Adaptive Model Optimization for Audio Classification on Edge Devices

Yuhe (Alice) Hu, Emily Shao, Yash Johar
Course: ECE661, Fall 2024
`alice.hu@duke.edu | emily.shao@duke.edu | yash.johar@duke.edu`

Abstract

Deploying deep learning models on edge devices presents challenges due to constraints in computational power, memory, and energy consumption. This study focuses on adaptive model optimization techniques to enhance the efficiency of audio classification models for resource-limited environments. Specifically, we explore methods to reduce the size and complexity of a ResNet-152 model trained on the GTZAN dataset for music genre classification while maintaining performance. Initial experiments with the ResNet-152 unoptimized model highlight latency and resource bottlenecks on a Raspberry Pi module. We investigate dynamic, FP16 and INT8 quantization techniques to compress the model while preserving classification accuracy. The results demonstrate significant improvements in inference speed and memory usage with minimal degradation in performance. These findings emphasize the potential of adaptive optimization strategies for deploying deep learning models on edge devices, facilitating efficient and accessible audio classification in low-resource settings.

1 Introduction

Deep learning models have revolutionized various domains, including computer vision, speech recognition, and natural language processing, enabling breakthroughs in automation and intelligence. However, as these models grow in size and complexity, they impose high computational demands, limiting their deployment on resource-constrained devices. Recently developed models contain billions to trillions of parameters, demanding substantial computational resources. For instance, the NVIDIA MegatronLM, a transformer model with 8.3 billion parameters, requires 33 GB of disk space and approximately 10 days of training on 512 V100 GPUs[2]. These models not only demand large computational resources but also create accessibility barriers for people trying to employ these models due to the added complexity of the models.

To address these limitations, researchers have focused on optimizing models for edge devices through lightweight architectures, model compression, and hardware-aware adaptations. Optimizing models for smaller and faster deployment remains an active area of research. Lightweight architectures such as MobileNet[3] and SqueezeNet[4] have been developed specifically for mobile deployment. Additionally, models like Perplexity AI have optimized mobile versions that maintain robustness despite their reduced size.

Audio classification on edge devices remains an underexplored area compared to other applications like image classification and speech recognition. While computer vision models have been widely optimized for real-time mobile inference, relatively few studies have focused on deploying deep learning-based music classification on low-power devices. Music genre classification poses unique challenges due to variability in musical styles, overlapping frequency components, and temporal dependencies. Existing research primarily focuses on large-scale cloud-based models, which limits accessibility for real-time applications on embedded systems such as IoT devices, hearing aids, and smart speakers[6].

In this project, we explore an approach to optimizing a deep learning model for audio classification, focusing on reducing the size and complexity of ResNet-152[5] for deployment on a Raspberry Pi module. Our model is trained on the GTZAN dataset[1], which consists of music classified into 10 genres. We used the ResNet-152 model as our baseline, and we modified it for genre classification of the 10 different audio genres seen in the GTZAN dataset.

2 Methodology and Workflow

This project aims to deploy a ResNet-152 model for GTZAN audio classification on a resource-constrained device like the Raspberry Pi. The workflow involves data preprocessing, model selection, optimization techniques, and deployment.

