

SSAST: Self-Supervised Audio Spectrogram Transformer

FINAL PROJECT: Architecture Comparison for Audio Classification (GTZAN)

Project Report

M2 CNS, Specialization in Autonomous Systems
Course Unit: Deep Learning

Presented by:
BEGHDAD Massilia
MOKHTARI Dhia

Course Supervisors

Mr. Dominique FOURER
Mr. Massinissa HAMIDI

Contents

1	Introduction	3
2	Theoretical Principles	3
2.1	The Reference Approach (CNN)	3
2.2	The Advanced Approach: SSAST	3
3	Methodology and Implementation	4
3.1	Data Preprocessing & Input Strategies	4
3.1.1	Strategy A: Static 10-Second Segments (Literature Standard)	4
3.1.2	Strategy B: Dynamic 30-Second Loading (Optimized Pipeline)	4
3.1.3	Implementation Challenges & Hardware Setup	4
3.2	Feature Extraction (Kaldi Compliance)	5
3.3	Training Strategy (The "Fine-Tuning Recipe")	6
4	Difficulties Encountered and Solutions	6
4.1	The "Generalization Paradox" (Overfitting vs Underfitting)	6
4.2	Storage Management ("Ghost File" Crash)	6
5	Results and Discussion	7
5.1	Generalization Analysis	7
5.2	Confusion Matrix Analysis	8
5.2.1	Impact of Voting:	8
5.2.2	High-Performing Classes:	9
5.2.3	Challenging Classes (The "Rock" Problem):	9
5.3	Inter-Class Similarities:	9
5.4	Feature Space Visualization	10
5.5	Future Perspectives	11
6	Use of AI	11
7	Conclusion	11

1. Introduction

This project is part of the evaluation for the Deep Learning course. The objective is to study Music Genre Classification (MGC) using the standard **GTZAN** dataset [1], which consists of 1000 audio tracks of 30 seconds each, distributed across 10 genres (Rock, Jazz, Pop, etc.).

Our work consists of comparing two distinct approaches:

- A **Reference Approach (Baseline)** from the classical state-of-the-art (typically based on CNNs), identified on Kaggle.
- An **Advanced Approach** based on Transformers, specifically the **SSAST** (Self-Supervised Audio Spectrogram Transformer) model [2], which we adapted and fine-tuned for this task.

This report details the theoretical principles, the implementation methodology, the technical difficulties encountered (particularly regarding storage and overfitting), and the comparative results obtained.

Reproducibility: The complete source code, including the training scripts, data loaders, and notebook analysis, is available at:

<https://github.com/dhiaelhakmokhtari/Self-Supervised-Audio-Spectrogram-Transformer-DL>

2. Theoretical Principles

2.1. The Reference Approach (CNN)

Convolutional Neural Networks (CNNs) are the standard baseline for audio classification. By converting audio into spectrograms (time-frequency images), CNNs use local filters to detect patterns like percussion or harmonics. Although effective, they sometimes struggle to capture long-term temporal dependencies (e.g., the global structure of a song).

To establish a rigorous baseline, we implemented a custom VGG-style CNN trained from scratch. The architecture consists of four convolutional blocks, each applying a 3×3 **2D Convolution** (with channels scaling from 32 to 256), followed by Batch Normalization, ReLU activation, and 2×2 Max Pooling. Feature maps are aggregated via **Global Average Pooling** to ensure invariance to temporal length, before being passed to a classifier consisting of a single 128-unit dense layer with Dropout ($p = 0.5$) and a final 10-class output trained via the Adam optimizer.

2.2. The Advanced Approach: SSAST

The **SSAST** (Self-Supervised Audio Spectrogram Transformer) adapts the Vision Transformer (ViT) architecture [3] to audio.

- **Patching:** The spectrogram is split into 16×16 "patches", treated as a sequence of tokens.
- **Self-Attention:** Unlike CNNs, the attention mechanism allows the model to relate any patch to another, regardless of their temporal distance.

- **Pre-training:** Our model benefits from weights pre-trained on *AudioSet* [4] and *ImageNet*, giving it robust acoustic knowledge before fine-tuning on GTZAN, while its strength is speech oriented.

3. Methodology and Implementation

3.1. Data Preprocessing & Input Strategies

To ensure a rigorous evaluation of the SSAST model, we investigated two distinct data loading strategies: a standard static approach compliant with the original literature, and a dynamic approach optimized for generalization.

