CSE676-Project Report

Ruijun Liu (50542228), Shailesh Mahto (50540379), Wendan Zhao (50537830)

May 2024

Previously, classification tasks such as emotion recognition have mostly focused on single-modal data, frequently ignoring the rich information accessible across several modalities. We aim to address these restrictions by investigating a multimodal method that utilize multi-modal data, deriving from the potential for multimodal analysis to improve the accuracy and depth of hate speech identification. This method can improves prediction accuracy in great extent.

1 Dataset

Our project builds on the extensive research conducted in the paper titled "HateMM: A Multi-Modal Dataset for Hate Video Classification"[1]. This paper provided the framework and the dataset for our project, which delves into the classification of hate speech across multiple modalities.

1.1 Data Overview

Video: The HateMM dataset includes 1083 videos in which 431 are hate videos and the rest are non-hate videos, each labeled as hate or non-hate. A deeper analysis of the video content reveals that certain objects appear more frequently in hate videos than non-hate videos. For instance, religious persons and cartoonish mockeries are present significantly more often in hate videos compared to non-hate videos. These visual frames serve as critical indicators for automated detection systems aiming to identify hate speech visually.

Audio: Audio analysis of the videos shows that the Root Mean Square (RMS) energy, which measures the audio signal's strength, is more unevenly distributed in hate videos. These videos often exhibit sharp rises and spikes in RMS energy, indicating intense or aggressive speech patterns which usually result from instances of shouting and are characteristic of hate speech. This shows that audio signal is also a contributing indicator of hate video calssification.

Transcript: We obtain the transcripts of these videos using Vosk. Lexical analysis of these transcripts show that certain words and phrases are much more prevalent in hate videos. For example, aggressive and derogatory terms are used significantly more often in hate speech contexts. This textual analysis is essential for developing algorithms that can detect hate speech based on language use, taking into account both the frequency of certain words and the context in which they appear.

The HateMM dataset's multimodal nature makes it an excellent fit for our project's objectives. By combining video, audio, and textual data, our models can gain a more comprehensive understanding of hate speech, similar to human perception but on a scale that human moderators cannot do alone. This method is crucial because it enables the detection of hate speech that may be obscured in one modality but visible in another.

1.2 Feature engineering

Video: We utilized Mel-Frequency Cepstral Coefficients (MFCCs) to capture the audio characteristics of each video. MFCCs can be used to represent the power spectrum of sound. To standardize the audio samples' duration and avoid any distortions during feature extraction, we employed padding techniques and ensure the audio feature vectors for all video maintain 40-dimension vectors. Furthermore, we condensed temporal information into representative feature vectors by computing the mean of these MFCC coefficients over time. Thus enhancing the robustness of our analysis and facilitating more effective machine learning model training.

Audio: For the video modality, we extracted a uniform set of 100 frames from each video, regardless of the original duration. This was achieved by sampling frames at regular intervals throughout the video. For shorter videos, we used white frames as padding to maintain consistency. Each of these frames was then divided into patches and processed through a pre-trained Vision Transformer (ViT) model. The ViT architecture linearly embeds patches of the input image and processes them through a series of Transformer encoder layers, allowed us to obtain a 768-dimensional feature vector for each frame, resulting a 768100 feature matrix for each video. This method captures complex spatial relations within the video content.

Transcript: For the textual data, we used the Vosk offline voice recognition toolkit to generate transcripts from the audio files, which were then placed in a pickle file tagged with video names. We carried out simple statistical analysis on the transcripts, finding an average word count of 253 for hate speech and 227 for non-hate speech films. Then, we utilized BERT's tokenizer to convert them into a model input format that included token IDs and attention masks. Given the different lengths of transcripts, we used padding and truncation as needed. The token, segmentation, and position embeddings were then fed into a 12-layer BertEncoder. The representation provided by the [CLS] token, which is intended to contain the entire text's context, was then used in the classification layer, yielding a 768 feature vector for each video transcript.

2 Model Description

For this project, we developed three unimodal models for each modality as well as a combined multimodal classifier. The goal is to exploit the specific characteristics and information contained within its respective modality.

