Hamza El-Mokadem, Maxence Jamme, Gabriel Martinez, Damien Pascual

Compte rendu de projet
Machine Learning

Professeur : Sébastien Harispe

Problématique :

Nous devons développer un modèle de deep learning capable de différencier les épisodes de fibrillation auriculaire (AF) à partir de différents électrocardiogrammes (ECG). L'objectif est de classifier les ECG non seulement entre les cas normaux (N) et les cas de fibrillation auriculaire (AF), mais également entre quatre classes distinctes : cas normaux (N), fibrillation auriculaire (AF), autres anomalies (O), et signaux bruyants (~).

Pour ce projet, nous disposons d'un jeu de données d'entraînement comprenant une grande quantité d'ECG labellisés selon ces quatre classifications (N, AF, O, ~). Le modèle sera construit et entraîné à partir de ce jeu de données.

Objectifs :

1. Analyser et formaliser la problématique.

2. Proposer différentes architectures pour résoudre le problème.

3. Évaluer et comparer les performances des différentes architectures proposées.

4. Identifier et proposer des approches pour améliorer les performances de l'architecture sélectionnée.

1. Analyse du problème

Dans ce problème, nous travaillons à partir de données d'ECG, disponibles dans notre dataset sous la forme de deux types de fichiers pour chaque enregistrement :

- **Fichier `.mat`** : Ce fichier contient les données brutes de l'ECG sous forme de tableau de données numériques représentant une série temporelle. Chaque valeur de ce tableau correspond à une mesure de la tension électrique (en mV) à un instant donné, permettant ainsi de capturer l'évolution du signal cardiaque au cours du temps. Ces tableaux ont des tailles variables.

- **Fichier `.hea`** : Ce fichier contient des métadonnées associées à l'enregistrement. Il fournit des informations générales telles que la date et l'heure de l'enregistrement, le fichier correspondant contenant les données ECG, ainsi que des détails techniques (ex : fréquence d'échantillonnage, amplitude du signal, etc.). Voici un exemple de ce que ces données peuvent contenir :

```
A00001 1 300 9000 05:05:15 1/05/2000

A00001.mat 16+24 1000/mV 16 0 -127 0 0 ECG
```

Ces fichiers sont associés aux données de sortie à prédire, qui correspondent à l'une des 4 classes suivantes :

- **A** : Fibrillation auriculaire.
- **N** : Signal normal.
- **O** : Autres anomalies.
- **~** : Signal bruité.

Les liens entre les labels et les fichiers du dataset sont présents dans un fichier CSV spécifique. les données de ce CSV ressemble à ceci :

1	A00001,N
2	A00002,N

Nous sommes donc confrontés à un problème de classification où l'entrée est une série temporelle (le tableau de données ECG) et la sortie est l'une des 4 classes (A, N, O, ou ~). Le défi consiste à entraîner un modèle capable de reconnaître et classer correctement chaque enregistrement ECG en fonction des caractéristiques du signal.

2. Proposition de différentes architecture de résolution du problème.

Pour résoudre ce problème de classification d'ECG, nous allons utiliser quatre architectures principales : MLP, CNN, RNN, et Transformers. Chacune de ces architectures apporte des avantages spécifiques en fonction des caractéristiques des données, et nous expliquerons en détail pourquoi elles sont pertinentes pour cette tâche.

a. MLP (Perceptron Multicouche)

Pourquoi l'utiliser :

Le MLP est une architecture simple qui peut servir de point de départ pour notre comparaison. Il est facile à implémenter et convient bien pour des données tabulaires ou vectorisées, comme les tableaux numériques des signaux ECG. Bien que le MLP ne soit pas spécifiquement conçu pour traiter des séries temporelles, il peut fournir une première évaluation des performances sur notre dataset.

b. CNN (Convolutional Neural Networks)

Pourquoi l'utiliser :

Le CNN est une architecture efficace pour capturer des motifs locaux dans les données. Dans le cas des ECG, certaines anomalies peuvent se manifester sous forme de motifs spécifiques dans le signal (par exemple, des pics ou des oscillations régulières). Le CNN peut aider à détecter ces schémas répétitifs.

c. RNN (Recurrent Neural Network)

Pourquoi l'utiliser :

Les RNN sont spécifiquement conçus pour traiter des séries temporelles et modéliser les dépendances entre des observations successives. Étant donné qu'un ECG est une série temporelle, un RNN est bien adapté pour capturer les informations dynamiques sur la durée.

d. Transformers

Pourquoi l'utiliser :

Les Transformers sont une architecture moderne qui excelle dans la capture de relations à longue portée dans des séquences. Grâce à leur mécanisme d'attention, ils peuvent se concentrer sur les parties pertinentes du signal, même lorsque ces informations sont dispersées dans le temps. Ce mécanisme pourrait être particulièrement utile pour les ECG, où certains segments du signal peuvent être cruciaux pour déterminer la présence d'une fibrillation auriculaire (AF) ou d'autres anomalies.

e. Conclusion :

En utilisant ces 4 architectures, nous allons pouvoir tester différentes approches pour résoudre notre problème. Chacune d'elle présente des avantages et des inconvénient et leur comparaison va nous permettre de mieux comprendre quelle approche est la plus adaptée.

Nous allons implémenter nos différents modèles en Python. L'ensemble des fichiers d'implémentation sera disponible sur Git, organisé par architecture.

