Exploring the Effectiveness of CNN-GRU and Attention-Enhanced CNN-LSTM Architectures for Speech Emotion Recognition

Azhar Saparova, Viktor Makhrinov

1. Abstract

Speech-emotion recognition (SER) enhances human-computer interaction by allowing machines to recognise human emotion and respond accordingly. Although multimodal emotion recognition is known for having higher accuracy than speech-only emotion recognition, some applications imply limited access to visual and textual data. There is a need to improve the performance of speech-based emotion recognition models due to challenges of environmental noise and individual emotional expressions of each speaker. This work compares the effectiveness of SER with noisy data by models of different computational complexity. This study investigates two deep learning architectures—CNN+GRU and CNN+LSTM with self-attention — using the CREMA-D dataset. The feature extraction is performed from the spectral data of the audio using a CNN rather than hand-crafted features. Experimental results show that emotions such as anger, neutral, and happiness have the highest test accuracy, while emotions such as fear and disgust are harder to distinguish. In addition, results demonstrate that the model employing GRU shows comparatively similar results to the model employing LSTM on the CREMA-D dataset, 55.88% and 55.2%, respectively. Both models are vulnerable to oversampling, but the second model had more obvious signs of oversampling. The attempts to fine-tune the second model by adding data augmentation and changing the model structure did not lead to any improvements. The dataset was downloaded to Google Drive, available at the following link: *https://drive.google.com/drive/folders/1zqbsIxbpuTkNj7S2jjDld26DE9XeJyH3?usp=sharing*

2. Introduction

2.1 Speech Emotion Recognition

The main use of emotion recognition is found in the interaction of humans with AI systems. Such interactions include learning environment, advertisements and shopping recommendations, virtual environment, and communication of humans with AI assistants [2]. Even though all these applications can work without understanding human emotions, they can benefit substantially from improving the social influence by understanding human psychological reactions [1]. These fields are rapidly developing, and as a result, emotion recognition is attracting a lot of researchers to investigate the models capable of understanding human emotions. The main requirement for the emotion recognition system is a real-time application. Emotions are not static and are constantly changing with the context [1][3]. This can be achieved using deep-learning models that can adapt to the emotional changes of the user.

Modalities in emotion recognition include speech, visual, and text data. While all of these modalities can be efficiently separated, the most abundant modalities in human-machine interactions are speech and text [1]. Furthermore, the speech is more practical in the sense that the same text can carry a different emotional background, while the variations in tone, rhythm, and intensity reflect true emotions. Additionally, in many practical applications, such as telecommunication and call centres, contextual visual and text data is not always available [2]. On the other hand, multimodal emotion recognition was found to have higher accuracy, but it comes with a cost of more complex infrastructure, the need for data synchronisation, and higher computational complexity [2]. All of this raises the need for speech-only emotion recognition systems.

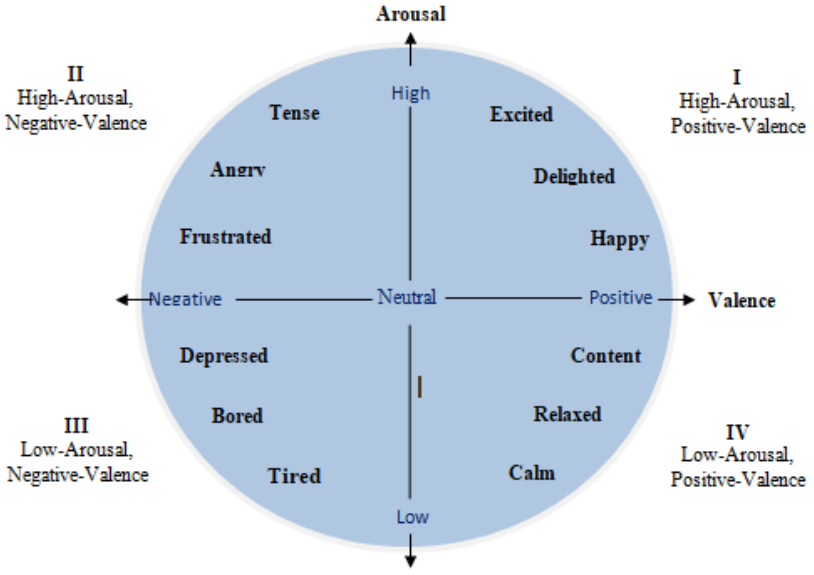

Figure 1. Emotion spectrum based on arousal level and valence

2.2 Challenges in Speech Emotion Recognition

Alongside the usefulness of the systems capable of understanding human emotions, they face a lot of related challenges. First of all, human emotions are a complex set of dynamic and context-dependent phenomena. Each instance of human action has different emotions, and everyone expresses them differently [1][2]. In particular, the speech differs with a person's age, gender, ethnicity, and nationality. Secondly, the real-world audio signals are produced in uncontrolled conditions, where speech is subject to noise and distortions [4]. As a result, speech emotion recognition systems must be able to successfully work in a noisy environment with various patterns of speech caused by the environment and the individuality of a speaker.

The challenge of speakers' individuality in expressing their emotions can be addressed with an extensive dataset. Sufficient training containing enough information on each emotion expressed by speakers of different ages, genders, and ethnicities will result in a model that will effectively work in real-life conditions. However, available datasets suffer from data imbalance and underrepresentation of various speaker features. Overall, the model must be inclusive and practical to be included in a real application with a variety of emotions and demographic backgrounds [1]. The solution to that problem is training data balancing, such as random oversampling, which will mitigate class imbalance and give enough training for minority classes [2].

The challenge of a noisy environment can be addressed by training a model with noisy data. Most audio datasets for emotion recognition are recorded in clean and controlled environments that will produce models suitable for synthetic conditions rather than real applications. The needed training data can be produced with noise injection that will bring the dataset closer to real conditions and improve its generality [4].

In the end, the challenges in Speech Emotion Recognition are caused by an imperfect training dataset, which can be solved with data augmentation with random oversampling and noise injection.

2.3 Machine learning approaches in Speech Emotion Recognition

The feature extraction in SER can be divided into handcrafted and deep features. The traditionally used handcrafted features rely on formant frequencies, signal energy, and other time-frequency features [4]. The summary of examples and best use cases for handcrafted features is shown in Table I. In contrast to manually selected features, the deep features are extracted automatically by deep learning models. These models automatically extract features by learning non-linear and high-level feature representations that outperform manual features [2].

