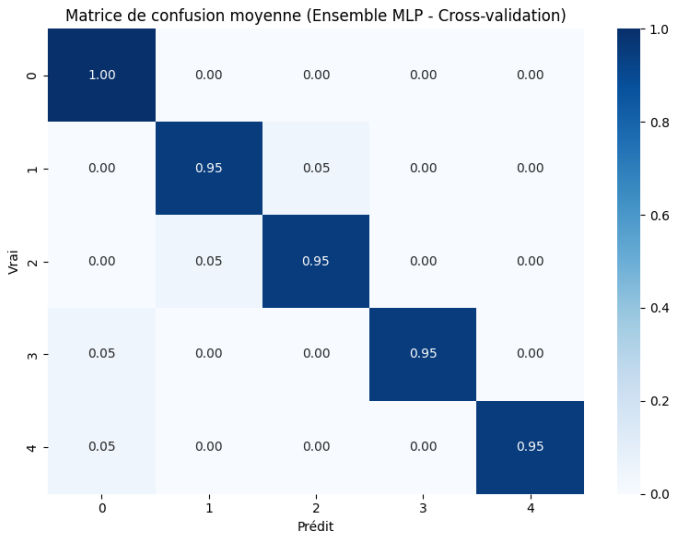
On fait alors comme précédemment, et on refait un grid-search avec les paramètres suivants, en fixant cette fois-ci le nombre de MLP à 50, car il ne semble pas pertinent pour la suite de travailler avec plus de MLP et cela augmente le temps de calcul pour des résultats similaires.

Paramètres	Valeur 1	Valeur 2	Valeur 3
Nombre d'epochs	100	200	500
Valeur du learning rate	10^{-3}	5×10^{-4}	10^{-4}
Taille du batch	30	60	/
Taille du subset pris au début	75 %	90 %	/

On obtient les meilleurs résultats pour plusieurs tuples de données. On prendra donc par la suite :

- **100 epochs**
- **learning rate** $= 10^{-3}$
- **batch size = 60**
- **75% de subset pris à chaque fois**

La cross validation nous donne les résultats suivants pour l'accuracy :



Avec une moyenne d'accuracy de **0.96.** Les résultats obtenus ici sont beaucoup plus convaincants que ce obtenus précédemment en prenant le jeu de donné augmenté aléatoirement.

c) Random Forest

Le 2e classifier faible que l'on est un Random Forest.

Comme précédemment on cherche les meilleurs paramètres à l'aide d'un gridsearch pour les valeurs suivants :

Paramètres	Val 1	Val 2	Val 3	Val 4	Val 5
Nombre d'arbres	50	100	200	300	500
Profondeur Maximal	None	10	20	/	/
Nb de features regardé à chaque split	sqrt	log	/	/	/

Les meilleurs résultats sont obtenus pour :

- **500 estimateurs**
- **pas de profondeur max**
- **un max de features qui s'arrête à racine carrée ou log du nombre total de features**

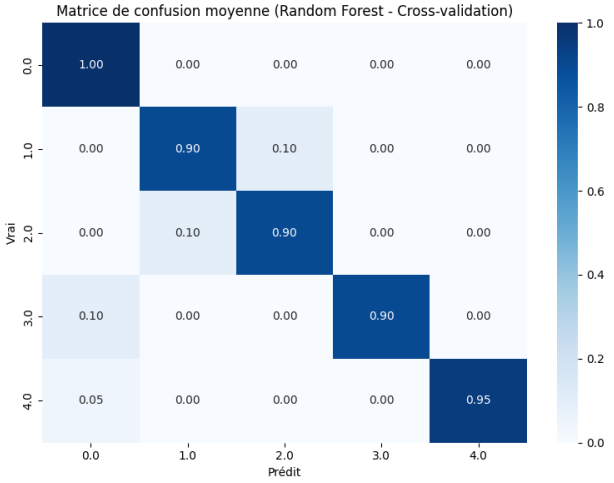
L'accuracy moyenne obtenue est alors de **0.94**, ce qui est globalement bien.

