CIS 4190/5190: Applied Machine Learning Fall 2024

Audio-Based Material Classification

Team Members: Alexis Powell, Zitong Ren, Jiaming Li *Project: Audio2Material*

1 Summary

Audio-based material classification is an emerging field within machine learning that focuses on identifying materials based on their sound signatures, with applications ranging from industrial quality control to environmental monitoring. Achieving accurate classification relies on effective sound feature extraction and selecting appropriate machine learning algorithm(s). Traditional approaches primarily use feature extraction techniques such as Mel-Frequency Cepstral Coefficients (MFCCs) and Mel spectrograms, which transform raw audio signals into structured representations that capture essential acoustic characteristics. Convolutional Neural Networks (CNNs) are frequently utilized for their ability to learn spatial patterns within these feature representations, demonstrating robust performance in distinguishing between different materials based on their sound signatures. Transformer-based architectures can be also used – these leverage self-attention mechanisms to model temporal dependencies more effectively across entire audio sequences. While transformers are effective, CNNs remain favored for shorter audio clips (and our data is 1-2 seconds long) due to its computational proficiency in capturing local features. The objective of this project is to explore and develop a robust machine learning model for audio-based material classification, focusing on objects found around Penn. Leveraging audio data collected from various objects, such as tables, sofas, handrails, and more, our goal is to classify each object by the unique sound it makes when knocked on.

A logistic regression baseline was implemented, with initial experiments highlighting its high performance due to suitability for small datasets (200 samples). Building on this, we developed a CNN to utilize Mel spectrogram features (1000+ samples), with an emphasis on exploring key hyperparameters such as the number of convolutional layers, kernel sizes, strides, fully connected layers, batch sizes, and learning rates. To ensure an efficient approach to model optimization, a mass iteration program was designed for exploratory experimentation, enabling the evaluation of 398 out of 576 possible hyperparameter combinations.

The optimal CNN configuration achieved high performance with the following characteristics:

- **2 Convolutional Layers**: Increasing the number of layers beyond this did not yield performance improvements. This architecture balances complexity and feature extraction efficiency.
- **Kernel Size of 5**: Larger kernel values captured more spatial details for our input.
- **Stride of 1**: A smaller stride preserved spatial information more effectively than larger strides, leading to better feature retention and accurate classification.
- **Max Pooling Layer**: Pooling reduced the dimensions of feature maps by half while preserving important information, preventing overfitting.
- **Fully Connected Layer**: A single dense layer with 7 output nodes simplified the architecture and maintained classification accuracy.
- **Batch Size of 16**: Selected to achieve reliable convergence with efficient memory utilization during K-Fold Cross-Validation.
- **Learning Rate of 0.0005**: This value provided stable convergence, avoiding overshooting or excessively slow learning, particularly during cross-validation training.
- **K-Fold Cross-Validation (k=5)**: Used to validate model generalization performance across splits, with accuracy consistently above 95% across all folds.

This project highlights the effectiveness of CNNs for audio-based material classification, showcasing how

careful hyperparameter tuning and model selection can lead to highly accurate results. Our findings provide valuable insights for further research and application in real-world scenarios where audio data is used for material identification.

2 Core Components

2.1 Data Collection and Dataset

To ensure generalizability in our dataset, all three teammates participated as data collectors. Over 1,000 1-second `.wav` files were recorded, representing sound samples from the seven specified objects. Audio recordings were conducted during varying times—weekend evenings, weekday afternoons, and weekday evenings—to account for potential variations in background noise.

Once collected, the raw audio files underwent preprocessing to enhance the quality of the data. Each audio clip was trimmed to remove any silence or irrelevant background noise, ensuring that only the sound signature of the object was preserved. To further process the data, we implemented a Python script to extract Mel spectrograms. The script organized outputs into a mirrored directory structure and processed each file using the `librosa` library. It applied a Short-Time Fourier Transform, mapped the resulting frequencies to the Mel scale, and converted the power spectrogram to decibels. The spectrograms were visualized with `matplotlib`, displaying time on the x-axis, Mel frequency on the y-axis, and intensity as a color gradient, before being saved as `.png` files. Examples of Mel spectrograms for each class 1, 2, 3, 4, 5, 6, and 7 are shown in the Appendix.

