DLTS_Projet_Habasque_Coerchon

7 Janvier 2025

Projet de Traitement du Signal et Deep Learning

Denoising

Réalisé par :

- **Chloé Habasque**
- **Colin Coërchon**

Date : 2024/2025

Sujet : Le but du projet est de retrouver le signal de voix non bruité, d'un signal de voix bruité.

Pour cela, nous avons accès à différents extraits de voix avec et sans bruit et nous avons fait tourner différents réseaux de neurones sur celles-ci.

---

1 Import

[11]:
```
import numpy as np
import matplotlib.pyplot as plt
import os

#torch
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.nn.functional as F
from torch import optim
import torchaudio

import librosa
from pesq import pesq
from pystoi import stoi
```

```
import IPython.display as ipd
import glob
import random

import os
os.environ["KMP_DUPLICATE_LIB_OK"] = "True"  # Temporaire pour éviter le crash
```

[12]:
```
noise_trainsmall_path = './denoising/train_small'
noise_train_path = "./denoising/train"
noise_test_path = "./denoising/test"

original_trainsmall_path = "./voice_origin/train_small"
original_train_path = "./voice_origin/train"
original_test_path = "./voice_origin/test"
```

2 Dataset

[13]:
```
class VoiceDataset(Dataset):
    def __init__(self, clean_paths, noisy_paths, sample_rate=8000):
        """
        clean_paths : liste des chemins vers les fichiers 'clean'
        noisy_paths : liste des chemins vers les fichiers 'noisy' (même ordre␣
↪que clean_paths)
        sample_rate : fréquence d'échantillonnage voulue
        """
        super().__init__()
        self.clean_paths = clean_paths
        self.noisy_paths = noisy_paths
        self.sample_rate = sample_rate

    def __len__(self):
        return len(self.clean_paths)

    def __getitem__(self, idx):
        clean_path = self.clean_paths[idx]
        noisy_path = self.noisy_paths[idx]

        clean_waveform, sr_clean = torchaudio.load(clean_path)
        noisy_waveform, sr_noisy = torchaudio.load(noisy_path)

        # Rééchantillonnage si nécessaire
        if sr_clean != self.sample_rate:
            resampler = torchaudio.transforms.Resample(sr_clean, self.
↪sample_rate)
            clean_waveform = resampler(clean_waveform)
        if sr_noisy != self.sample_rate:
```

```
            resampler = torchaudio.transforms.Resample(sr_noisy, self.
↪sample_rate)
            noisy_waveform = resampler(noisy_waveform)

        return noisy_waveform, clean_waveform
```

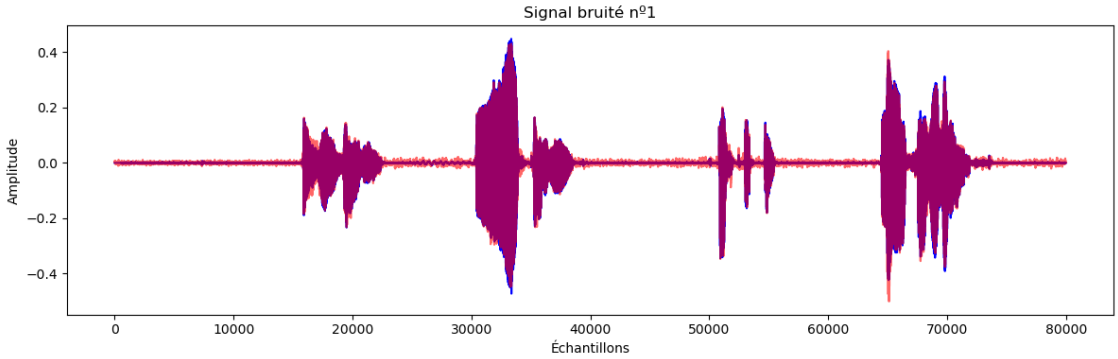
3 Aperçu des données

[14]:
```
class SpeechDenoisingDataset(Dataset):
    def __init__(self, clean_dir, noisy_dir, transform=None, sample_rate=8000):
        self.clean_dir = clean_dir
        self.noisy_dir = noisy_dir
        self.transform = transform
        self.sample_rate = sample_rate
        self.clean_files = sorted(os.listdir(clean_dir))
        self.noisy_files = sorted(os.listdir(noisy_dir))

        # Ensure both directories have the same number of files
        assert len(self.clean_files) == len(self.noisy_files), \
            "Les dossiers clean et noisy doivent avoir le même nombre de␣
↪fichiers"

    def __len__(self):
        return len(self.clean_files)

    def __getitem__(self, idx):
        clean_path = os.path.join(self.clean_dir, self.clean_files[idx])
        noisy_path = os.path.join(self.noisy_dir, self.noisy_files[idx])

        # Load audio files
        clean_wave, _ = librosa.load(clean_path, sr=self.sample_rate)
        noisy_wave, _ = librosa.load(noisy_path, sr=self.sample_rate)

        # Ensure both waves are of the same length
        min_len = min(len(clean_wave), len(noisy_wave))
        clean_wave = clean_wave[:min_len]
        noisy_wave = noisy_wave[:min_len]

        clean_wave = torch.tensor(clean_wave, dtype=torch.float32)
        noisy_wave = torch.tensor(noisy_wave, dtype=torch.float32)

        if self.transform:
            clean_wave = self.transform(clean_wave)
            noisy_wave = self.transform(noisy_wave)

        return noisy_wave, clean_wave
```

```
trainsmall_dataset = SpeechDenoisingDataset(original_trainsmall_path,␣
↪noise_trainsmall_path)
```

[15]:
```
# Display a few examples
nbre_exemples = 5
for i in range(nbre_exemples):

    print("-------------------------\n")
    print(f"Audio Original {i+1}:")
    noisy, clean = trainsmall_dataset[i]
    ipd.display(ipd.Audio(clean.numpy(), rate=trainsmall_dataset.sample_rate))
    print(f"Audio Bruité {i+1}:")
    ipd.display(ipd.Audio(noisy.numpy(), rate=trainsmall_dataset.sample_rate))

    # Convert tensors to numpy arrays
    noisy = noisy.numpy()
    clean = clean.numpy()

    # Plot waveforms
    plt.figure(figsize=(12, 4))

    plt.plot(clean, color="blue")
    plt.title(f"Signal propre (Original) n°{i+1}")
    plt.xlabel("Échantillons")
    plt.ylabel("Amplitude")

    plt.plot(noisy, color="red", alpha=0.6)
    plt.title(f"Signal bruité n°{i+1}")
    plt.xlabel("Échantillons")
    plt.ylabel("Amplitude")

    plt.tight_layout()
    plt.show()
```

```
-------------------------

Audio Original 1:

<IPython.lib.display.Audio object>

Audio Bruité 1:

<IPython.lib.display.Audio object>
```

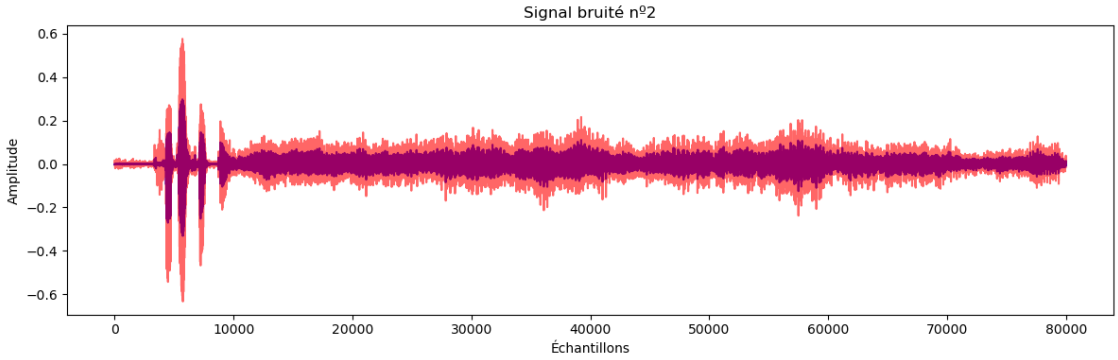


Audio Original 2:

<IPython.lib.display.Audio object>

Audio Bruité 2:

<IPython.lib.display.Audio object>

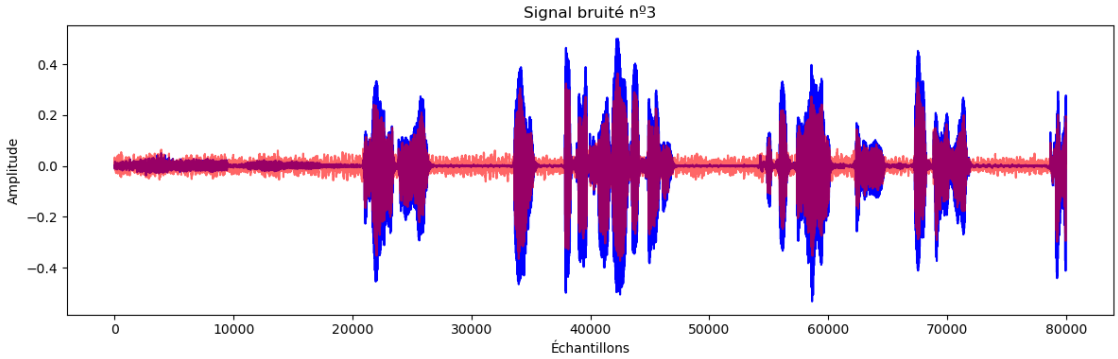


Audio Original 3:

<IPython.lib.display.Audio object>

Audio Bruité 3:

<IPython.lib.display.Audio object>

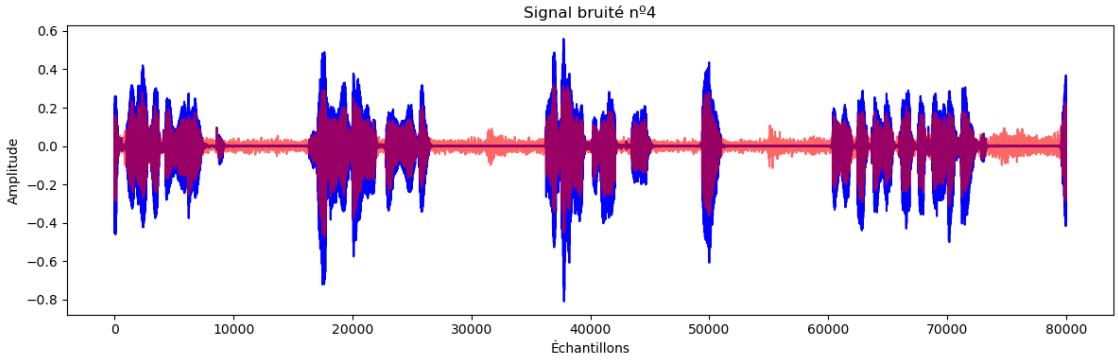


Audio Original 4:

<IPython.lib.display.Audio object>

Audio Bruité 4:

<IPython.lib.display.Audio object>

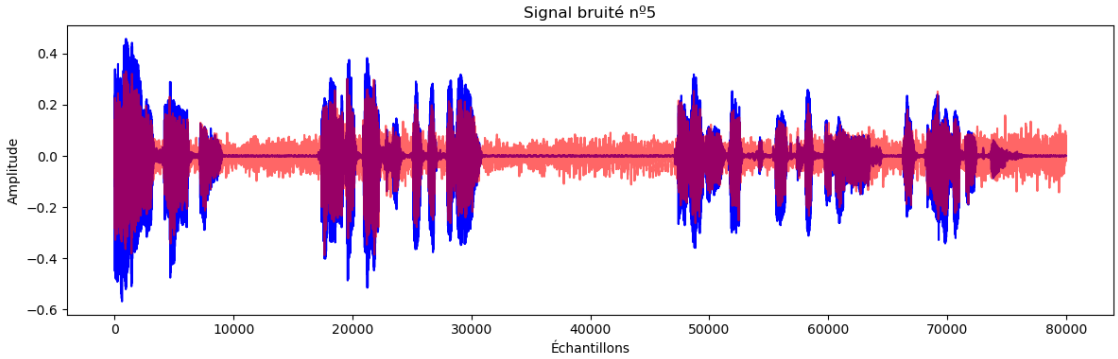


Audio Original 5:

