Urban Sound Classification: Optimizing Urban Sound Recognition Through Data Selection, Augmentation, and Transformation Techniques Using Advanced DL Architectures, on The Urban Sound Datasets

Nikhil Thota, Venu Anupati, Mounika Vempalli, and Hema V.S. Dumpala

Department of Applied Data Science, San Jose State University

DATA 270: Data Analytics Processes

Dr. Eduardo Chan

December 8, 2023

Abstract

This research delves into the field of urban audio classification, with pivotal applications in smart homes, traffic management, and assistive technologies for individuals with disabilities. The primary challenge tackled is the scarcity of labeled audio data, which is crucial for the effectiveness of models. The study's goal is to assess the performance of Vision Transformers (ViT) compared to prevalent DL architectures like CNN, CNN-LSTM, and CNN-BiGRU, specifically on the UrbanSound8k and ESC-50 datasets. Prior research in this sphere has achieved notable progress, particularly in audio classifications such as machinery sounds and health-related audio, with the highest recorded accuracy in urban audio classification reaching approximately 96% using a ViT encoder with a patch diffusion approach. Building on these advancements, the research applies Vision Transformers to urban audio data and contrasts its efficiency with that of leading models. It further examines the influence of Data Engineering Limprinciples like Data Augmentation and Feature Reduction on datasets with limited labeled data. These techniques were found to considerably boost performance metrics. The study also integrates Transfer Learning, employing pre-trained models like Google ViT, Facebook ViT, and EfficientNet, thereby enhancing efficiency and diminishing training duration. The findings reveal that after fine-tuning hyperparameters, ViT attains remarkable ~93% accuracy on the ESC-50 dataset, and ~96% accuracy on UrbanSound8k, surpassing other models in comparative analysis. This investigation not only underscores the prowess of Transformers in urban audio classification but also emphasizes the significance of Data Augmentation, Feature Reduction, and Transfer Learning in achieving high accuracy with a constrained supply of labeled data.

1 Introduction

1.1 Project Background and Executive Summary

Project Background, Needs, and Importance, Target Problem, Motivation, and Goals

Sound is a pervasive element in our environment, continuously providing us with information about the occurrences and activities in our surroundings. From recognizing a familiar voice to understanding the urgency of a siren, the capability to interpret sounds plays a pivotal role in how humans navigate their environment. Sounds also provide context, enabling humans to understand and interact adequately with their surroundings. Automated classification and understanding of environmental sounds, therefore, emerges as a critical aspect in the development of intelligent systems, enhancing their perceptual and interactive capabilities. The field of sound recognition, which is commonly associated with environment sound classification (ESC), has garnered significant attention due to its potential applications in various domains, including healthcare and surveillance. Developing models that can accurately classify various sounds provides a reliable channel of information, enhancing situational awareness for intelligent systems and offering enriched interactive experiences for Users.

Think about the bustling sounds in our cities - understanding these noises is crucial in making our cities more livable, from planning quieter and more peaceful public spaces to monitoring and preserving wildlife in urban parks. In our homes, distinguishing between regular kitchen sounds and a potential break-in through glass shattering could enhance our home security systems. In healthcare, quickly recognizing sounds of distress can be life-saving. For industries, identifying strange noises from machines can prevent unwanted downtimes. In places like shopping malls, interpreting customer sounds efficiently can improve service. And crucially, for people without sight, recognizing common urban sounds can provide vital clues about their

surroundings, aiding in safe navigation through cityscapes. Hence, developing models to accurately classify these environmental sounds holds paramount significance across varied, real-world scenarios, contributing to safer, efficient, and more responsive urban ecosystems. Recognizing and understanding diverse sounds in urban environments presents numerous advantages across various sectors, yet the journey toward developing models capable of accurately classifying these sounds grapples with a significant barrier: the scarcity of labeled audio data.

Collecting vast amounts of audio data from various environments is not only a meticulous and labor-intensive task but also requires rigorous labeling to train models effectively. Each sound snippet needs to be carefully identified and tagged with the correct label, a process that demands human labor, time, and expertise in both the sounds being labeled and the context they are used in. This Herculean effort becomes even more complex with the sheer diversity of sounds in an urban environment, where myriad noises overlap and interact in a continuous, dynamic symphony. The substantial resources, time, and logistical efforts required to amass such a dataset render it a daunting and often infeasible task, thereby posing a substantial obstacle to advancing the research and development of environmental sound classification models. Consequently, striking a balance between developing accurate models and the practicality of data collection and labeling becomes a pivotal challenge in this research domain.

This research is going to focus on the comparison study of several proposed deep learning models aimed at proficiently classifying urban sounds, especially when confronted with the challenges of limited labeled audio data. With the implementation of an array of innovative data selection algorithms, augmentation, and transformation techniques, the focus intensifies on

harnessing the power of the scarce labeled data to its utmost potential, ensuring accurate classifications in diverse acoustic environments found in urban settings. The goal of this investigation's conclusion is to improve urban sound classification's accuracy, F1 Score, and other performance metrics while also illuminating ways to better model and understand the nuances of soundscapes despite the difficulties presented by sparse, labeled data in a variety of real-world acoustic environments. Consequently, the implications and learnings from our study have the potential to significantly impact various applications, from enhancing urban planning and surveillance to refining assistive technologies, thereby contributing to the building blocks that shape smarter and more aware urban environments.

Project Approaches and Motivations

The project uses the widely used CRISP-DM approach, which consists of six phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment. After careful consideration, a CRISP-DM Agile methodology is preferable given the types of models and iterative nature of this machine learning project. Agile methodologies are designed to be flexible, iterative, and sensitive to changing requirements and new information, characteristics that are perfectly suited for a machine learning project. To transition to a CRISP-DM Agile methodology, the six phases of the CRISP-DM technique will be broken up into shorter, easier to manage iterations or sprints. The team is able to advance the project while preserving flexibility and teamwork as its top priority by concentrating on one or two CRISP-DM phases during each sprint.

Arnolt et al. (2022) quoted the necessity to investigate dimensionality reduction and feature extraction approaches because audio signals have high dimensionality, which means that

more than 1,000 floating point values are needed to represent a brief audio signal. Laith et al. (2021) quoted the development of data augmentation in sound classification tasks has proved its consistency in enhancing the performance of training models for limited data, in contrast to the widely held belief that training big datasets is necessary to obtain the best results for deep architecture models as mentioned by Zhao et al. (2022).

When compared with the traditional Machine Learning models, a CNN model delivered better accuracy rates, and performance metrics for environmental sound classification which is explained by Saeed et al. (2021). Li et al.(2022) explained how Pure attention-based neural architectures Vision transformers are rising to the top of the audio tagging (AT) leaderboards after achieving resounding success in vision and language tasks, potentially superseding more established convolutional neural networks (CNNs), feed-forward networks, or recurrent networks.

Moreover, the popular and groundbreaking transformers architecture was introduced in the work of Vaswani et al.(2017) where the whole convolution block is replaced with a multi head attention mechanism, which makes sure that the transformers architecture can perform parallel operations at scale. Another popular paper, which extended transformers to Image Classification work of Dosovitskiy et al. (2020) explains how any image can be considered as a sequence of patches, and even validates their Vision transformers architecture on the Image classification datasets achieving SOTA scores with the transformers architecture.

Project Contributions and Applications

This project significantly advances the capacity to work with minimal labeled audio data, a common challenge in audio research. Through employing advanced data selection strategies

and data augmentation techniques, it enriches the training process to improve model robustness and performance, even with limited labeled data. Additionally, the exploration of various dimensionality reduction techniques contributes to efficiently managing high-dimensional data, which is often a hurdle when dealing with sparse labels. The meticulous optimization, extensive hyperparameter tuning, and thorough performance evaluation methodologies adopted in this project further ensure that the models are fine-tuned to perform well despite the constraint of minimal labeled data. Hence, the project serves as a valuable resource for researchers facing challenges with limited labeled audio datasets, demonstrating effective strategies to maximize the utility of available labeled data in audio signal processing tasks.

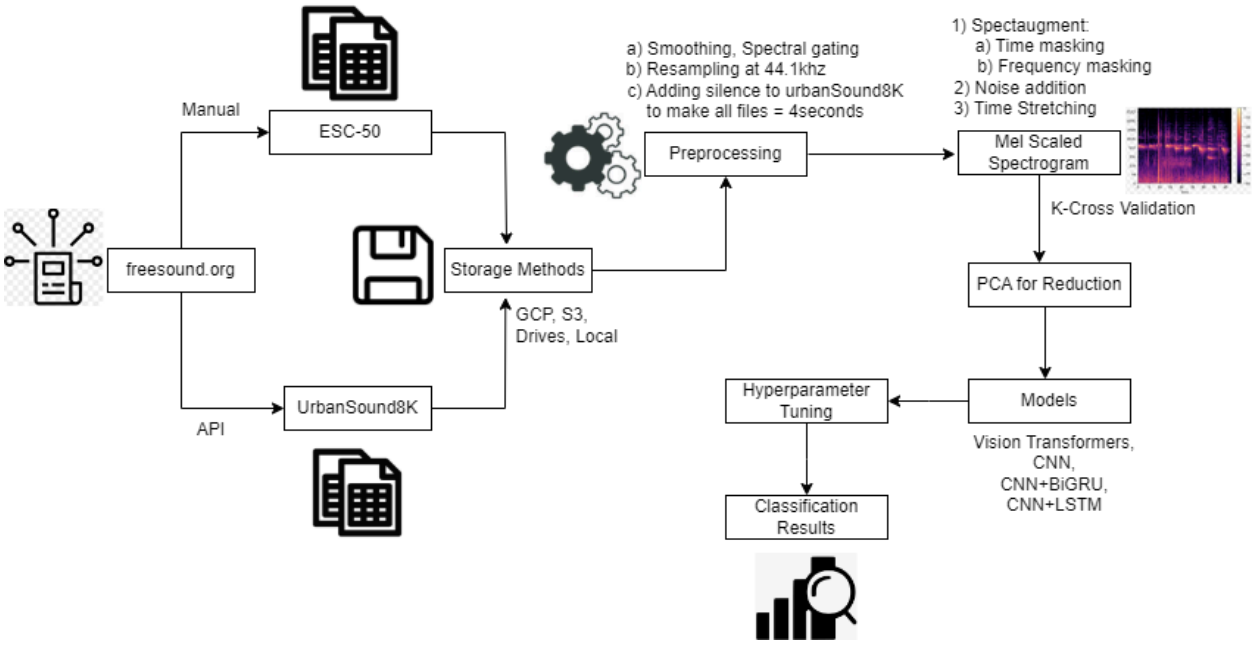
The proposed models in this project are meticulously designed to tackle challenges in audio signal processing, especially under the constraints of minimal labeled data. Their applications span across various crucial domains. In urban sound classification, they aid smart city initiatives like surveillance, noise pollution monitoring, and emergency response systems, where abundant labeled data is often hard to come by. Similarly, in environmental sound 9 recognition, they facilitate wildlife monitoring, biodiversity assessment, and eco-acoustic research, where the scarcity of labeled data is a common hurdle. The models' capability to work efficiently with sparse labels is a significant boon for audio surveillance in security systems, detecting anomalous sounds to enhance safety in public and private spaces. In healthcare, these models open avenues for acoustic monitoring of patients, enabling early detection and monitoring of medical conditions through the analysis of breathing or heart sounds, despite the common challenges of data privacy and labeled data scarcity. Furthermore, they hold promise in advancing Human-Computer Interaction (HCI) by enabling machines to understand and respond to natural acoustic environments, fostering more intuitive user interfaces. Lastly, the integration

of these models into assistive technologies can significantly aid individuals with hearing impairments, enhancing their situational awareness by interpreting environmental sounds, all while effectively leveraging the minimal labeled data available. Overall, the models proposed in this project demonstrate a substantial stride towards harnessing sparse labeled data in audio signal processing, extending a wide array of real-world applications from urban infrastructure to ecological conservation and beyond.

Below Figure 1 is the workflow diagram for this entire research project to get an overall understanding.

Figure 1

Workflow Diagram of Entire Research Project



1.2 Project Requirements

In the context of this research project, the development environment plays a crucial role in ensuring efficient model training and experimentation. Google Colab Pro emerges as a fitting choice due to its tailored features aligning with the project's specific requirements. The flexibility

to customize RAM and access additional GPUs proves invaluable for handling the complex computations involved in audio classification tasks. The on-demand scalability methods provided by Google Colab Pro are particularly advantageous when working with deep learning models on audio datasets, offering the computational resources necessary for enhancing model accuracy and performance. Overall, Google Colab Pro's customizable features cater to the specific demands of this research endeavor, providing an optimal development environment for robust experimentation and model refinement.

Functional Requirements

The proposed models, including ViT, CNN, CNN-LSTM, and CNN-BiGRU, must be precisely implemented for processing spectrogram images. GCP Buckets will store audio data, and data processing requirements involve accurate spectrogram segmentation, dimensionality reduction with techniques like PCA or Kernel PCA, and execution of data augmentation methods such as Time Stretching. ETL transformations will extract and prepare data, including necessary normalizations for consistency. Data selection strategies like Diversity or Importance Sampling ensure a varied dataset for training. Training occurs on specified datasets (ESC-50 and UrbanSound8k) using optimization techniques like Adam or SGD. Post-training, performance evaluation involves metrics like Accuracy, F1 Score, Precision, Recall, and Confusion Matrix on a separate validation set, with detailed reports generated. Optimization, scalability, and efficiency requirements ensure models process varying datasets efficiently. A documentation is done to cover model training, tuning, evaluation, and results interpretation. Robust error handling and logging are crucial for managing potential issues, providing comprehensive logs for analysis. The system will classify audio clips into predefined categories based on the specified research focus.

AI Requirements

Various libraries will play a crucial role in developing, training, and deploying the proposed models for this project. TensorFlow and Keras will serve as foundational tools for implementing deep learning architectures. TensorFlow's Vision Transformer library will be employed for the ViT model, efficiently handling spectrogram images segmented into 16x16 patches. The librosa library will facilitate Mel-scaled Spectrograms, while sklearn's PCA will handle dimensionality reduction. Diversity Sampling using sklearn's k-means clustering and data augmentation with SpecAugment are key components. The Simple CNN model, constructed with Keras, will use PCA for dimensionality reduction and custom Importance Sampling for data selection. Data augmentation will be performed with Time Stretching and Frequency Shifting using librosa. For the CNN-LSTM model, a hybrid architecture will combine Keras, LSTM layers, and sklearn's Recursive Feature Elimination for dimension reduction. Stratified random sampling and the mixup Python library will be used for data selection and augmentation. The CNN followed by the BiGRU model, developed with Keras, will use sklearn's Sparse PCA or Independent Component Analysis for dimension reduction. All models will undergo rigorous evaluation using Accuracy, F1 Score, and Confusion Matrix metrics facilitated by the sklearn library. The training, optimization, and evaluation will take place on Google Colab Pro, leveraging its CPUs, GPUs, and extensive RAM for computational demands and providing a scalable infrastructure for real-time data processing enhancements on Google servers.

Data Requirements

This ambitious project relies on two key datasets: ESC-50 and Urbansound8k. ESC-50, consisting of 2000 labeled environmental audio recordings across 50 categories, and Urbansound8k, with 8732 labeled sound excerpts grouped into 10 urban sound classes, provide a

diverse and comprehensive foundation for audio classification tasks. Both datasets feature meticulous organization, with audio files in standard WAV format and structured directory formats for streamlined data processing. The datasets are easily accessible from their official repositories, simplifying the data acquisition process.

For secure and scalable data storage, Google Cloud Storage (GCS) is the preferred solution, aligning with the project's use of Google Cloud products. Audio files are uploaded to dedicated GCS buckets, while metadata is stored in Cloud SQL, ensuring organized and accessible data storage. The audio data undergoes transformation into spectrogram images to enhance processing efficiency and model training. Both original audio data and spectrogram images are securely stored in GCS.

Preprocessing steps, including Mel-spectrogram transformations, dimension reduction, data selection, and augmentation, are meticulously executed to prepare the data for model training. Supervised learning leverages labeled data from both datasets, and the clear labeling and structured format allow for seamless integration, potentially combining data for unified model training and evaluation if needed.

1.3 Project Deliverable

The project begins with a comprehensive proposal, encompassing a literature review, problem statement, and research objectives. A Work Breakdown Structure (WBS) is then developed, aiding in effective project management and scheduling. Subsequently, an introductory report is crafted, covering project background, requirements, deliverables, and a review of literature and technologies. Data and project management plans are formulated for optimal resource usage and efficient data handling. JIRA is employed for project planning and management, enabling organized tracking of progress.

Moving to the data engineering phase, exploratory data analysis is conducted, employing dimension reduction techniques, data selection, and augmentation strategies. RAW files, modified files, and spectrogram images are loaded into GCP buckets, facilitating data preparation for the Modeling phase. During modeling, proposed models are implemented in parallel, with iterative evaluation and optimization processes documented. Few Graphs or Plots are made through a python notebook and a comprehensive presentation is prepared to explain findings and methodologies. The project culminates with an APA report, adhering to guidelines for professional and standardized documentation. Below Table 1 shows all the deliverables.

Table 1

A Tabular View of Deliverables, Descriptions, and Their Due Dates

Deliverable	Description	Due date
Project Proposal	Document containing the research objectives, including carrying out a literature survey, providing background information, and the problem statement.	09/22/2023
PM tool & WBS	For efficient Project management, a work breakdown structure has been created to have clear deadlines, assignees for all the tasks including start dates and end dates. The WBS has been implemented in JIRA	10/01/2023
Effort estimates & Gantt chart	To create an extensive list of tasks that needs to be carried out by respective people within the start date and end dates of these tasks. A Gantt chart in JIRA using the created hierarchy of Epics > Stories > Tasks	10/08/2023
PERT chart	This is to define various milestones and to come up with a Critical path based on the defined milestones, which help the project to stay on course wrt Critical path.	10/13/2023

Note. Table 1 is continued on the next page.

Deliverable	Description	Due date
Introduction	An APA formatted document encompassing the project's background, requirements, deliverables, exploration of technology and solutions, alongside a review of existing research wrt Proposed models and problem statement	10/15/2023
Data Collection	Both UrbanSound8K and ESC-50 are sourced from freesound.org and are collected for this research from their respective repositories. (UrbanSound8K,Github)	10/18/2023
Data Understanding	Use Urbansounds8k dataset to evaluate a diverse set of proposed deep learning models identified in the literature survey. Additionally, Conduct an in-depth examination of the metadata and assess the compatibility of the ESC50 dataset with the proposed deep learning architectures to determine its suitability for the research objectives.	10/20/2023
Data Engineering	Dealing with unstructured audio data, the idea is to apply Normalization techniques to come up with a normalized dataset, then apply Spectrogram transformations, Data selection strategies and Augmentation techniques to build a richer dataset with minimal labeled data	10/28/2023
Model Development	Implementing proposed models, Vision Transformers, 2-3 CNN, CNN+LSTM, and CNN-BiGRU on the final datasets created after the Data engineering phase. Includes the methodologies applied, comparison studies.	11/19/2023
Evaluation Results	Evaluating the created models on various performance metrics like F1 Score, ROC AUC, etc. based on the model. These results will be documented, and include the evaluation results after applying Optimizations.	11/27/2023
Individual APA formatted Report	APA formatted document which includes all the sections defined as part of the project scope. It's going to be on an individual level, pursuing one specific model from the proposed models.	12/04/2023

Note. Table 1 is continued on the next page.

Deliverable	Description	Due date
Group APA formatted report	This is an APA formatted document which covers all the work that has been done as a group covering Introduction, proposed problems, methodologies, approaches, models, results, evaluations, etc. The idea is to keep it as exhaustive as possible covering all the required details, without having any unnecessary verbose, so that the work can be extended in future, if needed.	12/04/2023

1.4 Technology and Solution Survey

The integration of multiple research findings forms the foundation of the proposed models, particularly relating to the utilization of deep learning architectures and sophisticated data management tactics for sound classification.

Bahmei et al. (2022) proposed environmental sound classification with a CNN-RNN network and DCGAN data augmentation. CNNs extract audio features but lack temporal modeling. RNNs model sequence data poorly. The paper addresses this by combining CNNs and RNNs. DCGANs generate diverse spectrogram samples for data augmentation to address limited labeled training data. A CNN-RNN network is proposed, where the CNN extracts spectrogram features and an RNN models label relationships over time. MFCC and other representations are extracted from audio clips. Transfer learning on VGG-19 extracts bottleneck features. A DCGAN generates new samples mixed with real data for training, improving performance. Experimental results show the approach outperforms other methods on UrbanSound8K, achieving a 93.5% frame-level accuracy. This enables environmental sound classification with limited labeled data through advances in networks and data augmentation.

Lim et al. (2023) introduced EnViTSA, an ensemble of pretrained Vision Transformers trained on log mel-spectrograms augmented using SpecAugment, for acoustic event

classification (AEC). AEC involves classifying acoustic signals/events based on attributes captured in time-frequency representations like spectrograms. While deep models achieve SOTA for AEC, they require large labeled datasets and are susceptible to overfitting with limited data. The paper addresses this by generating fixed-length spectrogram representations from raw audio using STFT, and synthetic data using SpecAugment to improve generalization. It also uses Vision Transformers which capture global dependencies better than CNNs for temporal spectrograms. An ensemble approach is employed to minimize biases and effectively reduce overfitting, combining predictions from bagged ViTs. Evaluated on UrbanSound8K, ESC-50 and ESC-10 benchmarks, EnViTSA achieves SOTA performance outperforming approaches like EnvNet and ESResNet, demonstrating the potential of ViTs and data augmentation for AEC, while providing valuable insights and a benchmark for future research in this domain.

According to Park et al. (2019), SpecAugment proposes a simple data augmentation technique for automatic speech recognition that applies deformations like time warping, frequency masking, and time masking directly to input log mel spectrograms. Various model architectures, training schedules, and augmentation policy parameters are evaluated on Listen, Attend and Spell models for LibriSpeech and Switchboard tasks, demonstrating that SpecAugment significantly improves performance over non-augmented training by converting the problem from overfitting to underfitting. The best models achieve state-of-the-art results, such as 2.8%/6.8% word error rate on LibriSpeech without a language model or 2.5%/5.8% with shallow fusion, while for Switchboard the WER is reduced to 7.2%/14.6% without an LM and 6.8%/14.1% with fusion. Analysis shows that underfitting is addressed through larger models and longer training, while time warping contributes less than masking and label smoothing can destabilize training with augmentation.

Arjit et al. (2021) proposed a new audio augmentation technique called SPLICEOUT, which improves upon time masking by entirely removing spliced intervals of consecutive time steps from the input spectrogram rather than masking them, making it more computationally efficient. Extensive experiments apply SPLICEOUT to tasks like ASR on Librispeech and CommonVoice, speech translation on LibriTrans, audio classification on ESC-50, UrbanSound8K and GTZAN, as well as representation learning using semi-supervised and self-supervised methods, demonstrating it performs comparably or better than time masking while establishing it as an effective drop-in replacement that serves as a simple yet broadly applicable augmentation technique for speech and audio tasks across different models, datasets and learning paradigms, sometimes providing additional gains when combined with other augmentations like time masking.

Gazneli et al. (2022) introduced an efficient end-to-end model called EAT (Audio Transformer) consisting of convolutional feature extraction followed by a transformer encoder, alongside two novel data augmentations - FreqMix and PhaseMix - that mix audio samples in the frequency and phase domains. Extensive experiments on datasets such as ESC-50, UrbanSound8K, SpeechCommands and AudioSet demonstrate that the proposed approach achieves state-of-the-art or competitive classification accuracy while maintaining a lightweight model size comparable to MobileNetV2, establishing the efficacy of end-to-end audio classification through efficient neural architectures and custom data augmentation techniques that leverage characteristics of the raw audio signal domain for improved performance.

The paper by Koutini et al. (2022) proposed Patchout, a technique to randomly drop parts of the transformer input sequence during training, which significantly reduces the quadratic training complexity of audio transformers to linear time. This approach, along with other

methods like disentangled positional encodings and reducing transformer depth, enables efficient training of audio transformers on large datasets using consumer GPUs. Patchout also acts as a regularizer and improves generalization, achieving new state-of-the-art results on Audioset tagging and outperforming CNNs on downstream tasks with up to 8x faster training. The paper presents Patchout and additional complexity reduction methods for audio transformers, demonstrating how they can be efficiently pre trained and fine-tuned for applications in audio classification, while improving accuracy over CNN baselines through superior modeling capabilities.

Mo and Lam (2020) addressed the difficulty of acquiring large annotated telephony speech datasets by proposing an effective data augmentation method using RPCA to artificially generate such data. RPCA is applied to separate telephone channel characteristics from genuine call recordings, and clean speech is augmented with extracted characteristics at different SNRs. Experimental results show automatic speech recognition models trained on the artificially generated telephony data achieve up to a 7.8% relative WER reduction over the baseline. Further analysis finds the characteristic with the lowest spectral centroid, narrowest bandwidth and most steady texture improves robustness the most, demonstrating this unsupervised technique can both augment data quantity and quality for telephony ASR in a low-cost manner by leveraging existing clean speech resources and minimal call recordings.

Chhikara (2021) addressed environmental noise classification by investigating transfer learning techniques to train CNNs on urban sound datasets. The models leverage audio augmentation to increase dataset size, extract normalized log-mel spectrograms as input, and are pre trained on ImageNet for initialization. Xception, MobileNetV2 and DenseNet are evaluated, with Xception achieving the highest accuracy of 0.81 outperforming the baseline. Training is

optimized through cyclic learning rate, AdamW optimizer with decoupled weight decay, allowing models to converge faster with augmented data. This effective transfer learning approach demonstrates how limited annotated audio data can be leveraged through preprocessing, augmentation, and optimized hyperparameters to develop tools for urban noise monitoring and regulation through environmental sound classification. Below is Table 2 comparing all the above discussed different technology surveys.

Table 2

Comparison of Technical Survey

Authors	Technical Approaches	Models Used	Evaluation Metrics	Results	Conclusion
Bahmei et al. (2022)	Proposed CNN-RNN network with DCGAN data augmentation	CNN-RNN feature extraction and DCGAN augmentation	Accuracy on ESC-50 dataset	CNN-RNN with augmentation outperforms baselines.	CNN-RNN with augmentation classifies sounds with less data
Lim et al. (2023)	Ensemble of Vision Transformers, SpecAugment	Vision Transformers and SpecAugment	Accuracy on UrbanSound 8K, ESC-50, and ESC-10 datasets	Achieved 93.5%, 85.85%, 83.2% accuracies	Vision Transformers and SpecAugment classify acoustic events.
Park et al. (2019)	Proposed SpecAugment with warping, time and freq masking.	Applied SpecAugment features for speech recognition.	Word error rate on LibriSpeech and Switchboard datasets.	Achieved best LibriSpeech and Switchboard WERs.	SpecAugment enhances speech recognition

Note. Table 2 is continued on the next page.

Authors	Technical Approaches	Models Used	Evaluation Metrics	Results	Conclusion
Arjit et al. (2021)	Proposed SPLICEOUT removes splices augmentation.	Applied SPLICEOUT augmentation for audio models.	Phone and word error evaluations on tasks	Achieved lower error than augmentation	SPLICEOUT augmentation enhances diverse audio machine learning
Gazneli et al. (2022)	Efficient end-to-end network with novel augmentations	MobileNetV2 backbone with SE blocks. Extensive augmentations	Accuracy and efficiency on audio datasets	Achieved best accuracy with fewer resources	Audio augmentations boost lightweight model classification efficiency
Koutini et al. (2022)	Proposed Patchout augmentation masking features	Applied Patchout to train transformers.	Accuracy on AudioSet and urban sounds	Achieved best results with speed	Patchout enables efficient audio transformer training
Mo and Lam (2020)	Proposed RPCA augmentation for telephone audio	Applied RPCA to decompose audio for augmentation	WER of ASR models with augmentation	RPCA augmentation bests baselines on phones	RPCA improves noisy audio ASR.
Chhikara (2021)	Transfer learning with augmentations for noise class	Compared models with cyclic rates and AdamW	Accuracy on urban sound datasets	Xception achieved highest urban accuracy	Transfer learning aids noise tools with less labels

1.5 Literature Survey

According to Diez et al. (2023), machine learning techniques have enabled sound monitoring systems to classify noise sources, improving on traditional systems that only measure

noise levels. Researchers have developed systems using deep neural networks like CNNs, achieving good results but facing challenges of limited real-world data and labeling large datasets. Previous datasets focused more on general sounds than urban noise. This research aims to address these challenges by collecting a new database called NoisenseDB from urban noise recordings and exploring semi-supervised learning to automate dataset labeling at scale. The paper evaluates state-of-the-art classification systems on the new database to provide baseline performance and investigates using self-training to iteratively refine auto dataset annotations.

Yazgaç and Kirci (2022) proposed two novel data augmentation methods for audio signals based on fractional-order calculus. The first method warps the Mel scale using fractional differentiation, while the second applies a fractional-order differential mask. By augmenting an environmental sounds dataset, experiments show the methods improve classification accuracy when used with CNNs. Deeper networks also benefit, gaining over 9% accuracy. The paper demonstrates how fractional calculus can be adapted to augment audio data, opening up a new area of research beyond conventional techniques. The aim was to better handle variations in audio and reduce overfitting in sound classification models.

Wang et al. (2022) proposed a lightweight convolutional neural network combined with data augmentation for vehicle detection using audio recordings from an intelligent sensor system. Spectrogram augmentation and a depth wise separable CNN structure are used to improve the model while reducing parameters. Experiments on field recordings show the approach achieves 94.64% detection accuracy and can run in real-time on sensor chips. The goal was to develop an efficient and robust solution for vehicle detection that addresses limitations from ambient noise and sensor placement.

According to Gong et al. (2021), Over the past decade, convolutional neural networks (CNNs) have become the dominant model for audio classification as they can learn representations from raw spectrograms. Recent work adds self-attention on top of CNNs to better capture long-range context. However, it is unclear if CNNs are necessary. The Transformer achieved success without CNNs in vision. This paper introduces the Audio Spectrogram Transformer (AST), the first pure attention model for audio classification. The paper evaluates AST on several benchmarks to show purely attention-based models can outperform CNN attention hybrids for audio tasks. AST achieves new state-of-the-art results without relying on CNNs by leveraging insights from vision Transformer models pretrained on ImageNet. Extensive experiments demonstrate AST's superior performance, support for variable-length inputs, and simpler architecture compared to CNN-attention hybrid baselines.

According to Dosovitskiy et al. (2021), Previous works have applied self-attention to computer vision by combining it with convolutional networks or replacing some CNN components. However, these models still rely heavily on CNN architecture and have not been scaled effectively. Inspired by the success of Transformers in NLP, this paper experiments with applying a standard Transformer directly to images with minimal modifications. Existing works that do this either use small patch sizes limiting the image resolution or are trained in an unsupervised manner, achieving lower classification accuracy. This paper aims to demonstrate that by training a pure Transformer (ViT) on large datasets in a supervised manner, it can achieve competitive or better performance than state-of-the-art CNNs on several image classification benchmarks, requiring substantially less computational resources for training.

Khan and Naseer (2022) in their survey provided a comprehensive overview of transformer models developed for computer vision applications. It develops a taxonomy of

network designs and highlights strengths and limitations of existing methods. Previous reviews mainly focus on NLP or generic attention, but this survey specifically organizes recent approaches based on visual transformers. It covers applications of transformers in popular vision tasks like classification, detection, segmentation as well as multi-modal tasks, video processing, low-level vision and 3D analysis. The author compares advantages and limitations of techniques in architectural designs and experiments. Finally, it analyzes open problems and outlines future research directions to further the application of transformers in computer vision.

Liu et al. (2023) reviewed recent work on using Transformers for audio classification. Early works applied CNNs to audio classification, but newer methods adapt Vision Transformers to audio data. However, existing audio Transformers mainly borrow structure from Vision Transformers without optimizing for audio. This paper aims to introduce an audio Transformer with specialized Multi-Resolution Multi-Filter feature extraction and an acoustic attention mechanism to better capture temporal-frequency semantics from audio. It also proposes a causal module to address overfitting issues and improve generalization. The paper surveys relevant prior work on audio classification, Transformers, causal inference techniques, and aims to develop a more optimized Transformer architecture for audio classification.

Das et al. (2020) utilized LSTMs for urban sound classification, demonstrating the effectiveness of recurrent networks for capturing temporal dependencies in audio data. The model achieved an outstanding 98.81% accuracy on the task. The LSTM architecture was able to learn patterns from the full audio sequences and classify sounds accordingly. This showed that recurrent networks are well-suited for audio classification given their ability to process sequences of data like audio.

Zhao et al. (2022) introduced an adaptive few-shot learning algorithm for sound event

detection, especially when limited training examples are available. The algorithm outperformed traditional single-shot methods under low-resource conditions. It adapted classification parameters dynamically based on the support set during meta-training and meta-testing. This allowed it to generalize better to new sound classes versus static parameter methods. The paper demonstrated the benefits of tailored few-shot learning for acoustic scenes with rare events.

Demir et al. (2020) proposed a hierarchical pyramidal CNN architecture integrating SVMs for classification. It achieved highly competitive accuracy scores of 94.8%, 81.4%, and 78.14% on the diverse ESC-10, ESC-50, and UrbanSound8K datasets respectively. The pyramidal structure extracted multi-level features by stacking convolutional and max pooling layers. Combining this with SVMs improved classification performance, indicating the advantages of ensemble methods.

Zhang et al. (2020) employed an attention-based convolutional recurrent network emphasizing important temporal relations within complex soundscapes. It obtained notable accuracies of 83.5% for ESC-50 and 78% for UrbanSound8K. The attention mechanism allowed the model to focus on discriminative sections in the audio sequences, improving environmental sound recognition.

Furthermore, Massoudi et al. (2021) used a conventional CNN model with MFCCs and achieved 91% accuracy. MFCCs captured characteristic features from the spectrograms which the CNN could then learn from, demonstrating the usefulness of standard features and classifiers for audio classification tasks.

According to Turab et al.(2022), Prior research has investigated audio classification using various audio features and deep learning algorithms with success. Frequently used audio attributes involve mel spectrogram, MFCC and Zero Cross Rate ZCR. While some efforts

utilized a single attribute, others amalgamated features or ensemble models. However, many failed to perform an exhaustive evaluation of diverse feature combinations. This work aims to explore merging multiple audio characteristics with different deep architectures seeking the highest performing grouping for enhanced audio categorization. Thorough experiments are conducted on multiple datasets to gauge alternative combinations.

Xueli et al.(2021) proposed a forecasting model using data-driven techniques with differing levels of achievement. Traditional machine learning algorithms struggle to analyze spatial-temporal connections in output statistics. Deep artificial intelligence architectures like CNNs, RNNs and their versions have accomplished enhanced outcomes but still have confines. This investigation targets creating a combined deep learning model joining CNN, BiGRU and attention mechanism to extrapolate spatio-temporal characteristics more competently and anticipate shale oil generation. Extensive experiments are led to assess the model and compare it with other strategies.

Earlier investigations have looked into heartbeat sorting utilizing distinctive machine learning and deep learning strategies with fluctuating degrees of achievement.According to Yadav et al.(2023), Conventional machine learning algorithms battle to decode spatial-temporal connections in sound information. Profound learning designs like CNNs and RNNs can accomplish better outcomes yet even now have disadvantages. This examination expects to build up a blended CNN-BiGRU-Attention system to all more productively remove spatial-transient qualities from coronary sound cuts and anticipate heartbeat order. Broad testing is led to systematically survey the display and look at it against other techniques considered in past examinations.

Zeng et al.(2021) research paper aims to develop a novel CNN-BiGRU network with an

attention mechanism to better extract the spatial and temporal characteristics from audio signals and predict audio classification. The network combines convolutional layers to extract spatial features from input variables, BiGRU layers to derive temporal features, and an attention layer. Extensive testing is conducted to systematically evaluate the model's ability to forecast audio categories and compare it to other state-of-the-art methods discussed in prior literature. Table 3 shows the summary of the literature survey.

Table 3

Comparison of Literature Survey

Paper	Year	Venue	Methodology
Diez et al. (2023)	2023	MDPI	CNNs and Transformer based Audio Classification System(AST)
Yazgaç and Kırcı (2022)	2022	MPDI	Fractional-order calculus-based data augmentation methods for audio signals
Wang et al. (2022)	2022	MPDI	MFCC features, transfer learning to train and compress CNNs with spec augmentation
Gong et al. (2021)	2021	Arxiv	Extracting MFCC features, augmenting the data with a DCGAN, training a CNN-RNN network with transfer learning on original and augmented data
Dosovitskiy et al. (2021)	2021	Arxiv	Applying a standard Transformer encoder directly to sequences of image patches
Khan and Naseer (2022)	2022	ACM Computing Surveys	Applying self-attention and Transformers to computer vision tasks

Note. Table 3 is continued on the next page.

Paper	Year	Venue	Methodology
Liu et al. (2023)	2023	IEEE Conference Publication	Extracting Multi-Resolution Multi-Filter (MRMF) spectrogram features
Das et al. (2020)	2020	ICDS	Combining convolutional and LSTM neural networks with multiple audio features
Zhao et al. (2022)	2022	International Joint Conference on Neural Networks (IJCNN)	Proposes an adaptive prototype learning approach using meta-learning to detect rare sound events
Demir et al. (2020)	2020	ACM International Conference on Multimedia	Proposes a pyramidal concatenated CNN approach for ESC
Zhang et al. (2021)	2021	Neurocomputing	Proposes attention convolutional RNN for ESC focusing on salient frames with classification experiments
Massoudi et al. (2021)	2021	International Conference on Inventive Computation Technologies	Proposes CNN for urban sound classification using Mel-spectrograms and data augmentation
Turab et al.(2022)	2022	(International Journal of Artificial Intelligence and Applications)	Proposes MFCC, MelSpectogram are major for features extraction among all other features extraction techniques
Zeng et al.(2021)	2021	EURASIP Journal on Advances in Signal Processing	Proposed audio data needs to be evaluated fully in terms of spatial and temporal features in order the maximum of features extraction from audio data

Note. Table 3 is continued on the next page.

Paper	Year	Venue	Methodology
Xueli et al.(2021)	2021	IEEEXPLORE	Proposed CNN-BiGRU-Attention model for extracting local and temporal features
Yadav et al.(2023)	2023	MDPI	Proposes overview of CNN-BiGRU models

2 Data and Project Management Plan

2.1 Data Management Plan

Data Collection Approaches

The datasets that have been shortlisted for this research project are publicly available. Urbansound8k and Environmental Sound Detection-50 (ESC-50), which basically are a set of audio recordings for various sound categories like Glass Shattering, Dog Barking, Motor, Engine Off, Engine On, etc. The ESC-50 dataset has 50 distinct classes of 40 recordings each of 5 seconds each. This dataset was curated and proposed by Piczak (2015) in his research paper titled “ESC: Dataset for Environmental Sound Classification”, where he introduced the dataset, and described various other details and use cases.The other dataset, Urbansound8k dataset, is introduced in “Dataset Taxonomy for Urban Sound Research” by Salamon et al. (2016). This dataset contains 10 classes like engine_idling, gunshot, siren, Jackhammer, etc. There are a total of 8732 audio excerpts which are <= 4 seconds, these are directly sourced from freesound.org. The primary aim of this research is to exploit the limited labeled audio data available from two datasets sourced from Freesound.org, specifically within the Categories tagged sounds.

Freesound.org, spearheaded by the Music Technology Group at Universitat Pompeu Fabra, functions as an extensive repository housing a varied collection of sound effects, ambient

sounds, and diverse audio snippets. Embracing a community-driven model, the platform actively encourages contributions from both seasoned professionals and amateur sound recordists.

The audio samples cover a spectrum from natural ambient sounds to synthesized effects and are predominantly hosted under various Creative Commons licenses, facilitating widespread accessibility. Nevertheless, some sounds may necessitate attribution or compliance with specific usage restrictions. In the process of recording and digitizing audio at a 44.1 kHz frequency, a meticulous seven-step approach is employed.