Audio Model: One reason we chose MFCC feature for audio is it can be trained on simple model and achieve good results, such as fully connected layers. Our model choice for audio is a simple fully connected layers, beginning with an input layer that takes 40 MFCC features. It includes one hidden layer with 128 neurons, utilizing Xavier initialization for optimal weight setting to promote effective training convergence. The model integrates batch normalization and ReLU activation functions to enhance stability and non-linearity. The final output layer reduces the dimension to 64, making it suitable for fusion in the multimodal setup. However, we can still test unimodal performance simply by reducing the output layer to a dimension of 2, this strategy can be applied to all following unimodal models.

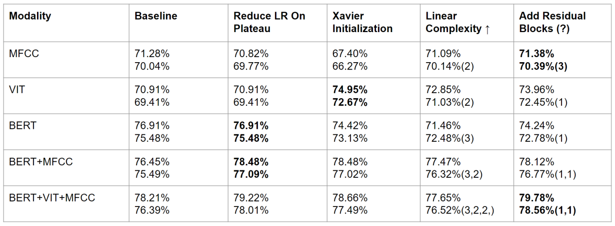
Video Model: We picked LSTM as our video model, which is paired with an attention mechanism to accommodate the sequential nature of video input. The LSTM processes 768-dimensional feature vectors collected from video frames, and depending on the design, it may include several layers. The attention mechanism, which is made up of linear layers and a tanh activation, focuses on important frames, improving the model's ability to grasp essential visual information. A completely connected layer condenses the attention mechanism's output into a 64-dimensional vector.

Text Model: The Text Model is identical to the Audio Model, with fully connected layers initialized using the Xavier technique, batch normalizing, and ReLU activations. It begins with an input size of 768, which reflects the dimensionality of BERT embeddings. Depending on the setup, many hidden layers can be added, each with the same size, to enhance the learning capability. The last output layer again compresses the characteristics into a 64-dimensional output.

The multimodal classifier aggregate the results from the previous unimodal classifiers. Each modality's output is the same dimension of 64, allowing for simple concatenation to form a 196*1 vector for final classification. A final fully connected layer combines these features to give the final categorization output. The fusion of modalities allows the system to use complimentary information, which is important for jobs that need sophisticated interpretation, such as hate speech identification. We've also tested multimodal classifier using two of the modalities. The logic is same as the multimodal discussed above, we simply combined the output from unimodal classifiers and pass them through a final fully connected layer to give the the classification result. This can be extended to any pair of unimodal classifier, but we've only tested the combination of text and audio unimodal classifiers.

Apart from the basic structures, we also integrated residual blocks to increase complexity. Each residual block consists of two linear layers with a ReLU activation and batch normalization, incorporating dropout for regularization. The key feature of these blocks is the residual connection that adds the input directly to the output of the block, allowing for better gradient flow during training and enabling the network to learn identity functions effectively, preventing performance degradation as the network depth increases. This approach helps in maintaining robust training dynamics and improves the model's ability to generalize across diverse datasets. We've also experimented increasing number of layers.

The graph below displays each of the models we tested. The Linear Complexity and Add Residual Blocks columns describe the complexity experiments we conducted. The bracketed number denotes the number of layers in the model. For multimoal classifiers, the number from left to right in the Modality column represents the number of layers for each classifier. For example, BERT+MFCC with the bracketed number (3,2) denotes three fully linked layers for BERT features and two fully connected layers for MFCC features. Our best performing model is the multimodal classifiers, utilizing all audio, text and video modaloties, with increased complexity and optimization methods. We achieved accuracy of 79.78% and Macro F1 score of 78.56%.