<IPython.lib.display.Audio object>

Audio Bruité 5:

<IPython.lib.display.Audio object>



4 Model

4.1 ConvTasNet Model

[16]:
```python
class ConvTasNet(nn.Module):
    def __init__(self, L=16, N=256, H=256, X=3, R=2):
      # self, L=16, N=256, H=512, X=8, R=3

        """
        L, N, B, H, Sc, X, R sont des hyperparamètres.
        - L : longueur du filtre de l'encoder (en échantillons)
        - N : dimension du la sortie de l'encodeur
        - H : dimension des couches internes du TCN
        - X : nombre de couches dans un bloc TCN
        - R : nombre de blocs TCN répétés
        """
        super().__init__()

        # 1) Encodeur
        self.encoder = nn.Conv1d(
            in_channels=1,
            out_channels=N,
            kernel_size=L,
            stride=L//2,  # exemple
            bias=False
        )

        # 2) Masque: TCN
        self.tcn = nn.Sequential(
            *[self._build_block(N, H, X) for _ in range(R)]
        )

        # 3) Décodeur
```

```
        self.decoder = nn.ConvTranspose1d(
            in_channels=N,
            out_channels=1,
            kernel_size=L,
            stride=L//2,
            bias=False
        )

    def _build_block(self, in_channels, H, X):
        """Construit un bloc TCN simplifié"""
        layers = []
        for x in range(X):
            dilation = 2 ** x
            layers.append(
                nn.Conv1d(in_channels, H, 3, padding=dilation, dilation=dilation)
            )
            layers.append(nn.ReLU())
            layers.append(nn.Conv1d(H, in_channels, 1))
        return nn.Sequential(*layers)

    def forward(self, mixture):
        """
        mixture: (batch, 1, T)
        """
        # 1) encode
        enc_out = self.encoder(mixture)  # shape: (batch, N, T')

        # 2) estimer un masque dans l'espace encodé
        mask = self.tcn(enc_out)         # shape: (batch, N, T')

        # 3) appliquer le masque
        sep_features = enc_out * mask    # (batch, N, T')

        # 4) décoder
        out_wav = self.decoder(sep_features)  # (batch, 1, T)

        return out_wav
```

4.2 TasNet Model

[17]:
```
class TasNet(nn.Module):
    def __init__(self, L=8, N=256, H=128, X=2):
        """
        L, N, B, H, Sc, X, R sont des hyperparamètres.
        - L : longueur du filtre de l'encoder (en échantillons)
        - N : dimension du la sortie de l'encodeur
        - H : dimension des couches internes du LSTM
```

```
        - X : nombre de couches dans le bloc LSTM
        """
        super(TasNet, self).__init__()

        # 1) Encodeur
        self.encoder = nn.Conv1d(
            in_channels=1,
            out_channels=N,
            kernel_size=L,
            stride=L // 2,
            bias=False
        )

        # 2) Séparation utilisant des LSTM
        self.layer_norm = nn.LayerNorm(N)
        self.rnn = nn.LSTM(
            input_size=N,
            hidden_size=H,
            num_layers=X,
            batch_first=True,
            bidirectional=True
        )
        self.fc = nn.Linear(H * 2, N)

        # 3) Décodeur
        self.decoder = nn.ConvTranspose1d(
            in_channels=N,
            out_channels=1,
            kernel_size=L,
            stride=L // 2,
            bias=False
        )

    def forward(self, mixture):
        """
        Forward pass :
        - mixture : signal d'entrée (batch, 1, T)
        """
        # 1) Encodeur
        enc_out = self.encoder(mixture)  # (batch, N, T')

        # 2) Séparation
        enc_out = enc_out.permute(0, 2, 1)  # (batch, T', N)
        norm_enc_out = self.layer_norm(enc_out)
        rnn_out, _ = self.rnn(norm_enc_out)  # (batch, T', H * 2)
        sep_features = self.fc(rnn_out)  # (batch, T', N)
        sep_features = sep_features.permute(0, 2, 1)  # Revenir à (batch, N, T')
```

```
        # 3) Décodage
        out_wav = self.decoder(sep_features)  # (batch, 1, T)

        return out_wav
```

4.3 WaveUNet Model

[18]:
```
class WaveUNet(nn.Module):
    def __init__(self):
        super(WaveUNet, self).__init__()

        def conv_block(in_channels, out_channels):
            return nn.Sequential(
                nn.Conv1d(in_channels, out_channels, kernel_size=5, padding=2),
                nn.ReLU(),
                nn.BatchNorm1d(out_channels)
            )

        def downsample_block(in_channels, out_channels):
            return nn.Sequential(
                conv_block(in_channels, out_channels),
                nn.MaxPool1d(2)
            )

        def upsample_block(in_channels, out_channels):
            return nn.Sequential(
                nn.Upsample(scale_factor=2, mode='linear', align_corners=True),
                conv_block(in_channels, out_channels)
            )

        self.down1 = downsample_block(1, 64)
        self.down2 = downsample_block(64, 128)
        self.down3 = downsample_block(128, 256)
        self.down4 = downsample_block(256, 512)

        self.bottleneck = conv_block(512, 1024)

        self.up4 = upsample_block(1024 + 512, 512)
        self.up3 = upsample_block(512 + 256, 256)
        self.up2 = upsample_block(256 + 128, 128)
        self.up1 = upsample_block(128 + 64, 64)

        self.final_conv = nn.Conv1d(64, 1, kernel_size=1)

    def forward(self, x):
        d1 = self.down1(x)
```

```
        d2 = self.down2(d1)
        d3 = self.down3(d2)
        d4 = self.down4(d3)

        bottleneck = self.bottleneck(d4)

        u4 = self.up4(torch.cat([bottleneck, d4], dim=1))
        u3 = self.up3(torch.cat([u4, d3], dim=1))
        u2 = self.up2(torch.cat([u3, d2], dim=1))
        u1 = self.up1(torch.cat([u2, d1], dim=1))

        return self.final_conv(u1)
```

5 Séparation Train/Validation

[19]:
```
def prepare_dataloaders(clean_dir, noisy_dir, batch_size=4, valid_ratio=0.2,␣
↪sample_rate=8000):
    """
    clean_dir  : dossier contenant les fichiers audio 'propres'  (ex: audio/
↪voice_origin/train)
    noisy_dir  : dossier contenant les fichiers audio 'bruités' (ex: audio/
↪denoising/train)
    batch_size : taille du batch
    valid_ratio: ratio pour la validation (ex: 0.2 => 20%)
    sample_rate: fréquence d'échantillonnage voulue
    """

    # Récupérer la liste des fichiers
    clean_files = sorted(glob.glob(os.path.join(clean_dir, "*.wav")))

    noisy_files = []
    for cf in clean_files:
        filename = os.path.basename(cf)  # ex: spk001_0001_clean.wav
        noisy_filename = filename.replace("clean", "noisy")
        noisy_files.append(os.path.join(noisy_dir, noisy_filename))

    # Vérifier qu'on a le même nombre de clean et noisy
    assert len(clean_files) == len(noisy_files), "Mismatch entre clean et noisy␣
↪files."

    # On fait un shuffle coordonné entre clean et noisy
    paired = list(zip(clean_files, noisy_files))
    random.shuffle(paired)  # mélange la liste
    clean_files, noisy_files = zip(*paired)  # re-sépare en deux listes

    # Split en 80%-20% (train/val)
```

```
    total_files = len(clean_files)
    split_index = int((1 - valid_ratio) * total_files)  # ex: 80% du total

    train_clean = clean_files[:split_index]
    train_noisy = noisy_files[:split_index]

    val_clean = clean_files[split_index:]
    val_noisy = noisy_files[split_index:]

    print("t clean",len(train_clean))
    print("t noisy",len(train_noisy))
    print("v clean",len(val_clean))
    print("v noisy",len(val_noisy))

    # On crée les datasets
    train_dataset = VoiceDataset(train_clean, train_noisy,␣
↪sample_rate=sample_rate)
    val_dataset   = VoiceDataset(val_clean,   val_noisy,  ␣
↪sample_rate=sample_rate)

    # On crée les DataLoaders
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_loader   = DataLoader(val_dataset,   batch_size=batch_size,␣
↪shuffle=False)

    return train_loader, val_loader
```

6 Loss function

L'objectif de l'entraînement est de maximiser le rapport signal-sur-bruit invariant (SI-SNR), qui est couramment utilisé comme métrique d'évaluation pour la séparation des sources. Le SI-SNR est défini comme suit :

$$\hat{x}_{\|} = \frac{\langle \hat{x}, x \rangle x}{\|x\|} \quad e = \hat{x} - \hat{x}_{\|}$$

$$\text{SI-SNR} = 10 \log_{10} \frac{\|\hat{x}_{\|}\|^2}{\|e\|^2}$$

[20]:
```
def si_snr_loss(clean, estimated):
    """
    Compute Scale-Invariant Signal-to-Distortion Ratio (SI-SDR) loss.
    clean: (batch, num_channels, time)
    estimated: (batch, num_channels, time)
    """
    losses = []
```

```
    for i in range(clean.size(0)):  # Parcourir le batch
        clean_i = clean[i].view(-1)
        estimated_i = estimated[i].view(-1)

        # Ajuster les tailles pour correspondre
        min_length = min(clean_i.size(0), estimated_i.size(0))
        clean_i = clean_i[:min_length]
        estimated_i = estimated_i[:min_length]

        # Calcul de alpha
        alpha = torch.dot(clean_i, estimated_i) / torch.dot(clean_i, clean_i)

        # Calcul du signal cible et du bruit
        target = alpha * clean_i
        noise = estimated_i - target

        # Calcul du SI-SDR
        sdr = 10 * torch.log10(torch.norm(target) ** 2 / torch.norm(noise) ** 2)

        # Ajouter la perte (négative car on veut minimiser)
        losses.append(-sdr)

    return torch.mean(torch.stack(losses))
```

