Create data store

```
AudioDataStoreTrain = audioDatastore("..\Spanish command voices\Voices without noise", ...
    'IncludeSubfolders',true, ...
    'FileExtensions','.wav', ...
    'LabelSource','foldernames')
```

```
AudioDataStoreTrain = 
  audioDatastore with properties:

                       Files: {
                              ' ...\Spanish command voices\Voices without noise\0\0_ALEX.wav';
                              ' ...\Spanish command voices\Voices without noise\0\0_COKE.wav';
                              ' ...\Spanish command voices\Voices without noise\0\0_FABIAN.wav'
                               ... and 848 more
                              }
                     Folders: {
                              ' ...\Paper\Pre-datascience\Spanish command voices\Voices without noise'
                              }
                      Labels: [0; 0; 0 ... and 848 more categorical]
    AlternateFileSystemRoots: {}
              OutputDataType: 'double'
      SupportedOutputFormats: ["wav"    "flac"    "ogg"    "mp4"    "m4a"]
         DefaultOutputFormat: "wav"
```

Read a lisent to first file

```
[cleanAudio, info] = read(AudioDataStoreTrain)
```

```
cleanAudio = 32880×1
    0.0000
   -0.0000
         0
    0.0000
   -0.0000
   -0.0000
    0.0000
    0.0000
   -0.0000
    0.0000
      ⋮

info = struct with fields:
    SampleRate: 48000
      FileName: 'G:\Mi unidad\__Usach__\BIGDATA\Paper\Pre-datascience\Spanish command voices\Voices without noise\0
         Label: 0
```

```
sound(cleanAudio, info.SampleRate)
```

Sample rate converter

```
% Create a sampling converter object to reduce the computational burden
Fs = info.SampleRate%8e3; % New sampling frequency. sampling theorem and frequency range in hur
```

```
Fs = 48000
```

```
src = dsp.SampleRateConverter("Bandwidth", 7980, "InputSampleRate", info.SampleRate, "OutputSar
```

```
src = 
  dsp.SampleRateConverter with properties:

        InputSampleRate: 48000
       OutputSampleRate: 48000
    OutputRateTolerance: 0
              Bandwidth: 7980
    StopbandAttenuation: 80
```

Adding stationary noise from a bath (air stripper) SNR=0[dB]

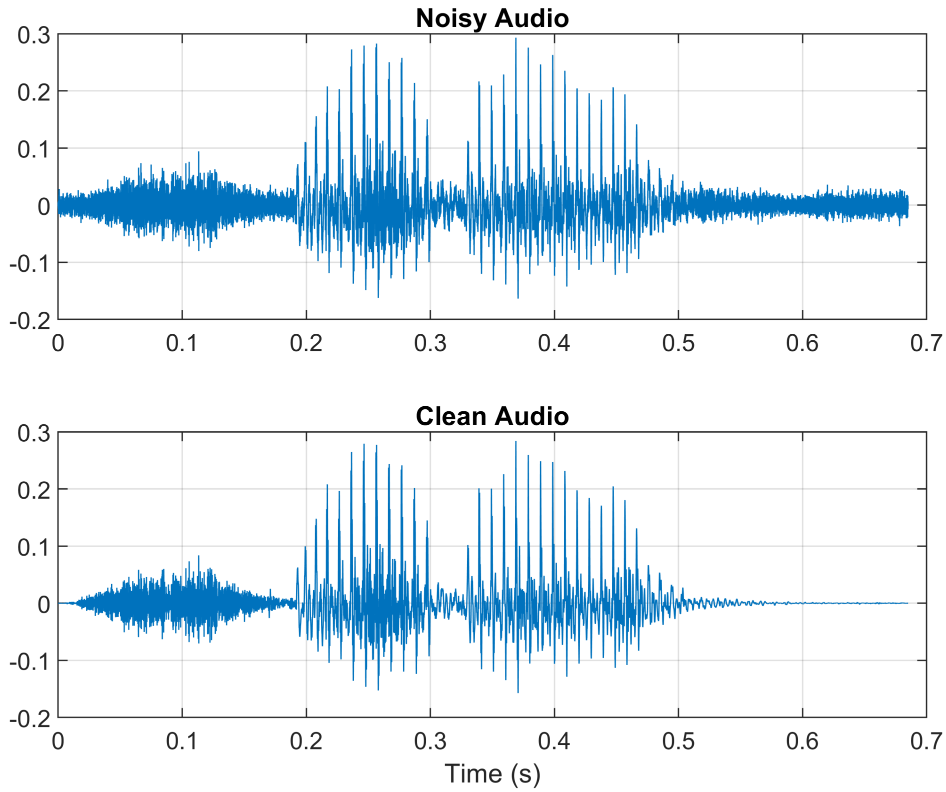
```
noise           = audioread("..\Spanish command voices\stationary noises in a bathroom\SHOWER W
randind         = randi(numel(noise) - numel(cleanAudio) , [1,1]);
noiseSegment    = noise(randind : randind + numel(cleanAudio)-1);
noisePower      = sum(noiseSegment.^2);
cleanPower      = sum(cleanAudio.^2);
noiseSegment    = noiseSegment .* sqrt(cleanPower/noisePower);
noisyAudio      = cleanAudio + .25*noiseSegment;
```