L'idée ici est d'appliquer une méthode ensemble combinant un réseau de neurones initial avec un Random Forest de 200 arbres. Nous limitons volontairement le nombre d'arbres afin de ne pas obtenir un modèle trop complexe. En effet, bien que le Random Forest soit robuste, un trop grand nombre d'arbres, couplé à des arbres très profonds, peut finir par capturer des motifs rares ou du bruit, ce qui augmenterait le risque d'overfitting. Nous préférons ainsi limiter à 200 pour favoriser la généralisation

On prendra pour la suite

- **200 estimateurs**
- **pas de profondeur max**
- **un max de features qui s'arrête à racine carrée ou log du nombre total de features**

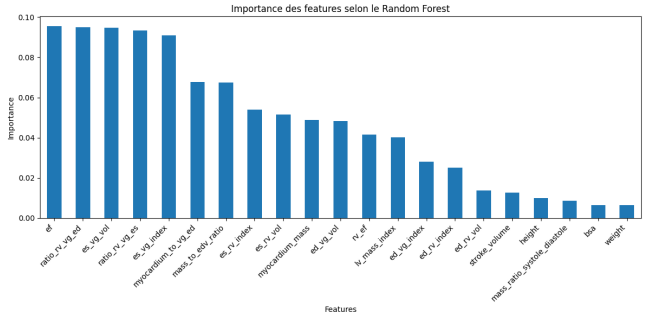
On obtient la matrice de confusion moyenne suivante :
Avec une accuracy moyenne de **0.93**.

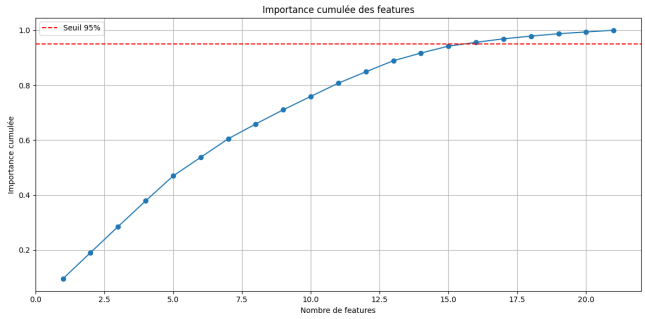


Améliorations possibles

Contrairement au réseau de neurones, le Random Forest peut se révéler sensible au bruit lorsque ses arbres deviennent trop spécifiques. En effet, un modèle trop complexe risque de mémoriser des détails non pertinents présents dans les données d'entraînement, ce qui nuirait à sa capacité de généralisation et dégraderait les performances en prédiction. Afin de limiter ce risque, nous pourrions ici conserver uniquement les features représentant 95 % de l'importance cumulée. Cette sélection vise à éliminer les variables les moins informatives et ainsi réduire l'impact potentiel du bruit sur l'apprentissage.

On trace donc l'importance de nos features et on obtient :





En ne prenant alors que les 16 features les plus importants, on obtient une accuracy de **0.91** en moyenne sur la validation croisée. Cette accuracy est moins bonne que précédemment mais on peut penser qu'elle est plus générale pour la prédiction finale.

Néanmoins comme nous avons déjà limité le nombre d'arbres pour que le modèle soit plus générale, nous ne nous attarderons pas sur le Random Forest avec le nombre de features réduit.

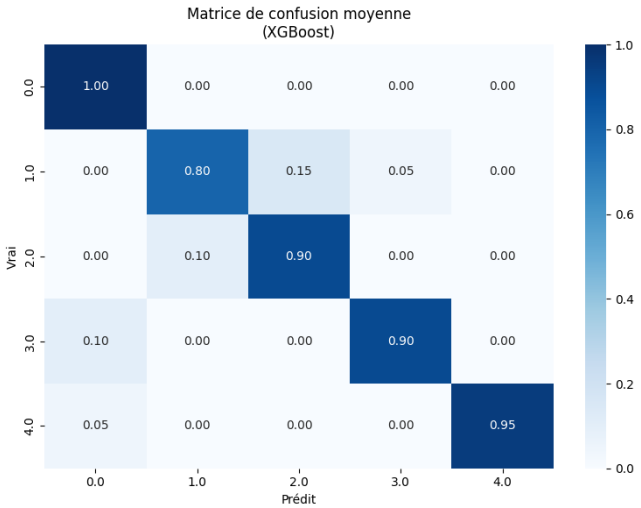
d) Autres estimateurs

L'estimateur que l'on va combiner avec le MLP pour la méthode ensemble peut-être un Random Forest tout comme un autre estimateur. L'idée est ici d'en essayer plusieurs, de récupérer leur meilleur paramètre et de voir par la suite lequel conviendra le mieux. On testera pour cela :

