**AUDIO SIGNAL PROCESSING - RRR (Restoration, Recognition, ReGeneration)**

### **INTRODUCTION**

The proposed project focuses on implementing advanced signal-processing algorithms for audio data. This project aims to address critical aspects of audio data, including classification, restoration, and generation. By leveraging a substantial music dataset, we aim to achieve a comprehensive understanding of the world of audio signals and the methods to process them for various research and industry use cases.

### **OBJECTIVES**

1. **Identification/Classification of Audio Signals:** Implement advanced signal processing and/or machine learning techniques to classify audio signals into distinct categories, such as genres, instruments, or vocal types.
2. **Restoration of Audio Signals:** Explore techniques for restoring audio signals that may have been affected by noise, distortion, or degradation. Apply signal processing methods to enhance the quality and fidelity of audio recordings, ensuring a pristine listening experience.
3. **Generation of Audio Signals:** Develop algorithms for the generation of new audio signals, which can include music compositions or sound effects. Utilize generative models and signal processing to create high-quality, original audio signals.

### **MOTIVATION**

Audio data plays a significant role in our daily lives, from entertainment and communication to critical applications like healthcare and security. The ability to classify, restore, and generate Music signals has a wide range of real-world applications, including music recommendation systems, audio restoration tools, and creative content generation. Additionally, this project provides a valuable opportunity to enhance our understanding of signal processing and machine learning techniques.

### **PLAN OF WORK**

1. Data Acquisition: Music Dataset
2. Data Format: Spectrogram/Time Series Representation
3. Preprocessing: Low/High pass filters, Normalization, Masking
4. Restoration: Kalman filter, AutoRegressive Models, SVD+NMF Imputation
5. Recognition: Gaussian classifier, HMM, ANN
6. Regeneration:
7. Exploring other algorithms/techniques if time permits:

* Spectral Clustering
* PCA aka Convolution
* K-medoids clustering
* PatchMatch: A Randomized Correspondence Algorithm

### **REFERENCES**

<http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf>

<https://www.cs.ubc.ca/~murphyk/Bayes/rabiner.pdf>

<https://gfx.cs.princeton.edu/pubs/Barnes_2009_PAR/index.php>

<http://www-sigproc.eng.cam.ac.uk/Main/SJGSpringer>

<https://mi.eng.cam.ac.uk/~mjfg/mjfg_NOW.pdf>

**Rough Work**

**Dataset**

1. **Gtzan :** [**https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification**](https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification)
2. **Million Song Dataset:** [**https://www.kaggle.com/datasets/gauravduttakiit/million-song-dataset**](https://www.kaggle.com/datasets/gauravduttakiit/million-song-dataset)
3. **Magnatagatune :** [**https://paperswithcode.com/dataset/magnatagatune**](https://paperswithcode.com/dataset/magnatagatune)

def spectrogram\_features(x, size):

# Get frames

lenl = len(x)//size

# print("no of values : ",len(x))

# print("no of frames : ",lenl)

x = reshape(x[:lenl\*size], (lenl,-1)).T

# Window and FFT them

w = hanning(size)[:,None]

FFT = rfft(w\*x, axis=0)

# Return log magnitude

return log(abs(FFT) + 1e-8)

# gaussian\_classifier for each class

def gaussian\_classifier(x, gauss=None, diagonal=False):

# If no model is provided, then train

if gauss is None:

# class mean

mean = np.mean(x, axis=1, keepdims=True)

# print("mean shape ", mean.shape )

# class covariance

covariance = ((x-mean) @ (x-mean).T) / (x.shape[1]-1)

if diagonal:

covariance = np.diag(np.diag(covariance))

return {"mean": mean, "icov": np.linalg.inv(covariance)}

# If a model is provided just compute log likelihoods

else:

mean = gauss["mean"]

cov = gauss["icov"]

# print("mean shape ", mean.shape )

# print("cov shape ", cov.shape )

# print("x shape ", x.shape )

x\_m = x - mean

delta = sum(x\_m \* (cov @ x\_m), axis=0)

# Compute class likelihood P(x|ωi)

return -.5\*(-np.log(np.linalg.det(cov)) + delta + x.shape[0] \* np.log(2 \* np.pi))

```  
from hmmlearn import hmm

import numpy as np

# Define the HMM model

model = hmm.MultinomialHMM(n\_components=2) # 2 hidden states

# Training data (observable outcomes)

X = np.array([[0, 1, 0, 1, 1, 0, 1, 0], # Example sequence 1 (0: walking, 1: staying indoors)

[1, 0, 1, 1, 0, 0, 1, 1]]) # Example sequence 2

# Fit the model with the training data

model.fit(X)

# Predict hidden states

hidden\_states = model.predict(X)

# Print the predicted hidden states for each sequence

for i in range(len(X)):

print("Sequence {}: Hidden States - {}".format(i + 1, hidden\_states))

# Predict the next observable outcomes given the current hidden states

predicted\_outcomes, \_ = model.sample(n\_samples=5)

print("Predicted Outcomes: {}".format(predicted\_outcomes))

```

Code n papers for references:

<https://sthalles.github.io/practical-deep-learning-audio-denoising/>