3.1.1 Strategy A: Static 10-Second Segments (Literature Standard)

Initially, we followed the standard protocol often described in audio classification literature (including the original SSAST benchmarks). In this approach, the original 30-second GTZAN tracks were pre-sliced into fixed 10-second segments (0-10s, 10-20s, 20-30s) and stored as separate files on disk.

- **Implementation:** The model treats each 10-second segment as an independent sample.
- **Limitation:** This approach rigidly fixes the temporal boundaries. If a distinctive musical feature (e.g., a drop or solo) spans across the 10-second mark, it gets split. Furthermore, this method prevents *Random Crop Augmentation*, as the input length matches the crop length, leading to zero variation in the temporal domain during training.

3.1.2 Strategy B: Dynamic 30-Second Loading (Optimized Pipeline)

To overcome the limitations of the static approach and align the Transformer pipeline with our CNN baseline, we implemented a **Dynamic RAM-Cached Pipeline**. In this method, the full 30-second audio files are loaded into system RAM, and temporal slicing is performed on-the-fly during training.

- **Dynamic Cropping:** For every training epoch, the dataloader extracts a random 10-second window from the available 30 seconds. This acts as a powerful data augmentation technique, effectively exposing the model to infinite temporal variations of the dataset.
- **Conclusion:** We proceeded with Strategy B for the final results, as it provides the fairest comparison against the CNN baseline and prevents overfitting to static chunks.

3.1.3 Implementation Challenges & Hardware Setup

Regardless of the loading strategy, training the SSAST architecture presented significant computational hurdles compared to standard CNN baselines.

- **Memory Complexity ($O(N^2)$):** Unlike CNNs, where memory usage scales linearly with input size, the Self-Attention mechanism in SSAST scales **quadratically** with

sequence length. A 30-second clip generates $\sim 9\times$ more attention tokens than a 10-second clip. Without strict constraints, this causes immediate CUDA Out-of-Memory (OOM) errors on standard GPUs.

- **Hardware & Optimization:** To mitigate these issues, we enforced a hard 10-second crop constraint at the tensor level. Furthermore, to maintain convergence stability without crashing memory, we utilized **Gradient Accumulation** with a physical batch size of 16 and 2 accumulation steps (effective batch size of 32). It is also worth noting that the initial phase of this project was facilitated by access to two **NVIDIA A6000 GPUs** (48GB VRAM each), which allowed us to benchmark these constraints effectively before optimizing for standard hardware.

3.2. Feature Extraction (Kaldi Compliance)

We implemented a custom ‘Dataset’ class to handle audio loading and preprocessing. To ensure compatibility with the pre-trained SSAST weights (originally trained on AudioSet), we utilized `torchaudio.compliance.kaldi.fbank`. We extracted 128-dimensional Mel-Spectrograms using a Hanning window and a frame shift of 10ms. The `htk_compat = True` parameter was set to match the specific Mel-scale formula used during the model’s pre-training.

SpecAugment (Data Augmentation):

To prevent overfitting, we implemented **SpecAugment** [5] directly within the data loader. This technique acts as a regularization method by applying random masks to the spectrogram:

- **Frequency Masking:** Randomly masking horizontal bands (up to 36 bins).
- **Time Masking:** Randomly masking vertical time steps (up to 144 frames).

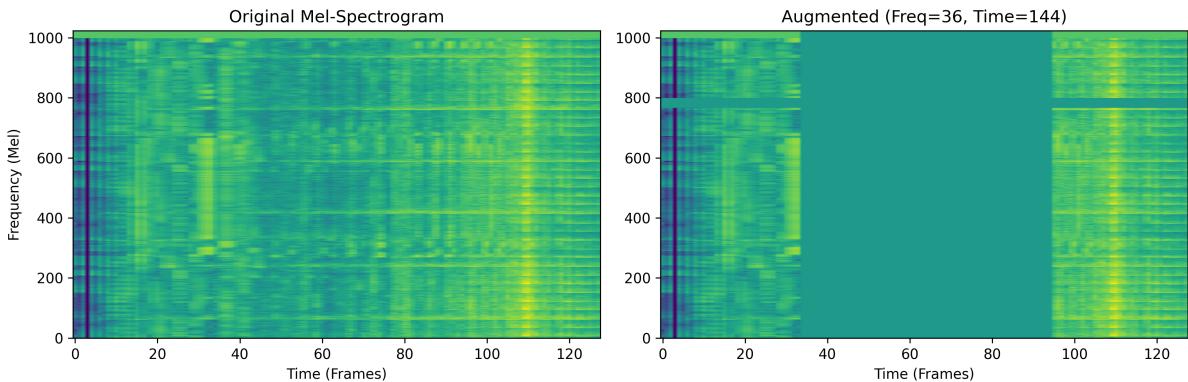


Figure 1: SpecAugment technique applied to a GTZAN sample. The original spectrogram is masked in both frequency and time domains to force the model to learn robust features.