Modality	Baseline	Reduce LR On Plateau	Xavier Initialization	Linear Complexity ↑	Add Residual Blocks (?)
MFCC	71.28% 70.04%	70.82% 69.77%	67.40% 66.27%	71.09% 70.14%(2)	**71.38%** **70.39%(3)**
VIT	70.91% 69.41%	70.91% 69.41%	**74.95%** **72.67%**	72.85% 71.03%(2)	73.96% 72.45%(1)
BERT	76.91% 75.48%	**76.91%** **75.48%**	74.42% 73.13%	71.46% 72.48%(3)	74.24% 72.78%(1)
BERT+MFCC	76.45% 75.49%	**78.48%** **77.09%**	78.48% 77.02%	77.47% 76.32%(3,2)	78.12% 76.77%(1,1)
BERT+VIT+MFCC	78.21% 76.39%	79.22% 78.01%	78.66% 77.49%	77.65% 76.52%(3,2,2,)	**79.78%** **78.56%(1,1)**

3 Loss Function

The loss function we choose is cross-entropy loss function. cross-entropy loss is a commonly used loss function in machine learning, particularly for classification problems. Since our problem is a binary classification problem, cross-entropy loss is a perfect choice.

Cross-entropy loss measures the difference between two probability distributions, in the context of our problem, it quantifies how well the model's predicted probabilities match the actual target probabilities. The cross-entropy loss function is used to find the optimal solution by adjusting the weights of a machine learning model during training. The objective is to minimize the error between the actual and predicted outcomes. Thus, a measure closer to 0 is a sign of a good model, whereas a measure closer to 1 is a sign of a poor-performing model.

We implement cross-entropy loss function by importing PyTorch class torch.nn.CrossEntropyLoss(). It computes the cross entropy loss between input logits and target. For training, we pass argument weight [0.41, 0.59] as we have a slightly unbalanced training set, while other arguments use default setting. As the default reduction is 'mean', the formula it uses to compute the loss is as following:

$\ell(x, y) = \frac{1}{\sum_{n=1}^{N} w_{y_n} \cdot \nVdash_{y_n \neq \text{ignore\_index}}} \sum_{n=1}^{N} l_n$

For testing, we set reduction argument as 'sum', therefore the formula it uses to compute the loss is as following:

$\ell(x, y) = \sum_{n=1}^{N} l_n$

4 Optimization Algorithm

The optimization algorithm we choose is Adam optimizer. Adam optimizer stands for "Adaptive Moment Estimation", it is an iterative optimization algorithm used to minimize the loss function during the training of neural networks. It was introduced in 2014 by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto. Adam combines features from two other optimization methods: RMSprop and Stochastic Gradient Descent (SGD) with momentum. It uses the squared gradients to scale the learning rate like RMSprop, and it takes advantage of momentum by using the moving average of the gradient instead of the gradient itself, like SGD with momentum. This combines Dynamic Learning Rate and Smoothening to reach the global minima, so it has better convergence and stability.

Adam optimization algorithm involves the following steps: (1) Initialize the first and second moments' moving averages (m and v) to zero.

(2) Compute the gradient of the loss function to the model parameters.

(3) Update the moving averages using exponentially decaying averages. This involves calculating m_t and v_t as weighted averages of the previous moments and the current gradient.

(4) Apply bias correction to the moving averages, particularly during the early iterations.

(5) Calculate the parameter update by dividing the bias-corrected first moment by the square root of the bias-corrected second moment, with an added small constant (epsilon) for numerical

stability.
(6) Update the model parameters using the calculated updates.
(7) Repeat steps 2-6 for a specified number of iterations or until convergence.

5 Metrics and Experimental Results

5.1 Metrics Used

5.1.1 Macro F1 Score

Recall (also known as true positive rate) is defined as the ratio of the number of true positive predictions to the total number of actual positive instances:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Precision (also known as positive predictive value) is defined as the ratio of the number of true positive predictions to the total number of positive predictions:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

F1 Score is the harmonic mean of precision and recall, providing a single score that balances both measures:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Macro F1 Score is the average of F1 Scores calculated for each class label in a multi-class classification problem:

$$\text{Macro F1 Score} = \frac{1}{N} \sum_{i=1}^{N} \text{F1 Score}_i$$

where N is the total number of classes and F1 Score$_i$ is the F1 Score for class i.