3. Implémentation des modèles

a. Dataset

i. méthode de resizing

Avant de commencer à implémenter, nous devons réfléchir à la manière d'utiliser notre dataset de départ. Les fichiers mat sont facilement convertibles en tableau grâce à des lib python spécifiques mais ce n'est pas le seul point critique dans la bonne utilisation de notre dataset, en effet la tailles des données dans nos ECG sont très variable (entre 2714 et 18286) il faut donc une méthode pour standardiser nos tailles d'enregistrement pour pouvoir les utiliser dans nos modèles. Certains modèles nécessitent des tailles de données d'entrées identiques.

Il existe plusieurs méthodes pour faire ceci, nous avons opté au début par une méthode de 0 padding qui est une méthode assez simple et facilement implémentable. Cette méthode consiste à remplir le tableau de valeur 0 jusqu'à sa taille maximale.

Le désavantage de cette méthode est qu'on va traiter uniquement des enregistrements très longs (avec des fois beaucoup de 0), pour pallier cela on peut ajouter une limite en taille pour les enregistrements. On perd de la donnée mais on gagne en rapidité. Cette valeur limite est à réfléchir et à tester (la médiane est la valeur souvent utilisée).

D'autres méthodes existent qui permette de ne pas enlever des données sans ajouter des 0. C'est le cas de l'interpolation (ou redimensionnement linéaire), elle consiste à redimensionner chaque enregistrement ECG à une longueur fixe, quelle que soit sa longueur originale. Cela préserve l'ensemble du signal tout en ajustant sa résolution temporelle.

Nous allons tester toutes ces différentes approches dans un premier temps sur le MLP pour voir les différences et quelle stratégie nous allons pouvoir établir par la suite.

Nous avons construit une classe commune appelée datasetResize qui gère toutes les méthodes nommé ci-dessus de redimensionnement de dataset. Cela nous permettra de gagner en rapidité sur le test des différentes dimensions.

ii. Choix du train_dataset et test_dataset

Ensuite il faut qu'on définisse correctement une taille pour notre jeu d'entraînement et notre jeu de test, car ce choix peut avoir des conséquences sur la performance de nos modèles.

La méthode standard nous dit de diviser notre dataset en 80/20, c'est à dire 80% du dataset pour l'entraînement et 20% pour le test. Cela permet d'avoir une bonne balance entre bon entrainement et test performant.

Dans notre cas, nous n'avons pas un dataset de très grande ampleur (8528 ECG) avec une répartitions assez inégales des label :

N : **5076** enregistrement soit **59.52%**

A : **758** enregistrement soit **8.89%**

O : **2415** enregistrement soit **28.32%**

~ : **279** enregistrement soit **3.27%**

Il serait intéressant de faire une stratification des label pour ne pas se retrouver avec des écart de proportions encore plus importantes et très différentes entre les jeux d'entrainement et ceux de test. Une stratification signifie qu'on essaie de garder la même proportion des différents labels dans les 2 datasets.

iii. Normalisation des données

En dernier et fin de dernière journée, une des approches que nous envisageons est la normalisation des données, qui pourrait potentiellement améliorer les performances de nos modèles en rendant les enregistrements ECG plus cohérents. L'idée est que, en réduisant l'impact des variations d'amplitude entre les signaux, nous pourrions permettre aux algorithmes d'apprentissage d'apprendre plus efficacement. Cependant, nous restons conscients que cette méthode pourrait ne pas produire les résultats escomptés et pourrait même nuire à la performance du modèle dans certains cas.

b. MLP

Emplacement du notebook : **MLP.ipynb**

Architecture du modèle :

Dans un premier temps, nous avons choisi de faire nos premiers test avec le modèle suivant:

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 128)	2,340,736
dense_4 (Dense)	(None, 32)	4,128
dense_5 (Dense)	(None, 4)	132

Résultats pour la comparaison des tailles d'enregistrements :

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5339 - loss: 23.3233
Précision sur l'ensemble de test : 0.5281
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           N       0.60      0.84      0.70      1015
           A       0.11      0.04      0.06       152
           O       0.26      0.06      0.10       483
           ~       0.08      0.16      0.10        56

    accuracy                           0.53      1706
   macro avg       0.26      0.28      0.24      1706
weighted avg       0.45      0.53      0.45      1706
```

52,81% d'accuracy

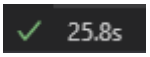
MLP avec dataset en 0 padding (taille max : 18286)

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5435 - loss: 4.9660
Précision sur l'ensemble de test : 0.5422
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           N       0.60      0.90      0.72      1015
           A       0.08      0.01      0.02       152
           O       0.24      0.01      0.02       483
           ~       0.04      0.09      0.05        56

    accuracy                           0.54      1706
   macro avg       0.24      0.25      0.20      1706
weighted avg       0.43      0.54      0.44      1706
```

54,22% d'accuracy



MLP avec dataset en 0 padding et truncate (taille médiane : 9000)

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5824 - loss: 2.0256
Précision sur l'ensemble de test : 0.5856
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           N       0.60      0.98      0.74      1015
           A       0.00      0.00      0.00       152
           O       0.12      0.00      0.01       483
           ~       0.00      0.00      0.00        56