- XGBoost
- SVM
- KNN

XGBoost

Contrairement à un Random Forest, qui repose sur l'agrégation de nombreux arbres construits indépendamment (bagging), XGBoost adopte une stratégie de boosting séquentiel : chaque nouvel arbre vise à corriger les erreurs résiduelles de ses prédécesseurs, améliorant ainsi l'apprentissage des exemples difficiles.
Cette méthode intègre une régularisation L1/L2 explicite et un paramètre de complexité (gamma) limitant le surapprentissage, ce qui est particulièrement avantageux sur de petits jeux de données bruités (et ce qui est notre cas). Grâce à son implémentation optimisée (parallélisation, gestion native des valeurs manquantes, fonctions de perte configurables), XGBoost peut bien se Combiner avec un réseau de neurones : ce dernier modélise les interactions non linéaires complexes, tandis que XGBoost affine les corrections résiduelles et renforce la robustesse globale de l'ensemble.



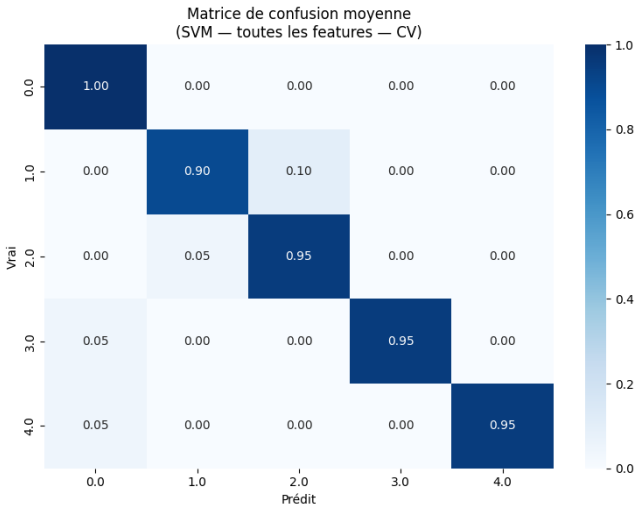
On obtient pourtant résultats moins significatifs que pour le Random Forest. Après avoir fait un grid-search sur les différents paramètres, et une cross validation avec les paramètres optimaux, on obtient une accuracy moyenne de **0.91**.
On remarque un faiblesse particulière du modèle à classifier les labels 1, avec seulement 80% de vrais.

Bien qu'en théorie ce modèle soit plus poussé, le Random Forest semble plus pertinent que le XGBoost dans notre cas.

SVM

Le Support Vector Machine (SVM) résout un problème convexe garantissant un optimum global et maximise la marge entre classes, assurant une forte généralisation même avec peu d'échantillons. Grâce au « kernel trick », il capture efficacement des relations non linéaires sans alourdir la complexité, et le paramètre C ajuste précisément le compromis biais-variance. Enfin, seule la fraction des vecteurs de support détermine le modèle, rendant l'inférence économe en mémoire. Combiné à un réseau de neurones, le SVM peut exploiter les représentations hiérarchiques apprises et affiner la décision finale pour une robustesse accrue.

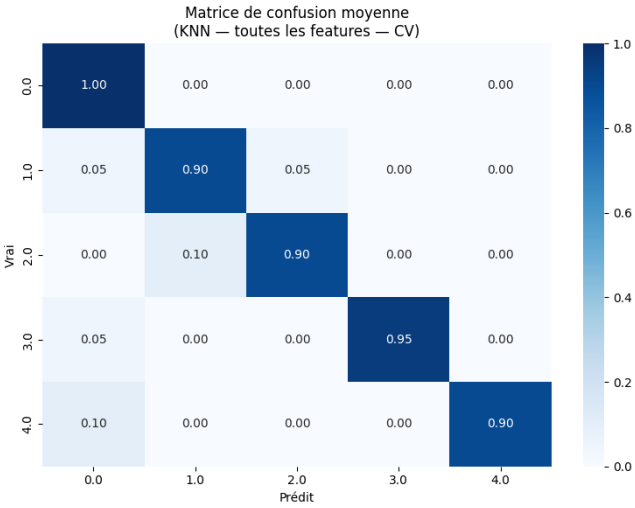
L'accuracy finale obtenue est très pertinente, on obtient au finale une accuracy de **0.95** avec un SVM linéaire. On conserve ce modèle pour la suite des tests.



