Rapport de projet

Mai 2025

IMA205

Cardiac Pathology Prediction

Clément Gilli

Table des matières

1 Introduction

Le but de ce projet, dans le cadre d'un challenge Kaggle [1], est de développer un algorithme capable de classifier automatiquement des patients à partir d'IRM cardiaques, en les répartissant dans l'une des cinq catégories suivantes :

1. Normal (NOR)
2. Infarctus du myocarde (MINF)
3. Cardiomyopathie dilatée (DCM)
4. Cardiomyopathie hypertrophique (HCM)
5. Anomalie du ventricule droit (ARV)

Ces pathologies affectent la structure et le fonctionnement du cœur de manière différente. Par exemple, l'infarctus du myocarde correspond à une zone de tissu endommagé suite à un manque d'oxygène [2]. La cardiomyopathie dilatée se manifeste par un élargissement des cavités cardiaques [3], tandis que la cardiomyopathie hypertrophique est caractérisée par un épaississement des parois du cœur [4]. L'ARV concerne principalement des altérations du ventricule droit, souvent visibles au niveau de sa forme ou de sa taille.

Pour distinguer ces différentes classes, nous exploitons des IRM cardiaques acquises à deux moments du cycle cardiaque : la fin de la diastole et la fin de la systole. À partir de ces images, nous extrayons un ensemble de features morphologiques et cliniques, utilisées ensuite pour entraîner des modèles de classification supervisée.

2 Data

Nous utilisons un petit dataset de 150 patients divisé en un training-validation set (100 patients) et un test set (50 patients), en ayant seulement accès aux labels du training set (c'est-à-dire les diagnostics des patients).

Pour chaque patient, nous avons accès à deux IRM (qui sont des images en 3D) correspondant à deux moments du cycle cardiaque du cœur : une à la fin de la diastole (fin de la dilatation, noté *ED*) et une à la fin de la systole (fin de la contraction, noté *ES*).

En complément de ces deux IRM, nous avons aussi, pour chacune d'entre elles, la segmentation associée de trois éléments de l'anatomie cardiaque (avec les labels donnés) : la cavité du ventricule droit (RV), le myocarde (MYO) et la cavité du ventricule gauche (LV) (Figure 1). À noter que pour le test set, nous n'avons pas la segmentation de la cavité du ventricule gauche.

Pour chaque IRM, il est également fournie une donnée très importante pour la suite qu'il ne faut surtout pas oublier : la résolution des voxels en mm. En effet, chaque IRM a une résolution propre à la machine utilisée, il faut donc prendre en compte ce paramètre pour construire des features robustes et classifier correctement les patients.

Enfin, nous avons aussi accès à deux méta-données : la taille et le poids de chaque patient.

Pour gérer toutes ces données de manière pratique et agréable pour la suite du projet, nous avons utilisé la bibliothèque Python *TorchIO* qui permet de manipuler des objets *Subject*, dans lesquels nous pouvons stocker pour chaque patient les deux IRM ainsi que leur segmentation, le poids et la taille, la résolution des voxels, ainsi que le diagnostic (dans le cas du training set).

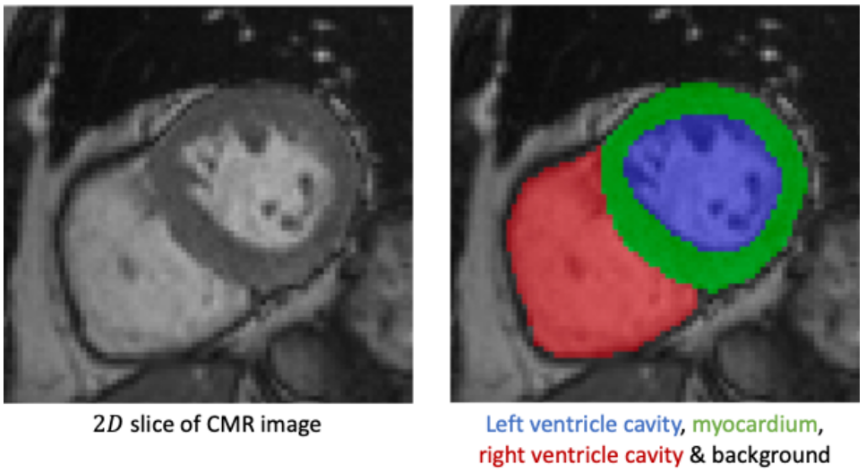

FIGURE 1 – Slice d'une IRM et sa segmentation associée.

3 Preprocessing

À présent que nous pouvons manipuler efficacement nos données, il ne reste plus qu'à faire un très simple preprocessing.

En effet, le dataset fourni est très généreux : il nous donne directement la segmentation des trois éléments de l'anatomie cardiaque qui vont nous servir pour la création de nos features. Cependant, comme nous l'avons fait remarquer plus tôt, nous n'avons pas accès à la segmentation de la cavité du ventricule gauche pour les patients du test set. Nous devons donc impérativement obtenir cette segmentation si nous voulons utiliser des features qui l'utilisent.

Au premier abord, cela peut sembler assez évident puisque nous avons accès au reste de la segmentation. En effet, l'idée naturelle est de remplir l'intérieur de la segmentation du myocarde en travaillant slice par slice, mais comment ? Il y a une fonction de *skimage* qui permet de remplir toute une zone uniforme jusqu'à ce qu'on rencontre un bord : *flood_fill*. Cependant, pour utiliser cette fonction, nous avons besoin d'un point à l'intérieur de la cavité du ventricule gauche, mais ce n'est pas toujours trivial à trouver, comme dans certains cas où la cavité est extrêmement petite.

Nous avons donc trouvé une astuce assez simple pour cela : en supposant que la segmentation du myocarde soit parfaite (qu'elle soit sans trou et donc sépare bien le LV du reste de l'image), il suffit de remplir la zone extérieure avec une valeur que nous n'utilisons pas pour les labels (par exemple -1) simplement en choisissant un point dans cette zone : le point $(0, 0)$. Il n'y a plus qu'à attribuer les bons labels au background et à la cavité du ventricule gauche. Enfin, pour ne pas perdre de temps de calcul, les segmentations sont calculées une unique fois, puis sauvegardées en fichier .nii.