As a result, this paper focuses on training machine learning models on deep features of the speech.

For deep feature extraction purposes, Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) are widely used to extract deep features from raw waveforms as well as spectral data [1-3]. For the spectral data, we choose the log-mel spectrograms alongside their delta and delta-delta features. The mel scale is used instead of Hertz to mimic human perception of speech. Just like the human hearing better perceives lower frequencies, the mel scale has more resolution for lower frequencies [4]. As a result, the log-Mel spectrograms are a compressed, perceptually meaningful representation of speech signals [2]. The delta and delta-delta features of lgo-mel spectrograms are first and second-order spatial derivatives that represent dynamic changes in speech signals [2]. In the end, the spectral data in the form of log-mel spectrogram with its first and second order derivatives is an effective input for a deep learning model for future feature extraction.

Table I. Summary of features used in SER

Type	Examples	Best use case
Prosodic	Pitch, energy, intensity, rate, and duration	Distinguishing emotions with different arousal levels
Spectral Features	MFCC, formants, spectral centroid, roll-off, flux	Identifying subtle tonal variations
Time-Domain Features	Zero-crossing rate, short-time energy	Simple but fast to calculate from the raw waveform
Statistical Features	Mean, max, min, standard deviation of pitch or energy, temporal variation, and range	Capture long-term emotional patterns
Deep features	Extracted by deep learning models	Automatically learning high-level feature representations

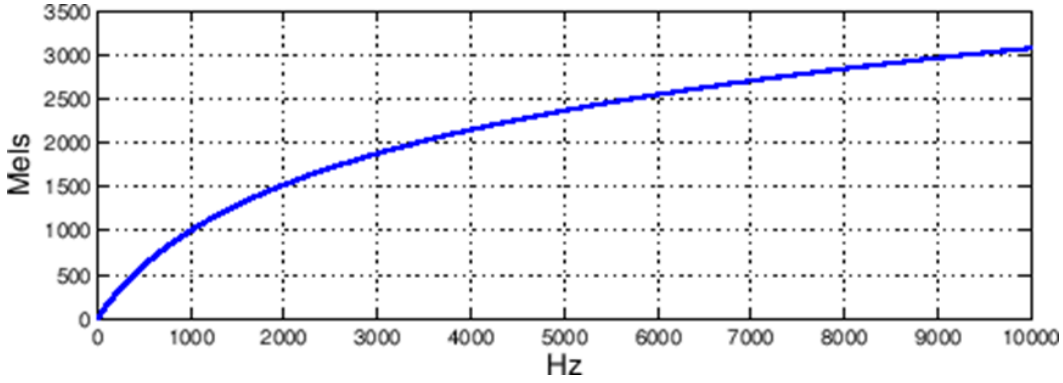

Figure 2. Comparison of the Mel scale to linear frequency

As the emotions represented in a speech are a time-dependent signal, the machine learning for emotion recognition must be efficient in time-series analysis. Recurrent Neural Networks are well-suited for sequenced data and can extract time series features and model the temporal dynamics in speech sequences. A significant drawback of RNNs is their vanishing gradient problem, as well as low effectiveness for longer dependencies, which are solved in the Long Short-Term Memory (LSTM) models [3]. LSTM was proven to be effective in complex multimodal SER systems such as shown in [2]. However, the complexity of the LSTM model may result in overfitting as well as being computationally expensive. Because of that, our research focuses on Gated Recurrent Units (GRU) that offer similar benefits with fewer parameters and are faster and simpler to train compared to LSTMs.

2.4 Motivation and Research Objective

The main goal of this study is to propose practical, resource-efficient strategies for improving SER performance under limited-data conditions. Specifically, we implement and compare two deep learning models—CNN+GRU and CNN+LSTM with self-attention—using the CREMA-D dataset. The choice of GRU is motivated by its lower computational complexity compared to LSTM, making it better suited for real-world, latency-sensitive applications. To address the limitations of small and imbalanced datasets, we apply a combined augmentation strategy that integrates random oversampling with controlled noise injection, creating a richer and more balanced training set. Additionally, we study how dropout regularisation affects generalisation in attention-enhanced models. By combining these techniques, this work aims to overcome key challenges in SER, offering insights into efficient model selection and robust data augmentation strategies.

3. Methodology

All of the code was written and compiled using Kaggle. Initially, two out of three models were run using Anaconda, but after the presentation, we employed Kaggle during the re-running process for all of our models discussed after that.
Firstly, the CREMA-D dataset was imported from Kaggle for further training [5]. Our choice of dataset was CREMA-D because widely used datasets like IEMOCAP require a license for download, and as it was mentioned before, we had a limited number of available datasets as well as computational power. Additionally, the CREMA-D dataset had audio recordings where actors spoke from a selection of 12 sentences, which was why the addition of multimodality into the model in terms of text recognition would not bring significant changes, as the textual context is fixed.

Secondly, initially, we made the mistake of performing data balancing before data splitting, the effect of which on the accuracy of the models will be discussed in the next section. So, the changed pre-processing includes the extraction of emotions from raw audio data using the extract_emotion() function and conversion of the data into logmel with the noise injection and normalisation using the save_logmel_with_deltas_unique() function before oversampling.

3.1 Audio to log-mel spectrogram conversion

The dataset contains 7442 audio files in .wav format that were loaded using the librosa.load() function. The loaded data was normalised to zero mean and unit variance, followed by standardising the duration to 3 seconds by padding with silence or trimming. This step is made to ensure all data is standardised to reduce speaker-specific variations and ensure consistency in input sizes for models.

The next step is noise injection, which aims to improve the robustness of the model in real conditions of a noisy environment. Gaussian noise is applied to all loaded audio with amplitude scaled relative to the peak amplitude of the audio. For the conversion to the frequency domain, the preprocessed signals were segmented into short overlapping frames of 25 milliseconds each, with a 10-millisecond frame shift. A Hamming window was applied to each frame to minimize spectral leakage and smooth the transition between frames.

The core feature extraction pipeline involved the conversion of each framed audio sequence into a spectrogram using a Short-Time Fourier Transform. Mel filter banks were then applied to transform the frequency to the Mel scale, which compresses higher frequencies and increases resolution at lower frequencies to better mimic human auditory perception [2][4]. A logarithmic scale transformation was applied to compress the dynamic range of the spectrogram, producing the final log-Mel spectrogram.