The process initiates with the capture of analog sound waves through a thoughtfully selected microphone. The subsequent steps involve processing these signals through an Analog-to-Digital Converter (ADC) at a rate of 44.1 kHz, ensuring precise amplitude determination at each sample point. The choice of microphone plays a pivotal role in signal quality, particularly concerning its representation in the .wav file format. The high sample rate guarantees accurate representation across a broad spectrum of audio frequencies. Post quantization, a digital audio file in the .wav format is generated, offering flexibility for storage on various media or digital transmission. During playback, a Digital-to-Analog Converter (DAC) is employed to transform the digital signal back into an analog format.

This comprehensive process adheres to industry standards, striking an optimal balance between capturing diverse frequencies and efficiently managing file sizes, especially when utilizing the .wav file format.

Regarding Environmental Sound Classification data, the recording duration is 5 seconds, whereas for urbansound8k, it is <=4 seconds.

Below is Table 4 and Table 5 that summarizes various details of both ESC-50 and UrbanSound8K datasets.

Table 4

Summary of ESC-50 Dataset

About Dataset	Details/Description
Dataset Name	Environmental Sound Classification - 50 (ESC-50)
Dataset Source	Github Repository sourced from freesound.org
Research Paper	ESC: Dataset for Environmental Sound Classification
Authors	Karol J. Piczak
Categories	Animal, Natural soundscapes & water sounds, Human, non-speech sounds, Interior/domestic sounds, Exterior/urban noises
Classes	Dog, Rooster, Pig, Cow, Frog, Cat, Hen, Insects (flying), Sheep, Crow, Rain, Sea waves, Crackling fire, Crickets, Chirping birds, Water drops, Wind, Pouring water, Toilet flush, Thunderstorm, Crying baby, Sneezing, Clapping, Breathing, Coughing, Footsteps, Laughing, Brushing teeth, Snoring. Drinking, sipping, Door knock, Mouse click, Keyboard typing, Door, wood creaks, Can opening, Washing machine, Vacuum cleaner, Clock alarm, Clock tick, Glass breaking, Helicopter, Chainsaw, Siren, Car horn, Engine, Train Church bells, Airplane, Fireworks, Handsaw.
No. of recordings	2000 (40 of each class)
duration	4 seconds
No. of classes	50
File Format	.wav
Formatting	44.1kHz,single channel,Ogg Vorbis compression at 192kbit/s

Table 5

Summary of UrbanSound8k Dataset

About Dataset	Details/Description
Dataset Name	Urbansound8k
Dataset Source	Urbansounddataset sourced from freesound.org
Research Paper	A Dataset And Taxonomy for Urban Sound Research
Authors	Justin Salamon,Christopher Jacoby,Juan Pablo Bello
Categories	Urban sounds
Classes	air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer, siren, street_music
No. of recordings	8732
duration	<= 4 seconds
No. of classes	10
File Format	.wav
Formatting	sampling rate, bit depth, and # channels same as those of the original file uploaded to Freesound, which means they may vary for file to file.

Key features of Freesound include an advanced search system, allowing users to efficiently locate specific sounds through filters like duration, file type, and bit rate. The platform

also facilitates an active forum for discussions, sound requests, and knowledge exchange on audio-related topics. Moreover, Freesound offers an API for integrating its sounds into other projects or applications, making it a useful resource for developers and researchers. Its tagging system, sound analysis tools, and visualizations like waveforms and spectrograms provide additional functionalities, enhancing user experience and understanding of sound properties. Freesound's broad utility spans across various domains, including podcast production, video game development, and filmmaking, marking it as an invaluable asset in digital audio and multimedia production.

Understanding the source and differences between Urbansound8k, and ESC-50 helps in understanding whether the audio files are good enough already (Urbansound8k directly from Freesound) or after the formatting (in the case of ESC-50). The size of ESC-50 is around 1 GB, and the urbansound8k is around 6 GB.

Data Storage and Management Methods

In terms of storing these audio datasets are stored in separate S3 buckets across various geos to create multiple backups of the files. The bucket is created in us-east regions for one copy, and another copy on shared secured drives. These datasets are easily accessible through the URL's, mentioned above. Urbansound8k requires the user to sign a consent form, and then allows the user to download the whole dataset. Each of these audio datasets also have their respective metadata files where the nomenclature of each file is mentioned, and how the 10-fold cross validation & 5-fold cross validation are the only ways to do the validation on these datasets to get accurate interpretable results. The metadata files are stored in their respective S3 buckets and drive folders. The total space for UrbanSound8k's raw data, totaling 21 GB (7 GB each in AWS S3, Google Cloud, and shared drive), ensures redundancy, accessibility, and backup.

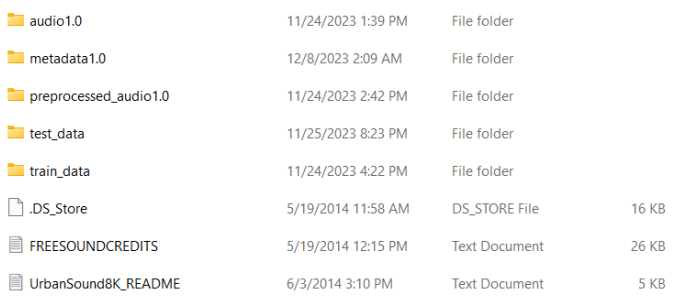
Simultaneously, 20 GB of pre-processed data is stored in AWS S3, Google Cloud, and a shared drive for retrieval and collaborative analysis. Copies of training and testing datasets (80% and 20% split) are maintained separately in AWS S3, Google Cloud, and shared drives for redundancy and accessibility. Additionally, 21 GB of UrbanSound8k data undergoes archival in AWS S3 Glacier and Google Cloud Coldline Storage for cost-effective, long-term preservation. For Environmental Sound Classification data, 1 GB of raw data is redundantly stored in AWS S3, Google Cloud, and shared drives.

The 6 GB pre-processed data is organized in AWS S3, Google Cloud, and shared drives for easy retrieval and collaborative analysis. Training and testing datasets, combining raw and pre-processed audio files in an 80% and 20% split, are housed in AWS S3, Google Cloud, and shared drives for redundancy and accessibility. The 7 GB Environmental Sound Classification data is archived in AWS S3 Glacier and Google Cloud Coldline Storage, following archival policies for efficient, cost-effective, and long-term data preservation. This structured approach ensures data integrity, accessibility, and cost-effectiveness across both datasets, aiding in disaster recovery and backup scenarios.

Below is Figure 2 for the folder structure and names.

Figure 2

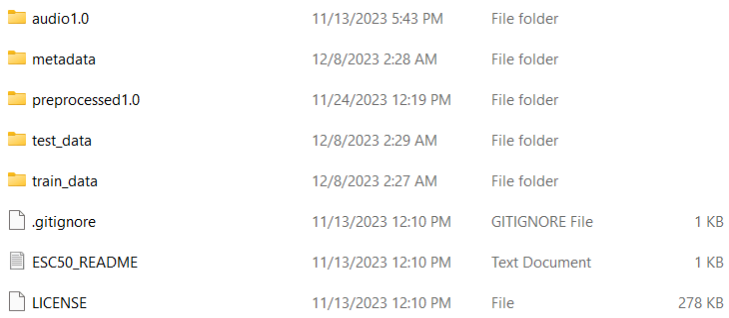
UrbanSound8k Folder Structure



The UrbanSound8k directory comprises five subfolders and three distinct files, each holding significance within the dataset. Within the audio version 1.0 folder, the unprocessed audio data is stored, sourced from freesound.org. The metadata version 1.0 folder contains a CSV file, illustrated in Figure 3, providing detailed audio metadata. Meanwhile, the preprocessing version 1.0 folder contains the audio files after undergoing preprocessing. Notably, the "UrbanSound8k_README" file offers version information, currently designated as version 1.0. If new data is added from freesound.org or recorded for project extension purposes, the version is incremented accordingly, moving from 1.0 to 2.0. This versioning is applied to the respective folder and individual audio files. Additionally, this file includes the "FREESOUNDCREDITS," recognizing contributors crucial to obtaining specific audio sounds. Figure 3 below depicts the file structure for the ESC-50 audio dataset.

Figure 3

Environment Sound Classification-ESC50 Folder Structure



The above Figure 3 shows the structure of the ESC-50 directory which is organized into five distinct folders, each serving a specific purpose. Initially, the audio folder contains the unprocessed audio data 1.0, while the metadata folder holds detailed information about the audio files. Additionally, the preprocessed folder 1.0 stores audio files that have undergone preprocessing and are saved in .wav format. Lastly, the dataset is divided into training and testing

data, with an 80% and 20% split, respectively.If there is any new audio data collected or recorded for the project extension purpose the version is going to be changed. For Example, Version 2.0 etc.This systematic structure facilitates efficient access and utilization of the ESC-50 dataset. Below is Figure 4 and Figure 5 which represents the sample of the metadata of respective files and its file formats along with start and end duration along with their respective class types of each audio.

Figure 4

Metadata of The UrbanSound8k Dataset (10 Folds, 10 Classes, 8732 .wav Files, <= 4 seconds Duration)

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.000000	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.500000	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.500000	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.000000	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.500000	72.500000	1	5	2	children_playing
...	...	...	...	...	...	...	...	...
8727	99812-1-2-0.wav	99812	159.522205	163.522205	2	7	1	car_horn
8728	99812-1-3-0.wav	99812	181.142431	183.284976	2	7	1	car_horn
8729	99812-1-4-0.wav	99812	242.691902	246.197885	2	7	1	car_horn
8730	99812-1-5-0.wav	99812	253.209850	255.741948	2	7	1	car_horn
8731	99812-1-6-0.wav	99812	332.289233	334.821332	2	7	1	car_horn

8732 rows × 8 columns

Figure 5

Metadata Snapshot of ESC-50 Dataset (5 Folds, 50 Classes, 2000 .wav Files, 40 .wav Files per Each Class, 5 Seconds Duration)

	filename	fold	target	category	esc10	src_file	take
0	1-100032-A-0.wav	1	0	dog	True	100032	A
1	1-100038-A-14.wav	1	14	chirping_birds	False	100038	A
2	1-100210-A-36.wav	1	36	vacuum_cleaner	False	100210	A
3	1-100210-B-36.wav	1	36	vacuum_cleaner	False	100210	B
4	1-101296-A-19.wav	1	19	thunderstorm	False	101296	A
...	...	...	...	...	...	...	...
1995	5-263831-B-6.wav	5	6	hen	False	263831	B
1996	5-263902-A-36.wav	5	36	vacuum_cleaner	False	263902	A
1997	5-51149-A-25.wav	5	25	footsteps	False	51149	A
1998	5-61635-A-8.wav	5	8	sheep	False	61635	A
1999	5-9032-A-0.wav	5	0	dog	True	9032	A

2000 rows × 7 columns

Data Usage Mechanism

For the UrbanSound8k dataset, each audio file follows a structure for file names, which is [fsID]-[classID]-[occurrenceID]-[sliceID].wav structure for filenames.

UrbanSound8k Dataset Filename Structure. Below is the file structure for the same.

[fsID]. Freesound ID of the recording from which this excerpt (slice) is taken.

[classID]. Numeric identifier of the sound class

[occurenceID]. Numeric identifier to differentiate distinct occurrences of the sound within the original recording.

[sliceID]. Numeric identifier to distinguish different slices taken from the same occurrence.

[File_Format]. It specifies the file format which is .wav which is wave format. Example is 7061-6-0-0.wav

7061 is the freesound_id which refers to the respective file, It belongs to the class_id = 6, which is gun_shot, 0 is the occurence_id, meaning it is the first occurrence of this type of sound within the original Free Sound recording, slice_id = 0 indicates that it is the first slice taken from this particular occurrence of the sound.

These audio files are not further sampled. They are similar to their sources from freesound.org. Components like sampling rate, bit depth, and # channels may vary for file to file. Some of the other mentioned heads up pointers for Urbansound8k dataset include,

Use Predefined 10-Fold Cross Validation Without Reshuffling Data. It's crucial to adhere to the 10-fold cross-validation method using the already defined splits in the UrbanSound8K dataset. Reshuffling the data or opting for a random train/test split can lead to inaccuracies. This happens because related samples might end up both in training and testing

sets, giving an inflated and unrealistic performance score. Sticking to the predefined splits ensures a valid comparison with prior studies, maintaining the integrity and comparability of results.

Adhere to Predefined 10-Fold Cross-Validation Method. For effective validation, it's important to conduct 10-fold cross-validation strictly following the predefined folds in the UrbanSound8K dataset. This involves using nine out of the ten predefined folds for training and the remaining one fold for testing. This process should be repeated ten times, with each iteration utilizing a different combination of nine folds for training and the remaining fold for testing. The final step is to report the classification accuracy, which should be calculated as an average score across all ten iterations. For a more comprehensive understanding, include the standard deviation or, ideally, present the results in a boxplot to visualize the variation and distribution of accuracy scores.

Similarly, for the Environmental Sound Classification-50 dataset, each audio file follows the [FOLD]-[CLIP_ID]-[TAKE]-[TARGET].wav structure for filenames.

ESC-50 Dataset Filename Structure. Below is the file structure for the same.

[FOLD]. Index of the cross-validation fold,

[CLIP_ID]. ID of the original Freesound clip,

[TAKE]. Distinction between different fragments from the same Freesound clip.

[TARGET]. Class in numeric format [0, 49]

[File_Format]. It specifies the file format which is .wav which is wave format.

Example is 1-137-A-32.wav

So, 1-137-A-32.wav is a sound file from fold 1, originating from Freesound clip ID 137. The above mentioned 1-137-A-32.wav file comes under the first fragment from this clip,

categorized in class 32, which refers to keyboard_typing.

These recordings are 5 seconds long, and have a sampling rate of 44.1 kHz (kilohertz). The sampling rate indicates how many times per second the audio signal is sampled to convert from an analog signal to a digital one. A rate of 44.1 kHz means that the audio is sampled 44,100 times per second, which is the standard rate used for CDs and is generally considered high-quality audio. It captures frequencies up to 22.05 kHz, encompassing the full range of human hearing (approximately 20 Hz to 20 kHz). They are also categorized as monophonic. Mono recordings are simpler and take up less storage space than stereo, but they do not provide the listener with a sense of spatial positioning of the sound sources.

Five-Fold Division for Cross-Validation. The ESC-50 dataset is divided into 5 distinct folds to facilitate cross-validation, ensuring an even and representative distribution of data for both training and testing phases in model evaluation.

Consolidation by Original Source. Fragments from the same original source file are grouped into the same fold. This arrangement prevents data leakage and ensures that the model's performance during testing accurately reflects its ability to generalize to new, unseen data, rather than simply recognizing repeated samples from the training phase.

Distribution of Responsibility between Team Members. The DMP stage of this project is divided into 4 broad phases carried out by different members of the group which is represented in the Table 6 below.

Table 6

Details of The DMP Stage

DMP Phase	Responsible POC	Work Items
Data Collection	Nikhil Thota	Investigating the relevance of these datasets to problem statements. Forms for issue less download of datasets. Understanding the metadata, Collection approaches.
Documentation of Metadata, and Legal compliances.	Hema V.S. Dumpala	Documenting the metadata by clear understanding of individual datasets, their source legal compliances to make the smooth transition into next phases.
Storage and Backup	Venu Anupati	Investigating various cost options for available cloud services. Storing the audio datasets in different buckets in GCP, S3, and storing a backup in the shared drive folders.
Data Sharing	Mounika Vempalli	Checking the compatibility of Data pipelines for processing and Modeling.

The above responsibilities for each POC are further reported to a reporter who is like Responsible, and will act like a bridge between individual tasks to validate whether the process is progressing as expected.

For streamlined Data Management, a central Git repository serves as the collaborative hub for team members to consistently upload project documents. To control costs related to storing extensive datasets and executing resource-intensive processes, the usage of GCP and AWS S3 storage is limited until the project's completion. This strategic decision enables the team to prioritize computational aspects such as sampling techniques, dimension reduction, data

augmentation, and the implementation of deep learning models, all of which significantly impact the overall project cost.

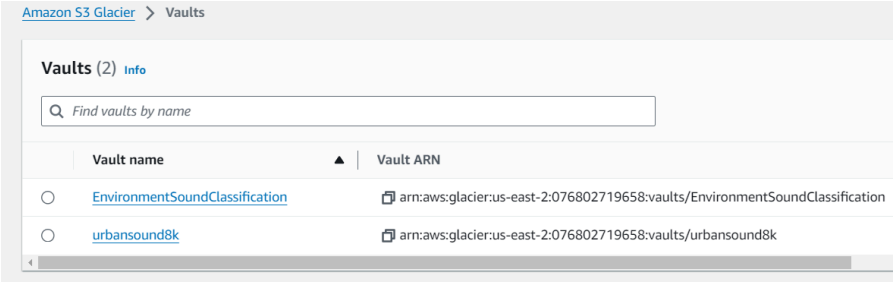
Addressing data security and access management, the team employs Identity and Access Management (IAM) protocols, integrating two-factor authentication (2FA) in place of relying solely on passwords. IAM roles are configured to regulate access, ensuring team members have appropriate permissions based on their roles and responsibilities. The inclusion of 2FA enhances security across the shared repository, GCP, and S3 storage, necessitating both a password and a secondary authentication method. These security measures are crucial for maintaining data integrity and confidentiality within the project, with datasets being freely accessible but subject to licensing restrictions.

The team commits to not sharing data beyond the project team and maintains restricted access on platforms like GitHub, aligning with licensing agreements and safeguarding the intellectual property associated with the datasets. Through these measures, the team adeptly balances collaborative development, cost management, and the rigorous protection of data integrity and intellectual property throughout the project's lifecycle.

Data Archiving and Data Disaster Management. In summary, the decision to store audio data like UrbanSound8k and Environmental Sound Classification in AWS S3 Glacier is driven by a commitment to cost-effectiveness, the prolonged preservation of data, regulatory compliance, efficient storage management, heightened security measures, and consideration for potential disaster recovery scenarios. Archiving aligns with strategic data management principles, emphasizing resource optimization and ensuring the resilience of valuable audio datasets. Refer to Figure 6 for a visual representation of the vaults housing urbansound8k and Environment Sound Classification ESC-50 files for the next two years.

Figure 6

Overview of Audio Files Archival in S3 Glacier Vaults For ESC-50 And UrbanSound8k



2.2 Project Development Methodology

Data Analytics or Intelligent System Development Life Cycle

This research project aims to develop and progress by utilizing the combination of Crisp-dm methodology and Agile Scrum settings. The development cycles will be defined as Sprints (14 days) which are further grouped under Stories, further into Epics respectively. This broad division of work items under various key phases of the project when tracked using the Sprint cycles, and Scrum methods can result in successful delivery of the project deliverables, and conduct the research in the proposed manner.

Planned Project Development Processes And Activities

CRISP-DM methodology has six different phases, which constitute this research project. These phases are Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation and Optimization, and Deployment.

Business Understanding. The primary idea behind this research project was to work on Data Augmentation strategies, and combining them with Deep learning models to get better accuracies and metrics. While looking out for solid problem statements, the plan was to apply the Data Augmentation techniques to those datasets which are very hard to create, or require a lot of

cost and effort. Further from the exhaustive literature survey, various problem domains have been recognized where Data Augmentation can be effectively used to improve the accuracy, further helping in classification use cases.

Once the problem area of Classification has been decided, the next step was to short list the dataset and exact problem statement. There were two serious problem statements this research considered, One was with the EEG Signal data, to classify the human emotion, and the other was for Environmental Sound Classification. The former problem statement couldn't be pursued due to access issues as EEG data was Human data. So, the finalized problem statement that has been decided involved Data Augmentation Strategies, Deep learning models, and various Data Engineering techniques like Dimensional reduction, and Sampling methods to selectively leverage the data, this way there are further minimizing the dataset, to explore better accuracies and better performance metrics.

The information collected during the business understanding phase, and is going to serve as the base for subsequent phases of the project which are Data Understanding, Data Preparation, Data Preparation, Modeling, Evaluation and Optimization, and Deployment.

Data Understanding. Post Business Understanding, another key phase to this research project is Data Understanding. The objective of this phase is pretty much clear. It was to identify the datasets that are going to be used to test and validate the proposed Deep learning models. Data Understanding is considered as one of the key phases, as the basic understanding of data is created and built upon starting this phase.

The datasets collected for this research project are Urbansound8k, and ESC-50 datasets. Each of these datasets have been sourced from Freesound.org, but vary in duration and differ in the pre-processing that was done to increase the quality of the sound. ESC-50 has already been

formatted using sampling at 44.2 kHz, to make the dataset consistent. For the Urbansound8k dataset, there was no particular method followed, and the audio excerpts were directly taken from freesound.org, whereas ESC-50 underwent some sampling formatting, for better quality purposes. These audio datasets are available as .wav files with metadata for the datasets available as separate CSV files respectively. The duration of the clipping also varies in both the datasets.

Due to these visible inconsistencies between the datasets, the team has decided to conduct the validation, and testing phases separately on these datasets. Considering ESC-50 as the primary dataset, and Urbansound8k as a secondary dataset. This will also help in understanding the effects of pre-sampling to improve the quality of the audio clippings from freesound.org.

The next step of this phase is to Clean the dataset. The metadata which was shared along with the datasets is perfectly clean, with no missing values, and misaligned rows, covering all the Labels as per the expectations. This will be covered in the data quality report of the respective datasets, along with the descriptive statistics and various EDAs on Metadata to understand the distribution of Labels across the dataset.

With respect to understanding of the actual audio .wav files, the plan is to convert them into Visual representations such as spectrograms utilizing Mel spectrograms technique, so that Frequency and Amplitude features can be further explored. Temporal and Spectral Features like zero-crossing rate, spectral centroid, spectral bandwidth, and spectral roll-off can also be explored.

Data Preparation. For this research project, the Data Preparation is going to be the key phase, as the whole idea behind this project is to leverage various Data processing techniques like Sampling, Dimension Reduction and Data Augmentation to deliver better Classification accuracies and Performance metrics. Even before starting with any of the mentioned steps, first

the dataset needs to be split into Training, Validation and Testing datasets for ESC-50 and Urbansound8k individually. K-fold validation techniques are to be used for Testing and validation of the Deep learning models.

In the data preparation phase for the proposed audio classification models, each approach begins by converting audio files from the ESC-50 and UrbanSound8k datasets into Mel-scaled spectrograms. For the Vision Transformers (ViT-B/16) and Convolutional Neural Network (CNN) approaches, Principal Component Analysis (PCA) and Kernel PCA are used respectively for dimensionality reduction. The ViT-B/16 approach aims to preserve 95% of the variance, while Kernel PCA in the CNN approach focuses on retaining non-linear variance. The Recurrent Convolutional Neural Network (LSTM) method employs Recursive Feature Elimination for pruning less contributive features, whereas the CNN-BiGRU approach opts for either Sparse PCA or Independent Component Analysis (ICA) to highlight significant features while reducing data dimensions. In terms of data selection, the ViT-B/16 approach utilizes Diversity Sampling with k-means clustering for a varied dataset. The CNN approach employs Importance Sampling to focus on instances with higher prediction errors, the CNN-LSTM approach uses Stratified Random Sampling to maintain class distribution, and the CNN-BiGRU approach opts for Uncertainty Sampling, selecting instances that the model finds most uncertain.

For data augmentation, the ViT-B/16 model employs CycleGAN with a custom loss function designed to retain significant frequency characteristics during style transfer. The CNN model uses Time Stretching and Frequency Shifting to provide a broader representation of data variance. In contrast, the CNN-LSTM model adopts the Mix Up technique, blending spectrogram pairs and their labels. The CNN-BiGRU approach uses "Roll " Augmentation and Random Cropping to enhance dataset variety. These augmentation techniques are carefully

chosen to suit the nature of each model and its handling of audio data, ensuring a comprehensive coverage of possible real-world variations and enhancing the robustness of each model against diverse sound environments and conditions.

Modeling. The modeling phase in the sound classification project involves constructing and tuning deep learning models, specifically Vision Transformers (ViT-B/16), Convolutional Neural Networks (CNN), Convolutional Neural Networks with LSTM (CNN-LSTM), and CNN with Bidirectional Gated Recurrent Units (BiGRU). Each model is adapted and optimized to process complex audio signals for sound classification tasks.

Vision Transformers (ViT-B/16) are adapted to handle audio by processing spectrogram images, focusing on patch size and transformer layers' optimization to effectively capture global acoustic patterns. The CNN model, with its convolutional layers, is fine-tuned for detailed feature extraction from audio spectrograms. Parameters such as kernel size, number of filters, and network depth are adjusted for efficient learning.

The CNN-LSTM model combines CNN's spatial feature extraction with LSTM's temporal dynamics handling. This hybrid architecture is tuned to capture both immediate and sequential patterns in sound data, emphasizing the balance between convolutional and recurrent layers. The CNN-BiGRU model utilizes 2D convolutional layers for frequency pattern identification and BiGRU layers for temporal dependency modeling. This model's optimization involves adjusting the granularity of convolutional layers and the bidirectional processing of GRU layers.

Training each model involves using the prepared datasets, with a focus on fitting the models to data and validating on separate subsets to ensure generalization. Post-training, models are tested using a subset of data, evaluating their performance with metrics like precision, recall,

F1 score, and accuracy. Hyper-parameter tuning is conducted at the final stage, employing techniques like grid search and random search to refine learning rate, batch size, and regularization parameters. This step enhances each model's performance, ensuring accurate and efficient audio data classification.

Evaluation. The evaluation phase of the sound classification project centers around assessing the performance of various deep learning architectures — Vision Transformers (ViT-B/16), Convolutional Neural Networks (CNN), Convolutional Neural Networks with LSTM (CNN-LSTM), and CNN with Bidirectional Gated Recurrent Units (BiGRU) — using k-fold cross-validation. This technique, crucial for ensuring model robustness and generalizability, divides the dataset into 'k' subsets, using each in turn for validation while training on the remainder. This approach not only maximizes the use of available data but also mitigates the risk of model overfitting.

Performance metrics such as accuracy, precision, recall, F1 score, and the confusion matrix are applied to each fold of validation, providing a comprehensive view of each model's strengths and weaknesses. Accuracy offers an overall success rate, while precision and recall delve into the correctness and completeness of the models' predictions in each category. The F1 score bridges the gap between precision and recall, particularly useful in cases of imbalanced class distributions. The confusion matrix further illustrates how well the models distinguish between different sound categories and identifies any common misclassifications.

This phase also includes a comparative analysis, going beyond individual model performance. Factors like computational efficiency, complexity, and scalability are considered, offering insights into the practical applicability of each model. This analysis is instrumental in

identifying not just the most accurate models, but also those best suited for deployment in specific contexts, balancing the trade-offs between performance and resource requirements.

K-fold cross-validation plays a pivotal role in the evaluation phase, providing a reliable measure of model performance and informing decisions on future improvements and potential ensemble strategies. By thoroughly examining each model across various metrics and folds, the project aims to develop sound classification models that are both precise in classification and adaptable to the diverse and dynamic nature of real-world audio data. The criteria for determining the best performing model are multifaceted as below.

High Classification Accuracy. The primary criterion is the model's ability to accurately classify audio data, as reflected in high scores across accuracy, precision, recall, and F1 score.

Consistency Across Folds. Stability and consistency in performance across different folds in the k-fold cross-validation process are crucial, indicating the model's reliability and robustness to different data subsets.

Balance Between Performance and Complexity. Models that maintain a balance between high accuracy and manageable computational complexity are preferred, ensuring feasible deployment in various environments.

Scalability. The model's ability to maintain performance with increasing data volume and complexity is essential, highlighting its applicability to real-world scenarios.

Generalizability. The ability to perform well across diverse audio types and environmental conditions, demonstrating adaptability to real-world audio data.

With its comprehensive approach to performance assessment and criteria for best performing model, it guides the selection of the most suitable model(s) for deployment.

Deployment. For deployment, the idea is to create multiple visualizations to

show the findings, results and observations from this research project. These visualizations will be informative. The documentation will cover all the various phases of this research project. The deployment of this model on cloud is out of the scope of this research project, so the aim is to study the impacts of Data Augmentation and Dimension reduction techniques on Deep learning architectures.

The documentation, and a presentation for the wider audience groups, along with the interactive Power Bi dashboards to show the results, findings and EDA's of the research project are the only tasks as part of the deployment stage of this project. Once the defined deployment is carried out, it's vital to document the methodologies and results for future reference. All related data and code implementations are uploaded to a GitHub repository, which is set with restricted access to maintain confidentiality and security. A specific SSH key is generated for team members, allowing secure access to this data while safeguarding the information.

2.3 Project Organization Plan

CRISP-DM methodology combined with Scrum Agile methodology is going to form the base of the Project Organization and Management plan. The first level of work breakdown is done through WBS (work breakdown structure) diagram which basically divides the project into six phases as per the CRISP-DM methodology, these six phases are managed using Agile methodology with a defined sprint duration. There can be multiple sprints spanning over a single phase, in this case Modeling spans over two sprints due to the amount of work it requires to complete.

Work Breakdown Structure

WBS is the highest level of overview which helps in dividing the work into defined phases with a set of defined work packets and deliverables. Throughout the accumulation of

these work packets and deliverables, the research project will be carried out. WBS for this project broadly goes as follows with the Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation and Optimization, and Deployment. Below are the detailed explanations of all the deliverables.

Business Understanding. This phase sets the foundation for the project by defining objectives, requirements, and expected outcomes. It involves a thorough understanding of the project's goals, the problem domain, and how the proposed solution aligns with the business strategy. This understanding guides the entire project, shaping decisions and strategies in subsequent phases.

Data Understanding. In this phase, a deep dive into the available data is conducted. It involves analyzing the nature, quality, and intricacies of the data collected. Key activities include exploring the data's characteristics, identifying potential challenges, and understanding the data's structure, which are crucial for informed preprocessing and modeling decisions.

Data Preparation. This phase focuses on converting raw data into a format suitable for modeling. Tasks include data cleaning, feature selection, feature engineering, and scaling. Techniques like PCA are used to emphasize important features, and the data is partitioned into training and testing sets, preparing it for application in various machine learning models.

Modeling. Various models such as Vision Transformers (ViT-B/16), CNNs, CNN-LSTM, and CNN-BiGRU are developed and tuned to process and classify audio data. This phase involves selecting appropriate model architectures, fine-tuning parameters, and balancing model accuracy with computational efficiency.

Evaluation And Optimization. Models are rigorously evaluated using methods like k-fold cross-validation and assessed with metrics such as accuracy, precision, recall, and F1

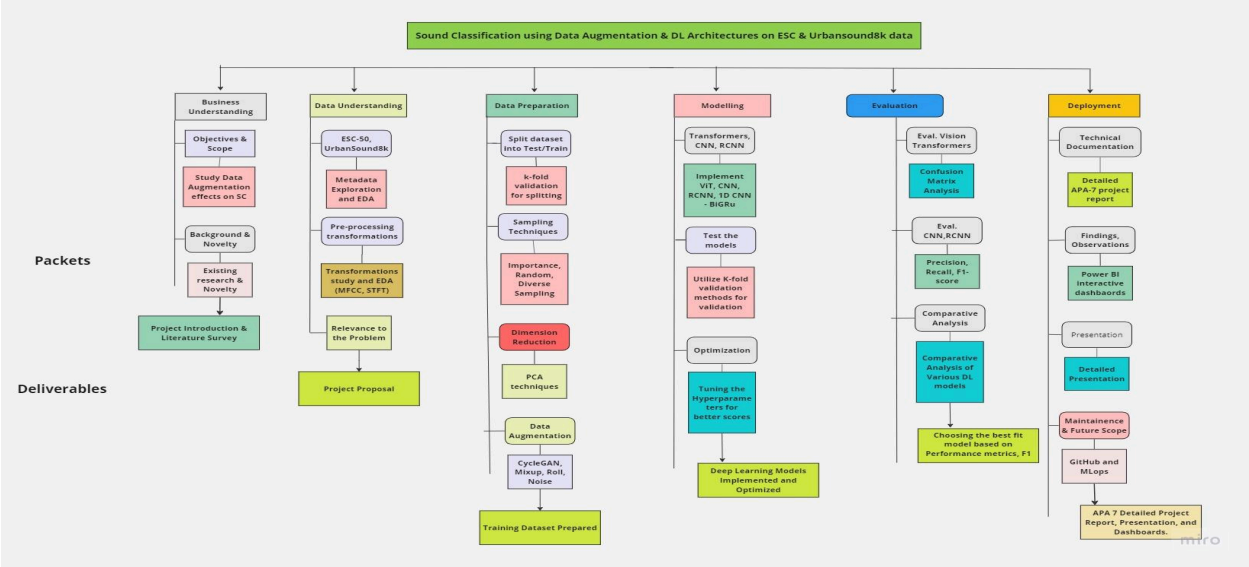
score. The focus is on not just assessing model performance but also optimizing them for better accuracy, reliability, and applicability in real-world scenarios.

Deployment. This includes setting up systems for performance tracking, documenting the models and processes, and ensuring secure data storage. Continuous monitoring and maintenance are crucial for adapting the model to new data and operational conditions, ensuring long-term effectiveness and efficiency.

Below is Figure 7 that represents the complete Work Breakdown Structure of the entire project.

Figure 7

Work Breakdown Structure of The Entire Project



The above provided Work Breakdown Structure offers a comprehensive outline of the processes and different stages involved in the urban sound audio classification research project. It serves as a foundation for creating both Gantt and PERT charts. Each phase of the project is accompanied by specified work packages and deliverables, facilitating effective project management.

2.4 Project Resource Requirements and Plan

This outlines the recommended tools, libraries, infrastructure and licenses for an urban sound classification project.

Python 3.9 will serve as the main programming language. Key libraries for audio processing, data handling, machine learning and deep learning are included from popular packages like NumPy, Pandas, Scikit-Learn, PyTorch/TensorFlow and HuggingFace. Visualization libraries like Matplotlib and Seaborn are also recommended.

Google Colab provides a cloud-based Jupyter notebook environment for model development. Apache Spark can help with distributed processing if needed. A multi-tier storage infrastructure is proposed across GCP, AWS and Google Drive for primary, secondary and tertiary backups of raw audio data totaling 7GB each. This ensures data reliability through geographic redundancy. Several free and open source tools are recommended for project management, documentation and collaboration. GitHub, JIRA, Atlassian tools and Microsoft 365 subscriptions enable version control, tracking, planning and productivity. Zoom supports online meetings while Canvas aids presentations.

In summary, the suite of tools, libraries, infrastructure and licenses outlined holistically address the varied requirements of the urban sound classification project across development, storage, collaboration and deployment needs in a cost-effective manner leveraging cloud services and open source solutions.

Software Requirements

The table recommends tools and libraries for an urban sound classification project. Python 3.9 serves as the overall programming language. Key libraries for audio processing include Librosa 0.8.1 and PyDub 0.25.1. Libraries for data processing consist of Pandas 1.3.3

and NumPy 1.21.2. Matplotlib 3.4.2 and Seaborn 0.11.2 support visualization needs. Deep learning can utilize PyTorch 2.1.0 or TensorFlow 2.14.0-rc1. HuggingFace Transformers 4.25.1 provides pre-trained models. Scikit-Learn 0.24.2 supports machine learning model evaluation. Hyperopt 0.2.5 and Optuna 2.9.1 facilitate hyperparameter tuning and optimization. scipy.optimize 1.7.3 serves general-purpose optimization. Google Colab provides a cloud-based Jupyter notebook environment without specifying a configuration. Apache Spark 3.5.0 helps with distributed processing if needed. Ensuring the latest compatible configurations for all libraries listed is advised. Overall this suite of tools addresses major requirements across the entire project pipeline from data to models to deployment. The Table 7 below outlines recommended software stack to handle audio processing, data handling, visualization, deep learning, machine learning and distributed needs for urban sound classification.

Table 7

Recommended Softwares

Software	Libraries	Configuration	Purpose
Python	-	Version 3.9	General-purpose programming language
-	Librosa	Version 0.8.1	Audio feature extraction
-	PyDub	Version 0.25.1	Audio file manipulation
-	Pandas	Version 1.3.3	Data manipulation and analysis
-	NumPy	Version 1.21.2	Numerical operations

*Not*e: The table 7 is continued in next page

Software	Libraries	Configuration	Purpose
-	Matplotlib	Version 3.4.2	Data visualization
-	Hugging Face Transformers	Version 4.25.1	Pre-trained models for ViTs
-	Scikit-Learn	Version 0.24.2	Machine learning model evaluation
-	Hyperopt or Optuna	Hyperopt Version 0.2.5 or Optuna Version 2.9.1	Hyperparameter tuning and optimization
-	scipy.optimize	Version 1.7.3	General-purpose optimization
Google Colab	-	-	Cloud-based Jupyter notebooks (alternative to local Jupyter)
-	Seaborn	Version 0.11.2	Enhanced data visualization

Hardware Requirements

The urban sound classification project requires an adequately provisioned computational and storage setup to effectively address the needs of data preprocessing, model development and evaluation. The prescribed infrastructure leverages local computing power along with cloud-based services to guarantee comprehensive capabilities catering to the varied requirements of the project.

The storage configuration allocates cloud resources from three different providers to back

up the raw and pre-processed .wav files from the urban sound classification project. GCP Standard storage in the US-West 1 region is primary storage for 7GB of audio data. AWS S3 storage acts as the secondary backup region, also with 7GB allocated to duplicate files. Additionally, 7GB of storage is distributed through a shared team drive on Google for a tertiary level of data backup across regions. This multi-tiered backup approach leveraging different cloud platforms helps ensure data reliability and geographic redundancy to support continued project work even if a single storage location encounters issues. Below is Table 8 that summarizes the hardware/software details and its intended functions.

Table 8

Summary of Hardware/Software Details With Its Intended Functions

Function	**Hardware/Software**	**Configuration**	**Purpose**
Training and testing models	Local Machine	8 CPUs, NVIDIA GeForce RTX 30 series GPUs and 32 GB RAM	Train, evaluate deep learning models efficiently using GPU acceleration
Data Storage	GCP Standard Storage (region: US-West 1)	7GB Storage	Primary Region to store raw and preprocessed audio (.wav) files as primary dataset
Data Storage	AWS S3 Standard (region: US-East 1)	7GB storage	Secondary Region to backup storage of audio files for redundancy
Data Storage	Google Shared Drive -Team (Region: OU)	7GB storage	Ternary Region for Additional backup storage of audio files

Tools and Licenses

The project requires a couple of tools for the project management and deliverables which are available with free license and subscriptions.

To begin, GitHub provides a free license for its code hosting platform. Highly popular for software development, it allows collaborative management and sharing of repositories. This supports tracking changes over time as projects evolve.Additionally, JIRA offers a free tier aimed at smaller teams. Acting as both a project tracker and for agile workflows, its user-friendly interface streamlines associated processes. Ease of use contributes to its status as a go-to solution.Furthermore, the WBS Gantt Chart tool within the Atlassian Marketplace helps with creating visual schedules. Integrated seamlessly with JIRA, it maps out project timelines including tasks, durations and dependencies. Moreover, another Atlassian Marketplace tool is Move and Organize. Available for free, it assists with breaking projects into logical work breakdown structures. This aids planning by decomposing high-level scope.In another aspect, Microsoft 365's collection of cloud apps becomes accessible with student subscription. Commonly used for documents, sheets, presentations and more, they support collaborative work. Everyday duties like documentation are simplified.Besides, Zoom presents a strong choice for video meetings. Facilitating online sessions and training, it maintains remote relationships. Simultaneous video and content sharing allow live collaboration.On another front, Canvas offers free presentation design capabilities. Insightful decks developed with it can effectively engage distant audiences. Lastly, Online Visual Paradigm provides PERT charting at no cost.

The project management tool contextually portrays schedules and dependencies, supporting analytical planning. Below is Table 9 which lists the tools and its purposes.

