FrontEra Report

Aditya Manjunatha

IISc BTech : Mathematics and Computing

Contents

1 Methodology

PLEASE READ THIS REPORT ONLY AFTER READING THE README.MD
FINAL RESULTS AND PLOTS ARE FROM PAGE 3 OF THIS REPORT

1.1 Introduction

Fine-tuning a pretrained model generally involves:

- Loading a pretrained model with learned weights.
- Freezing most layers and unfreezing specific layers to train on a new dataset.
- Replacing the classification head to match the number of classes in our dataset.
- Training the model on the new dataset to adapt it to the new task.

1.2 Dataset and Data Augmentation

One of the key challenges in our project was the **limited number of available datapoints** for certain classes. Specifically, the dataset was highly imbalanced:

- **Cry:** 562 samples
- **Scream:** 800 samples
- **Normal:** 4000 samples (out of which we sampled 900)

Due to this imbalance, the model might develop a bias toward the **majority class** (Normal), leading to poor generalization for underrepresented classes like Cry and Scream. To address this, we applied **data augmentation techniques** to artificially increase the number of training samples and improve model robustness.

1.2.1 Data Augmentation Techniques

For each existing audio file, we applied one of the following transformations to generate **synthetic variations** while preserving the core characteristics of the original sound:

- **Time Stretching:** Speeding up or slowing down the audio slightly while maintaining pitch.
- **Pitch Shifting:** Raising or lowering the pitch within a small range.

- **Adding Background Noise:** Introducing low-level white noise or environmental sounds.
- **Random Cropping:** Extracting random segments from longer audio clips.
- **Volume Perturbation:** Increasing or decreasing the loudness randomly.
- **Reverberation Effects:** Simulating echoes and room effects to create natural-sounding variations.

Each original sample was augmented using **one of these transformations**, effectively **doubling the dataset size** for the underrepresented classes. This **helped balance the dataset**, reduced overfitting, and improved generalization performance.

Impact of Augmentation:

- The increased dataset size allowed better **representation of minority classes** (Cry, Scream).
- The model learned to generalize better by **exposure to diverse variations** of similar sounds.
- Augmentation **introduced robustness** against environmental variations (e.g., background noise, reverberation).

1.3 My Approach: Feature Extraction Instead of Fine-Tuning

This approach can be thought of as a form of fine-tuning, because we are implicitly replacing the classifier head of the pretrained models with a new FFNN.

Given our **limited dataset size** and **compute constraints**, we opted to use **pretrained models as feature extractors** instead of fine-tuning the entire model. The steps involved are:

1. **Extract embeddings from pretrained models (YAMNet and Wav2Vec2).**
2. **Perform mean pooling** on the embeddings to obtain a fixed-size feature vector for each audio file.
3. **Use these feature vectors as input to a Feed-Forward Neural Network (FFNN)** for classification.

1.4 Feature Extraction: YAMNet

- YAMNet is a pretrained model for **audio event classification**.
- Instead of using the full model, we extract embeddings from the **second-last layer** of YAMNet.
- These embeddings have a shape of **(T, 1024)**, where:
  - T is the number of time frames (varies for each audio file).
  - 1024 is the feature dimension.
- To obtain a fixed-size vector, we apply **mean pooling** across time, resulting in a **final feature vector of shape (1024,)**.

1.5 Feature Extraction: Wav2Vec2

- Wav2Vec2 is a self-supervised model trained for **speech representation learning**.
- We extract embeddings from the **Transformer encoder output**.
- These embeddings have a shape of **(T, 768)**, where:
  - T is the sequence length (depends on audio duration).
  - 768 is the feature dimension.
- Similar to YAMNet, we apply **mean pooling** over time, resulting in a **fixed-size vector of shape (768,)**.

1.6 Creating the New Dataset

After extracting embeddings for each audio file:

- We form a new dataset consisting of **feature vectors** instead of raw audio.
- Each data point is now a fixed-size vector of either:
  - **(1024,)** from YAMNet.
  - **(768,)** from Wav2Vec2.
- Labels are mapped as: **Cry (0), Scream (1), Normal (2)**.
- This dataset is then split into **Train (70%) / Validation (15%) / Test (15%)**.