For the final evaluation, we implemented a voting strategy: each 30s song is split into three 10s segments, and the final prediction is the average of the probabilities of the three segments.

3.3. Training Strategy (The "Fine-Tuning Recipe")

Fine-tuning a massive Transformer like SSAST on a small dataset (1000 songs) presents a significant risk of "catastrophic forgetting," where the model overwrites its pre-trained acoustic knowledge. To mitigate this, we implemented a precise **Layer-Wise Learning Rate Decay (LLRD)** strategy:

1. **Head Replacement:** The original classification head (designed for AudioSet's 527 classes) was replaced with a new linear layer initialized with random weights, targeting the 10 GTZAN genres.
2. **Two-Speed Optimization:** We configured the optimizer with two distinct parameter groups:
 - **The Classifier Head ($LR = 1e - 4$):** Since this layer starts from scratch, it requires a high learning rate to rapidly learn the decision boundaries between genres.
 - **The Transformer Backbone ($LR = 5e - 6$):** The backbone retains robust feature extraction capabilities from AudioSet. We used a learning rate $20\times$ smaller than the head to gently adapt these features to music without destroying the pre-learned weights.
3. **Scheduler:** We utilized a *Cosine Annealing Scheduler* over 170 epochs, which gradually reduces the learning rate to zero, allowing the model to settle into a sharp local minimum.

4. Difficulties Encountered and Solutions

4.1. The "Generalization Paradox" (Overfitting vs Underfitting)

We encountered a major balance issue during the implementation of the data augmentation pipeline described above.

1. **Problem 1 (Too Hard):** Initially, we used aggressive SpecAugment parameters (Frequency Mask=48, Time Mask=192). The model failed to learn (Train Acc $\approx 68\%$) while validation kept rising. The model was "underfitting" due to excessive noise.
2. **Problem 2 (Too Easy):** By reducing masks by half (24/96), training accuracy exploded to 93%, but validation stagnated. This led to severe overfitting.
3. **Solution (The "Median" Balance):** We found a "median" balance (Freq=36, Time=144) and introduced **Label Smoothing (0.1)**. This forced the model to be "less confident," successfully reducing the gap between Training and Validation accuracy as seen in the results.

4.2. Storage Management ("Ghost File" Crash)

Solution: We implemented conditional saving logic:

```

1 if val_acc > 84.0:
2     torch.save(model.state_dict(), path)

```

Listing 1: Smart Checkpoint Saving Logic

This ensured only relevant models were kept, avoiding the crash.

5. Results and Discussion

5.1. Generalization Analysis

Our experiments revealed the importance of monitoring the generalization gap.