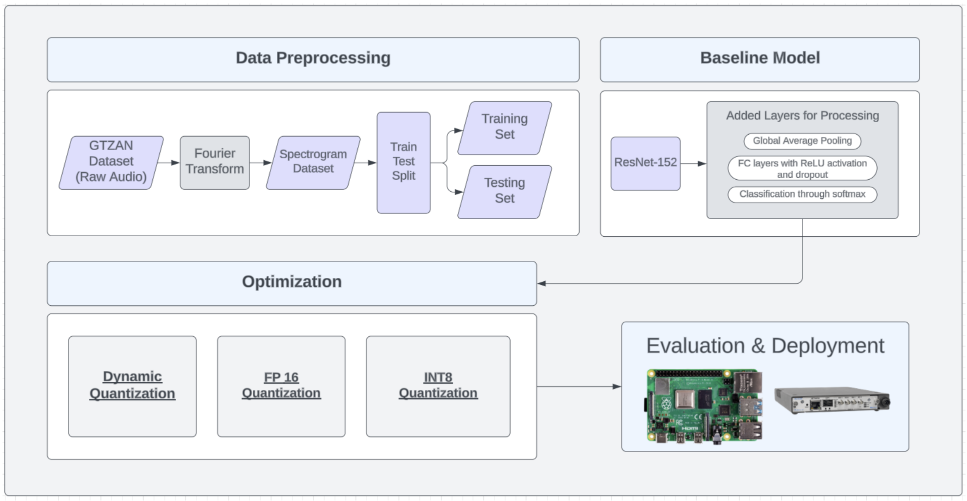

Figure 1: Workflow of Methodology

2.1 Data and Data Preprocessing

Our study uses the publicly available GTZAN dataset, which contains 1,000 music tracks (30 seconds each) evenly distributed across 10 genres. Since raw audio signals exist in the time domain, they were first transformed into the frequency domain using the Short-Time Fourier Transform (STFT), which computes the frequency content over small windows of time.

$$X(m,k) = \sum_{n=0}^{N-1} x(n)w(n-mR)e^{-j2\pi kn/N} \tag{1}$$

where: - $X(m,k)$ represents the **spectrogram** at frame m and frequency bin k. - $x(n)$ is the input signal. - $w(n)$ is the windowing function. - R is the hop size.

The Mel spectrogram representation was then computed using a log-scaled Mel filter bank, converting the raw waveform into a 2D image-like structure suitable for CNN-based classification:

$$S(m,k) = \log\left(\sum_{j=0}^{M-1} H(k,j)|X(m,j)|^2\right) \tag{2}$$

where $H(k,j)$ represents the **Mel filter bank**.

Finally, the dataset was split into 80% training and 20% testing subsets to ensure balanced genre representation and robust model generalization. The complete data preprocessing workflow is shown below.

2.2 ResNet-152 Model Selection and Modification

We selected ResNet (Residual Network) as our deep convolutional neural network due to its ability to mitigate the vanishing gradient problem through residual connections. Unlike traditional CNNs, which struggle

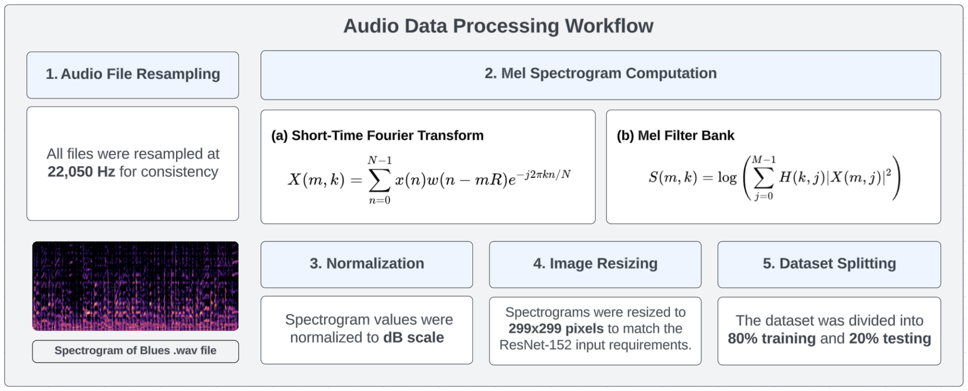
Audio Data Processing Workflow
1. Audio File Resampling
All files were resampled at 22,050 Hz for consistency
2. Mel Spectrogram Computation
(a) Short-Time Fourier Transform
$X(m,k) = \sum_{n=0}^{N-1} x(n)w(n-mR)e^{-j2\pi kn/N}$
(b) Mel Filter Bank
$S(m,k) = \log\left(\sum_{j=0}^{M-1} H(k,j)|X(m,j)|^2\right)$
Spectrogram of Blues .wav file
3. Normalization
Spectrogram values were normalized to dB scale
4. Image Resizing
Spectrograms were resized to 299x299 pixels to match the ResNet-152 input requirements.
5. Dataset Splitting
The dataset was divided into 80% training and 20% testing

Figure 2: Data Preprocessing Workflow

with learning deep hierarchical features as layers increase, ResNet introduces skip connections that allow gradients to flow through deeper layers and overcome the vanishing gradient problem by enabling easier gradient flow during backpropagation. ResNet's power lies in its ability to learn increasingly complex features across layers without encountering optimization difficulties that typically arise in deep architectures.