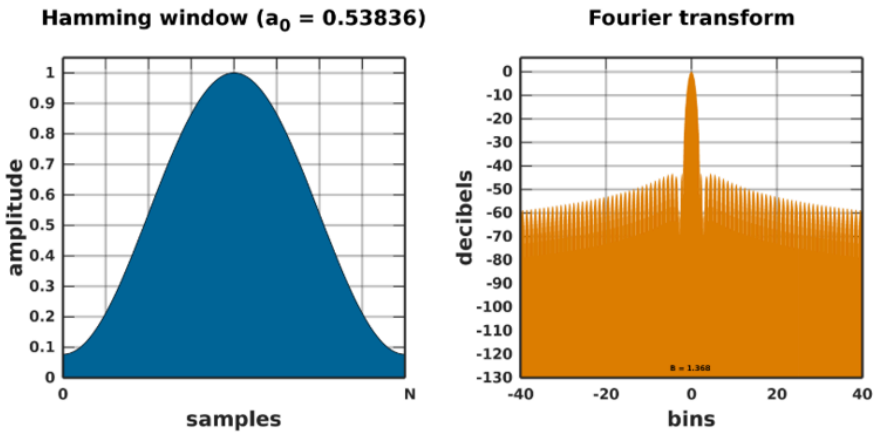

Figure 3. Shapes of the Hamming window and the Fourier transform used for the conversion of audio to a spectrogram [6]

In addition to the base log-Mel spectrogram, temporal derivatives were computed. First-order and second-order features were computed to capture dynamic changes in the signal. Following decent results reported in prior SER research [2], the log-Mel spectrogram, delta, and delta-delta matrices were stacked along a new dimension, creating a three-channel tensor (time × frequency × 3 channels). This 3D input structure is processed by convolutional neural networks, which can simultaneously learn spatial and temporal features.

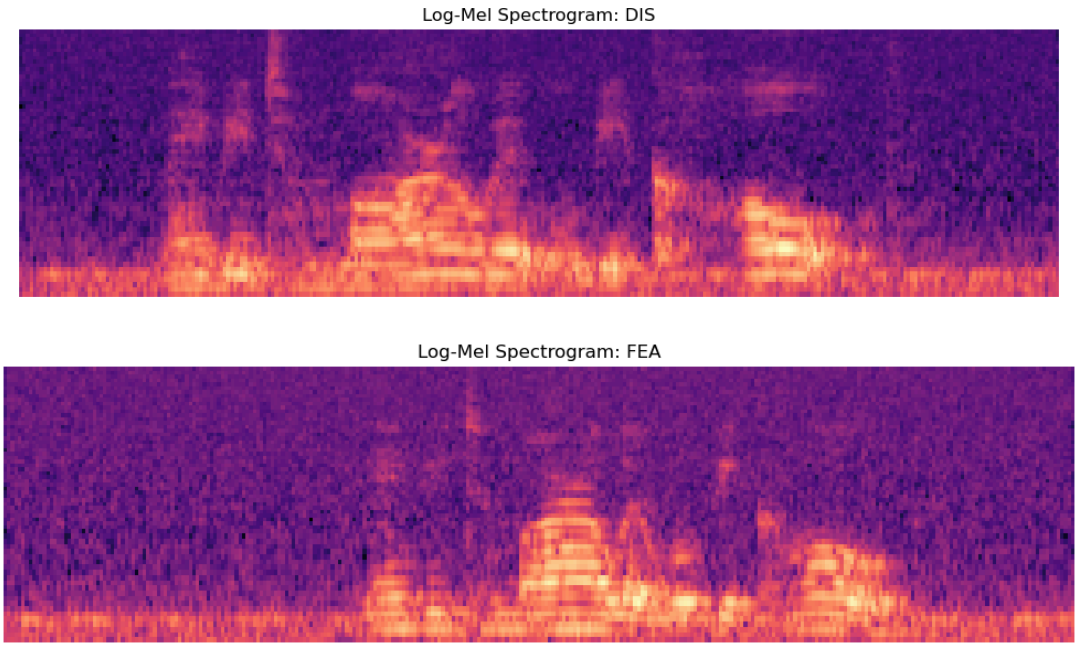

Figure 4. Log mel spectrograms of the “Maybe tomorrow it will be cold” phrase said by the same actor with disgust (top) and fear (bottom) emotions.

3.2 Rearranged data split and oversampling

All preprocessed tensors were stored in .npy format to provide faster training of models with different architectures, avoiding the data preprocessing for each of them separately.

After conversion of data into 3D npy arrays, we made a data split with the following proportions: 80% to the training set, 20% preserved for the validation set, and 20% to the test set. Now, after the data split, we performed oversampling only on the training set. The distribution before and after the data balancing can be seen in Figures 5 and 6.