1.7 Training the Feed-Forward Neural Network (FFNN)

WE USE THE CROSS ENTROPY LOSS BEACUSE OF MULTI-CLASS CLASSIFICATION

- Instead of modifying the original models, we define a separate **FFNN classifier**.
- The FFNN architecture consists of:
  - Input Layer: Size **1024** (for YAMNet) or **768** (for Wav2Vec2).
  - Fully Connected Layers: Multiple hidden layers with **ReLU activations**.
  - Dropout layers for regularization.
  - Output Layer: **3 neurons (Softmax activation) for Cry, Scream, and Normal**.
- The FFNN is trained using **cross-entropy loss** and optimized using **Adam optimizer**.
- We perform **K-fold cross-validation** to fine-tune hyperparameters.

1.8 Final Pipeline

1. Preprocess the audio (Convert to 16kHz, PCM16, Mono).
2. Extract embeddings using **YAMNet or Wav2Vec2**.
3. Perform mean pooling to obtain a **fixed-size feature vector**.
4. Pass the feature vector through the trained **FFNN** classifier.
5. Get the final prediction: **Cry, Scream, or Normal**.

1.9 Advantages of this Approach

- **Computational Efficiency:** No need to fine-tune large pretrained models.
- **Data Efficiency:** Avoids overfitting on a small dataset.
- **Flexibility:** The same FFNN can be trained with embeddings from different models.

2 K-Fold Cross-Validation Results

To ensure that our models generalize well to unseen data, we performed **5-Fold Cross-Validation** on different combinations of hyperparameters for both YAMNet and Wav2Vec2.

2.1 Hyperparameter Evaluation for YAMNet

The following table summarizes the cross-validation results for different hyperparameter settings on the **YAMNet-based classifier**:

Combo	Learning Rate	Epochs	Batch Size	Avg Accuracy
1	0.001	30	32	0.9752
2	0.0005	40	64	0.9730
3	0.01	50	16	0.9416
4	0.002	25	32	0.9682
5	0.0003	35	64	**0.9759**

Table 1: 5-Fold Cross-Validation Accuracy for YAMNet Model

From the results, we observe:

- **Combo 5 (LR = 0.0003, Batch Size = 64, Epochs = 35)** achieved the **highest accuracy** of **97.59%**.
- The other combinations also performed well, consistently above **94%**, indicating that the model is robust across different hyperparameters.
- The **third combination (LR = 0.01, Epochs = 50, Batch Size = 16)** showed a **drop in accuracy (94.16%)**, suggesting that a higher learning rate may have led to instability in training.

Thus, we selected **Combo 5** as the optimal hyperparameter configuration for full training.

—

2.2 Hyperparameter Evaluation for Wav2Vec2

We performed the same 5-Fold Cross-Validation for the **Wav2Vec2-based classifier**:

Key Observations:

- **Combo 2 (LR = 0.0005, Batch Size = 64, Epochs = 40)** achieved the **best accuracy of 91.53%**.
- The Wav2Vec2 classifier showed slightly lower accuracy compared to YAMNet, suggesting that **Wav2Vec2 embeddings might not capture the relevant features as well for this task**.

Combo	Learning Rate	Epochs	Batch Size	Avg Accuracy
1	0.001	30	32	0.9074
2	0.0005	40	64	**0.9153**
3	0.01	50	16	0.8926
4	0.002	25	32	0.9037
5	0.0003	35	64	0.8944

Table 2: 5-Fold Cross-Validation Accuracy for Wav2Vec2 Model

- Again, **Combo 3 (LR = 0.01, Epochs = 50, Batch Size = 16) had the lowest accuracy (89.26%)**, indicating instability due to high learning rate.

Thus, we selected **Combo 2** as the optimal hyperparameter configuration for full training.

—

2.3 Conclusion

- The **high validation accuracy across all folds** indicates that our models are **not overfitting**, but rather learning meaningful representations of the data.
- The selected hyperparameters were then used to train the models on the **entire training dataset (70%)** and evaluate them on the **test set (15%)**.

3 Training and Validation Results

After selecting the best hyperparameters from the **K-Fold Cross-Validation**, we proceeded with full training on the **70% training set** and evaluated on the **15% validation set**.

To monitor performance, we tracked:

- **Accuracy vs. Epochs** - Helps understand how well the model is learning over time.
- **F1-Score vs. Epochs** - Provides insights into the balance between precision and recall.

We trained both **YAMNet-based** and **Wav2Vec2-based** classifiers and visualized their training dynamics.