Table 9

List of The Tools and Its Purposes

Tool	Configuration	Purpose
GitHub	Free	To enable code collaboration and version control
JIRA	Free	To facilitate issue tracking and project management
WBS Gantt Chart	Free	To allow project scheduling visualization
Move and Organize	Free	To aid task breakdown and organization - Work Breakdown Structure
MS Office 365	Student Subscription	To provide productivity tools for documents
Zoom	Student Subscription	To support online meetings and collaboration
Canvas	Free	To design impactful project presentations
Online Visual Paradigm	Free	To design PERT Chart

Specifications, Costs, and Justification

The project will require both hardware and software resources spanning data processing, model development, evaluation and deployment as well as project management tools.

On the hardware front, a local workstation with minimum 64-bit configuration will be needed for three months at an estimated cost of $2775. Data storage options that will be explored

free of cost include GCP bucket, AWS S3 bucket and shared drive to store the ECS-50 & Urban Sound 8k dataset for three months. Software resources involve Google Colab for data preprocessing and model deployment free of cost as well as Colab Pro ($19.98/3 months) for model development and Colab Pro+ ($49.99/3 months) for model evaluation and validation stages. Project management will be handled using free tools such as JIRA for task & issue tracking, Online Visual Paradigm for project schedules, Zoom for meetings and GitHub for code repository.

The total estimated cost of resources required for the three month duration of the project is $2844.97 accounting for the local workstation and paid Google Colab plans only, as other software and tools are available free of cost. A variety of data storage and computing options will also be explored without additional expense. Below is the Table 10 that represents the cost estimations and justifications

Table 10

Cost Estimation and Justification

Function	Type of Resource	Resource	Time Duration	Cost Estimation
Local WorkStation	Hardware	Minimum 64-bit	3 Months	$2775
Data Storage	Hardware	ECS-50 & Urban Sound 8k GCP Bucket	3 Months	Free
Data Storage	Hardware	ECS-50 & Urban Sound 8k AWS S3 Bucket	3 Months	Free

Note. The Table 10 is continued on the next page.

Function	Type of Resource	Resource	Time Duration	Cost Estimation
Data Storage	Hardware	ECS-50 & Urban Sound 8k Shared Drive	3 Months	Free
Data PreProcessing	Software	Google Colab	3 Months	Free
Model Development	Software	Google Colab pro	3 Months	$19.98
Model Evaluation & Validation	Software	Google Colab Pro+	3 Months	$49.99
Model Deployment	Software	Google Colab	3 Months	Free
Project Management	Tool	JIRA	3 Months	Free
Project Organization & Schedule	Tool	Online Visual Paradigm	3 Months	Free
Project Meetings	Tool	Zoom	3 Months	Free
Git Repository	Tool	GitHub	3 Months	Free
Total Cost				$2,844.97

2.5 Project Schedule

Gantt Chart

A Gantt chart is a popular visualization tool in project management, used to outline a project's timeline, tasks, and their interdependencies. It typically displays a horizontal axis marking the project's duration and a vertical axis listing the tasks. Each task appears as a

horizontal bar spanning its required completion time, and arrows between these bars show the dependencies between tasks, indicating the order of completion. Gantt charts are extensively used in various fields such as construction, software development, and event planning, providing project managers with a clear overview of project timelines, task relationships, and progress tracking. In essence, a Gantt chart is a crucial visual tool in project management that helps in efficiently tracking project timelines and managing tasks and resources.

For the ongoing research project, which is structured in two-week sprints from the Business Understanding phase through to Deployment, weekends are marked as non-working days, following standard organizational work schedules. The project team has decided to work through the spring break, with continuous task allocation throughout the project's duration. This ensures that all team members are consistently engaged with tasks, except when waiting on dependencies. Subtasks are allocated for periods of 1-2 days, depending on their complexity. The team's approach to task assignment and management is systematic and professional, employing effective methodologies and tools for successful project execution. Each project phase is meticulously planned, with detailed breakdowns of the efforts required.

Business Understanding. This epic includes all the basic understanding of the problem statement, domain, Existing research and Novelty. The stories are created to group multiple relevant tasks under them, the tasks are highly relatable to the current project problem statement, and have been assigned to individuals. The second round of Business Understanding started on 26th September, 2023 and 3rd October, 2023 - spanning only one week due to the existing research the team has already done as part of the initial proposed problem statement.

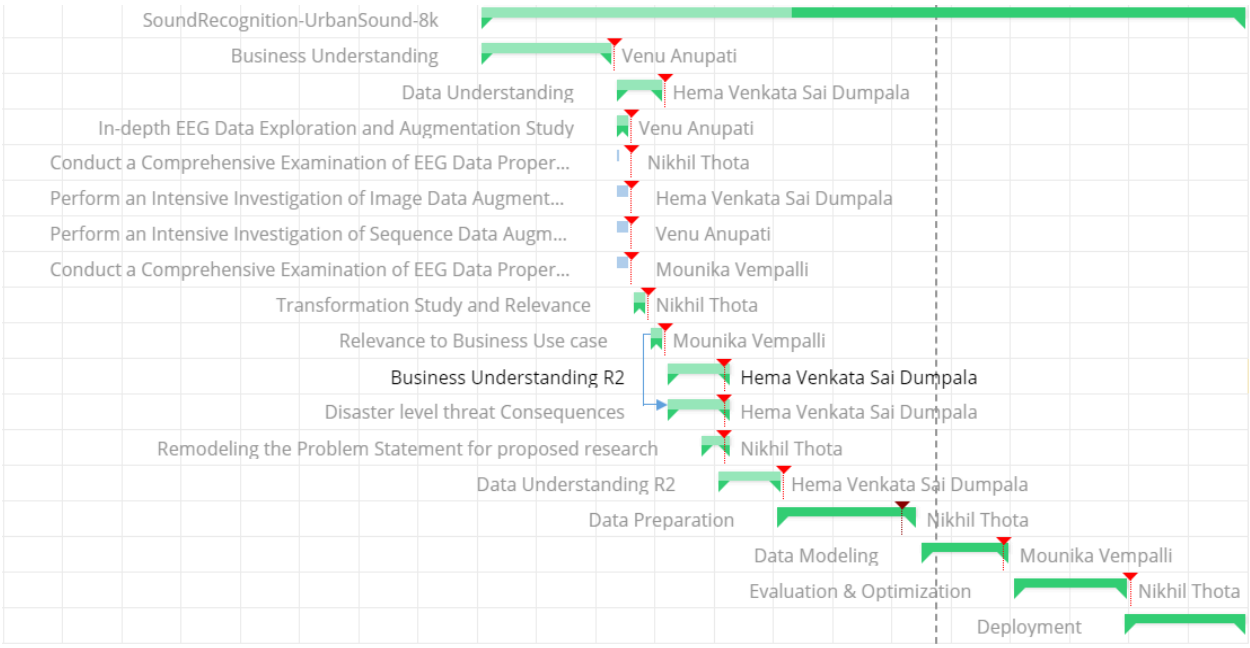
Note. It's mentioned as Business Understanding R2, as the initial problem statement which was worked upon did not materialize into the Research project due to the Data access

issues. The dataset that was required was Human related EEG data which posed issues.

Below is Figure 8 that represents the Tasks, Timelines, and Progress in Business Understanding are depicted in a Gantt chart.

Figure 8

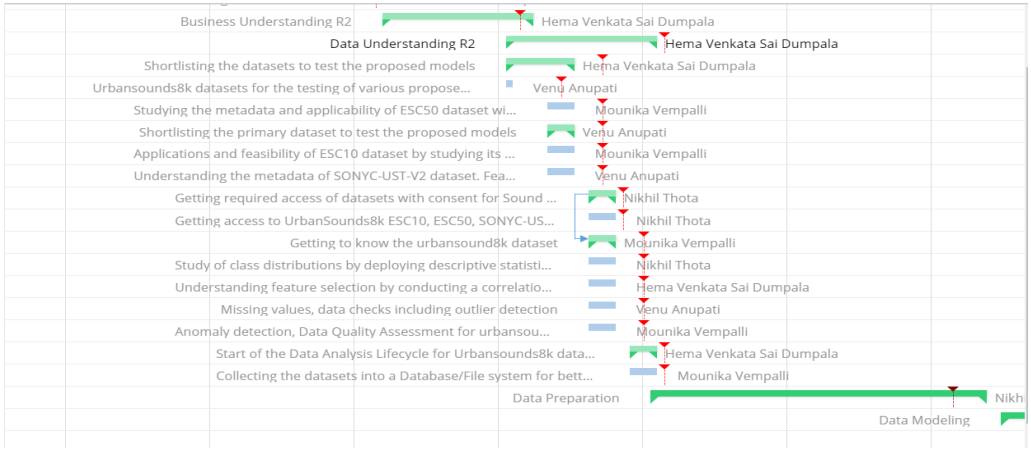
Representation of The Tasks, Timelines, and Progress in Business Understanding



Data Understanding. This phase includes collection of the audio datasets, understanding the metadata, including Quality checks for the metadata of both the Urbansound8k and ESC-50 datasets. This is crucial because these datasets are sourced by external sources, and it's really important to understand the approaches the curators have used to curate the dataset. This phase also includes Exploratory data analysis of the Metadata, as well as EDA of Mel scaled Spectrograms. The objective of the EDA is to understand any underlying outliers, and unnecessary trends in the data. Below is Figure 9 that represents the Tasks, timelines, and progress in Data Understanding are depicted in a Gantt chart.

Figure 9

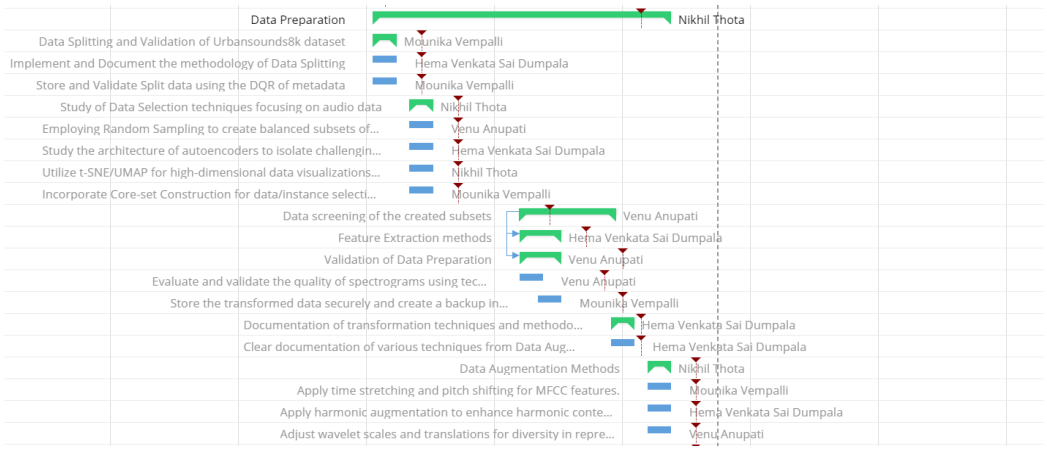
Representation of The Tasks, Timelines, and Progress in Data Understanding



Data Preparation. It is the key phase of this project as it includes various data processing algorithms like various kinds of sampling, Dimensional reduction, and Data Augmentation techniques to prepare the data for the next phases, i.e., Modeling. These techniques have been chosen carefully by keeping their compatibility with the Audio datasets, and the proposed Deep learning architectures. The dependencies are shown in the below Gantt chart Figure 10.

Figure 10

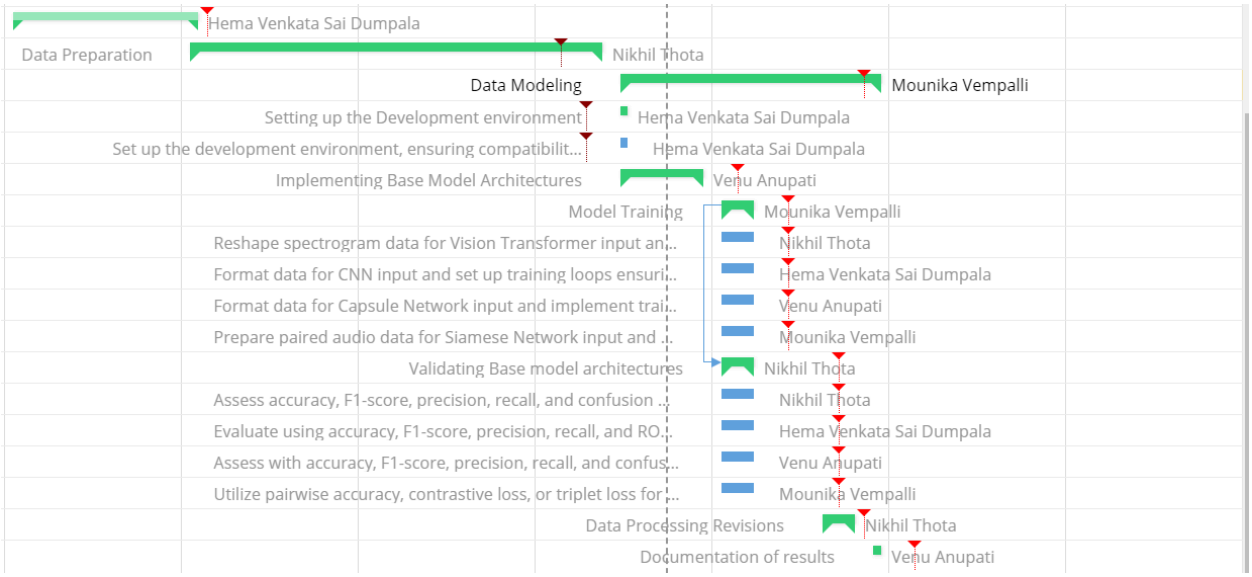
Tasks, Timelines, and Progress in Data Preparation



Modeling. It is definitely of highest importance, as the team aims to implement Advanced deep learning models like Transformers, CNN, CNN-LSTM, and CNN-BiGRU. Various stories and tasks like Implementing the baseline DL models, evaluating them using K-fold validation, Training using the DL models, and testing using the standard metrics, this is the flow of the Modeling phase, which is shown below in Figure 11.

Figure 11

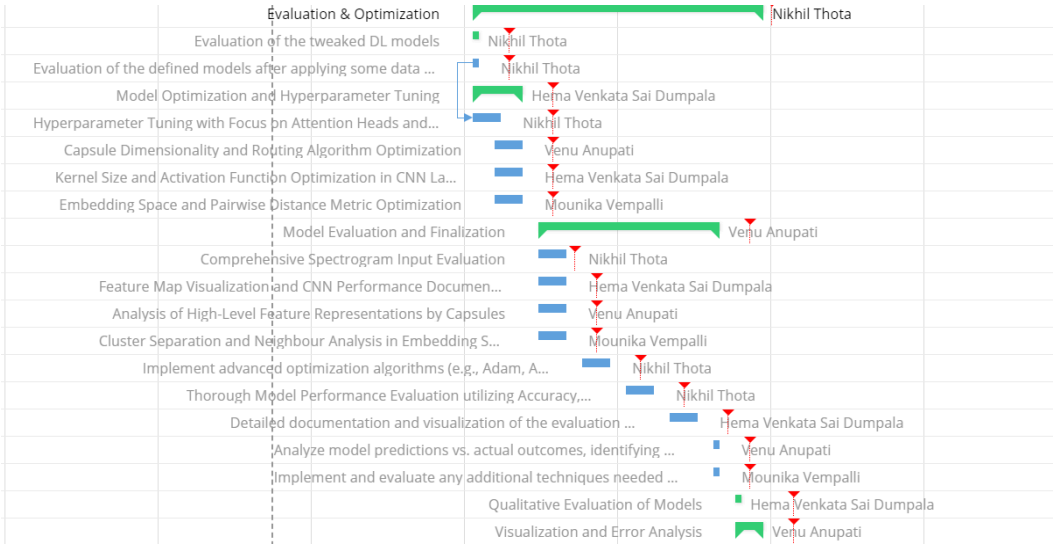
Tasks, Timelines, and Progress in Modeling



Evaluation. It is considered to be another important phase of this project, as this describes how good the models are performing based on the various Performance metrics calculated using Confusion matrix, ROC AUC, F1-score, etc. The tasks and stories mentioned as part of the Evaluation phase, along with the respective dependencies provide further information on how this phase is carried out. It also includes Optimization stages like Hyperparameter Tuning, to further improve the performance metrics of the respective models. This is represented in the below Figure 12.

Figure 12

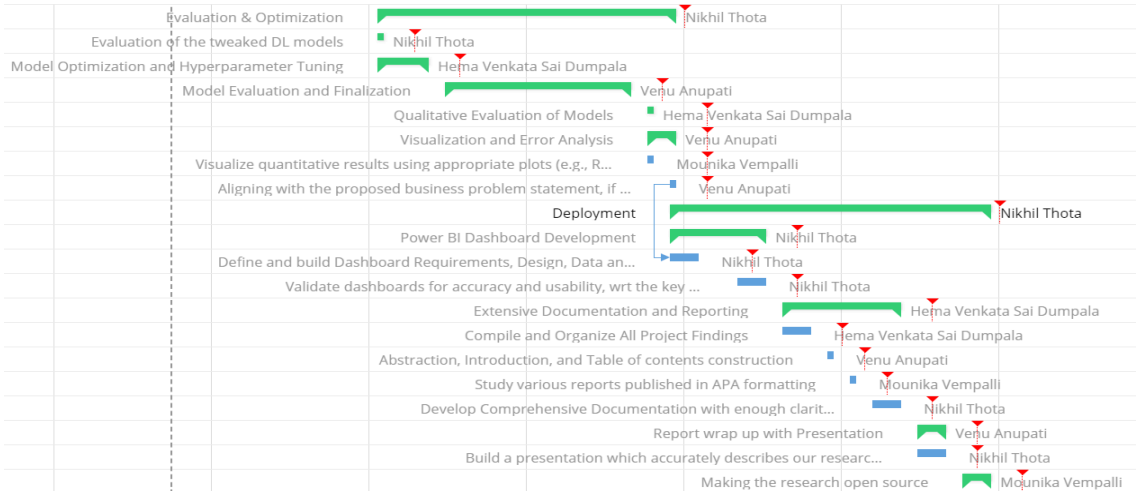
Tasks, Timelines, and Progress in Evaluation



Deployment. As mentioned the only deliverables are Extensive APA formatted documentation, Power BI dashboards to establish the findings, observations of the research study. Along with a powerpoint presentation to communicate the key findings, methodologies from this research project. This is represented in the below Figure 13.

Figure 13

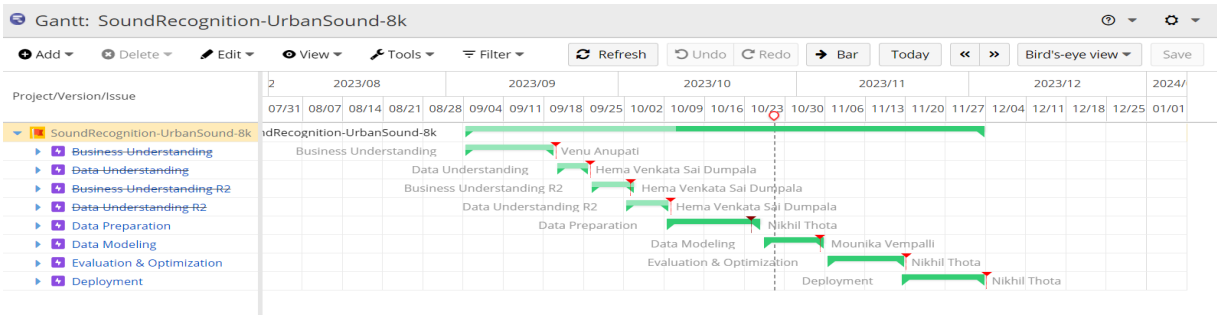
Tasks, Timelines, and Progress in Deployment



The below Figure 14 is the bird eye view of the Gantt chart, the team has come up with smooth progress over the course of this research project.

Figure 14

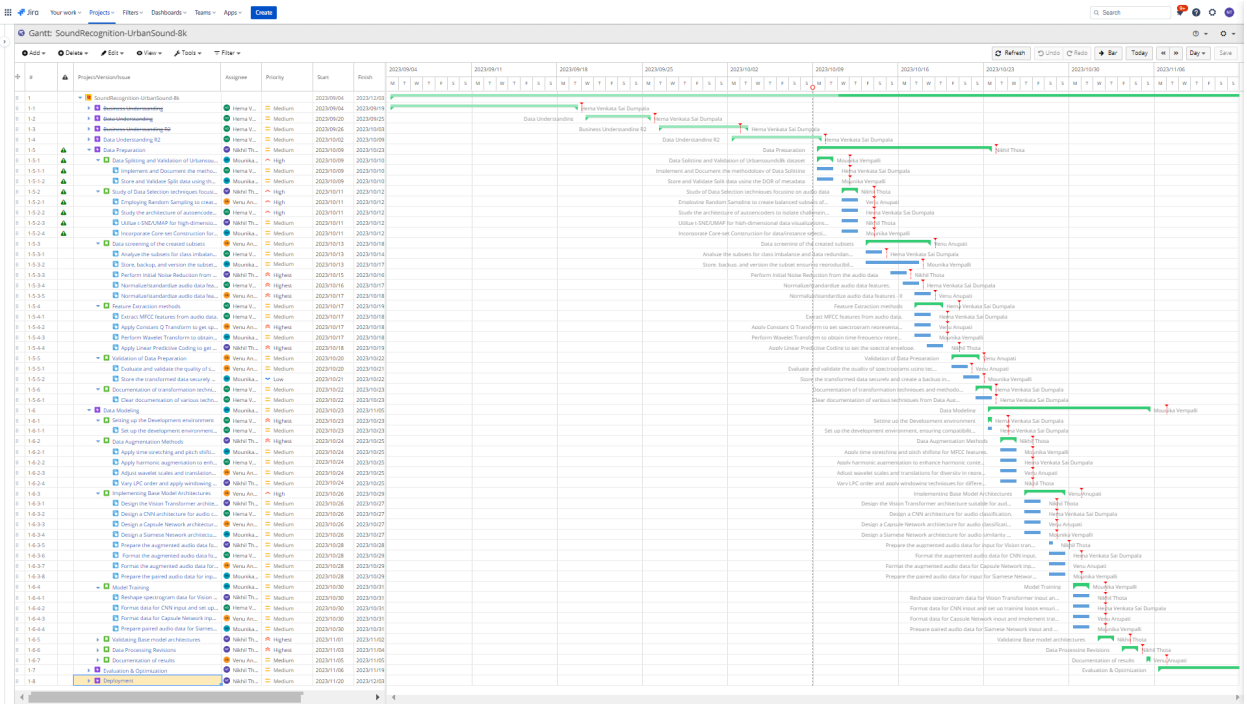
Bird Eye View of Gantt Chart



The below Figure 15 shows all the tasks, stories and epics the team has created, assigned evenly across all the team members, making sure that the contributions are evenly distributed.

Figure 15

Tasks, Stories and Epics

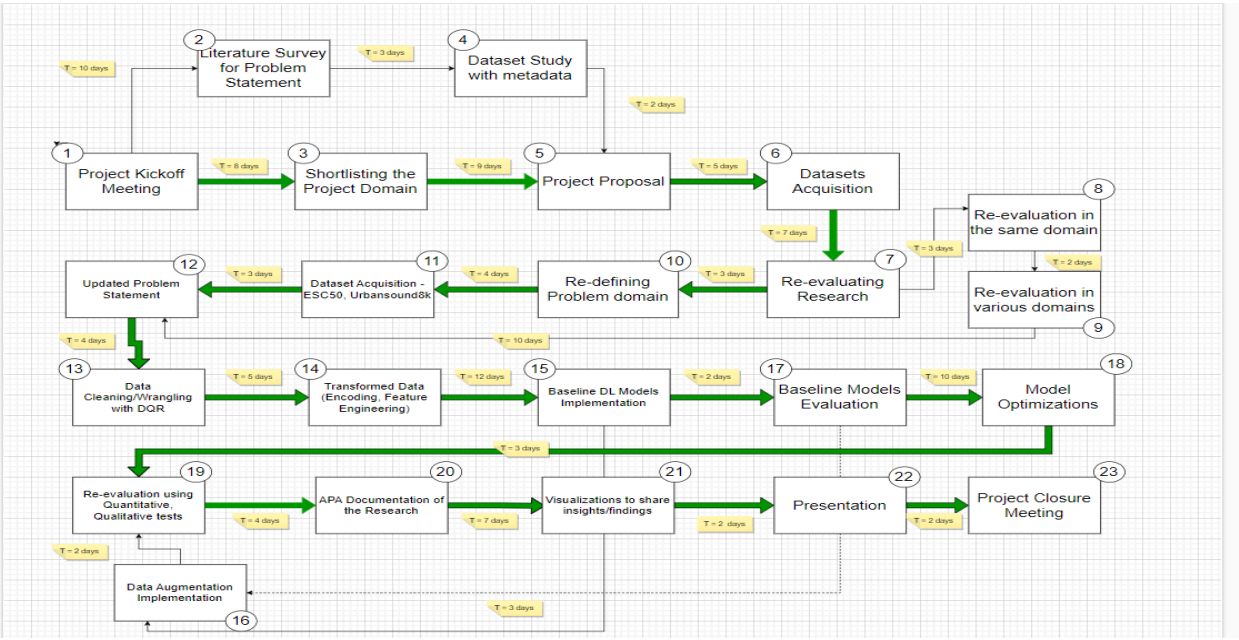


PERT Chart

A PERT(Program Evaluation Review Technique) chart is a planning and scheduling instrument for managing a project's duties and activities. Unlike Gantt charts that display the timeline of a project, PERT charts concentrate on the interdependencies of tasks and the critical path. The critical path determines the longest possible path, meaning every milestone in this path is key, and is directly dependent on how quickly the project is completed. In PERT charts, nodes symbolize the tasks/milestones, and arrows indicate the relationships between these tasks. Initially, the PERT chart has been designed using tasks, and respective dependencies between them, with the duration for each task, thereby deciding the Critical path. Then the team aggregated it to Milestone level for better understanding and granularity. The below Figure 16 shows the PERT Chart

Figure 16

PERT Chart (Milestone Level)



These milestones are selected in such a way that all the planned phases are touch based,

and key deliverables/outcomes of these individual phases link up, and lead to Project Closure. The above PERT chart includes all the key milestones that have been identified as part of this project, aiming to build DL models for Sound classification. The project timeline spans over three months, from September 4th, 2023 to December 3rd, 2023. Each milestone is depicted by the node, and the chronological order of these milestones is mentioned in the form of a number in the right corners of each node. The duration and the dependencies are mentioned using the arrows between these milestones, indicating the flow of the Project. The chart is useful for illustrating how tasks are interconnected, and the critical path represents the sequence of essential tasks that crucially influence the project's completion date.

From the above Figure 16 PERT chart, it can be seen that there are various Paths to complete the project. Calculations are below.

1 > 2 > 4 > 5 > 6 > 7 > 10 > 11 > 12 > 13 > 14 > 15 > 17 > 18 > 19 > 20 > 21 > 22 > 23

The total duration for the above Path comes up to = 86 days (a)

Other path is as below.

1 > 2 > 4 > 5 > 6 > 7 > 8 > 9 > > 12 > 13 > 14 > 15 > 16 > 19 > 20 > 21 > 22 > 23

The total duration for the above Path comes up to = 83 days (b)

Another path is as below.

1 > 3 > 5 > 6 > 7 > 10 > 11 > 12 > 13 > 14 > 15 > 17 > 18 > 19 > 20 > 21 > 22 > 23

The total duration for the above Path comes up to = 88 days (c)

With the above calculations, and the way the Critical path is defined for a project it can be inferred that (c) is the critical path for the research Project and for the defined PERT chart, It also makes sense due to the numerous processes this problem statement is pursuing. This Critical path is highlighted in **GREEN** arrows in the above PERT chart.

The problem statement starts with Project kickoff meeting, surfing across Business Understanding, Data Understanding, Business Understanding R2 (Redo it due to accessibility issues with the first problem statement), Data Understanding R2, followed by Modeling, Evaluation and Optimization, and Deployment. These phases may not be indicated directly, but the milestones are linked to the mentioned phases. Relationships and duration between these milestones are key, as they decide the Critical path and its duration.

3 Data Engineering

This chapter focuses on various Data engineering and management methodologies the project has employed to classify the urban audio data through ESC-50 & Urbansound8k.

3.1 Data Process

To start with the data processes, Initially the ESC-50 and Urbansound8k datasets are downloaded from the respective sources. The first step was to explore the audio files through iterative audio exploration by following the respective metadata files.

The raw datasets were initially collected by downloading them from their sources, like the ESC-50 and UrbanSound8K from freesound.org. This collection involved recording sounds and converting them into a digital format through Analog-to-Digital Converters (ADCs), entailing steps such as sampling and quantization. The ESC-50 dataset, in particular, was carefully curated to facilitate the classification of urban sounds.

Preprocessing was the next critical step, where the audio files were standardized to ensure uniform duration through padding, sampled at a consistent rate of 44.1 kHz in mono, and subjected to noise reduction and smoothing to improve quality. Specifically, the UrbanSound8K files were padded to four seconds, and the ESC-50 files were cleaned, using noise reduction techniques and cleaning the pops and clicks sounds.

The transformation of the audio files into formats suitable for modeling, such as spectrograms, was performed next. Techniques utilized included Mel-Scaled spectrograms and Mel-Frequency Cepstral Coefficients (MFCCs), preparing the data for compatibility with proposed deep learning architectures like ViT, CNN, RNN, CNN-BiGRU. To enhance the diversity of the data and bolster the model's ability to generalize, data augmentation was applied. SpecAugment, including time warping and frequency masking, was employed to the spectrogram images, enriching the training samples without altering the fundamental characteristics of the sounds.

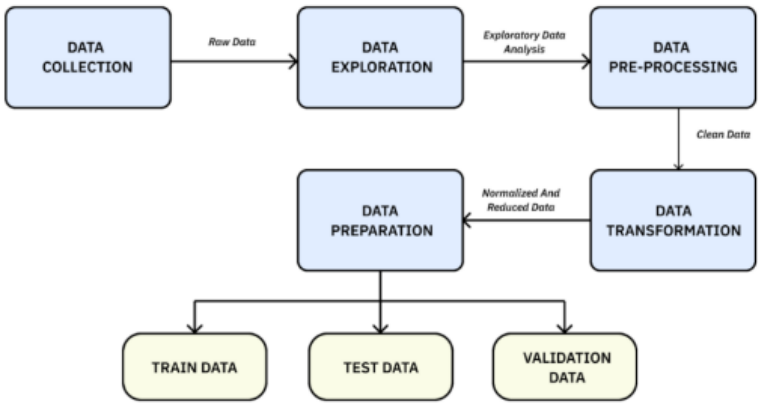
Dimensionality reduction was also conducted using PCA, t-SNE to focus on the informative features and discard the redundant ones, thus streamlining the feature set for the defined deep learning models. K-fold cross-validation played a pivotal role in data preparation, ensuring the robustness and generalization of the models. This technique divided the data into several subsets, with the model being trained and validated on different subsets in each iteration to prevent overfitting. For validation and testing, the data was divided into distinct training, validation, and test sets as delineated by the metadata's fold information. This division was crucial to maintain integrity in model evaluation and to avoid data leakage, which could artificially inflate performance metrics. Finally, the process involved comprehensive Exploratory Data Analysis (EDA), where data statistics and visualizations like waveforms, histograms of spectral centroid values, and zero-crossing rates were used to understand the characteristics of the datasets. These analyses ensured the data transformations preserved the original datasets' distributions and were representative for urban sound classification tasks. Through this meticulous data engineering process, the datasets were optimally prepared to develop accurate and generalizable machine learning models for classifying urban sounds. The entire overview of

data processing that has been employed in this research is represented in Figure 13 below.

All the data files that have been generated for the Data Engineering phase are backed up in Drive, GCP and S3 in respective buckets, these files will act as backup. Audio files are stored in BLOB format, and the respective metadata files are stored in form of csv's.

Figure 13

Overview of Data Process



Below Table 11 shows the entire data process plan with each individual task with assigned team members.

Table 11

Data Process Plan

Exploratory Data Analysis Task	Assigned Team Member
Data Collection	Nikhil Thota, Venu Anupati
Initial EDA and understanding	Nikhil Thota, Mounika
Data Pre-processing (Noise, outliers)	Nikhil Thota, Hema Sai

Note. The Table 11 is continued on the next page.

Exploratory Data Analysis Task	Assigned Team Member
Data Transformation (Augmentation, features)	Nikhil Thota, Hema Sai, Venu
Input features Preparation for DL models	Nikhil Thota, Mounika

3.2 Data Collection

This research based on Urban sound classification is validated on ESC-50, and Urbansound8k datasets. These datasets are sourced from freesound.org, by Piczak (2015) and Justin Salamon et al. (2016) respectively. The details with respect to these datasets have been shared in the previous chapter in detail. The primary focus here is to discuss the various data engineering methods that were adapted to this research project. This section, Data Collection will describe and focus on various Audio collection mechanisms, and the exact procedures the respective dataset curators have followed to create these datasets. It also aims to brief the general audio collection methods.

Digital audio files are created through a multi-step process that transforms sound into a digital format. Initially, sound is recorded using devices like microphones, capturing the audio as analog signals. These signals are then converted into digital form using an Analog-to-Digital Converter (ADC) by employing Sampling, and Quantization methods. This conversion involves sampling the sound at regular intervals, known as the sampling rate, and measuring the intensity of the sound wave at each point, referred to as the bit depth. The resulting digital data is often large, so it is typically compressed to manage file sizes. Compression can be either lossless, where no audio quality is lost, or lossy, where some quality is compromised for smaller file sizes. Finally, the compressed audio is stored in digital formats like MP3 or WAV, ready for playback. The audio files of freesound.org must have followed a similar process to come up with respective

digital audio formats.

For ESC-50, which was sourced manually from freesound.org by author Piczak (2015) to improve the dataset accessibility for Urban audio use cases in Machine learning and Deep learning. This dataset has been quoted by numerous researchers and published papers majorly dealing with Audio classification, Acoustic sound improvement, and in various other Urban audio settings. This dataset like mentioned is curated manually by picking out various categories of audio files from freesound.org and is arranged into five folds, so that the fragments of the original audio file are contained in the same fold. The idea behind this is to reduce the overfitting and increase the generalization and accuracy of the respective classifiers. The metadata file of ESC-50 has the respective mapping of that particular mapping file to its parent audio file from freesound.org. Each audio file is identified using the filename column, which is tagged against a particular class of audio. ESC-50 has 50 distinct classes broadly categorized into Animal, Nature and Water sounds, Human, non-speech sounds, Interior/Domestic sounds, and Exterior/Urban sounds. There are 40 audio files for each of the 50 classes, which sums up to 2000 audio files for this dataset, there are an additional 250,000 audio files which have been curated as another dataset for unsupervised learning methods.

The primary focus for this research would only be on ESC-50 as the problem statement only includes Urban audio classification using the labeled data and various data engineering methods which will be discussed in further detail in the next sections. The metadata of the ESC-50 dataset is projected using the data collection forms below, this dataset is briefly described and understood using the metadata file which was also shared as part of the ESC-50 dataset. Below forms also show the features that are extracted from the respective audio files, and used as the input features to the defined deep learning architectures. It is represented in the

below Figure 14. The metadata is a very clean dataset for both the Urbansound8k and ESC-50 datasets. Various EDA on the metadata, and the audio file distribution have been shared as part of the data analytics, data statistics sections of this report in the end.

Figure 14

Data Collection Plan for Environmental Sound Classification (ESC-50)

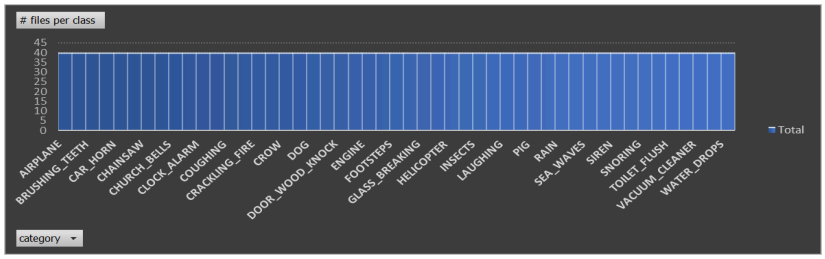
Data Collection Plan

Project number:	Group 6	Project title:	Engineering techniques and Deep learning Architeotures on ESC-50 and Urbansound8k	Project leader:	Nikhil Thota			Date:	11/03/2023
Description of the data collection									
	Environmental Sound Classification (ESC-50) dataset was curated by the authors to help the research community to boost the research in the field of Urban audio classification. ESC-50 has been employed in various existing research studies, and helped establish bench marks in the field of audio classification. This dataset has 50 audio classes defined, with 40 .wav files per class, totalling to 2000 .wav audio files of consistent length of 5 seconds.								
What will be done with the data once it has been collected?									
	yes	Identify how well the process is meeting customer needs	yes	Analyze a problem, or identify the causes of variation					
	yes	Obtain exploratory view of the process	yes	Test a hypothesis about the process output					
	yes	Evaluate the measurement system	yes	Test a hypothesis about the effects of one or more inputs					
	yes	Check the stability of the process (is it in control?)	yes	**Control a process input or monitor a process output**					
	yes	Conduct a capability study		**Other reason_**					
Key Variables – A summary of the chosen input variables (Y's) and/or output variables (X's)									
			1	**2**	**3**	**4**	**5**	**6**	
	Variable title		file_name	fold	Mel-scaled spectrograms	MFCC Co-efficients	Waveforms	class_name	
	Input (X) or output (Y) variable?		X	X	X	X	X	Y	
	Unit of measurement		N/A	N/A	N/A	N/A	N/A	N/A	
	Data type		Object	Integer	Image (Object)	Array	Image (Object)	string	
What?	Collection method		Manual Scraping	Manual Scraping	Automated	Automated	Automated	Manual Scraping	
	Gauge/Instrument		N/A	N/A	N/A	N/A	N/A	N/A	
	Location		N/A	N/A	N/A	N/A	N/A	N/A	
	Gauge calibrated?		N/A	N/A	N/A	N/A	N/A	N/A	
	Measurement system checked?		N/A	N/A	N/A	N/A	N/A	N/A	
	Precision (R&R) adequate?		N/A	N/A	N/A	N/A	N/A	N/A	
MSA	Accuracy adequate?		N/A	N/A	N/A	N/A	N/A	N/A	
	Historical data exist?		N/A	N/A	N/A	N/A	N/A	N/A	
	Source of historical data		N/A	N/A	N/A	N/A	N/A	N/A	
	Historical data representative/reliable?		N/A	N/A	N/A	N/A	N/A	N/A	
	Mean and Standar deviation		N/A	N/A	N/A	N/A	N/A	N/A	
	Upper and lower specification limit		N/A	N/A	N/A	N/A	N/A	N/A	
Historical data	Target		N/A	N/A	N/A	N/A	N/A	N/A	
	Minimum sample size (MSS)		N/A	N/A	N/A	N/A	N/A	N/A	
	Sampling frequency		44.1khz	N/A	N/A	N/A	N/A	N/A	
	Sub-grouping needed?		N/A	N/A	N/A	N/A	N/A	N/A	
	Sub-group size		N/A	N/A	N/A	N/A	N/A	N/A	
Sampling	Stratification needed?		N/A	N/A	N/A	N/A	N/A	N/A	
	Data collector		Nikhil Thota	Nikhil Thota	Nikhil Thota	Nikhil Thota	Nikhil Thota	Nikhil Thota	
Who?	Operational definition exist? Data collector trained? Resources available for data collector?		The only pre-processing specs that are identified for this dataset are pre-sampling at 44.1khz, mono channel, and padded silence at the end to make the duration consistent. The datasets have been captured from freesound.org, so the collection methods are specific to the collection methods of the respective audio files from freesound.org						
	Start date		1-Oct	1-Oct	1-Oct	1-Oct	1-Oct	1-Oct	1-Oct
	Due date		7-Oct	7-Oct	7-Oct	7-Oct	7-Oct	7-Oct	7-Oct
When?	Duration (in days)		7	7	7	7	7	7	

Above Figure 14 describes the features extracted and will be employed in this project, which are mel-scaled spectrograms. Below Figure 15 shows the visualization of .wav files per class from the raw ESC-50 dataset, it's symmetric.