Avec la taille petite de notre test set (50 données), nous avons pu vérifier si les segmentations obtenues étaient correctes : sur toutes les slices de toutes les IRM (aux phases ED et ES), seulement une seule (slice) n'était pas bonne. En effet, la segmentation du MYO de cette slice donnée initialement n'était pas complètement fermée, et donc le LV n'a pas changé de label (était toujours *background*). Comme cela touchait seulement une slice, nous avons décidé de ne pas corriger ce problème, car assez anecdotique (la solution étant d'utiliser une fonction du type *binary_fill_holes* de *scipy*, mais dans notre cas ce n'était pas nécessaire).

Ainsi, nous avons donc la segmentation de chaque partie qui nous intéresse, que ce soit dans le training ou bien le test set. Comme nous allons le voir dans la suite, toutes nos features vont uniquement reposer sur ces segmentations et pas du tout sur les IRM. Les IRM sont utiles seulement pour segmenter les parties qui nous intéressent, or ici, cela était déjà presque entièrement fait. À partir de maintenant, nous travaillerons donc exclusivement sur les images segmentées, c'est pourquoi aucun autre preprocessing (en tout cas des images) n'est utile.

4 Feature extraction

4.1 Feature search

Pour trouver des features efficaces, nous avons commencé par lire les différents papiers fournis. D'un côté, il y a Khened [5] et Wolterink [6] qui utilisent une dizaine de features à-peu-près similaires. De l'autre côté, il y a Isensee [7] qui utilise environ une cinquantaine de features, qui englobent celles de Khened et Wolterink.

Nous avons donc pensé au premier abord qu'il était une bonne idée d'implémenter toutes les features données par Isensee, supposant que plus de features impliquent de meilleurs résultats. Or, ce n'est pas toujours vrai, des features peuvent être redondantes ou bien simplement inutiles, ce qui pourrait faire baisser les performances de notre futur modèle.

Le dernier papier fourni [8] compare justement les performances de ces différents papiers. Nous aimerions donc choisir les features correspondant au papier ayant les meilleurs résultats. Cependant, il ne faut pas oublier que ces trois papiers basent leurs classifications à partir de leurs propres segmentations : ils n'avaient pas accès au dataset parfait comme nous qui était déjà segmenté. C'est pourquoi nous pouvons penser que la qualité de la segmentation de ces papiers a un impact direct sur les performances de leur classification. Le deuxième point à noter, si nous voulons pouvoir comparer ces trois papiers, est le modèle utilisé. Les trois utilisent un algorithme de Random Forest, donc ce n'est pas le modèle qui joue sur les performances.

Ainsi, en regardant les différents résultats de segmentation, nous voyons que Isensee obtient les meilleurs, suivi d'un peu plus loin par Khened et Wolterink qui obtiennent également de très bons résultats. Cependant, bien que Isensee obtienne de meilleurs résultats pour la segmentation, Khened obtient de meilleurs résultats pour les métriques cliniques, ce qui est fondamental pour la classification. Et d'ailleurs, nous voyons qu'il obtient de bien meilleurs résultats pour la classification, avec une accuracy de 96%, contre 92% pour Isensee.

Pourtant, si nous regardons les résultats dans le papier de Khened, il n'obtient que 90% d'accuracy. En faisant des recherches, nous avons réalisé que le papier dont faisait référence [8] était finalement un article plus récent de Khened [9]. Dans ce papier, il explique le modèle et les features utilisés pour obtenir un tel score. Nous allons donc maintenant détailler les features utilisées ainsi que les implémenter.

Toutes les features proposées dérivent directement des caractéristiques utilisées par les médecins pour diagnostiquer les patients : par exemple, comme expliqué dans [8], un individu sain a une fraction d'éjection supérieure à 50% ainsi qu'une épaisseur de la paroi myocardique plus petite que $12mm$ en diastole, ou encore un individu atteint de MINF ou DCM a une fraction d'éjection inférieure à 40%. Il est alors naturel de considérer les différentes features suivantes.

4.2 Primary features

Les features primaires sont calculées directement à partir des images segmentées aux deux différentes phases du cycle cardiaque (ED et ES). Pour rappel, les segmentations sont composées de trois éléments : la cavité du ventricule droit (RV), le myocarde (MYO) et la cavité du ventricule gauche (LV). Les features primaires utilisés sont donc :

1. Volume du LV aux phases ED et ES.
2. Volume du RV aux phases ED et ES.
3. Masse (resp. Volume) du MYO à la phase ED (resp. ES).
4. Épaisseur du MYO à chaque slice (MWT).

Pour chacune de ces features, il ne faut pas oublier, comme nous l'avons dit plus tôt, de prendre en compte la résolution des voxels de l'image que nous regardons. Autrement dit, chaque feature sera calculée en pixels/voxels, puis multipliée par la résolution des voxels associés : si on regarde un volume, on multiplie par le volume d'un voxel, si on regarde une distance sur une slice, on multiplie par la longueur d'un côté d'un voxel (pixel spacing $\neq$ épaisseur des slices).

Pour les trois premières features, nous y accédons très facilement grâce à notre implémentation avec *TorchIO*. En effet, les images segmentées sont chargées en tant que *LabelMap* et nous pouvons accéder directement à un dictionnaire avec le nombre de voxels dans chaque classe (RV, MYO, LV).

À noter que dans tous les papiers cités jusqu'ici, les auteurs calculent le volume des RV, LV ainsi que du MYO mais seulement à la phase ES pour ce dernier. En effet, pour la phase ED, ils parlent de la masse du MYO. Or, après avoir fait quelques recherches, nous avons trouvé que la masse du MYO se calcule directement en multipliant le volume du MYO par la densité du myocarde qui est une constante (≈ 1.05 g/mL). Ce n'est donc pas vraiment clair de pourquoi ils parlent de la masse car pour nos futurs modèles de machine learning, multiplier toute une colonne de features par une constante ne sert à rien, dès lors que nous allons normaliser les données. Notre explication est que c'est probablement une donnée que les médecins utilisent plutôt que le volume, d'où son utilisation pour garder une bonne interprétation.