    accuracy                           0.59      1706
   macro avg       0.18      0.25      0.19      1706
weighted avg       0.39      0.59      0.44      1706
```

58,56% d'accuracy

MLP avec dataset en interpolation (taille médiane : 9000)

Au vue de nos résultats nous pouvons en déduire que la taille des enregistrements a bien une incidence sur les résultats de nos classification.

Résumé des résultats :

- Zéro padding (taille max : 18 286) :
  - accuracy = **52.81%**
  - temps = **1m5s**
- Zéro adding + truncate (taille médiane : 9000) :
  - accuracy = **54.22%**
  - temps = **25s**
- Interpolation (taille médiane : 9000) :
  - accuracy = **58.56%**
  - temps = **26s**

On peut déduire que l'interpolation est bien plus performante sur notre modèle. Cette méthode a le meilleur équilibre entre perte de données et taille des enregistrements par rapport aux 2 autres. Nous allons pouvoir continuer à améliorer notre MLP en mettant à jour ces hyper paramètre en utilisant l'interpolation pour les enregistrements.

En plus de la performance, le temps est bien moindre dans les cas où l'on a une taille médiane plutôt que la taille maximale. Pour le MLP on peut encore tester tous les cas mais pour les prochains modèles qui seront plus long a entraîné on se tournera plus vers les méthode a taille médiane par gain de temps.

Test avec la normalisation des données:

Nous avons effectué les mêmes étapes que précédemment citées mais cette fois en normalisant les données.

Résultats :

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5934 - loss: 2.2461
Précision sur l'ensemble de test : 0.5920
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
              precision    recall  f1-score   support

           N       0.59      0.99      0.74      1015
           A       0.33      0.01      0.01       152
           O       0.40      0.00      0.01       483
           ~       0.00      0.00      0.00        56

    accuracy                           0.59      1706
   macro avg       0.33      0.25      0.19      1706
weighted avg       0.50      0.59      0.45      1706
```

59,20% d'accuracy

MLP avec dataset en 0 padding (taille max : 18286)

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4793 - loss: 3.2187
Précision sur l'ensemble de test : 0.4818
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
              precision    recall  f1-score   support

           N       0.61      0.68      0.64      1015
           A       0.09      0.08      0.09       152
           O       0.32      0.24      0.28       483
           ~       0.03      0.04      0.03        56

    accuracy                           0.48      1706
   macro avg       0.26      0.26      0.26      1706
weighted avg       0.46      0.48      0.47      1706
```

48,18% d'accuracy

MLP avec dataset en 0 padding et truncate (taille médiane : 9000)

Pour ce résultat, il semblerait que l'on est fait de l'overfitting. En effet, en 20 epoch nous sommes arrivés à 98% d'accuracy sur le jeu d'entraînements. Malheureusement, nous n'avons pas le temps de continuer davantage les recherches en baissant le nombre d'epoch notamment.

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4546 - loss: 3.6498
Précision sur l'ensemble de test : 0.4678
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
              precision    recall  f1-score   support

           N       0.62      0.64      0.63      1015
           A       0.10      0.12      0.11       152
           O       0.31      0.26      0.28       483
           ~       0.08      0.12      0.10        56

    accuracy                           0.47      1706
   macro avg       0.28      0.29      0.28      1706
weighted avg       0.47      0.47      0.47      1706
```

46,78% d'accuracy

MLP avec dataset en interpolation (taille médiane : 9000)

De même pour ce résultat, une tendance d'overfitting peut être analysée. Nous sommes également à 98% d'accuracy sur le jeu d'entrainements.

En conclusion, à ce stade, il est difficile de déterminer si la normalisation est une option pertinente. Il serait judicieux de poursuivre les tests en veillant à entraîner suffisamment notre modèle sans tomber dans le surapprentissage (overfitting) afin de parvenir à une conclusion plus définitive.

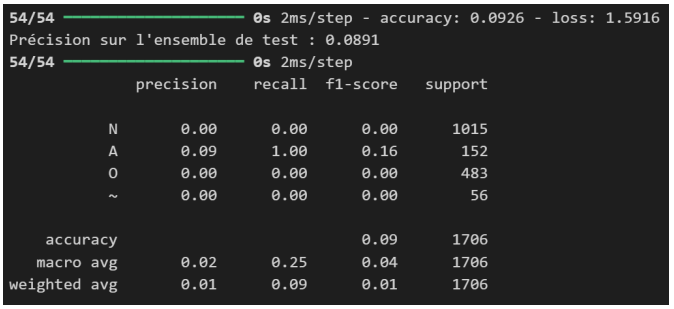
Modification des hyperparamètres:

Pour continuer les tests sur le MLP, nous avons choisi d'exclure la lecture du dataset en 0 padding (taille max : 18286), à cause de son accuracy et de son temps de traitement.

Ensuite, nous avons choisi de garder le dataset en 0 padding et truncate (taille médiane : 9000) et de changer les hyperparamètres pour voir comment se comporte notre MLP. Pour cela, nous avons choisi de faire 4 test:

1. **Truncate Test 1 :** On effectue un test avec un grand nombre d'epoch

   On passe de 20 epoch des précédents tests à 300, avec le même modèle



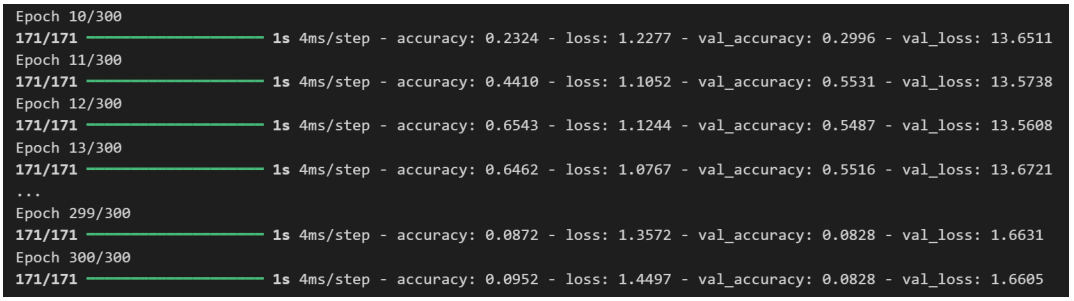
```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0926 - loss: 1.5916
Précision sur l'ensemble de test : 0.0891
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           N       0.00      0.00      0.00      1015
           A       0.09      1.00      0.16       152
           O       0.00      0.00      0.00       483
           ~       0.00      0.00      0.00        56

    accuracy                           0.09      1706
   macro avg       0.02      0.25      0.04      1706
weighted avg       0.01      0.09      0.01      1706
```