7 Train+Validation function

[21]:
```
def train_one_epoch(model, dataloader, optimizer, criterion, device):
    model.train()
    total_loss = 0.0

    for batch_idx, (noisy, clean) in enumerate(dataloader):
        if batch_idx%10==0:
          print(batch_idx)

        # Envoyer sur le GPU si disponible
        noisy, clean = noisy.to(device), clean.to(device)

        # Sortie du modèle
        predicted = model(noisy)  # shape: (batch, 1, T)

        # # Empiler les prédictions pour correspondre à la taille de clean
        # predicted = torch.stack(predicted, dim=1)  # (batch_size, num_spks, T)

        # # Ajuster la taille de clean si nécessaire
        # clean = clean.unsqueeze(1) if clean.dim() == 2 else clean
```

```
        # Calcul de la perte
        # Par exemple, MSE sur l'onde
        loss = criterion(predicted, clean)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    return total_loss / len(dataloader)

# def validate(model, dataloader, criterion, device):
#     model.eval()
#     val_loss = 0.0

#     with torch.no_grad():
#         for batch_idx, (noisy, clean) in enumerate(dataloader):
#             noisy, clean = noisy.to(device), clean.to(device)
#             predicted = model(noisy)
#             loss = criterion(predicted, clean)
#             val_loss += loss.item()

#     return val_loss / len(dataloader)

def validate(model, dataloader, criterion, device, sr=8000):

    model.eval()
    val_loss = 0.0

    pesq_scores = []
    stoi_scores = []

    # Choix du mode PESQ (wide band ou narrow band)
    pesq_mode = 'wb' if sr == 8000 else 'nb'

    with torch.no_grad():
        for batch_idx, (noisy, clean) in enumerate(dataloader):
            if batch_idx%10==0:
              print(batch_idx)

            noisy = noisy.to(device)
            clean = clean.to(device)
            predicted = model(noisy)  # shape: (batch, 1, T)
```

```
            # predicted = torch.stack(predicted, dim=1)  # Ajoutez une dimension␣
↪pour correspondre à la taille attendue
            # clean = clean.unsqueeze(1) if clean.dim() == 2 else clean



            loss = criterion(predicted, clean)
            val_loss += loss.item()

            # Calcul de PESQ et STOI
            # clean_np = clean.cpu().numpy().squeeze(1)
            # pred_np  = predicted.cpu().numpy().squeeze(1)

            # # Calcul des métriques pour chaque élément du batch
            # for i in range(clean_np.shape[0]):
            #     # PESQ
            #     pesq_val = pesq(sr, clean_np[i], pred_np[i], pesq_mode)
            #     pesq_scores.append(pesq_val)

            #     # STOI
            #     stoi_val = stoi(clean_np[i], pred_np[i], sr, extended=False)
            #     stoi_scores.append(stoi_val)

    # Moyenne des pertes et des métriques
    mean_val_loss = val_loss / len(dataloader)
    mean_pesq = np.mean(pesq_scores) if pesq_scores else 0.0
    mean_stoi = np.mean(stoi_scores) if stoi_scores else 0.0

    return mean_val_loss, mean_pesq, mean_stoi
```

8 Train

[12]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Charger un modèle ConvTasNet
model = ConvTasNet()
# model = TasNet()
# model = WaveUNet()
model.to(device)


print(device)
# Hyperparamètres
batch_size = 10
learning_rate = 1e-3
num_epochs = 25
```

```
best_val_loss = float('inf')

train_loss_list=[]
valid_loss_list=[]
pesq_list=[]
stoi_list=[]

# Répertoire contenant les .wav "propres"
train_clean_dir = "./voice_origin/train"
# Répertoire contenant les .wav "bruités"
train_noisy_dir = "./denoising/train"

# Préparation des DataLoaders (80%/20% split)
train_loader, val_loader = prepare_dataloaders(
    clean_dir=train_clean_dir,
    noisy_dir=train_noisy_dir,
    batch_size=batch_size,
    valid_ratio=0.2,   # 20% pour validation
    sample_rate=8000
)

# Test rapide : on itère sur un batch
# for noisy_batch, clean_batch in train_loader:
#     print("noisy_batch shape :", noisy_batch.shape)
#     print("clean_batch shape :", clean_batch.shape)
#     break

print("Nombre de batchs en train :", len(train_loader))
print("Nombre de batchs en validation :", len(val_loader))


# Choix de la fonction de perte
#criterion = nn.MSELoss()
criterion = si_snr_loss
# (On peut aussi utiliser la L1, la SI-SDR Loss, etc.)

# Optimiseur
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Boucle d'entrainement
for epoch in range(num_epochs):
    train_loss = train_one_epoch(model, train_loader, optimizer, criterion,␣
 ↪device)
    val_loss, mean_pesq, mean_stoi = validate(model, val_loader, criterion,␣
 ↪device)
```

```
    train_loss_list.append(train_loss)
    valid_loss_list.append(val_loss)
    pesq_list.append(mean_pesq)
    stoi_list.append(mean_stoi)

    print(f"Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Val␣
 ↪Loss: {val_loss:.4f}")#, PESQ: {mean_pesq:.3f}, STOI: {mean_stoi:.3f}")

    # Check if this is the best accuracy so far and save if it is
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        path_to_save = "./model/"
        torch.save(model.state_dict(), os.path.join(path_to_save, 'tasnet1_extra.
 ↪pt'))

print("Entraînement terminé !")

# Création de la figure
plt.figure(figsize=(18, 6))

# Sous-plot 1 : Évolution de la Loss
plt.subplot(1, 3, 1)
plt.plot(train_loss_list, label="Train Loss")
plt.plot(valid_loss_list, label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.title("Évolution de la Loss")

# # Sous-plot 2 : PESQ
# plt.subplot(1, 3, 2)
# plt.plot(pesq_list, color='orange')
# plt.xlabel("Epochs")
# plt.ylabel("PESQ")
# plt.title("PESQ Score")

# # Sous-plot 3 : STOI
# plt.subplot(1, 3, 3)
# plt.plot(stoi_list, color='green')
# plt.xlabel("Epochs")
# plt.ylabel("STOI")
# plt.title("STOI Score")

# Affichage de la figure
plt.tight_layout()
plt.show()
```

```
cuda
t clean 1694
t noisy 1694
v clean 424
v noisy 424
Nombre de batchs en train : 170
Nombre de batchs en validation : 43
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [1/25], Train Loss: -11.9738, Val Loss: -14.1807
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
```

```
10
20
30
40
Epoch [2/25], Train Loss: -14.9637, Val Loss: -15.7276
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [3/25], Train Loss: -16.0265, Val Loss: -16.3468
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
```

```
30
40
Epoch [4/25], Train Loss: -16.5237, Val Loss: -16.6994
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [5/25], Train Loss: -16.9226, Val Loss: -16.9884
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
```

```
Epoch [6/25], Train Loss: -17.1873, Val Loss: -17.3440
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [7/25], Train Loss: -17.4585, Val Loss: -17.4737
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [8/25], Train Loss: -17.6196, Val Loss: -17.5190
0
```

```
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [9/25], Train Loss: -17.7105, Val Loss: -17.7000
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [10/25], Train Loss: -17.8326, Val Loss: -17.6313
0
10
20
```

```
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [11/25], Train Loss: -17.9489, Val Loss: -17.8878
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [12/25], Train Loss: -18.0350, Val Loss: -17.9999
0
10
20
30
40
```

```
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [13/25], Train Loss: -18.1372, Val Loss: -18.0028
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [14/25], Train Loss: -18.1992, Val Loss: -18.0061
0
10
20
30
40
50
60
```

```
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [15/25], Train Loss: -18.2413, Val Loss: -18.1464
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [16/25], Train Loss: -18.3385, Val Loss: -18.0443
0
10
20
30
40
50
60
70
80
```

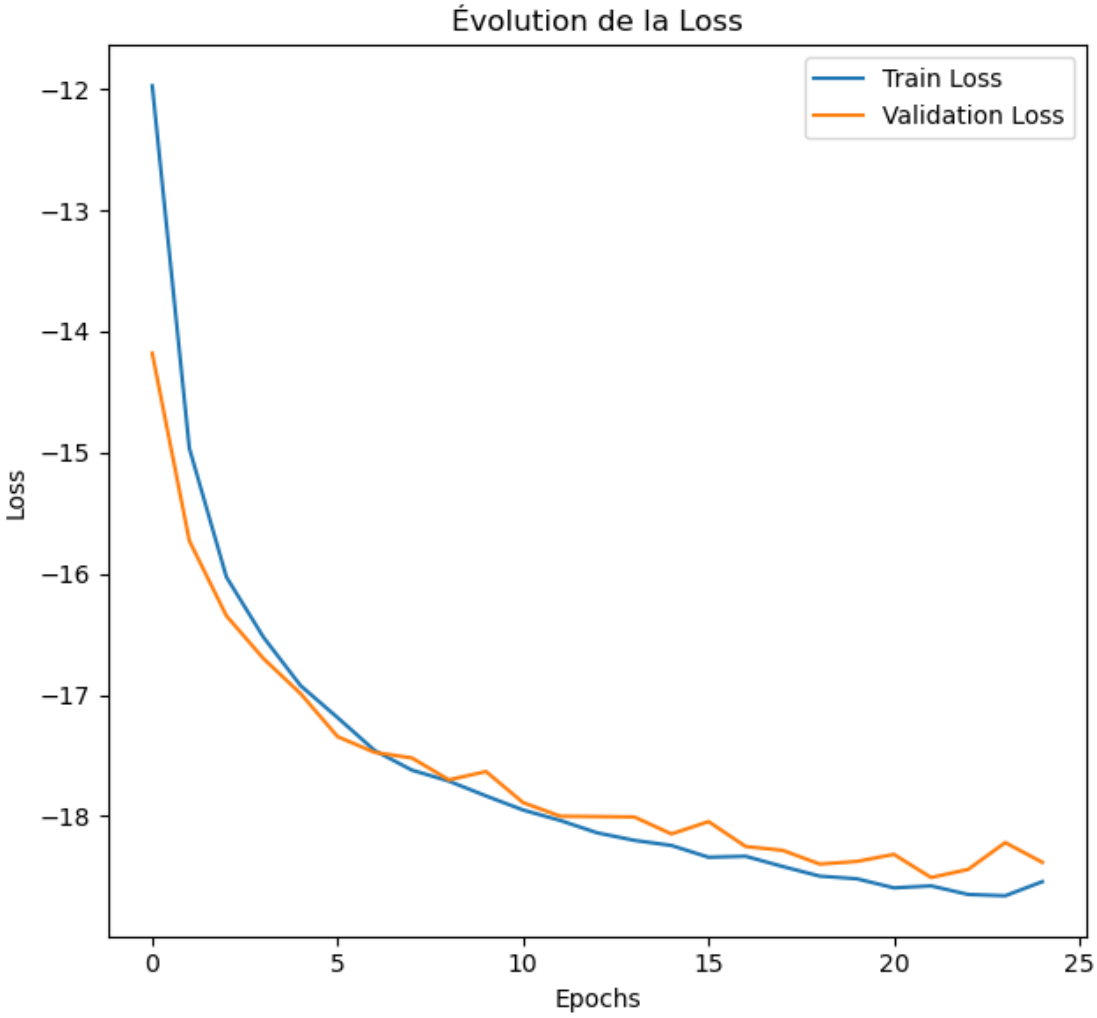
```
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [17/25], Train Loss: -18.3304, Val Loss: -18.2497
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [18/25], Train Loss: -18.4151, Val Loss: -18.2816
0
10
20
30
40
50
60
70
80
90
100
```

```
110
120
130
140
150
160
0
10
20
30
40
Epoch [19/25], Train Loss: -18.4944, Val Loss: -18.3949
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [20/25], Train Loss: -18.5163, Val Loss: -18.3730
0
10
20
30
40
50
60
70
80
90
100
110
120
```

```
130
140
150
160
0
10
20
30
40
Epoch [21/25], Train Loss: -18.5912, Val Loss: -18.3148
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [22/25], Train Loss: -18.5748, Val Loss: -18.5059
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
```

```
150
160
0
10
20
30
40
Epoch [23/25], Train Loss: -18.6456, Val Loss: -18.4389
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
0
10
20
30
40
Epoch [24/25], Train Loss: -18.6564, Val Loss: -18.2177
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
```

```
0
10
20
30
40
Epoch [25/25], Train Loss: -18.5401, Val Loss: -18.3802
Entraînement terminé !
```



[8]:
```python
# Chemin complet pour le fichier PDF
output_file = os.path.join("./images/", "tasnet1_extra_loss.pdf")

# Création de la figure
plt.figure(figsize=(12, 6))

# Plot 1 : Évolution de la Loss
```

```
plt.plot(train_loss_list, label="Train Loss")
plt.plot(valid_loss_list, label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.title("Évolution de la Loss")

plt.tight_layout()