Figure 15

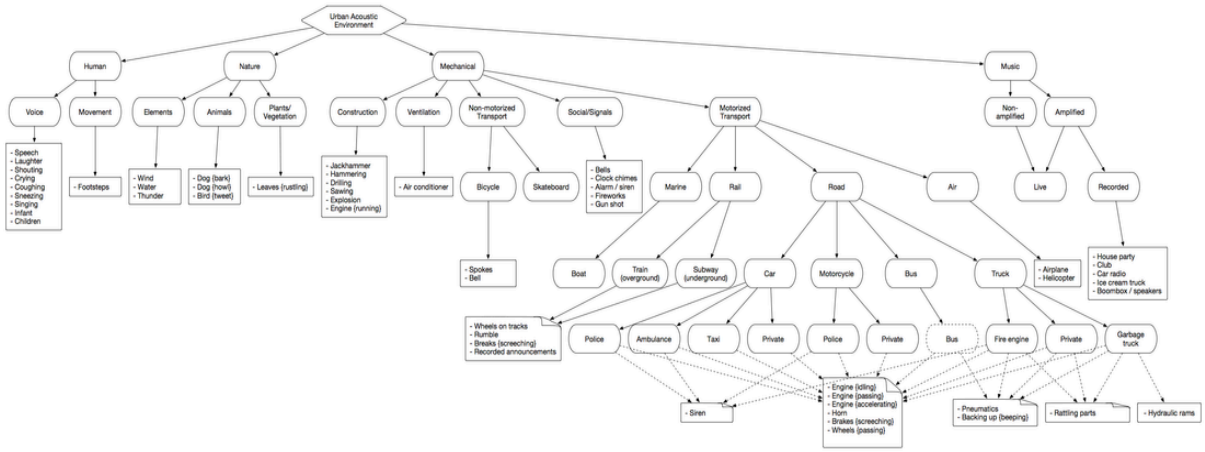
Overview of ESC-50 Files Per Class



Urbansound8k dataset is used as a secondary dataset over the course of this research. This dataset was curated by Justin Salamon et al. (2016) before ESC-50, this dataset was curated by using API's, so the audio files are directly extracted from freesound.org, which means the sampling rate, bit rate, duration and other parameters are going to vary for each audio file in this dataset, as they are directly curated from freesound.org. Urbansound8k has a total of 10 classes divided across 10 folds, for K-cross validation purposes. This dataset has been collected by keeping the Urban taxonomy in context. The Urban Sound taxonomy classification is shown in Figure 16 with each sound classification.

Figure 16

Hierarchical Representation of Urban Sound 8k



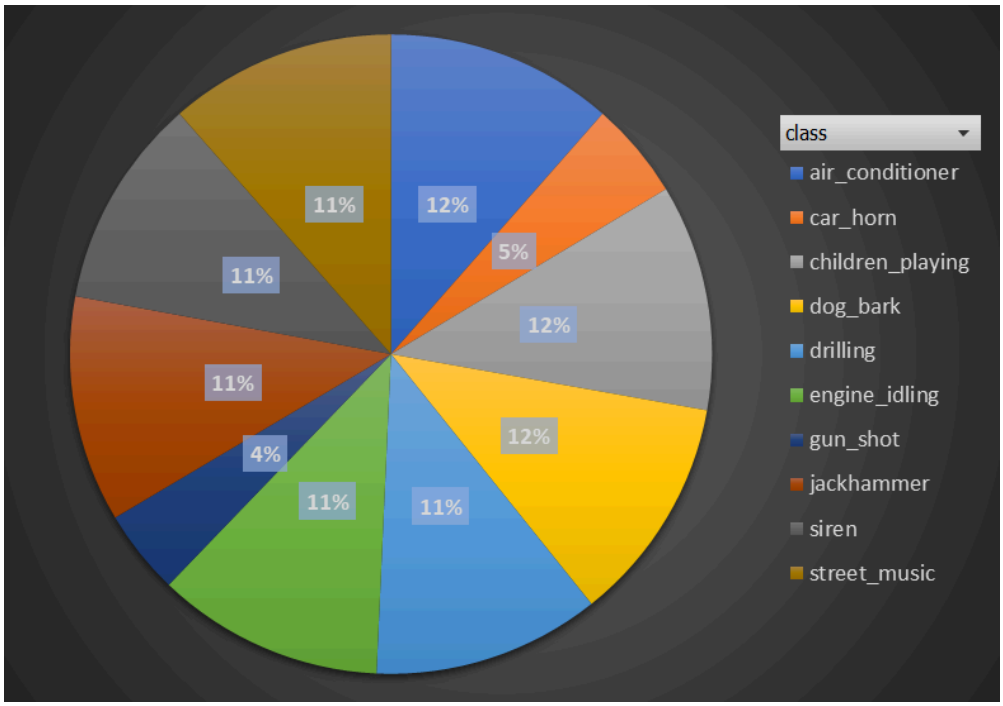
Note. Referred from the work of Salamon et al. (2014)

The authors consider this taxonomy hierarchy, and the classes of sounds are possibly infinite for urban settings, so the idea is to keep the taxonomy dynamic, and not fixed. Urbansound8k has a total of 8732 .wav audio files of varied duration, <= 4 seconds, and distributed across 10 classes, as per the above taxonomy. This dataset is not symmetric in terms of distribution of the .wav audio files across 10 classes. The distribution of the audio files is

below, there will be significant pre-processing steps for the UrbanSound8k dataset. The pre-processing for Urbansound8k is important to make the audio data consistent. The below Figure 17 shows the availability of each class sounds in the entire dataset.

Figure 17

Percentage of Each Class Files in Urban Sound 8k



It can be seen that there's a major class imbalance in car_horn, and gun_shot. While this data is being validated upon, the team needs to make sure to employ some kind of sampling/normalization techniques for UrbanSound8k to make sure to reduce the effects of the class imbalance here.

The data collection plan remains pretty much the same for the UrbanSound8k dataset, the dataset is available under public commons license, and easily accessible after filling out a consent form. The number of classes are however reduced but the quality and the characteristics of the data are very much real world like.

The data collection plan document with recorded features for urbansound8k is shared in the Figure 18 below.

Figure 18

Data Collection Plan For UrbanSound8K Dataset

Data Collection Plan

Project number:	Group 6	Project title:	Engineering techniques and Deep learning Architectures on ESC-50 and Urbansound8k	Project leader:	Nikhil Thota			Date:	11/03/2023
Description of the data collection									
	Environmental Sound Classification (ESC-50) dataset was curated by the authors to help the research community to boost the research in the field of Urban audio classification. ESC-50 has been employed in various existing research studies, and helped establish bench marks in the field of audio classification. This dataset has 50 audio classes defined, with 40 .wav files per class, totaling to 2000 .wav audio files of consistent length of 5 seconds.								
What will be done with the data once it has been collected?									
	yes	Identify how well the process is meeting customer needs	yes	Analyze a problem, or identify the causes of variation					
	yes	Obtain exploratory view of the process	yes	Test a hypothesis about the process output					
	yes	Evaluate the measurement system	yes	Test a hypothesis about the effects of one or more inputs					
	yes	Check the stability of the process (is it in control?)	yes	**Control a process input or monitor a process output**					
	yes	Conduct a capability study		**Other reason_**					

Key Variables – A summary of the chosen input variables (Y's) and/or output variables (X's)								
		1	**2**	**3**	**4**	**5**	**6**	
	Variable title	file_name	fold	Mel-scaled spectrograms	MFCC Co-efficients	Waveforms	class_name	
	Input (X) or output (Y) variable?	X	X	X	X	X	Y	
	Unit of measurement	N/A	N/A	N/A	N/A	N/A	N/A	
	Data type	Object	Integer	Image (Object)	Array	Image (Object)	string	
What?	Collection method	Manual Scraping	Manual Scraping	Automated	Automated	Automated	Manual Scraping	
	Gauge/instrument	N/A	N/A	N/A	N/A	N/A	N/A	
	Location	N/A	N/A	N/A	N/A	N/A	N/A	
	Gauge calibrated?	N/A	N/A	N/A	N/A	N/A	N/A	
	Measurement system checked?	N/A	N/A	N/A	N/A	N/A	N/A	
	Precision (R&R) adequate?	N/A	N/A	N/A	N/A	N/A	N/A	
MSA	Accuracy adequate?	N/A	N/A	N/A	N/A	N/A	N/A	
	Historical data exist?	N/A	N/A	N/A	N/A	N/A	N/A	
	Source of historical data	N/A	N/A	N/A	N/A	N/A	N/A	
	Historical data representative/reliable?	N/A	N/A	N/A	N/A	N/A	N/A	
	Mean and Standar deviation	N/A	N/A	N/A	N/A	N/A	N/A	
	Upper and lower specification limit	N/A	N/A	N/A	N/A	N/A	N/A	
Historical data	Target	N/A	N/A	N/A	N/A	N/A	N/A	
	Minimum sample size (MSS)	N/A	N/A	N/A	N/A	N/A	N/A	
	Sampling frequency	44.1khz	N/A	N/A	N/A	N/A	N/A	
	Sub-grouping needed?	N/A	N/A	N/A	N/A	N/A	N/A	
	Sub-group size	N/A	N/A	N/A	N/A	N/A	N/A	
Sampling	Stratification needed?	N/A	N/A	N/A	N/A	N/A	N/A	
	Data collector	Nikhil Thota	Nikhil Thota	Nikhil Thota	Nikhil Thota	Nikhil Thota	Nikhil Thota	
Who?	Operational definition exist? Data collector trained? Resources available for data collector?	The only pre-processing specs that are identified for this dataset are pre-sampling at 44.1khz, mono channel, and padded silence at the end to make the duration consistent. The datasets have been captured from freesound.org, so the collection methods are specific to the collection methods of the respective audio files from freesound.org						
	Start date	1-Oct	1-Oct	1-Oct	1-Oct	1-Oct	1-Oct	1-Oct
	Due date	7-Oct	7-Oct	7-Oct	7-Oct	7-Oct	7-Oct	7-Oct
When?	Duration (in days)	7	7	7	7	7	7	

The key differences between ESC-50 and Urbansound8k dataset are below.

1. The duration of .wav files in ESC-50 is 5 seconds, and consistent across all the audio files in ESC-50. For urbansound8k, the duration varies, and is <= 4 seconds.
2. The sampling rate and the bit rate is varied in urbansound8k, whereas ESC-50 is sampled at 44.1 kHz in the mono channel.
3. As ESC-50 is already pre-sampled, and uses a mono channel for further processing, this makes the ESC-50 much more precise, and suited for accuracy improvement problems.

In the next sections, the data pre-processing, and other data engineering steps will be detailed

which will result in improvement of audio quality for the Urbansound8k. Once both the datasets are preprocessed, the next steps would be to apply transformations, Augmentations and feature engineering phases. Some raw .wav files of both the datasets are presented below, they are transformed into spectrograms, and waveforms, for better visualization purposes. The below Figure 19 and Figure 20 shows some raw.wav file representations of both datasets respectively.

Figure 19

Sample ESC-50 Raw Audio .wav Files with Audio Controls

Figure 20

Sample UrbanSound8k Raw Audio .wav Files with Audio Controls

The above are the samples from the respective datasets, it can be observed that the duration varies in Urbansound8k, and not in ESC-50.

3.3 Data Pre-processing

Data preprocessing is one of the most important steps in any data lifecycle. In this step, data collected is undergone a set of pre-processing steps like Data cleaning, formatting, Noise removal, padding silence if the duration is inconsistent, Sampling the audio files, Using mono channel, and employing compression techniques. The resulting processed files are then used as input data for the next phase, that is Data transformation.

As ESC-50 is mostly pre-processed, the focus of this section would be on briefly understanding the pre-processing steps that are employed for Urbansound8k through the course of this project, eventually it will also brief about pre-processing the ESC-50 dataset, and show the respective data pre-processing results on ESC-50 dataset as well. For Urbansound8k, below are the steps that are being employed as part of the data pre-processing.

1. Padding the audio files with silence if necessary, so that all the audio files are converted into 4 secs duration.
2. Sampling the audio files at 44.1 kHz, and mono channel tuning, bit rate of 192.
3. Noise reduction (1.5 secs), and Clicks & Pop sound removal techniques to reduce the unnecessary noise, and improve the quality utilizing noisereduce python library.
4. Moving average Smoothing with a window size of 5.

These steps are applied one by one. The below screenshot summarizes step 1, and step 2 for a set of samples from Urbansound8k dataset, below step shows that the duration is made consistent as 4 seconds for all the audio files of Urbansound8k dataset. Data Preprocessing results are provided in the Figure 21 below.

Figure 21

Data Pre-Processing Execution Results Summary of Urban Sound 8k Dataset Sample

```
File: 101415-3-0-2.wav
  Original Duration: 4.00 seconds
  Processed Duration: 4.00 seconds
  Processed File: processed_files\101415-3-0-2_processed.wav
File: 102305-6-0-0.wav
  Original Duration: 2.61 seconds
  Processed Duration: 4.00 seconds
  Processed File: processed_files\102305-6-0-0_processed.wav
File: 103074-7-0-0.wav
  Original Duration: 4.00 seconds
  Processed Duration: 4.00 seconds
  Processed File: processed_files\103074-7-0-0_processed.wav
File: 103258-5-0-0.wav
  Original Duration: 4.00 seconds
  Processed Duration: 4.00 seconds
  Processed File: processed_files\103258-5-0-0_processed.wav
File: 105415-2-0-1.wav
  Original Duration: 4.00 seconds
  Processed Duration: 4.00 seconds
  Processed File: processed_files\105415-2-0-1_processed.wav
Preprocessing complete. Files saved in processed_files
```

Post pre-sampling, and mono channel tuning using Librosa, and SoundFile libraries from Python, the next step involves noise reduction, so that the audio files are made more consistent, and are prepared for the next phase, i.e. Data Transformation.

Below are the results of the defined pre-processing steps for the Urbansound8k dataset which are visualized using spectrograms, and waveforms for identifying the visual differences between the raw and pre-processed audio files. Data Preprocessing end results for UrbanSound8k are provided in Figure 22 below along with raw audio and processed audio in Figures 23, 24, and 25.

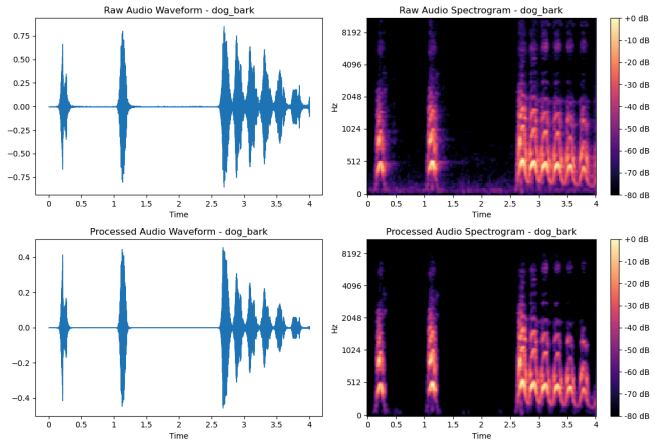
Figure 22

Results of Data Pre-Processing Steps For UrbanSound8k

```
Processed 101415-3-0-2_processed.wav
Processed 102305-6-0-0_processed.wav
Processed 103074-7-0-0_processed.wav
Processed 103258-5-0-0_processed.wav
Processed 105415-2-0-1_processed.wav
All files have been processed and saved in final_processed_files
```

Figure 23

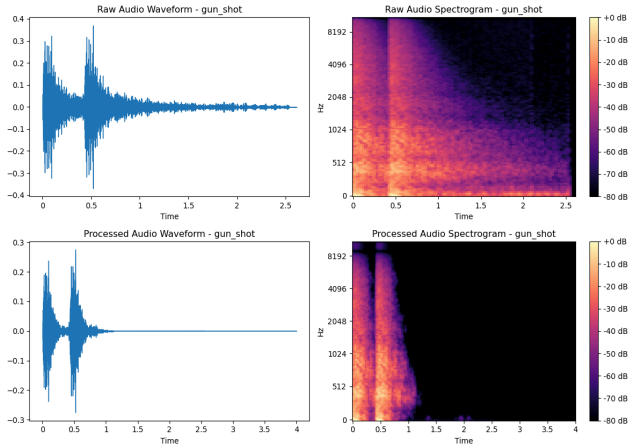
Representation of Dog Bark's Raw Audio and Processed Audio Using Waveforms and Spectrograms Graph From UrbanSound8k



The visual differences are clear between the raw and processed audio files from the above waveforms and spectrogram representations of the audio file. These above representations of the raw and pre-processed audio are just a way to convey the changes that occurred during the pre-processing phase.

Figure 24

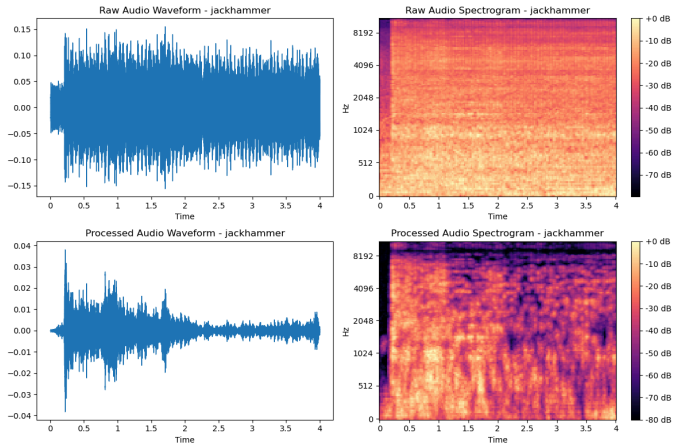
Representation of Gunshot Raw Audio and Processed Audio Using Waveforms and Spectrograms Graph From UrbanSound8k



The noise which is shown in the raw waveform across the x-axis is reduced in the corresponding processed waveforms and spectrograms.

Figure 25

Representation of Jackhammer Raw Audio and Processed Audio Using Waveforms and Spectrograms Graph From UrbanSound8k



With the above visualizations, the differences between raw and pre-processed are clearly shown, and validated against the respective pre-processing methods used.

For ESC-50, as the audio files are pre-processed with similar sampling rates, and are padded to 5 seconds for consistency, the remaining pre-processing steps are described below:

1. Using audio moving average with window=5, smoothing techniques (spectral gating) for the ESC audio files.
2. Applying noise reduction techniques from noisereduce library using the first 1.5 seconds of the video to gauge the noise, and then remove it.
3. Remove the pop and click sounds from the .wav audio files.

The results of the data pre-processing for the ESC-50 audio dataset is represented in the below Figure 26.

Figure 26

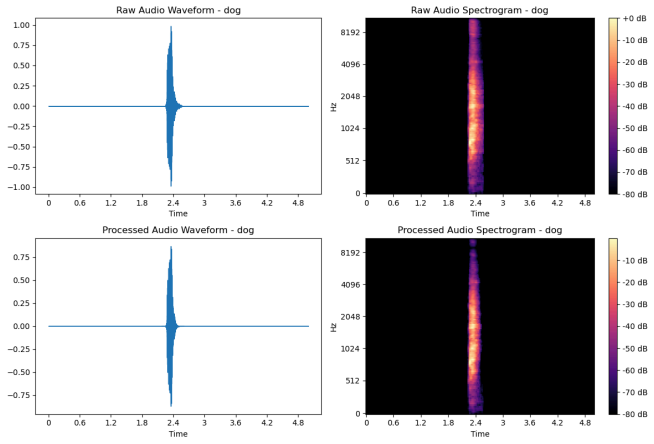
Results of Data Pre-Processing Steps For ESC-50 Audio Dataset

```
Processed 1-100032-A-0.wav
Processed 1-100038-A-14.wav
Processed 1-100210-A-36.wav
Processed 1-101296-A-19.wav
Processed 1-101336-A-30.wav
All files have been processed and saved in processed_files
```

The processed audio files are then transformed into waveforms and spectrograms, which are shown below Figure 27.

Figure 27

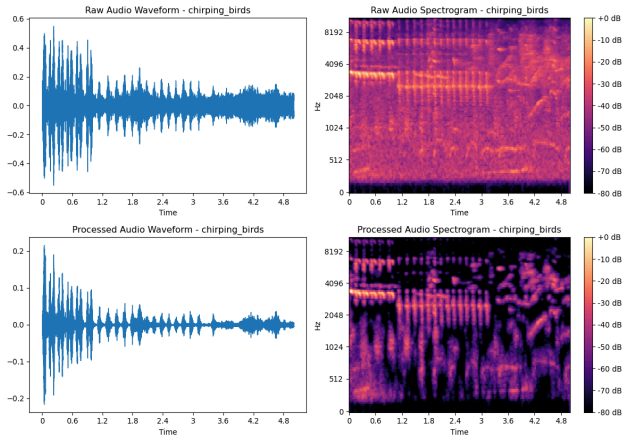
Representation of Dog's Raw Audio and Processed Audio Using Waveforms and Spectrograms Graph From ESC-50 Audio Dataset



There's no difference between the raw and pre-processed audio files, Basically, Dog's sudden bark has a clear spike in the middle which is represented in waveform and in spectrogram as well. Waveform focuses on frequency over time and Spectrogram focuses on the frequency itself but the raw and processed audio waveform shows there is no change in the audio. Some of the other files are shown below Figure 28.

Figure 28

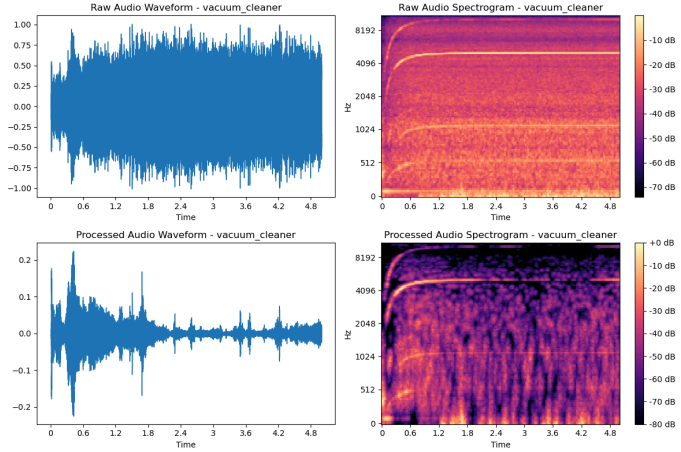
Representation of Chirping_birds Raw Audio and Processed Audio Using Waveforms and Spectrograms Graph From ESC-50



The noise has been largely removed from the raw audio files, which is depicted through the above and below comparisons. These processed files have been verified by physically listening to the processed audio files. The processed audio files are then transformed into waveforms and spectrograms, which are shown below Figure 29.

Figure 29

Representation of Vaccum_Cleaner Raw Audio and Processed Audio Using Waveforms and Spectrograms Graph From ESC-50 Audio Dataset



3.4 Data Transformation

This phase is the second most important phase for the defined research. As part of this phase, the team has employed the below techniques to the pre-processed files of the ESC-50 and Urbansound8k datasets respectively. Techniques have been listed down below step by step:

1. Convert pre-processed audio files into mel-scaled spectrograms and MFCC features.
2. Augment the derived mel-scaled spectrograms using SpecAugment, time stretching, and adding noise to the mel-scaled spectrograms.
3. Augment the pre-processed audio files using the same techniques and then extract the MFCC features for the respective audio files.
4. Using Min-max normalization on both augmented mel-scaled spectrograms and MFCC feature vectors extracted from pre-processed audio files..
5. PCA for dimensionality reduction for both mel-scaled spectrograms and MFCC feature vectors extracted from pre-processed files.

Feature Extraction: Audio to Image Conversion

The first step of this phase is to convert the pre-processed audio files from ESC-50 & Urbansound8k to convert into Image format, so that the data can be prepped for the modeling phase for Vision Transformers and CNN proposed models. This will also play a key role, as further data transformation techniques. There are numerous ways to transform audio data into Image, by making sure the signal is captured in the time-frequency domain. Some of the popular transformation techniques are described below, and it also discussed how this research employs these techniques to transform the audio data for both the datasets.

Mel-Scaled Spectrograms. Mel-scaled spectrograms are a type of spectral representation that display how the spectral density of a signal varies over time, with a frequency scale that

approximates human hearing sensitivity. In these spectrograms, frequency bins are converted to the Mel scale, which is a perceptual scale of pitches judged by listeners to be equal in distance from one another. This characteristic is particularly useful in urban audio classification tasks, as it emphasizes the frequencies more relevant to human listeners, such as distinguishing between different types of urban sounds like car horns, shouting, or machinery. When applied to both the datasets, which consist of diverse urban soundscapes, mel-scaled spectrograms can effectively capture and highlight distinguishing features that are crucial for accurate audio classification by deep learning models. For this conversion, pre-processed audio files from the previous phase will be used. These files are also transformed into Standard spectrograms, so that a comparative analysis can be drawn between the Mel-scaled spectrograms from pre-processed audio files versus Standard Spectrograms from pre-processed audio files. This visual comparative analysis can show the visual difference between standard spectrograms and mel-scale spectrograms.

Below, the pre-processed audio for ESC-50 files are run to create normal spectrograms, and also create mel-scaled spectrograms using Librosa Python library and the same shown in the below Figures 30 and 31.

Figure 30

Graphical Representation of Standard Spectrogram and Mel-Scale Spectrogram of Dog Bark's Audio From ESC-50 Dataset

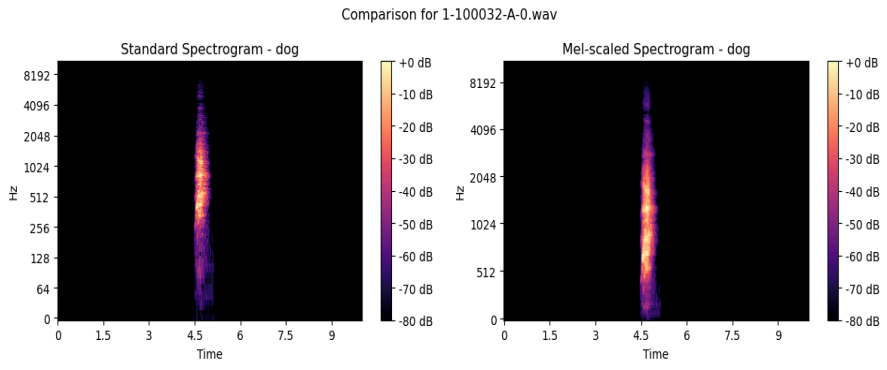
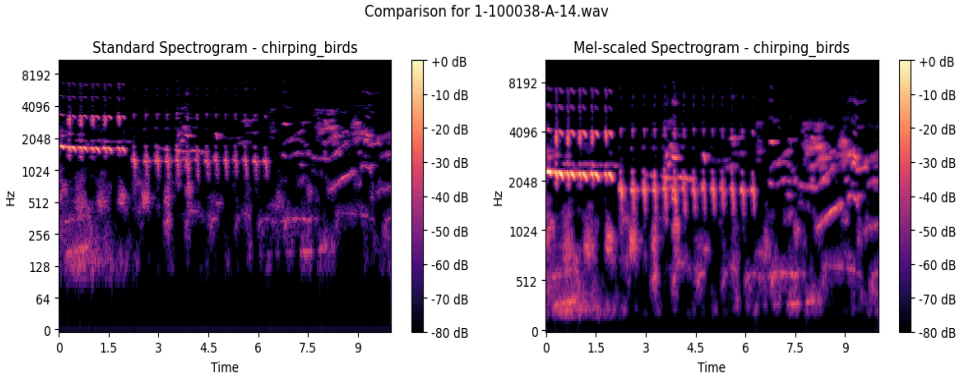


Figure 31

Graphical Representation of Standard Spectrogram and Mel-Scale Spectrogram of Chirping Birds Audio From ESC-50 Dataset



It's really clear that the Mel-scale spectrogram requires less signal data compared to the standard spectrograms created from the pre-processed files. Mel-scale transformation is more about re-scaling and re-binning the frequency axis, such that it aligns with the human auditory perception.

These transformation techniques have also been employed on Urbansound8k processed audio files, and to showcase the visual differences between the pre-processed audio files and mel-scaled spectrograms, the pre-processed audio files are converted in Standard spectrograms, so that the comparative analysis can be easily interpreted. Below are some of the illustrative examples for the UrbanSound8k dataset.

Below are Figures 32 and Figure 33 which are the Graphical Representations of Standard Spectrogram and Mel-Scale Spectrogram of Dog Bark's Audio from UrbanSound8k.

Figure 32

Graphical Representation of Standard Spectrogram and Mel-Scale Spectrogram of Dog Bark's Audio from UrbanSound8k

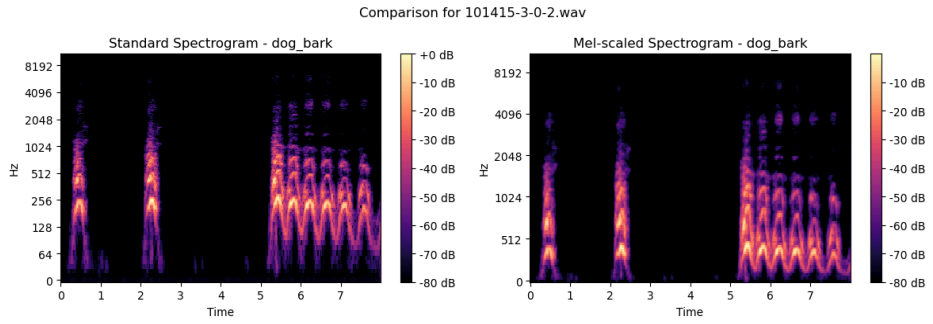
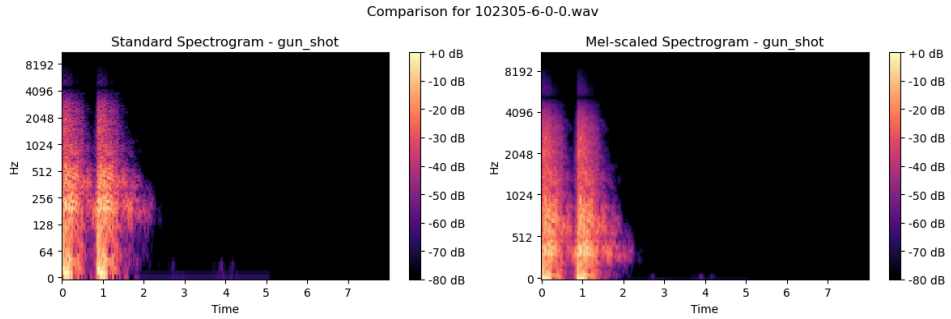


Figure 33

Graphical Representation of Standard Spectrogram and Mel-Scale Spectrogram of gun_shot Audio for UrbanSound8k



The standard spectrograms and mel-scaled spectrograms are processed from the previously processed audio files from Urbansound8k dataset. These mel-scaled spectrograms are created and stored in the drive, for further transformations.

Feature Extraction: Audio to Numerical Vectors Conversion

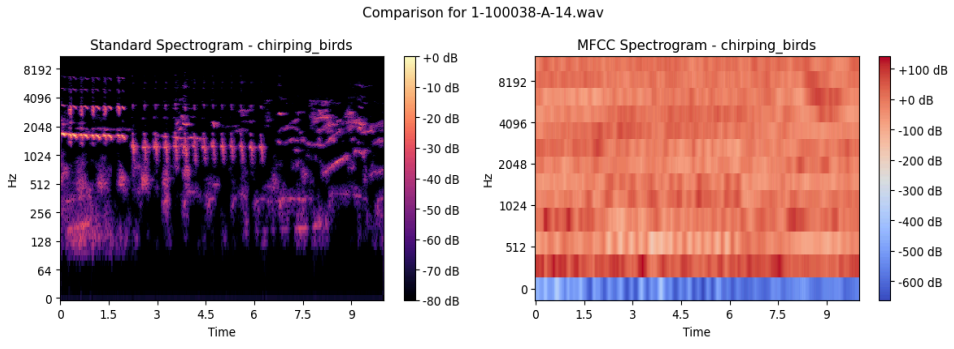
There are other ways to transform the audio files, like converting the audio files into numerical vectors such that the signal is captured in the numbers, so that they can be used as inputs for the proposed Machine learning and deep learning architectures. One of the most

popular ways to transform audio files & is employed in this research is discussed below;

Mel-Frequency Cepstral Coefficients (MFCC). MFCCs are a widely used feature in audio signal processing, especially in speech and music analysis. They represent the short-term power spectrum of a sound, derived by taking the logarithm of the spectrum and then the discrete cosine transform (DCT) of that log spectrum. MFCCs capture the timbral aspects of sound, which includes characteristics such as pitch and tone. In urban audio classification, MFCCs are valuable because they can efficiently represent the unique acoustic properties of various urban sounds, like sirens or footsteps, in a compact form. This compact representation is beneficial for machine learning models trained on data sets like UrbanSound8K and ESC-50, as it allows these models to learn to distinguish between different classes of urban sounds based on their timbral features. As the MFCC coefficients are a way to capture the signal across time, the idea is to convert these MFCC extracted features into spectrograms so that a comparison analysis can be drawn with the standard spectrograms created from the preprocessed audio files which can be observed in the below Figure 34.

Figure 34

Graphical Representation of Standard Spectrogram and MFCC of Chirping Birds Audio



This comparison may look a little different from the previous method, but it is expected as they are being visualized from the resulting MFCC feature vectors. The signal trends are

captured but not clearly visible in the above comparison. The below Figure 35 shows the MFCC vectors for a couple of categories.

Figure 35

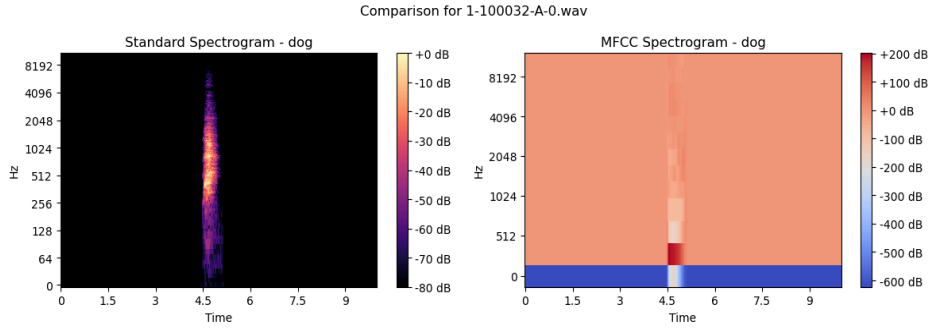
MFCC Vectors For a Set of Sample Files

file_name	category	MFCC_1	MFCC_2	MFCC_3	MFCC_4	MFCC_5	MFCC_6	MFCC_7	MFCC_8	MFCC_9	MFCC_10	MFCC_11	MFC
1-100032-A-0.wav	dog	-606.545471	8.536103	-5.553989	-3.939031	-1.440753	-0.493759	-0.831205	-0.045825	-0.071242	0.393014	0.251205	-0.18
1-100038-A-14.wav	chirping_birds	-548.692688	55.496609	-58.161198	1.075114	0.829863	-25.287722	-22.782166	-9.119798	-1.544654	-4.140959	-16.904331	0.35
1-100210-A-36.wav	vacuum_cleaner	-442.080261	124.681030	-35.587589	8.177324	6.392406	-8.669076	4.538768	11.500849	-4.472717	20.490097	-6.265850	-4.86
1-101296-A-19.wav	thunderstorm	-749.452820	70.292564	35.034378	17.034328	7.951870	2.294438	4.291696	4.632459	0.846530	0.787330	-0.609412	-4.10
1-101336-A-30.wav	door_wood_knock	-549.362122	61.235104	26.296762	13.740664	9.953383	2.586718	-0.556913	1.474708	3.315955	2.599650	-0.084419	-1.43

These numerical vectors are used as a feature set for the respective audio files, they are also processed as input features for proposed deep learning models. Below is another example which can be seen in Figure 36.

Figure 36

Graphical Representation of Standard Spectrogram and MFCC Spectrogram

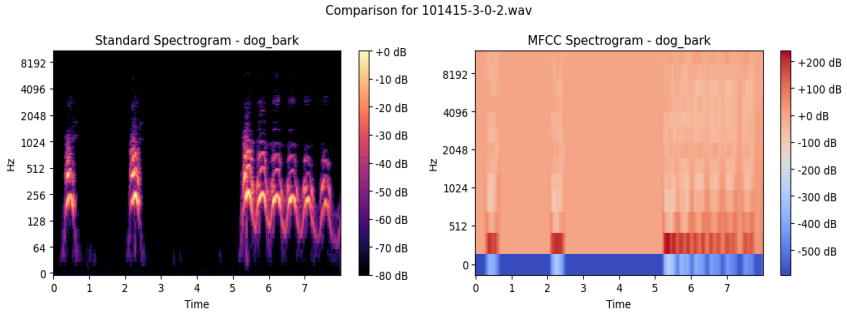


Similar approach is taken towards the Urbansound8k dataset, and below are the illustrative samples for some of a couple of example pre-processed audio files.

For Urbansound8k, below Figure 37 is the example from the MFCC transformation for the dog_bark.

Figure 37

Graphical Representation of Standard Spectrogram and MFCC Spectrogram of Dog_bark



The MFCC vectors for the couple of more categories are represented in Figure 38 below.

Figure 38

MFCC Vectors For the Sample Audio Files

file_name	category	MFCC_1	MFCC_2	MFCC_3	MFCC_4	MFCC_5	MFCC_6	MFCC_7	MFCC_8	MFCC_9	MFCC_10	MFCC_11	MFC
101415-3-0-2.wav	dog_bark	-525.202393	62.633968	15.415367	-5.001544	-11.657887	-7.621784	-4.184807	-11.152961	-12.913863	-9.254848	-11.155966	-7.14
102305-6-0-0.wav	gun_shot	-644.535828	53.733849	-1.404936	7.292768	0.776627	-4.377898	-1.278019	-2.275000	-2.673810	0.958518	-0.791422	-1.09
103074-7-0-0.wav	jackhammer	-566.210571	158.789078	-51.647129	33.974342	0.153390	-7.838573	-2.441485	16.915689	-19.259159	22.549768	-11.013209	-0.37
103258-5-0-0.wav	engine_idling	-788.971802	135.738739	3.828102	5.723444	21.613245	7.595826	12.924355	18.456434	12.804948	13.564838	12.199659	8.60
105415-2-0-1.wav	children_playing	-524.910889	149.577881	-31.069843	-16.213606	-1.077966	-15.220208	-6.479201	-3.838156	-17.736740	-6.075511	0.249043	-2.84

In the above example, the key signal components are preserved, it can be verified for the most part visually. And below Figure 39 is from the MFCC transformation for the engine_idling.

Figure 39

Graphical Representation of Standard Spectrogram and MFCC

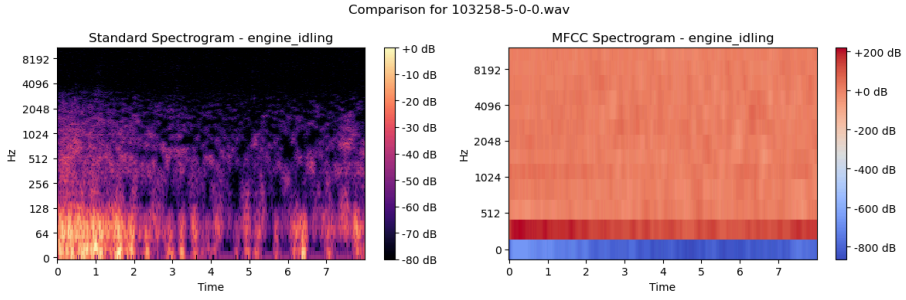


Image Data Augmentation: SpecAugment

One of the primary modes of input features for this research project are images for deep learning architectures like Vision transformers, CNN which are basically mel-scaled spectrograms created from pre-processed files. That is the first level of Data Transformation for the ViT, CNN models which is going to consider mel-scaled spectrograms as primary input features. The key idea behind doing this level of augmentation on Images arises from the usage of DL models like Vision Transformers, and at the same time to increase the generalization capability of the model, so that overfitting can be avoided as much as possible. It's also a given that Deep learning models need tons of training data, so augmentation also aids in creating the required synthetic training data.