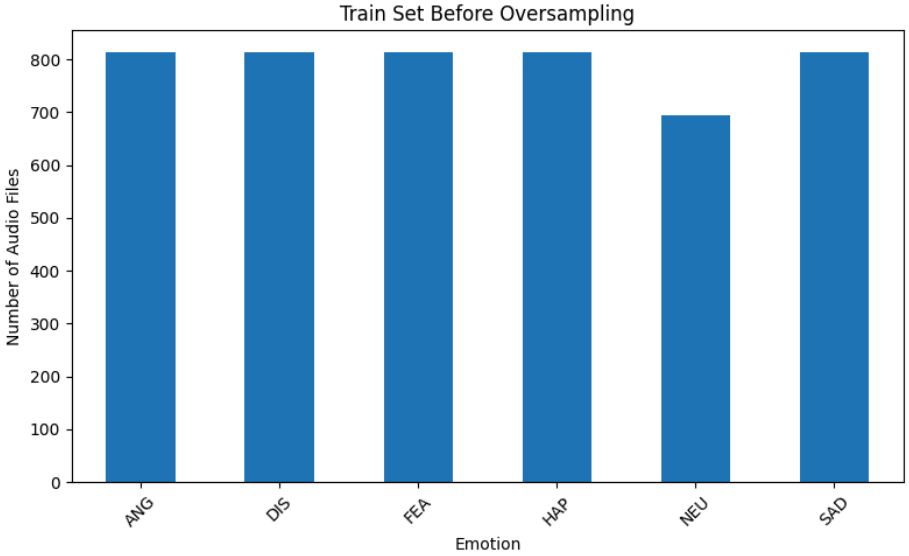

Figure 5. Training set distribution by classes before oversampling

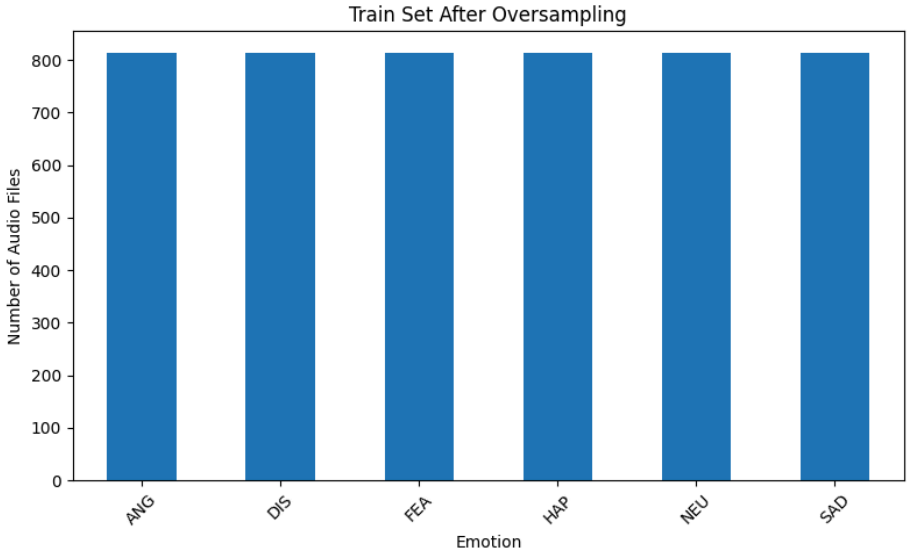

Figure 6. Training set distribution by classes after oversampling

3.3 Implementation of CNN+GRU+self-attention mechanism

Regarding the model training, as it was mentioned before, GRU was initially chosen for the training because it is less computationally expensive compared to LSTM. So, the model consists of two 2D convolutional CNN layers with 128 and 256 neurons, each layer having a kernel size of (5, 3), padding of (2, 2) with 2D max pooling and ReLU activation function. The CNN structure is used for capturing spatial dependencies in the input, which is used for feature extraction. Next, the output is reshaped and fed into a dense layer with a ReLU activation function for the conversion of data of CNN layers as a sequence that later will be fed into the GRU model for capturing temporal dependencies. After that, the GRU model with 128 neurons and dropout of 0.3 to prevent overfitting is used to capture the features in sequential data. Finally, the multi-head self-attention mechanism is used to focus selectively on important parts of the sequence and capture long-term dependencies that could be missed by the GRU model because it does not have an update gate compared to LSTM, which keeps long-term memory. The output is fed to global average pooling that reduces the number of parameters and prevents overfitting, and dense layers with ReLU and softmax activation functions for learning higher-level emotional representations and making the final classification. The overview of the described model structure can be seen in Figure 7.

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 300, 64, 3)	0	-
conv2d (Conv2D)	(None, 300, 64, 128)	5,888	input_layer[0][0]
max_pooling2d (MaxPooling2D)	(None, 150, 32, 128)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 150, 32, 256)	491,776	max_pooling2d[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 75, 16, 256)	0	conv2d_1[0][0]
reshape (Reshape)	(None, 75, 4096)	0	max_pooling2d_1[0][0]
dense (Dense)	(None, 75, 768)	3,146,496	reshape[0][0]
gru (GRU)	(None, 75, 128)	344,832	dense[0][0]
dropout (Dropout)	(None, 75, 128)	0	gru[0][0]
multi_head_attention (MultiHeadAttention)	(None, 75, 128)	33,088	dropout[0][0], dropout[0][0]
add (Add)	(None, 75, 128)	0	dropout[0][0], multi_head_attention[…
layer_normalization (LayerNormalization)	(None, 75, 128)	256	add[0][0]
global_average_pooling1d (GlobalAveragePooling1D)	(None, 128)	0	layer_normalization[0…
dense_1 (Dense)	(None, 64)	8,256	global_average_poolin…
dense_2 (Dense)	(None, 6)	390	dense_1[0][0]

Total params: 4,030,982 (15.38 MB)

Trainable params: 4,030,982 (15.38 MB)

Non-trainable params: 0 (0.00 B)

Figure 7. Summary of CNN+GRU+multi-head self-attention model

Next, for the training, we used early stopping with patience of 15 to prevent model overfitting, the number of epochs was set to 50, and the batch size was set to 32. We used categorical cross-entropy loss and Adam optimizer with lr_schedule with the following parameters:

```
tf.keras.optimizers.schedules.ExponentialDecay(
  initial_learning_rate=1e-4,
  decay_steps=1000,
  decay_rate=0.95,
  staircase=True
)
```

3.4 Implementation of CNN+LSTM+self-attention mechanism

Later, we decided to use a more complex model to increase the accuracy further (the results of training can be seen in the next section). For that, we employed LSTM instead of GRU. However, this time we decided to keep the multi-head attention mechanism because our GRU model had worse results compared to the one shown in the lecture (to be discussed later). All of the training parameters, data split proportions, and the number of epochs were the same as during the training of the GRU model. The overview of the model structure can be seen in Figure 8.