La dernière feature, le MWT à chaque slice, n'est pas totalement évidente à calculer : Khened nous explique comment le calculer proprement dans [9]. Tout d'abord, il faut détecter les contours intérieur et extérieur du MYO. Pour cela, il suffit d'appliquer un filtre de Canny sur le masque du LV (pour le contour intérieur), puis sur le masque du MYO + LV (pour le contour extérieur). FIGURE 2 illustre ces étapes.

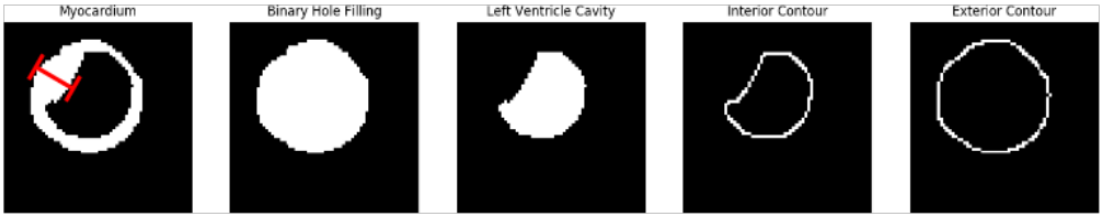

FIGURE 2 – Illustration de la procédure pour l'estimation du MWT [9].

Soit I et E les ensembles de pixels correspondant aux contours intérieur et extérieur obtenus. Alors MWT est l'ensemble des plus courtes distances euclidiennes mesurées à partir d'un pixel du contour intérieur I vers un pixel du contour extérieur E. Ici, nous précisons que c'est le MWT pour le SA (Short-Axis), qui correspond à la coupe du cœur où nous regardons nos slices :

$$MWT|SA = \{\min_{e \in E} d(i, e) : i \in I\}$$

On stocke la moyenne et l'écart-type de cet ensemble de distances pour chaque slice et pour chaque phase (ED et ES). Cela va nous permettre d'obtenir un nouvel ensemble de features dans la suite.

4.3 Derived features

Les features dérivées sont des combinaisons des features primaires. Nous les obtenons alors très simplement :

1. Fraction d'éjection du LV et RV.
2. Ratio de features primaires aux phases ED et ES.
3. Variations du MWT dans le Short-Axis (SA) et Long-Axis (LA).

La formule de la fraction d'éjection est donnée pour le LV et le RV par

$$EF = \frac{V_{ED} - V_{ES}}{V_{ED}} \times 100$$

où V_{ED} et V_{ES} correspondent aux volumes du LV ou RV aux phases ED et ES.

À titre indicatif, une fraction d'éjection normale du ventricule gauche est généralement supérieure à 50%. Des valeurs inférieures à 40% peuvent indiquer une insuffisance cardiaque [8]. Cette mesure est donc un critère très important dans la classification de certaines pathologies comme la DCM.

Pour les variations du MWT, nous regardons des statistiques dans le LA à partir des statistiques calculées ci-dessus dans le SA. FIGURE 3 illustre les différents axes.

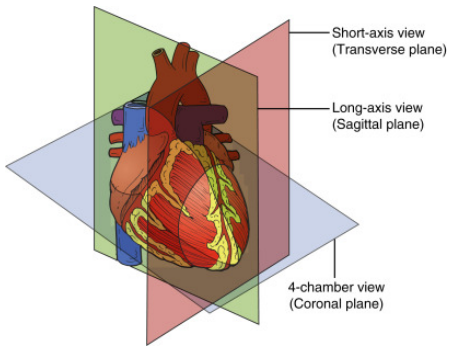

FIGURE 3 – Les différentes coupes du cœur.

Le récapitulatif des features utilisées se trouve dans la FIGURE 4. Nous obtenons donc un total de 20 features.

Features	**LV**	**RV**	**MYO**
Cardiac volumetric features			
volume at ED	✓	✓	
volume at ES	✓	✓	✓
mass at ED			✓
ejection fraction	✓	✓	
volume ratio: $ED[vol(LV)/vol(RV)]$	✓	✓	
volume ratio: $ES[vol(LV)/vol(RV)]$	✓	✓	
volume ratio: $ES[vol(MYO)/vol(LV)]$	✓		✓
mass to volume ratio: $ED[mass(MYO)/vol(LV)]$	✓		✓
Myocardial wall thickness variation profile			
$ED[max(mean(MWT\|SA)\|LA)]$			✓
$ED[stdev(mean(MWT\|SA)\|LA)]$			✓
$ED[mean(stdev(MWT\|SA)\|LA)]$			✓
$ED[stdev(stdev(MWT\|SA)\|LA)]$			✓
$ES[max(mean(MWT\|SA)\|LA)]^*$			✓
$ES[stdev(mean(MWT\|SA)\|LA)]^*$			✓
$ES[mean(stdev(MWT\|SA)\|LA)]^*$			✓
$ES[stdev(stdev(MWT\|SA)\|LA)]^*$			✓

FIGURE 4 – Tableau récapitulatif des features utilisées. $Statistic(MWT|SA)|LA$: Ensemble de statistiques (comme moyenne, écart-type) pour toutes les coupes selon le SA, vues selon le LA, à une phase cardiaque donnée [9].

5 Two-stage ensemble approach

À présent, il nous reste seulement à créer un modèle robuste afin de classifier nos données à partir des features extraites que nous avons pris soin de normaliser.

Pour cela, nous allons encore une fois nous baser sur la méthode utilisée dans [9]. En effet, dans la plupart des autres papiers, le modèle utilisé est Random Forest, qui est très puissant pour la classification à partir de données tabulaires, mais à chaque fois, la classification n'est pas parfaite. L'idée de ce papier est d'utiliser une méthode d'ensemble pour combiner plusieurs classifieurs.