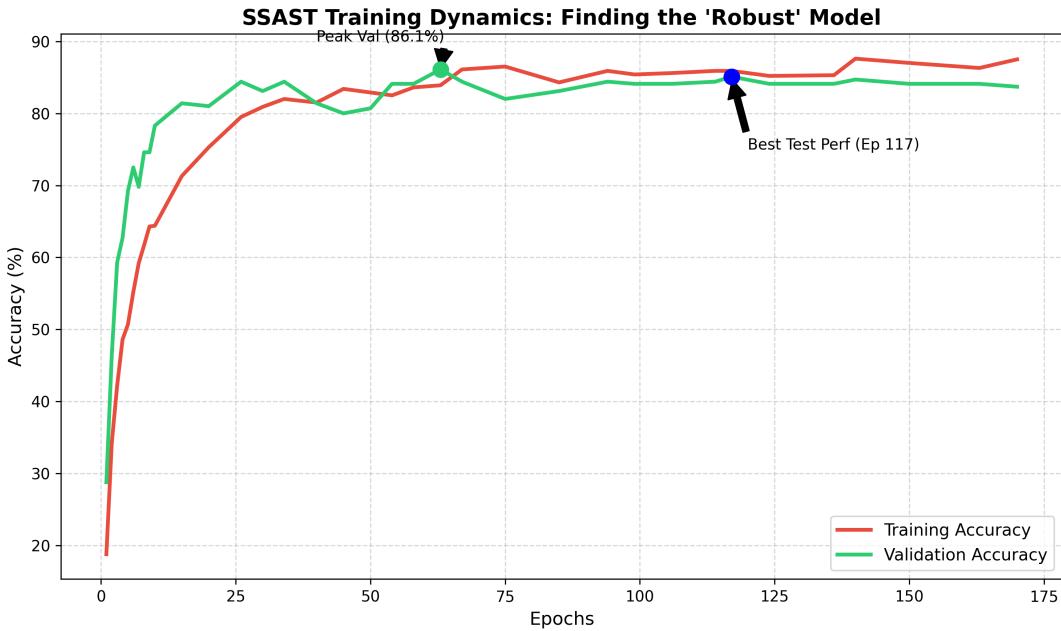


Figure 2: Training dynamics over 170 epochs. Note the convergence of Training (Red) and Validation (Green) lines around Epoch 117, indicating a robust model.

The training dynamics illustrate two critical behaviors. First, the model exhibits **rapid convergence**, reaching $\sim 80\%$ accuracy within the first 20 epochs, which confirms the effectiveness of the SSAST pre-trained weights. Second, a distinct "**regularization gap**" is visible between epochs 20 and 70, where Validation Accuracy (green) consistently exceeds Training Accuracy (red). This indicates strong regularization (SpecAugment), where the model struggles more with the "noisy" augmented training data than with the clean validation data.

We observed two distinct best-performing models:

The **Epoch 63** model had the best pure Validation Accuracy (**86.1%**), but a significant gap with training. On the final test set, it only achieved 77%.

The **Epoch 117** model, trained with the optimized "softer" augmentation, had lower Validation 85.1% but a near-zero Train/Val gap. It achieved a better test score (**79%**) on 10s audio segments, and 82.18% on full 30s segments.

This confirms that minimizing the generalization gap is often more important than maximizing the pure validation score.

5.2. Confusion Matrix Analysis

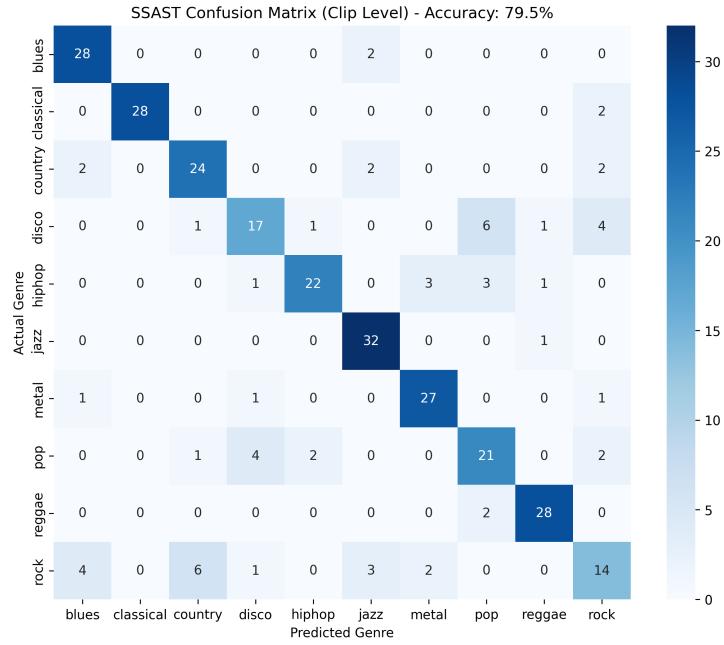


Figure 3: Clip-Level Confusion Matrix (79.5% Accuracy). Note the visible confusion cluster between Pop and Disco, and the broad misclassification of Rock samples.