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 300, 64, 3)	0	-
conv2d (Conv2D)	(None, 300, 64, 128)	5,888	input_layer[0][0]
batch_normalization (BatchNormalization)	(None, 300, 64, 128)	512	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 300, 32, 128)	0	batch_normalization[0…
conv2d_1 (Conv2D)	(None, 300, 32, 256)	491,776	max_pooling2d[0][0]
batch_normalization_1 (BatchNormalization)	(None, 300, 32, 256)	1,024	conv2d_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 300, 16, 256)	0	batch_normalization_1…
reshape (Reshape)	(None, 300, 4096)	0	max_pooling2d_1[0][0]
lstm (LSTM)	(None, 300, 128)	2,163,200	reshape[0][0]
dropout (Dropout)	(None, 300, 128)	0	lstm[0][0]
multi_head_attention (MultiHeadAttention)	(None, 300, 128)	131,968	dropout[0][0], dropout[0][0]
global_average_pooling1d (GlobalAveragePooling1D)	(None, 128)	0	multi_head_attention[…
dense (Dense)	(None, 64)	8,256	global_average_poolin…
dropout_2 (Dropout)	(None, 64)	0	dense[0][0]
dense_1 (Dense)	(None, 6)	390	dropout_2[0][0]

Total params: 2,803,014 (10.69 MB)
Trainable params: 2,802,246 (10.69 MB)
Non-trainable params: 768 (3.00 KB)

Figure 8. Summary of CNN+LSTM+multi-head self-attention model

3.5 Fine-tuning of CNN+LSTM+self-attention mechanism with data augmentation

The second model employing LSTM had a smaller difference in accuracy compared to the previous training process with incorrect pre-processing, but both versions had severe overfitting. So, the model could be fine-tuned to resist it, but the results from the presentation show that major changes can be made with data augmentation or the combination of the CREMA-D dataset with other datasets. Thus, we decided to make data augmentation on the dataset and expand by 25%, adding new augmented data. The reason for the addition of only 25% was to keep the natural data in the majority.

Data was augmented with the following functions:

```
# Augmentation functions
def add_noise(y, noise_level=0.035):
    noise_amp = noise_level * np.random.uniform() * np.amax(y)
    return y + noise_amp * np.random.normal(size=y.shape)

def pitch_shift(y, sr, n_steps=2):
    return librosa.effects.pitch_shift(y, sr=sr, n_steps=n_steps)

def volume_perturbation(y, gain_db=5):
    return y * (10 ** (gain_db / 20))

def specaugment(mel):
    """Apply SpecAugment to log-mel spectrogram."""
    num_masks = 2
    freq_mask_param = 8
    time_mask_param = 8

    augmented = mel.copy()
    for _ in range(num_masks):
        f = np.random.randint(0, freq_mask_param)
        f0 = np.random.randint(0, augmented.shape[0] - f)
        augmented[f0:f0+f, :] = 0

        t = np.random.randint(0, time_mask_param)
        t0 = np.random.randint(0, augmented.shape[1] - t)
        augmented[:, t0:t0+t] = 0
    return augmented
```

So, the noise injection used before is kept. Pitch shift shifts it up or down slightly, volume perturbation increases or decreases loudness, and SpecAugment applies maskings on the spectrogram (time masking and frequency masking). In such a way, the number input .npy arrays increased from 7442 to 9302.

Also, the model is changed a little bit by reducing the number of neurons in the CNN model to 64 and 128, and in the LSTM model to 62. Also, the model was changed by adding L2 regularizers in CNN layers, and adding dropouts of 0.5 after the LSTM and dense layer. The overview of the model can be seen in Figure 9. Also, the data split was changed as follows: 70% for the training set, 15% for the validation set, and 15% for the test set. Finally, the lr_schedule parameters were changed to the following:

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=5e-4,
    decay_steps=1000,
    decay_rate=0.9,
    staircase=True
)
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 300, 64, 3)	0	-
conv2d_4 (Conv2D)	(None, 300, 64, 64)	2,944	input_layer_2[0][0]
batch_normalization_4 (BatchNormalization)	(None, 300, 64, 64)	256	conv2d_4[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 300, 32, 64)	0	batch_normalization_4…
conv2d_5 (Conv2D)	(None, 300, 32, 128)	123,008	max_pooling2d_4[0][0]
batch_normalization_5 (BatchNormalization)	(None, 300, 32, 128)	512	conv2d_5[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 300, 16, 128)	0	batch_normalization_5…
reshape_2 (Reshape)	(None, 300, 2048)	0	max_pooling2d_5[0][0]
lstm_2 (LSTM)	(None, 300, 128)	1,114,624	reshape_2[0][0]
dropout_4 (Dropout)	(None, 300, 128)	0	lstm_2[0][0]
multi_head_attention_2 (MultiHeadAttention)	(None, 300, 128)	33,088	dropout_4[0][0], dropout_4[0][0]
layer_normalization_2 (LayerNormalization)	(None, 300, 128)	256	multi_head_attention_…
global_average_pooling1d… (GlobalAveragePooling1D)	(None, 128)	0	layer_normalization_2…
dense_4 (Dense)	(None, 64)	8,256	global_average_poolin…
dropout_6 (Dropout)	(None, 64)	0	dense_4[0][0]
dense_5 (Dense)	(None, 6)	390	dropout_6[0][0]

Total params: 1,283,334 (4.90 MB)

Trainable params: 1,282,950 (4.89 MB)

Non-trainable params: 384 (1.50 KB)

Figure 9. Summary of CNN+LSTM+multi-head self-attention model with fine-tuning

4. Results and Discussion

4.1 Results of CNN+GRU+self-attention mechanism model

The training process of the first model (CNN+GRU+multi-head self-attention) stopped at epoch 18 because of early stopping. The training and validation accuracy and loss can be seen from Figure 10.