ResNet-152, a variant with 152 layers, was specifically chosen for this project due to its strong feature extraction capabilities. Its architecture allows it to capture both low-level and high-level audio features in music classification tasks. However, this model was also chosen because of its depth. Since deployment onto an edge device means a decrease in computational resources and memory, large enough models may overexert the device's limited processing power and fail upon deployment. Because of this, optimizations were necessary to ensure feasibility on an edge device.

To tailor the ResNet-152 model for audio classification, several modifications were introduced:

- **Global Average Pooling (GAP)**: The fully connected layers of the model were replaced with GAP to reduce the number of trainable parameters while retaining essential features. This was done to reduce the size of the model.
- **ReLU Activation Functions**: ReLU activations were applied to introduce non-linearity, enhancing feature learning and improving classification performance. They are also stored separately compared to the weights, which will help in later quantization stages.
- **Dropout Regularization (0.5 probability)**: Dropout was implemented to mitigate overfitting and improve generalization of the model.
- **Softmax Layer**: Final outputs were converted into probabilistic genre predictions, enabling multi-class classification.

2.3 Model Optimization

Given the computational limitations of edge devices, three quantization techniques were applied to improve memory size while maintaining accuracy:

- **Dynamic Quantization**: Selectively applied quantization only to the model weights while keeping activations in higher precision. During inference, floating-point activations are converted to INT8 before matrix multiplications/operations, and results are dequantized back to floating-point when needed.
- **FP16 Quantization**: Reduced the precision of model weights and activations from 32-bit floating point to 16-bit floating point (FP16), significantly lowering memory requirements while preserving performance.
- **INT8 Quantization**: Compressed model weights to 8-bit integer representations (INT8), leading to faster inference but a slight drop in accuracy.

2.4 Deployment on Raspberry Pi

Deploying deep learning models on edge devices like the Raspberry Pi presents challenges due to constrained computational power, memory limitations, and lower energy efficiency compared to traditional computing environments. Unlike desktop GPUs or cloud-based inference systems, edge devices must operate efficiently under resource constraints while maintaining accuracy.

We deployed the optimized model using TensorFlow, which provided hardware acceleration for running inference on the Raspberry Pi. Our final deployment workflow included converting the trained model into a TFLite format, optimizing for hardware-aware quantization, and running real-time classification over the testing dataset on the edge device. When deploying, we placed everything on the RPi's onboard RAM, which depending on the complexity and dependencies of the task. Including python and all other dependencies, the quantized and compressed model, and the script, the total amount of memory it took up was close to the maximum amount of 4GB of memory space available.

2.5 Evaluation Metrics

To assess the performance of our audio classification model, we employed standard classification metrics that evaluated both accuracy and storage efficiency.

- **Train Accuracy & Validation Accuracy**: Measures the percentage of correctly classified samples in training and validation sets. Higher validation accuracy indicates better generalization to unseen data.
- **Model Size (Memory Footprint)**: The total size of the trained model, which affects deployment feasibility on resource-limited devices like the Raspberry Pi. Smaller models require less storage and enable faster inference.

3 Experiments and Results

3.1 Performance Comparison

To evaluate the impact of our optimization techniques, we compare the train accuracy, validation accuracy, and model size across different versions of the ResNet-152 model. Table 1 summarizes the results.

Table 1: Summary of Model Performance

Model	**Train Accuracy**	**Validation Accuracy**	**Model Size**
Baseline (ResNet 152)	89%	74%	256 MB
Baseline (ResNet 152) with Modification	89%	78%	256 MB
Dynamic Quantization	– %	77.5%	59 MB
FP16 Quantization	– %	78%	116 MB
INT8 Quantization	– %	6%	60.1 MB
Pruned Model (Pending Optimization)	– %	74%	235 MB
FP16 ResNet-152 on Edge Device	– %	77.21%	116 MB
Dynamic ResNet-152 on Edge Device	– %	76.34%	59 MB

We began by running the full-precision ResNet-152 model on the GTZAN dataset as a baseline. While this achieves a high training accuracy of 89%, it suffers from overfitting, with validation accuracy dropping to 74%. After modifying the architecture to include a layer of each of Global Average Pooling (GAP), dropout regularization, and softmax activation, we observed a significant improvement in generalization, bringing validation accuracy to 78%. However, the model remained too large for efficient deployment.