# Enregistrement dans un fichier PDF
plt.savefig(output_file, format="pdf")
plt.close()

print(f"Figure enregistrée dans {output_file}")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 8
      5 plt.figure(figsize=(12, 6))
      7 # Plot 1 : Évolution de la Loss
----> 8 plt.plot(train_loss_list, label="Train Loss")
      9 plt.plot(valid_loss_list, label="Validation Loss")
     10 plt.xlabel("Epochs")

NameError: name 'train_loss_list' is not defined
```

```
<Figure size 1200x600 with 0 Axes>
```

9 Test

9.0.1 PESQ (Perceptual Evaluation of Speech Quality)

- **Définition** : Méthode d'évaluation objective de la **qualité perçue** de la parole.
- **Échelle** : -0.5 à 4.5 (similaire au MOS), plus le PESQ est élévé plus le signal est bon.
- **Principe** : Compare un signal vocal propre avec un signal dégradé en tenant compte de la perception humaine.

9.0.2 STOI (Short-Time Objective Intelligibility)

- **Définition** : Méthode d'évaluation objective de **l'intelligibilité** de la parole.
- **Échelle** : 0 à 1 (1 = parfaitement intelligible)
- **Principe** : Compare les caractéristiques temporelles du signal propre et du signal dégradé sur de courtes fenêtres temporelles.

[24]:
```python
def evaluate_metrics(model, test_loader, device, sr=8000):
    model.eval()
    pesq_scores = []
    stoi_scores = []
    k=0

    pesq_mode = 'wb' if sr == 16000 else 'nb'

    with torch.no_grad():
        for noisy, clean in test_loader:
            if k%10==0:
                print(k)
            k+=1
            noisy, clean = noisy.to(device), clean.to(device)
            pred = model(noisy)

            # Convertir en numpy + squeeze
            clean_np = clean.cpu().numpy().squeeze(1)
            pred_np  = pred.cpu().numpy().squeeze(1)

            for i in range(clean_np.shape[0]):
                # Pesq : l'ordre correct est pesq(fs, ref, deg, mode)
                pesq_val = pesq(sr, clean_np[i], pred_np[i], pesq_mode)
                pesq_scores.append(pesq_val)

                # Stoi
                stoi_val = stoi(clean_np[i], pred_np[i], sr, extended=False)
                stoi_scores.append(stoi_val)

    mean_pesq = np.mean(pesq_scores)
    mean_stoi = np.mean(stoi_scores)

    print(f"Mean PESQ: {mean_pesq:.3f}, Mean STOI: {mean_stoi:.3f}")
    return mean_pesq, mean_stoi
```

[25]:
```python
def prepare_test_dataset(clean_dir, noisy_dir, batch_size=1, sample_rate=8000):
    """
    Crée un DataLoader pour l'ensemble de test,
    avec la même logique clean/noisy qu'en train/val.
    """
    # Récupération des chemins "clean"
    clean_files = sorted(glob.glob(os.path.join(clean_dir, "*.wav")))
    # Création des chemins "noisy" correspondants
    noisy_files = []
    for cf in clean_files:
        filename = os.path.basename(cf)
        # Adapter le remplacement si nécessaire.
```

```
        # Ex: si vous avez *clean.wav et *noisy.wav :
        noisy_filename = filename.replace("clean", "noisy")
        noisy_files.append(os.path.join(noisy_dir, noisy_filename))

    # On crée le Dataset
    test_dataset = VoiceDataset(clean_files, noisy_files,␣
↪sample_rate=sample_rate)

    return test_dataset
```

[26]:
```
def prepare_test_loader(clean_dir, noisy_dir, batch_size=1, sample_rate=8000):
    """
    Crée un DataLoader pour l'ensemble de test,
    avec la même logique clean/noisy qu'en train/val.
    """
    # Récupération des chemins "clean"
    clean_files = sorted(glob.glob(os.path.join(clean_dir, "*.wav")))
    # Création des chemins "noisy" correspondants
    noisy_files = []
    for cf in clean_files:
        filename = os.path.basename(cf)
        # Adapter le remplacement si nécessaire.
        # Ex: si vous avez *clean.wav et *noisy.wav :
        noisy_filename = filename.replace("clean", "noisy")
        noisy_files.append(os.path.join(noisy_dir, noisy_filename))

    # On crée le Dataset
    test_dataset = VoiceDataset(clean_files, noisy_files,␣
↪sample_rate=sample_rate)
    # On crée le DataLoader
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    return test_loader
```

[27]:
```
def main_test(Model, path):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # Charger votre modèle
    model = Model
    model.load_state_dict(torch.load(path, map_location=device,␣
↪weights_only=True))
    model.to(device)

    test_clean_dir = "./voice_origin/test"
    # Répertoire contenant les .wav "bruités"
    test_noisy_dir = "./denoising/test"
```

```
    # Création du DataLoader de test
    test_loader = prepare_test_loader(test_clean_dir, test_noisy_dir,␣
↪batch_size=1, sample_rate=8000)

    print("Nombre de batchs en test :", len(test_loader))

    # Boucle de test (exemple)
    for noisy_batch, clean_batch in test_loader:
        print("noisy_batch shape :", noisy_batch.shape)
        print("clean_batch shape :", clean_batch.shape)
        break

    evaluate_metrics(model, test_loader, device, sr=8000)
```

[28]:
```
def show_and_listen_examples(model, test_dataset, device, desired_indices,␣
↪sr=8000):
    """
    model           : votre réseau de débruitage (ex: TasNet)
    test_dataset    : dataset qui renvoie (noisy_waveform, clean_waveform)
    device          : "cuda" ou "cpu"
    desired_indices: liste d'indices d'exemples à visualiser
    sr              : sample rate (16 kHz => PESQ mode 'wb', 8 kHz => PESQ mode␣
↪'nb')
    """

    model.eval()

    # Déterminer le mode PESQ en fonction de sr
    # - 8000 Hz => 'nb'
    # - 16000 Hz => 'wb'
    pesq_mode = 'wb' if sr == 16000 else 'nb'

    for i in desired_indices:
        print(f"\n-----------------------------------------------")
        print(f"Échantillon index = {i}")

        # 1) Récupération de l'exemple (noisy, clean)
        noisy_waveform, clean_waveform = test_dataset[i]
        # Dimensions attendues : (1, T). On veut (batch=1, 1, T)
        noisy_waveform = noisy_waveform.unsqueeze(0).to(device)    # (1, 1, T)
        clean_waveform = clean_waveform.unsqueeze(0).to(device)    # (1, 1, T)

        # 2) Inférence (débruitage)
        with torch.no_grad():
            estimated_waveform = model(noisy_waveform)  # (1, 1, T)

        # 3) Convertir en numpy pour écoute/tracé
```

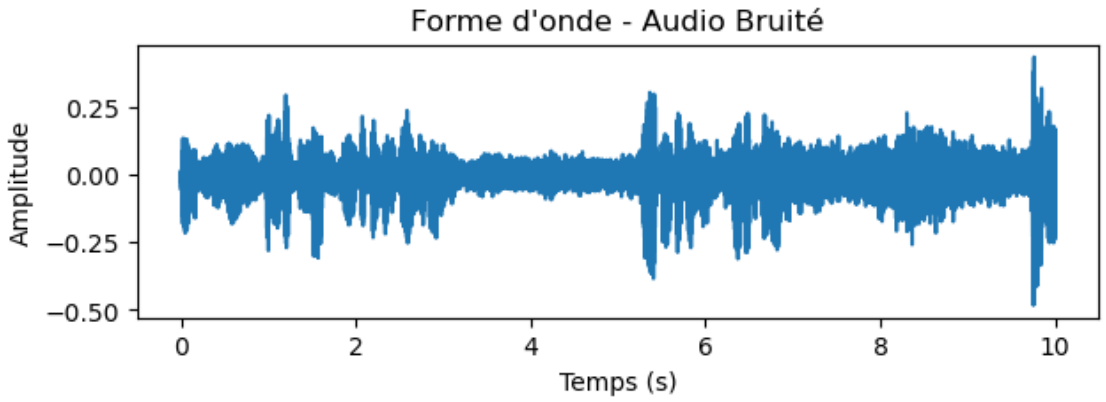
```
        # noisy_audio     = noisy_waveform[0, 0].cpu().numpy()     # shape (T,)
        # clean_audio     = clean_waveform[0, 0].cpu().numpy()     # shape (T,)
        # estimated_audio = estimated_waveform[0, 0].cpu().numpy() # shape (T,)

        # 3) Extraire les audios en tant que numpy arrays
        noisy_audio = noisy_waveform[0, 0].cpu().numpy()     # shape (T,)
        clean_audio = clean_waveform[0, 0].cpu().numpy()     # shape (T,)

        if isinstance(estimated_waveform, list):
            estimated_audio = estimated_waveform[0].squeeze().cpu().numpy()  #␣
↪shape (T,)  # Liste
        else:
            estimated_audio = estimated_waveform[0, 0].cpu().numpy()  # Tensor


        # 4) Écoute des échantillons
        print(f"Audio bruité:")
        ipd.display(ipd.Audio(noisy_audio, rate=sr))

        print(f"Audio propre (référence):")
        ipd.display(ipd.Audio(clean_audio, rate=sr))

        print(f"Audio débruité par {model.__class__.__name__}:")
        ipd.display(ipd.Audio(estimated_audio, rate=sr))

        # 5) Visualisation des formes d'onde
        # Noisy
        plt.figure(figsize=(7,2))
        plt.plot(np.linspace(0, len(noisy_audio)/sr, num=len(noisy_audio)),␣
↪noisy_audio)
        plt.title("Forme d'onde - Audio Bruité")
        plt.xlabel("Temps (s)")
        plt.ylabel("Amplitude")
        plt.show()

        # Clean
        plt.figure(figsize=(7,2))
        plt.plot(np.linspace(0, len(clean_audio)/sr, num=len(clean_audio)),␣
↪clean_audio)
        plt.title("Forme d'onde - Audio Propre (Clean)")
        plt.xlabel("Temps (s)")
        plt.ylabel("Amplitude")
        plt.show()

        # Débruité
        plt.figure(figsize=(7,2))
```

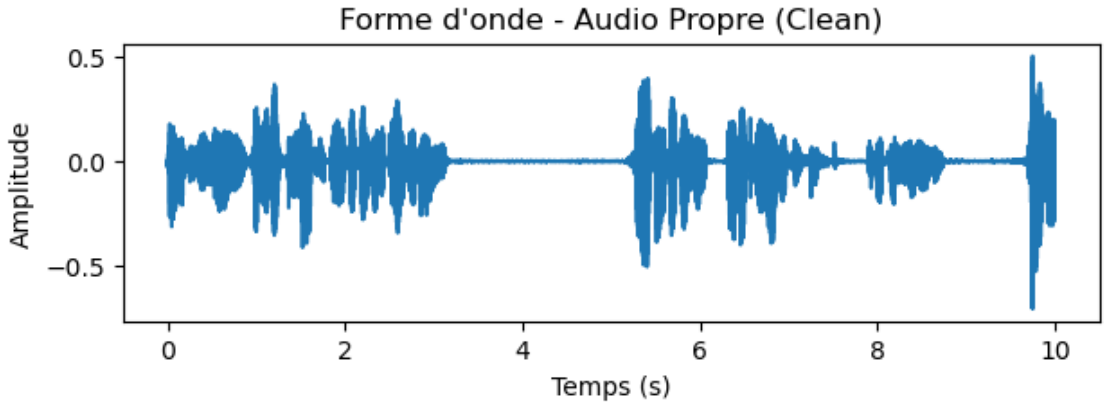
```
        plt.plot(np.linspace(0, len(estimated_audio)/sr,␣
↪num=len(estimated_audio)), estimated_audio)
        plt.title("Forme d'onde - Audio Débruité")
        plt.xlabel("Temps (s)")
        plt.ylabel("Amplitude")
        plt.show()