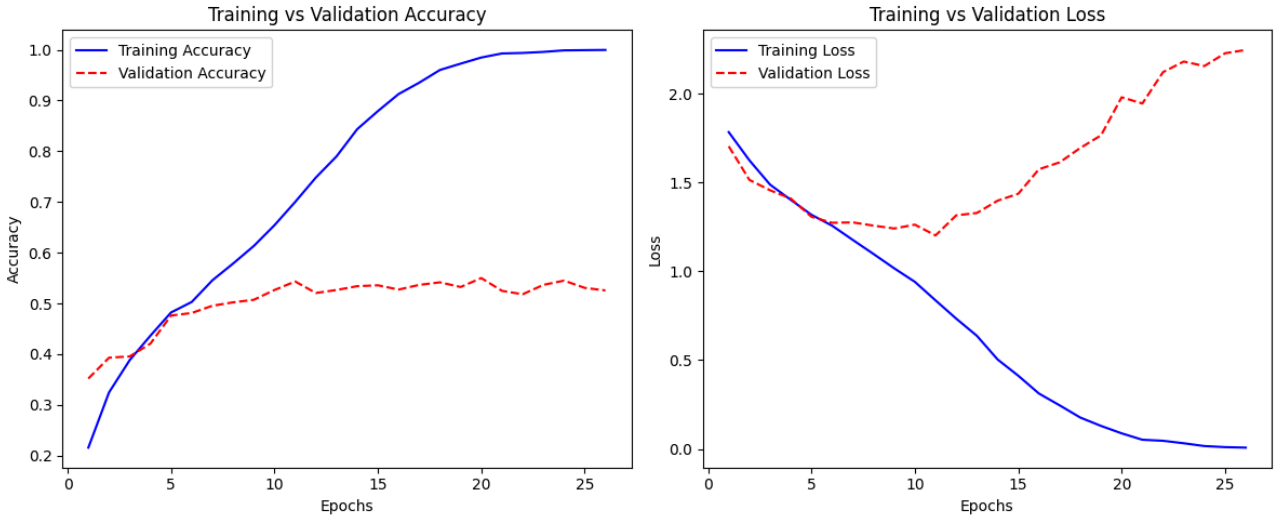

Figure 10. Accuracy and loss change for CNN+GRU+multi-head self-attention model

As can be seen from Figure 10, the training accuracy is increasing fast compared to validation accuracy, which is mostly always stable after about epoch 5. Meanwhile, from the graph of loss, validation loss starts to increase from about epoch 10. In general, we see relatively poor performance and signs of overfitting.

The test accuracy resulted to be 55.88%, which decreased dramatically compared to the previous results of 71.95% with incorrect pre-processing (oversampling before data splitting). It can be concluded that the test set with the correct pre-processing is closer to real-world data; thus, the new score is more honest in terms of the data presented in the test set. Also, per-class test accuracy resulted to be the following:

ANG: 0.6457 (164/254) compared to 0.8189 before (with incorrect pre-processing)
DIS: 0.5276 (134/254) compared to 0.6706 before (with incorrect pre-processing)
FEA: 0.4803 (122/254) compared to 0.6863 before (with incorrect pre-processing)
HAP: 0.5490 (140/255) compared to 0.7283 before (with incorrect pre-processing)
NEU: 0.5688 (124/218) compared to 0.7992 before (with incorrect pre-processing)
SAD: 0.5827 (148/254) compared to 0.6142 before (with incorrect pre-processing)

Analyzing the per-class test accuracy, we can see that the model performs best on classes like anger, sad, and neutral, while previously the best scores were for classes like anger, happiness,

and neutral. Comparing previous and new test results, it can be assumed that classes like disgust, fear, and happiness were underrepresented previously, because the classes have larger differences in the test scores, meaning that the model had previously benefited from seeing oversampled examples of these classes in both training and testing.

We also obtained the confusion matrix and classification report with precision, recall, f1-score, and support for each class, which can be seen in Figures 11 and 12, respectively.

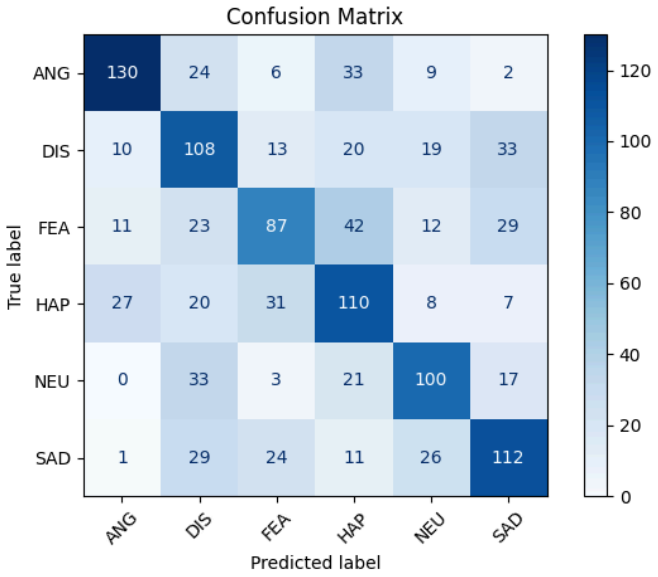

Figure 11. Confusion matrix for CNN+GRU+multi-head self-attention model

Classification Report:

	precision	recall	f1-score	support
ANG	0.73	0.64	0.68	204
DIS	0.46	0.53	0.49	203
FEA	0.53	0.43	0.47	204
HAP	0.46	0.54	0.50	203
NEU	0.57	0.57	0.57	174
SAD	0.56	0.55	0.56	203
accuracy			0.54	1191
macro avg	0.55	0.54	0.55	1191
weighted avg	0.55	0.54	0.54	1191

` Figure 12. Classification report for CNN+GRU+multi-head self-attention model

4.2 Results of CNN+LSTM+self-attention mechanism model

Now, analyzing the results of the second model (CNN+LSTM+multi-head self-attention model), the training process stopped at epoch 24 because of early stopping. The training and validation accuracy and loss can be seen in Figure 13. This time, the LSTM model is more robust to overfitting compared to the model with incorrect pre-processing. So, there is no need to

aggressively change the model, adding dropouts of large values, as was with the previous model (with incorrect pre-processing). But the two models cannot be compared side by side because previously we used the following mechanism instead of the multi-head attention mechanism:

```
score = layers.Dense(1)(x)
attn_weights = tf.nn.softmax(score, axis=1)
 x = tf.reduce_sum(x * attn_weights, axis=1)
 context = x
```