As this research is dealing with Audio classification, the augmentation techniques need to be carefully chosen as some specific augmentations can directly change the basic nature of the audio file which can cause misclassification cases, which needs to be avoided at any costs. This research uses one of the most popular, and use case defined SpecAugment augmentation techniques, which is detailed below.

SpecAugment With Time Stretching and Noise Induction for Mel-Scaled Spectrograms. SpecAugment, introduced by Park et al. (2019), is a data augmentation technique used in machine learning, particularly for tasks involving audio signal processing such as speech recognition. It operates directly on the spectrogram (time-frequency representation) of the audio signal. The key idea of SpecAugment is to apply random alterations to the spectrogram, such as time warping, frequency masking, and time masking. These alterations modify certain parts of the spectrogram by masking blocks of consecutive frequency channels or time steps, effectively simulating variations in the audio that might occur naturally.

In the context of urban audio classification, using datasets like UrbanSound8K and ESC-50, SpecAugment is highly suitable for several reasons.

Below are some of the reasons.

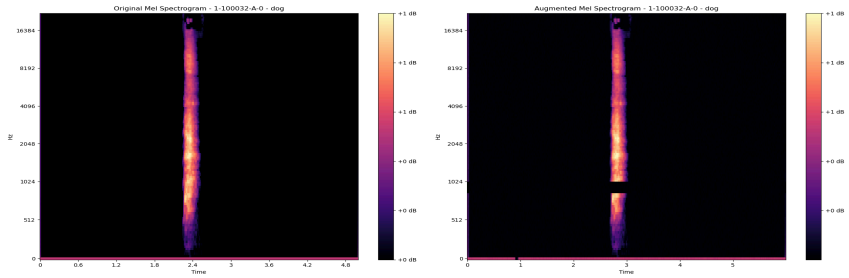
1. Robustness to Variability - Urban environments are characterized by a wide range of acoustic variations. SpecAugment introduces artificial variability into the training process, which can make ML models more robust to real-world, unforeseen variations in urban sounds.
2. Data Augmentation - Audio datasets may have limitations in terms of the variety and quantity of samples. SpecAugment artificially expands the dataset, providing more diverse training samples and helping prevent overfitting.
3. Preservation of Class Identity - While SpecAugment alters the spectrogram, it does so without changing the fundamental nature or class of the audio signal. For instance, a siren sound with a portion of its frequencies masked is still identifiable as a siren. This characteristic is crucial, as it ensures that the augmentation process does not mislead the training of classification models.
4. Improved Generalization - By training on augmented data, models can learn to focus on the most crucial features for classification and ignore irrelevant variations, leading to improved generalization to new, unseen data.

Along with SpecAugment, other augmentation techniques like Time stretching, and Noise induction have also been employed on the previously created mel-scaled spectrograms. This also aids in re-creating real urban audio, which will help in generalization of the model.

Below Figure 40 is representation of the comparative analysis that has been done on the original Mel-scaled spectrograms and their respective Augmented spectrograms.

Figure 40

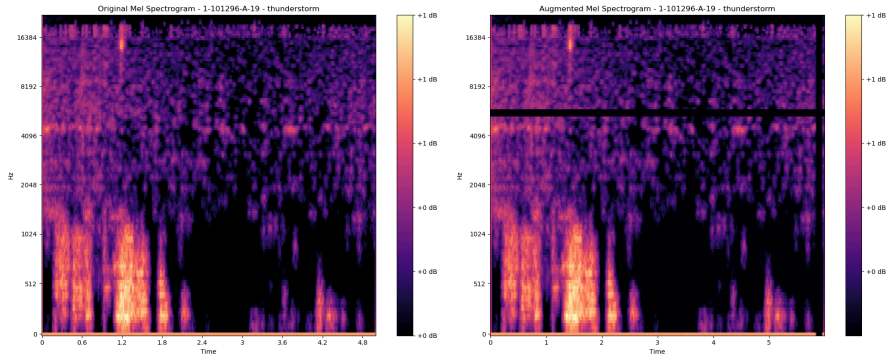
Comparison of Original Mel-Spectrogram and Augmented Mel Spectrogram



The Below Figure 41 is the visual representation of the original Mel-scaled spectrograms and their respective Augmented spectrograms.

Figure 41

Visualization of The Original Mel-Spectrogram and Augmented Mel Spectrogram



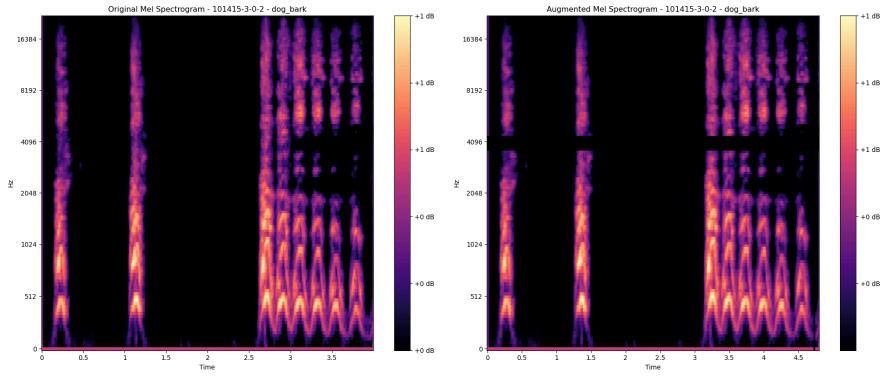
It can be clearly observed that some parts of the spectrogram have been masked, this is due to the fact that SpecAugment augments the data using Frequency masking, and time masking. The added noise can also be visualized if observed carefully. This is done for all the pre-processed ESC-50 audio files. This augmentation technique will act as primary augmentation technique employed for the mel-scaled spectrograms that were created from the pre-processed audio files of both the datasets, Urbansound8k & ESC-50.

Similarly, employing the same techniques for the pre-processed audio files of

Urbansound8k dataset, the results are visually represented below as a comparison between the original spectrograms (pre-processed audio files) and the augmented spectrograms. This can be seen below in Figure 42 for the dog_bark and in Figure 43 for the jackhammer respectively.

Figure 42

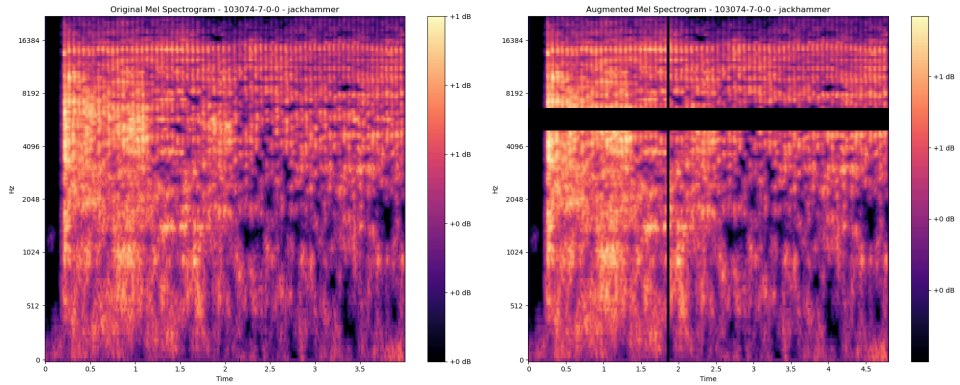
Mel Spectrogram from Pre-processed Audio Files vs. Augmented Mel Spectrograms for Dog_bark



The masking differences are clearly visible here as well, this is considered satisfactory for the Augmentation phase of this research project.

Figure 43

Mel Spectrograms from Pre-processed Audio Files vs. Augmented Mel Spectrograms for Jackhammer



Audio Augmentation For MFCC Extracted Features

As this research project is also utilizing MFCC extracted features, data augmentation on the raw pre-processed audio files can provide more training samples in the form of MFCC features, these augmented features can help in avoiding the overfitting of the respective DL architectures. The audio augmentation techniques that are being employed in this phase are similar to that of SpecAugment, which are Time masking, frequency masking, white noise induction, and time stretching. The augmented MFCC features are then extracted and stored in respective data frames and csv files.

1. SpecAugment: F = 27 , T = 100, num_masks = 1
2. Time-stretching: stretching_rate = 0.9
3. White Noise addition: 0.05 of the original audio file

Below Figure 44 is the comparison of the time stretch between the original MFCC Spectrogram and the Augmented MFCC Spectrogram.

Figure 44

Comparison of Time Stretch for Original MFCC Spectrogram and Augmented MFCC Spectrogram.

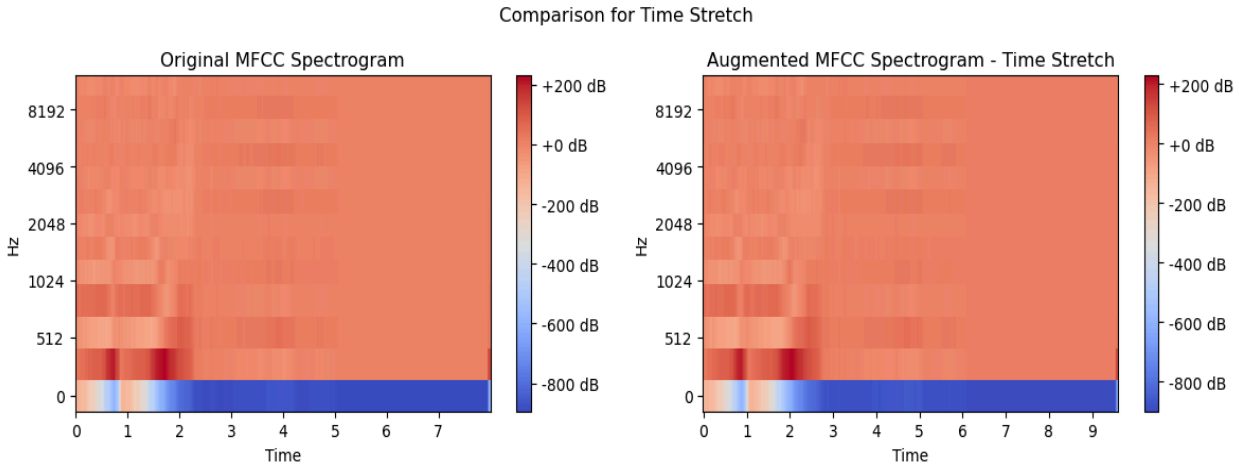
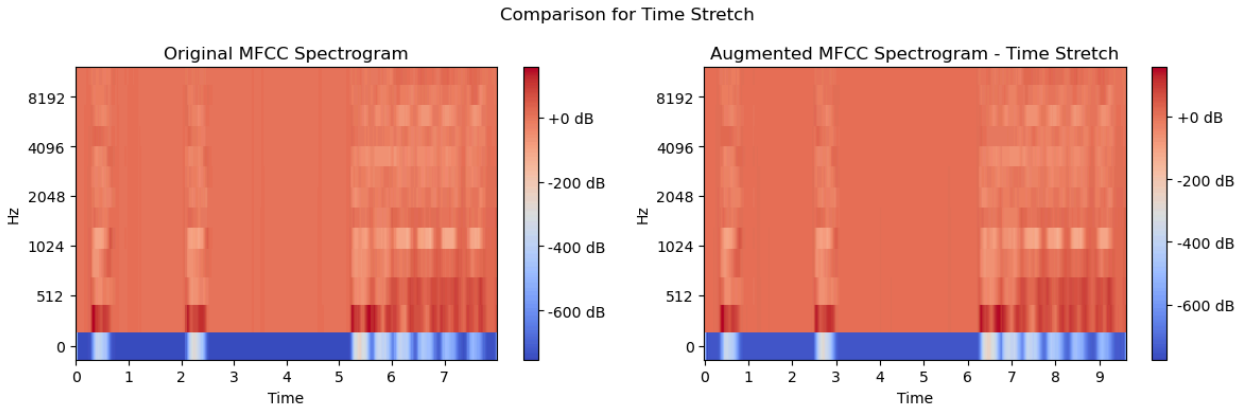


Figure 44 is the comparison of the time stretching augmentation on the MFCC extracted features. The distribution of the data is not completely changed by the time it has stretched in the x-axis. And below Figure 45 represents the comparison for the ESC-50 Audio sample.

Figure 45

Comparison of The Time Stretch Augmentation for ESC-50 Audio Sample



All the Augmented MFCC features are appended in the respective MFCC features CSVs for the respective datasets and are stored with a unique identifier as shown in Figure 46 below.

Figure 46

Augmented MFCC Features for a Set of Audio Files for UrbanSound8k Dataset

	file_name	category	MFCC_1	MFCC_2	MFCC_3	MFCC_4	MFCC_5	MFCC_6	MFCC_7	MFCC_8	MFCC_9	MFCC_10	M
0	101415-3-0-2.wav	dog_bark	-525.202393	62.633968	15.415367	-5.001544	-11.657887	-7.621784	-4.184807	-11.152961	-12.913863	-9.254848	-11
1	102305-6-0-0.wav	gun_shot	-644.535828	53.733849	-1.404936	7.292768	0.776627	-4.377898	-1.278019	-2.275000	-2.673810	0.958518	-0
2	103074-7-0-0.wav	jackhammer	-566.210571	158.789078	-51.647129	33.974342	0.153390	-7.838573	-2.441485	16.915689	-19.259159	22.549768	-11
3	103258-5-0-0.wav	engine_idling	-788.971802	135.738739	3.828102	5.723444	21.613245	7.595826	12.924355	18.456434	12.804948	13.564838	12
4	105415-2-0-1.wav	children_playing	-524.910889	149.577881	-31.069843	-16.213606	-1.077966	-15.220208	-6.479201	-3.838156	-17.736740	-6.075511	0
5	101415-3-0-2_aug.wav	dog_bark_augmented	-498.942273	59.502270	14.644599	-4.751466	-11.074992	-7.240695	-3.975566	-10.595313	-12.268170	-8.792106	-10
6	102305-6-0-0_aug.wav	gun_shot_augmented	-612.309036	51.047156	-1.334689	6.928130	0.737796	-4.159003	-1.214118	-2.161250	-2.540120	0.910592	-0
7	103074-7-0-0_aug.wav	jackhammer_augmented	-537.900043	150.849624	-49.064773	32.275625	0.145720	-7.446645	-2.319411	16.069905	-18.296201	21.422280	-10
8	103258-5-0-0_aug.wav	engine_idling_augmented	-749.523212	128.951802	3.636697	5.437272	20.532583	7.216035	12.278137	17.533613	12.164700	12.886596	11
9	105415-2-0-1_aug.wav	children_playing augmented	-498.665344	142.098987	-29.516351	-15.402926	-1.024067	-14.459198	-6.155241	-3.646248	-16.849903	-5.771736	0

Min-Max Normalization

Firstly, min-max normalization aids in standardizing the range of audio features. Both the ESC-50 and UrbanSound8K datasets comprise a diverse array of urban sounds, from animal noises to street music. These sounds inherently possess varying amplitudes and frequencies. Without normalization, these differences can lead to a skewed representation of data, where certain louder or more dominant sounds might overshadow subtler audio features.

By applying min-max normalization, each feature is scaled to a specified range, often between 0 and 1. This scaling ensures that all features contribute equally to the learning process, allowing the deep learning model to learn patterns without bias towards louder or more prominent sounds.

Secondly, min-max normalization enhances the convergence speed of deep learning architectures. Deep learning models, especially those used for audio classification like Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), often perform better and train faster when the input features are normalized. Features with large variances and different scales can cause the training process to be unstable, leading to longer training times and making it harder for the network to converge.

By normalizing the input data, gradients are more stable, and the learning process is more efficient. This is particularly beneficial in urban audio classification, where models need to be trained on large datasets like ESC-50 and UrbanSound8K, containing thousands of audio samples.

Below are Figure 47 and Figure 48, that represent the comparison of Augmented Mel-scale Spectrogram and the Normalized Mel-scale Spectrogram for dog_bark and jackhammer of UrbanSound8K dataset respectively.

Figure 47

Normalized Augmented Mel-Scale Spectrogram vs. Augmented Mel-Scale Spectrogram From The UrbanSound8k Dataset For Dog_bark

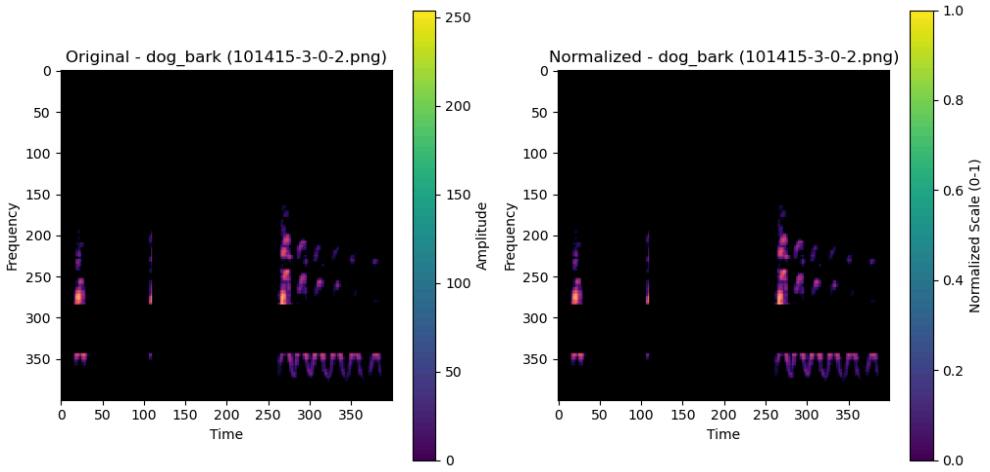
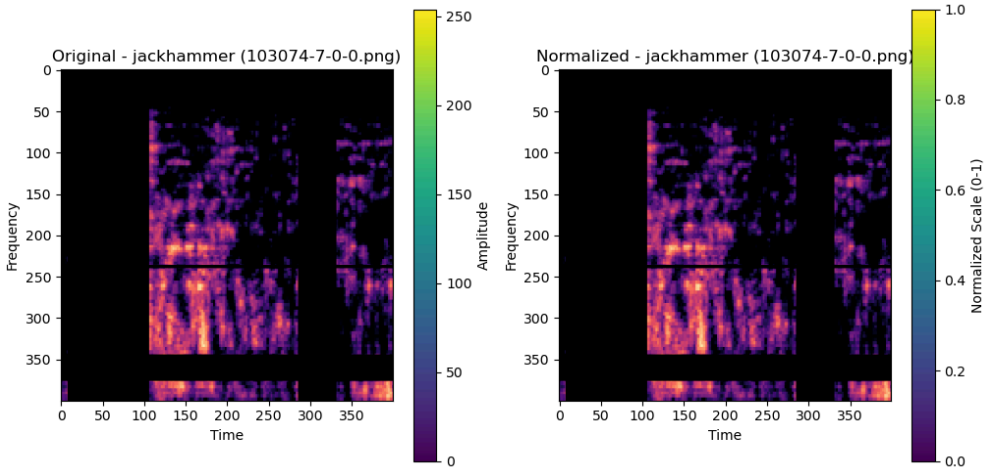


Figure 48

Normalized Augmented Mel-Scale Spectrogram vs. Augmented Mel-Scale Spectrogram From the UrbanSound8k Dataset For Jackhammer



If observed visually, the scale of the amplitude is normalized between both the spectrograms but the distribution is undisturbed, this will help in avoiding the overfitting, and

increase the generalization of the testing. Below is a similar analysis that is shown in Figures 49 and 50 for the ESC-50 dataset, where min-max normalization is applied across the dataset.

Figure 49

Normalized Augmented Mel-Scale Spectrogram vs. Augmented Mel-Scale Spectrogram From the ESC-50 Dataset

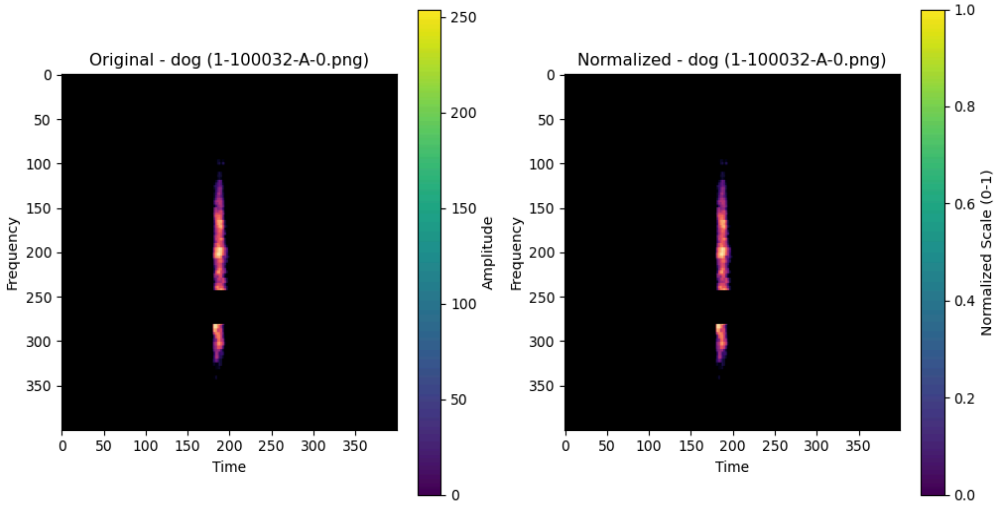
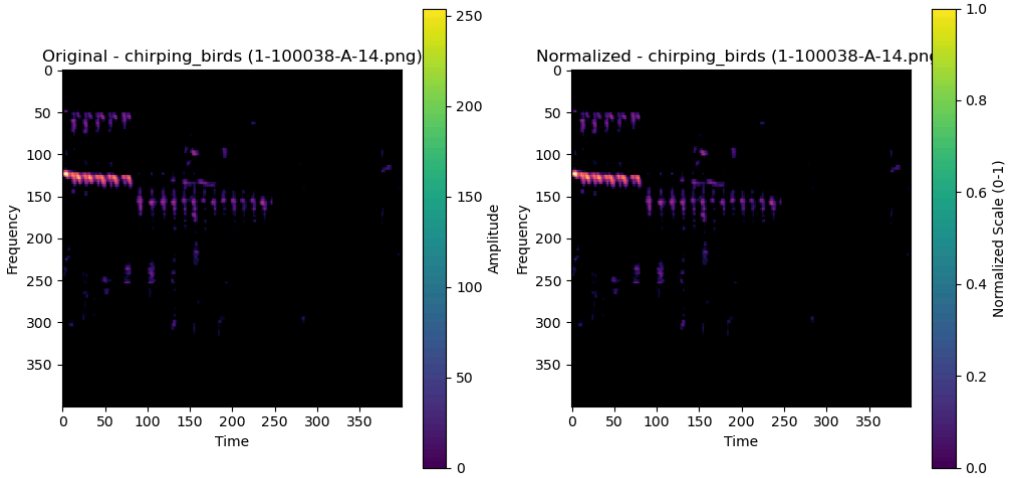


Figure 50

Normalized Augmented Mel-Scale Spectrogram vs. Augmented Mel-Scale Spectrogram From the ESC-50 Dataset



Feature Reduction Techniques: PCA

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction, allowing large datasets to be compressed into a smaller set of variables while retaining most of the important information. The fundamental working principle of PCA involves identifying the directions, or 'principal components,' in which the data varies the most. It starts by computing the eigenvectors and eigenvalues from the covariance matrix of the data, which represent the directions of maximum variance and the magnitude of these directions, respectively.

The first principal component captures the most variance, the second captures the next highest variance under the constraint that it is orthogonal to the first, and so on. By projecting the original data onto these principal components, PCA transforms the data to a new coordinate system, reducing its dimensionality. This process results in a more compact representation of the original dataset, highlighting the underlying structure and making it easier to explore and visualize, especially useful in fields like machine learning and data analytics.

Applying Principal Component Analysis (PCA) on Mel Frequency Cepstral Coefficients (MFCCs) holds significant relevance for urban sound classification tasks, particularly when working with datasets like ESC-50 and UrbanSound8K. Urban sound environments are inherently complex and contain a wide range of frequencies and amplitudes, stemming from various sources like traffic, human activities, or natural elements. MFCCs are highly effective in capturing the key frequency characteristics of these audio signals, making them a go-to feature for audio analysis. However, the high dimensionality of MFCCs can pose challenges in terms of computational efficiency and model performance.

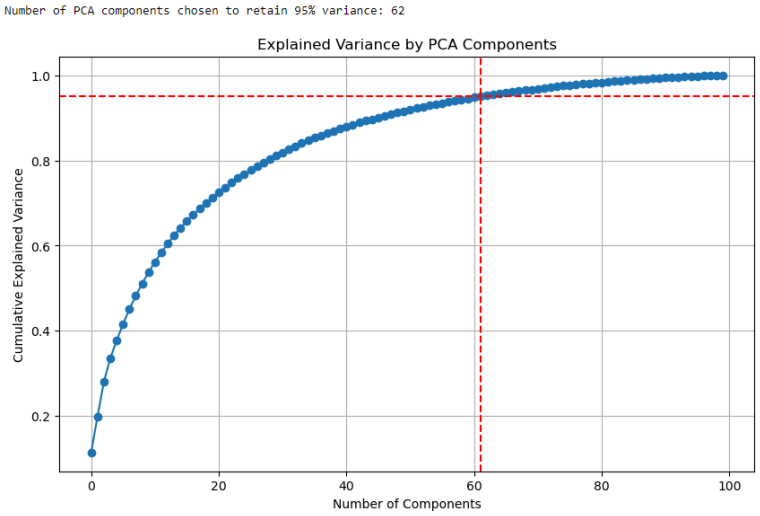
This is where PCA becomes instrumental. By applying PCA to the MFCC features, we can reduce the dimensionality of the dataset while retaining most of the crucial information. This dimensionality reduction is particularly beneficial in urban sound classification as it leads to simpler, more efficient machine learning models without significantly compromising the accuracy or the ability to differentiate between diverse sound classes in datasets like ESC-50 and UrbanSound8K. Moreover, reducing the number of features helps in mitigating issues like overfitting, especially when dealing with the limited size of datasets typically available in the field of environmental sound classification.

Furthermore, PCA can also aid in uncovering underlying patterns within the data, which might not be immediately apparent. By transforming the MFCCs into principal components, PCA can help in highlighting the features that contribute most significantly to the variance in the dataset, thereby potentially revealing distinguishing characteristics of different urban sounds. This aspect of PCA not only contributes to building more effective classification models but also offers insights into the nature of the soundscapes captured in datasets like ESC-50 and UrbanSound8K, which is invaluable for urban planners, environmentalists, and researchers in the field of acoustic ecology.

The below Figure 51 shows a plot which gives the ideal number of Principal components for the UrbanSound8k dataset. This is done for Fold 1 and visualized, the other folds have also been processed the same way, it’s just that showing everything in a single visualization is quite computationally expensive. Also Figure 52 shows the distribution of the PCA components of the dataset(class_id in legend).

Figure 51

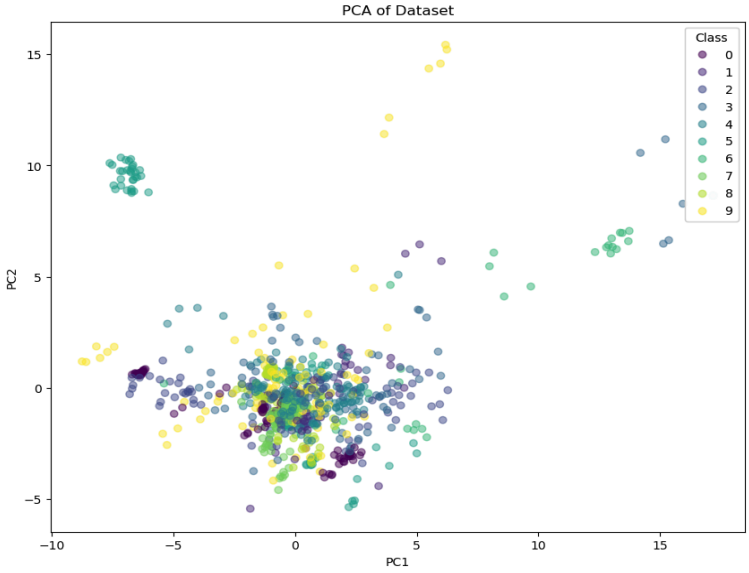
Number of Principal Components Chosen for The UrbanSound8k Dataset



From the above plot, it's clear that we need 62 PCA components for the UrbanSound8k dataset, to further visualize the distribution of the dataset, class_id has been used as the legend, and the below distribution is PCA components distribution of the dataset.

Figure 52

Distribution of the PCA Components of the Dataset (Class_id in Legend)



Below are the heatmap distributions in Figure 53 for the UrbanSound8k dataset samples, a comparison between the MFCC extracted features and the PCA reduced MFCC features for individual samples of the dataset including x.

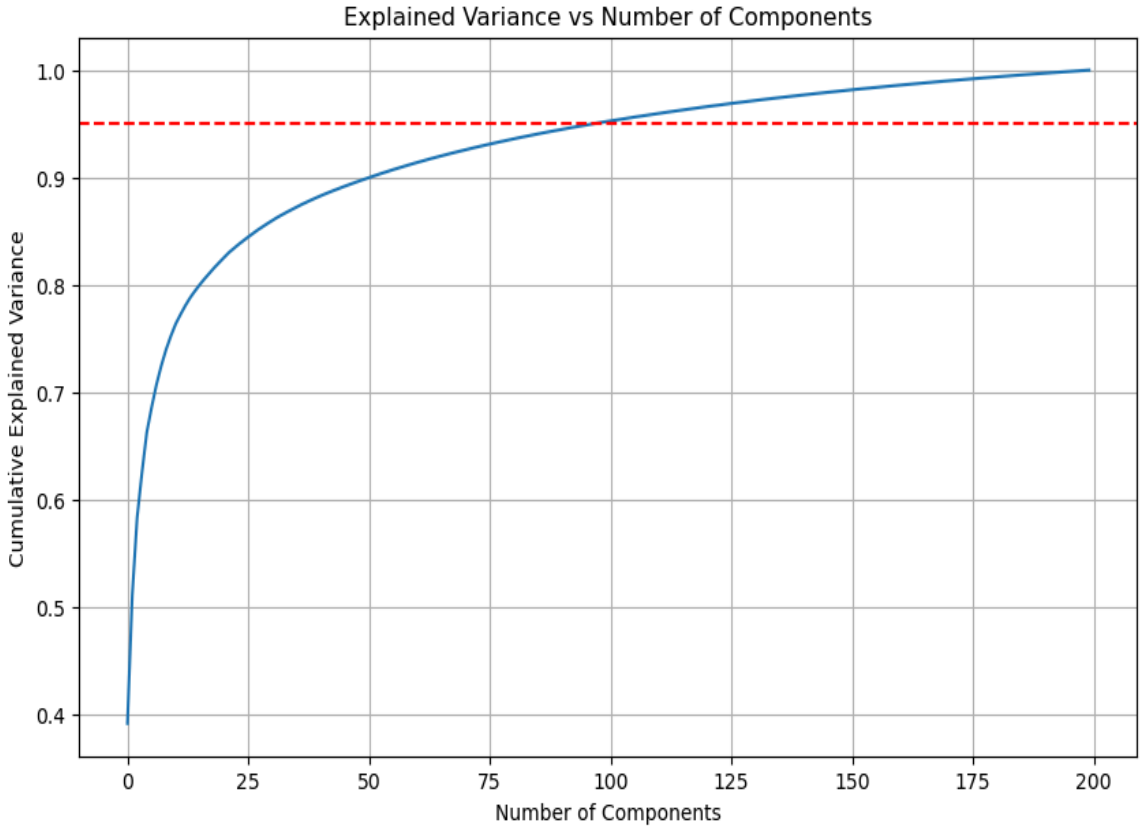
Figure 53

Comparison of the MFCC Extracted Spectrograms vs. PCA Reduced MFCC

The next step is to extract PCA components for the augmented-mel-scaled-spectrograms for ESC-50 dataset which were created as part of the previous steps, PCA analysis for the spectrograms in Figure 54 below.

Figure 54

Optimal Number of Components for PCA Dimension Reduction



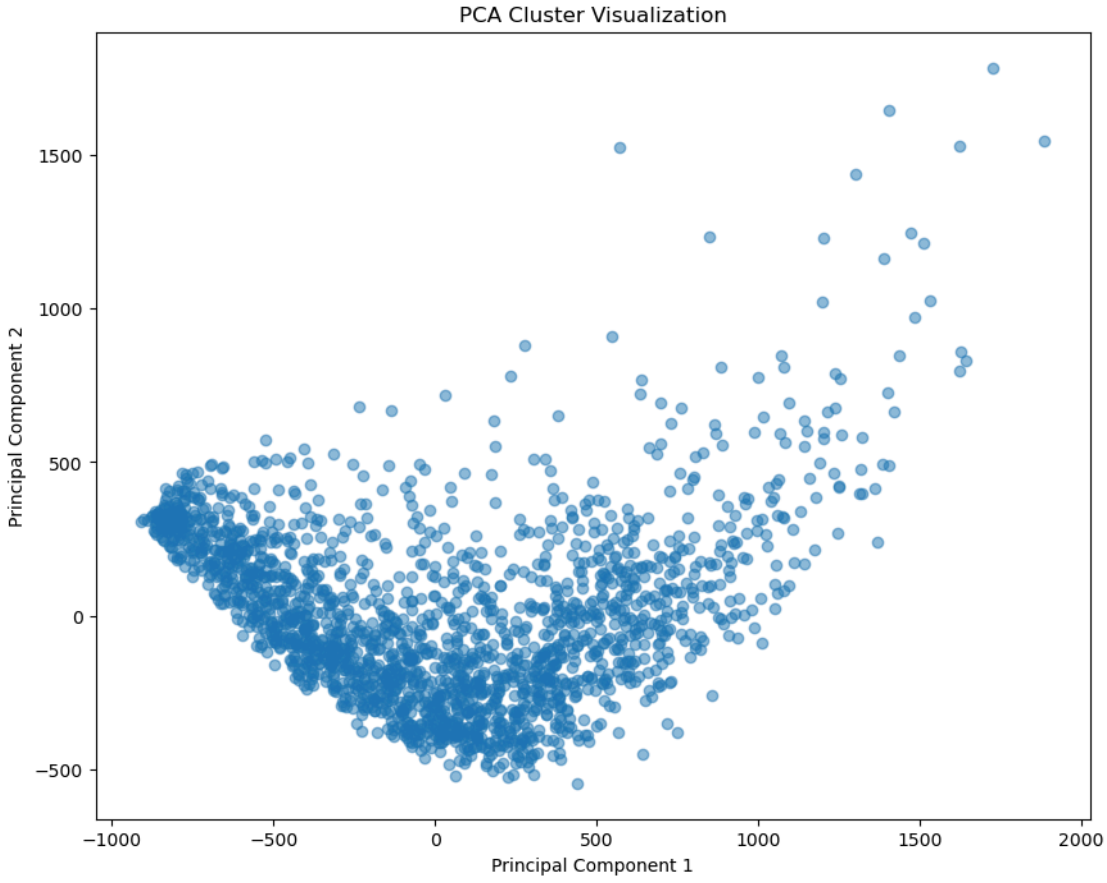
Number of components to explain 95% variance: 98

To maintain the 95% variance from the original distribution, the team used 98 components for ESC-50 derived augmented mel-scaled spectrograms.

Below is Figure 55 that visualizes the distribution of the reduced PCA components across two principal component values mapped against each other on x-axis and y-axis.

Figure 55

Reduced PCA Components

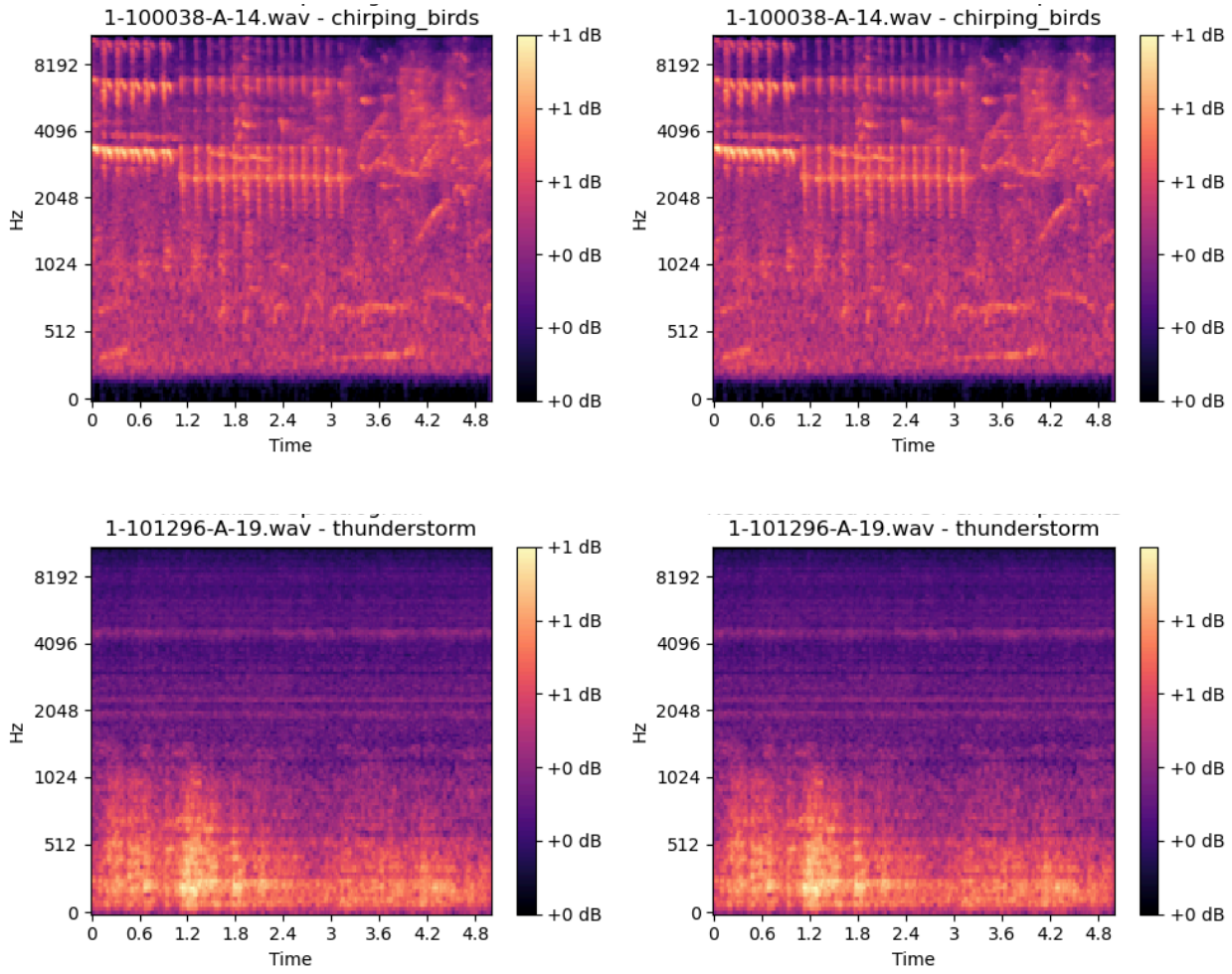


This distribution can also include class information using the class_id which were generated using One-hot coding mechanism, but plotting 50 different classes would be infeasible, so this visualization can only be used to understand the distribution of the reduced PCA components.