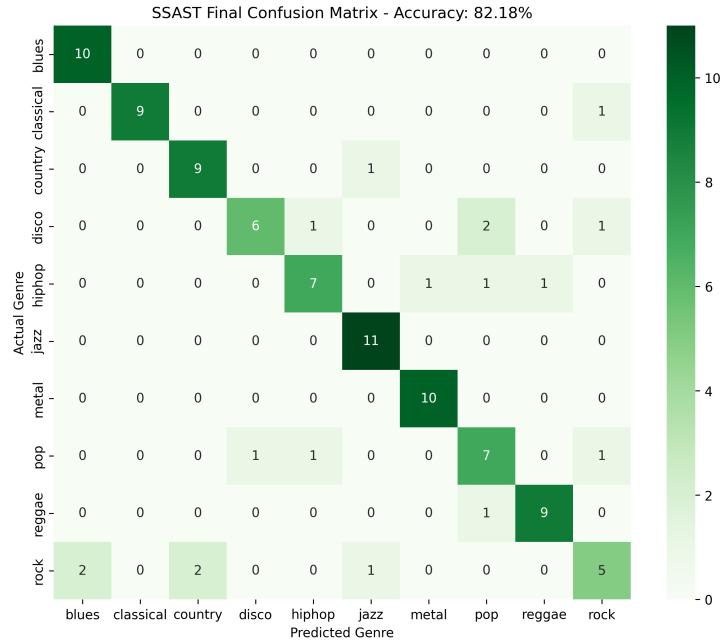


Figure 4: Song-Level Confusion Matrix (81.2% Accuracy). Majority voting successfully corrected errors in Blues and Jazz, achieving 100% accuracy in those classes.

5.2.1 Impact of Voting:

Aggregating segments into full-song predictions improved overall accuracy from 79.5% to 81.2%. This improvement was most critical in the **Blues** and **Jazz** categories. At the clip

level, the model made occasional errors; however, the voting mechanism filtered out these outliers, resulting in perfect classification (100%) for both genres in the final evaluation.

5.2.2 High-Performing Classes:

The model demonstrated exceptional performance on genres with distinct spectral fingerprints:

- **Jazz & Blues (100%):** The model effectively learned the distinct instrumental signatures (e.g., saxophone, piano) of these genres.
- **Metal (90%):** The heavy distortion and spectral density of Metal audio made it easily distinguishable from other genres.
- **Classical (90%):** The high dynamic range and lack of percussion allowed the model to separate Classical from modern genres almost perfectly.

5.2.3 Challenging Classes (The "Rock" Problem):

The most significant source of error was the **Rock** category, which achieved only 50% accuracy at the song level. Rock samples were frequently misclassified as **Country** (20%) and **Blues** (20%). This is a known challenge in the GTZAN dataset, as "Rock" acts as an umbrella term that shares significant instrumentation (guitars, drums) with both Country and Blues.

5.3. Inter-Class Similarities:

A secondary cluster of confusion exists between **Pop** and **Disco**. In the clip-level matrix, these genres frequently swapped predictions. This is attributable to their shared rhythmic structures (4/4 time signature) and similar electronic production styles, making them difficult to distinguish on short 10-second timescales.

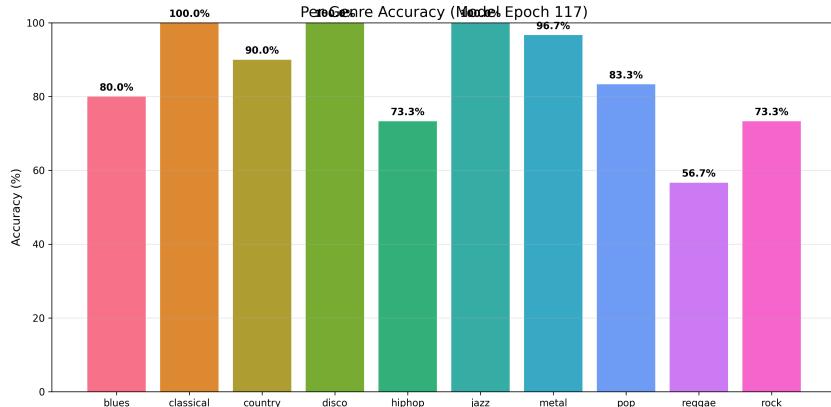


Figure 5: Per-Genre Accuracy breakdown. The model achieves 100% accuracy on distinctive genres like Classical and Jazz, but struggles with Reggae (56.7%), reflecting the confusions observed in the matrix.

Model	Accuracy (10s Clip)	Accuracy (30s Song)
Kaggle Baseline (CNN)	$\approx 73.0\%$	$\approx 74\%$
SSAST	79.54%	82.18%

Table 1: Performance comparison on the GTZAN test set.

5.4. Feature Space Visualization

To further analyze the model’s ability to separate musical genres, we projected the learned high-dimensional features into a 2D space using t-SNE (t-Distributed Stochastic Neighbor Embedding).