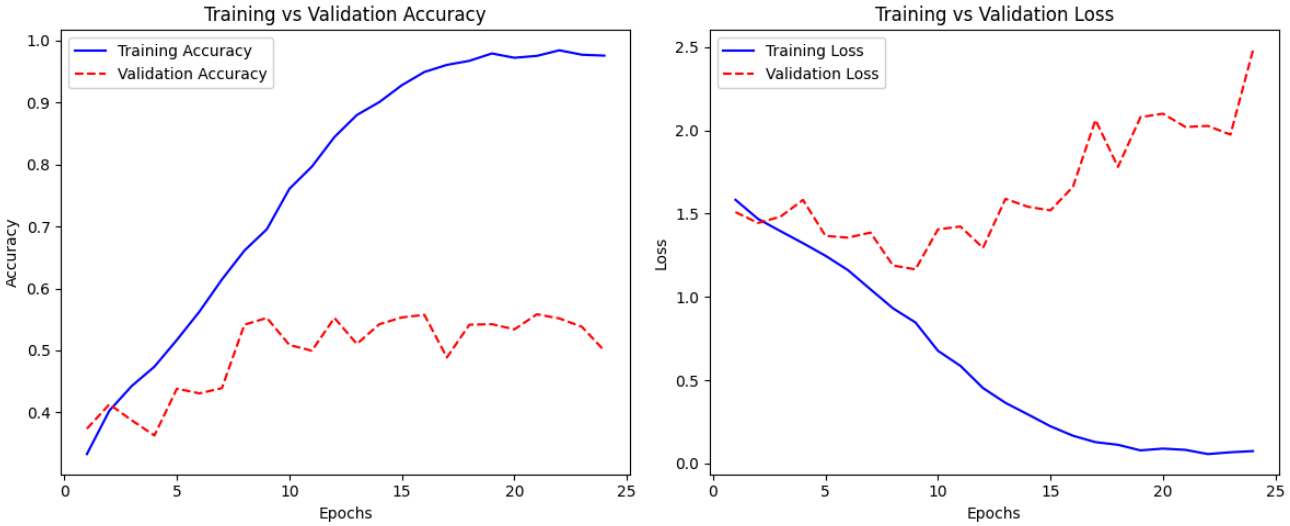

Figure 13. Accuracy and loss change for CNN+LSTM+multi-head self-attention model

Talking about the test accuracy, it resulted to be 55.2%, which is very close to the performance of the GRU model. Also, compared to the GRU model, which has a difference of 16.07% between implementation with incorrect and correct pre-processing, the LSTM model has a difference of 10.99%. Per-class accuracy shows that neutral is now classified better than anger, fear has an even smaller accuracy, and disgust has a higher accuracy. The differences are probably due to the fact that LSTM better memorizes long-term dependencies:

ANG: 0.6024 (153/254)
DIS: 0.5787 (147/254)
FEA: 0.3150 (80/254)
HAP: 0.5255 (134/255)
NEU: 0.7064 (154/218)
SAD: 0.6063 (154/254)

We also obtained a confusion matrix and classification report with precision, recall, f1-score, and support for each class, which can be seen in Figures 14 and 15, respectively.

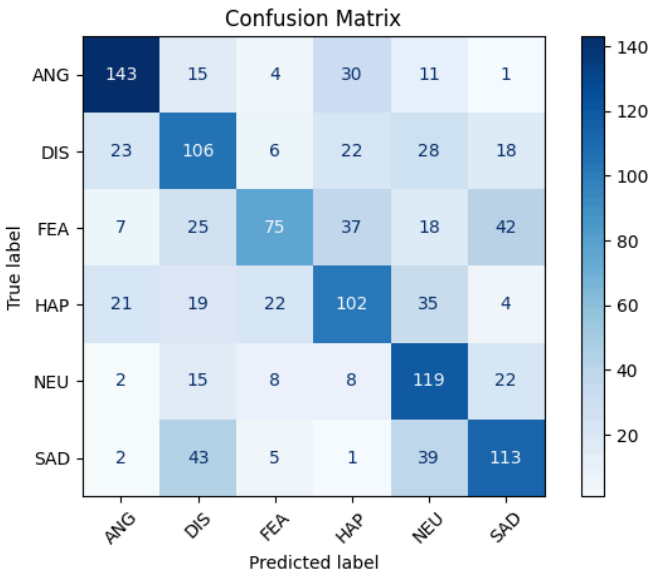

Figure 14. Confusion matrix for CNN+LSTM+multi-head self-attention model

\Classification Report:

	precision	recall	f1-score	support
ANG	0.72	0.70	0.71	204
DIS	0.48	0.52	0.50	203
FEA	0.62	0.37	0.46	204
HAP	0.51	0.50	0.51	203
NEU	0.48	0.68	0.56	174
SAD	0.56	0.56	0.56	203
accuracy			0.55	1191
macro avg	0.56	0.56	0.55	1191
weighted avg	0.56	0.55	0.55	1191

Figure 15. Classification report for CNN+LSTM+multi-head self-attention model

4.2 Results of CNN+LSTM+self-attention mechanism model with fine-tuning

Finally, the LSTM model was fine-tuned and trained with data augmentation. Looking at Figure 16, it can be noticed that despite the change of model structure and the addition of dropouts and regularizers, to model still struggles with overfitting. The training was stopped at epoch 22.

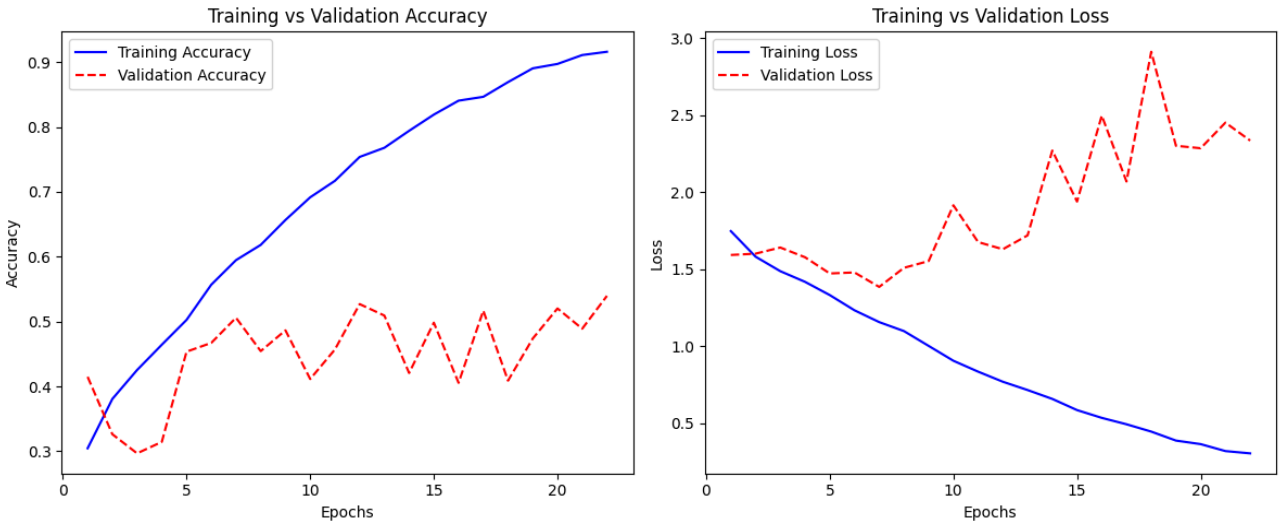

Figure 16. Accuracy and loss change for fine-tuned CNN+LSTM+multi-head self-attention model