To improve generalization, data was transformed using resizing and normalization. The images were resized to 128 x 128 pixels to ensure consistent input dimensions, and normalized to a mean of 0.5 and a standard deviation of 0.5. For internal evaluation, our data was split into the training set (for model learning and optimization), the validation set (to tune hyperparameters and prevent overfitting), and the test set (for unbiased evaluation of the model's performance). Our final dataset consisted of over 1,000 high-quality audio samples, uniformly distributed across the seven object classes. Each sample captures the distinct sound signature of the object, processed into Mel spectrograms for feature extraction.

2.2 Model Design

We reviewed relevant literature, including a study that identified a model with a single convolutional layer followed by two fully connected layers as optimal, recommending a low number of filters and narrow layer sizes to minimize overfitting [7]. In contrast, another approach employed a parallel structure with three hidden layers, each containing 200 nodes, showcasing a different method for acoustic object recognition [3]. Our model focused on a CNN architecture aimed at classifying images of Mel spectrograms into seven distinct classes: **water, table, sofa, railing, glass, blackboard,** and **ben**. The design process included several considerations to improve model performance, and the final model was built to maximize accuracy using K-fold cross-validation.

At the outset, this prior research guided us to select a CNN as our model architecture, given its effectiveness in processing and extracting features from audio spectrogram data. We used the following baseline parameters and data setup:

- **Learning Rate**: Initially set to 0.001, aiming for faster convergence. However, this proved too aggressive, leading to instability in the optimization process.
- **Cross-Validation**: The initial design did not include K-fold cross-validation, which limited our ability to effectively estimate model performance and prevent overfitting.
- **Batch Normalization Layers**: Batch normalization layers were introduced to stabilize the training process. However, in the early stages, the configuration led to significant fluctuations in loss.
- **Dataset Size**: We initially collected only 200 audio samples; while sufficient for initial experimen-

tation, it was later determined that this dataset was inadequate for training a CNN, which typically requires more extensive data for effective feature extraction.

To address these initial challenges, several iterative enhancements were made, informed by feedback from TAs and experimentation. Since the initial learning rate of 0.001 was unstable, smaller values were required. Ultimately, a learning rate of 0.0005 was chosen for its convergence speed and stability – this rate allows the CNN to make more gradual updates to weights, reducing the risk of overshooting the optimal solution. To mitigate overfitting, K-fold cross-validation was integrated into the training process, offering robust performance evaluation across data splits. Our dataset was also expanded from 200 to over 1,000 Mel spectrograms, enabling the model to learn more generalized features. Architectural improvements included adding additional convolutional and pooling layers to enhance feature extraction. We additionally experimented with incorporating dropout layers in the model, but ultimately chose not to include them as they did not significantly improve performance.

Our final model is a CNN-based architecture tailored for classifying audio data represented as Mel spectrograms. The model comprises 2 convolutional layers with kernel sizes of 5 x 5, each followed by ReLU activation and max pooling, reducing spatial dimensions for feature extraction. The extracted features are fed into a fully connected layer for classification into seven classes corresponding to the label mappings.

The training workflow incorporates K-fold cross-validation (k=5), which splits the dataset into distinct training and validation subsets for each fold, ensuring the model generalizes well across unseen data. The model is trained using the Adam optimizer with a learning rate of 0.0005 and a Cross-Entropy Loss function, suitable for multi-class classification tasks. Epoch-wise logging of losses and accuracies ensured controlled training. Key hyperparameters such as batch size and model depth were fine-tuned for optimal performance, as we discuss in later sections.

2.3 Evaluation and Model Performance

Cross-validation results demonstrated that the CNN effectively generalized across the dataset, with minimal overfitting:

Training Accuracy: Consistently above 96% across all folds.

Validation Accuracy: Ranged between 95.62% and 98.34% across folds.

Validation Loss: Demonstrated stable minimization across epochs.