listen to noisy data

```
pause(1)
sound(noisyAudio, info.SampleRate)
```

plote noisy and clean audio

```
t = 1/Fs * (0:numel(cleanAudio)-1);
subplot(211)
plot(t,noisyAudio)
title("Noisy Audio")
grid on
subplot(212)
plot(t,cleanAudio)
title("Clean Audio")
xlabel("Time (s)")
grid on
```



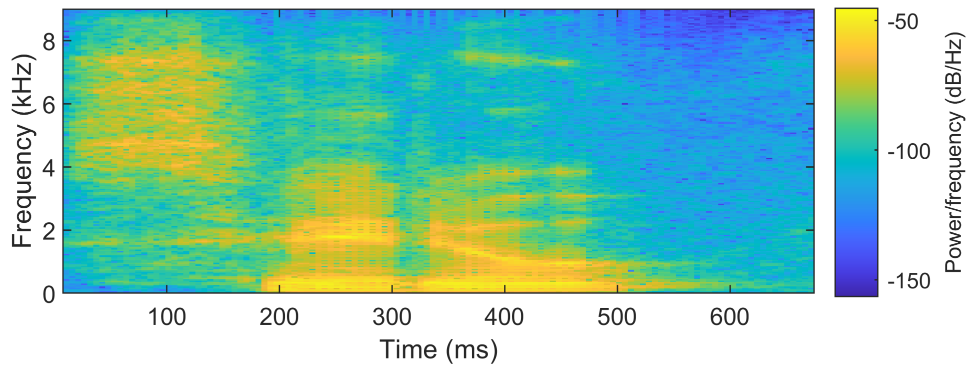
Generate magnitude STFT vectors

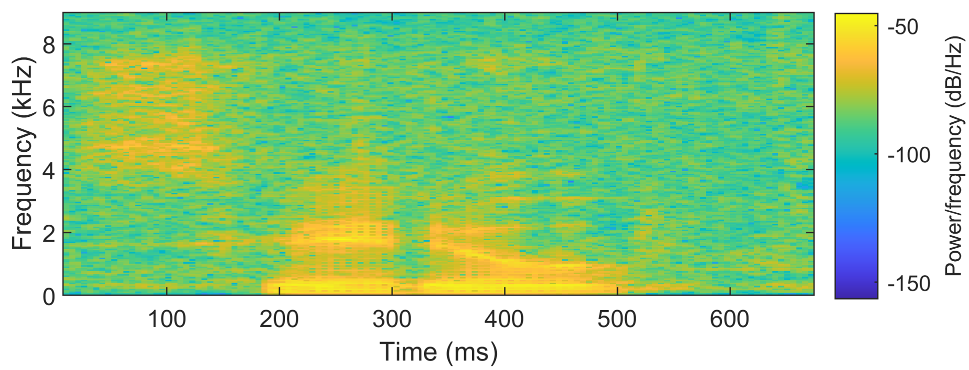
The patameters needed to do the spectograms.

```
windowLength    = 2^10;
win             = hamming(windowLength, "periodic");
Overlap         = round(.75 * windowLength);
FFTLength       = windowLength;
NumFeatures     = FFTLength/2 + 1;
NumSegments     = 8;
```

