Speech Understanding (CSL7770)
Assignment 1 - Question 2

Soham Niraj Deshmukh (B21EE067)

The code can be found here: GitHub Repository

1 OBJECTIVES

This task involves experimenting with spectrograms and windowing techniques using the UrbanSound8k dataset. The objective is to -

a. Implement three windowing methods—Hann, Hamming, and Rectangular—while generating spectrograms using the Short-Time Fourier Transform (STFT). A comparative visual and analytical evaluation of the spectrograms is required to assess the correctness of the windowing process. Deep learning-based classifiers such as CNN and machine learning classifiers such as SVMs, Decision Tree, Random Forest, and K-Nearest Neighbors (KNN) are to be trained on features extracted from the spectrograms, and their performance evaluated across different windowing techniques.

b. Generate spectrograms for 4 songs of 4 different genres and analyze their characteristics to derive inferences.

2 DATASET

The dataset used for Part A of this question is 'UltraSound8K'. It consists of **10 folds** of audio sounds. Each fold contains multiple audio files (.wav) format.
Class labels are assigned from a set of [*dog bark, children playing, car horn, air conditioner, street music, gun shot, siren, engine idling, jackhammer, drilling*] i.e. 10 classes.

Class Label	Class Name
0	Air Conditioner
1	Car Horn
2	Children Playing
3	Dog Bark
4	Drilling
5	Engine Idling
6	Gun Shot
7	Jackhammer
8	Siren
9	Street Music

Table 1: Mapping of Labels to Class Names

There are a total of **8732** audio files. The dataset also provides a metadata CSV file that provides class labels for each audio file.

3 TASK A

3.1 Importing and creating dataset

The first and foremost task is importing the dataset. Upon importing the files in .wav format, we first read the files using ***librosa*** library. ***Librosa*** converts the audio files from audio (.mp3, .wav, etc) formats to a vector. The quality of importing can be decided using the *sampling rate (sr)*.

Corresponding to the name of the audio file, it's class names is also extracted from the metadata CSV file. The extracted vectors of audio files are stored as **X**. Their corresponding class names are mapped to the above labels (according to Table 1) and stored as **Y**.

3.2 Implementation of Windowing Techniques

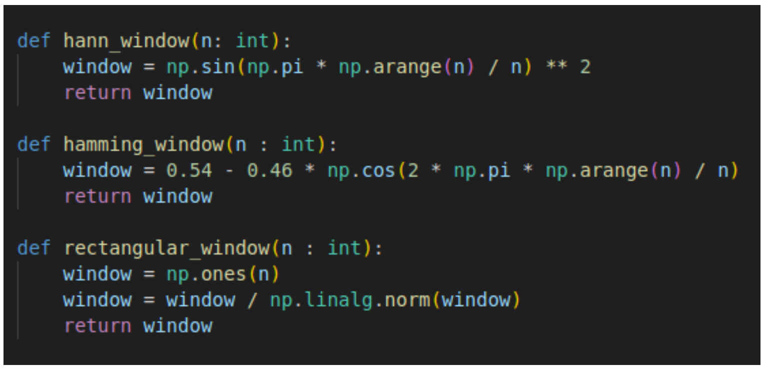
I have implemented all the 3 windowing techniques from scratch - ***Hann, Hamming, Rectangular*** windows. The formulas of each of these windows are given in equations below.

$$w(n) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right), \quad 0 \le n \le N-1 \tag{1}$$

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \le n \le N-1 \tag{2}$$

$$w(n) = \begin{cases} 1, & 0 \le n \le N-1 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Equations (1), (2), and (3) represent the Hann, Hamming, and Rectangular windows, respectively. Based on these 3 formulas, I implemented these windows using *numpy*. The implementations are as follows -



```python
def hann_window(n: int):
    window = np.sin(np.pi * np.arange(n) / n) ** 2
    return window

def hamming_window(n : int):
    window = 0.54 - 0.46 * np.cos(2 * np.pi * np.arange(n) / n)
    return window

def rectangular_window(n : int):
    window = np.ones(n)
    window = window / np.linalg.norm(window)
    return window
```

Fig. 1: NumPy implementations of Windowing Techniques

3.3 Applying Windowing and Short-Time Fourier Transform

For every audio in the dataset, I applied all 3 windowing techniques - Hann, Hamming and Rectangular. The resulting output is called a ***'windowed signal'***.