        # 6) Calcul du PESQ et du STOI
        # Ordre pesq => pesq(sr, ref, deg, mode)
        #    * ref = clean
        #    * deg = estimated
        pesq_score = pesq(sr, clean_audio, estimated_audio, pesq_mode)
        stoi_score = stoi(clean_audio, estimated_audio, sr, extended=False)

        print(f"PESQ Score : {pesq_score:.2f}")
        print(f"STOI Score : {stoi_score:.2f}")
```

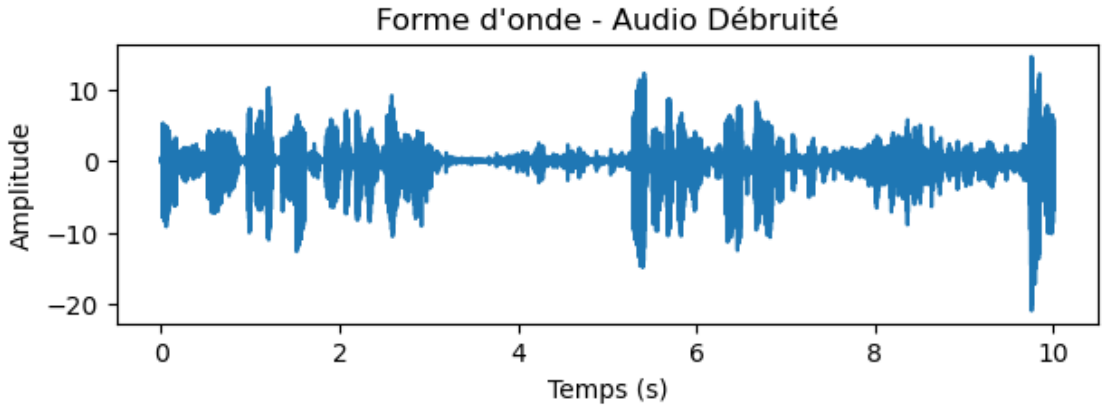
9.1 Test WaveUNet

[31]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = WaveUNet()
path = "./model/wave_u_net_denoising2.pt"
state_dict = torch.load(path, map_location=device, weights_only=False)
model.load_state_dict(state_dict)
model.to(device)

test_clean_dir = "./voice_origin/test"
test_noisy_dir = "./denoising/test"
test_dataset = prepare_test_dataset(test_clean_dir, test_noisy_dir,␣
↪batch_size=1, sample_rate=8000)

# desired_indices = [1,2,3]
desired_indices = random.sample(range(len(test_dataset)), 3)

show_and_listen_examples(model, test_dataset, device, desired_indices, sr=8000)
```

```
--------------------------------------------
Échantillon index = 149
Audio bruité:

<IPython.lib.display.Audio object>

Audio propre (référence):

<IPython.lib.display.Audio object>

Audio débruité par WaveUNet:

<IPython.lib.display.Audio object>
```

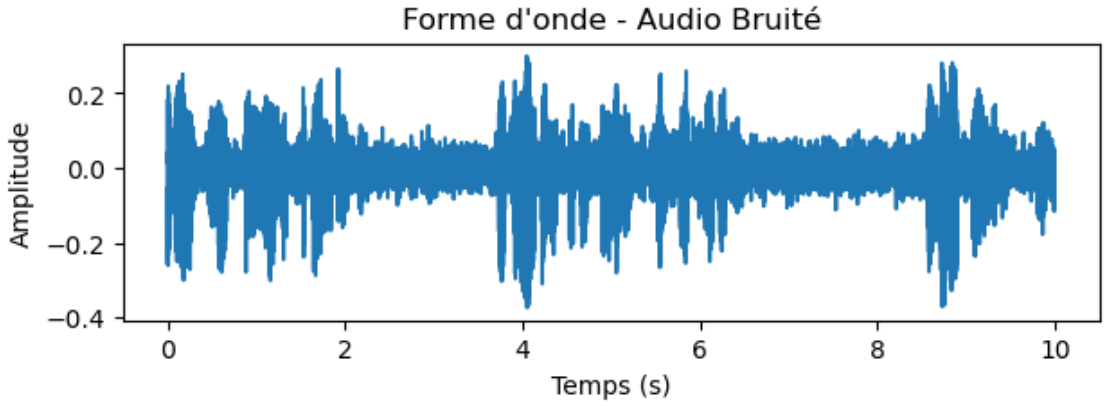


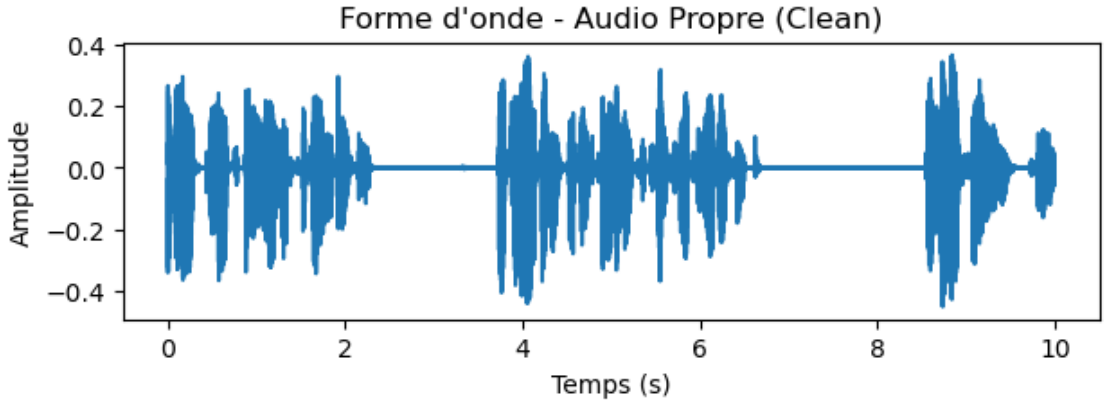


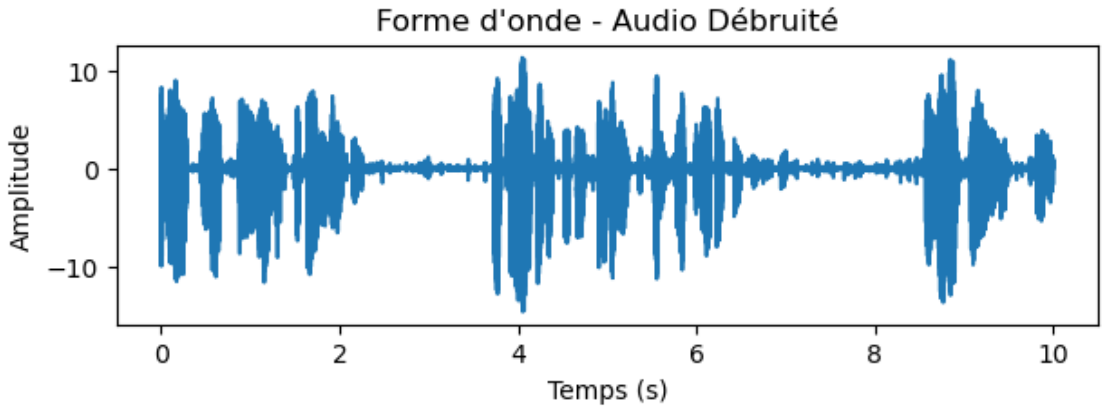



```
PESQ Score : 2.06
STOI Score : 0.82
```

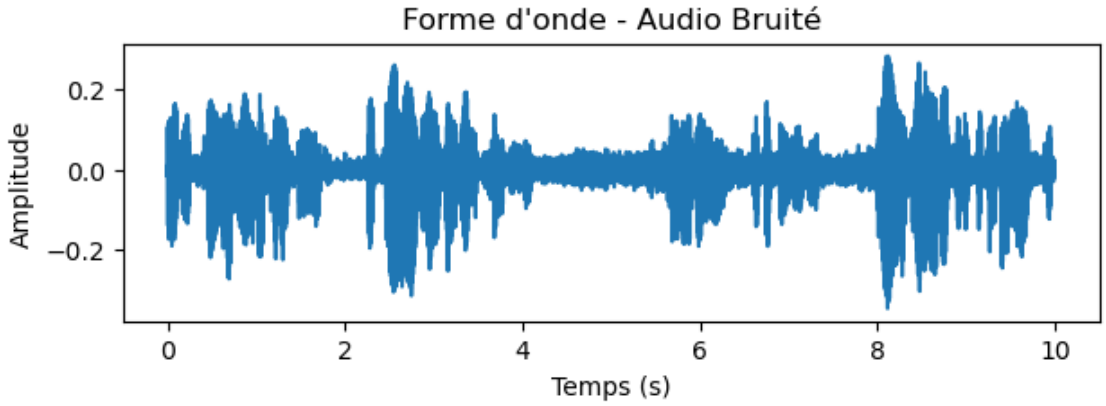
```
--------------------------------------------
Échantillon index = 398
Audio bruité:

<IPython.lib.display.Audio object>

Audio propre (référence):

<IPython.lib.display.Audio object>

Audio débruité par WaveUNet:

<IPython.lib.display.Audio object>
```

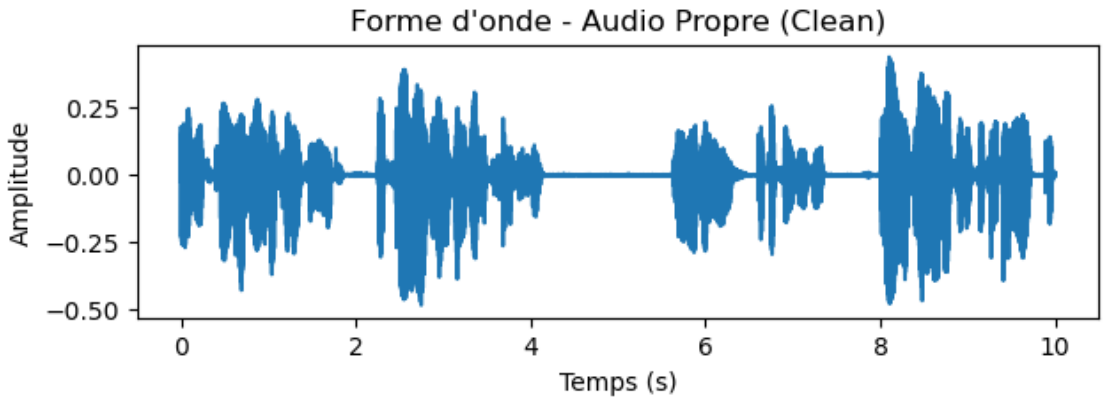


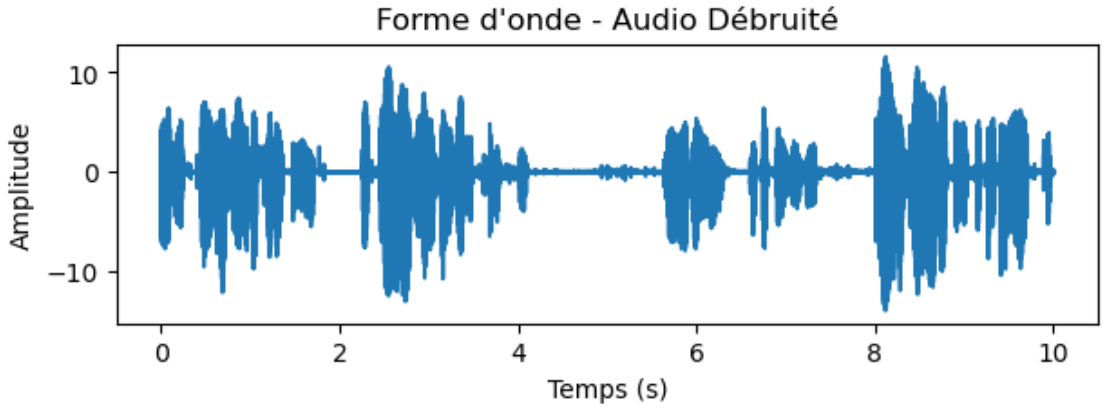





```
PESQ Score : 2.02
STOI Score : 0.84

-------------------------------------------
Échantillon index = 246
Audio bruité:

<IPython.lib.display.Audio object>

Audio propre (référence):

<IPython.lib.display.Audio object>

Audio débruité par WaveUNet:

<IPython.lib.display.Audio object>
```








```
PESQ Score : 2.05
STOI Score : 0.87
```