   Ici, nous avons une accuracy de **8,91%**

   Dans un premier temps, nous avons pensé avoir atteint un surapprentissage (overfitting) sur l'ensemble d'entraînement. Cependant, après avoir analysé les résultats, nous avons constaté que le réseau était également devenu inefficace sur cet ensemble.



```
Epoch 10/300
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2324 - loss: 1.2277 - val_accuracy: 0.2996 - val_loss: 13.6511
Epoch 11/300
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4410 - loss: 1.1052 - val_accuracy: 0.5531 - val_loss: 13.5738
Epoch 12/300
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6543 - loss: 1.1244 - val_accuracy: 0.5487 - val_loss: 13.5608
Epoch 13/300
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6462 - loss: 1.0767 - val_accuracy: 0.5516 - val_loss: 13.6721
...
Epoch 299/300
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.0872 - loss: 1.3572 - val_accuracy: 0.0828 - val_loss: 1.6631
Epoch 300/300
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.0952 - loss: 1.4497 - val_accuracy: 0.0828 - val_loss: 1.6605
```

   Cela nous a amenés à comprendre qu'il ne s'agissait pas d'un problème de surapprentissage, mais plutôt d'un comportement très inhabituel du modèle. Pour résoudre ce problème, nous devrions envisager de diminuer le taux d'apprentissage.

2. **<u>Truncate Test 2 :</u>** On effectue un test avec un nombre d'epoch égal à 50

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5913 - loss: 2.9774
Précision sur l'ensemble de test : 0.5897
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           N       0.60      0.99      0.74      1015
           A       0.25      0.01      0.01       152
           O       0.14      0.00      0.00       483
           ~       0.00      0.00      0.00        56

    accuracy                           0.59      1706
   macro avg       0.25      0.25      0.19      1706
weighted avg       0.42      0.59      0.45      1706
```

Lors du précédent test, avec 20 epoch, nous avions une accuracy de **54,22%**

Sur ce test, avec 50 epoch, nous avons un accuracy de **58,97%**

3. **<u>Truncate Test 3 :</u>** On effectue un test avec deux couches cachée

Voici les couches du modèle:

Layer (type)	Output Shape	Param #
dense_88 (Dense)	(None, 128)	1,152,128
dense_89 (Dense)	(None, 64)	8,256
dense_90 (Dense)	(None, 32)	2,080
dense_91 (Dense)	(None, 4)	132

On effectue un test avec 20 epoch:

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4433 - loss: 4.5311
Précision sur l'ensemble de test : 0.4490
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           N       0.60      0.67      0.63      1015
           A       0.07      0.14      0.10       152
           O       0.31      0.12      0.18       483
           ~       0.03      0.04      0.03        56

    accuracy                           0.45      1706
   macro avg       0.25      0.24      0.23      1706
weighted avg       0.45      0.45      0.44      1706
```

Avec deux couches cachées, nous sommes à **44,9%** d'accuracy.

On refais un test en augmentant le nombre d'epoch de 20 à 50:

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5824 - loss: 1.9865
Précision sur l'ensemble de test : 0.5809
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           N       0.60      0.97      0.74      1015
           A       0.06      0.01      0.02       152
           O       0.33      0.00      0.00       483
           ~       0.06      0.02      0.03        56

    accuracy                           0.58      1706
   macro avg       0.26      0.25      0.20      1706
weighted avg       0.46      0.58      0.44      1706
```

Avec ces nouvelles valeurs, nous sommes à **58,09%** d'accuracy.

4. **<u>Truncate Test 4 :</u>** On effectue un test en changeant le nombre de neurones par couche

   Voici les couches du modèle:

Layer (type)	Output Shape	Param #
dense_99 (Dense)	(None, 256)	2,304,256
dense_100 (Dense)	(None, 32)	8,224
dense_101 (Dense)	(None, 4)	132

   On effectue un test avec 20 epoch:

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5403 - loss: 10.0775
Précision sur l'ensemble de test : 0.5457
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           N       0.59      0.89      0.71      1015
           A       0.12      0.04      0.06       152
           O       0.23      0.03      0.05       483
           ~       0.07      0.07      0.07        56

    accuracy                           0.55      1706
   macro avg       0.25      0.26      0.22      1706
weighted avg       0.43      0.55      0.45      1706
```

   Avec ce changement du nombre de neurones, nous sommes à **54,03%** d'accuracy.

   On refais un test en augmentant le nombre d'epoch de 20 à 50:

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5902 - loss: 2.9884
Précision sur l'ensemble de test : 0.5897
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           N       0.60      0.99      0.74      1015
           A       0.00      0.00      0.00       152
           O       0.50      0.00      0.00       483
           ~       0.00      0.00      0.00        56

    accuracy                           0.59      1706
   macro avg       0.27      0.25      0.19      1706
weighted avg       0.50      0.59      0.44      1706
```

   Avec ces nouvelles valeurs, nous sommes à **58,97%** d'accuracy.

Ensuite, nous utilisons les modèles qui ont données les meilleurs résultats avec le dataset en interpolation (taille médiane : 9000) :

1. **<u>Interpolated Test 1 :</u>** On effectue un test avec un nombre d'epoch égal à 50

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.0893 - loss: 2.1355
Précision sur l'ensemble de test : 0.0879
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
              precision    recall  f1-score   support

           N       0.00      0.00      0.00      1015
           A       0.09      0.98      0.16       152
           O       0.07      0.00      0.00       483
           ~       0.00      0.00      0.00        56

    accuracy                           0.09      1706
   macro avg       0.04      0.25      0.04      1706
weighted avg       0.03      0.09      0.02      1706
```