The next step is to reconstruct the spectrogram using librosa, to validate whether the PCA reduced spectrogram actually captures the signal information or not. This is shown in Figure 56 below. This will help in further tuning the hyperparameters of PCA, to either experiment with a higher number of principal components or vice versa. This

Figure 56

Comparison of PCA Reduced Spectrogram (Right) vs. Augmented Normalized-Spectrogram (Left)



From the above comparisons, it is easily observed that with 98 components the spectrogram images are almost recreated in the identical manner. This means, there is still a scope to decrease the n_components while deciding the parameters for PCA on the dataset.

As the idea of having 98 components is applicable to the training dataset, the hyper parameter tuning for PCA must be carefully considered, as any rapid or unexpected change in PCA directly relates to the variance, and this directly affects the loss function of the respective

DL architectures.

Feature Reduction Techniques: t-SNE (Optional)

The next step of the proposed data engineering approach is to use suitable feature reduction techniques. As the project is dealing with audio data, it was important that a non-linear feature reduction technique needs to be implemented to capture the reduced features from the augmented spectrograms, this helps in reducing the amount of input, and focuses on the most informative features which can help in predicting the class of the respective audio file.

t-SNE (t-distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique well-suited for handling high-dimensional datasets like augmented spectrograms. It simplifies high-dimensional data into a lower-dimensional space (usually 2D or 3D), making it easier to visualize and interpret. By converting similarities between data points into joint probabilities and minimizing the divergence between these probabilities in both high and low-dimensional spaces, t-SNE reveals patterns and clusters in the data. When applied to augmented spectrograms, it can help in identifying patterns or groupings based on audio characteristics, which are not readily apparent in the original high-dimensional space.

Using t-SNE-reduced features as inputs to vision transformers and other deep learning architectures is an effective strategy. The reduced features, serving as a preprocessing step, alleviate the curse of dimensionality, ensuring that the essential information is retained while reducing the data complexity. Vision transformers, which are effective in processing image data, can efficiently utilize these reduced features for various tasks, including classification and clustering. Similarly, other deep learning models can benefit from these condensed representations, leading to improved learning efficiency and model performance. This approach, combining t-SNE and advanced neural networks, facilitates a more thorough analysis and

understanding of complex data like augmented spectrograms.

Employing t-SNE on the augmented spectrograms, the output will be in the form of numerical vectors for the respective augmented audio files, shown in Figure 57 below for ESC-50 dataset.

Figure 57

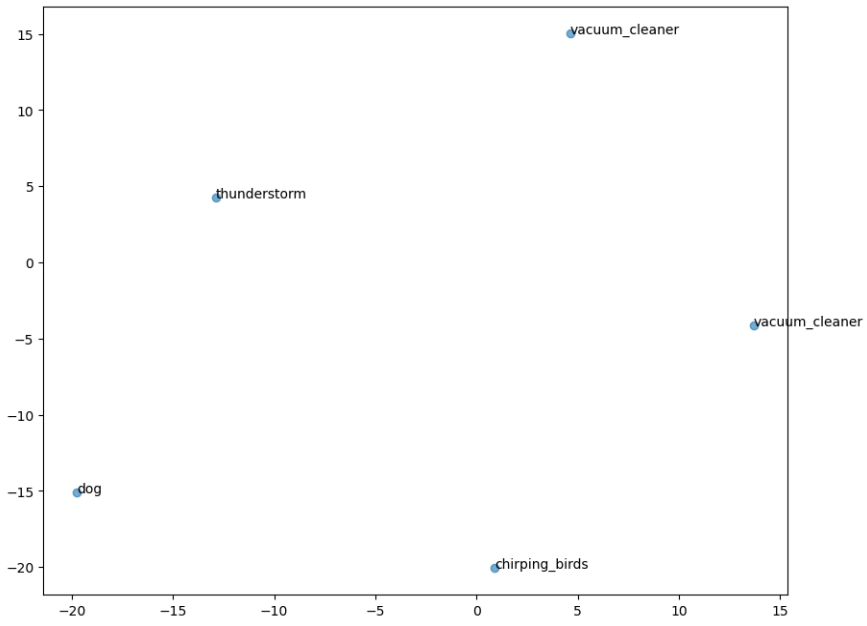
t-SNE on The Augmented Spectrograms

```
1-100032-A-0_dog_augmented.png: [-83.40003    27.205458   -0.6991224]
1-100038-A-14_chirping_birds_augmented.png: [-55.51846  -98.95195  -11.118279]
1-100210-A-36_vacuum_cleaner_augmented.png: [29.29944 79.26536 37.31968]
1-101296-A-19_thunderstorm_augmented.png: [ 41.11243  -19.378893 -45.660362]
1-101336-A-30_door_wood_knock_augmented.png: [  1.3918738 -37.194252   88.32068  ]
```

Plotting the extracted t-SNE features on a scatter plot in 2D results. This can be seen in the below Figure 58.

Figure 58

Plotting the Above Extracted t-SNE Features of a Sample in 2D Plot



For Urbansound8k, the same feature reduction technique has been employed, results shown in Figure 59 below.

Figure 59

t-SNE Features for the Pre-Processed Augmented Files From Urbansound8k

```
101415-3-0-2_dog_bark_augmented.png: [  32.963177 -101.95952    75.10803 ]
102305-6-0-0_gun_shot_augmented.png: [-133.46867 -157.89616  190.67015]
103074-7-0-0_jackhammer_augmented.png: [ 120.72343   -59.472706 -112.71623 ]
103258-5-0-0_engine_idling_augmented.png: [-24.628077 114.50226   78.40461 ]
101415-3-0-2.ogg_unknown_augmented.png: [ -50.704124   85.92215  -146.7653  ]
```

With the reduced feature vectors for the respective audio files, the idea is to experiment with both numerical vectors and spectrograms as inputs based on DL architecture use cases.

Data Regularization

Data regularization in machine learning refers to a set of techniques aimed at preventing models from overfitting to their training data, thereby improving their ability to generalize to unseen data. In the context of audio classification tasks, such as those using the ESC-50 and UrbanSound8K datasets, regularization is particularly crucial due to the complexity and variability inherent in audio data. Different deep learning models like Vision Transformers (ViT), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and hybrid models combining CNNs with Bidirectional Gated Recurrent Units (BiGRUs) might react differently to the same regularization techniques due to their unique architectural characteristics.

Data augmentation is a powerful tool in the realm of data regularization, particularly for complex tasks like audio classification. It works by artificially expanding the training dataset through the creation of modified versions of existing data. In the context of audio processing, this could involve techniques such as adding background noise, altering the pitch or speed of audio clips, or even generating synthetic audio samples. The key benefit of data augmentation

lies in its ability to introduce a greater variety of scenarios and conditions into the training process. This diversity helps in training more robust models that are less likely to overfit to the specific characteristics of the training data, thereby improving their ability to generalize to new, unseen data.

Min-max normalization, another critical technique, plays a different but equally important role in data regularization. It involves scaling all numeric features in the dataset to a common scale, typically between 0 and 1. By ensuring that all features contribute equally to the model's training, min-max normalization prevents issues where certain features dominate the learning process due to differences in scale. This is particularly relevant in audio classification where features like mel-scaled spectrograms and MFCCs can vary widely in scale and range. Normalization ensures that the model doesn't become biased towards certain features, promoting a more balanced learning and generalization capability.

Feature reduction using PCA (Principal Component Analysis) complements these techniques by simplifying the feature space. PCA reduces the dimensionality of the data by transforming the original features into a smaller set of uncorrelated components that capture the most significant variance in the data. In audio classification tasks, where features like spectrograms or MFCCs can be high-dimensional, PCA helps in condensing the data into a more manageable form. This reduction not only makes the computational process more efficient but also helps in mitigating overfitting. By focusing on the most informative aspects of the data and discarding redundant or less informative features, PCA enables models to learn more general patterns rather than memorizing specific details of the training data.

Together, these three techniques - data augmentation, min-max normalization, and feature reduction using PCA - form a comprehensive approach to data regularization. They address

different aspects of overfitting and generalization, making them valuable in developing effective and robust models for audio classification tasks.

Below is the representation of the performance metrics of a pre-trained Vision transformers model from Facebook, it can be clearly seen that while employing extra 500 samples for ESC-50 from augmentation, the model has given slightly better accuracies at early epochs. Below Figure 60 shows the results of before using augmentation, and other techniques, just simply using the ESC-50 dataset as it is. Recorded accuracies are below.

Figure 60

Recorded Accuracies Using the ESC-50 Dataset

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	No log	2.008242	0.584000	0.550404
2	No log	0.929154	0.820000	0.817615
3	2.156900	0.618666	0.888000	0.881716
4	2.156900	0.514609	0.864000	0.860825
5	0.261100	0.380467	0.912000	0.908747
6	0.261100	0.360609	0.908000	0.904065
7	0.020500	0.330251	0.912000	0.909176
8	0.020500	0.344970	0.908000	0.905317
9	0.006700	0.317899	0.908000	0.904954
10	0.006700	0.317916	0.904000	0.900190
11	0.003000	0.315967	0.908000	0.905216
12	0.003000	0.314842	0.904000	0.900190
13	0.002300	0.313799	0.908000	0.905216
14	0.002300	0.313546	0.908000	0.905216
15	0.002100	0.313496	0.908000	0.904954

From the above model performance metrics, it can be seen that the accuracy at 8th epoch, the accuracy came up to be 90.8%, it had 91.2% in earlier epoch, but it can be seen that the validation loss still increased and it's not stable. Below is another chart, shown in Figure 61,

which shows the updated performance metrics after applying the data engineering principles - SpecAugment augmentation, PCA feature reduction which closely relate to the L2 regularization.

Figure 61

Performance Metrics After Applying Feature Extraction, Data Augmentation, and Feature Reduction Techniques

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	No log	0.824296	0.770000	0.758251
2	1.484100	0.517964	0.854000	0.845833
3	0.163800	0.526585	0.874000	0.868784
4	0.023800	0.379786	0.896000	0.893926
5	0.023800	0.347611	0.912000	0.910281
6	0.001200	0.340421	0.914000	0.912292
7	0.000300	0.339606	0.914000	0.912292
8	0.000200	0.339820	0.914000	0.912292
9	0.000200	0.339961	0.914000	0.912292
10	0.000200	0.341098	0.916000	0.914272
11	0.000100	0.341989	0.918000	0.916220
12	0.000100	0.343512	0.918000	0.916220
13	0.000100	0.343873	0.918000	0.916443

It can be clearly seen that the data engineering principles have worked here, the model is diverging much sooner than previous, at 7th epoch its giving 91.4% accuracy with a stable validation loss, it achieved 91.2% in the 5th epoch only, which shows the improvement in the model training, as per the expectations. This provides us an understanding of how L2 regularization works. The team will implement respective L1/L2 regularization techniques as per the use case, but the idea is to implement it as part of the Optimization processes which are usually a part of the Modeling phase. Below is the theory of L2 Regularization, and how it is applicable to this research project.

L1 regularization, also known as Lasso regularization, adds a penalty equal to the absolute value of the magnitude of coefficients. This approach can lead to models where some weights are effectively reduced to zero, performing a sort of automatic feature selection. In the context of audio classification, L1 regularization can be particularly useful in models like CNN+LSTM or CNN-BiGRUs, which process time-series data. These models benefit from L1's ability to identify and prioritize the most relevant features in a complex audio signal. However, L1 is less commonly used in architectures like ViT and CNNs but can still be valuable for feature selection in these high-parameter models.

On the other hand, L2 regularization, or Ridge regularization, adds a penalty equal to the square of the magnitude of coefficients. This encourages the model to maintain smaller, more evenly distributed weights, preventing any single feature from dominating. For architectures such as ViT and CNNs, which often contain a large number of trainable parameters, L2 regularization is particularly important. These models are prone to overfitting when dealing with high-dimensional data like audio spectrograms. L2 regularization ensures a more balanced learning process, preventing the model from relying too heavily on any particular subset of features. Similarly, in RNNs and CNN-BiGRU models, L2 regularization helps control the complexity by managing the impact of each weight, an essential factor in processing the temporal dynamics of audio signals.

When dealing with diverse and complex datasets like ESC-50 and UrbanSound8K, which encompass a wide range of environmental sounds and urban noises, the risk of overfitting is significant. Models might learn to memorize specific sounds from the training set rather than generalizing to new sounds. Regularization techniques ensure that the models learn robust, generalizable features representative of the broader categories of sounds. The choice between L1

and L2 regularization (or a combination of both, known as Elastic Net) depends on the specific model architecture and the characteristics of the audio data. It requires careful experimentation and validation to determine the most effective approach for each unique task.

3.5 Data Preparation

K-fold cross-validation is a crucial statistical method in machine learning, particularly for complex tasks like urban sound classification. This method divides the data into 'k' subsets and iteratively uses one subset for testing and the rest for training. This approach is particularly beneficial in urban sound classification for several reasons.

Firstly, urban sound classification involves a diverse array of audio sources and environments. K-fold cross-validation ensures that a model's performance is tested across a wide range of data samples, providing a more accurate measure of its ability to generalize to new, unseen data. This is essential in urban environments where the variety and unpredictability of sound sources are high. Secondly, it helps in reducing the bias that might arise from unbalanced or non-representative training data. Urban sound datasets can have imbalances, such as certain sounds being more prevalent than others. By rotating the training and validation sets in k-fold cross-validation, the model is less likely to overfit to specific data characteristics and more likely to capture a broad range of audio features.

Moreover, k-fold cross-validation is invaluable for model tuning and selection. By evaluating various models or parameter configurations across multiple folds, it's possible to identify the most effective model that consistently performs well. This rigorous evaluation is crucial in urban sound classification, where the correct identification of sound types can be challenging due to overlapping frequencies and ambient noise. Another significant advantage of k-fold cross-validation is the reduction of evaluation variance. A single train-test split can lead to

variability in model performance due to the random nature of data division. By averaging performance across multiple folds, k-fold cross-validation provides a more stable and reliable estimate of the model's effectiveness in real-world scenarios.

In contexts where data is limited, which is often the case in urban sound datasets, k-fold cross-validation maximizes data utilization. Since each data point is used for both training and testing across different folds, it ensures that limited data is exploited to its fullest potential. This aspect is particularly beneficial for urban sound classification, where collecting and labeling diverse sound data can be challenging.

Additionally, k-fold cross-validation is instrumental in handling class imbalance, a common issue in urban sound environments where some sounds occur more frequently than others. It provides insights into how well a model performs across various classes, ensuring that the model is robust and effective in practical applications. Finally, for research and development purposes, k-fold cross-validation offers a standardized way of evaluating models. This standardization is crucial for benchmarking new models or techniques against existing ones, providing a fair and consistent comparison framework.

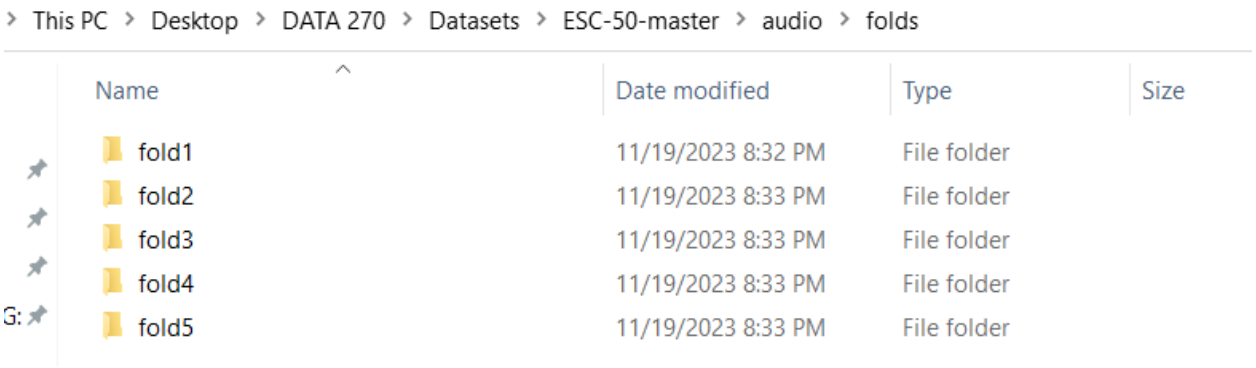
Both ESC-50 and Urbansound8k have folds mentioned in their respective metadata, the idea is to use this information and create the respective folds for the training, and validation purposes.

For ESC-50, from the metadata, there are 5 folds, a python script has been written and executed to create the folds respectively. The idea is to train the model using various folds combinations, for the first iteration it will consider Fold 1 to Fold 4, as training dataset, and Fold 5 as a validation dataset, for the next iteration, it will consider Fold 2 to Fold 5 as training, and Fold 1 as validation, and so on, till all the folds are used for training, and validation. Below

Figure 62 is the folder structure the 5-cross validation script creates for the ESC-50 dataset.

Figure 62

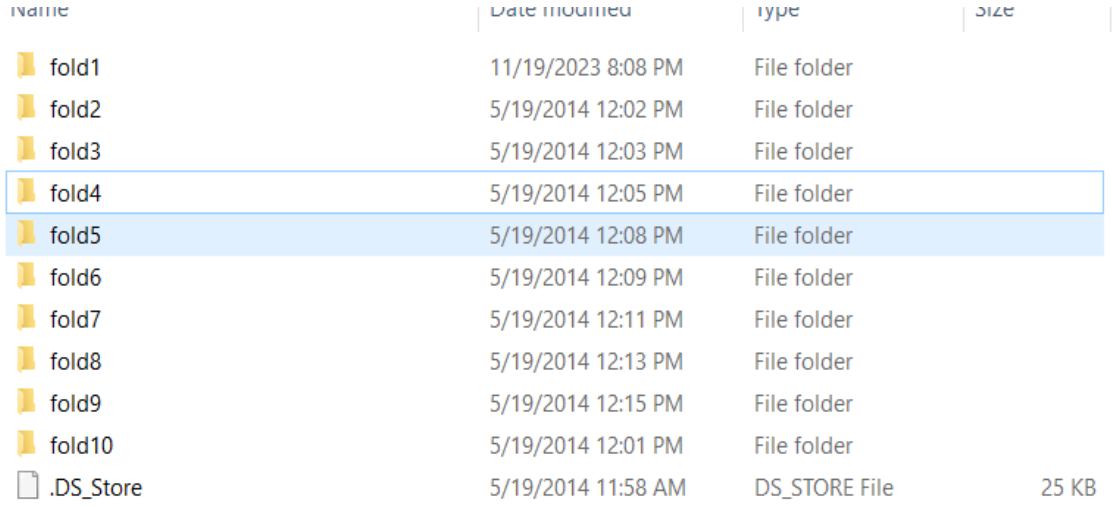
Folder Structure After Employing K-Cross Validation for ESC-50



Similarly, Urbansound8k has 10-fold Cross validation, it has 10 default folds as shown in the Figure 63, which are used to create the respective folder structure and are loaded.

Figure 63

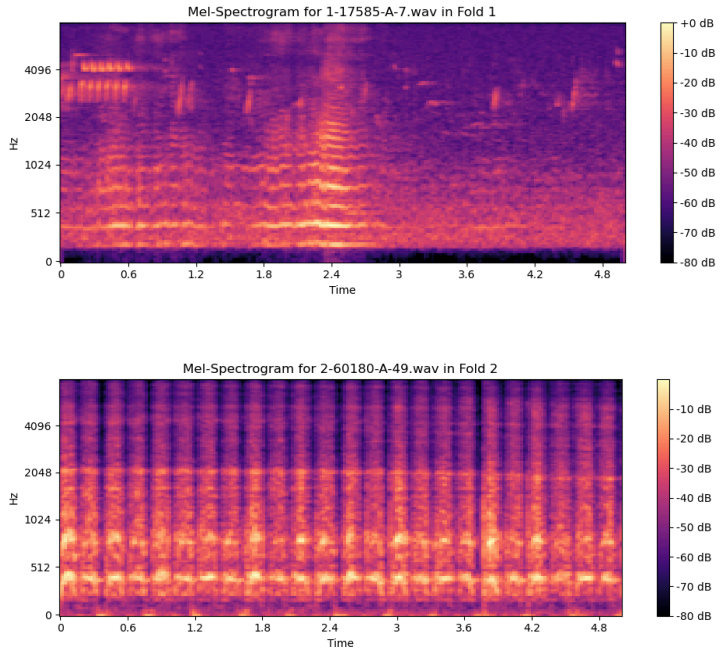
Folder Structure After Employing K-Cross Validation for Urbansound8k



For ESC-50, presenting some of the pre-processed augmented features reduced spectrograms from the few folds from respective datasets in Figure 64 below.

Figure 64

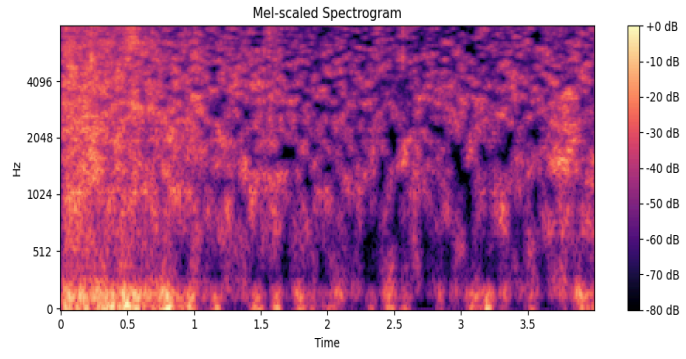
Mel-Scaled Augmented Spectrograms From Respective Folds



For Urbansound8k, presenting some of the pre-processed feature reduced spectrograms below in the Figure 65.

Figure 65

Mel-scaled Augmented Spectrograms from Respective Folds

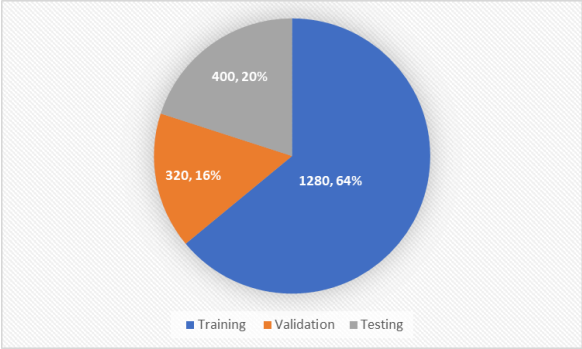


The below visualization in Figure 66 gives the split of the training, validation, and test splits for any given fold. For better interpretation, consider training on Fold 1 - 4, validation on

Fold 5, and testing on curated testing dataset by picking random files from each fold.

Figure 66

Number of Samples Per Training, Validation, and Testing Splits for ESC-50 After K-Cross Validation for Original Files

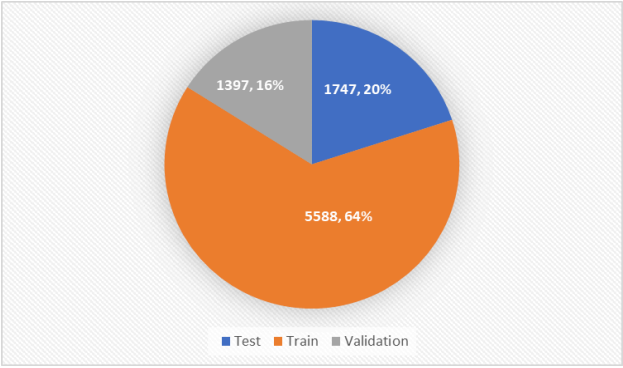


The splits are going to remain consistent for any combination of folds for training and validation, and 320 samples as new unseen data points after the k-cross validation to test the model for classification accuracy and other performance metrics.

For UrbanSound8k, a similar methodology is followed. Below is the split distribution which can be seen in Figure 67.

Figure 67

Number of Samples Per Training, Validation, and Testing Splits for Urbansound8k After K-cross Validation for Original Files



These splits make sure K-cross validation is followed, and at the same time makes sure there is no data leak, whatsoever, helping in improving the accuracy of classification, and improving the generalization in general

3.6 Data Statistics

As given in the work by Piczak et al. (2015) in ESC-50 compilation, and in the work of Salamon et al.(2014), the distribution of the samples across test, train and validation has been visualized above. While employing K-cross validation for UrbanSound8k (10 fold) & ESC-50 (5 fold), and at the same time preserving the test data as unseen data is very important to avoid any kind of data leaks, and improve the generalization of the model, thereby improving the accuracy, and other performance metrics.

To compare and understand the effectiveness of the pre-processing and various data engineering steps that have been used in this process, it's first important to make sure the distribution of the data is still intact, and represents the original dataset. Below are some of the visualizations to compare the statistics between transformed files.

Comparing the raw audio files to pre-processed audio files for Urbansound8k, as the major pre-processing has been done on this dataset. The below visualizations are box plots comparing three distinct audio features between raw and pre-processed audio files: RMS Energy, Spectral Centroid, and Zero Crossing Rate.

The RMS Energy box plot indicates that the pre-processed audio generally has a lower RMS energy level compared to the raw audio. The presence of outliers in the pre-processed group suggests that while most of the audio files had their energy level reduced, some retained higher levels or the noise reduction process introduced variability.

In the Spectral Centroid plot, there is a noticeable difference in the distributions. The raw

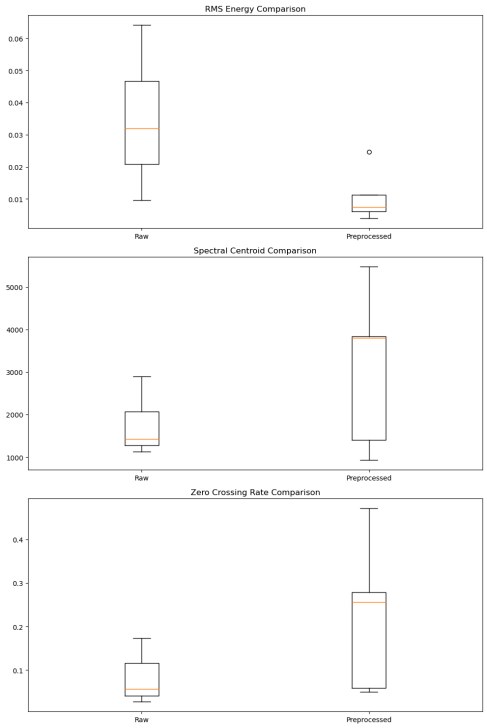
audio files have a wider interquartile range (IQR), suggesting more variability in the brightness of the sounds. The pre-processed files exhibit higher median spectral centroid values, which could indicate that the pre-processing emphasized higher frequencies or the noise reduction selectively attenuated lower frequencies.

Lastly, the Zero Crossing Rate plot shows that pre-processing has led to a higher median rate, with a significantly larger IQR. This might be a result of the noise reduction process, which can modify the waveform and create more zero-crossings, or it might reflect the removal of low-frequency components that contribute to fewer zero-crossings.

The above said all the three visualization plots (RMS Energy, Spectral Centroid comparison, Zero Crossing Rate box plots), comparison of statistical metrics of the raw vs. preprocessed audio files are shown in the below Figure 68.

Figure 68

Comparison of Statistical Metrics of Audio Files for Raw vs. Pre-Processed



The below Figure 69 shows the raw numerical metrics to compare the pre-processed results against raw audio files

Figure 69

Raw Numerical Metrics to Compare the Pre-processed Results Against Raw Audio Files

```
Raw RMS Energy:
Mean: 0.03466733
Median: 0.031922102
Mode: 0.06412702

Raw Spectral Centroid:
Mean: 1756.2737070597066
Median: 1427.121008572298
Mode: 1272.3225956379474

Raw Zero Crossing Rate:
Mean: 0.08269800405391581
Median: 0.05694310011061947
Mode: 0.04097046604046243

Preprocessed RMS Energy:
Mean: 0.01069781
Median: 0.0075226533
Mode: 0.0060871635

Preprocessed Spectral Centroid:
Mean: 3087.4492955746327
Median: 3797.931828608468
Mode: 1399.8023954912642

Preprocessed Zero Crossing Rate:
Mean: 0.2229322841401734
Median: 0.25612750632225434
Mode: 0.04920350885115607
```

The differences between these statistics is caused due to the specific pre-processing methods that have been employed. Noise removal, Click and pop sounds removal, and Min-max Normalization might have majorly tweaked the values, but the audio characteristics have been preserved for the respective files.

The next phase is to compare the statistics between the mel-scaled spectrograms and the augmented mel-scaled spectrograms, below are the respective results in the Figures 70 and 71.

Figure 70

Mean Comparison for Mel-Scaled Spectrograms Against Augmented Mel-Scaled Spectrograms

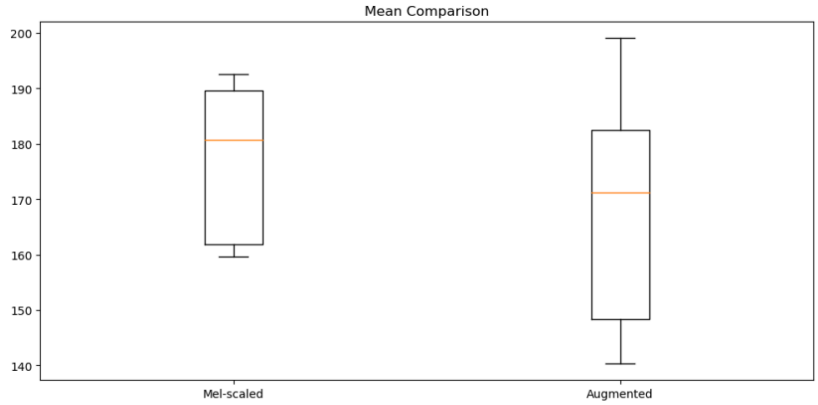
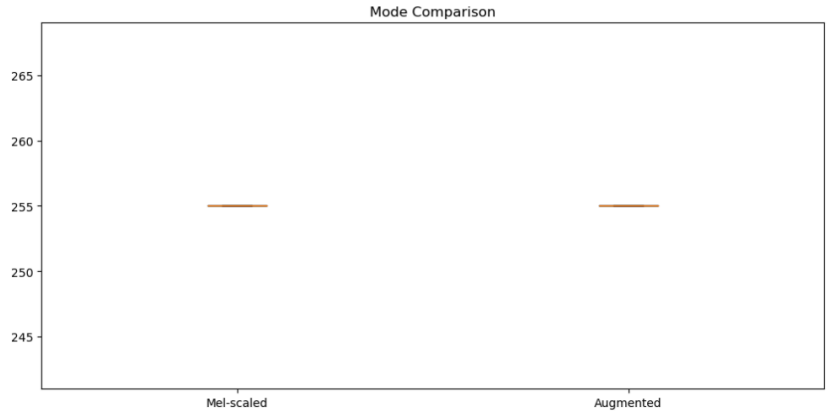


Figure 71

Mode Comparison for Mel-Scaled Spectrograms Against Augmented Mel-Scaled



From the above visualizations, the statistical measures have not been changed rapidly, this confirms the fact that the augmentation was a fruitful phase for this research project.

Similarly, if we check the respective augmented mel-scaled spectrograms from the respective folds, below Figure 72 is a comparison of the metrics from different folds and Figure 73 shows the distribution of augmented and original audio files.

Figure 72

Statistical Comparison for Mel-scaled Augmented Mel-Scaled Across Folds After k-Cross

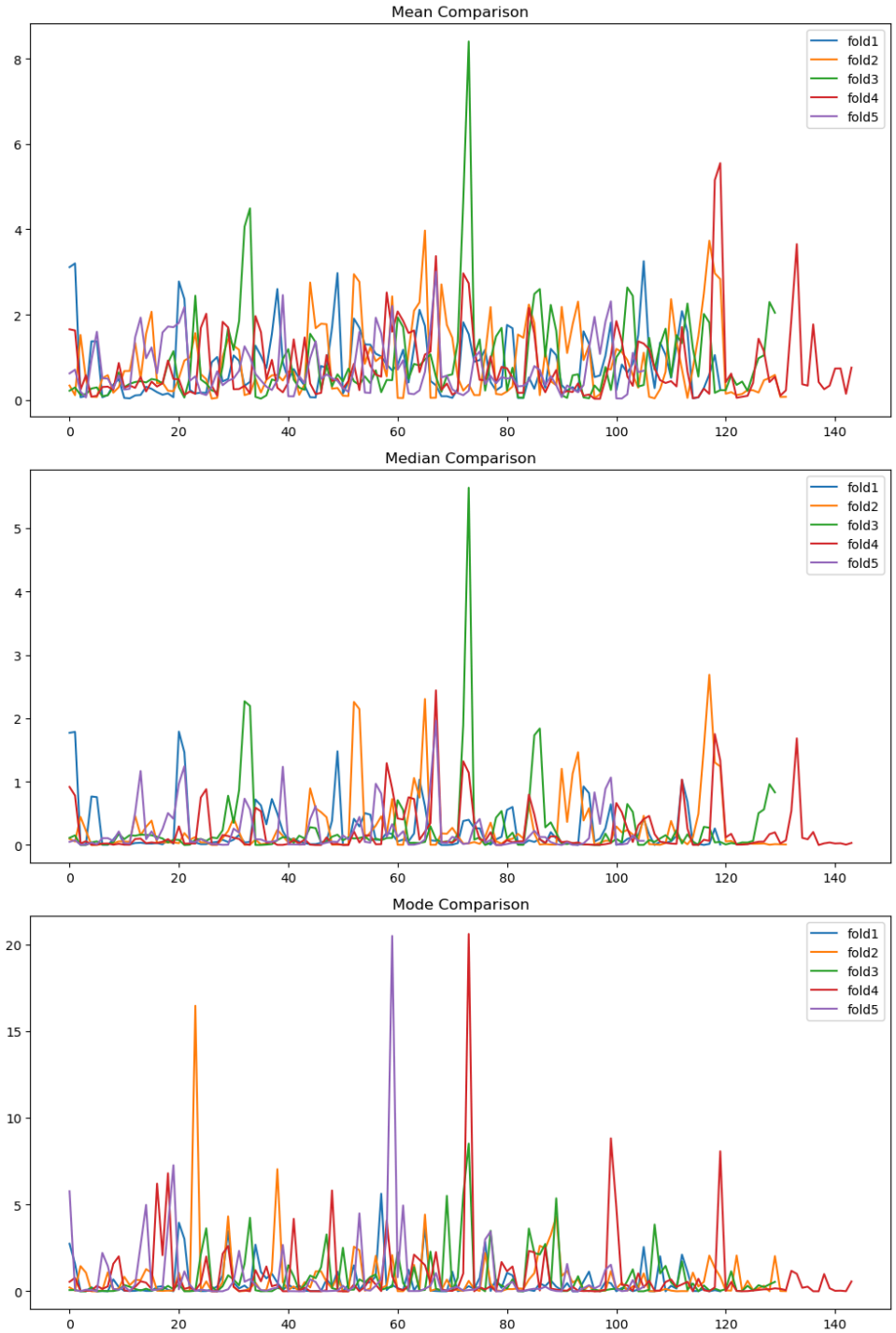
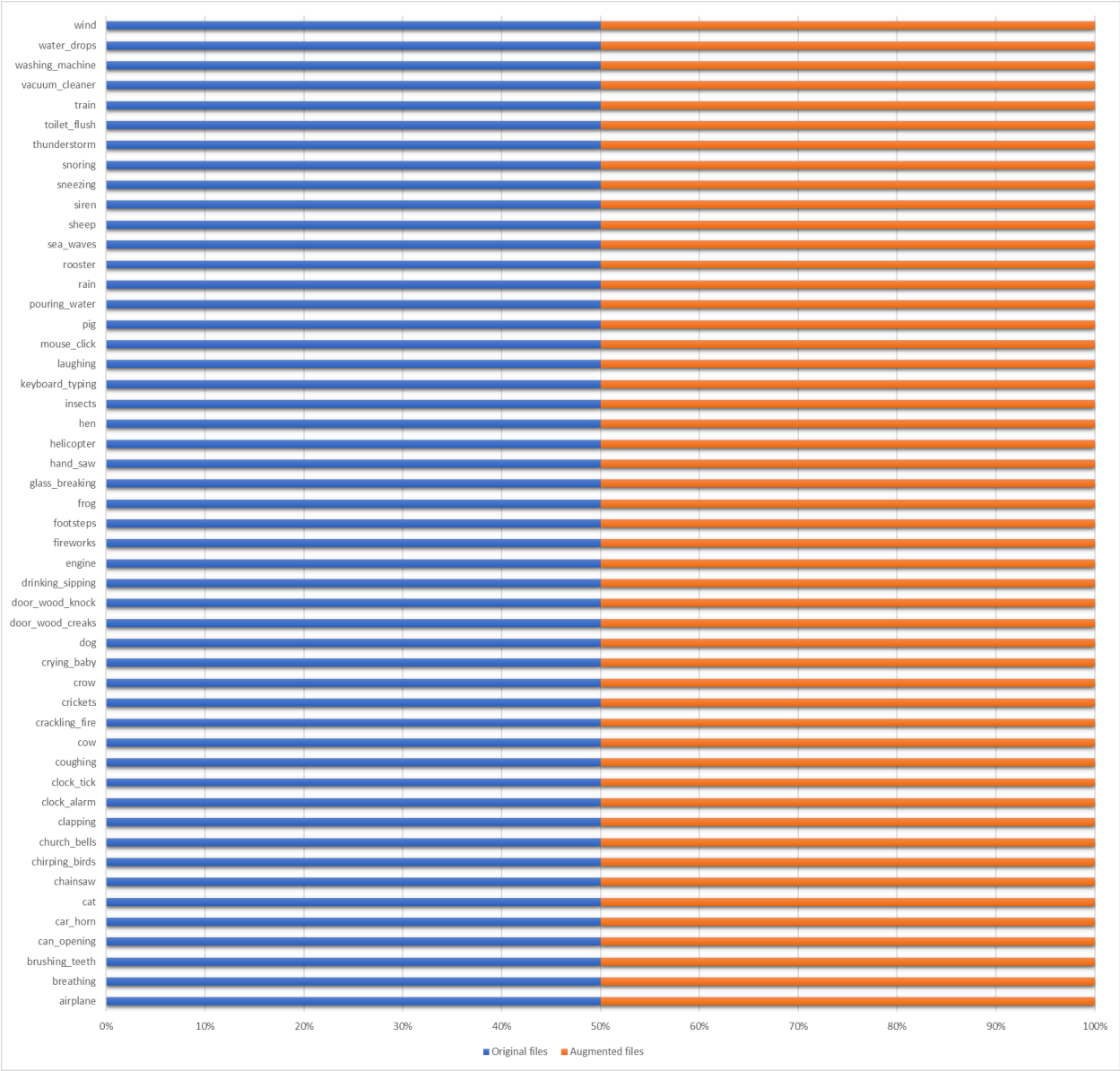


Figure 73

Distribution of the Augmented Files and Original Audio Files From ESC-50 Dataset

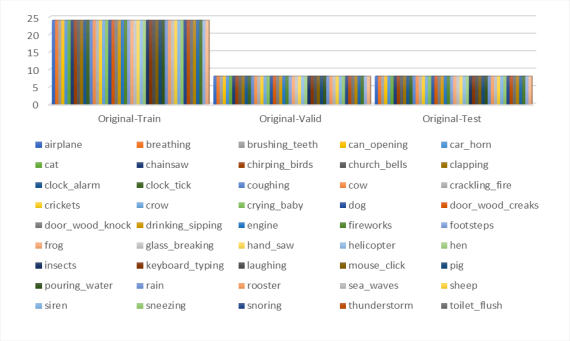


The above visualization shows the distribution of the original audio files from ESC-50, and the number of Augmented files for all the classes, this was done symmetrically, as the team had to make sure to apply suitable data augmentation techniques like SpecAugment (Time masking, Frequency Masking) along with Noise inclusion, and Time Stretching.

The below Figure 74 is the split of the UrbanSound8k for the original spectrograms used for Modeling purposes, the splits had to be even across classes, as the idea is to avoid overfitting for any particular class as well.

Figure 74

Split of the UrbanSound8k



As the augmentation is applied on the whole pre-processed datasets of UrbanSound8k and ESC-50 datasets, the splitting is done post that, so the below Figure 75 shows the augmented files division for UrbanSound8k across train, validation and test splits, just for the sake of the record, so that when there is a need to understand how the model performed on augmented data, it would be easier to pull the classification accuracies and other performance metrics quickly. Below Table 12 shows the split of original and augmented files.