Step 1: Calculate the spectograms

```
cleanSTFT = spectrogram(cleanAudio, win, Overlap, FFTLength, Fs);
cleanSTFT = abs(cleanSTFT);
noisySTFT = spectrogram(noisyAudio, win, Overlap, FFTLength,Fs);
noisySTFT = abs(noisySTFT);
figure
subplot(211)
spectrogram(cleanAudio, win, Overlap, FFTLength, Fs, 'yaxis' )
ylim([0 9])
subplot(212)
spectrogram(noisyAudio, win, Overlap, FFTLength, Fs, 'yaxis')
ylim([0 9])
```

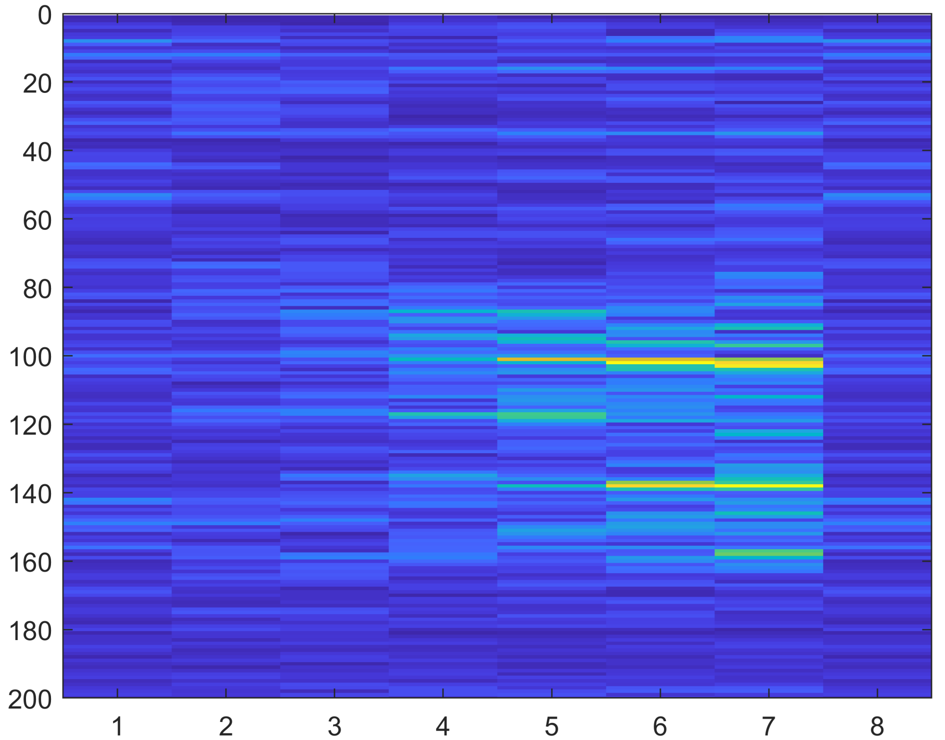




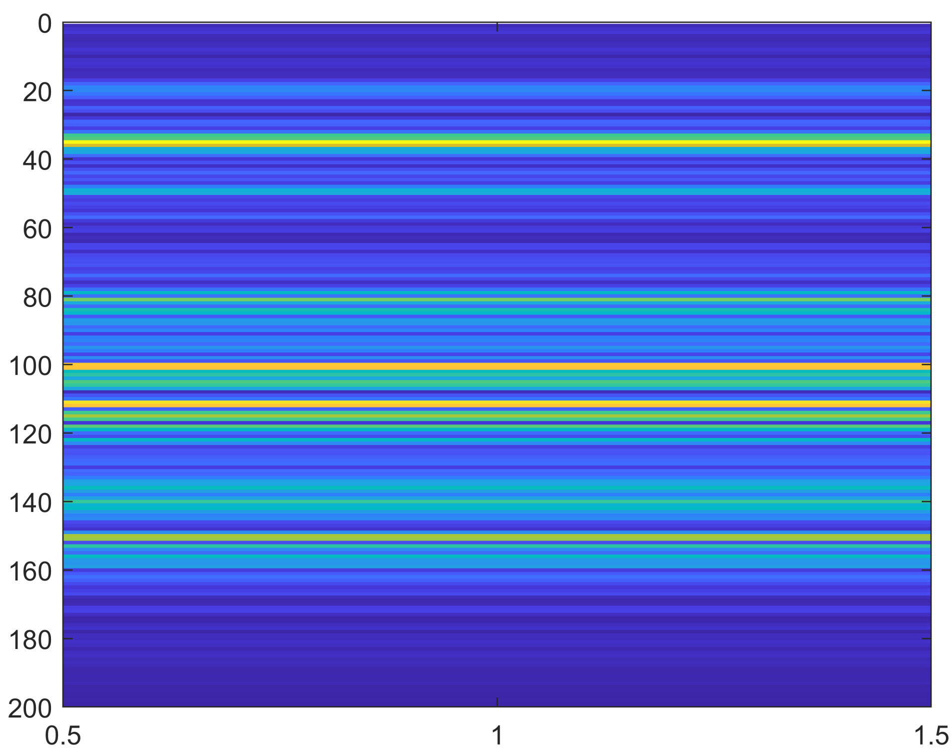
Step 2: Generate 8-segment training predictor signals

```
noisySTFT = [noisySTFT(:,1:NumSegments-1) noisySTFT];
STFTSegments = zeros(NumFeatures, NumSegments , size(noisySTFT,2) - NumSegments + 1);
for index = 1:size(noisySTFT,2) - NumSegments + 1
   STFTSegments(:,:,index) = (noisySTFT(:,index:index+NumSegments-1));
end

targets = cleanSTFT;
size(targets);
predictors = STFTSegments;
size(predictors);
figure, imagesc(STFTSegments(:,:,1)); ylim([0 200])
```




```
figure, imagesc(targets(:,1)); ylim([0 200])
```



Work with the langer data set

```
reset(AudioDataStoreTrain)
T = tall(AudioDataStoreTrain)
```

```
T =

  M×1 tall cell array

    {32880×1 double}
    {26207×1 double}
    {19779×1 double}
    {29025×1 double}
    {31938×1 double}
    {28001×1 double}
    {29215×1 double}
    {33040×1 double}
       :        :
       :        :
```