   Lors du précédent test, avec 20 epoch, nous avions une accuracy de **58,56%**

   Sur ce test, avec 50 epoch, nous avons un accuracy de **8,79%**

   Cette diminution du pourcentage d'accuracy peut être due à deux choses, soit notre pas est trop grand pour la descente du gradient soit nous sommes en overfitting. Nous n'avons pas eu le temps de changer le pas pour savoir si cela permet d'avoir une meilleur accuracy, nous sommes donc parti du principe que nous sommes en overfitting dans ce cas.

2. **<u>Interpolated Test 2 :</u>** On effectue un test avec deux couches cachée

   Voici les couches du modèle:

Layer (type)	Output Shape	Param #
dense_88 (Dense)	(None, 128)	1,152,128
dense_89 (Dense)	(None, 64)	8,256
dense_90 (Dense)	(None, 32)	2,080
dense_91 (Dense)	(None, 4)	132

   On effectue un test avec 30 epoch (pour éviter l'overfitting):

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5870 - loss: 2.0162
Précision sur l'ensemble de test : 0.5862
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           N       0.60      0.98      0.74      1015
           A       0.00      0.00      0.00       152
           O       0.50      0.01      0.01       483
           ~       0.06      0.04      0.04        56

    accuracy                           0.59      1706
   macro avg       0.29      0.26      0.20      1706
weighted avg       0.50      0.59      0.45      1706
```

Avec deux couches cachées, nous sommes à **58,7%** d'accuracy.

3. **<u>Interpolated Test 3 :</u>** On effectue un test en changeant le nombre de neurones par couche

   Voici les couches du modèle:

Layer (type)	Output Shape	Param #
dense_99 (Dense)	(None, 256)	2,304,256
dense_100 (Dense)	(None, 32)	8,224
dense_101 (Dense)	(None, 4)	132

   On effectue un test avec 20 epoch (pour éviter l'overfitting):

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5609 - loss: 4.1180
Précision sur l'ensemble de test : 0.5645
54/54 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
              precision    recall  f1-score   support

           N       0.59      0.94      0.73      1015
           A       0.05      0.01      0.02       152
           O       0.23      0.01      0.01       483
           ~       0.02      0.02      0.02        56

    accuracy                           0.56      1706
   macro avg       0.23      0.25      0.20      1706
weighted avg       0.42      0.56      0.44      1706
```

   Avec ce changement du nombre de neurones, nous sommes à **56,09%** d'accuracy.

Par manques de temps, nous n'avons pas pu faire d'autre changement sur les hyperparamètres, mais voici quelques piste que nous aurions souhaité réaliser:

- Modification du taux d'apprentissage
- Essayer différentes fonction d'activation comme sigmoid ou tanh
- Modification de l'optimiser, remplacer adam que nous avons par SGD ou RMSprop
- Modifier la taille des batchs
- Essayer de faire du dropout etc…

c. CNN

Emplacement du notebook : **CNN.ipynb**

Architecture :

Layer	Input	Output	Kernel Size
Conv 1 (conv1d)	1	16	7
Conv 2 (conv1d)	16	32	5
Conv 3 (conv1d)	32	64	3
Linear 1	64*(taille ECG // 8)	128	
Linear 2	128	4	

Nous avons commencé par une architecture de CNN assez classique d'une forme similaire à celle qu'on avait vu dans le cours.

Le premier objectif est de retester sur un nombre d'époques restreintes (10) quel impact vont avoir nos différentes stratégie de dimensionnement des signaux.

Résultats pour la comparaison des tailles d'enregistrements (cpu) :

```
Accuracy globale: 50.00%
Accuracy pour la classe N: 67.19%
Accuracy pour la classe AF: 3.95%
Accuracy pour la classe O: 33.33%
Accuracy pour la classe ~: 7.14%
```
✓ 11m 18.8s

CNN avec dataset en 0 padding (taille max : 18286)

```
Accuracy globale: 51.11%
Accuracy pour la classe N: 66.80%
Accuracy pour la classe AF: 0.66%
Accuracy pour la classe O: 37.47%
Accuracy pour la classe ~: 21.43%
```
✓ 5m 19.8s

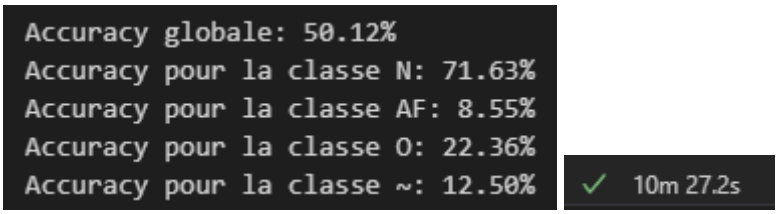
CNN avec dataset en 0 padding et truncate (taille médiane : 9000)

```
Accuracy globale: 49.59%
Accuracy pour la classe N: 69.06%
Accuracy pour la classe AF: 5.26%
Accuracy pour la classe O: 26.50%
Accuracy pour la classe ~: 16.07%
```
✓ 5m 19.9s