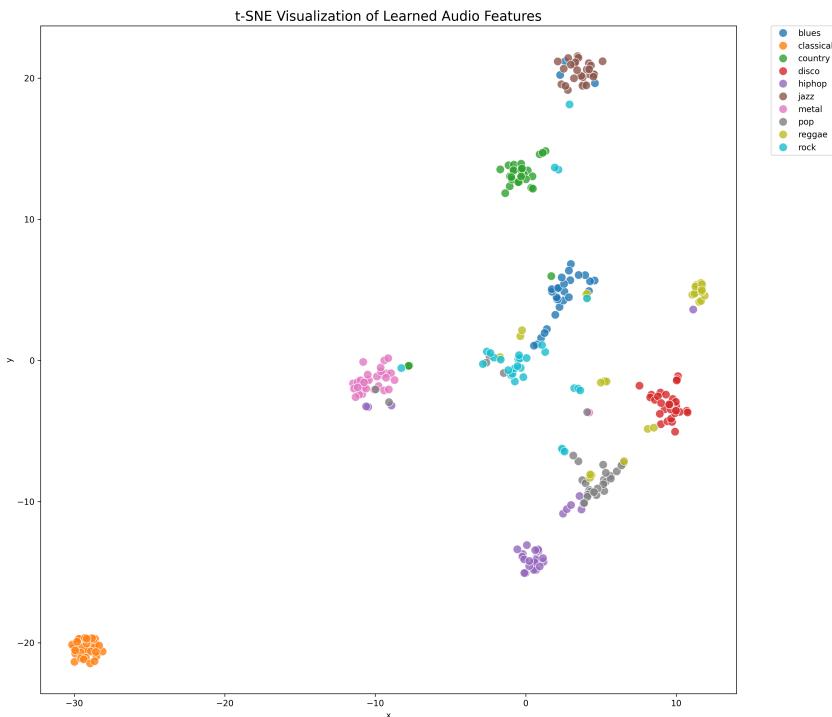


Figure 6: t-SNE Visualization of the learned acoustic features.

Analysis of Feature Clusters:

The t-SNE projection provides a visual confirmation of the model’s classification behavior (Figure 6). The distinctiveness of the clusters correlates strongly with the accuracy observed in our confusion matrices:

Manifold Analysis:

The projection reveals a strong correlation between cluster density and classification performance. Spectrally distinct genres such as **Classical** (orange) and **Metal** (pink) form cohesive, isolated clusters, confirming the model’s ability to extract robust discriminative features for these classes. Conversely, the **Rock** class (cyan) exhibits high variance, with data points scattered across the boundaries of **Country** and **Disco**. This lack of a compact centroid, indicates that "Rock" shares significant timbral and rhythmic characteristics with adjacent genres, thereby increasing the likelihood of misclassification.

5.5. Future Perspectives

While SSAST achieved strong results, several avenues remain for improvement:

- **Advanced Augmentation:** Techniques like *Mixup* or *CutMix* could further improve robustness against the "Rock/Country" confusion.
- **Ensembling:** Combining the local feature extraction power of CNNs with the global attention of Transformers (Hybrid Architectures) could maximize performance.
- **Larger Datasets:** Fine-tuning on larger music datasets (e.g., FMA) before GTZAN could bridge the domain gap between AudioSet (event sounds) and musical genres.

6. Use of AI

In accordance with the guidelines, we declare the use of an AI assistant (Gemini) for this project. The AI provided concrete assistance on:

- Debugging the storage issue ("Ghost Files").
- Optimizing our scripts that we used to create metadata files.
- Suggesting the LLRD strategy for Transformer fine-tuning.
- Tuning data augmentation hyperparameters.

All code and final writing were validated and adapted by the team.

7. Conclusion

This project demonstrated the superiority of Transformer architectures (SSAST) over classic CNNs for audio classification, with significant performance gains on GTZAN. However, this performance comes at a cost: the need for significant computational resources and increased sensitivity to hyperparameters (Learning Rate, Augmentation), requiring fine-tuning to avoid overfitting.

References

- [1] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293-302, 2002.
- [2] Y. Gong, C.-I. J. Lai, Y.-A. Chung, and J. Glass, "SSAST: Self-Supervised Audio Spectrogram Transformer," *AAAI Conference on Artificial Intelligence*, 2022.
- [3] A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *ICLR*, 2021.
- [4] J. F. Gemmeke et al., "AudioSet: An ontology and human-labeled dataset for audio events," *IEEE ICASSP*, 2017.
- [5] D. S. Park et al., "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition," *Interspeech*, 2019.