Also, unfortunately, the test accuracy decreased to 50.14%, disproving the idea of data augmentation being helpful for increasing the test accuracy. Our suggestion is that augmentation is not realistic or aggressive enough, and thus the model is trained on the less quality data. So, in this case, the data augmentation brought more harm than benefit to our model. Nevertheless, it was worth taking a try, because previously (with incorrect pre-processing), the fine-tuning without data augmentation improved the robustness of the model a little bit, and the test accuracy only by 2%. Per-class test accuracy resulted as follows:

ANG: 0.6287 (149/237)
DIS: 0.5042 (121/240)
FEA: 0.3502 (83/237)
HAP: 0.4435 (106/239)
NEU: 0.4976 (102/205)
3SAD: 0.5840 (139/238)

We also obtained a confusion matrix and classification report with precision, recall, f1-score, and support for each class, which can be seen in Figures 17 and 18, respectively.

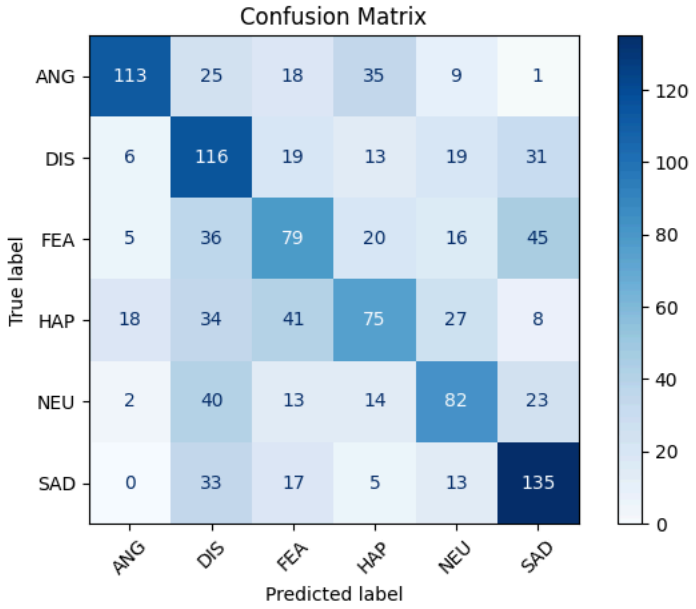

Figure 17. Confusion matrix for fine-tuned CNN+LSTM+multi-head self-attention model

Classification Report:

	precision	recall	f1-score	support
ANG	0.78	0.56	0.66	201
DIS	0.41	0.57	0.48	204
FEA	0.42	0.39	0.41	201
HAP	0.46	0.37	0.41	203
NEU	0.49	0.47	0.48	174
SAD	0.56	0.67	0.61	203
accuracy			0.51	1186
macro avg	0.52	0.50	0.51	1186
weighted avg	0.52	0.51	0.51	1186

Figure 18. Classification report fine-tuned for CNN+LSTM+multi-head self-attention model

5. Conclusion

This work implements two deep learning models, namely CNN+GRU and CNN+LSTM with a self-attention mechanism, for speech emotion recognition using the CREMA-D dataset under class-imbalanced and noisy data conditions. Experimental results show that both models have comparable moderate test accuracies (55.88% for CNN+GRU and 55.2% for CNN+LSTM), demonstrating that models involving GRU could compete with LSTM models but with lower computational complexity. Also, the second model was changed by reducing the number of neurons in the model structure and the addition of data augmentation, but it led to even worse results in terms of test accuracy (50.14%). All models suffered from overfitting, particularly when improperly oversampling before splitting the data. Correcting the preprocessing process by splitting data before the oversampling resulted in a drop in test accuracy, which demonstrates the importance of proper data pre-processing to avoid optimistic results and bias. Also, the idea of data augmentation led to opposite results compared to initial expectations, because previously, the addition of dropouts and regularizers improved test accuracy by 2%. Nevertheless, it was worth trying to add data augmentation, the problem could be unrealistic augmented data. Per-class accuracy analysis concluded the struggle to distinguish the emotions of fear and disgust, which underperformed uniformly compared to all the other classes of anger, neutral, and happiness. The per-class accuracy also differed between the two models, showing the difference between GRU and LSTM in terms of keeping long-term dependencies.

Lastly, although CNN+GRU and CNN+LSTM architectures show potential in SER tasks, data preprocessing and augmentation processes greatly affect their performance. The opportunity to utilize larger and more diverse datasets, explore the use of hybrid structures, and improve attention mechanisms would lead to better results, showing the full potential of deep learning techniques.

6. References

[1] T. Ahmed, I. Begum, M. S. Mia, and W. Tasnim, "Multimodal Speech Emotion Recognition Using Deep Learning and the Impact of Data Balancing," 2023 5th International Conference on Sustainable Technologies for Industry 5.0 (STI), 2023. doi: 10.1109/STI59863.2023.10464522.

[2] S. Zhang, Y. Yang, C. Chen, X. Zhang, Q. Leng, and X. Zhao, "Deep learning-based multimodal emotion recognition from audio, visual, and text modalities: A systematic review of recent advancements and future prospects," Expert Systems With Applications, vol. 237, 2024. doi: 10.1016/j.eswa.2023.121692.

[3] S. Chen, M. Zhang, X. Yang, Z. Zhao, T. Zou, and X. Sun, "The Impact of Attention Mechanisms on Speech Emotion Recognition," Sensors, vol. 21, no. 22, 2021. doi: 10.3390/s21227530.

[4] D. Campo, O. L. Quintero, and M. Bastidas, "Multiresolution analysis (discrete wavelet transform) through Daubechies family for emotion recognition in speech," Journal of Physics: Conference Series, vol. 705, no. 1, p. 012034, 2016. doi: 10.1088/1742-6596/705/1/012034.

[5] https://www.kaggle.com/datasets/ejlok1/cremad

[6] Kawade, Rupali & Jagtap, Sonal. (2023). Comprehensive Study of Automatic Speech Emotion Recognition Systems. International Journal on Recent and Innovation Trends in Computing and Communication. 11. 709-717. 10.17762/ijritcc.v11i9s.7743.