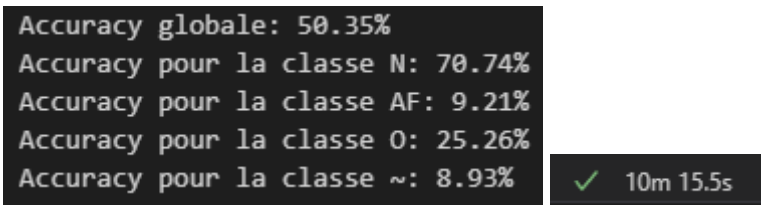
CNN avec dataset en interpolation (taille médiane : 9000)

On peut voir qu'avec un modèle encore plutôt simple et un nombre d'époque plus restreinte que le modèle avec le 0 padding en taille maximale n'apporte rien de plus que les 2 autres modèles, on peut l'abandonner dû à son temps 2x supérieur aux autres méthodes.Nous allons essayer d'augmenter le nombre d'époque pour voir si les performance augmente encore. On peut aussi voir que la méthode en padding et truncate permet de mieux repérer les O et ~, tandis que l'interpolation repérer mieux les AF.

Résultats pour 20 époque (cpu) :



CNN avec dataset en 0 padding et truncate (taille médiane : 9000)



CNN avec dataset en interpolation (taille médiane : 9000)

On voit ici qu'en augmentant les époques on n'augmente pas forcément l'accuracy globale mais on n'augmente nos taux de détection d'AF (en compensation on perd en détection de signaux bruité). Ce qui est une bonne chose dans l'objectif de détection de la maladie.

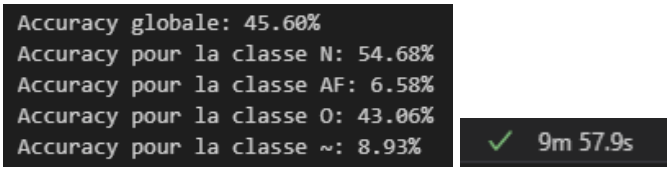
Piste d'améliorations du modèles :

Nous disposons de plusieurs pistes pour améliorer les performances de notre modèle. Tout d'abord, sur le plan de l'architecture, il pourrait être intéressant d'ajouter davantage de couches convolutionnelles pour permettre au modèle de capturer des caractéristiques plus complexes des signaux ECG. Nous pourrions également implémenter des couches de Dropout, qui aideraient à réduire le surapprentissage en désactivant aléatoirement une fraction des neurones pendant l'entraînement. Une autre approche consisterait à modifier notre stratégie de **pooling** en testant des méthodes telles que le Global Average Pooling, afin de mieux résumer les informations extraites par les convolutions. En outre, augmenter le nombre de couches fully connected pourrait permettre au modèle d'apprendre des

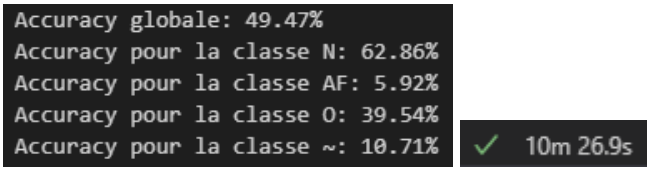
relations non linéaires plus complexes. Il serait aussi possible d'expérimenter avec différentes fonctions d'activation, comme LeakyReLU ou ELU, pour voir si elles améliorent la convergence et les performances. Enfin, nous pourrions explorer des variations d'hyperparamètres comme le taux d'apprentissage, la taille des batches, et l'utilisation de schedulers pour optimiser le processus d'entraînement. Ces différentes pistes offrent un large éventail de possibilités pour renforcer les capacités de notre modèle.

Par manque de temps nous n'allons pas mettre en place toutes ses différentes stratégies même si elles peuvent être intéressantes.

Changement de la fonction pooling (AvgPool1d) :



CNN avec dataset en 0 padding et truncate (taille médiane : 9000) - 20 époques

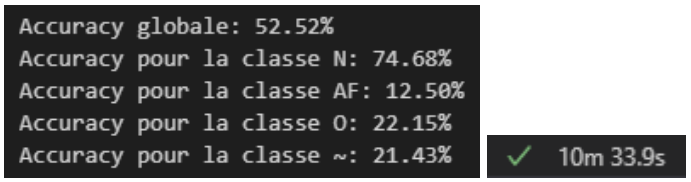


CNN avec dataset en interpolation (taille médiane : 9000) - 20 époques

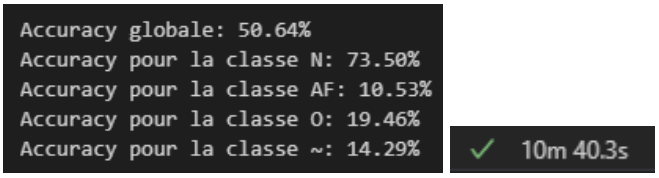
Au vue des résultats cette améliorations n'a pas vraiment d'influence positives sur nos résultats.Cela peut être dû au fait que les caractéristique qui définissent si un signal appartient à tel ou tel catégorie sont des caractéristique plus saillante de l'ECG (comme des variations soudaines ou encore des motifs fort).

Pour l'explication, L'AvgPooling a tendance à lisser les données tandis que le MaxPooling à tendance à faire ressortir les extrêmes.