La question est comment choisir quels classifieurs nous allons utiliser. Le papier répond lui-même à cette question : ils ont calculé l'accuracy de plusieurs modèles en utilisant une cross-validation, puis ils ont retenu les meilleurs. Les meilleurs modèles trouvés sont Support Vector Machine (SVM), Multi-layer Perceptron (MLP), Random Forest (RF) et Gaussian Naive Bayes (GNB). Pour vérifier que nous obtenions bien la même chose, nous avons utilisé la même méthodologie (accuracy avec la cross-validation) en rajoutant d'autres modèles, comme Linear et Quadratic Discriminant Analysis, K-Nearest Neighbors ou encore XGBoost. Et en effet, nous trouvons bien que les quatre modèles cités par le papier sont les plus performants, mais il y a également XGBoost qui obtient le même score que le MLP.

À noter que notre jeu de données est équilibré : nous avons 20 sujets par classe, c'est pourquoi nous pouvons utiliser la métrique *accuracy* sans aucun problème (nous utilisons toujours des *stratified* 5-fold cross-validation, ce qui conserve les proportions de classe).

Nous avons donc trouvé les hyperparamètres optimaux de chacun de ces classifieurs en utilisant *GridSearchCV* (pour SVM et GNB) et *RandomizedSearchCV* (pour RF et XGBoost car il y a trop de combinaisons possibles de paramètres pour GridSearch) de *scikit-learn* en utilisant une stratified 5-fold cross-validation. Pour le MLP, nous n'avons pas utilisé de recherche par grille : nous avons utilisé la bibliothèque *Optuna* qui permet de chercher de manière plus optimisée les hyperparamètres du modèle. Il est important de noter que nous avons trouvé tous les hyperparamètres optimaux de manière indépendante pour chaque classifieur avant de les combiner.

Les hyperparamètres optimaux notables pour les différents modèles sont : noyau RBF pour SVM, 1000 arbres pour RF, 2 hidden layers avec 100 neurones chacun pour MLP.

Ainsi, une fois les hyperparamètres optimaux trouvés, nous obtenons les accuracy suivantes pour chacun des modèles en utilisant une stratified 5-fold cross-validation :

- SVM : 0.96 ± 0.05
- RF : 0.95 ± 0.05
- GNB : 0.95 ± 0.06
- MLP : 0.94 ± 0.06
- XGBoost : 0.94 ± 0.06

À présent, nous pouvons combiner ces différents classifieurs dans un voting classifier qui prédit les maladies sur la base d'un vote majoritaire. Il y a 2 façons de faire ce vote : la méthode *hard*, qui consiste à littéralement choisir la classe la plus votée (prédite) par les classifieurs, et la méthode *soft*, qui consiste à prédire la classe sur la base de l'argmax des sommes des probabilités prédites. Bien que la méthode soft soit en théorie plus robuste (d'après la documentation même de *scikit-learn*), dans notre cas, la version hard a donné de meilleurs résultats empiriques : nous obtenons une accuracy (par cross-validation) de $\mathbf{0.98 \pm 0.04}$ pour la méthode hard, contre une accuracy de 0.97 ± 0.04 pour la méthode soft. Nous avons donc choisi d'utiliser la méthode hard. Nous avons donc réussi à réduire la variance de notre modèle tout en augmentant son accuracy et ainsi obtenu un modèle plus robuste et généralisable.

Pour voir plus en détail les performances du modèle, nous avons calculé puis stocké les matrices de confusion en les sommant lors de la 5-fold cross-validation (Figure 5).

Le modèle obtenu se montre particulièrement robuste, avec seulement deux erreurs sur l'ensemble des validations croisées. Cela suggère une bonne généralisation, même si la taille réduite du jeu de données impose de rester prudent vis-à-vis du surapprentissage.

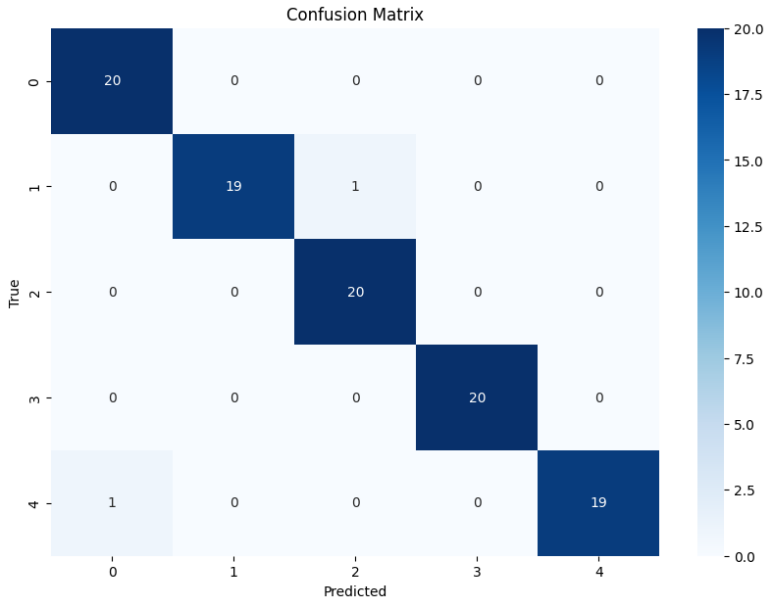

FIGURE 5 – Somme des matrices de confusion de la 5-fold cross-validation.

Comme notre Voting Classifier utilise un Random Forest, il est intéressant de regarder l'importance des différentes features (FIGURE 6). C'est d'autant plus intrigant car dans la plupart des papiers, c'est la fraction d'éjection du ventricule gauche qui est la feature la plus importante (même dans [9] !). Or ici, la fraction d'éjection du ventricule gauche n'arrive qu'en cinquième position, derrière des ratios de volumes ainsi que de simples volumes. Nous pouvons supposer que cette feature est corrélée avec une de nos autres features, ce qui divise son importance. En effet, en traçant la fraction d'éjection du LV en fonction du volume du LV à la phase ED (la feature juste derrière la fraction d'éjection du LV en importance), on obtient une droite parfaite, ce qui est finalement logique d'après la définition de la fraction d'éjection.