—

3.1 YAMNet Training Results

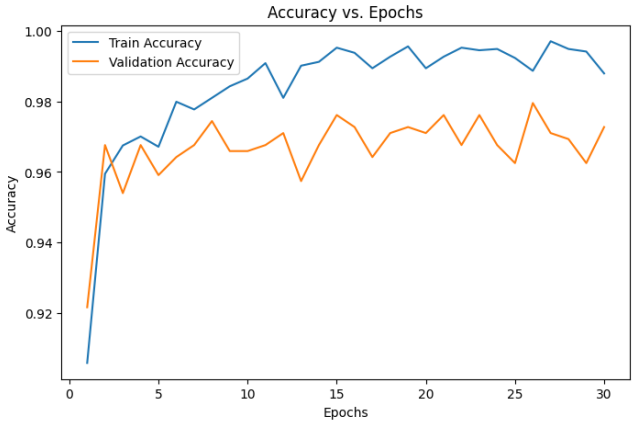

Figure 1: Accuracy vs. Epochs for YAMNet Classifier

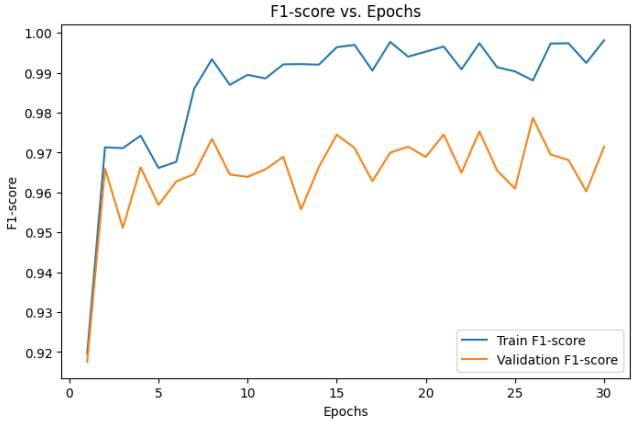

Figure 2: F1-Score vs. Epochs for YAMNet Classifier

3.2 Wav2Vec2 Training Results

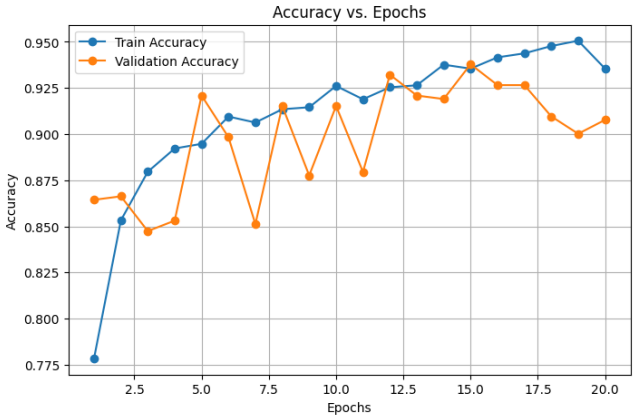

Figure 3: Accuracy vs. Epochs for Wav2Vec2 Classifier

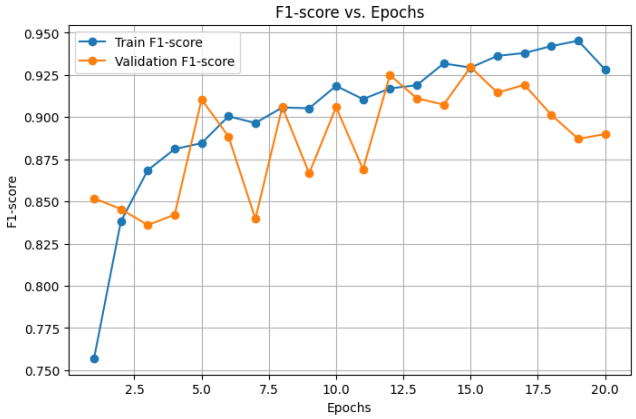

Figure 4: F1-Score vs. Epochs for Wav2Vec2 Classifier

4 Test Set Classification Reports

After training and validation, we evaluated both models on their respective **test sets** to assess their final performance.

For each model, we present the **Test Accuracy, Precision, Recall, and F1-Score**, along with observations on their effectiveness.