This configuration balances complexity and performance, making it suitable for the classification of spectrogram images into the classes. We evaluated model iterations internally by analyzing additional performance metrics such as F1 score, precision, and recall, all of which consistently achieved values of 1.0 (proof from an earlier iteration of the model is attached in the Appendix 8). Such results indicate perfect performance across these metrics. This robust internal evaluation, along with the accuracy and loss, gave us confidence in the model's effectiveness. Thus, we submitted our top-performing model to the leaderboard, which had an overall accuracy of 56%. Per class accuracies included: 28.57% for water, 64.29% for table, 72.73% for sofa, 65.38% for railing, 30.43% for glass, 0% for blackboard, and 96.88% for ben.

3 Exploratory Questions

Methods: These experiments were conducted in a separate file purely for exploration. While the insights gained from this process informed our understanding, not all of the knowledge was incorporated into the final model due to the variability observed in the results. In this investigation, we explored a range of hyperparameters to optimize the performance of a CNN for audio classification. We developed an iteration framework featuring a dynamic CNN function, which enabled the efficient construction and testing of various CNN architectures. To expedite the experimentation process, the dataset was limited (while still maintaining sufficient diversity across classes) to reduce computation time. The hyperparameters explored included the number of convolutional layers (ranging from 2 to 5), kernel size (3 to 5), stride (1 to 2), number of fully connected layers (1 to 3), batch size (16, 32, 64), learning rates (0.001, 0.0005, 0.0001, 0.00005), and number of epochs (set to 20). The model's architecture was dynamically adjusted based on these parameters, and experiments were conducted using a 50% train-test split. Training and evaluation were carried out using the Adam optimizer and a Cross-Entropy Loss function to stay consistent with our final model.

3.1 How does the number of convolutional layers and other architectural choices (e.g., kernel size, stride, fully connected layers) affect the accuracy and performance of the model?

Our motivation is to understand how architectural variations influence the CNN's ability to effectively extract and process features from Mel spectrograms. Based on prior research and course material, our expectation was that increasing the number of convolutional layers will improve feature extraction, especially for complex audio-based data like Mel spectrograms. Each convolutional layer captures increasingly complex patterns. However, beyond a certain point, adding too many layers could lead to overfitting, particularly with smaller datasets [10]. Larger kernel sizes may help capture broader patterns, but could lose local detail, while smaller kernels might better preserve spatial resolution at the cost of computational efficiency. Similarly, stride size could affect the model's ability to capture fine details; smaller strides preserve more of the input resolution, while larger strides reduce the input's spatial dimensions faster. The final stage of a CNN, the fully connected layers, play a crucial role in combining the extracted features for accurate classification, but too many fully connected layers could lead to overfitting. Our expectation was that an optimal balance of these factors will lead to the best performance.

Findings: The model with 91.33% accuracy had the following hyperparameters: ConvLayers=2, KernelSize=5, Stride=1, FCLayers=2, BatchSize=16, and LR=0.001. The high accuracy suggests that a combination of 2 convolutional layers, a kernel size of 5, and 2 fully connected layers was an optimal configuration for the dataset. This model reinforced the importance of carefully balancing hyperparameters to optimize CNN performance. It confirmed that using a smaller number of convolutional layers (2 in this case) still leads to strong results. Additionally, the finding that a kernel size of 5 outperformed smaller kernels (e.g., size 3) supports the notion that larger kernels capture more spatial information, which is crucial for image classification tasks. Lastly, using a stride of 1 helped retain spatial information, contributing to better feature extraction and improved accuracy. Our model integrated these results by using 2 convolutional layers, a kernel size of 5, and a stride of 1.

3.2 What are the optimal hyperparameters (e.g., learning rate, batch size) for training the CNN model efficiently, while achieving high accuracy?

Our motivation stems from the significant influence of these hyperparameters on the model's convergence speed, stability, and accuracy. For our project, where the dataset comprises processed spectrogram images, choosing the right hyperparameters is critical to achieving a model that generalizes well without overfitting or underfitting. From prior research and course material, we expect that the learning rate will significantly influence both the stability and speed of convergence. A learning rate that is too high can cause the model to overshoot the optimal solution, while a rate that is too low may result in slowed convergence [9]. For batch size, smaller values may lead to more frequent updates, which can help with faster learning. However, this may be at the cost of increased variance in gradient estimates, potentially destabilizing the training process. Based on similar studies, we anticipated that tuning these hyperparameters to the specific characteristics of our dataset would be crucial for achieving both efficient training and high model accuracy.