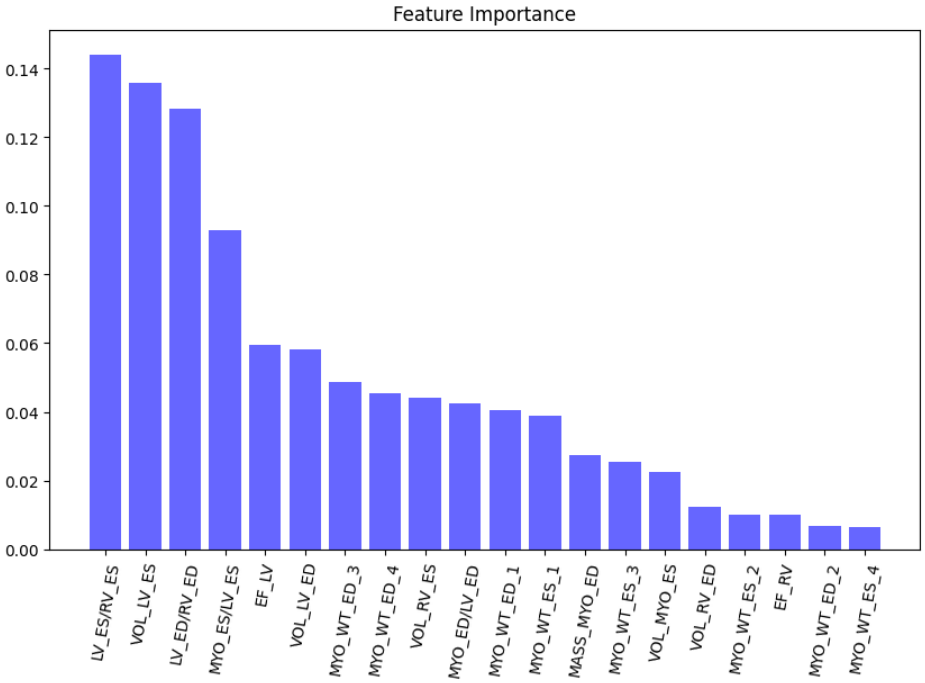

FIGURE 6 – Importance des features dans notre Random Forest.

Nous pouvons à présent, à partir des features les plus importantes, tracer les différentes surfaces de décision du Random Forest en regardant chaque couple possible de features. Nous avons ici décidé de le faire avec les quatre features les plus importantes (FIGURE 7). Nous remarquons que dans chaque cas, il n'y a pas de surface de décision pour la classe MINF (points noirs) : ces couples de features ne sont pas assez discriminants pour cette classe en particulier, mais nous remarquons que les autres classes sont plutôt bien séparées, et ce, bien que nous utilisions de simples projections en 2D ! Nous arrivons de plus à observer (sans parler des résultats de cross-validation) que le modèle n'est pas du tout overfit : les surfaces de décision sont homogènes avec des formes simples (quasiment rectangulaires), ce qui permet une généralisation de notre modèle.

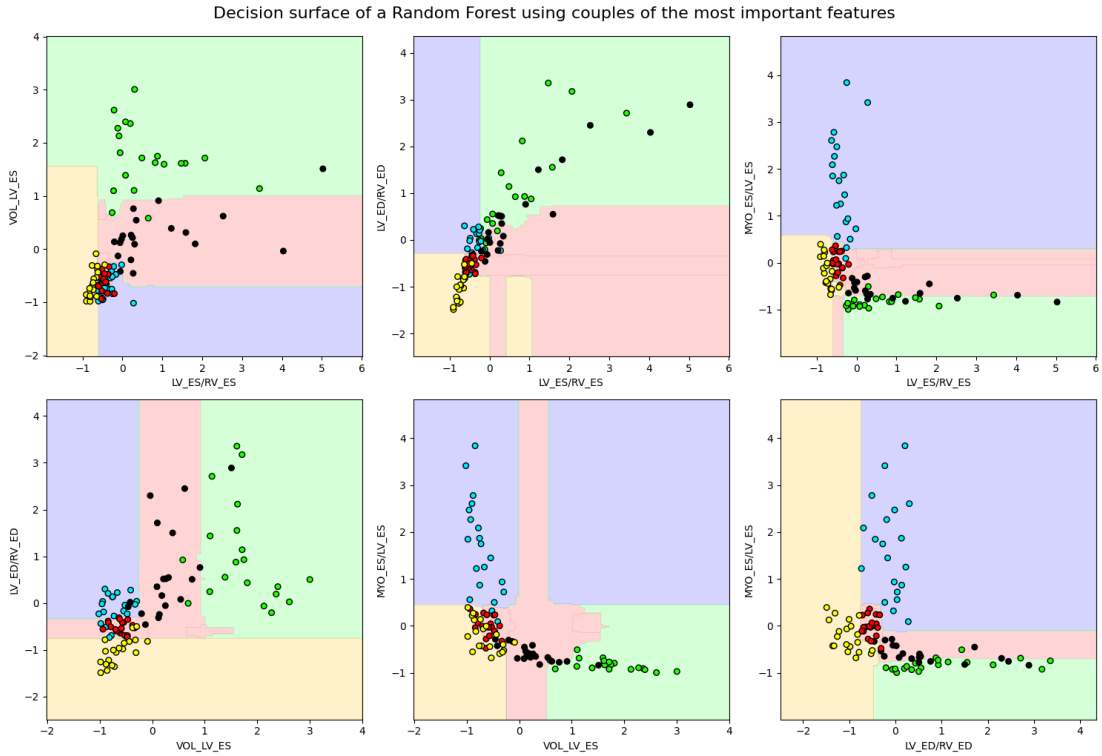

FIGURE 7 – Surfaces de décision de notre Random Forest en utilisant les couples de features les plus importantes (Rouge = NOR, Noir = MINF, Vert = DCM, Bleu = HCM, Jaune = ARV).

