

Rapport Projet IC - Antonetti Levis - BOUDJENIBA Oussama

1. Résumé du sujet

La première étape de ce projet est de reproduire les résultats du papier *Noise Reduction in ECG Signals Using Fully Convolutional Denoising Autoencoders*. Nous avons récupéré le code du projet réalisé en TensorFlow v1. et nous l'avons reproduit sur un notebook avec pytorch.

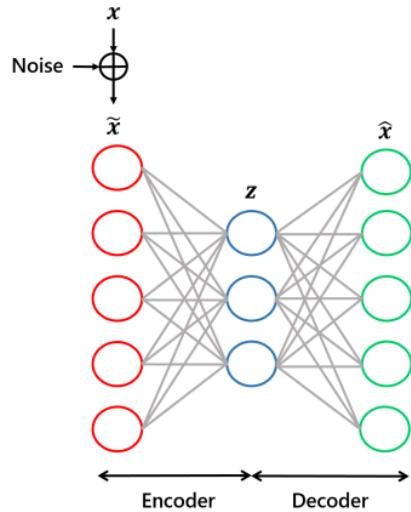


FIGURE 1. Schematic diagram of DAE.

Cet article propose une architecture auto-encoder (DAE) entièrement convolutionnelle (FCN) dans le but de débruitre du signal provenant d'ECG (électrocardiogrammes). Les signaux ECG ont été bruité avec différents niveaux de SNR.

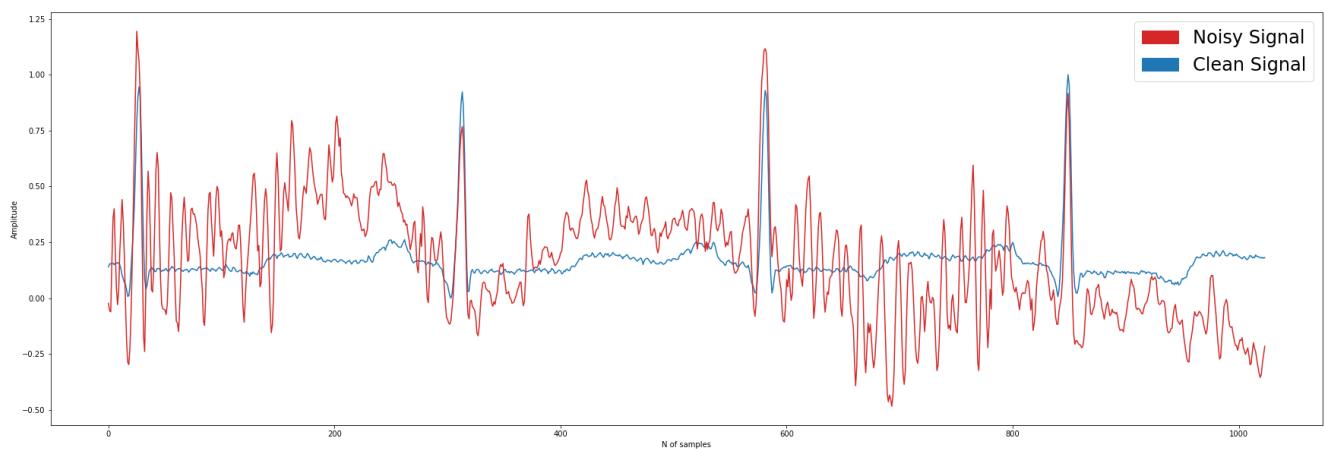


fig1. échantillon signal ECG bruité et non bruité

En ce référant à l'article, les performances de débruitage sur cette tâche ont été comparé à des architectures de réseaux de neurones plus classiques, CNN et DNN, et montre de meilleurs résultats.

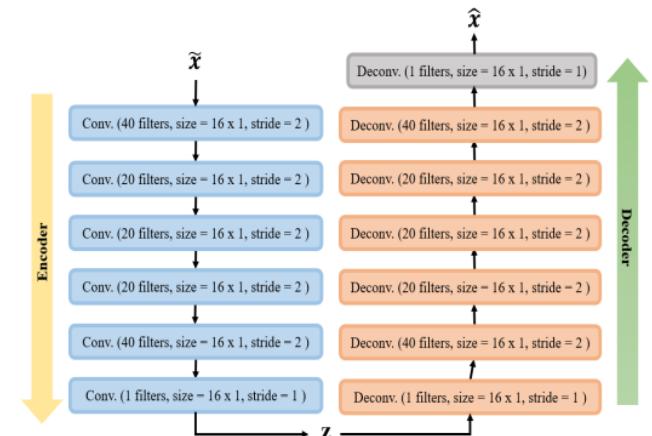


FIGURE 2. Architecture of the proposed FCN-based DAE.