5.1.2 Accuracy

Accuracy is a common metric used to evaluate the overall performance of a classification model. It measures the ratio of correctly predicted instances to the total number of instances:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

5.2 Experimental Setup

We wanted to experiment with the effectiveness of various strategies to increase the model performance. Hence we decided to keep the following parameters constant[1]:

1. Learning rate: 1e-4
2. Epochs: 20
3. Batch Size: 10
4. Optimizer: Adam
5. AUDIO_FEATURE_EXTRACTOR: MFCC
6. VIDEO_FEATURE_EXTRACTOR: VIT
7. TEXT_FEATURE_EXTRACTOR: BERT

The following parameters were used to assess the model improvement:

1. ReduceLROnPlateau Scheduler
2. Xavier Initialization
3. Increase model complexity
4. Add Residual Blocks

Furthermore, we first calculated the baseline model performance using none of the four model improvement techniques. Then we sequentially added the parameters to check if it improved the model performance. Using Scheduler and Xavier Initialization was straightforward. Only the techniques that improved performance were taken forward in the analysis. Hence the baseline model had three modality-specific layers and one fusion-layer. For increasing model complexity either by adding more linear layers or residual blocks, we tried various number of layers and report the best model achieved. Only for the vision modality, increasing model complexity meant adding more hidden layers in LSTM network and one attention layer to combine the output from all the sequential outputs.

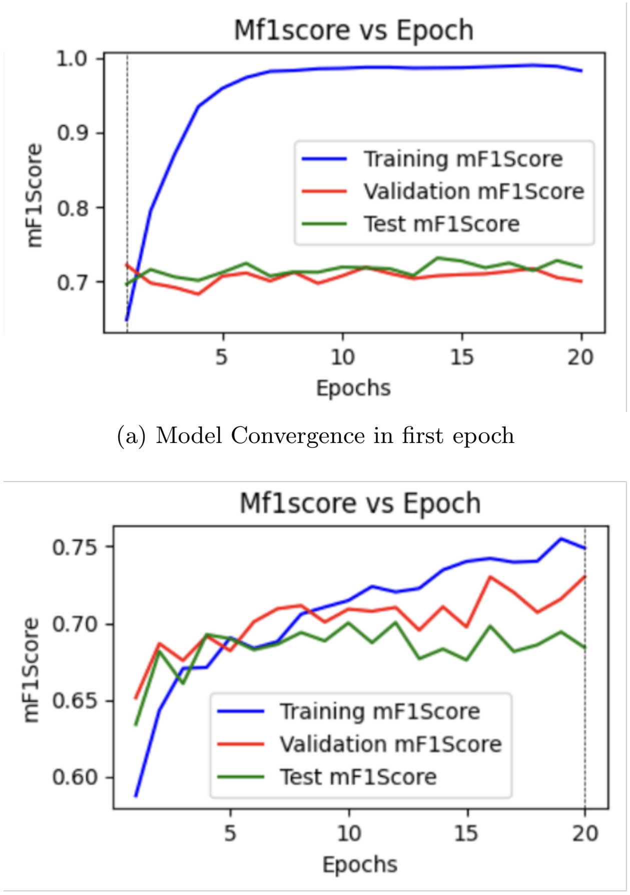
5.3 Results

Initially, the baseline model was training in low epochs, we introduced ways to ensure model's training upto the last epoch (Fig 1). These techniques have been enumerated in 5.2.

Dashboard view was developed to get an overall idea of each model(Fig 2). Only the Macro F1 Score and Accuracy have been reported in Table 1 due to limited space.

6 Contributions and GitHub

Contribution of each teammate was equal 33.33%. The code for the project can be found at xxxxx.



(a) Model Convergence in first epoch

(b) Model continues learning till last epoch

Figure 1: Model Performance Plot Before And After Model Improvement Parameters

References

[1] Mithun Das et al. “Hatemm: A multi-modal dataset for hate video classification”. In: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 17. 2023, pp. 1014–1023.