Next, I applied Short-Time Fourier Transform (STFT) on these ***'windowed signals'***. For this, I used *'stft'* function from *scipy.signal* package. It takes the following parameters as input -

– Input vector signal
– Window
– Overlap size (noverlap)
– Length of FFT (nfft)
– Padded (boolean)

The values of ***nfft*** and ***noverlap*** are usually kept in powers of 2, for eg., ***nfft=256, overlap=128***.

The output of this function can be complex (real+imaginary components). So, in order to create the spectrogram, we consider the magnitude of the frequency domain component. This is done by taking the **absolute** value of the spectrogram points.

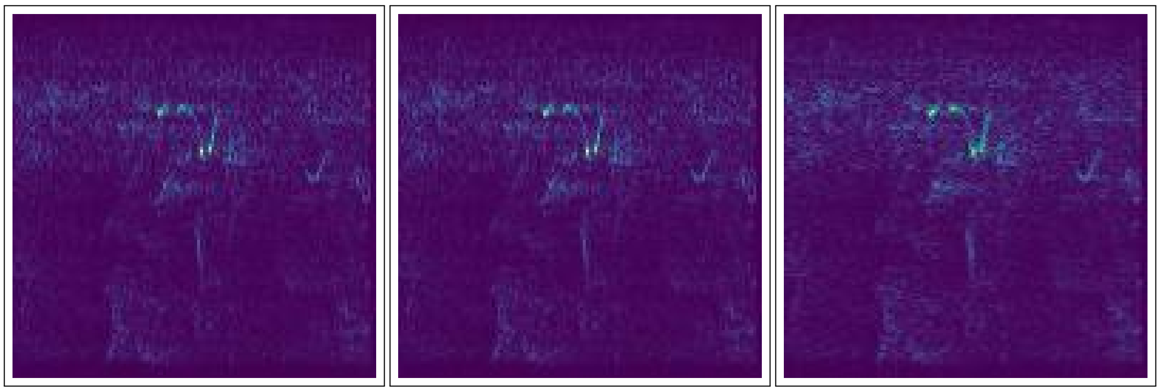
$$\text{spectrogram} = \text{np.abs(spectrogram)} \tag{4}$$

In order to limit the spectrogram to equivalent ranges, we use **padding** and **truncation** - padding increases the length of the spectrogram by adding null components whereas truncation slices off the additional components of the spectrogram. This enables us to get uniform spectrograms across audio inputs of varying lengths. The ***fixed length*** for this task is calculated using the following formula.

$$\text{fixed\_shape} = \left(\frac{\text{window\_size}}{2} + 1, 128\right) \tag{5}$$

Corresponding to the above mentioned values of 'window_size', the 'fixed_shape' is obtained as (129,128). Spectrograms are stored in a folder *'spectrograms'* based on the type of the window used. The labels are stored in a *'labels.csv'* file.

Some examples of spectrograms of signals are given below.



(a) Hann (b) Hamming (c) Rectangular

Fig. 2: Windowed Spectrograms using different windowing techniques

We can notice that there appears to be very slight modification in the spectrograms when viewed by naked eyes.

3.4 Defining CNN Architecture

For classification of audio signals using the spectrograms obtained, I created a CNN-based architecture. the structure of the CNN is as follows.

```
CNNClassifier(
(conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(fc1): Linear(in_features=10368, out_features=512, bias=True)
(fc2): Linear(in_features=512, out_features=10, bias=True)
)
```

The dataset of all 3 windows are split into ***train*** and ***val*** and loaded into DataLoaders. Three seperate CNN instances are then trained on each of the windowed datasets. The training setup is as follows -

- n_epochs = 25
- optim = Adam
- lr=1e-4
- loss_fn = nn.CrossEntropyLoss()
- metric = accuracy_score

The trained models are saved at *'..Question 2/Task A/models/'*. The training, validation accuracy and loss curves along with metrics are provided in the Metrics section.