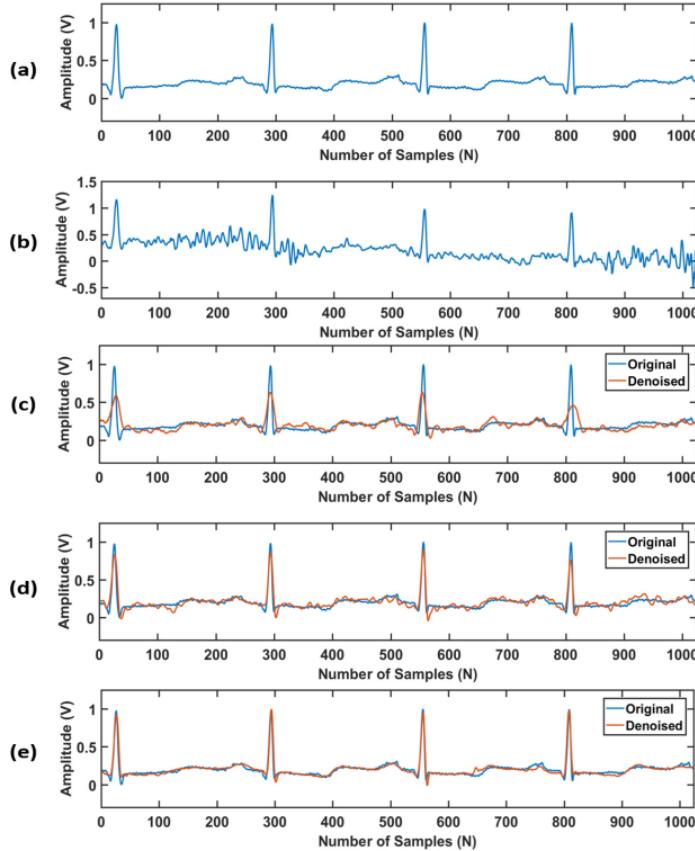


FIGURE 4. Comparison of denoised results of all evaluated methods on record 100 (a) original ECG signal (b) noisy ECG signal with an input SNR of 3dB (c) DNN (d) CNN (e) FCN.

Pour l'évaluation du modèle les métriques *RMSE* et *PRD* ont été utilisé afin d'évaluer les performances du débruitage des signaux de sortie du modèle avec les signaux originaux et la métrique *SNR_imp* qui indicate la différence de SNR après le débruitage entre les signaux de sortie et les signaux originaux.

root mean square error:

$$RMSE = \sqrt{\frac{1}{N} \times \sum_{n=1}^N (x_i - \hat{x}_i)^2}$$

percentage root mean square difference:

$$PRD = \sqrt{\frac{\sum_{n=1}^N (x_i - \hat{x}_i)^2}{\sum_{n=1}^N x_i^2}} \times 100$$

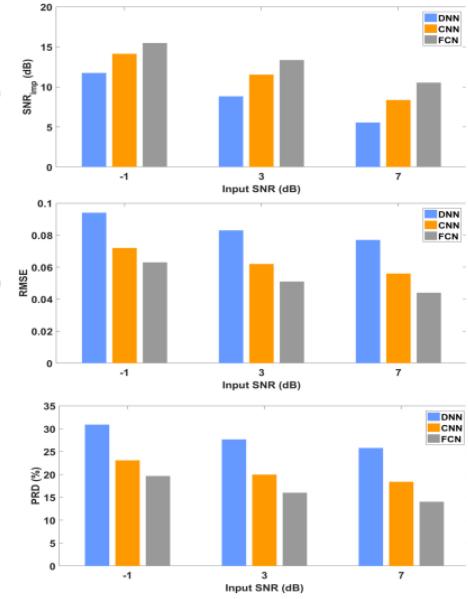


FIGURE 3. Comparison of the denoising performance of all evaluated methods at different input SNR levels (a) SNR_{imp} of DNN, CNN and FCN with varying input SNR levels (b) RMSE of DNN, CNN and FCN with varying input SNR levels (c) PRD of DNN, CNN and FCN with varying input SNR levels.

SNR difference between orginal et denoised input:

$$SNR_{in} = 10 \times \log_{10} \left(\frac{\sum_{n=1}^N x_i^2}{\sum_{n=1}^N (\tilde{x}_i - x_i)^2} \right)$$

$$SNR_{out} = 10 \times \log_{10} \left(\frac{\sum_{n=1}^N x_i^2}{\sum_{n=1}^N (\hat{x}_i - x_i)^2} \right)$$

$$SNR_{imp} = SNR_{out} - SNR_{in}$$

2. Reproduction et comparaison aux résultats du papier

Dans un premier temps nous avons adapté et réalisé l'architecture DAE-FCN ainsi que tout le code auxiliaire afin d'importer les données, de les traiter et d'entrainer le modèle en ce référant à l'article.

```

class DAE(nn.Module):
    def __init__(self, input_size=1024, kernel_size=16):
        super(DAE, self).__init__()
        self.activation = nn.ELU()
        self.kernel_size = kernel_size
        self.input_size = input_size
        self.padding = (kernel_size - 1) // 2
        slot = self.input_size
        self.conv1 = nn.Conv1d(1, 40, self.kernel_size, stride=2, padding=self.padding)
        self.conv1.bn1 = nn.BatchNorm1d(40)
        self.conv2 = nn.Conv1d(40, 20, self.kernel_size, stride=2, padding=self.padding)
        self.conv2.bn2 = nn.BatchNorm1d(20)
        self.conv3 = nn.Conv1d(20, 20, self.kernel_size, stride=2, padding=self.padding)
        self.conv3.bn3 = nn.BatchNorm1d(20)
        self.conv4 = nn.Conv1d(20, 20, self.kernel_size, stride=2, padding=self.padding)
        self.conv4.bn4 = nn.BatchNorm1d(20)
        self.conv5 = nn.Conv1d(20, 40, self.kernel_size, stride=2, padding=self.padding)
        self.conv5.bn5 = nn.BatchNorm1d(40)
        self.conv6 = nn.Conv1d(40, 1, self.kernel_size, stride=1, padding='same')
        self.deconv0 = nn.ConvTranspose1d(1, 1, self.kernel_size - 1, stride=1, padding=self.padding)
        self.deconv0.bn0 = nn.BatchNorm1d(1)
        self.deconv1 = nn.ConvTranspose1d(40, 20, self.kernel_size, stride=2, padding=self.padding)
        self.deconv1.bn1 = nn.BatchNorm1d(20)
        self.deconv2 = nn.ConvTranspose1d(20, 40, self.kernel_size, stride=2, padding=self.padding)
        self.deconv2.bn2 = nn.BatchNorm1d(40)
        self.deconv3 = nn.ConvTranspose1d(40, 20, self.kernel_size, stride=2, padding=self.padding)
        self.deconv3.bn3 = nn.BatchNorm1d(20)
        self.deconv4 = nn.ConvTranspose1d(20, 20, self.kernel_size, stride=2, padding=self.padding)
        self.deconv4.bn4 = nn.BatchNorm1d(20)
        self.deconv5 = nn.ConvTranspose1d(20, 40, self.kernel_size, stride=2, padding=self.padding)
        self.deconv5.bn5 = nn.BatchNorm1d(40)
        self.deconv6 = nn.ConvTranspose1d(
            40,
            1,
            self.kernel_size - 1,
            stride=1,
            padding=(self.kernel_size - 1) // 2
        )