Nous remarquons cependant que, selon certains réglages d'hyperparamètres (notamment le mode de vote), le modèle éprouve des difficultés à distinguer les classes 1 (MINF) et 2 (DCM). Nous le remarquons également dans nos surfaces de décision que ces classes ne sont pas parfaitement distinctes (même s'il ne faut pas oublier que nous affichons seulement une projection avec des sous-ensembles de features).Cette confusion persistante entre ces deux classes a également été identifiée dans l'article de Khened [9].

Pour y remédier, il propose une architecture à deux étapes : un classifieur expert est entraîné spécifiquement sur les patients prédits comme MINF ou DCM. Ce modèle secondaire est un MLP spécialisé, reposant uniquement sur les features de variation du profil de l'épaisseur myocardique (MWT) à la phase ES, jugées les plus discriminantes pour cette sous-classification (features avec un * dans le tableau FIGURE 4). Comme nous utilisons à présent un sous-ensemble de seulement 4 features, nous pouvons afficher une projection 2D de chaque couple de features possible (FIGURE 8). Nous remarquons que dans chaque cas, les deux classes ne sont pas séparables simplement : il y a toujours des points bleus très proches des rouges, il y a donc potentiellement des points que nous ne pourrons jamais bien classifier, sous risque de faire de l'overfitting.

Un schéma récapitulatif de la pipeline globale se trouve dans la FIGURE 9. Nous avons exactement implémenté cela, à l'exception du Voting Classifier où nous y avons ajouté un classifieur XGBoost en plus des quatre autres.

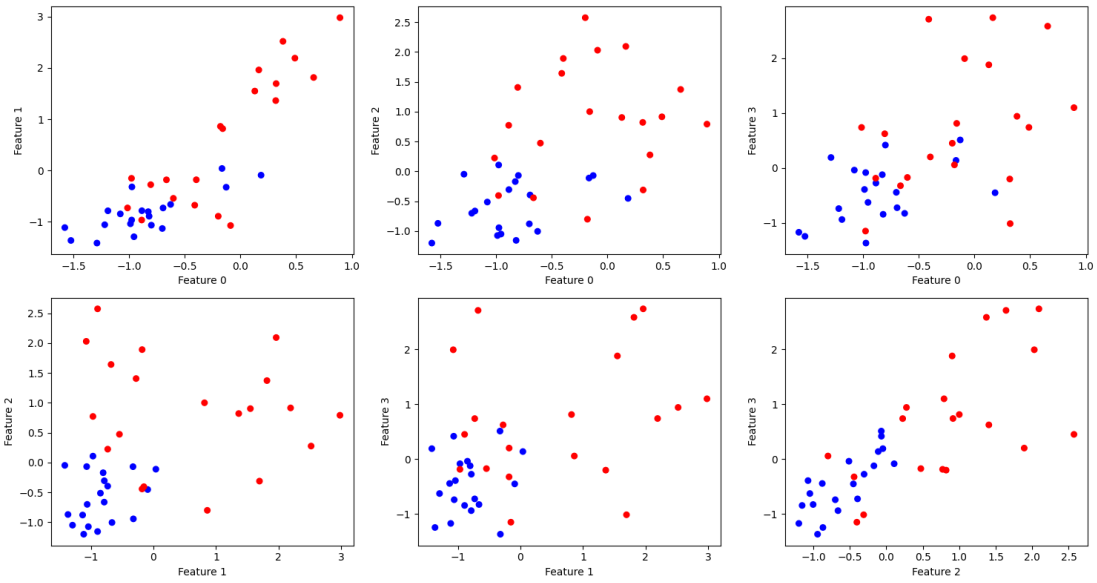

FIGURE 8 – Projection 2D des 4 features que nous utilisons pour la deuxième phase de classification (Rouge = MINF, Bleu = DCM).