KNN

Contrairement aux méthodes paramétriques, le k-Nearest Neighbors (kNN) est un estimateur à base d'exemples qui ne s'appuie pas sur un modèle préalablement entraîné, mais conserve l'intégralité du jeu d'apprentissage. Pour chaque nouvelle observation, il identifie les k points les plus proches dans l'espace des features (euclidien, Manhattan, etc.) puis agrège leurs étiquettes par vote majoritaire ou moyenne, ce qui lui permet de capturer naturellement les structures locales et les non-linéarités sans hypothèse de forme. En revanche, cette simplicité se paie par un coût de calcul et de mémoire élevé lors de l'inférence, surtout sur de grands ensembles de données. Mis en combinaison avec un réseau de neurones, kNN profite des représentations denses et hiérarchiques que celui-ci apprend, tandis que le MLP bénéficie de la robustesse locale du kNN pour affiner la décision finale.

En considérant 3 voisins avec des poids uniformes et une distance de « manhattan » on obtient une accuracy de **0.93** par validation croisée. Ce résultats plus faible que celui obtenu avec le Random Forest n'en fait pas un meilleur candidat : on ne le retient pas.



Résultats

La pipeline que initiale visait combiner le réseau de neurones avec un autre classifier faible, qui était dans ce cas un Random Forest. Après en avoir testé plusieurs, seul le SVM propose des résultats qui le surpasse. On testera donc les 2 dans la méthode ensemble qui suit, en laissant de côté le XGBoost et le KNN.

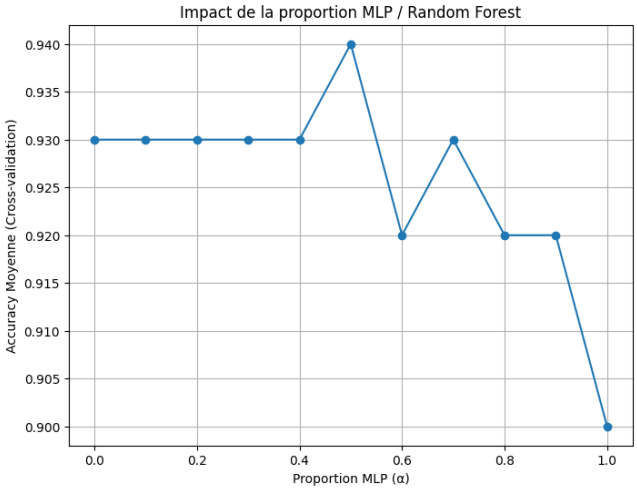
IV Méthode ensemble

a) MLP et Random Forest (RF)

Pour commencer étudions le Random Forest combiné avec le MLP. L'idée est de trouver le

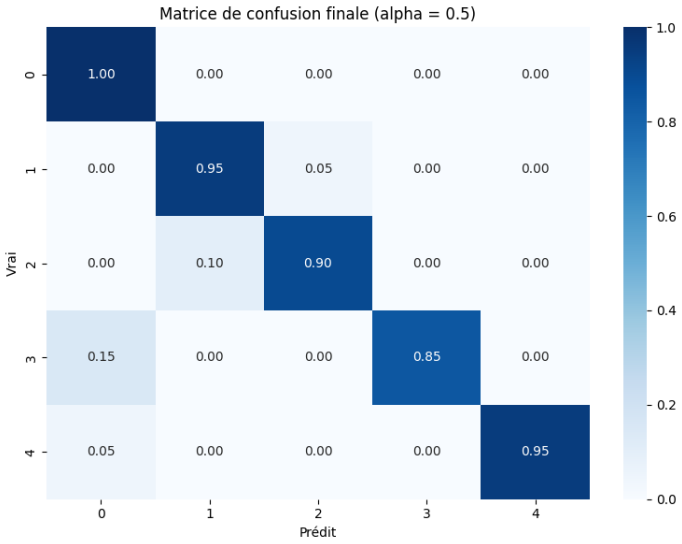
meilleur paramètre α pour la répartition $Res = \alpha MLP + (1 - \alpha)RF$.