```

```

def forward(self, x):
    # encoder
    x = self.conv1(x)
    x = self.conv.bn1(x)
    x = self.activation(x)

    x = self.conv2(x)
    x = self.conv.bn2(x)
    x = self.activation(x)

    x = self.conv3(x)
    x = self.conv.bn3(x)
    x = self.activation(x)

    x = self.conv4(x)
    x = self.conv.bn4(x)
    x = self.activation(x)

    x = self.conv5(x)
    x = self.conv.bn5(x)
    x = self.activation(x)

    x = self.conv6(x)
    x = self.activation(x)

    # decoder
    x = self.deconv0(x)
    x = self.deconv.bn0(x)
    x = self.activation(x)

    x = self.deconv1(x)
    x = self.deconv.bn1(x)
    x = self.activation(x)

    x = self.deconv2(x)
    x = self.deconv.bn2(x)
    x = self.activation(x)

    x = self.deconv3(x)
    x = self.deconv.bn3(x)
    x = self.activation(x)

    x = self.deconv4(x)
    x = self.deconv.bn4(x)
    x = self.activation(x)

    x = self.deconv5(x)
    x = self.deconv.bn5(x)
    x = self.activation(x)

    x = self.deconv6(x)
    x = self.activation(x)

    return x

```

fig2. Notre Architecture du modèle DAE-FCN

Entrainement avec les données propres/bruitées disponibles dans le répertoire

Pour l'entraînement, le jeu de données a été divisé en 80% de données d'entraînement, 10% de données de test et 10% pour la validation.

Concernant les résultats de l'entraînement du modèle, nous avons dans un premier temps entraîné notre modèle sur les données expérimentales d'ECG disponible dans le répertoire du projet et nous avons utilisé les mêmes métriques d'évaluation afin de les comparer aux résultats de reconstruction des signaux du modèle original du papier.

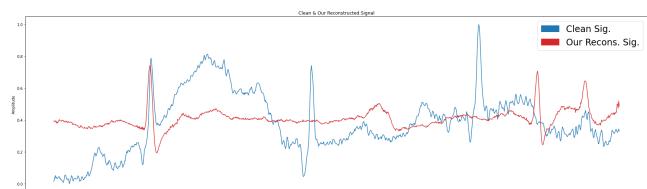
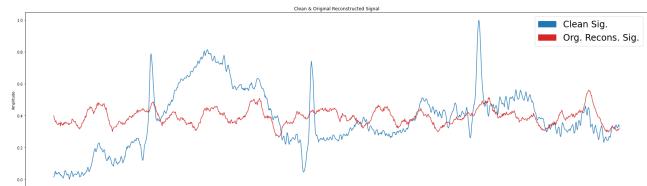
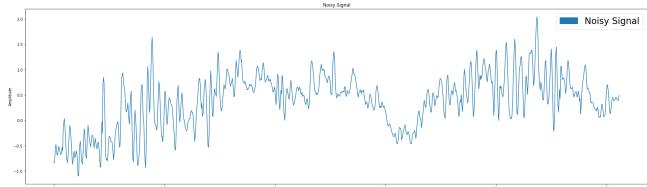
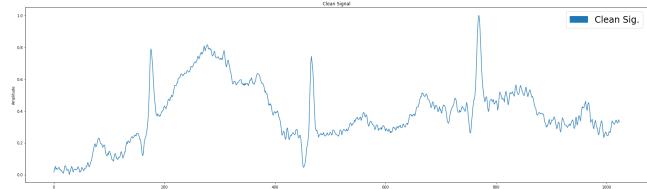
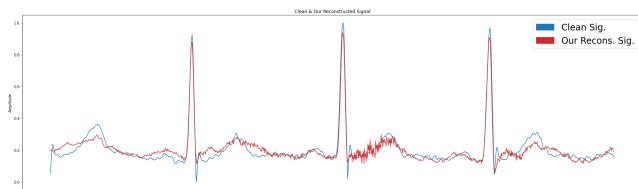
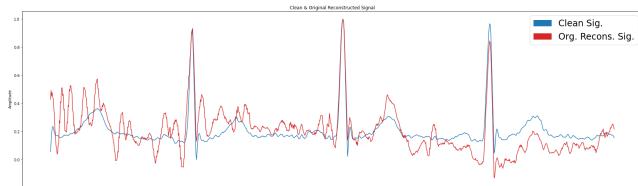
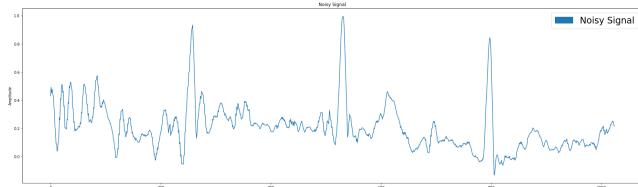
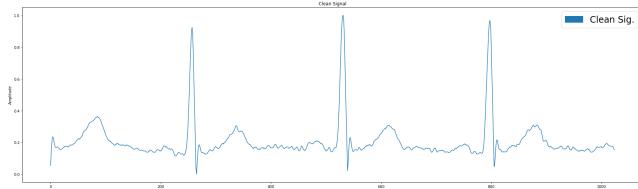


fig3. échantillon signal ECG propre, bruité et débruité (meilleur cas)

fig4. échantillon signal ECG propre, bruité et débruité (pire cas)