[25]:
```
main_test(model, path)
```

```
Nombre de batchs en test : 782
noisy_batch shape : torch.Size([1, 1, 80000])
clean_batch shape : torch.Size([1, 1, 80000])
0
10
20
30
40
50
60
70
80
90
```

100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
300
310
320
330
340
350
360
370
380
390
400
410
420
430
440
450
460
470
480
490
500
510
520
530
540
550
560
570

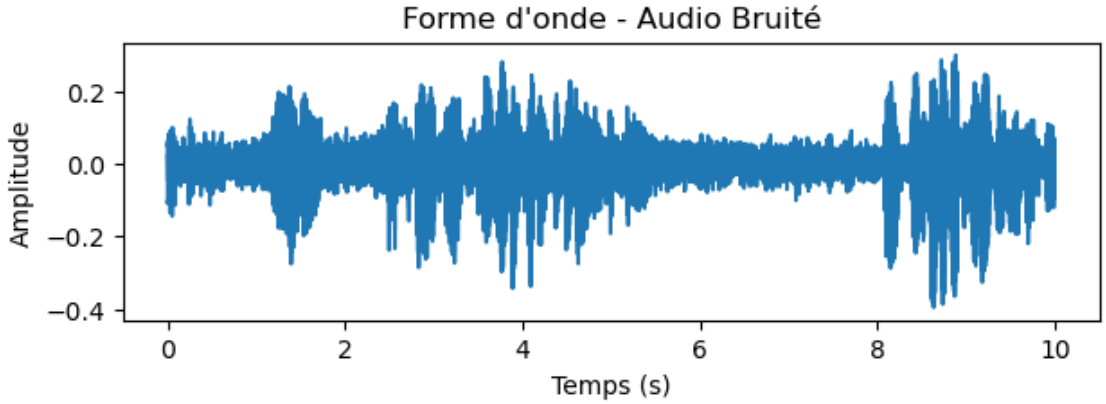
```
580
590
600
610
620
630
640
650
660
670
680
690
700
710
720
730
740
750
760
770
780
Mean PESQ: 2.664, Mean STOI: 0.889
```

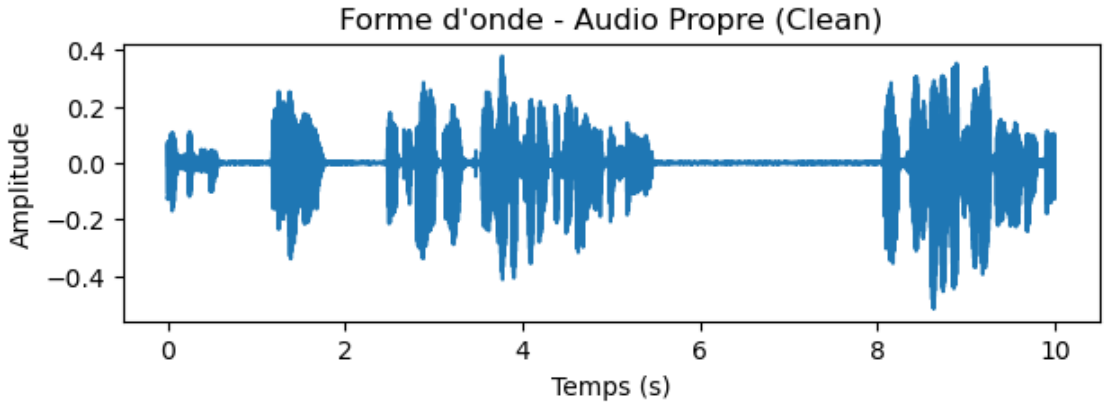
9.2 Test TasNet

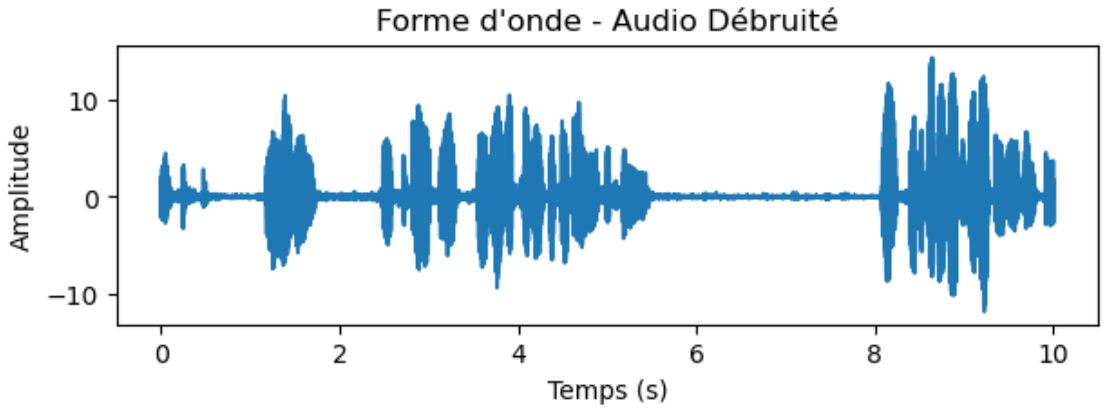
[101]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
path = "./model/tasnet3.pt"
model = TasNet()
state_dict = torch.load(path, map_location=device, weights_only=True)
model.load_state_dict(state_dict)
model.to(device)


test_clean_dir = "./voice_origin/test"
# Répertoire contenant les .wav "bruités"
test_noisy_dir = "./denoising/test"

# Création du DataLoader de test
test_dataset = prepare_test_dataset(test_clean_dir, test_noisy_dir,␣
 ↪batch_size=1, sample_rate=8000)

# desired_indices = [1,2,3]
desired_indices = random.sample(range(len(test_dataset)), 3)

show_and_listen_examples(model, test_dataset, device, desired_indices, sr=8000)
```

```
--------------------------------------------
```

```
Échantillon index = 568
Audio bruité:

<IPython.lib.display.Audio object>

Audio propre (référence):

<IPython.lib.display.Audio object>

Audio débruité par TasNet:

<IPython.lib.display.Audio object>
```







PESQ Score : 2.45
STOI Score : 0.90

Échantillon index = 613
Audio bruité:

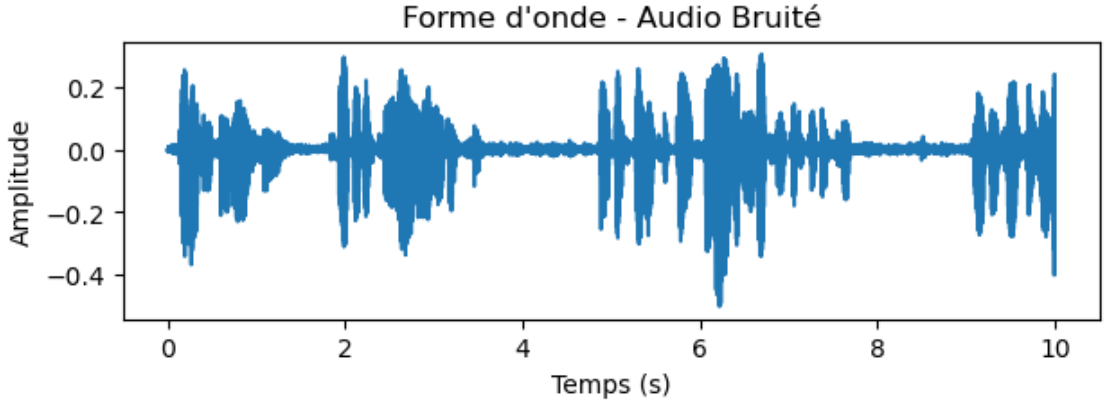
<IPython.lib.display.Audio object>

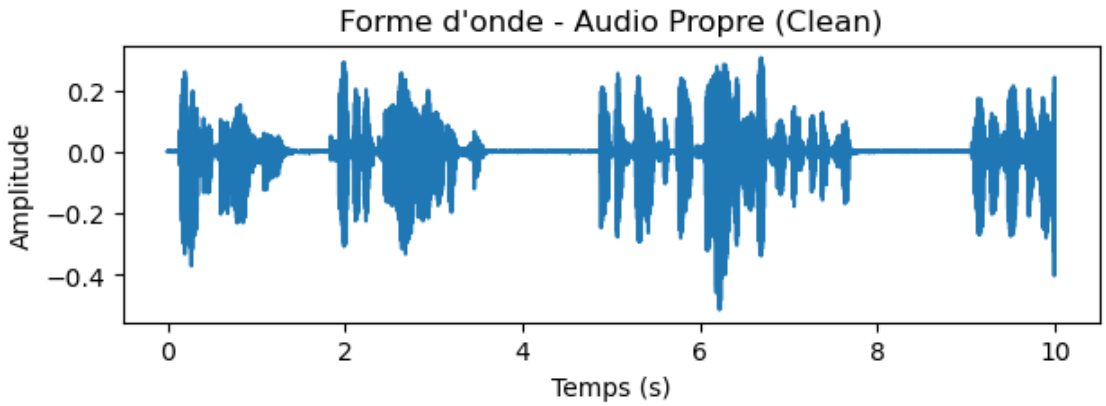
Audio propre (référence):

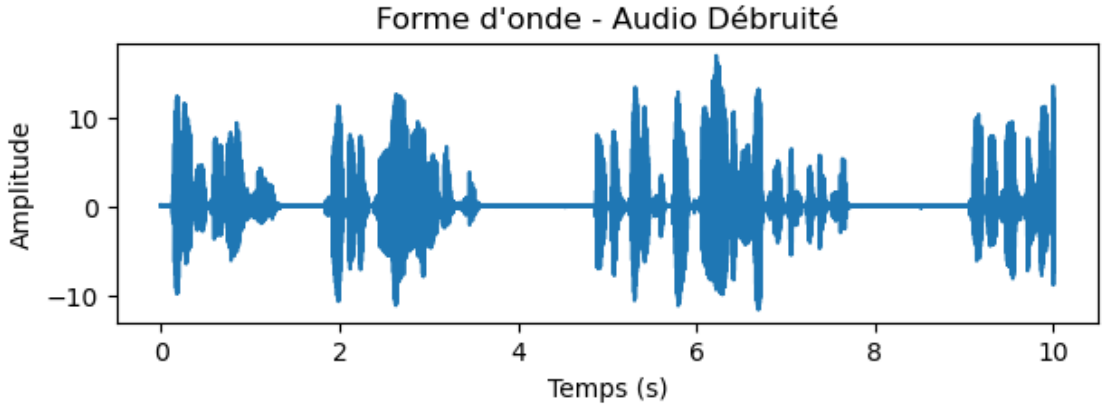
<IPython.lib.display.Audio object>

Audio débruité par TasNet:

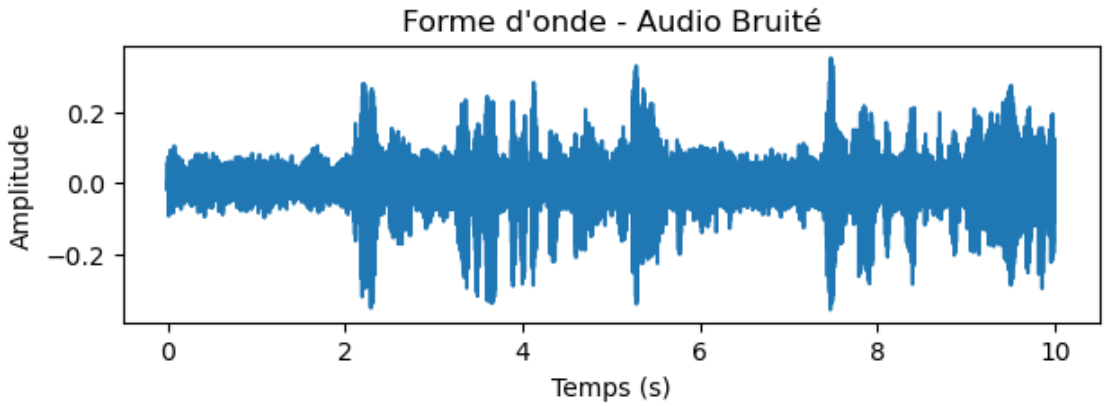
<IPython.lib.display.Audio object>

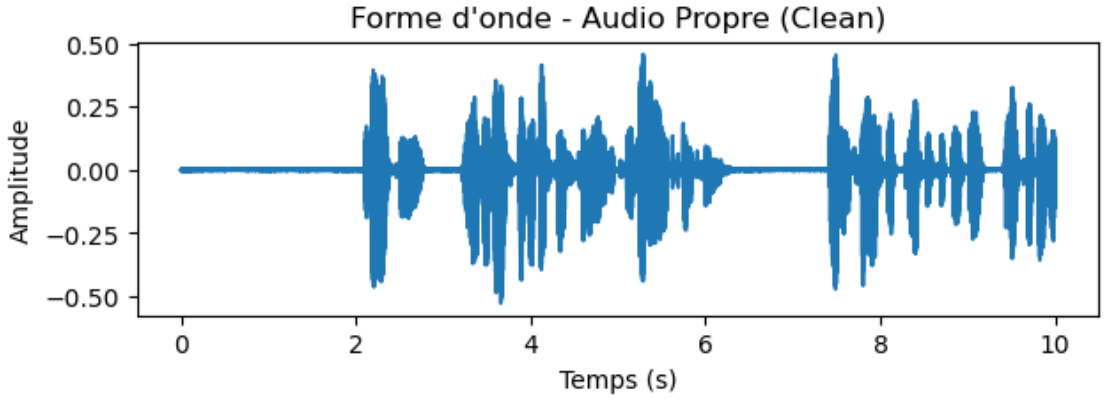


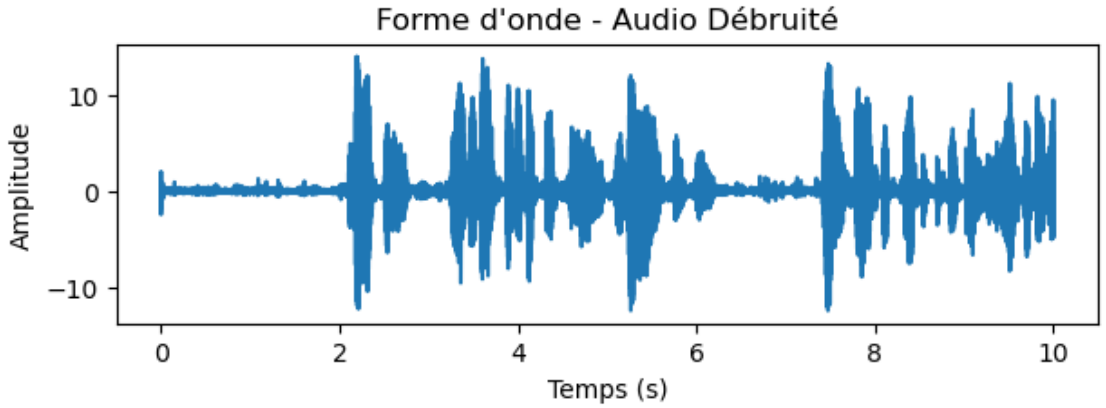