Ajout d'une couche fully connected :



CNN avec dataset en 0 padding et truncate (taille médiane : 9000) - 20 époques

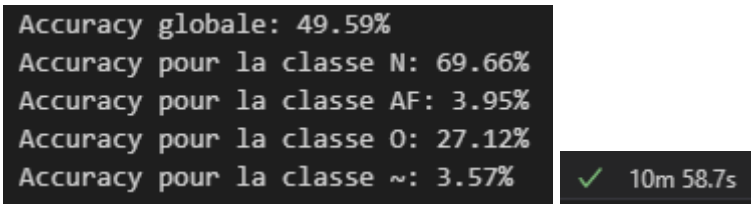


CNN avec dataset en interpolation (taille médiane : 9000) - 20 époques

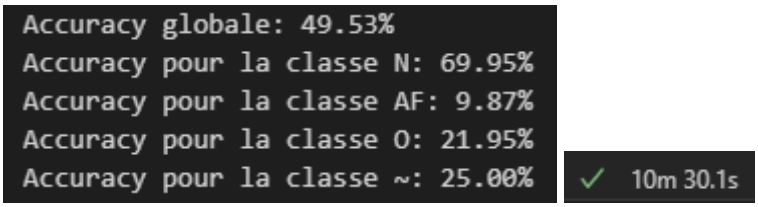
De manière assez logique l'ajout d'une couche fully connected augmente notre temps de traitement ainsi que nos performances (assez légèrement) Cela peut être une piste d'améliorations pour la suite.

Essayons d'ajouter des couches convolutionnelles à notre modèle pour voir si nous gagnons aussi en précisions.

Ajout d'une couche convolutionnelles :



CNN avec dataset en 0 padding et truncate (taille médiane : 9000) - 20 époques



CNN avec dataset en interpolation (taille médiane : 9000) - 20 époques

Vis à vis des résultats obtenus sur l'ajout d'une nouvelle couche de convolution on ne peut pas déterminer avec certitude que ça améliore nos résultats, en réalité nos résultats se dégradent légèrement.

Cela peut avoir plusieur cause, overfitting ou encore modèle trop complexe

d. RNN

Pour cette partie, nous avons rapidement implémenter un RNN, mais nous n’avons pas eu le temps de l’optimiser.

Le modèle est constitué de plusieurs couches, dont une couche d'entrée qui reçoit les signaux ECG prétraités sous forme de séquences temporelles.
Nous avons utilisé une couche RNN de type SimpleRNN pour extraire les caractéristiques temporelles des signaux, suivie d'une couche de dropout pour réduire le surapprentissage.
La sortie du modèle est une couche dense avec une activation softmax, permettant de classifier les enregistrements ECG en quatre catégories : Normal (N), Fibrillation Auriculaire (A), Autre (O) et Bruit (~).

Pour l'entraînement, nous avons utilisé la fonction de perte de type categorical_crossentropy, adaptée à des problèmes de classification multiclasse, et l'optimiseur Adam pour ajuster les poids du réseau. Nous avons donc entraîné le modèle sur 5 epoch par manque de temps.

```
Epoch 1/5
214/214 ━━━━━━━━━━━━━━━━━━━━ 123s 573ms/step - accuracy: 0.4606 - loss: 3.4123
Epoch 2/5
214/214 ━━━━━━━━━━━━━━━━━━━━ 93s 435ms/step - accuracy: 0.5295 - loss: 1.6454
Epoch 3/5
214/214 ━━━━━━━━━━━━━━━━━━━━ 101s 471ms/step - accuracy: 0.5495 - loss: 1.4070
Epoch 4/5
214/214 ━━━━━━━━━━━━━━━━━━━━ 105s 489ms/step - accuracy: 0.5541 - loss: 1.3254
Epoch 5/5
214/214 ━━━━━━━━━━━━━━━━━━━━ 108s 504ms/step - accuracy: 0.5701 - loss: 1.1720
```

Ce modèle a été évalué sur un ensemble de tests, et nous avons généré un rapport de classification pour analyser ses performances en termes de précision, de rappel et de score F1 pour chaque classe.

```
54/54 ━━━━━━━━━━━━━━━━━━━━ 8s 143ms/step
              precision    recall  f1-score   support

           N       0.00      0.00      0.00       140
           A       0.61      0.99      0.76      1044
           O       0.39      0.02      0.04       473
           ~       0.00      0.00      0.00        49

    accuracy                           0.61      1706
   macro avg       0.25      0.25      0.20      1706
weighted avg       0.48      0.61      0.47      1706
```

e. Resnet et Transformers

i. Architecture proposé

Notre modèle combine ResNet pour l'extraction de caractéristiques locales à partir des signaux ECG avec Transformers pour exploiter les relations globales au sein de ces signaux. Voici la structure globale du modèle :

Bloc ResNet : Les premiers blocs du modèle sont des convolutions résiduelles, où chaque bloc résiduel capture les motifs locaux dans les signaux ECG. Ces motifs peuvent correspondre à des caractéristiques physiologiques spécifiques du signal, comme les ondes P, les complexes QRS, ou les ondes T.

Bloc Transformer : Une fois que les caractéristiques locales ont été extraites par ResNet, elles sont ensuite traitées par un bloc Transformer. Ce bloc applique des mécanismes d'attention pour identifier des relations complexes entre les différentes parties des signaux, permettant au modèle de capturer des dépendances à long terme et des informations pertinentes sur l'état du cœur.

Couche de classification : La sortie du bloc Transformer est réduite par un mécanisme de pooling global et passe à travers une couche entièrement connectée avec une activation softmax pour classer les signaux dans les différentes catégories (A, N, O, ~).

ii. Pourquoi combiner ResNet et Transformers ?