Extract target and predictor from tall table

```
[targets,predictors] = cellfun(@(x)HelperGenerateSpeechDenoisingFeatures(cleanAudio,noise,src)
% [targets,predictors] = gather(targets,predictors);
% addAttachedFiles (gcp (), 'HelperGenerateSpeechDenoisingFeatures' )
```

Normalize and reshape the data

```
% predictors      = cat(3, predictors{:});
% targets         = cat(2, targets{:});
% noisyMean       = mean(predictors(:));
% noisyStd        = std(predictors(:));
% predictors(:)   = (predictors(:)-noisyMean)/noisyStd;
% cleanMean       = mean(targets(:));
% cleanStd        = std(targets(:));
% targets(:)      = (targets(:)-cleanMean)/cleanStd;
```

Reshape predictors and targets to the dimensions expected by the deep learning network.

```
% predictors  = reshape(predictors,size(predictors,1),size(predictors,2),1,size(predictors,3))
% targets     = reshape(targets,1,1,size(targets,1),size(targets,2));
```

Split into Training and Validation data

```
% inds                  = randperm(size(predictors,4));
% L                     = round(0.99 * size(predictors,4));
% trainPredictors       = predictors(:,:,:,inds(1:L));
% trainTargets          = targets(:,:,:,inds(1:L));
% validatePredictors    = predictors(:,:,:,inds(L+1:end));
% validateTargets       = targets(:,:,:,inds(L+1:end));
```

Speech Denoising with Convolutional Layers