3.2 Analysis of Quantized Models

The three quantization techniques—Dynamic, FP16, and INT8—yielded varying trade-offs between accuracy, model size, and computational efficiency. Figure 3 provides a visual comparison of the models' validation accuracy against their storage footprint.

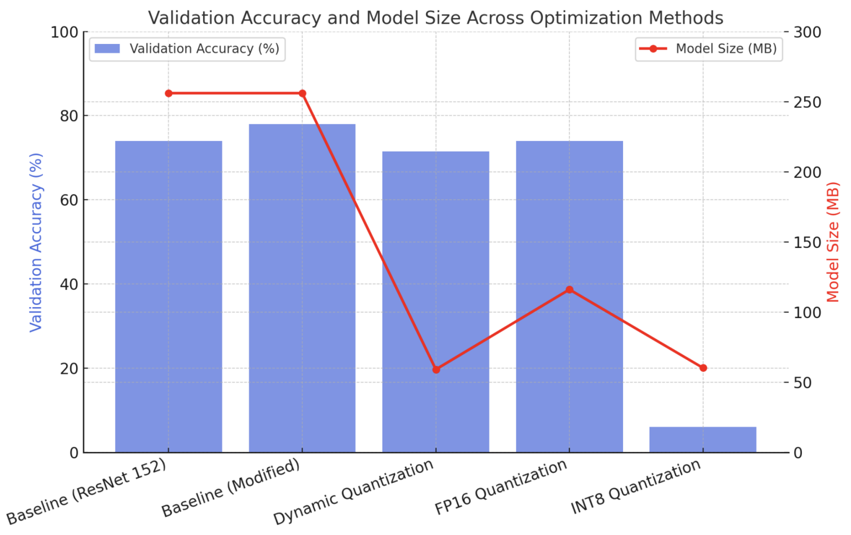

Figure 3: Comparison of quantized models: Accuracy vs. Model Size.

3.2.1 Dynamic Quantization

Dynamic quantization reduces model size by converting weights from FP32 to a lower-bit representation (typically INT8) at inference time, while keeping activations in full precision. This allows efficient storage and improved inference latency while preserving most of the model's predictive performance. Our results show that dynamic quantization reduced the model size from 256 MB to 59 MB—a 77% reduction—while maintaining a validation accuracy of 71.5%. This demonstrates that dynamic quantization can be highly effective when working with large deep networks on edge devices.

3.2.2 FP16 Quantization

FP16 quantization reduces both weights and activations to 16-bit floating point values, striking a balance between precision and efficiency. Unlike dynamic quantization, FP16 preserves floating-point arithmetic, making it more suitable for models that are sensitive to drastic changes in precision. In our experiments, FP16 quantization reduced the model size to 116 MB (a 55% reduction) while achieving 74% validation accuracy—higher than dynamic quantization. This suggests that FP16 maintains better numerical stability for deep models like ResNet-152.

3.2.3 INT8 Quantization and Accuracy Collapse

Unlike dynamic quantization, which applies INT8 conversion only to weights, full INT8 quantization applies it to both weights and activations, significantly reducing memory and computational requirements. However, INT8 models require explicit de-quantization before performing arithmetic operations, which can introduce rounding errors and loss of precision. Our results showed a sharp drop in validation accuracy to just 6%, despite the model size being reduced to 60.1 MB.

This dramatic loss in accuracy can be attributed to 1) the loss of fine-grained floating-point precision in both activations and weights, 2) the sensitivity of audio classification models to small variations in spectral features, which are crucial for distinguishing music genres, and finally 3) the lack of per-channel quantization, which can better preserve information in models where different layers exhibit varying sensitivities to quantization.

3.3 Comparison of Architecture Accuracies

While quantization significantly reduces model size and improves inference speed, it is important to compare these techniques against how it would perform on an edge device.

Below includes a table of how the memory was utilized within the Raspberry Pi, given around 2-4 times the amount of space for our model as room for any intermediary calculations it performed during inference.