Si on trace l'accuracy obtenir par validation croisée en fonction de la valeur de α on obtient :



On remarque alors que la meilleurs valeur est obtenue pour $\alpha = 0.5$, soit pour une moyenne entre le MLP et le Random Forest.

Les détails des classifications sont données dans la matrice de confusion ci-dessous :



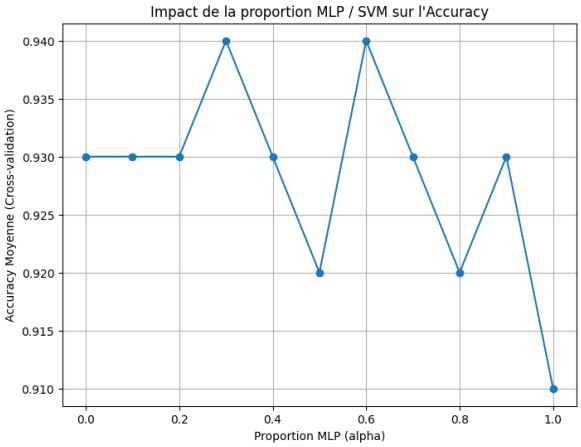
Bien que le score finale soit relativement bon, l'accuracy finale, les résultats obtenus restent assez surprenant. On retrouve bien l'accuracy obtenue par cross-validation quad $\alpha = 0$, soit l'accuracy du Random Forest seul, mais on ne retrouve pas celle du réseau de neurone seul quand $\alpha = 1$.

Notons quand même que les résultats précédent soulignent l'intérêt de combiner les deux, c'est bien la combinaison du MLP et du RF qui donne le meilleur résultat, quand la pondération est la même pour chacun.

b) MLP et SVM

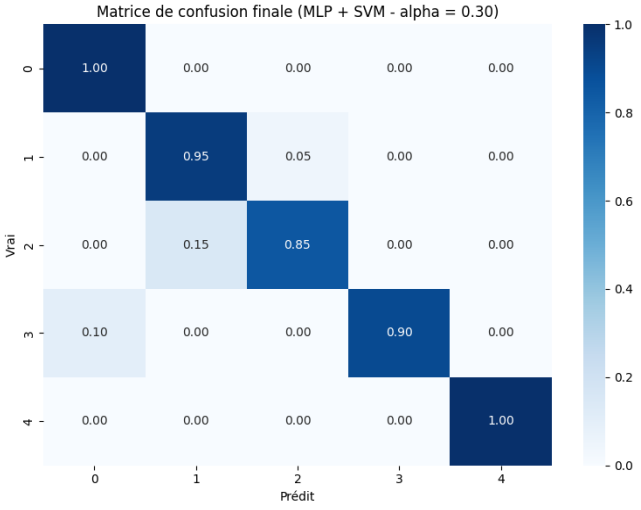
On a vu précédemment qu'il était utile de considérer le SVM comme classifier à combiner avec le MLP.

Si on fait comme précédemment et qu'on trace l'accuracy obtenue par validation croisée en fonction de α pour le résultat $Res = \alpha MLP + (1 - \alpha)SVM$ on obtient :



On remarque ici que contrairement au cas précédemment, la meilleure pondération n'est pas celle pour laquelle α vaut 1/2, mais 0.3 ou 0.6, c'est à dire un résultat pour lequel on accorde plus d'importance au MLP que au SVM ou l'inverse.

L'accuracy obtenue dans le meilleur cas est toujours de **0.94,** la combinaison du SVM ou du Random Forest avec le MLP est intéressante dans les 2 cas.