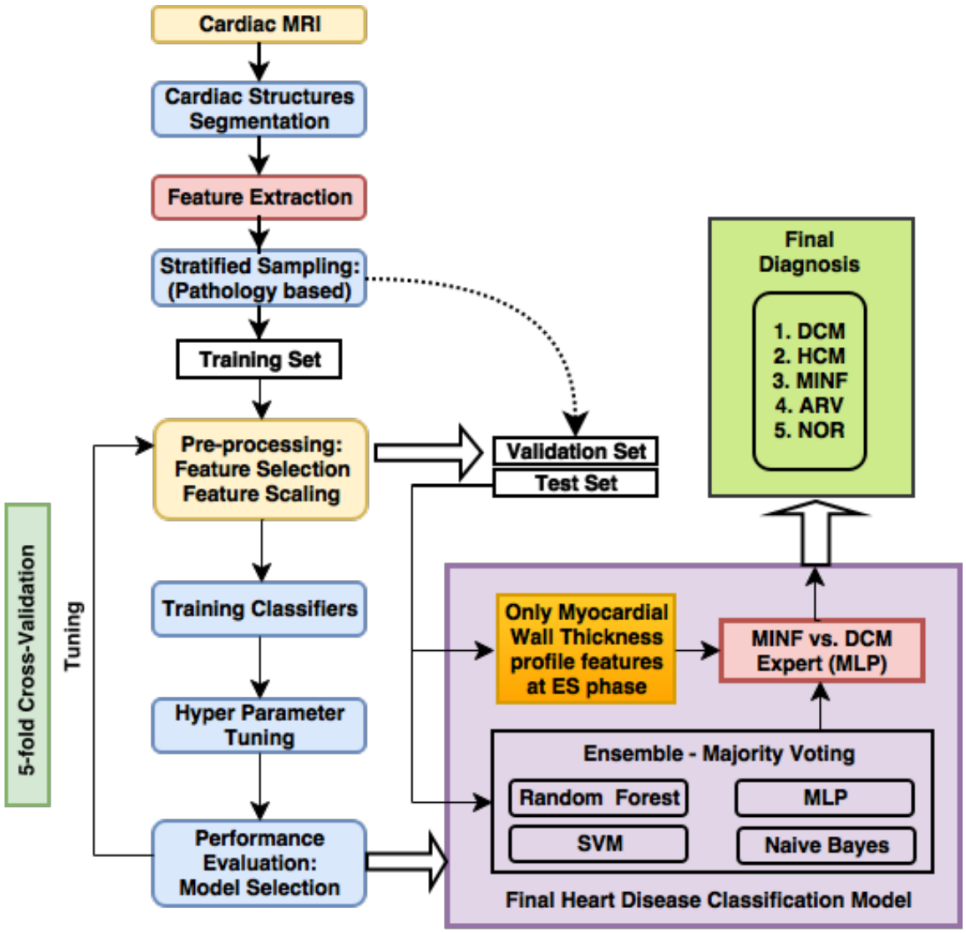

FIGURE 9 – Pipeline globale de la classification utilisant une approche avec un voting classifier et un classifieur expert [9].

Pour implémenter ce MLP expert, il y a plusieurs solutions. Tout d'abord, il y a la méthode assez haut niveau qui consiste à utiliser le modèle *MLPClassifier* de *scikit-learn* : c'est ce que nous avons utilisé dans la première phase de classification. L'avantage est qu'il est très simple à utiliser, tout comme les autres classifieurs de la librairie. Il permet notamment de le combiner sans souci dans le *VotingClassifier*, ce qui nous fait gagner en clarté dans le code. L'inconvénient est que ce MLP est en pratique assez peu modulable, et donc nous sommes rapidement restreints par son architecture basique.

L'autre solution est d'implémenter un MLP ex nihilo avec *Pytorch*. En effet, la deuxième phase est complètement indépendante des autres classifieurs, donc il n'y aura pas de problème de compatibilité (comme nous pouvions avoir avec le *VotingClassifier*). De cette manière, nous sommes clairement libres pour le choix de son architecture. Nous avons donc construit un modèle simple mais suffisamment expressif, comprenant plusieurs couches cachées (le nombre est encore à déterminer) entièrement connectées, chacune suivie d'une normalisation par batch, d'une activation LeakyReLU ainsi que d'un dropout.

À ce titre, nous avons laissé Optuna rechercher les hyperparamètres optimaux pour ce modèle : nombre de couches, taille des couches cachées, taux de dropout, taux d'apprentissage, etc. De manière similaire, nous avons également optimisé la version scikit-learn du MLP. Le meilleur modèle obtenu avec PyTorch comportait une seule couche cachée de 30 neurones, tandis que le MLP de scikit-learn comportait trois couches de 100 neurones.

Bien que les deux approches soient fondamentalement différentes, l'une étant construite à bas niveau avec PyTorch, l'autre utilisant une interface toute faite, elles parviennent toutes deux à des performances très compétitives. En cross-validation sur l'ensemble MINF/DCM, le MLP scikit-learn atteint une accuracy moyenne de 0.975 ± 0.05, tandis que celui implémenté en PyTorch atteint 0.95 ± 0.06.

Ce qui est encore plus remarquable, c'est que malgré leurs différences d'implémentation, d'optimisation et d'architecture, les deux modèles produisent exactement les mêmes prédictions sur les données de test. Ce résultat suggère que la séparation entre les deux pathologies est particulièrement nette dans l'espace de features utilisé. Cela vient conforter notre intuition initiale sur la pertinence des features morphologiques extraites à la phase ES.

Enfin, dans le cadre de la soumission finale au challenge Kaggle, nous ne disposons pas des labels du dataset de test, ce qui nous empêche d'évaluer précisément les performances de notre pipeline. Toutefois, nous pouvons formuler quelques hypothèses et observations. Nous supposons donc que le dataset de test est équilibré entre les cinq classes (comme le dataset d'entraînement) : nous nous attendons logiquement à obtenir un nombre de prédictions similaire pour chaque pathologie.

Or, sans la seconde phase de raffinement par le MLP expert, notre modèle avait tendance à prédire davantage certaines classes au détriment d'autres, notamment en confondant MINF et DCM. L'ajout du MLP expert a corrigé ce déséquilibre : le modèle prédit désormais exactement 10 patients dans chaque classe, ce qui est encourageant et laisse espérer une classification correcte.

Pour renforcer la robustesse de cette seconde phase, nous avons également entraîné une version bagging du MLP expert, composée d'un ensemble de 50 MLP entraînés sur des sous-échantillons du dataset d'entraînement et des sous-ensembles de features. Ce modèle utilise exactement les mêmes hyperparamètres optimisés qu'auparavant. Le bagging produit des prédictions quasiment identiques à la version simple (seulement deux différences), tout en conservant une parfaite répartition entre les cinq classes.

Étant donné que le challenge autorise deux soumissions finales, nous avons choisi de soumettre les prédictions issues du MLP expert implémenté en PyTorch (qui sont les mêmes qu'avec la version scikit-learn), ainsi que celles obtenues par la version bagging. Cela nous permet de maximiser nos chances d'obtenir un score élevé sur les données de test cachées.

6 Conclusion

Ce projet avait pour objectif de développer une pipeline de classification automatique de pathologies cardiaques à partir d'IRM, dans le cadre d'un challenge Kaggle. Nous avons proposé une approche basée sur l'extraction de features morphologiques et dynamiques, combinée à une stratégie de classification en deux phases.

Après avoir testé différents modèles (Random Forest, SVM, XGBoost, etc.), nous avons mis en place un système robuste combinant un ensemble de classifieurs via un VotingClassifier, suivi d'un MLP expert spécialisé pour distinguer les classes les plus confondues (MINF et DCM). Nous avons également expérimenté une version bagging de ce MLP expert, construite à partir d'un ensemble de 50 MLP entraînés de manière aléatoire, afin de renforcer la stabilité du modèle.