Table 2: Comparison of Quantized ResNet-152 Accuracy on Different Devices

Model	**Validation Accuracy**	**Model Size**
FP16 Quantized ResNet-152	78%	116 MB
Dynamic Quantized ResNet-152	77.5%	59 MB
FP16 ResNet-152 on RPi	77.21%	116 MB
Dynamic ResNet-152 on RPi	76.34%	59 MB

Component	**Estimated Memory (MB)**
Python Interpreter	5–10 MB
TensorFlow (Installed + Runtime)	1,600–2,200 MB
NumPy	100 MB
Matplotlib	120 MB
SciPy	100 MB
Model (Loaded into RAM)	300–500 MB
TensorFlow Execution Overhead	500–1,500 MB
Total Estimated Memory	**2.7 GB – 4.5 GB**

Table 3: Estimated Memory Usage Breakdown

This is a significant reduction from computational resources we used which trying to train and run inferece tests on our GPUs. More specifically, 9.5 GB of system RAM as well as 5.6 GB of GPU RAM had to be used in order to train and validate our initial baseline ResNet152 model. The entire training dataset itself, and including other dependences that needed to be installed took up almost 31 GB of Disk space.

4 Conclusion

This study demonstrates that adaptive optimization techniques can significantly enhance the feasibility of deploying deep learning models for audio classification on edge devices. The results highlight three key takeaways for deploying deep learning models on edge devices: optimizing mixed-precision quantization, evaluating efficient tradeoffs for the models, and physically testing out the model and system on alternative edge devices.

Without modifications, the original ResNet-152 is impractical for edge devices due to size and computational constraints. While FP16 and dynamic quantization successfully reduce storage and inference costs with minimal accuracy loss, full INT8 quantization significantly degrades performance and generally ability to conduct any inference. Thus, there must be a balance of keeping high accuracy results while trying to minimize memory space. With a 77.21% accuracy rate and the smallest compressed model size at 59 MB, we conclude that a model with dynamic quantization would be best in compressing ResNet152 for edge deployment and inference.

4.1 Future Work and Enhancements

Building on these findings, several opportunities exist to further optimize model efficiency and deployment viability. If this project were to be extended beyond the current time constraints, we could explore more into the mixed quantization space. For example, instead of applying uniform quantization across the entire model, a mixed-precision approach that keeps critical layers in FP16 while quantizing others to INT8, or even more varied mixtures of FP32, FP16, and INT8 depending on what can be optimized on the hardware, could mitigate accuracy loss while maintaining storage efficiency. Future experiments could also analyze which layers are most sensitive to precision reductions.

Expanding past improvements solely focused on the model, future work could include power profiling the quantized models to assess their energy consumption during inference and explore optimizations that minimize computational overhead without compromising classification performance. Currently, the model has been tested on Raspberry Pi, but additional deployment experiments will be conducted on alternative edge hardware, including ESP32, NVIDIA Jetson Nano, and Google Coral TPU. These experiments will allow for a comparative analysis of inference speeds, power consumption, and performance trade-offs across different edge computing platforms.

References

[1] Tzanetakis, G., & Cook, P. (2002). Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, *10*(5), 293-302. `https://doi.org/10.1109/TSA.2002.800560`

[2] NVIDIA ADLR. (2019, August 13). MegatronLM: Training Billion+ Parameter Language Models using GPU Model Parallelism. `https://nv-adlr.github.io/MegatronLM`

[3] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016, February 24). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv.org*. `https://arxiv.org/abs/1602.07360`

[4] Howard, A., Sandler, M., Chu, G., Chen, L., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019, May 6). Searching for MobileNetV3. *arXiv.org*. `https://arxiv.org/abs/1905.02244`

[5] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770-778). `https://doi.org/10.1109/CVPR.2016.90`

[6] Luo, X., Liu, D., Kong, H., Huai, S., Chen, H., Xiong, G., & Liu, W. (2024). Efficient Deep Learning Infrastructures for embedded Computing Systems: A Comprehensive Survey and Future Envision. ACM Transactions on Embedded Computing Systems, 24(1), 1–100. `https://doi.org/10.1145/3701728`

[7] Raspberry Pi 4 specs and benchmarks — The MagPi magazine. (n.d.). The MagPi Magazine. https://magpi.raspberrypi.com/articles/raspberry-pi-4-specs-benchmark