```
PESQ Score : 3.96
STOI Score : 0.97

--------------------------------------------
Échantillon index = 325
Audio bruité:

<IPython.lib.display.Audio object>

Audio propre (référence):

<IPython.lib.display.Audio object>

Audio débruité par TasNet:

<IPython.lib.display.Audio object>
```








```
PESQ Score : 1.64
STOI Score : 0.82
```

[102]:
```
main_test(model, path)
```

```
Nombre de batchs en test : 782
noisy_batch shape : torch.Size([1, 1, 80000])
clean_batch shape : torch.Size([1, 1, 80000])
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
300
310
320
330
340
350
360
370
380
390
400
410
420
```

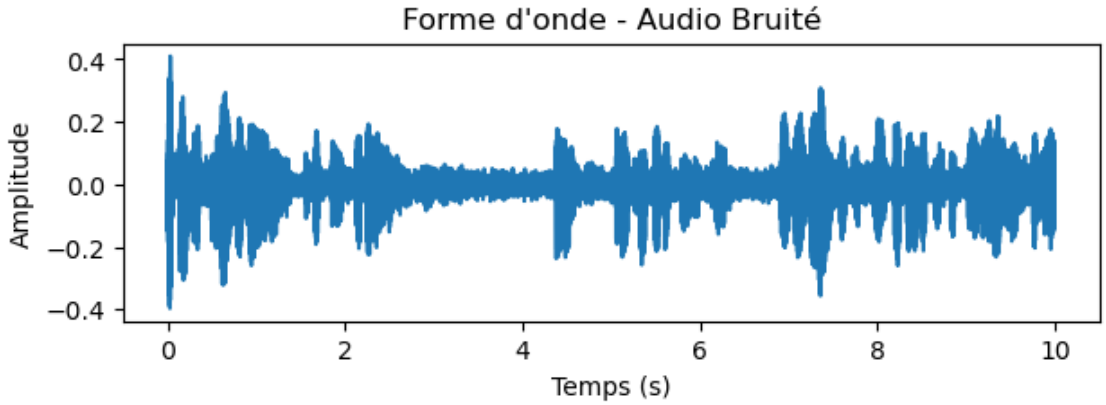
```
430
440
450
460
470
480
490
500
510
520
530
540
550
560
570
580
590
600
610
620
630
640
650
660
670
680
690
700
710
720
730
740
750
760
770
780
Mean PESQ: 2.783, Mean STOI: 0.915
```

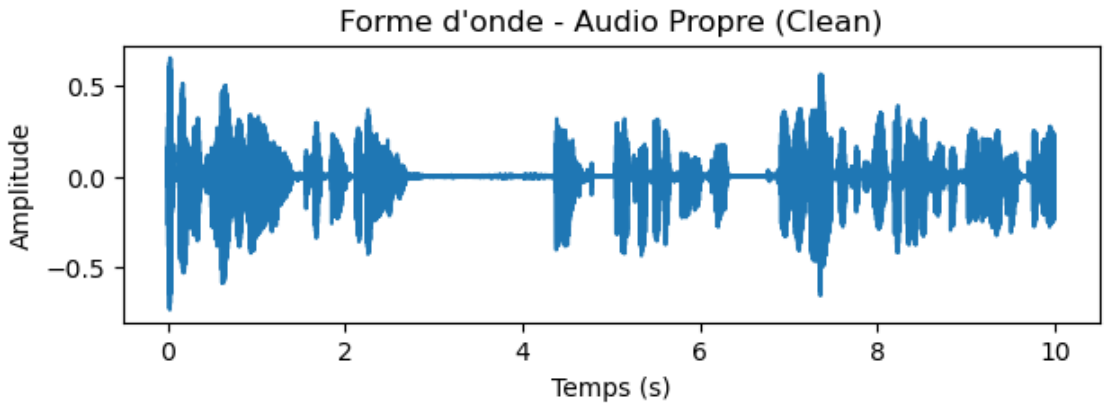
9.3 Test ConvTasNet

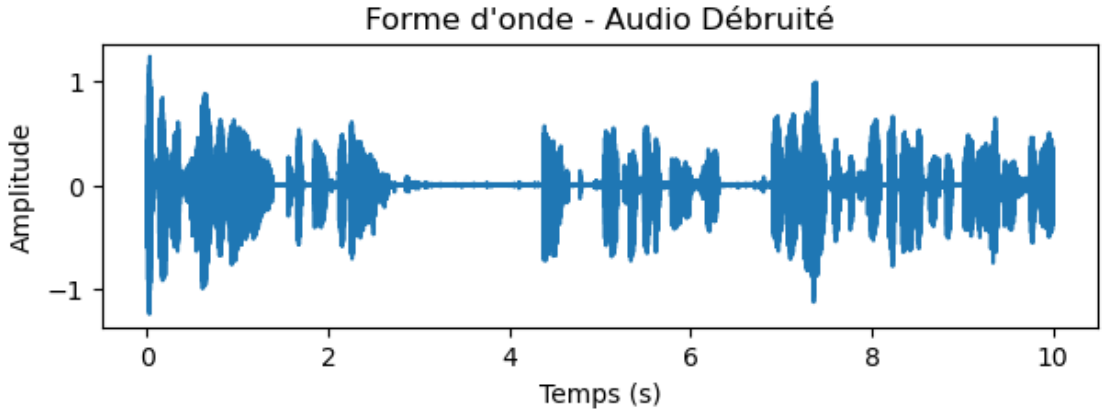
[30]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
path = "./model/tasnet1_extra.pt"
model = ConvTasNet()
state_dict = torch.load(path, map_location=device, weights_only=True)
model.load_state_dict(state_dict)
model.to(device)
```

```
test_clean_dir = "./voice_origin/test"
# Répertoire contenant les .wav "bruités"
test_noisy_dir = "./denoising/test"

# Création du DataLoader de test
test_dataset = prepare_test_dataset(test_clean_dir, test_noisy_dir,␣
 ↪batch_size=1, sample_rate=8000)

# desired_indices = [1,2,3]
desired_indices = random.sample(range(len(test_dataset)), 3)

show_and_listen_examples(model, test_dataset, device, desired_indices, sr=8000)
```

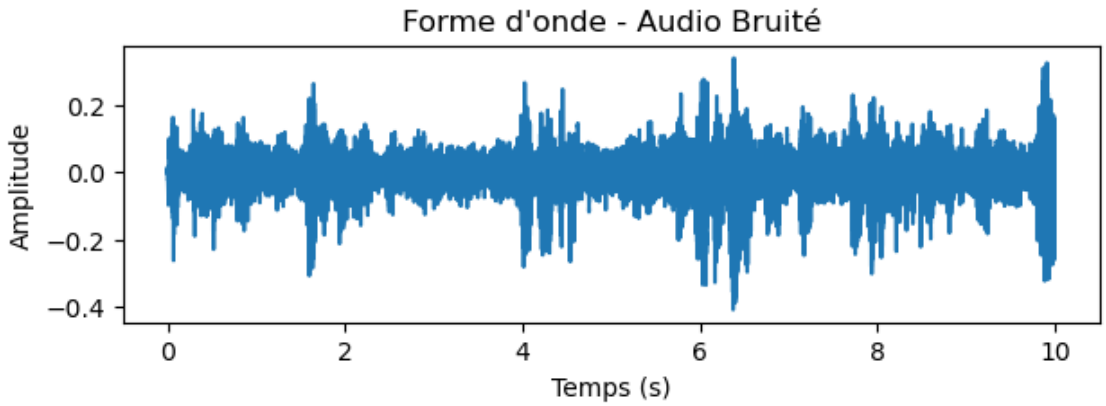
```
--------------------------------------------
Échantillon index = 482
Audio bruité:

<IPython.lib.display.Audio object>

Audio propre (référence):

<IPython.lib.display.Audio object>

Audio débruité par ConvTasNet:

<IPython.lib.display.Audio object>
```