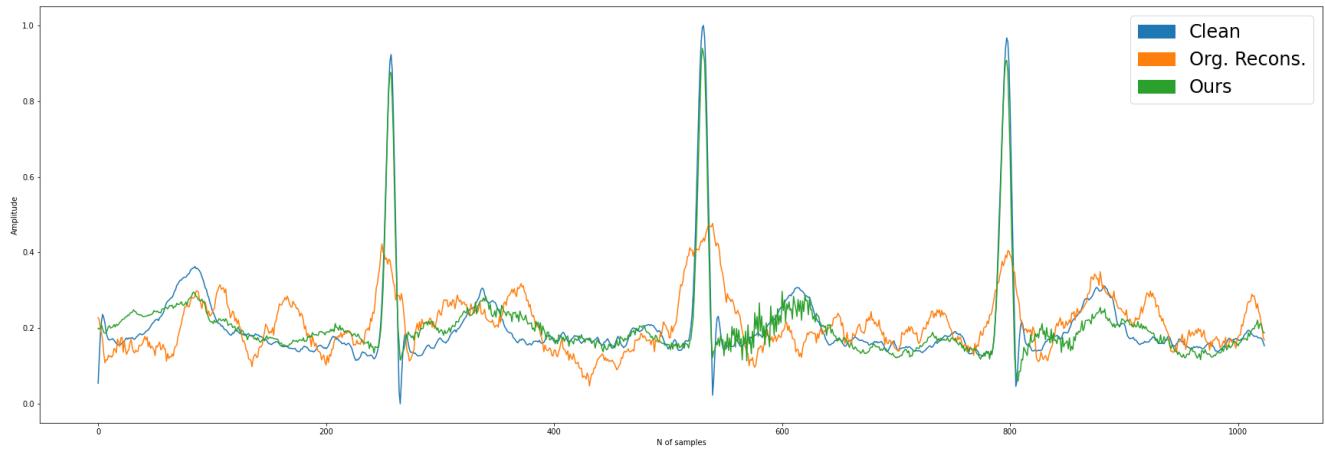


fig5. échantillon signal ECG propre, bruité et débruité superposés

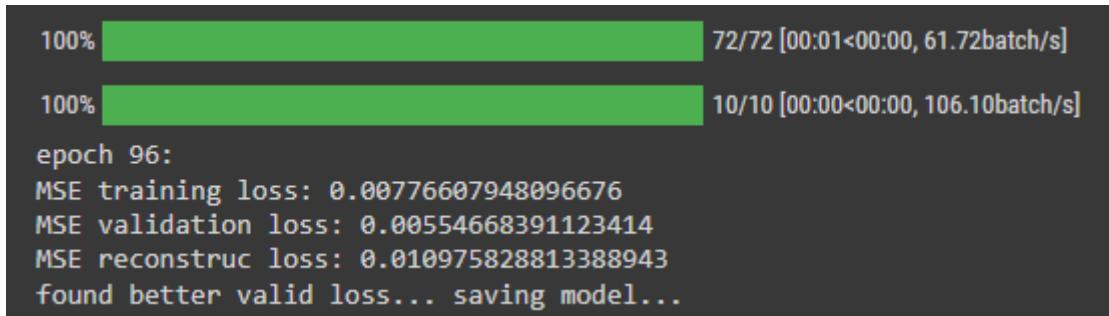


fig6. best performances model 1

Entrainement avec notre dataset de signaux bruités

par la suite, et de la même manière que décrit sur le papier, nous avons constitué un dataset plus large de signaux bruités en ajoutant du bruit aux signaux d'origines avec différents SNR de -2.5, -1, 0, 2.5, 3, 5, et 7.5.