- **Extraction locale (ResNet) et relation globale (Transformer)** : Les signaux ECG comportent des caractéristiques locales importantes (par exemple, des pics dans le signal) que les convolutions résiduelles de ResNet peuvent extraire efficacement. Cependant, les relations entre les différentes parties du signal, à travers le temps, sont tout aussi importantes. Les **Transformers**, grâce à leur mécanisme d'attention, sont bien adaptés pour capturer ces relations globales.
- **Résilience aux données bruyées** : Les signaux ECG sont souvent bruyés, et les Transformers, en combinant plusieurs "têtes" d'attention, permettent au modèle de mieux se concentrer sur les parties les plus pertinentes du signal, réduisant ainsi l'impact du bruit.
- **Modularité** : L'architecture est modulaire, avec une séparation claire entre l'extraction des caractéristiques locales par ResNet et l'analyse des relations globales par les Transformers. Cela permet une plus grande flexibilité lors de l'entraînement et de l'ajustement des hyperparamètres.

iii. méthodologie d'évaluation

a Validation croisée avec K-fold

Nous avons utilisé une validation croisée à **K plis (K-fold)** dans la methode `train_model_with_cross_validation()` pour évaluer la performance du modèle. Cette méthode permet de diviser le dataset en plusieurs sous-ensembles et d'entraîner le modèle sur des partitions différentes à chaque itération, garantissant une évaluation plus robuste.

b Métriques d'évaluation

Les **métriques d'évaluation** utilisées comprennent l'**accuracy**, mais aussi le **F1-score**, qui est essentiel dans le cas des données déséquilibrées, comme les arythmies rares. Le **F1-score** prend en compte à la fois la précision et le rappel, ce qui est particulièrement pertinent pour évaluer la capacité du modèle à détecter des événements rares dans les signaux ECG.

iv. Améliorations apportées au modèle

a Augmentation des données

Nous avons utilisé des techniques d'augmentation des données, comme l'ajout de bruit gaussien aux segments ECG, pour rendre le modèle plus robuste aux

variations et aux données bruitées. Cela a permis d'améliorer la capacité de généralisation du modèle lors des tests sur des données jamais vues.

b Utilisation de scheduler d'apprentissage

Un scheduler de taux d'apprentissage a été intégré au processus d'entraînement. Cela permet de réduire dynamiquement le taux d'apprentissage lorsque la performance sur le set de validation stagne, favorisant ainsi une convergence plus rapide et plus stable du modèle.

v. méthodologie d'évaluation

Performances :

Le modèle combiné ResNet + Transformer a atteint un **loss accuracy** de **67.07 %**, avec des performances particulièrement (acceptable) sur les classes majoritaires (N), mais également des améliorations notables sur les classes minoritaires grâce à l'attention multi-tête des Transformers.

vi. piste d'amélioration

Le modèle combinant **ResNet** et **Transformers** s'est révélé efficace pour la problématique. L'approche hybride permet de capturer à la fois des caractéristiques locales grâce à ResNet et des relations temporelles globales grâce aux Transformers.

Perspectives :

- **Exploration de modèles plus profonds** : Envisager des versions plus profondes du modèle Transformer pour capter encore plus de relations complexes entre les segments ECG.
- **Techniques d'augmentation avancées** : Explorer des techniques d'augmentation plus sophistiquées, comme les réseaux génératifs adverses (GANs), pour générer des données synthétiques et améliorer encore la robustesse du modèle.

4. Croisement des résultats

Le modèle qui a donné les meilleurs résultats est un modèle hybride combinant un CNN (ResNet) et un Transformer. Comme expliqué précédemment, ce modèle bénéficie de la capacité à combiner les avantages des deux architectures, permettant de capturer un maximum de caractéristiques de notre signal.

Il n'est pas surprenant que ce modèle ait surpassé les autres, car il est non seulement le plus complexe (bien que la complexité n'implique pas toujours de meilleures performances, dans notre cas, elle était justifiée et bien maîtrisée), mais aussi celui qui a bénéficié du plus de temps d'entraînement et de corrections/améliorations.

5. Perspectives d'amélioration générale

Un des principaux défis de ce projet a été le manque de temps. Avec seulement deux jours à disposition, il était difficile de mener à bien l'entraînement de modèles complexes nécessitant de nombreuses époques. Nous avons donc privilégié l'analyse et la réflexion sur la diversité des stratégies possibles plutôt que l'implémentation et le perfectionnement de modèles spécifiques. Cela explique la taille de notre rapport par rapport à la précision maximale obtenue.

Les ressources matérielles ont également été une contrainte importante. Bien que nous disposions d'un conteneur avec un GPU, il était difficile de progresser rapidement en partageant cette ressource entre les quatre membres de l'équipe. Une augmentation des ressources temporelles et matérielles aurait probablement permis de mettre en place des modèles plus complexes et de les entraîner plus en profondeur.

Par ailleurs, la modélisation du dataset et des données pourrait encore être optimisée. Un approfondissement sur l'impact de la normalisation serait particulièrement pertinent, notamment en la combinant avec l'ajout d'un vecteur décrivant les paramètres influencés par cette normalisation, tels que les amplitudes maximales et minimales des signaux avant normalisation, ou encore la moyenne et l'écart-type utilisés pour centrer les données. Cela permettrait de conserver des informations sur les caractéristiques originales des signaux, enrichissant ainsi les données fournies aux modèles.

Enfin, il serait judicieux de mettre en place une stratégie d'augmentation des données. Notre dataset se compose de moins de 8528 enregistrements, dont seulement 758 cas de fibrillation auriculaire (AF) et 279 cas de bruit. Cette distribution déséquilibrée révèle un manque de données pour l'entraînement, notamment pour les cas AF et de bruit. Avec un dataset aussi limité, il est difficile de développer un modèle véritablement performant. L'application de techniques d'augmentation des données pourrait ainsi être une voie intéressante à explorer pour enrichir le dataset et améliorer la robustesse des modèles.