Findings: The model with 86.00% accuracy had the following hyperparameters: ConvLayers=3, KernelSize=5, Stride=1, FCLayers=1, BatchSize=32, and LR=0.001. This model's slightly lower accuracy compared to the one from the first exploratory question suggests that increasing the number of convolutional layers from 2 to 3 does not necessarily lead to better performance, giving us confidence to use 2 in our final model. While the kernel size of 5 and stride of 1 remained optimal, reducing the number of fully connected layers to 1, along with a larger batch size, might have contributed to overfitting or under fitting, leading to a slight drop in performance. The impact of fully connected layers and batch size was evident, suggesting that the architecture should be tailored to the specific dataset and task. These results also highlighted the need to be cautious with larger batch sizes, as they may negatively impact the convergence and generalization of the model. Our model integrated this finding by using a batch size of 16 instead of 32.

3.3 Conclusions

After defining a total of 576 parameter combinations, 398 combinations were executed during the mass iteration process. The experiment was halted early because configurations with a high number of convolutional layers consistently achieved accuracy below 80%, making them less promising to explore further. From the results, we selected the top 5 combinations with the highest accuracy and trained them on the full dataset. Upon analyzing the data across all combinations, we observed several trends: models with 2 convolutional layers generally outperformed those with 3 or 4 layers, a kernel size of 5 produced higher accuracies compared to a kernel size of 3 (larger kernels capture more spatial information), a stride of 1 consistently outperformed a stride of 2 (it preserves more spatial information in the feature maps), and a batch size of 16 resulted in the highest accuracy, (smaller batch sizes help the model converge more reliably).

3.4 Limitations & Potential Future Exploration

While grid search is a systematic approach to hyperparameter optimization, it proved to be time-consuming in this experiment. Each iteration took approximately 5 minutes, and with 576 combinations, the total runtime would have been 48 hours, creating a significant computational burden. This inefficiency was particularly evident when evaluating models with 3 fully connected layers, which consistently performed poorly, but still required substantial time to evaluate. The uniform exploration across all combinations failed to account for early signs of underperformance in certain hyperparameter regions, resulting in wasted computational resources. With more time and resources, we would consider using the Tree-structured Parzen Estimator (TPE) algorithm [8]. TPE is a probabilistic model-based technique that models "good" and "bad" configurations separately, allowing it to prioritize promising parameters and avoid wasting time on unlikely configurations. This approach scales better with high-dimensional problems and can find optimal hyperparameters with fewer evaluations, making it more efficient than traditional grid search or random search methods, especially for complex search spaces.

4 Team Contributions

Alexis Powell: Collected audio data, wrote the project report, preprocessed data, converted audio recordings to Mel spectrograms, designed and developed the CNN, and analyzed exploratory experiment results.

Zitong Ren: Collected audio data, wrote and ran the mass iteration program, worked on Mel spectrogram conversion, contributed to the report, and developed the CNN.

Jiaming Li: Collected audio data, read papers, and uploaded our final model to GitHub.

5 Components

GitHub, Model in Colab

Dataset, Mel Spectrograms

Exploratory Experiments, Exploratory Results

6 Acknowledgments

Daniel Alexander, Edward Hu, Tongtong Liu

References

[1] Choi, Keunwoo, et al. A Comparison on Audio Signal Preprocessing Methods for Deep Neural Networks on Music Tagging. *ArXiv*, September 2017, doi:10.48550/arXiv.1709.01922.

[2] Clarke, Samuel, et al. RealImpact: A Dataset of Impact Sound Fields for Real Objects. *ArXiv*, 2023, eprint: 2306.09944, https://arxiv.org/abs/2306.09944.

[3] Luo, Shang, et al. Knock-Knock: Acoustic Object Recognition by Using Stacked Denoising Autoencoders. *ArXiv*, abs/1708.04432 (2017): n. pag.

[4] Neumann, M., et al. Material Classification through Knocking and Grasping by Learning of Structure-Borne Sound under Changing Acoustic Conditions. *IEEE 14th International Conference on Automation Science and Engineering (CASE)*, Munich, Germany, 2018, pp. 1269-1275, doi: 10.1109/COASE.2018.8560527.