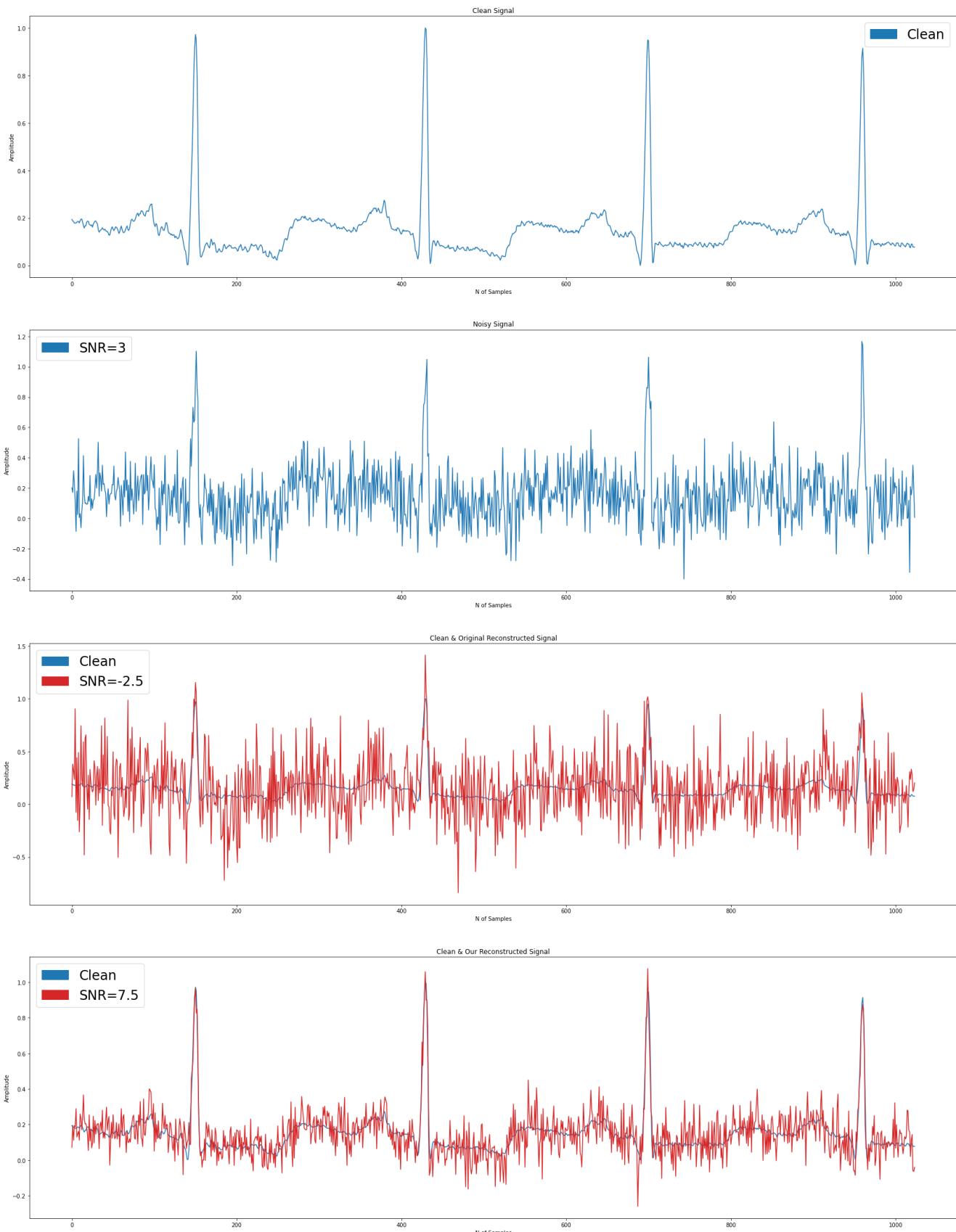


fig7. échantillon signal ECG propre, bruité, bruité avec un SNR=-2.5 et bruité avec SNR=7.5

Puis nous avons réentraîné un modèle avec ces nouvelles données.

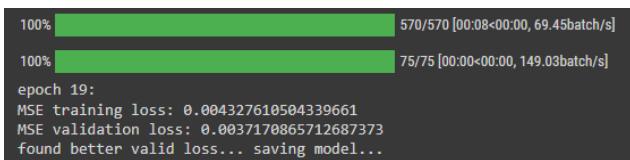


fig8. best performances model 2

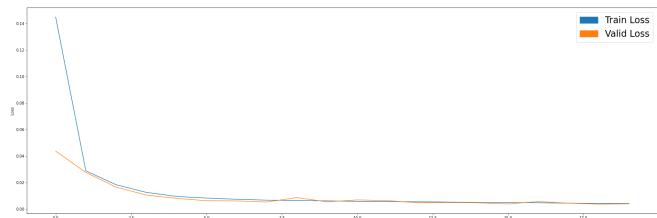


fig9. train and validation curves

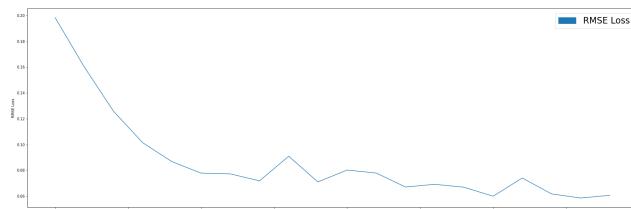


fig10. loss curve RMSE

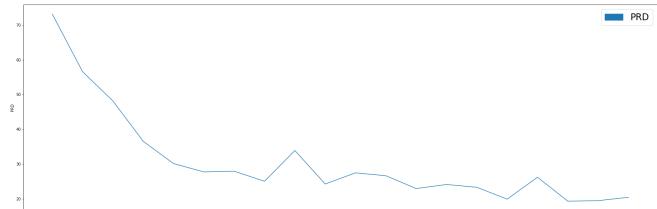


fig11. loss curve PRD

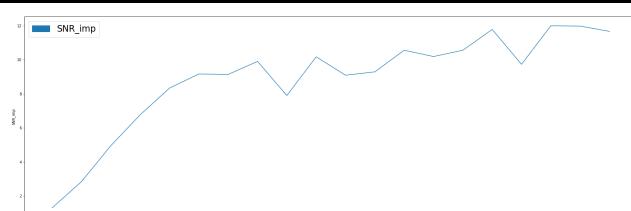


fig12. loss curve SNR_img

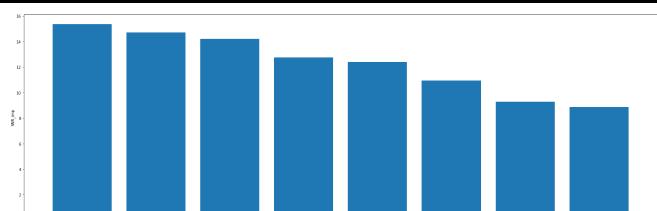


fig13. SNR_img ratio

Echantillons de signaux propres, bruités avec SNR de -2.5 et 7.5 et débruités:

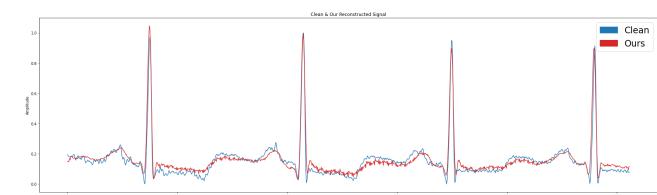
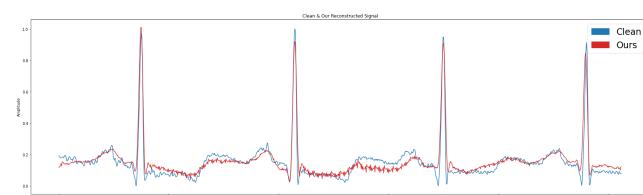
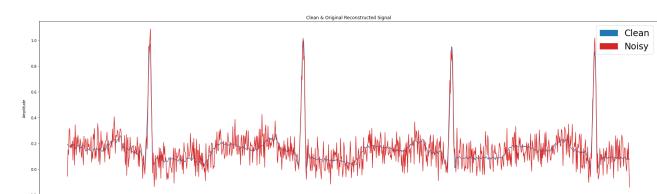
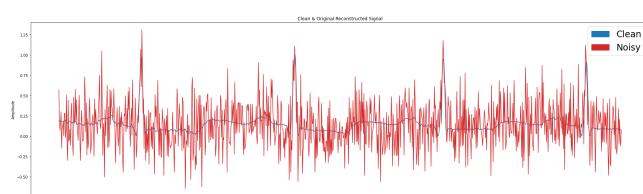
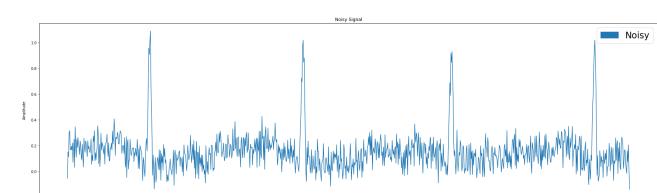
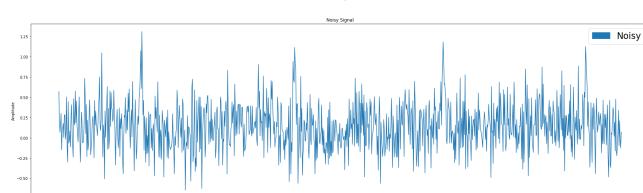
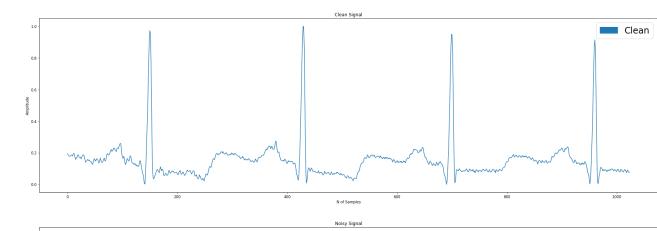
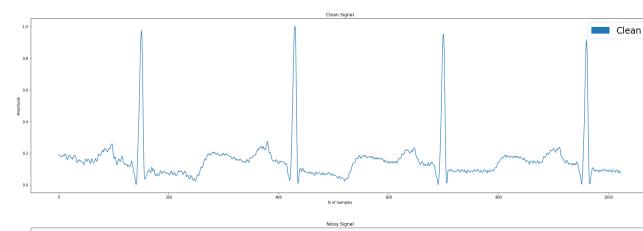


fig14. échantillon signal ECG propre, bruité avec SNR=-2.5 et débruité

fig15. échantillon signal ECG propre, bruité avec SNR=7.5 et débruité

3. Application du modèle sur des signaux PCG

Le but final de ce projet est d'appliquer cette approche de débruitage sur des signaux phonocardiogrammes (PCG) qui sont des enregistrements sonores des bruits cardiaques.

Pour cette tâche, nous avions accès à des signaux PCG propres et bruités que nous avons standardisés:

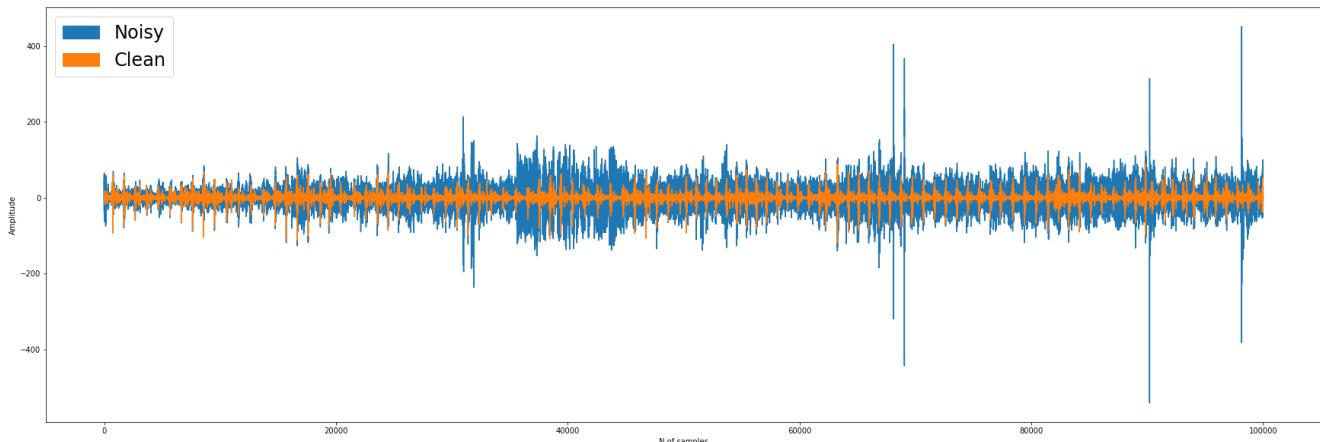


fig16. échantillon signal PCG brut

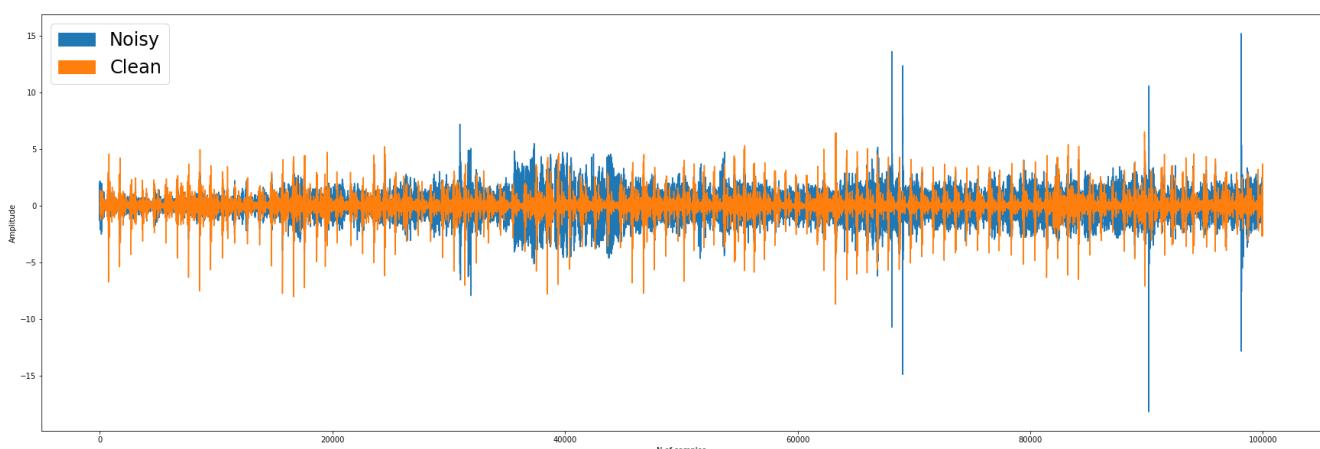


fig17. échantillon signal PCG standardisé

On peut voir que les signaux PCG propres sur lesquelles a été effectué l'expérimentation sont à la base nettement plus bruité que les signaux ECG et le bruit est surtout réparti de manière inégale sur les signaux.

Et finalement nous avons entraîné le modèle avec les signaux PCG de la même manière que précédemment avec les signaux ECG.

Résultats de l'entraînement sur signaux PCG

```
100% [██████████] 75/75 [00:01<00:00, 67.71batch/s]
100% [██████████] 10/10 [00:00<00:00, 87.00batch/s]
epoch 97:
MSE training loss: 0.04177211955189705
MSE validation loss: 0.29463613107800485
found better valid loss... saving model...
```

fig18 best performances model 3 (PCG)

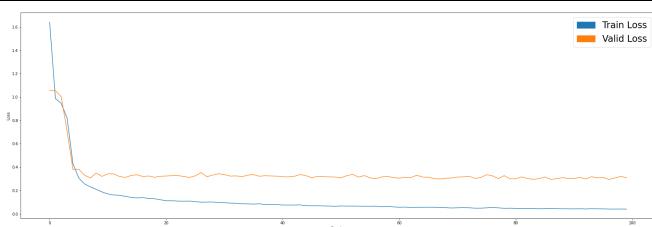


fig19. train and validation curves

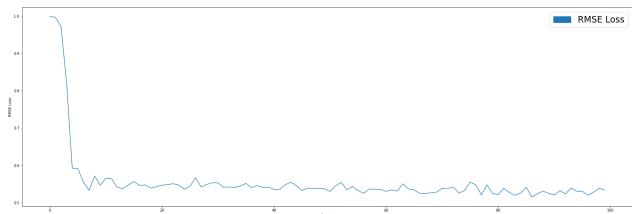


fig20. loss curve RMSE

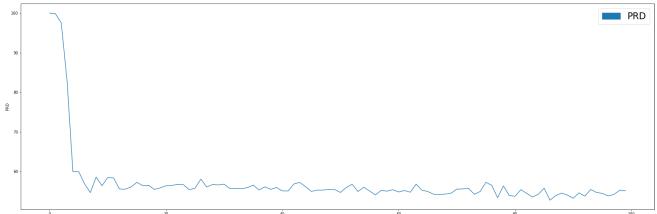


fig21. loss curve PRD

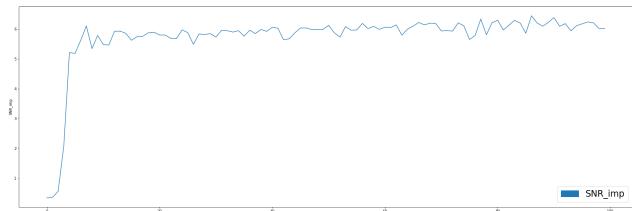


fig22. loss curve SNR_imp

Echantillons de signaux PCG propres, bruités et débruités:

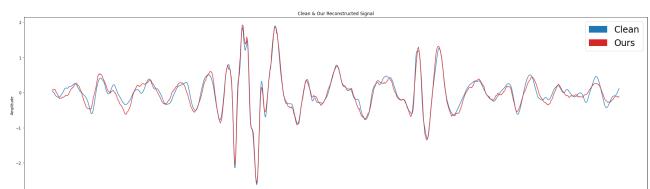
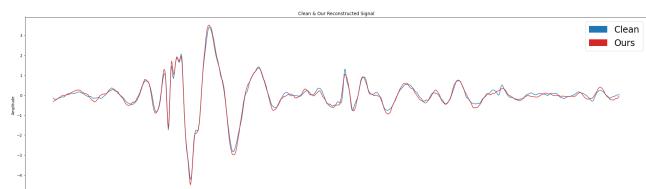
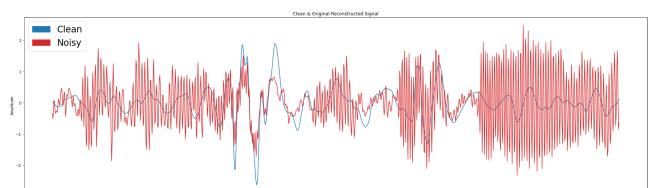
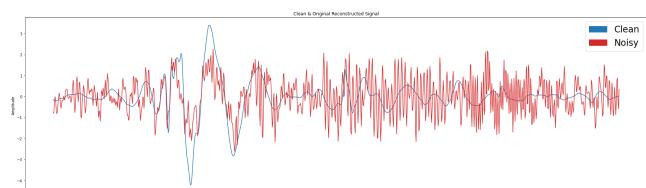
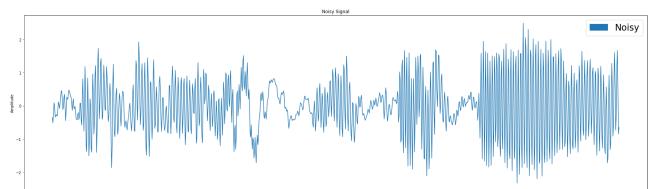
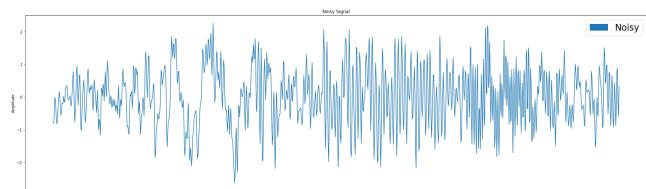
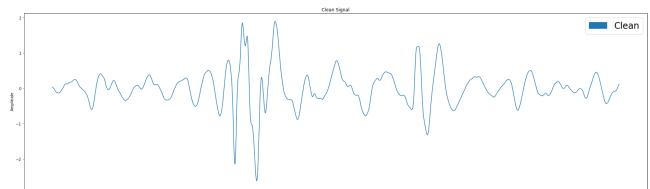
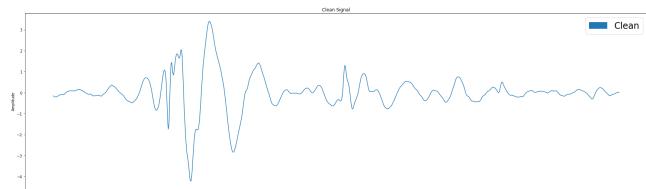


fig23. échantillon signal PCG propre, bruité et débruité (meilleur cas 1)

fig24. échantillon signal PCG propre, bruité et débruité (meilleur cas 2)

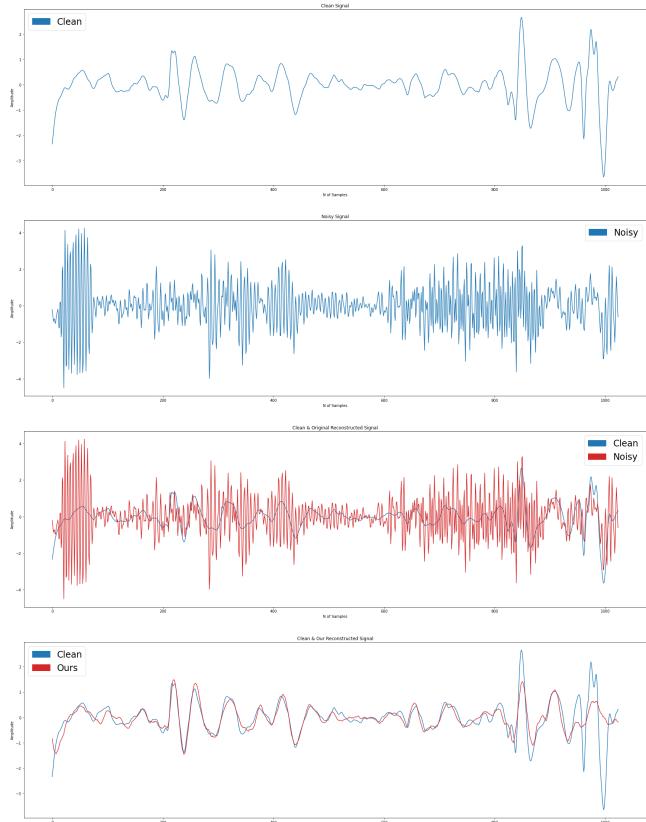


fig25. échantillon signal PCG propre, bruité et débruité (pire cas)

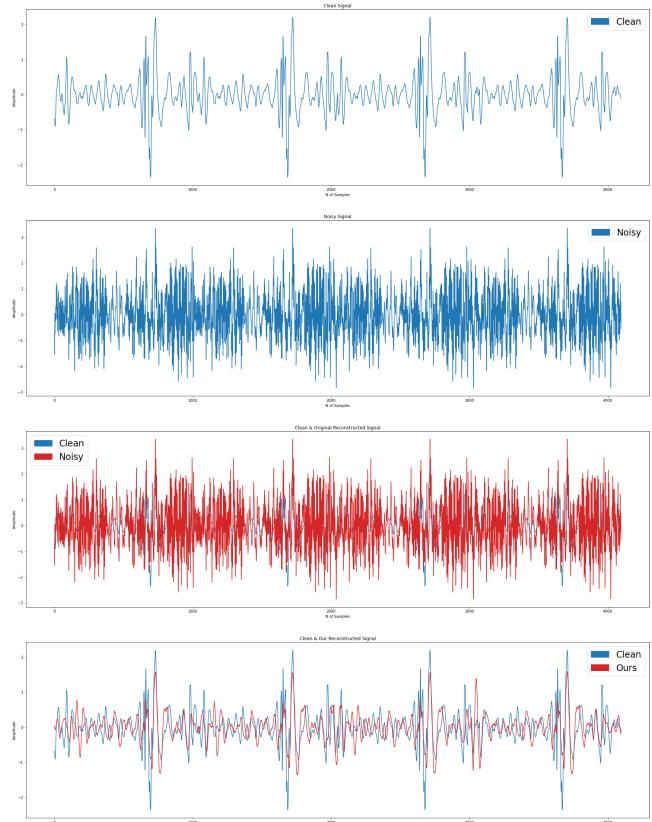


fig26. échantillon signal PCG propre, bruité et débruité (test set)