Durant la compétition, nous avions accès uniquement à un score partiel calculé sur 30% des données de test (soit 15 patients sur 50). Il était donc possible d'obtenir une accuracy parfaite sur ce sous-ensemble sans garantie de performance sur le reste des données. Pour cette raison, nous avons accordé une grande importance à la cross-validation, en cherchant à réduire la variance du modèle au maximum et à renforcer sa généralisation.

Ne disposant pas des labels du test set, nous avons formulé l'hypothèse que celui-ci était équilibré entre les cinq classes, comme dans les données d'entraînement. Or, nos deux modèles finaux, bien que différents, prédisaient chacun exactement 10 patients par classe, ce qui nous a confortés dans leur fiabilité.

À la publication des résultats du challenge, les deux soumissions ont toutes deux atteint une **accuracy de 0.96** ($1.0 \times 30\%$ (public) $+ 0.94 \times 70\%$ (private)), confirmant ainsi la robustesse de l'approche et la pertinence des features extraites. Ce score correspond exactement à celui obtenu par Khened dans son article de référence [9], que nous avons cherché à reproduire et à améliorer.

Un dernier point intéressant mérite d'être souligné concernant ces résultats. Nous avons en effet soumis deux prédictions distinctes, qui diffèrent uniquement dans le traitement de la deuxième phase de classification (celle permettant de départager les classes MINF et DCM) via deux stratégies différentes : un MLP expert et un bagging de MLP. Or, ces deux soumissions obtiennent exactement le même score final de 96% d'accuracy sur les données de test, tout en produisant des prédictions équilibrées avec 10 individus affectés à chaque classe. Cela implique nécessairement que les erreurs de classification se situent exclusivement entre les classes MINF et DCM, puisque les prédictions sont identiques pour les trois autres classes (NOR, HCM, ARV), et qu'il y a exactement 10 prédictions par classe dans chaque soumission. En d'autres termes, la classification de ces trois premières classes est parfaitement maîtrisée, avec une accuracy de 100%, et les deux erreurs restantes résultent d'une confusion entre MINF et DCM, bien que nous ayons utilisé un classifier expert pour limiter au maximum cette confusion. Ce constat est d'autant plus crédible que ces deux pathologies peuvent présenter des caractéristiques proches, rendant leur distinction particulièrement délicate même pour des experts [8].

Ainsi, ce score élevé, très proche de l'optimal, montre que la combinaison d'un bon travail de recherche de features, d'une architecture adaptée et d'une cross-validation rigoureuse permet d'obtenir d'excellentes performances même avec un dataset aussi limité.

Toute amélioration supplémentaire nécessiterait probablement l'accès à la séquence complète d'IRM du cycle cardiaque (et pas seulement aux phases ED et ES), qui permettrait d'extraire des features temporels plus riches, une possibilité hors de portée dans le cadre de ce challenge, mais qui pourrait ouvrir des perspectives intéressantes. Par exemple, ce papier plus récent proposé par Zheng [10] s'appuie sur la génération de séries temporelles de features à partir du calcul de flux apparents locaux (*local apparent flow*), features potentiellement plus discriminantes pour les classes MINF et DCM ...

Bibliographie

[1] Loic Le Folgoc et Pietro Gori. *IMA205 Challenge 2025*. https://kaggle.com/competitions/ima-205-challenge-2025. Kaggle. 2025.

[2] Moussa Saleh et John A Ambrose. "Understanding myocardial infarction". en. In : *F1000Res.* 7 (sept. 2018), p. 1378.

[3] Matthew R G Taylor, Elisa Carniel et Luisa Mestroni. "Cardiomyopathy, familial dilated". en. In : *Orphanet J. Rare Dis.* 1.1 (juill. 2006), p. 27.

[4] Martin S. Maron et al. "Hypertrophic Cardiomyopathy Is Predominantly a Disease of Left Ventricular Outflow Tract Obstruction". In : *Circulation* 114.21 (2006), p. 2232-2239.

[5] Mahendra Khened, Varghese Alex et Ganapathy Krishnamurthi. "Densely Connected Fully Convolutional Network for Short-Axis Cardiac Cine MR Image Segmentation and Heart Diagnosis Using Random Forest". In : *Statistical Atlases and Computational Models of the Heart. ACDC and MMWHS Challenges*. Sous la dir. de Mihaela Pop et al. Cham : Springer International Publishing, 2018, p. 140-151. isbn : 978-3-319-75541-0.

[6] Jelmer M. Wolterink et al. "Automatic Segmentation and Disease Classification Using Cardiac Cine MR Images". In : *Statistical Atlases and Computational Models of the Heart. ACDC and MMWHS Challenges*. Sous la dir. de Mihaela Pop et al. Cham : Springer International Publishing, 2018, p. 101-110. isbn : 978-3-319-75541-0.

[7] Fabian Isensee et al. "Automatic Cardiac Disease Assessment on cine-MRI via Time-Series Segmentation and Domain Specific Features". In : *Statistical Atlases and Computational Models of the Heart. ACDC and MMWHS Challenges*. Sous la dir. de Mihaela Pop et al. Cham : Springer International Publishing, 2018, p. 120-129. isbn : 978-3-319-75541-0.

[8] Olivier Bernard et al. "Deep Learning Techniques for Automatic MRI Cardiac Multi-Structures Segmentation and Diagnosis : Is the Problem Solved ?" In : *IEEE Transactions on Medical Imaging* 37.11 (nov. 2018), p. 2514-2525. issn : 1558-254X.

[9] Mahendra Khened, Varghese Alex Kollerathu et Ganapathy Krishnamurthi. *Fully Convolutional Multi-scale Residual DenseNets for Cardiac Segmentation and Automated Cardiac Diagnosis using Ensemble of Classifiers*. 2018.

[10] Qiao Zheng, Hervé Delingette et Nicholas Ayache. *Explainable cardiac pathology classification on cine MRI with motion characterization by semi-supervised learning of apparent flow*. 2019.

Lien du git : https://github.com/clementgilli/Cardiac-Pathology-Prediction