—

4.1 YAMNet Test Results

Final Test Accuracy: 97.79%

Class	Precision	Recall	F1-Score	Support
Cry	0.96	0.97	0.97	146
Scream	0.97	0.97	0.97	216
Normal	1.00	0.99	0.99	226
Accuracy		0.98		588
Macro Avg	0.98	0.98	0.98	588
Weighted Avg	0.98	0.98	0.98	588

Table 3: Classification Report for YAMNet

—

4.2 Wav2Vec2 Test Results

Final Test Accuracy: 91.97%

Class	Precision	Recall	F1-Score	Support
Cry	0.96	0.75	0.84	129
Scream	0.84	0.97	0.90	175
Normal	0.98	0.99	0.99	194
Accuracy		0.92		498
Macro Avg	0.93	0.90	0.91	498
Weighted Avg	0.93	0.92	0.92	498

Table 4: Classification Report for Wav2Vec2

—

5 Confusion Matrices

To further analyze the performance of our models, we visualize the confusion matrices for both **YAMNet** and **Wav2Vec2** on the test set.

Confusion matrices provide insights into **misclassifications** and help us understand which classes are more prone to errors.

5.1 Confusion Matrix for YAMNet

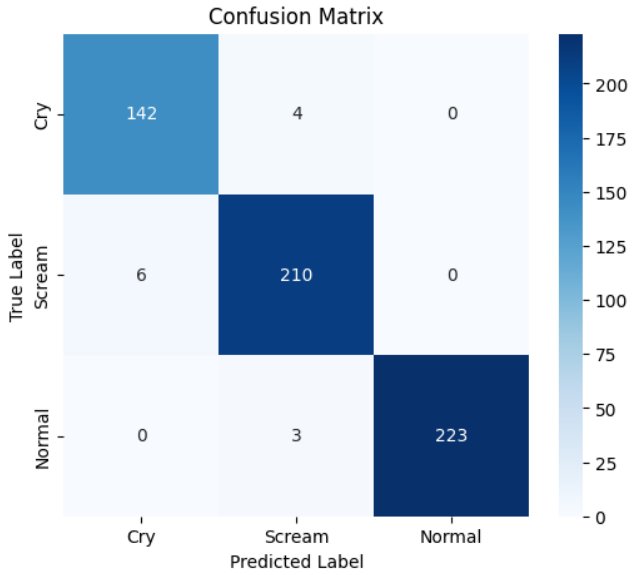

Figure 5: Confusion Matrix for YAMNet

5.2 Confusion Matrix for Wav2Vec2

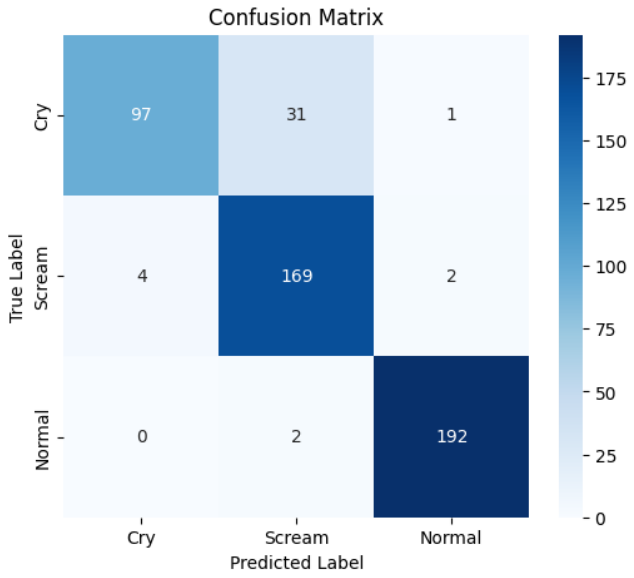

Figure 6: Confusion Matrix for Wav2Vec2

6 ROC-AUC Curves

To further evaluate our models' performance, we visualize the Receiver Operating Characteristic (ROC) curves along with the Area Under the Curve (AUC)** for both YAMNet and Wav2Vec2.

ROC curves help analyze the trade-off between true positive rate (sensitivity) and false positive rate, providing insights into the model's ability to distinguish between the three classes.