Figure 75

Augmented Files Division for UrbanSound8k Across Train, Validation, and Test Splits

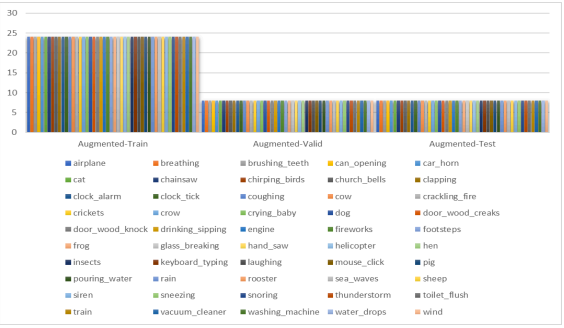


Table 12

Split of Original Files and Augmented Files Across Respective Folds of ESC-50 Dataset

Fold	Original Files	Augmented Files
1	400	1600
2	400	1600
3	400	1600
4	400	1600
5	400	1600

As it is given and mentioned in this chapter, that for Modeling the data, it is mandatory to employ k-cross validation techniques for the respective datasets. Below is the split original audio dataset Figure 76 and augmented audio data Figure 77 that was created for the Urbansound8k dataset also the fold wise comparison in Figure 78.

Figure 76

Split of Original Data Points From Urbansound8k

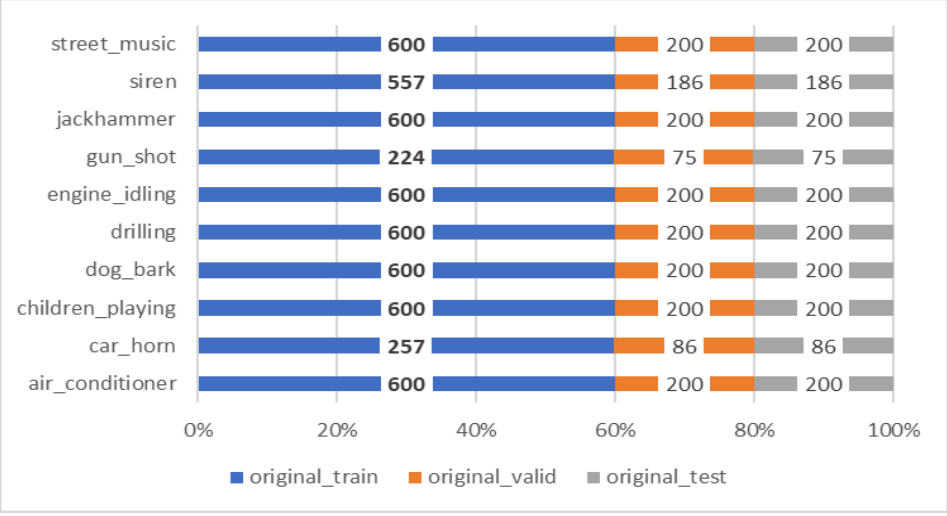


Figure 77

Split of Augmented Data in the UrbanSound8k per fold

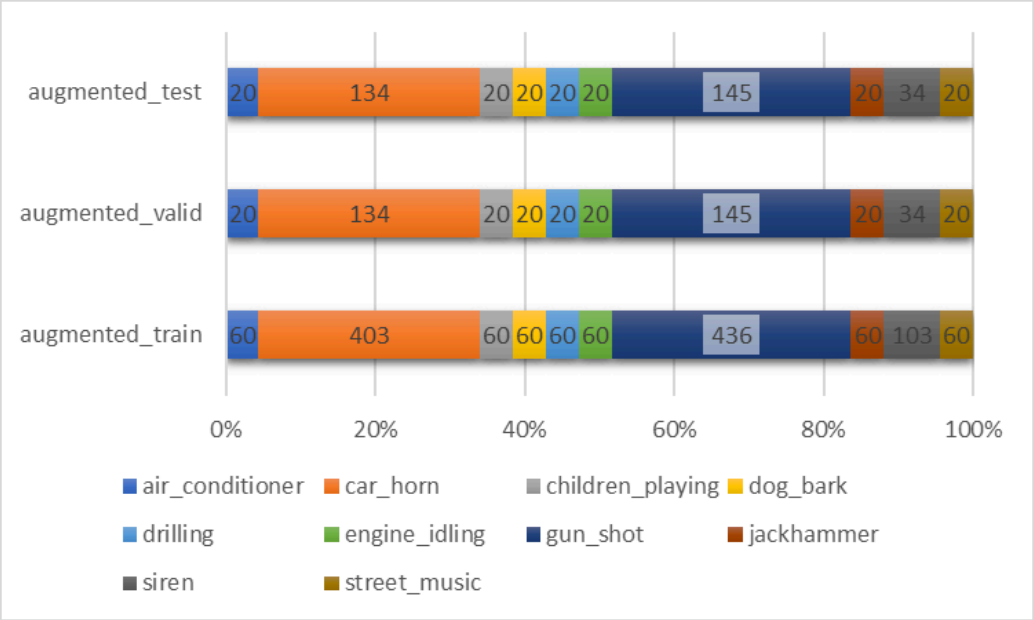
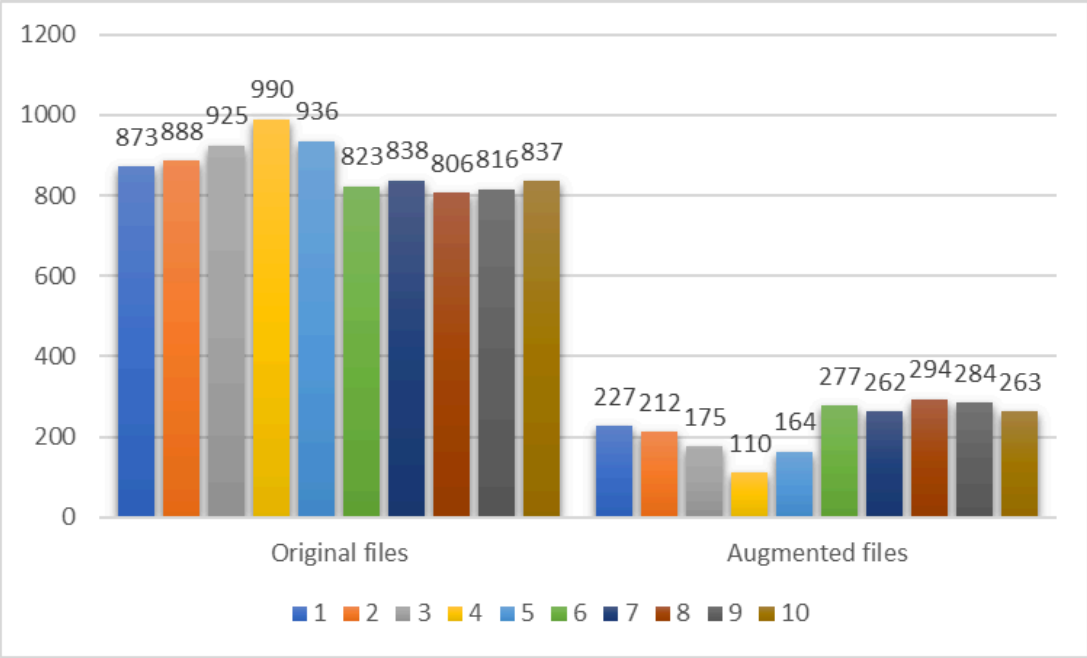


Figure 78

Fold Wise Split of the Augmented vs. Original Data From Urbansound8k per fold

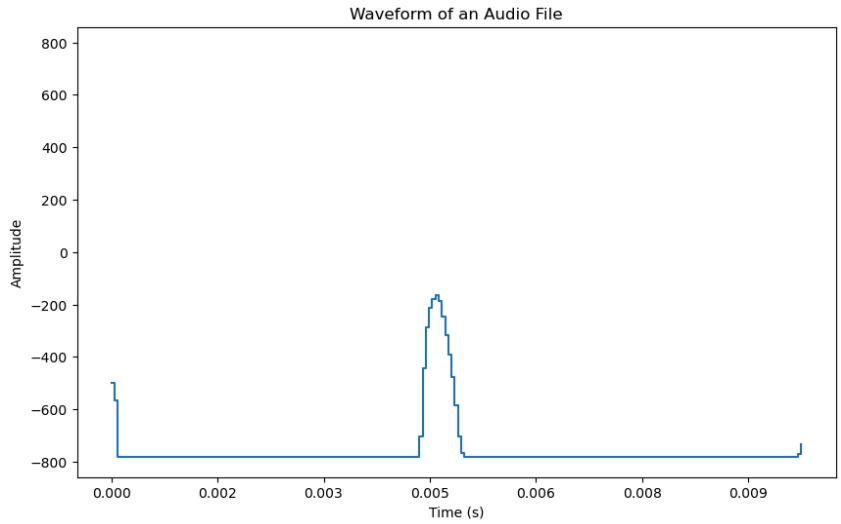


3.7 Data Analytics

There have been various EDA done to understand the audio files, some of these are displayed below Figure 79 for the ESC-50 dataset.

Figure 79

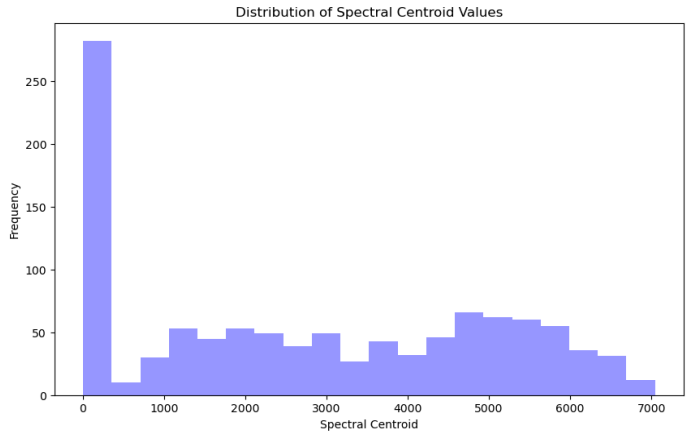
Waveform of the Audio Sample Audio Files



The above visualization illustrates the amplitude variation of an audio signal over a brief period, with time marked along the x-axis in seconds and amplitude on the y-axis. The prominent peak around the 0.005-second mark suggests a significant audio event, possibly a loud, isolated sound like a clap, beep, or snap within the recording. Given the ESC-50 dataset's focus on environmental sounds, this peak could correspond to a specific environmental event intended for classification. The waveform indicates a clear structure, which might imply a short, distinctive sound commonly used for testing audio classification algorithms. The rest of the waveform shows little variation in amplitude, suggesting a lack of other dominant sounds before and after the peak event. Figure 80 below shows the Distribution of the Spectral Centroid Values for a Sample Audio file.

Figure 80

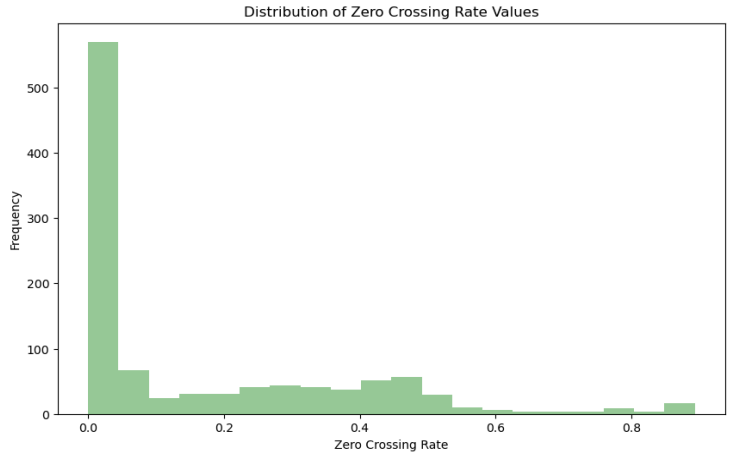
Distribution of the Spectral Centroid Values for a Sample Audio File



The above histogram visualizes the distribution of spectral centroid values for a collection of sounds, with a pronounced skew towards lower frequencies. This suggests a prevalence of sounds with lower pitches or less brightness, characteristic of environmental noises or non-musical audio within the dataset. Higher frequency sounds are less common, indicating a dataset with a darker sound profile.The below Figure 81 shows the distribution of Zero Cross Rate.

Figure 81

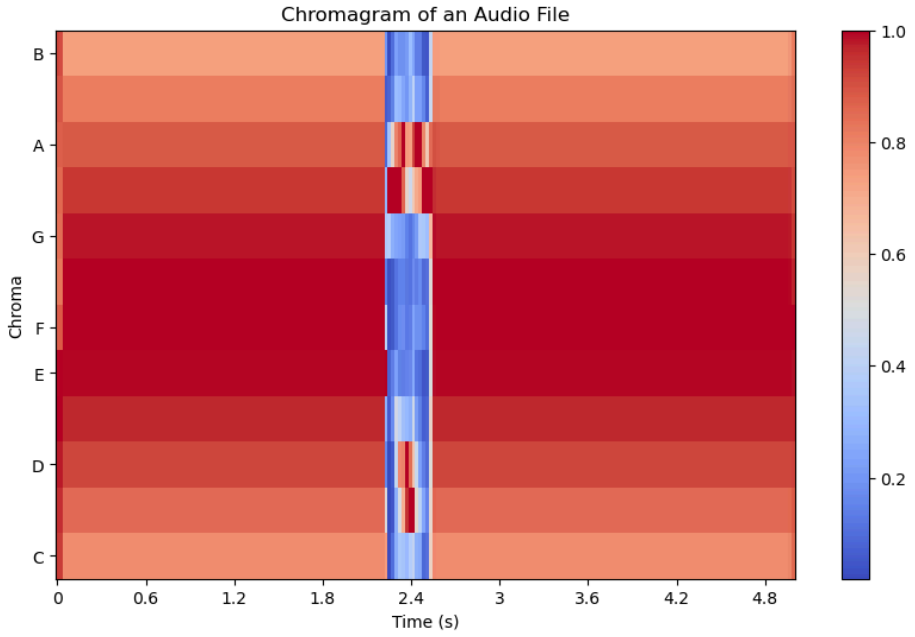
Distribution of Zero Cross Rate for a Sample Audio File



The histogram shows the distribution of zero crossing rate values, heavily skewed towards lower rates. This indicates that the audio files in the dataset predominantly contain smoother, less complex sounds with fewer frequency changes, such as sustained tones or hums, rather than noisy, high-frequency, or percussive sounds. The below Figure 82 shows the chromagram of a audio file

Figure 82

Chromagram of a Sample Audio File



The chromagram displays the intensity of different musical notes over time in an audio file. The strong vertical bands, particularly in the regions corresponding to certain notes, indicate moments where these notes are played with high intensity. This suggests the presence of distinct, possibly repetitive musical elements or motifs. The concentration of energy in specific chromatic bands also points to a harmonic structure, possibly indicating music with a focus on particular chords or a melody line centered around these notes. The below Figure 83 shows the Frequency

Distribution Over Time for a Sample Spectogram File and 3D Spectogram in Figure 84 and correlation matrix in Figure 85 for PCA Reduced Features.

Figure 83

Frequency Distribution Over Time for a Sample Spectrogram File from ESC50

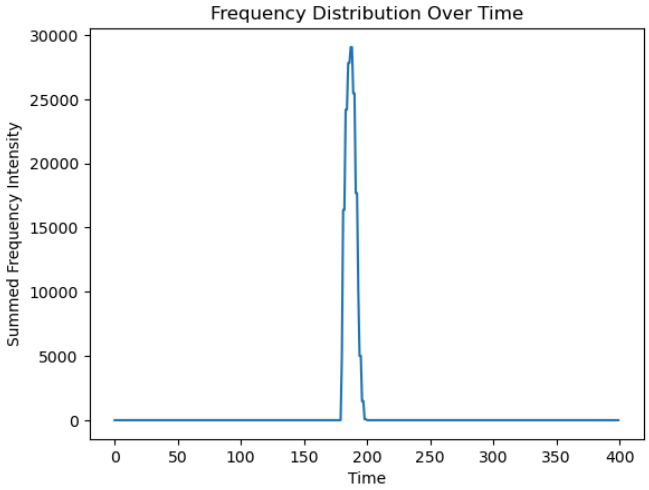


Figure 84

3-Dimensional Spectrogram for a Sample in ESC50 Showing Frequency, Time and Amplitude

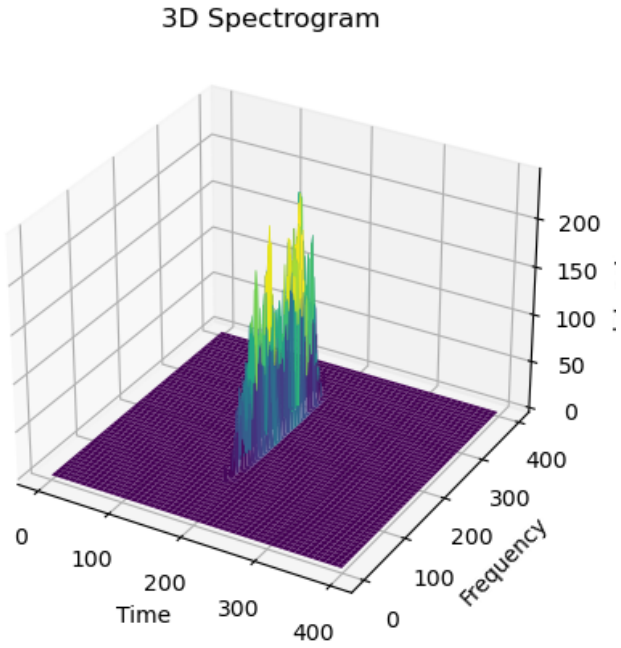
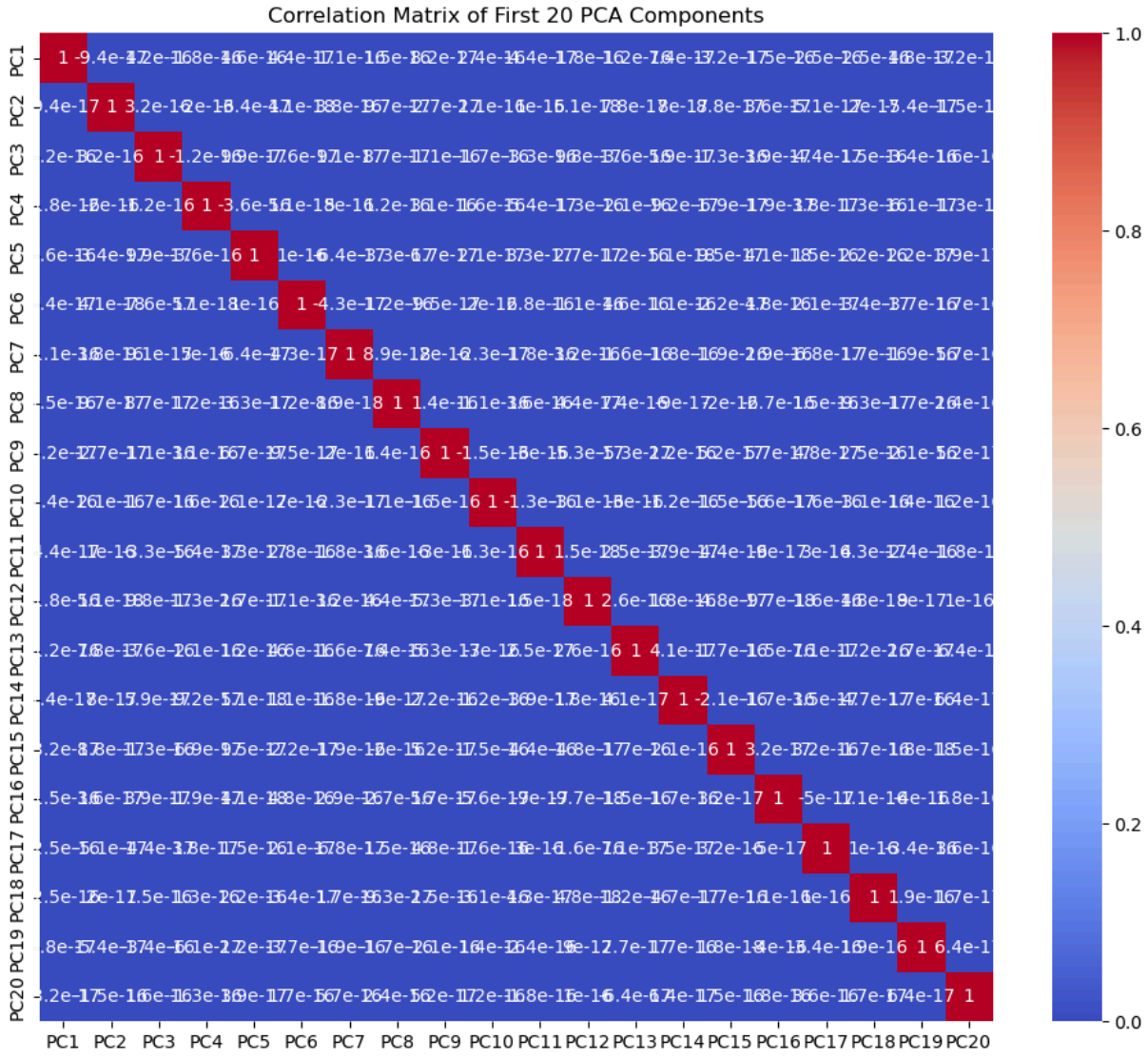


Figure 85

Correlation Matrix for the PCA Reduced Features (First 20)



Here is the heatmap visualizing the correlation matrix for the first 20 PCA components. As expected with PCA, the components are orthogonal to each other, meaning they should be uncorrelated, which is typically reflected as zeros (or close to zero) off the diagonal in the correlation matrix heatmap. The diagonal elements represent the correlation of each principal component with itself, which is always one.

4 Modeling

4.1 Model Proposals

Following the data engineering phase where research ideas from the study conducted by Simona Domazetovska et al.(2021) comprehensively explains the process of deriving features such as Mel Frequency Cepstral Coefficients (MFCCs), Chromagrams, and Mel-scaled spectrograms from audio files, In the work presented by Park et al. (2019) a new Spectrograms Augmentation technique called SpecAugment is introduced which banks on Time masking and frequency masking. Other augmentation techniques like white noise addition (based on 1.5 seconds of the audio), and time shifting have also been employed to create diverse augmented spectrograms datasets from the mel-scaled spectrograms which were generated from pre-processed audio files of ESC-50 and UrbanSound8k. As the primary mode of input features for this research project is Images i.e. mel-scaled spectrograms, the base behind choosing the deep learning architectures emerged from the idea of Image classification models. This idea was showcased in numerous research papers such as the work done by Piczak (2015) where the author introduced ESC-50 as a dataset to enable the research in the Urban audio classification, and used CNN for the classifier purposes, the accuracies achieved were ~60% for the baseline CNN model, meaning CNN and its variants are a good method to classify the Audio data, this is also proved in the work by Gong et al. (2021), when they achieved accuracies close to ~96% by implementing Vision transformers architecture on the ESC-50 dataset. Moreover, the work of Salamon et al. (2017), which utilized the UrbanSound8k dataset, advanced the use of CNN techniques, focusing on data augmentation and deep learning to enhance sound classification accuracy.

The evolution from traditional Convolutional Neural Networks (CNNs) to CNN + LSTM

marks a significant advancement in urban audio classification. CNN + LSTM integrate the spatial feature extraction capabilities of CNNs with the sequential data processing strength of Recurrent Neural Networks (RNNs). This combination allows CNN + LSTM to capture not just the immediate acoustic features but also their temporal dynamics, a crucial aspect when dealing with the sequential nature of urban sounds. Abdel-Hamid et al. (2014) demonstrated the efficacy of CNN + LSTM in speech recognition, highlighting their ability to model temporal dependencies in audio data. Similarly, Toth (2014) explored the use of CNN+LSTM for phone recognition, further establishing their utility in audio processing tasks. These studies underscore the potential of CNN + LSTM in urban audio classification, where understanding both the spatial and temporal aspects of sound is vital.

Parallelly, the integration of CNN with Bidirectional Gated Recurrent Units (CNN+BiGRU) represents another innovative step forward from the foundational CNN architecture. By combining the spatial analysis prowess of CNNs with the forward and backward temporal processing capabilities of BiGRUs, this architecture offers a comprehensive analysis of audio data. Sainath et al. (2015) were pioneers in blending CNNs with RNNs for speech recognition, demonstrating how such an amalgamation enhances the model's ability to capture a wide range of acoustic features and their temporal evolution. This aspect is particularly beneficial for urban audio classification, where the chronological context of sounds can play a critical role in accurate categorization.

With the emergence of Transformers architecture, introduced and briefed in the popular work by Vaswani et al. (2017), this groundbreaking work of Dosovitskiy et al. (2020) in introducing ViT for image recognition laid the foundation for its application in audio classification. Their research demonstrated that transformers, previously renowned in the field of

natural language processing, could also excel in interpreting visual data. By extension, this revelation opened up new possibilities for audio analysis, particularly in the context of urban sound classification where the diversity and complexity of sounds require a model capable of understanding broad patterns and relationships. Furthermore, the exploratory study by Gong et al. (2021) into the application of ViT for audio tagging tasks sheds light on the model's adaptability and effectiveness in handling audio data. These studies collectively validate the potential of Vision Transformers in urban audio classification, highlighting their ability to provide a more comprehensive understanding of soundscapes compared to traditional CNN-based approaches.

CNN + LSTM

The CNN + LSTM architecture for urban audio classification ingeniously marries the spatial feature extraction of convolutional layers with the sequential data processing capability of recurrent layers. As discussed in the work by Zhao et al. (2019) Each convolutional layer applies a set of learnable filters to the input spectrogram, extracting various acoustic features. Mathematically, the convolution operation within these layers is defined in the Equation 1 as

$$F_{ij}^{(l)} = \mathrm{ReLU}(W^{(l)} * X^{(l-1)} + b^{(l)}) \quad (1)$$

where $W^{(l)}$ represents the weight, $X^{(l-1)}$ is the input, $b^{(l)}$ is the bias, and ReLu introduces the non-linearity required. Post convolution, as Hershey et al. (2017) suggested, the max-pooling layers reduce the feature maps spatial dimensions to lower computational requirements and bolster model generalization defined in the Equation 2.

$$P_{ij}^{(l)} = \max_{a\in[i,i+k],b\in[j,j+k]} F_{ab}^{(l)} \quad (2)$$

This operation selects the maximum value within a defined window k of the feature map, effectively condensing the information. Subsequently, the feature maps are flattened and fed sequentially into recurrent layers, such as Long Short-Term Memory (LSTM) units, which process the temporal information. LSTMs are designed to mitigate the vanishing gradient problem, enabling the network to learn from data points that are temporally distant from each other. This is briefly described in the work by Etienne et al. (2018). It explains that the update of the hidden state in LSTM is governed by Equation 3 below.

$$H_t = \mathrm{LSTM}(H_{t-1}, F_t, \Theta) \tag{3}$$

With $H_{(t-1)}$ being previous hidden state and F_t representing the input feature at time t

The culmination of the process is explained by McFee et al. (2018) where the authors passed the recurrent layer output and then passed through a fully connected layer that serves as the classification head. A softmax activation function is applied to the output of this layer to yield a probability distribution over the various urban sound classes, as depicted by Equation 4.

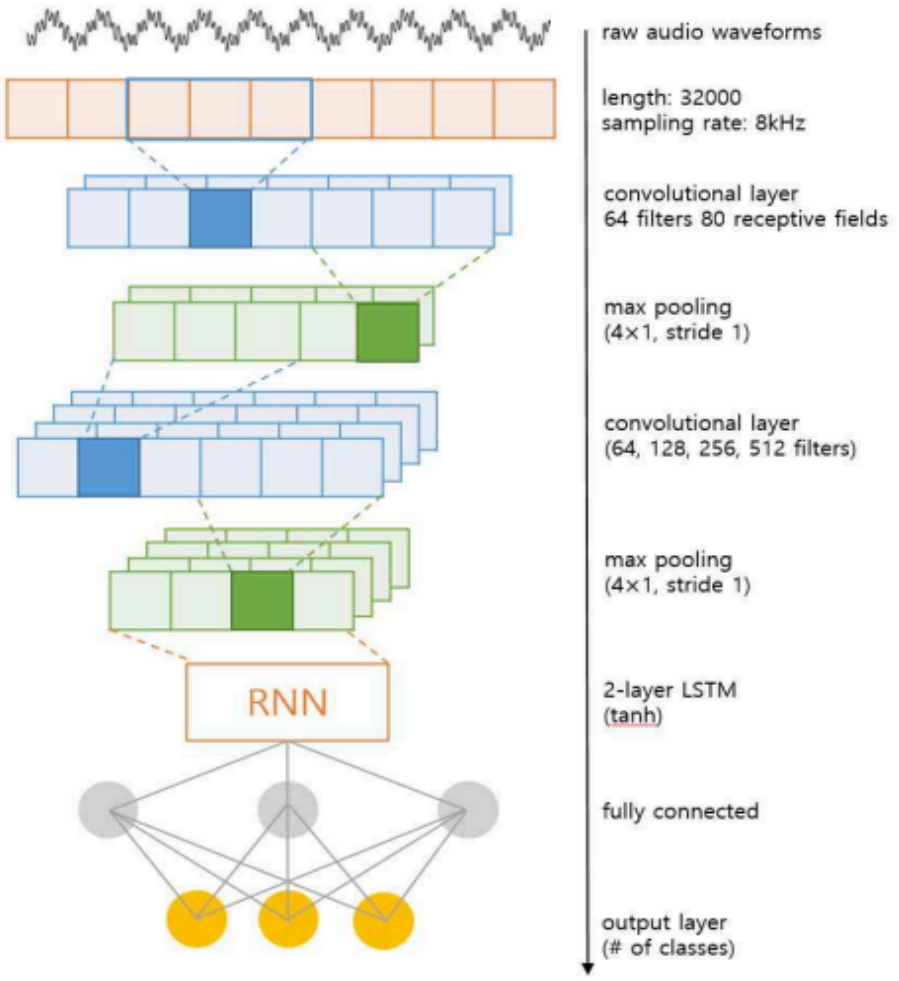
$$Y = \mathrm{softmax}(W_h H_T + b_h) \tag{4}$$

This CNN + LSTM architecture is effective for Urban Audio Classification tasks due to its dual focus on spatial feature extraction and temporal sequence modeling, providing a nuanced analysis of urban sounds.

The below Figure 86 is referred from Sang et al. (2018), it describes how the step by step mathematics which powers the Recurrent Convolutional Neural Network is transformed into an architecture.

Figure 86

Proposed Method of Convolutional Recurrent Neural Network for Urban Environmental Sound Classification



Note. Referred from the work of Sang et al. (2018)

The step by step approach is further clearly showcased in the below pseudocode for the Recurrent-CNN architecture that has been developed for the Urban Audio Classification task. Table 13 below shows the pseudocode for CNN+LSTM Model.

Table 13

Step by Step Pseudocode for the CNN + LSTM Model

Pseudocode Description of the CNN + LSTM Algorithm for Urban Audio Classification

```
Require: Set of pre-processed spectrogram images S
Require: Number of convolutional layers n_conv
Require: Number of recurrent layers n_recur
Require: Number of classes for classification n_classes

# Begin with input spectrogram
for each spectrogram in S do:
  # Apply convolutional layers
  for layer in range(1, n_conv) do:
    Apply convolution operation with ReLU activation
    Apply pooling to reduce the dimensionality
  end for

  # Flatten the output of convolutional layers
  Flatten the final pooled feature map to create a feature vector

  # Feed the flattened feature vector into recurrent layers
  Initialize the hidden state H for the recurrent layer
  for each time step t in feature vector do:
    Update H using the recurrent layer (e.g., LSTM or GRU)
  end for

  # Classification head
  Initialize a classification vector with zeros for each class
  for class in range(1, n_classes) do:
    Calculate the score for each class using the final hidden state
    Apply the softmax function to get probabilities
  end for

  # Assign the class with the highest probability as the predicted class
  Predicted class <- argmax of classification vector
end for

return Predicted class for each spectrogram in S
```

This CNN+LSTM architecture is effective for Urban Audio Classification tasks due to its dual focus on spatial feature extraction and temporal sequence modeling, providing a nuanced analysis of urban sounds. The CNN+LSTM architecture explained in the above pseudocode can

be optimized using hyperparameters starting with the number of convolutional filters, defaults often begin at 32 or 64 for initial layers, and it's common to increase this number in deeper layers to capture more complex features. However, too many filters may lead to overfitting, especially with limited training data. Kernel size, usually set to 3x3 or 5x5 in many models, determines the scope of feature extraction. Smaller kernels are adept at capturing fine details, while larger ones provide a broader view of the input data. Adjusting the kernel size can help the model focus on relevant features in audio signals. Stride, typically with a default value of 1, controls the step size of the convolutional filters across the input. Increasing the stride can reduce computational load by decreasing the spatial dimensions of the feature maps but may result in loss of detail. The number of recurrent layers and the number of LSTM units within these layers are set based on the complexity of the temporal patterns to be learned. Commonly, one starts with one or two layers, each having 128 or 256 units. More layers and units can model more intricate temporal information but increase the risk of overfitting and computational expense.Dropout rate, often between 0.2 and 0.5, is a regularization technique to prevent overfitting. The rate can be adjusted based on the model's performance on validation data; an increase may be warranted if overfitting is observed, while a decrease can be beneficial if the model is underfitting.

Learning rate is a critical parameter that affects the convergence speed of the model. A typical default value for the Adam optimizer is 0.001. If convergence is erratic, reducing the learning rate can help, whereas an increase might speed up convergence but risks overshooting the minimum loss. Batch size impacts the gradient estimation; defaults such as 32 or 64 are common starting points. A larger batch size may lead to faster training but requires more memory. A smaller batch size can act as a form of regularization and might lead to better generalization. Finally, the sequence length for recurrent layers should be set according to the

temporal scope necessary for the task. It must be long enough to capture the relevant temporal context without making the computational requirements infeasible.

ViT - Vision Transformers

The Vision Transformer (ViT), a deep learning model, was introduced by Dosovitskiy et al. (2020) through the influential paper "An Image is Worth 16*16 Words: Transformers for Image Recognition at Scale." This model extends the original Transformers architecture, a concept formulated by Vaswani et al. (2017), beyond its traditional application in processing sequential text data, demonstrating its adaptability. In the context of audio classification with ViT, prerequisite would be the mel-scale spectrogram for the audio files, which has been employed in another work by Gong et al. (2021) where the authors used 16*16 patches of the mel-scale spectrogram as input features for the ViT architecture,

The process involves dividing each spectrogram into uniform-sized patches (for instance, 16x16 pixels), which are then flattened into one-dimensional vectors. These vectors undergo a linear transformation to a higher-dimensional space, creating embedding vectors. This process is analogous to converting words in a sentence into embeddings in natural language processing tasks. As discussed by the work from Gong et al. (2021) and Dosovitskiy et al. (2020), The embedding E of each patch P is computed using the Equation 5 below.

$$E = WP + b \tag{5}$$

Note: where W, and b are considered as the trainable parameters of the ViT model

To preserve the order and spatial relationships within the spectrogram, positional encodings are added to these patch embeddings. These encodings, denoted as PE_i for the i^{th} patch, are crucial for the model to comprehend the placement of each patch, enabling it to

interpret the structure of the sound. The patch embedding with positional encoding is represented in the Equation 6 below.

$$E` = E_i + PE_i \tag{6}$$

Note: where PE_i is the positional encoding for the ith patch.

The core of the Vision Transformer is its encoder, comprising multiple layers of multi-head self-attention and feed-forward neural networks. The self-attention mechanism, crucial for the model's functionality is mentioned and explained in the following Equation 7.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{7}$$

Where Q, K, V are the query, key, and value matrices derived from the patch embeddings, and dk is the dimension of the key. This mechanism, which operates on the dimension dk of the key, allows the model to focus on and integrate information from different patches, considering the context of each within the whole spectrogram. As explained by Gong et al. (2021) and Dosovitskiy et al. (2020), The multi-headed attention feature of the Vision Transformer (ViT) is pivotal when it's adapted for audio classification tasks, particularly in the context of urban soundscapes. This mechanism allows the model to simultaneously focus on various aspects of an 'audio image', such as a spectrogram. For example, while one attention head might be concentrating on textural elements indicative of typical urban noises, another could be discerning the temporal features that distinguish different sounds, like speech versus music. Below formula for Multi-headed attention is directly referred from Vishwani et al. (2017)

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

Below Table 14 is a step by step pseudocode for the defined ViT architecture,

Table 14

Step by Step Pseudocode for the Proposed ViT Architecture

Pseudocode for Vision Transformer (ViT)

Require: An input image I

Require: Image size HxW, Patch size PxP, Number of patches N (where N = (H/P) * (W/P))

Require: A pre-trained transformer model with L layers, D-dimensional embeddings.

1: Divide the image I into N patches of size 16x16

2: Flatten each patch into a 1D vector of size 16*16*3 (assuming 3 color channels)

3: Project each flattened patch to a D-dimensional embedding using a trainable linear projection

4: Add positional embeddings to the patch embeddings to retain positional information

5: Append a learnable [CLS] token to the sequence of embedded patches

6: Pass the sequence of embeddings to the transformer encoder:
for each layer l in L:
a. Apply multi-headed self-attention to the sequence - No. of attention heads:16
b. Apply layer normalization
c. Apply a feed-forward neural network - ReLu activation network
d. Apply another layer normalization

7: Take the output corresponding to the [CLS] token from the final layer of the transformer

8: Pass the [CLS] token through a feed-forward neural network (classification head).

9: Apply softmax to get probabilities for each class

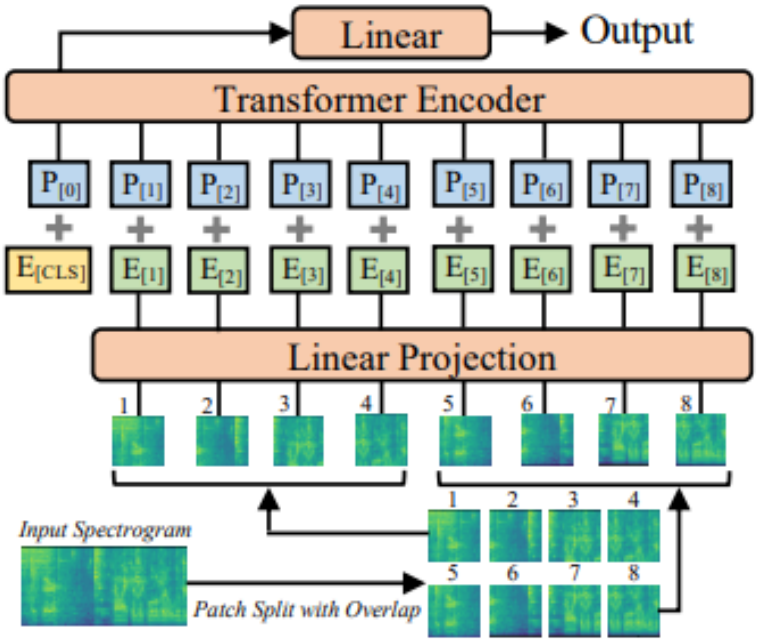
10: The predicted class is the one with the highest probability

Output: The predicted class for the input image I

The ViT architecture as proposed and implemented with Spectrograms in the work by Gong et al. (2021) explains how a spectrogram is converted into 16*16 patches and fed into the proposed ViT, this architecture is clearly shown and explained below Figure 87.

Figure 87

*Model Architecture of the Transformer Encoder and Linear Projection of the 16*16 Patches*



For Compute heavy architectures, the best way to optimize the deep learning model is to feed it diverse data and doing it in high volumes, So Data augmentation methods could be employed to diversify the existing labeled audio data.