Modality	Baseline	Reduce LR On Plateau	Xavier Init	Linear Complexity	Residual Blocks
MFCC	71.28% 70.04%	70.82% 69.77%	67.40% 66.27%	71.09% 70.14(2)%	**71.38%** **70.39%(3)**
VIT	70.91% 69.41%	70.91% 69.41%	**74.95%** **72.67%**	72.85% 71.03%(2)	73.96% 72.45%(1)
BERT	76.91% 75.48%	**76.91%** **75.48%**	74.42% 73.13%	71.46% 72.48%(3)	74.24% 72.78(1)%
BERT + MFCC	76.45% 75.49%	**78.48%** **77.09%**	78.48% 77.02%	77.47% 76.32% (3,2)	78.12% 76.77% (1,1)
BERT + VIT + MFCC	78.21% 76.39%	79.22% 78.01%	78.66% 77.49%	77.65% 76.52% (3,2,2)	**79.78%** **78.56% (1,1)**

Table 1: Accuracy and Macro-F1 Score by Model Modalities and Model Improvement technique for Hate Video Classification

7 Appendix

The following dashboard view was developed for each model that was trained, which allowed a holistic intuition about the each model's strengths and weaknesses.

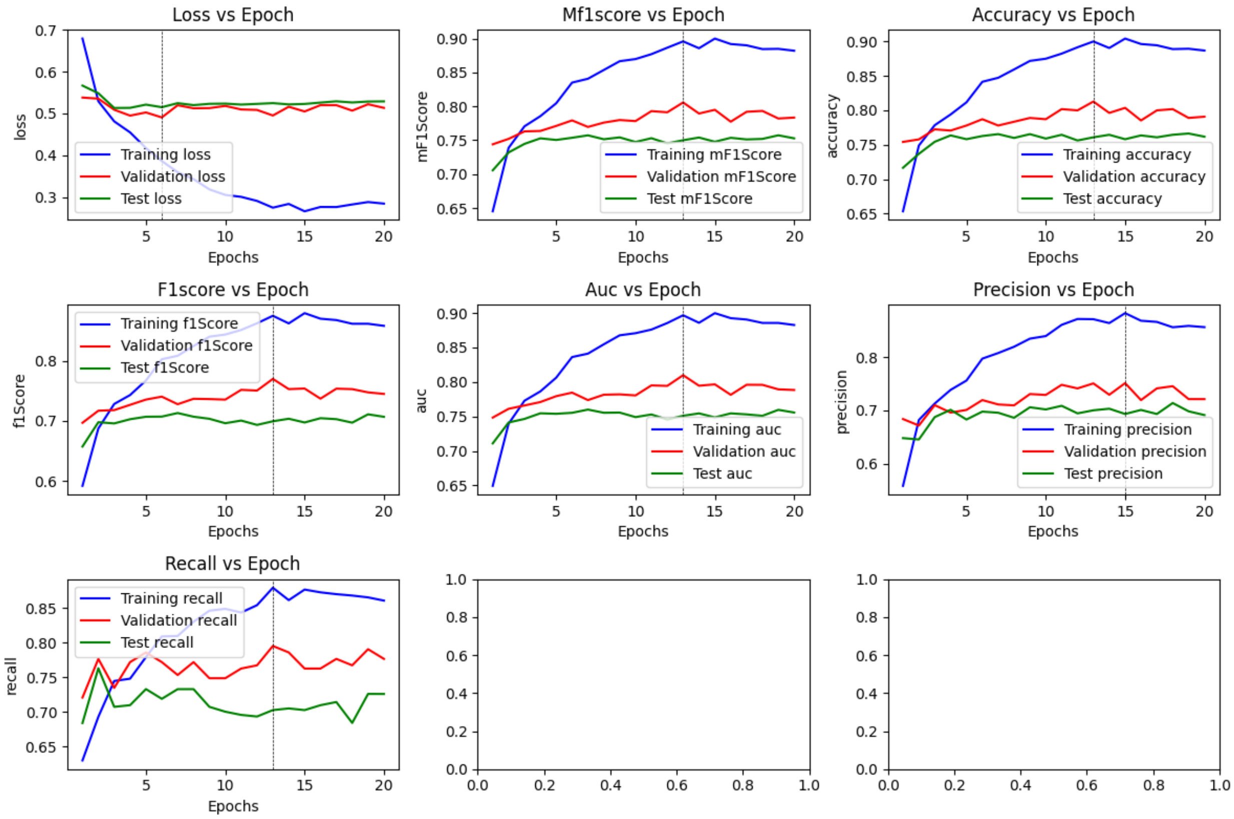


Figure 2: Dasboard View consisting of seven different measures of model performance