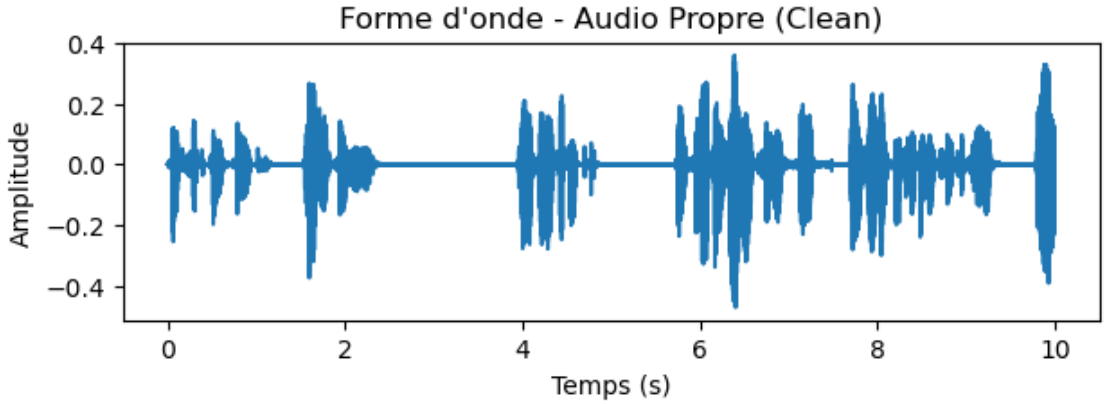


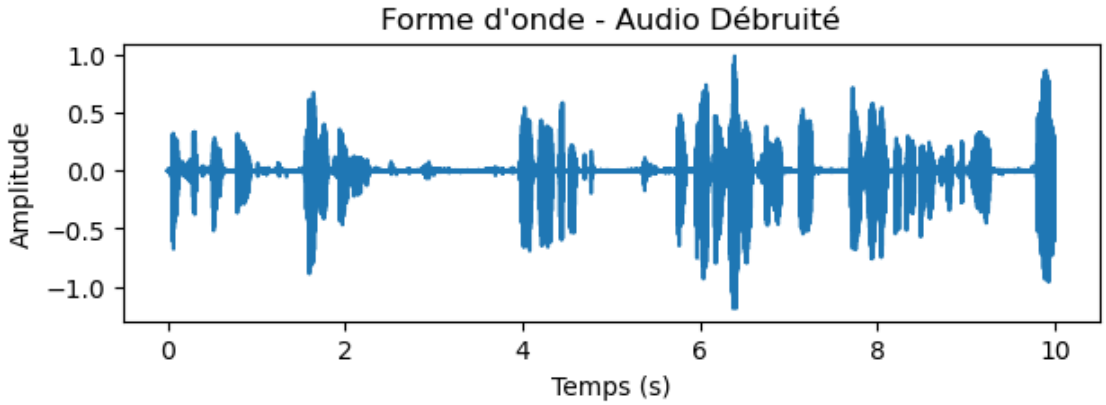





```
PESQ Score : 2.06
STOI Score : 0.83

------------------------------------------------
Échantillon index = 644
Audio bruité:

<IPython.lib.display.Audio object>

Audio propre (référence):

<IPython.lib.display.Audio object>

Audio débruité par ConvTasNet:

<IPython.lib.display.Audio object>
```

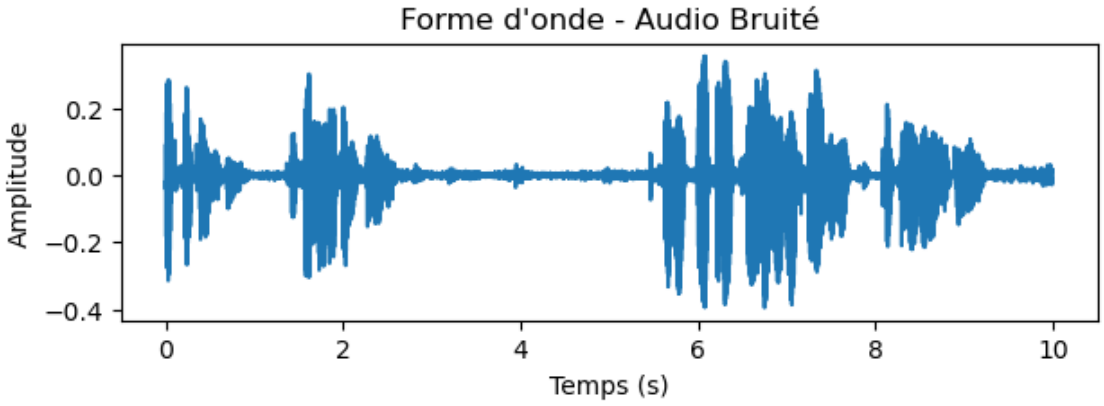


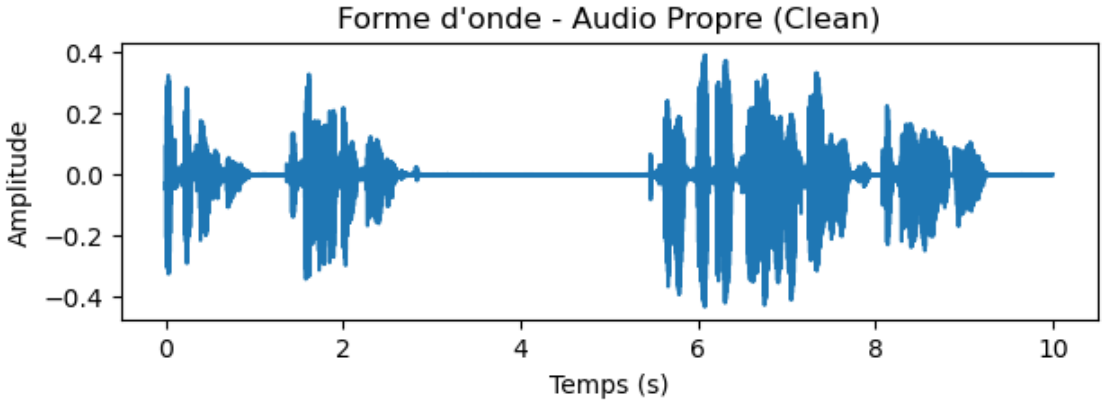


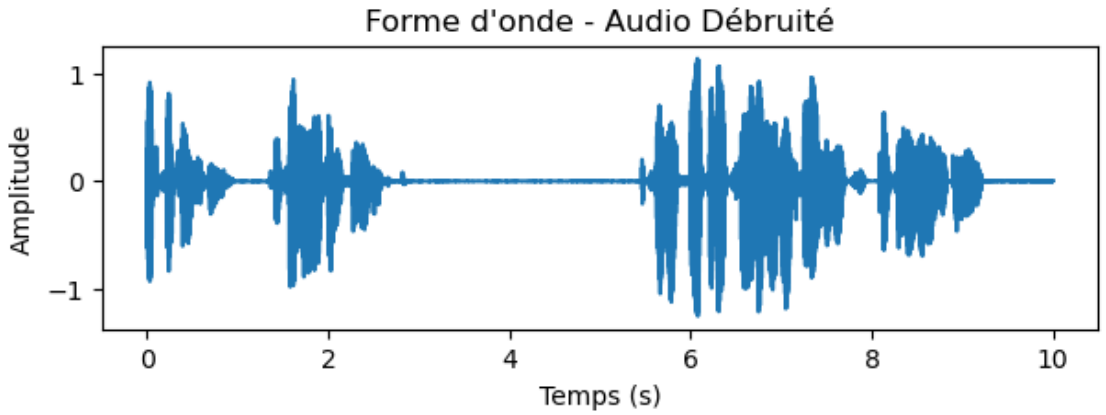



```
PESQ Score : 2.21
STOI Score : 0.89
```

```
--------------------------------------------
Échantillon index = 367
Audio bruité:

<IPython.lib.display.Audio object>

Audio propre (référence):

<IPython.lib.display.Audio object>

Audio débruité par ConvTasNet:

<IPython.lib.display.Audio object>
```








```
PESQ Score : 4.15
STOI Score : 1.00
```

[22]:
```
main_test(model, path)
```

```
Nombre de batchs en test : 782
noisy_batch shape : torch.Size([1, 1, 80000])
clean_batch shape : torch.Size([1, 1, 80000])
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
```

250
260
270
280
290
300
310
320
330
340
350
360
370
380
390
400
410
420
430
440
450
460
470
480
490
500
510
520
530
540
550
560
570
580
590
600
610
620
630
640
650
660
670
680
690
700
710
720

```
730
740
750
760
770
780
Mean PESQ: 2.892, Mean STOI: 0.931
```

10 Conclusion

[23]:
```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def count_n_param(model):
    return sum(p.numel() for p in model.parameters())

model = ConvTasNet().to(device)
total_params = count_n_param(model)
print("Nombre total de paramètres (pratique) :", total_params)

model = TasNet().to(device)
total_params = count_n_param(model)
print("Nombre total de paramètres (pratique) :", total_params)

model = WaveUNet().to(device)
total_params = count_n_param(model)
print("Nombre total de paramètres (pratique) :", total_params)
```

```
Nombre total de paramètres (pratique) : 1584128
Nombre total de paramètres (pratique) : 860928
Nombre total de paramètres (pratique) : 8713217
```

Dans ce projet, nous avons expérimenté trois architectures de débruitage basées sur des réseaux profonds : **ConvTasNet**, **TasNet** et **WaveUNet**. Nous avons évalué leurs performances grâce à : - Une fonction de coût personnalisée, la `si_snr_loss`, inspirée à la fois du cours et des articles de référence. - Deux métriques de qualité audio, **STOI** (Short-Time Objective Intelligibility) et **PESQ** (Perceptual Evaluation of Speech Quality), telles que présentées en cours.

Les résultats obtenus sont globalement satisfaisants, comme l'illustre le tableau ci-dessous (résumé des moyennes sur l'ensemble de test, en entraînant chaque modèle sur 10 *epochs*).

Modèle	STOI (moy.)	PESQ (moy.)	Nb. Paramètres	Temps d'exécution du train
ConvTasNet	0.919	2.692	1 584 128	36min 53s
TasNet	0.915	2.783	860 928	80min 49s
WaveUNet	0.889	2.664	8 713 217	1h 11min 23s

Et le dernier essai (sur 25 *epochs*):

Modèle	STOI (moy.)	PESQ (moy.)	Nb. Paramètres	Temps d'exécution du train
ConvTasNet	0.931	2.892	1 584 128	-

Malgré ces progrès, nous pensons qu'il reste une marge d'amélioration. Le temps et les capacités de calcul disponibles pour ce projet étaient limités, ce qui a restreint le champ de nos expérimentations (en particulier pour tester des configurations de plus grande taille ou affiner la recherche d'hyperparamètres).

Enfin, comme le deep learning est un domaine relativement nouveau pour nous, ce projet nous a permis d'appliquer et tester nos premières connaissances en la matière.