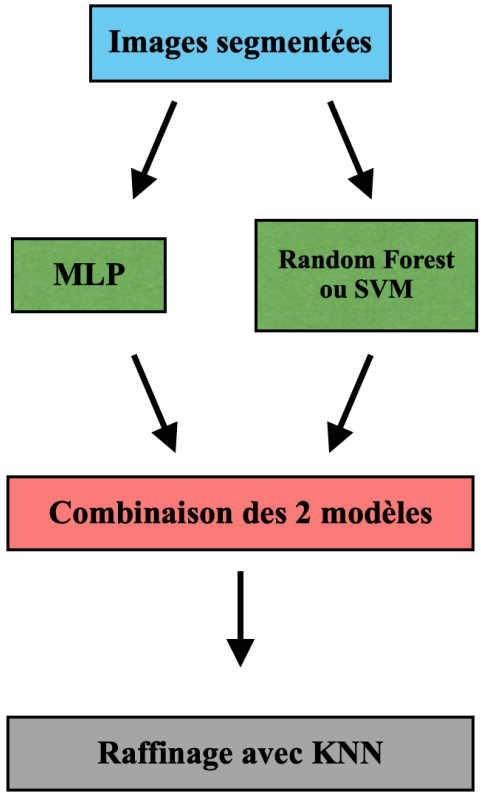
V Post processing

Quand on regarde toutes les cross-validations qui ont été faites, on se rend compte que nos modèles ont généralement du mal à classifier les labels 1, 2, et 4. Dans notre cas, on peut penser que les images classées en labels 1 et 2 sont assez proches, et qu'il faudrait réaffirmer le modèle.
Pour cela, nous souhaiterions intégrer un module de post-traitement dédié à la reclassification de ces cas limites. Une solution pertinente serait d'utiliser un K-Nearest Neighbors (KNN) entraîné directement sur les sorties du modèle principal pour capter les motifs fins et les distributions locales des classes. Ce KNN permettrait d'affiner la décision lorsque le modèle hésite entre les labels 1 et 2, voire de corriger des erreurs résiduelles. Ce mécanisme agirait ainsi comme un filet de sécurité pour renforcer la précision globale sur ces classes particulièrement délicates.

(Cette partie n'a pas été implémentée mais est proposée comme solution pour augmenter le nombres de données de test bien classifiées, dont un problème principal est la confusion entre les labels 1 et 2).

VI Conclusions

Pour notre problème de classification nous avons opté finalement pour la pipeline suivante :



(non implémenté)

Après avoir réalisé de nombreux essais et comparé différents modèles, le Random Forest utilisant l'ensemble des features s'est révélé être le plus performant. Cela peut s'expliquer par plusieurs raisons. Tout d'abord, le Random Forest est particulièrement robuste face aux données hétérogènes et aux éventuelles corrélations entre les variables. Grâce à sa méthode d'agrégation (bagging), il réduit considérablement la variance du modèle tout en évitant le surapprentissage, à condition que le nombre d'arbres et la profondeur soient maîtrisés. De plus, contrairement à des modèles comme le SVM ou les réseaux de neurones, qui nécessitent souvent un ajustement précis des hyperparamètres et une normalisation stricte des données, le Random Forest gère naturellement les données bruitées et ne dépend pas fortement de la mise à l'échelle des variables. Enfin, dans des contextes où il existe de nombreuses features potentiellement utiles, l'approche d'ensemble des arbres permet de capter efficacement diverses interactions complexes entre les variables, offrant ainsi une capacité de généralisation souvent supérieure sans nécessiter de pipeline trop sophistiqué.

Toutefois, il est possible que les performances de notre modèle globale auraient pu être encore améliorées par un affinage postérieur des prédictions, notamment en reclassifiant certains labels proches comme 1 et 2, via un modèle léger en post-processing. Cette étape n'a cependant pas été mise en œuvre dans notre pipeline actuelle.

VII Méthodes utilisées

Les principales méthodes et fonctions utilisées dans ce projet sont les suivantes :

Prétraitement des données : numpy, pandas pour la manipulation des données, StandardScaler pour la normalisation.

Modèles de classification :

•Random Forest avec sklearn pour une classification robuste.
•SVM (Support Vector Machine) pour une alternative linéaire avec estimation des probabilités.
•MLP (réseau de neurones) conçu sous PyTorch, utilisé en ensemble de 50 modèles pour améliorer la robustesse via une moyenne des prédictions.

Validation et évaluation :

•StratifiedKFold pour la validation croisée.
•accuracy_score et confusion_matrix pour mesurer les performances.

Ensembles et fusions de modèles :

•Combinaison des MLP et de la Random Forest ou du SVM via un facteur alpha pour exploiter les forces de chaque modèle.
•Sauvegarde des modèles : joblib pour stocker et réutiliser les modèles entraînés.

Post-processing envisagé : Une méthode d'affinage des labels, notamment entre les classes proches comme les labels 1 et 2, a été envisagée mais non mise en œuvre.