3.5 Machine Learning Models

For classical ML models, we make use of -

- Support Vector Machines (SVM) - linear, polynomial, sigmoid, rbf kernels
- Decision Tree Classifier (DTC)
- Random Forest Classifier (RFC
- KNearestNeighbors (KNN)

The input (X) for these models is the embeddings obtained at the **'fc1 layer'** of the CNN architecture, which contains 512 features. These embeddings are obtained for all 3 types of windowed datasets. The training and validation accuracy scores of all these models are provided in the Metrics section.

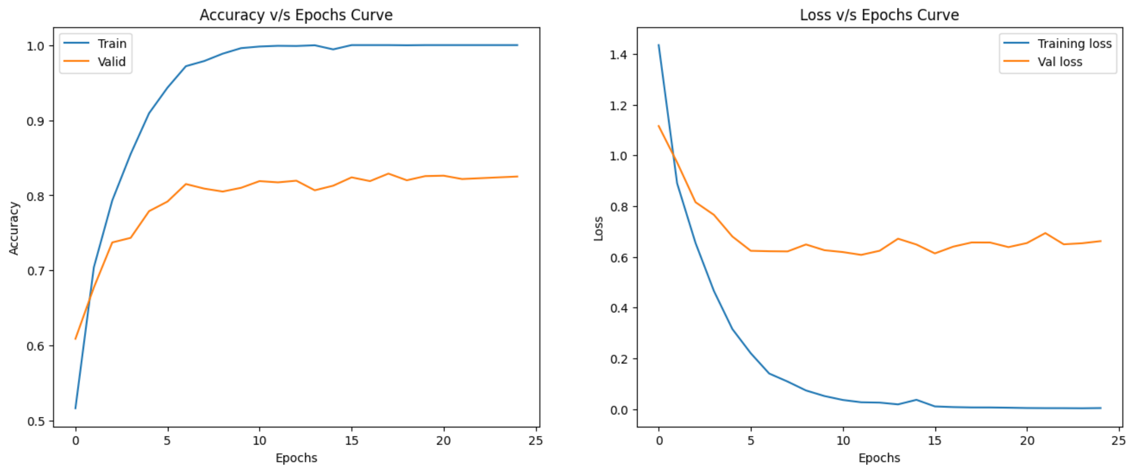