6.1 ROC-AUC Curve for YAMNet

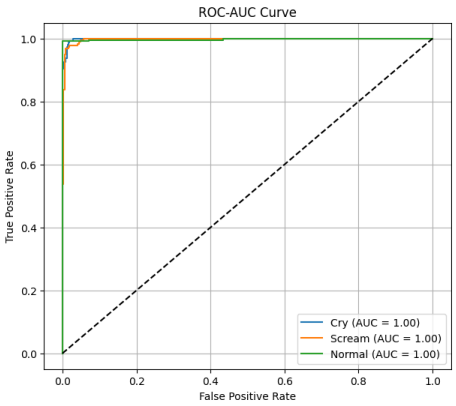

Figure 7: ROC-AUC Curve for YAMNet

6.2 ROC-AUC Curve for Wav2Vec2

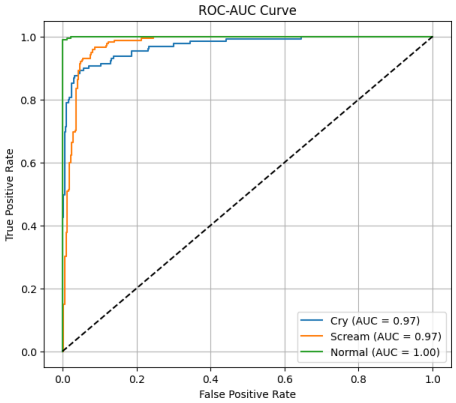

Figure 8: ROC-AUC Curve for Wav2Vec2

6.3 In-depth Observations from the Plots

After examining the accuracy vs. epochs, F1-score vs. epochs, confusion matrices, and ROC-AUC curves for both **YAMNet** and **Wav2Vec2**, we make the following detailed observations:

- **YAMNet Accuracy vs. Epochs**
  - The training accuracy quickly climbs above 98% within the first 10 epochs and stabilizes around 99% by epoch 20.
  - The validation accuracy closely follows the training curve, indicating **minimal overfitting**.
  - The small gap between training and validation accuracies suggests that the model generalizes well to unseen data.
  - By the final epochs, the model saturates near **98–99% accuracy**, reflecting a robust and stable learning process.
- **YAMNet F1-Score vs. Epochs**
  - The F1-scores for both training and validation remain high, typically above 97%.
  - Fluctuations are minimal, implying that **precision and recall are balanced** across classes (Cry, Scream, Normal).
  - The model converges quickly, showing no significant F1 degradation in later epochs.
- **YAMNet Confusion Matrix**
  - Very few misclassifications overall. Most errors occur between **Cry** and **Scream**, but the numbers are small (single digits).
  - **Normal** class is almost perfectly recognized, reflecting its distinct acoustic characteristics.
  - The high diagonal values in the confusion matrix confirm the model's strong discriminative ability.
- **YAMNet ROC-AUC Curve**
  - The AUC for all three classes is at or near **1.00**, indicating an almost perfect separation between Cry, Scream, and Normal.
  - This result corroborates the high accuracy and F1-scores observed, reinforcing the notion that YAMNet embeddings capture the relevant features very effectively.
- **Wav2Vec2 Accuracy vs. Epochs**
  - The training accuracy starts lower (around 75–80%) but improves steadily to around 92–95% by epoch 15–20.

  - Validation accuracy follows a similar trend, albeit with slightly more fluctuation, indicating some sensitivity to hyperparameter changes.
  - The final gap between training and validation accuracies remains small, suggesting decent generalization.

- **Wav2Vec2 F1-Score vs. Epochs**
  - Early epochs show noticeable variance in F1-score, especially in the validation curve, implying the model initially struggles to classify minority classes accurately.
  - By epoch 15+, both training and validation F1-scores converge to around 90–95%, demonstrating improved balance between precision and recall.

- **Wav2Vec2 Confusion Matrix**
  - **Cry** class exhibits the highest misclassification rate, often being confused with **Scream**.
  - **Normal** class remains consistently well-classified, reflecting its distinct vocal profile.
  - The confusion matrix indicates that Wav2Vec2 embeddings are slightly less effective in separating the subtle differences between Cry and Scream compared to YAMNet.

- **Wav2Vec2 ROC-AUC Curve**
  - The AUC values hover around **0.97** for Cry and Scream, while Normal approaches 1.00.
  - Although still high, the model's ability to differentiate Cry from Scream is not as perfect as YAMNet's near-1.00 performance.
  - These results align with the confusion matrix observations, confirming that Wav2Vec2 embeddings occasionally blend Cry and Scream features.

Overall Analysis:

- **YAMNet** demonstrates consistently higher performance, with near-perfect ROC-AUC and minimal misclassifications.
- **Wav2Vec2** performs strongly overall but struggles more with distinguishing Cry from Scream, as evidenced by its confusion matrix and slightly lower ROC-AUC for those classes.
- Both models show **high accuracy** and **balanced precision-recall**, indicating a robust classification pipeline despite the inherent challenges of detecting subtle audio differences.
- These insights motivate our decision to explore **ensemble methods**, aiming to combine the strengths of both YAMNet and Wav2Vec2.