[5] Owens, Andrew, et al. Visually Indicated Sounds. *ArXiv*, 2016, eprint: 1512.08512, https://arxiv.org/abs/1512.08512.

[6] Papayiannis, Constantinos, et al. Detecting Sound-Absorbing Materials in a Room from a Single Impulse Response using a CRNN. *ArXiv*, 2019, eprint: 1901.05852, https://arxiv.org/abs/1901.05852.

[7] Sterling, Auston, et al. ISNN: Impact Sound Neural Network for Audio-Visual Object Classification. *European Conference on Computer Vision*, 2018, pp. 555–572

[8] Watanabe, Shuhei. Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance. 2023, eprint: 2304.11127, archivePrefix: arXiv, primaryClass: cs.LG. https://arxiv.org/abs/2304.11127.

[9] Wojciuk, M., et al. Improving Classification Accuracy of Fine-Tuned CNN Models: Impact of Hyperparameter Optimization. *Heliyon*, vol. 10, no. 5, e26586, 2024, doi: 10.1016/j.heliyon.2024.e26586.

[10] Yamashita, R., et al. Convolutional Neural Networks: An Overview and Application in Radiology. *Insights Imaging*, vol. 9, pp. 611-629, 2018, doi: 10.1007/s13244-018-0639-9.

A Appendix

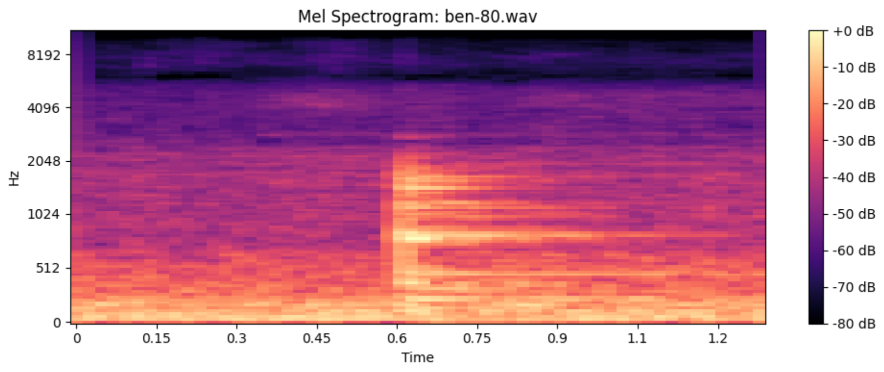


Figure 1: Mel Spectrogram - ben-80.wav

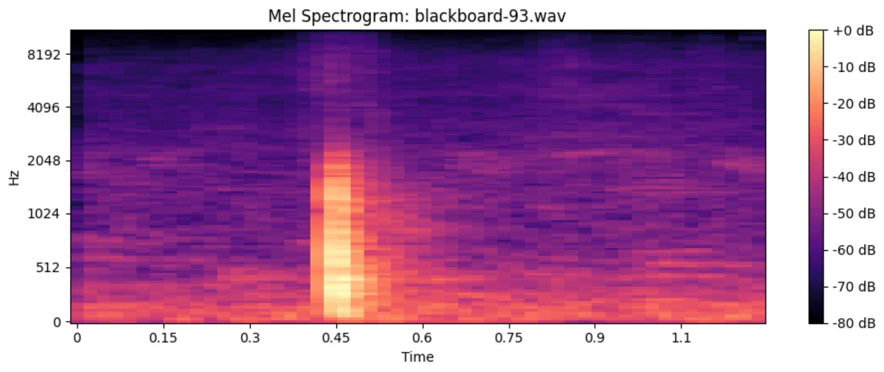


Figure 2: Mel Spectrogram - blackboard-93.wav

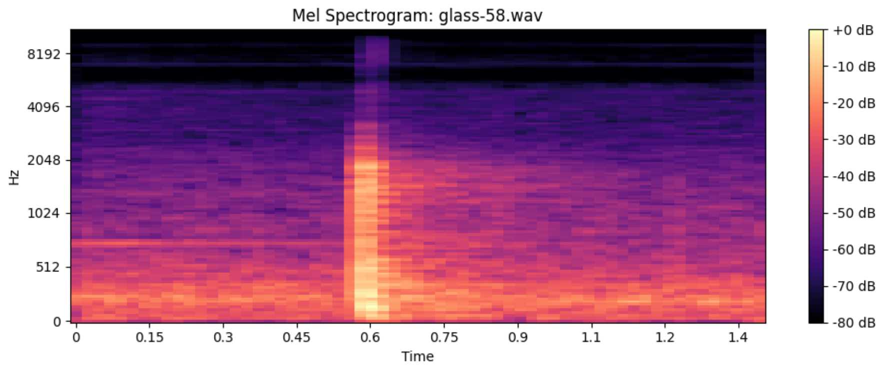


Figure 3: Mel Spectrogram - glass-58.wav

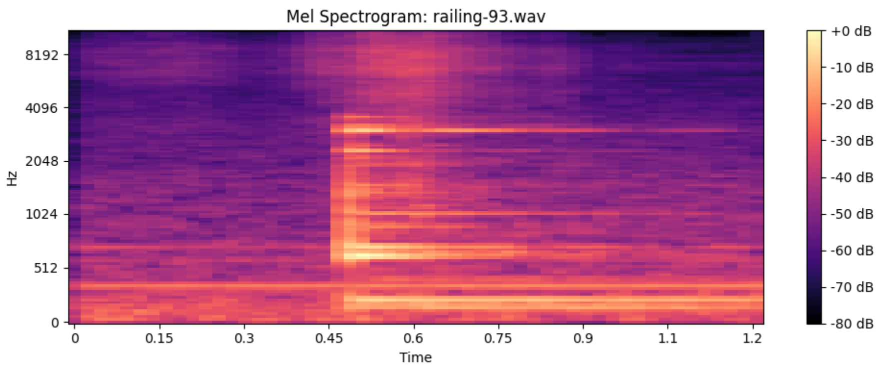


Figure 4: Mel Spectrogram - railing-93.wav

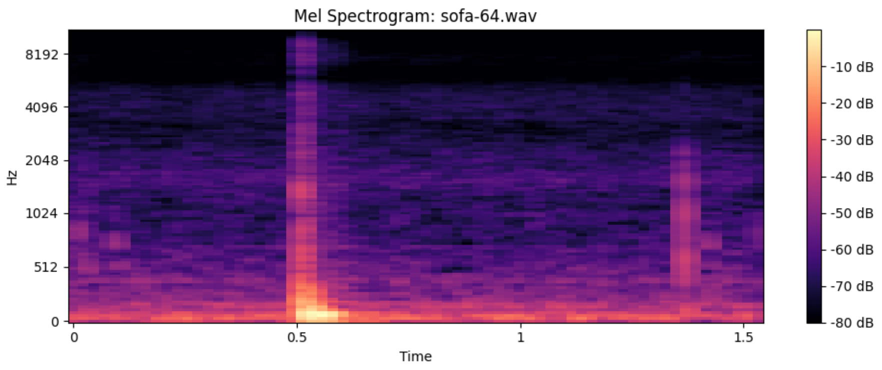


Figure 5: Mel Spectrogram - sofa-64.wav

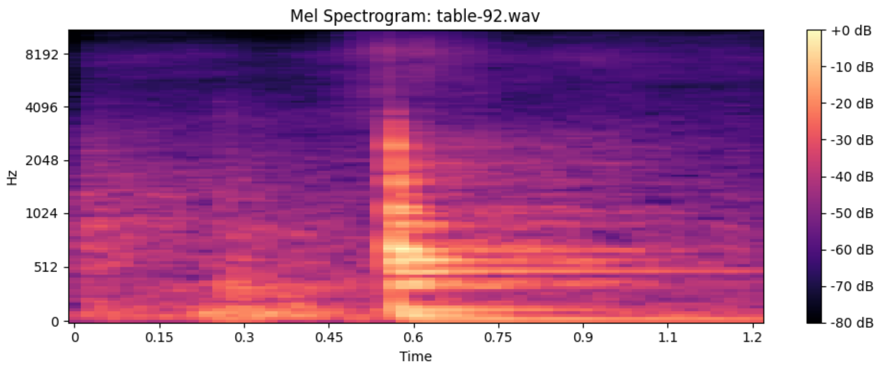


Figure 6: Mel Spectrogram - table-92.wav

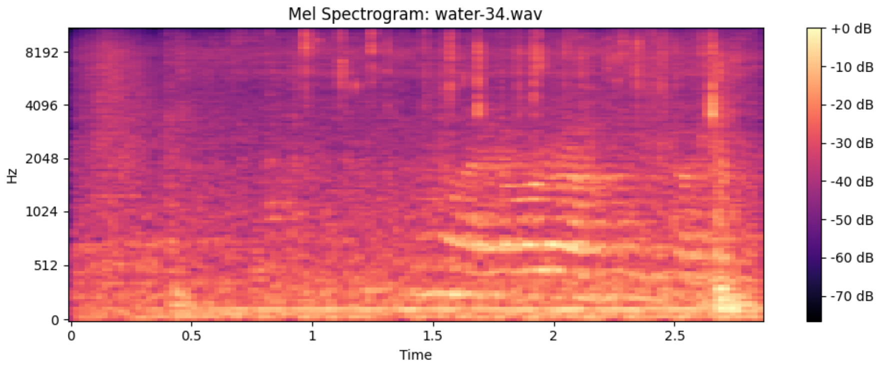
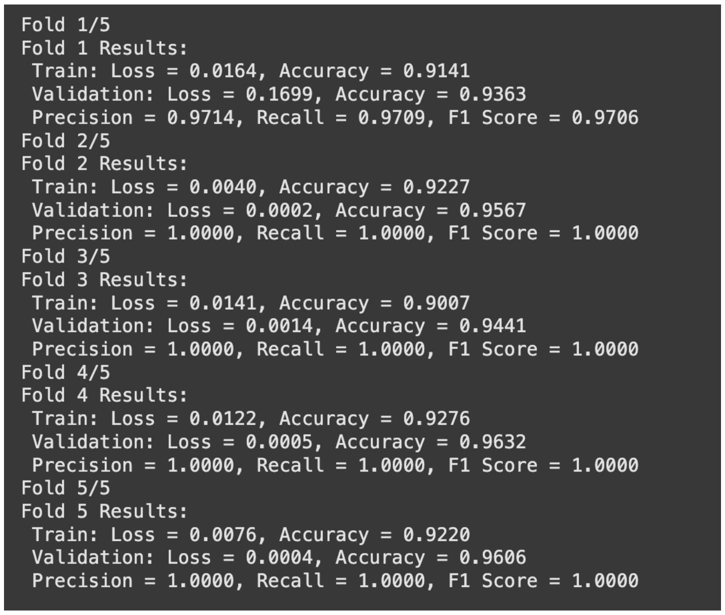


Figure 7: Mel Spectrogram - water-34.wav



```
Fold 1/5
Fold 1 Results:
 Train: Loss = 0.0164, Accuracy = 0.9141
 Validation: Loss = 0.1699, Accuracy = 0.9363
 Precision = 0.9714, Recall = 0.9709, F1 Score = 0.9706
Fold 2/5
Fold 2 Results:
 Train: Loss = 0.0040, Accuracy = 0.9227
 Validation: Loss = 0.0002, Accuracy = 0.9567
 Precision = 1.0000, Recall = 1.0000, F1 Score = 1.0000
Fold 3/5
Fold 3 Results:
 Train: Loss = 0.0141, Accuracy = 0.9007
 Validation: Loss = 0.0014, Accuracy = 0.9441
 Precision = 1.0000, Recall = 1.0000, F1 Score = 1.0000
Fold 4/5
Fold 4 Results:
 Train: Loss = 0.0122, Accuracy = 0.9276
 Validation: Loss = 0.0005, Accuracy = 0.9632
 Precision = 1.0000, Recall = 1.0000, F1 Score = 1.0000
Fold 5/5
Fold 5 Results:
 Train: Loss = 0.0076, Accuracy = 0.9220
 Validation: Loss = 0.0004, Accuracy = 0.9606
 Precision = 1.0000, Recall = 1.0000, F1 Score = 1.0000
```

Figure 8: F1 Score, Precision, Recall