Fig. 3: Training of CNN Architecture on Hann Windowed Dataset

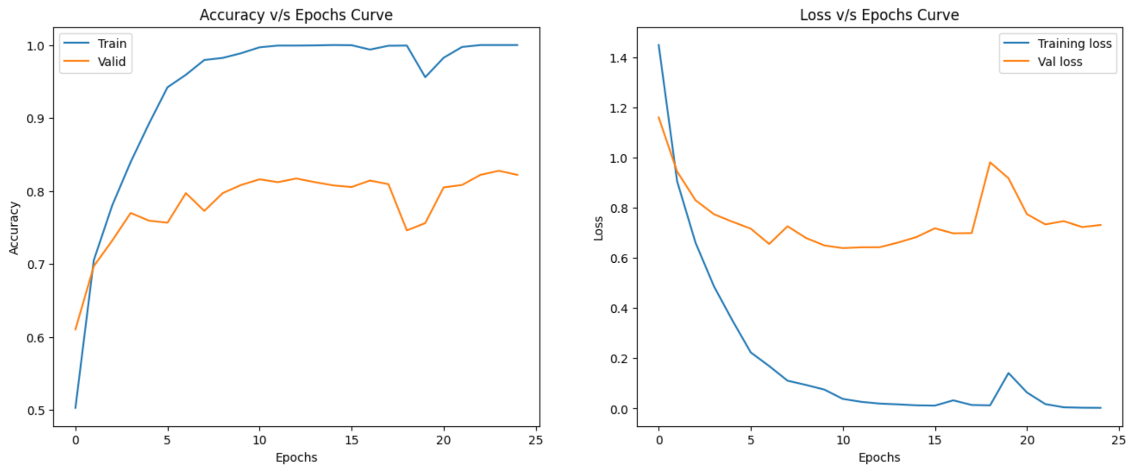


Fig. 4: Training of CNN Architecture on Rectangular Windowed Dataset

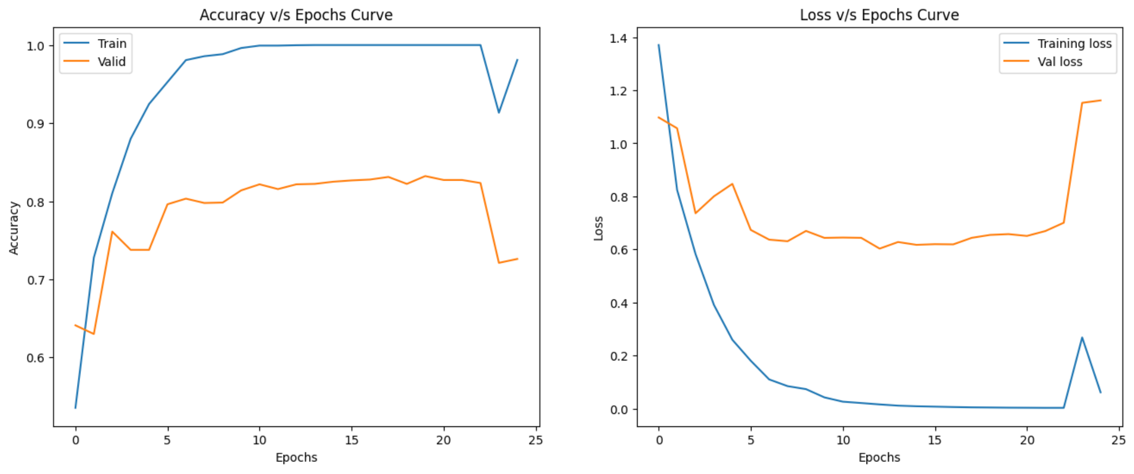


Fig. 5: Training of CNN Architecture on Hamming Windowed Dataset

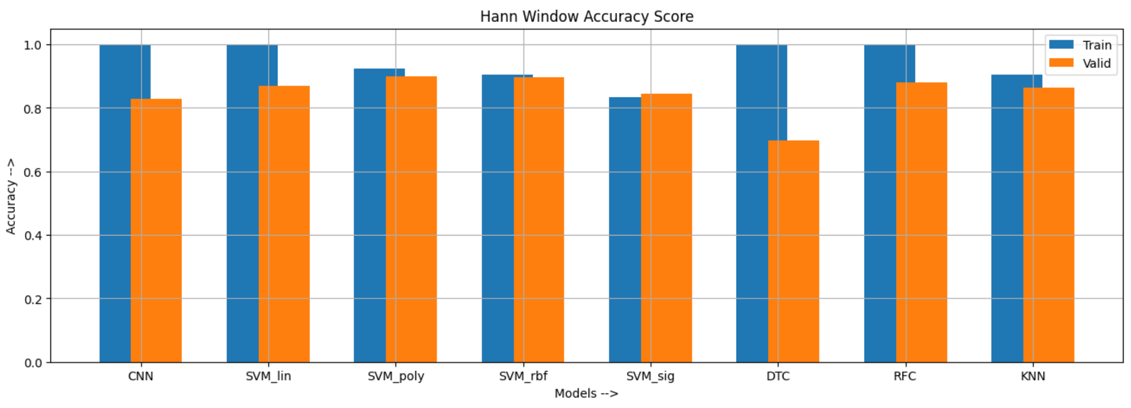


Fig. 6: Model summary on Hann Windowed Dataset

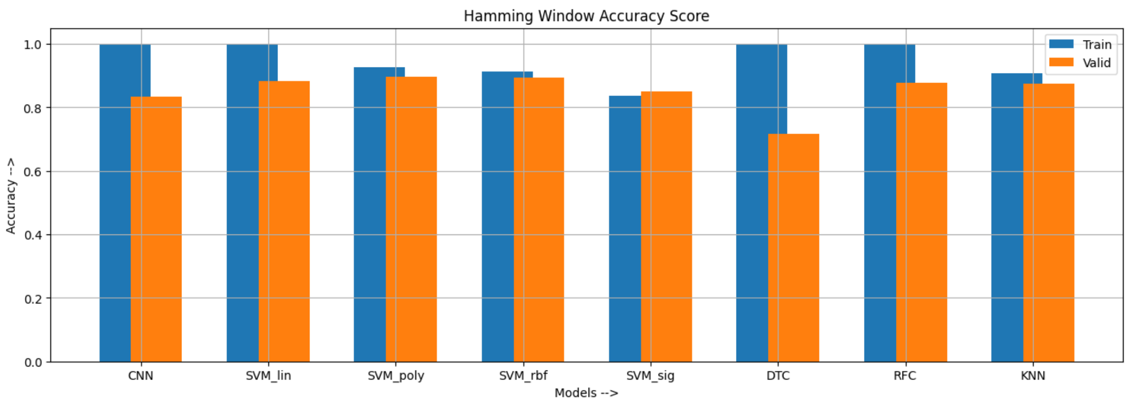


Fig. 7: Model summary on Hamming Windowed Dataset

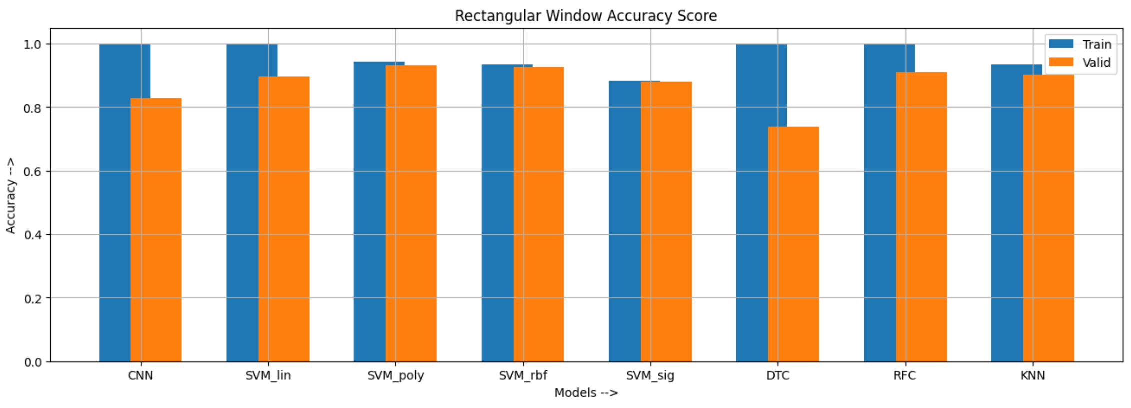


Fig. 8: Model summary on Rectangular Windowed Dataset

Window	Model	Training Accuracy	Testing Accuracy
Hann	CNN	1.0	0.825
	SVM Linear	0.98	0.86
	SVM Poly	0.92	0.89
	SVM rbf	0.90	0.895
	SVM sig 5	0.84	0.84
	DTC	1.0	0.697
	RFC	1.0	0.879
	KNN	0.90	0.86
Hamming	CNN	1.0	0.832
	SVM Linear	0.99	0.88
	SVM Poly	0.926	0.895
	SVM rbf	0.913	0.894
	SVM sig	0.835	0.849
	DTC	1.0	0.72
	RFC	1.0	0.877
	KNN	0.908	0.873
Rectangular	CNN	1.0	0.827
	SVM Linear	1.0	0.895
	SVM Poly	0.943	0.931
	SVM rbf	0.935	0.927
	SVM sig 5	0.882	0.88
	DTC	1.0	0.74
	RFC	1.0	0.91
	KNN	0.90	0.87

Table 2: Training and Testing Accuracy for Different Models and Windowing Techniques

4 TASK B

4.1 Dataset

I collected 4 songs - each song of a different genre.

- Agar Tum Saath Ho - Bollywood
- Counting Stars - Pop
- Laal Ishq - Semi-classical
- Saadda Haq - Rock

4.2 Methodology

I repeated the same steps from Task A - importing the audio file using Librosa in the form of vector, converting vector to spectrogram using *scipy.signal.stft()* and then plotting its absolute magnitude. The results are provided in the Observations section.

4.3 Observations

- **Laal Ishq (Semi-Classical)** The spectrogram shows smooth and continuous bands, which means the song does not have much sudden, sharp beats. There is high focus on low-to-mid frequencies due flat, percussion sounds like that of drums, tabla, etc, traditional to the semi-classical format.

- **Counting Stars (Pop)** As in the song, there are two vocal ranges. One is a low range () and the other a high range (). There is relatively less focus on middle frequency ranges. The transistion is more or less smooth, which means there is gradual change in the vocal octaves in the song, which is how pop songs are characterized.

- **Sadda Haq (Rock)** There is a uniform distribution of intensity between mid-frequency and high-frequency regions. But there is considereable intensity of low frequency sounds, which is due to flat, percussion sounds of heavy drums used in rock music. Also there is slight peak in intensity at the extreme top frequency due to some high vocals.

- **Agar Tum Saath Ho (Bollywood)** There is a great variation in the distribution of intensities due to varying ranges of sound. There is a mix of low, high and mid ranges of vocal frequencies in this song and lack of dominating bass sounds (low frequencies) which gives a uniform spectrum over all frequemcy ranges.

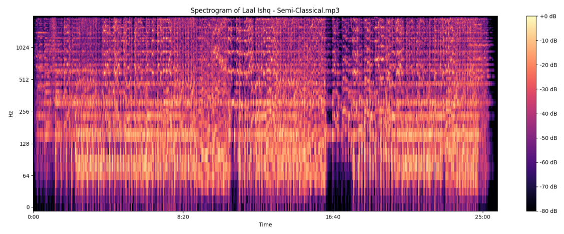


Fig. 9: Laal Ishq (Semi Classical)

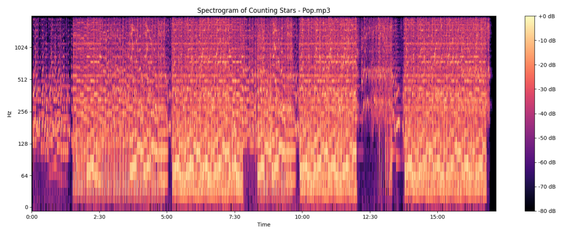


Fig. 10: Counting Stars (Pop)

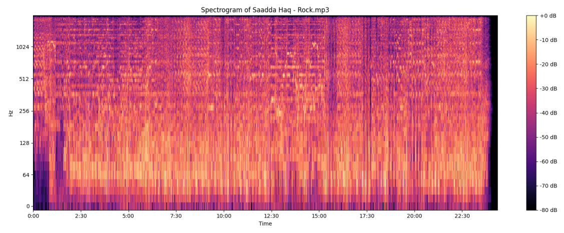


Fig. 11: Sadda Haq (Rock)

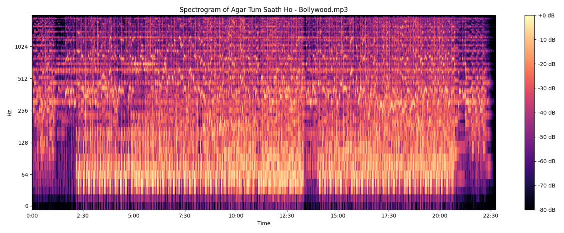


Fig. 12: Agar Tum Saath Ho (Bollywood)

Fig. 13: Spectrograms of songs of 4 different genres

5 REFERENCES

- Scipy Official Documentation [Link]
- Librosa Official Documentation [Link]
- Window types - Siemens Blogs [Link]
- Audio files download - [https://www.pagalfree.com]