To expedite the training process and enhance model performance, practitioners frequently utilize pre-trained models. These models, already trained on extensive image datasets like ImageNet, have developed an extensive understanding of various features. When applied to specific tasks such as urban audio classification, these pre-trained weights can be adapted through transfer learning. By fine-tuning on a more specialized, smaller dataset, training time is

reduced, and performance is enhanced due to the pre-acquired generalized feature set.

In the context of audio classification, the application of augmentations to spectrograms acts as a method of regularization, aiding in the model's ability to generalize more effectively. Common augmentation techniques include modifying the tempo, altering the pitch, introducing background noise, and adjusting the loudness. These methods enrich the training dataset's variety without necessitating additional labeled data.

Adjusting hyperparameters is crucial in optimizing Vision Transformers (ViTs). Important hyperparameters encompass the patch size, the depth of the transformer indicated by the number of layers, the quantity of attention heads, the dimensions of the feed-forward networks within the layers, and the learning rate. The size of the transformer's patches, which determine the number of patches in each spectrogram, plays a critical role in balancing performance and training efficiency.

Additional hyperparameters that can be tuned include the dropout rate, which helps prevent overfitting by randomly deactivating a fraction of the neurons during training, and the learning rate decay, which gradually reduces the learning rate over time to refine the training process towards the end. The choice of optimizer, such as Adam or SGD, also impacts the speed and stability of convergence. The batch size, which influences the gradient estimation and training speed, is another factor to consider. Larger batch sizes can accelerate training but might require more computational resources.

Finally, the inclusion of positional encoding is vital in maintaining the sequence order of the spectrogram patches, which influences how the model perceives temporal and spatial relationships within the audio data. Each of these hyperparameters must be carefully calibrated to

achieve an optimal balance between model accuracy, generalization, and computational efficiency.

CNN

Classifying environmental sounds is a challenging task due to their unstructured nature, lacking clear repeating patterns. Despite this, environmental audio exhibits strong spectral and temporal characteristics that dynamically change over time, requiring models that can effectively capture non-stationary properties. Multi-scale analyses and representation learning approaches are essential to disentangle patterns in the variability of environmental audio.

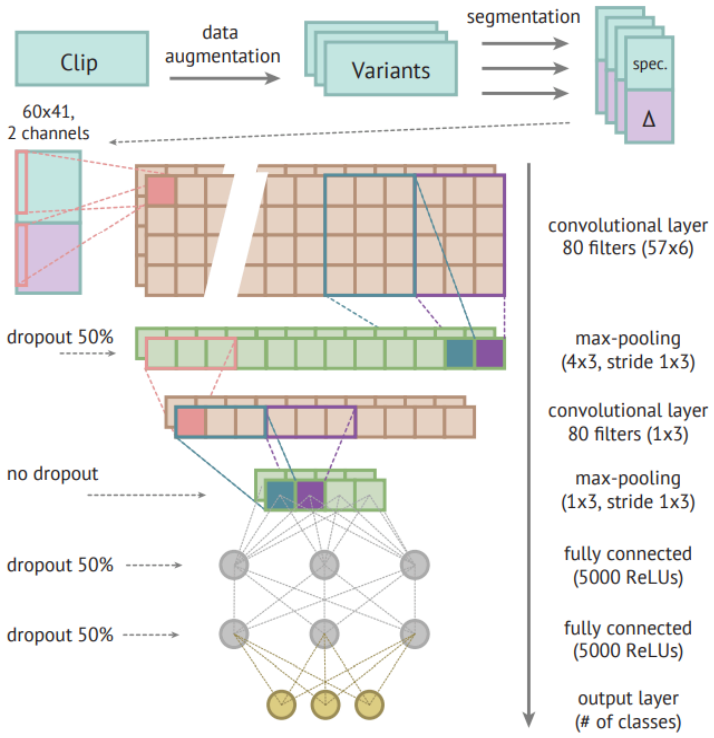
Various studies, such as those by Mkrtchian and Furletov (2022), Massoudi et al. (2021), and Gao et al. (2023), highlight the effectiveness of CNNs in accurately classifying environmental audio recordings. These studies emphasize CNNs' ability to learn rich features directly from audio spectrograms, especially when optimized for mel-spectrogram feature learning. Additionally, Piczak's (2015) CNN model architecture serves as a reference for building classification systems for environmental sounds, demonstrating state-of-the-art results in discriminative representation learning through careful tuning of hyperparameters and effective data augmentation.

In essence, the collective evidence suggests that CNNs, particularly when tailored for mel-spectrogram feature learning, provide a robust and efficient approach for the accurate classification of environmental sounds in audio recordings. These models exhibit versatility across various datasets and offer a promising framework for developing classifiers in the field of environmental sound analysis.

The below Figure 88 shows the CNN Architecture overview.

Figure 88

Overview of CNN Architecture



Note. This is the architecture that was referred from Piczak (2015) to come up with a CNN model.

The CNN model developed for audio classification in this project draws inspiration from established methodologies, ensuring a robust framework for analyzing intricate audio patterns within the UrbanSound8K and ESC-50 datasets. Specifically tailored to process pre-processed audio data represented as mel-scaled spectrograms, the input layer is set with dimensions (128, 128, 1), aligning with the shape of the spectrogram images.

At the model's core are its convolutional layers, applying filters to the input data to learn hierarchical features. The initial convolutional layer uses 24 filters of size 5x5, maintaining this structure in subsequent layers with 48 filters of the same size. Following each convolutional

layer, Rectified Linear Unit (ReLU) activation is applied to introduce non-linearity, enhancing the model's ability to capture intricate relationships in the data (Piczak, 2015). Max-pooling layers strategically reduce spatial dimensions, focusing on salient features in the audio data, and are positioned after the first and second convolutional layers, employing max-pooling with strides of (4, 2).

The Flatten layer acts as a bridge, transforming the multi-dimensional output into a vector suitable for further processing. Dropout layers, each with a rate of 0.5, are strategically introduced to prevent overfitting, enhancing the model's generalization capabilities (Srivastava et al., 2014). The model's dense layers, responsible for final classification, include a layer with 64 units and ReLU activation, followed by a final layer with 10 units and softmax activation, meeting the multi-class classification requirements. This architecture is optimized for processing Mel spectrograms, offering a comprehensive approach to audio classification.

ReLU Activation Function represented in the following Equation 8 :

$$f(x) = \max(0, x) \tag{8}$$

Softmax Activation Function represented in the following Equation 9.

$$P(class = i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{9}$$

The ReLU activation function introduces non-linearity after each convolutional layer. The softmax activation function governs the final layer's output, providing probabilities for each class. Below is Table 15 showing pseudocode for the CNN model that is used in this project.

Table 15

Pseudocode for Convolutional Neural Network

Step	Operation
Step - 1	Initialize the input layer with image I
Step - 2	For each layer l in L:
	a. If layer l is convolutional:
	Apply convolution operation with learned filters to the input
	Add bias to the convolution output
	Apply activation function (e.g., ReLU) to the result
	b. If layer l is pooling:
	Apply pooling operation (e.g., max pooling) to reduce dimensions
	c. If layer l is fully connected:
	Flatten the input if the first fully connected layer after conv/pool
	Apply matrix multiplication with learned weights
	Add bias to the output
	Apply activation function (e.g., ReLU) if not the final layer
Step - 3	Apply dropout if necessary to prevent overfitting
Step - 4	The last fully connected layer outputs logits for each class
Step - 5	Apply softmax function to logits to get class probabilities

Here is the summary of Table 15, This sequence of operations outlines the forward pass in a neural network during the inference phase. Initially, the input layer is initialized with the image (I). Subsequently, each layer in the network undergoes specific operations based on its type. Convolutional layers apply learned filters to the input, add bias, and activate using functions like ReLU. Pooling layers, such as max pooling, reduce dimensions. Fully connected layers involve flattening input (if after conv/pool), matrix multiplication with learned weights, adding bias, and optional activation (except for the final layer). Dropout is applied if needed to prevent overfitting. The last fully connected layer produces logits for each class, representing unnormalized scores. Finally, a softmax function is applied to these logits, transforming them into class probabilities. This stepwise process ensures the neural network processes input data, learns hierarchical features, and outputs probabilities for class membership.

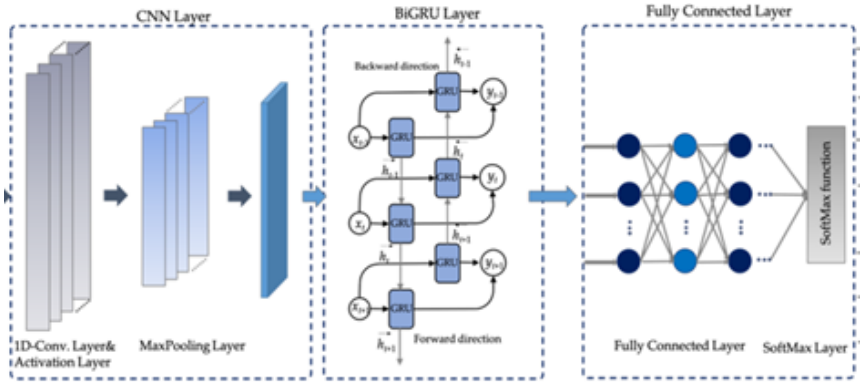
CNN-BiGRU-Attention Mechanism

In their recent study, Turab et al. (2022) recommended the utilization of Mel-Scaled Spectrograms and MelFrequency Cepstral Coefficients (MFCCs) as effective methods for feature extraction in inputting proposed Deep Learning Models. These transformations play a vital role in ensuring compatibility with the suggested deep learning architectures, establishing a solid groundwork for subsequent model development. According to Zeng et al. (2021), it is crucial to consider both temporal and spatial features for fully leveraging audio data. Xueli (2021) addressed this by proposing the CNN-BiGRU-Attention Model, specifically designed to extract the necessary features. In a complementary study, Yadav et al. (2023) supported the effectiveness of the CNN-BiGRU-Attention Model, employing MFCC for future extraction, particularly in mitigating overfitting by using synthetic data generated for audio classes with limited recordings. Additionally, Ashraf et al. (2023) introduced a similar model incorporating Mel Spectrogram for

feature extraction, comparing its accuracy with other state-of-the-art models and proves that Mel Spectogram is a best option for feature extraction. These diverse approaches contribute to the evolving landscape of effective feature extraction methodologies in the realm of deep learning for audio data processing. The below Figure 89 shows the CNN-BiGRU Architecture.

Figure 89

CNN-BiGRU Architecture



Note. CNN-BiGRU Architecture taken reference from Mekruksavanich et al. (2021)

Yadav et al. (2023) research explains the CNN-BiGRU with Attention Mechanism works on different layers. The convolution layer performs a convolution operation on an input image to generate an output. This involves adding bias to an intermediate convolved output and convolving it with a learnable kernel matrix, resulting in an output image. This process is akin to a dense layer but is specifically designed for images, offering the benefit of parameter sharing. The operation is mathematically represented by the Equation 1 below. Where Y specifies output image, X specifies input image, W specifies kernel weights, ⊗ symbol specifies convolution operation and b specifies the bias. The Equation 10 follows the CNN as follows and then followed by activation layer and max pool along with BiGRU Algorithm Pseudo code for working procedure of BiGRU in Table 1.

$$\mathbb{Y} = \mathbb{F}(\mathbb{X} \bigotimes \mathbb{W}) + b \tag{10}$$

In this research Yuyu et al. (2022), the rectified linear unit (ReLU) function is chosen as the activation function for the convolution layer. Serving as an unsaturated nonlinear function, ReLU is capable of expediting the convergence speed during the training process and significantly enhancing the performance of convolutional neural networks (CNNs). The definition of the ReLU function is presented in Equation 11, where the input variable x is subjected to the condition that if x is less than 0, the output is set to 0, and if x is greater than 0, the output is equal to the input. This choice of activation function contributes to the efficiency and improved performance of the CNN model. Below Table 16 shows the BiGRU algorithm that comes after CNN.

$$g(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{11}$$

Table 16

Pseudo Code for BiGRU Algorithm

Algorithm 1 represents the algorithm of the entire training process.

Algorithm 1: Attention-based Bi-GRU architecture for audio classification

```
hyper parameters ← manual selection
loss_function ← binary cross entropy
optimizer ← Adam
N ← numberOfEpochs
dataset ← loadDataset()
model ← buildModel(hyperparameters)
compileModel(model, loss_function, optimizer)
prevAccuracy ← 0
batch_size ← hyperparameters.get('batch_size')
While N > 0 do
    iterations ← (numOfSamples(dataset)/batch_size)
    While iterations ≠ 0 do
        batch ← createMiniBatch(dataset)
        Foreach input ∈ batch do
            mfcc ← getMFCC(input)
            loss + = train(model, input)
        endFor
        backpropagateLoss(model, loss)
        iterations ← iterations − 1
    endWhile
    N ← N − 1
    accuracy ← validate(model, getTestData(dataset))
    If accuracy > prevAccuracy do
        storeModelWeights(model)
        prevAccuracy ← accuracy
    endIf
endWhile
```

Note. Attention based Bi-GRU Algorithm by Yadav et al.(2023)

In the research by Yadav et al. (2023) proposes the algorithm details an attention-based architecture using Bidirectional Gated Recurrent Unit (Bi-GRU) tailored for audio classification. Initial steps involve manually selecting hyperparameters, specifying a binary cross-entropy but instead of this CNN-BiGRU model uses categorical cross-entropy loss function, and utilizing the Adam optimizer for efficient model optimization. Training spans a designated number of epochs (N), where the algorithm loads the audio dataset, constructs the Bi-GRU model with the defined hyperparameters, and iterates through mini-batches during training. Mel Frequency Cepstral Coefficients (MFCCs) are extracted for each input, and model updates occur through backpropagation based on accumulated loss and there is a slight variation to this algorithm is inclusion of Mel Spectrograms according to the research of Ashraf et al. (2023) to improve the accuracy on audio sounds. Post each epoch, the model undergoes validation on a separate test dataset, with weights stored if accuracy surpasses the prior best. The embedded attention mechanism enhances the model's focus on relevant audio features, making it adept for effective audio classification.

As per the research of Yadav et al.(2023), The attention mechanism is integrated into BiGRU for pinpointing pertinent segments within a sound clip. This mechanism empowers BiGRU to make informed decisions at specific time steps, disregarding irrelevant portions. Attention mechanism role lies in extracting discriminative information, augmenting the efficacy of CNN-LSTM/GRU-based architectures by concentrating on specific data segments. In sound classification, this proves crucial as the entire dataset doesn't uniformly contribute to representing a specific sound class. Enhancing traditional BiGRU model performance with reduced computational costs, the attention mechanism becomes an asset. BiGRU model aids in

generating a dense vector representing the output after attentive consideration of required timesteps, with the underlying Equation 12 governing the GRU mechanism articulated in the below.

$$\mathbf{h_i} = \mathrm{GRU}\,(\mathbf{s_i}), i \in [1, L] \tag{12}$$

The given Formula 12, denoted as **hi**, represents a column vector of hidden states corresponding to the provided input, where L denotes the number of cells in the GRU network, and **si** represents the input. Additionally, an attention mechanism is incorporated to capture the network's hidden states, as depicted in the provided equation. According to Yadav et.al(2023) Before the Softmax layer , The dense layer is a fundamental component in neural networks employed to alter the dimension of a 2D layer through specific calculation functions. In the context of sound classification, the dense layer serves as a layer of neurons with input weights determined by a linear function to produce an output. The function associated with a dense layer can be expressed in the Formula 13 as follows:

$$\mathbf{Y} = f(X \times w + b) \tag{13}$$

where **Y** denotes the output layer, X represents the input layer, f is the activation function, w denotes the weight matrix, and b represents the bias vector.

In a CNN-BiGRU model, the Fully Connected Layer is tasked with extracting high-level abstract features from the input data, while the subsequent Softmax layer transforms the model's output into probability distributions. This process enables the network to make final classifications or predictions across multiple classes. These layers collectively play a pivotal role in the neural network's decision-making process, with the Fully Connected Layer capturing intricate patterns, and Softmax layer providing the necessary probability-based outputs for effective classification.

4.2 Model Supports

Environment, Platform, and Tools

For efficient and effective data engineering and preprocessing of audio data, the Google Colab Integrated Development Environment (IDE) has been employed. This platform facilitates collaborative coding, visualization plotting, and other team-based activities. The training and evaluation of models are conducted on a system equipped with a minimum of 16 CPUs, NVIDIA GeForce RTX 40 series GPUs, and 64 GB RAM.

Python stands as a popular choice for implementing deep learning models, primarily due to its extensive range of libraries tailored for machine learning and deep learning tasks. The division of mel-scaled spectrograms into training and testing sets is accomplished through k-cross validation using the sklearn.model_selection library. For Principal Component Analysis (PCA), sklearn.decomposition is employed, while the hugging face's transformers library is utilized for Vision Transformer (ViT) architectures. Additionally, Librosa is used for SpecAugment augmentations and transforming audio datasets into Mel-scale. Noise reduction is achieved using the noisereduce library, focusing on diminishing noise based on the audio file's initial 1.5 seconds. Metrics are extracted using sklearn.metrics and visualized through matplotlib and seaborn. Pandas is extensively used for data manipulation and dataframe operations. For constructing the Convolutional Neural Network (CNN) layers in the CNN+BiGRU model, TensorFlow or PyTorch libraries are used. These libraries provide efficient functions for convolutional operations, pooling, and activation functions.

TensorFlow, along with its high-level Keras API, offers a robust platform for building and training complex neural network architectures. The Sequential model from tensorflow.keras.models is widely used to initiate the architecture of CNN, CNN+BiGRU, and

CNN+LSTM. It allows for the linear stacking of layers, making it straightforward to build the models layer by layer. For the CNN components, Conv2D, MaxPooling2D, Dense, Flatten, and Dropout layers from tensorflow.keras.layers are essential. They handle various tasks from convolutional operations, pooling to reduce dimensionality, to fully connecting layers for classification. In CNN+BiGRU and CNN+LSTM models, the Bidirectional wrapper and GRU or LSTM layers are pivotal for adding recurrent layers to the model. These layers enable the models to process sequential data effectively, capturing temporal dependencies in audio signals. The Adam optimizer from tensorflow.keras.optimizers is often the choice for model training, known for its efficiency in handling large datasets and complex architectures and overview in Table 17.

Table 17

Python Libraries Used Across the Implementation of This Project

Library	Library Class	Method	Usage
scikit-learn	sklearn.model_selection	train_test_split	Splitting the datasets into test-valid-train
	sklearn.model_selection	StratifiedKfold	Employing k-cross validation for respective folds for respective datasets
	sklearn.metrics	Confusion_matrix, accuracy_score, precision, f1, classification_report	These metrics are used for evaluating the ViT architecture, and are compared against other scores of other models.
	sklearn.preprocessing	StandardScaler, Normalizer	Used for normalizing the features post-augmentation and post-PCA

Note. Table 17 is continued on pages 154 and 155.

Library	Library Class	Method	Usage
Librosa	librosa.feature	spectrogram	Converting the audio into mel-spectrogram
	librosa.display		Displaying the mel-scale spectrogram which is created from an audio file
PyTorch	pytorch.torchaudio	FrequencyMasking, TimeMasking	This is used for SpecAugment, specific to timemasking and frequencymasking
HuggingFace	Transformers	TrainingArguments, Trainer	This library has the implemented and pre-trained ViT which were employed in this research
Pandas	Dataframe	Drop, shape, head, merge	Used for any kinds of dataframe manipulation.
Numpy	Random	seed	Used for basic numerical array operations while feature extraction
noisereduce	noisereduce		Reducing the noise based on the audio signal
Matplotlib & seaborn	pyplot		Plotting the evaluation metrics into visualizations which are highly informative
tensorflow	tensorflow.keras.models	Sequential	Create deep learning models layer by layer.
		Conv2D, MaxPooling2D	Used for convolutional and pooling layers in CNNs, extracting features from spectrograms.
	tensorflow.keras.layer	Dense, Flatten, Dropout	Dense for output layers, Flatten for reshaping data, and Dropout for regularization to prevent overfitting.

		Bidirectional, GRU, LSTM	Capturing temporal dependencies in audio signals.
	tensorflow.keras.optimizers	Adam	Optimizer for tuning, and enabling better training practices
Datasets	load_dataset		To load the dataset from the drive

The above set of python libraries were all instrumental and key in building the respective individual models, enabling the Urban Audio Classification using ESC-50 and UrbanSound8k.

Data Workflow and Architecture

In machine learning, particularly for urban audio classification using datasets like ESC-50 and UrbanSound8k, it's essential to include training, validation, and testing phases in the model's life cycle. Given the diverse nature of the labeled audio data in these datasets, implementing K-fold cross-validation - 5-fold for ESC-50 and 10-fold for UrbanSound8k - is highly recommended. This approach ensures the model is trained and validated across different data subsets, promoting better generalization to new, unseen data. K-fold cross-validation is especially valuable when dealing with limited training data, as it rotates training and validation sets to maximize data utilization.

In urban audio classification, data augmentation and feature reduction significantly function as hyperparameters. Techniques like adding noise, adjusting pitch and speed, or applying SpecAugment have a substantial impact on a model's capacity to generalize from limited training data. Especially for computationally intensive models like Vision Transformers (ViT), tuning these hyperparameters often requires consideration of resource constraints. To identify the most effective configurations, automated hyperparameter tuning approaches such as

Bayesian optimization, grid search, or random search are commonly utilized.

Additionally, the strategy of transfer learning, where a pre-existing model pre-trained on a large dataset is adapted for a specific task, introduces another dimension to hyperparameter tuning. Crucial decisions in this context include determining which layers of the model to unfreeze for retraining and the degree to which their weights should be adjusted. This approach is particularly beneficial in models like ViT, CNN, CNN+BiGRU, and CNN+LSTM used in audio classification tasks.

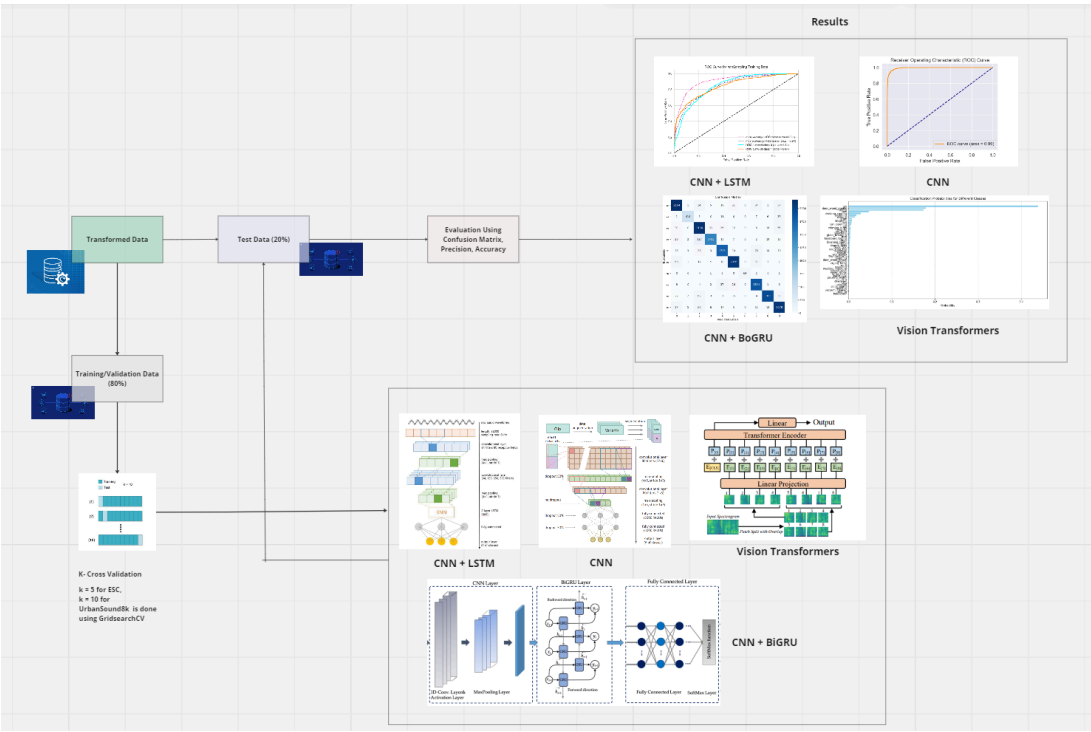
Starting with Vision Transformers (ViT), the choice of patch size is pivotal, as it dictates how the input, such as a spectrogram, is segmented. A smaller patch size leads to a higher number of tokens, potentially capturing more detailed information but at the expense of increased computational demand. The depth of the transformer, i.e., the number of layers, directly correlates with its ability to capture complex features, though it heightens the risk of overfitting and elevates computational requirements. The number of attention heads in ViT is also significant; more heads allow the model to concurrently focus on various aspects of the input, enhancing its feature-capturing diversity. However, this comes with a higher computational cost. The dimensions of the feed-forward networks within the transformer blocks influence the model's capacity to model complex functions, where larger networks can handle more complexity but require more computational power. Additionally, the learning rate and its scheduling play a crucial role in how effectively the model converges and performs.

In the case of CNN architectures, the number and size of convolutional filters are instrumental in defining the model's feature extraction capabilities. An increased number or size of filters can allow for more detailed feature extraction but may also lead to overfitting. The kernel size in convolutional layers influences the field of view for convolution operations, where

smaller kernels can detect finer details and larger kernels capture a broader perspective. The implementation of pooling layers, whether max or average pooling, affects the reduction of spatial dimensions, thus impacting the model's sensitivity to feature locations. The dropout rate is another critical hyperparameter in CNNs, serving as a regularizer to prevent overfitting, a vital consideration in larger and more complex network architectures. For CNN+BiGRU models, the number of units in the GRU layers is key to the model's efficiency in capturing temporal dependencies, with more units providing greater capacity for modeling complex dependencies. Figure 90 below shows how the data flow happens, how the transformed data is split into test, and train and other processes follow.

Figure 90

Dataflow Architecture of the Research Project



The bidirectionality in these models is crucial for understanding temporal sequences in audio, as it allows the model to process past and future context simultaneously. Furthermore, the

depth and complexity of the CNN layers preceding the GRU layer significantly affect the quality of feature extraction. Similarly, in CNN+LSTM models, the number of LSTM units determines the model's ability to learn and retain information over longer sequences, which is especially valuable in audio processing. The configuration and depth of the CNN layers preceding the LSTM layer play a significant role in the effectiveness of spatial feature extraction from the input data. Regularization techniques, including dropout and recurrent dropout, are crucial in both CNN and LSTM layers to mitigate the risk of overfitting. In all these architectures, universally relevant hyperparameters such as learning rate, batch size, and the number of training epochs also play an influential role. To identify the most effective configurations, automated hyperparameter tuning approaches such as Bayesian optimization, grid search, or random search are commonly utilized. Post hyper-parameter tuning, the model is tested on the best hyper parameters and the classification results are provided through various visualizations.

4.3 Model Comparison and Justification

The four models which were employed to work on this Urban Audio Classification problem, are Vision transformers, CNN+biGRU, CNN+LSTM, and CNN architectures. The input features for all these models are Mel-scaled spectrograms which include the augmented mel-spectrograms from both ESC-50 and UrbanSound8k datasets from their respective folds. These mel-spectrograms are generated from the audio files, pre-processed, and then converted into mel-scaled spectrograms, these mel-scaled spectrograms are then undergone a round of SpecAugment augmentation, as well as time shifting and white noise induction techniques to diversify the minimal labeled audio dataset from the respective Datasets.

ViT, as described by Dosovitskiy et al. (2020), employs a parallel processing architecture that adapts the transformer's self-attention mechanism for image-like data analysis, treating

spectrograms as images. While ViTs are highly capable of processing various data types, including audio, they tend to excel with larger datasets. However, they can struggle with smaller or sparser datasets, often leading to overfitting as explained in Rahman et al. (2022). The complexity of ViTs necessitates substantial preprocessing, such as patch extraction and positional encoding, demanding significant computational resources, typically requiring GPUs. The strength of ViTs lies in capturing global dependencies and complex patterns in data, but they are limited by high computational requirements and potentially lengthy training times.

The CNN+BiGRU architecture, explored by Yadav et al. (2023), merges the spatial feature extraction strength of CNNs with the sequential data processing capability of BiGRUs. This model is particularly effective for urban audio classification, handling both spatial and temporal features in audio data. The CNN component efficiently extracts textural patterns from spectrograms, crucial for distinguishing different urban sounds. Conversely, the BiGRU segment processes these features over time, capturing the temporal dynamics. The model, however, can be computationally demanding and carries a risk of overfitting, especially in the CNN layers.

CNN+LSTM, as per Sak et al. (2014), combines CNN's spatial feature extraction with LSTM's ability to model extended temporal sequences. This architecture is adept at managing both the spatial and temporal aspects of urban sound data. It shares the computational intensity and overfitting risks of CNN+BiGRU but is particularly effective at modeling longer temporal dependencies found in complex urban soundscapes.

CNNs, established by Piczak (2015) for audio signal analysis, are naturally suited for urban audio classification. Their parallel processing architecture is efficient for extracting spatial features from audio spectrograms. CNNs are less complex and faster to train than models like

ViT or CNN+BiGRU/LSTM, making them more suited for smaller datasets. However, their limitation lies in not capturing temporal dependencies as effectively as recurrent models.

Vision Transformers (ViT) are adept at handling large datasets and complex patterns in urban audio classification but may require hyperparameter tuning to mitigate overfitting on smaller datasets. CNN+BiGRU offers a balanced approach by extracting both spatial and temporal features, yet it could benefit from careful regularization to optimize performance. CNNs provide a computationally efficient solution for spatial feature extraction, with potential improvements achievable through tuning to better capture temporal dependencies. Below Table 18 gives us a clear comparison between various aspects of the proposed deep learning architectures.

Table 18

Comparison of Proposed Models Compatibility With Various Aspects

Characteristic	**ViT**	**CNN+BiGRU**	**CNN+LSTM**	**CNN**
Architecture	Parallel (Self-Attention)	Parallel + Linear (CNN + RNN)	Parallel + Linear (CNN + RNN)	Parallel
Data Type	All types (incl. audio, image)	Audio, Time-Series	Audio, Time-Series	Audio, Image
Small/Sparse Data	Struggles with small data	Moderate	Moderate	Good
Overfitting/ Underfitting	Overfitting on small datasets	Overfitting risk in CNN layers	Overfitting risk in CNN layers	Lower risk
Complexity	High	High	High	Moderate

Note. This Table 18 is continued on the next page.

Characteristic	ViT	CNN+BiGRU	CNN+LSTM	CNN
Preprocessing Required	Extensive	Significant	Significant	Moderate
Training Time	Long	Long	Long	Shorter
Space Complexity	High	High	High	Moderate
Computational Complexity	GPU intensive	GPU intensive	GPU intensive	CPU/GPU
Strengths	Captures complex patterns	Spatio-temporal feature capture	Long-term temporal modeling	Efficient spatial feature extraction
Limitations	High resource demand, Overfitting	Computational demand, Overfitting	Computational demand, Overfitting	Limited temporal modeling

CNN+BiGRU and CNN+LSTM models are powerful for capturing spatio-temporal features but can be computationally expensive and prone to overfitting. In contrast, CNNs offer efficient spatial feature extraction with lower complexity and quicker training times, though they may struggle with capturing long-term temporal information.

4.4 Model Evaluation Methods

Commonly, classifiers are assessed by comparing their predicted labels to the actual labels of audio files, with metrics derived from the confusion matrix at the class label level. As outlined by Bubashait and Hewahi (2021), evaluation metrics such as ROC-AUC, Accuracy, Precision, F1 score, and Recall are calculated from the confusion matrix, capturing the performance outcomes for all the proposed deep learning models.

Confusion Matrix

The Confusion Matrix serves as a crucial instrument for evaluating the accuracy of a model in the domain of urban audio classification. It provides a structured overview by splitting predictions into four distinct categories: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). This breakdown is especially valuable in urban environments where it's essential to accurately differentiate between various types of sounds, such as vehicle horns or human speech. Barchiesi et al. (2015) have demonstrated the effectiveness of employing a confusion matrix to gauge the performance of CNN models tasked with categorizing urban sounds. The matrix not only sheds light on the model's overall success rate but also pinpoints specific sound categories that may be misclassified by the model.

TP - True Positives reflect the correct identification of sounds from the urban environment. TN - True Negatives represents the accurate recognition of sounds not associated with urban settings. FP - False Positives denotes instances where non-urban sounds are mistakenly tagged as urban. FN - False Negatives occur when sounds from the urban environment are wrongly classified as non-urban. Below Table 19 is an example of a simple confusion matrix which is a summary of the classification results.

Table 19

Example of a Confusion Matrix for a Simple 3-Class Classification Problem

	Predicted: A	**Predicted: B**	**Predicted: C**
Actual:A	TP_A	FN_B	FN_C
Actual:B	FP_A	TP_B	FN_B
Actual:C	FP_C	FP_B	TP_C

Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. This matrix provides a clear picture of the model's performance.

Precision

Salmon et al. (2016) describe Precision as a performance metric that quantifies the correctness of positive predictions from a classifier. It assesses the proportion of accurate positive identifications among all positive classifications made. Within urban audio classification, precision provides insight into the reliability of the model's predictions for a specific sound type, confirming the extent to which sounds classified into a category truly belong to it. Precision is particularly relevant in urban sound scenarios where the accurate identification of sounds such as car horns or sirens is crucial for the system's credibility and effectiveness. The Precision Equation 8 is mentioned below.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \tag{8}$$

In urban sound classification, for example, if 'True Positives' represents the number of times a model correctly identifies a sound as a car horn, and 'False Positives' represents the number of times other sounds (like sirens or voices) were incorrectly identified as car horns, then precision tells us how often the model was actually right when it claimed to hear a car horn. Precision becomes crucial when there is a need to minimize false alarms, especially when certain urban sounds are rare, and false positives can be disruptive.

Recall (Sensitivity)

Recall, often referred to as Sensitivity or True Positive Rate (TPR), is an evaluation metric that gauges a model's proficiency in identifying all actual cases of a particular class. In

urban audio classification, recall indicates the model's effectiveness in detecting and correctly classifying every instance of a specific type of sound. As highlighted in the comparative research by Nogueira et al. (2022), recall is a crucial metric for assessing a model's performance, especially in scenarios where missing a positive instance (like failing to detect a specific urban sound) could have significant implications. It essentially measures the model's ability to minimize false negatives in its predictions. The Equation 9 is given by,

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{9}$$

In urban audio classification, recall is vital when it is crucial to capture as many instances of a particular urban sound as possible, particularly when dealing with rare or important sounds.

Accuracy Score

Accuracy stands as a fundamental metric in the field of urban audio classification, representing the ratio of correctly identified observations to the overall number of observations. This metric offers a general assessment of a model's effectiveness. However, its interpretation requires caution, particularly in cases of class imbalance, as it might not fully reflect the model's performance in distinguishing between different classes. This aspect of accuracy, along with its comparison to other performance indicators in urban audio classification, has been explored in depth in the study by Lamrini et al. (2023). The calculation of accuracy is crucial in understanding a model's overall predictive capabilities, yet it should be complemented with other metrics for a more comprehensive evaluation of model performance.

The Equation 10 for Accuracy score is given by,

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{10}$$

In an urban audio classification task with a balanced dataset, where sounds like traffic, human chatter, and emergency sirens are equally represented, a high accuracy score would effectively indicate the model's ability to correctly distinguish among these sound types. Conversely, in an imbalanced dataset heavily skewed towards, say, traffic noise, a model might achieve high accuracy by primarily identifying most sounds as traffic noise, while failing to accurately detect the less frequent siren or human sounds, thus skewing the true assessment of its classification ability.

F1-score

The F1-score is a crucial performance metric, especially in the context of urban audio classification. It is the harmonic mean of precision and recall, providing a balance between the two. The F1-score Equation 11 is calculated by,

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{11}$$

In urban audio classification, where it's vital to accurately identify and differentiate between various sound types like car horns, human voices, or sirens, the F1-score becomes particularly informative. It offers a more nuanced view of the model's performance than accuracy alone, especially in imbalanced datasets. The F1-score has been employed and clearly explained in the study of Arnault et al. (2020). Interpreting the F1-score in this context involves considering both the model's precision (its ability to correctly identify a specific sound type without falsely labeling other sounds as that type) and its recall (its ability to identify all instances of a specific sound type). A high F1-score indicates that the model effectively recognizes a specific sound type with minimal confusion, a critical factor in urban environments where accurately distinguishing between different sound sources is essential. Conversely, a low

F1-score might indicate that the model is either missing a significant number of occurrences of a sound type (low recall) or incorrectly identifying other sounds as that type (low precision), signaling areas where the model's performance could be improved.

Receiver Operating Curve (ROC) Area Under the Curve (AUC)

The Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) are significant performance metrics in urban audio classification. The ROC curve plots the True Positive Rate (TPR, or Recall) against the False Positive Rate (FPR) at various threshold settings, providing a comprehensive view of the classifier's performance across different levels of sensitivity and specificity which has been showcased in various Urban audio classification studies like Luz et al. (2021). The calculation of AUC is done using the below Equation 12.

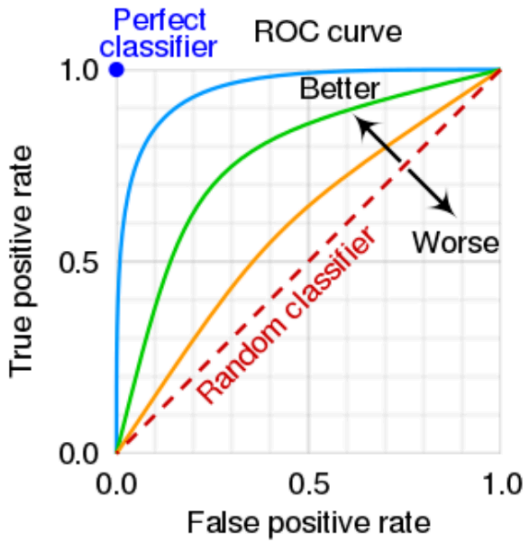
$$\mathrm{AUC} = \int_0^1 \mathrm{TPR}(x)\, dx \tag{12}$$

The ROC score's utility shines when evaluating different modeling methods. Interpreted as a probability, it consistently ranges between 0 and 1, making it a reliable metric for comparative analysis across various models. This concept entails calculating the True Positive Rate (TPR) across all potential False Positive Rate (FPR) values, ranging from 0 to 1. In essence, this is the integration of the TPR in relation to the FPR. The ROC AUC is derived by charting the TPR against the FPR at various thresholds to create the ROC curve, with the TPR plotted on the y-axis and the FPR on the x-axis. The Area Under the Curve (AUC) corresponds to the area under this ROC curve, extending from the lower left (0,0) to the upper right (1,1) of the graph. This integration of the TPR over the FPR spectrum results in the AUC, which measures the probability that the classifier will correctly prioritize a randomly selected positive instance over a

negative one. An AUC value of 0.5 reflects a lack of discriminative capacity, equivalent to chance, while an AUC of 1 indicates an ideal classifier with perfect ability to differentiate positive from negative classes. Figure 91 demonstrates the ROC plot for TPR and FPR.

Figure 91

An Example of ROC AUC Curve



Performance above the line signifies enhanced outcomes, where the model's true positive rate surpasses random expectations, showcasing effective performance. Being on the line indicates average, random outcomes, suggesting the model's class discrimination is equivalent to making guesses at random. Falling below the line implies subpar results, with a true positive rate lower than random chance, reflecting inadequate performance of the model.

4.5 Model Validation and Evaluation

To get this phase started, there are some data transformations that are implemented on the audio files from UrbanSound8k and ESC-50 dataset, as part of the Data Engineering phase, these sets of transformations are common for all both the models. The pre-processing steps of converting the audio files into mel-scaled spectrograms is broken down into step by step process

and documented in the below Table 20.

Table 20

Augmented File Distribution for Datasets

Dataset Name	No. of Original Spectrograms	No. of Augmented Spectrograms	Total Dataset Size Post-Augmentation
UrbanSound8k	8,732	26,196	34,928
ESC-50	2,000	6,000	8,000

The below Table 21 shows the PCA Components, Reduced PCA and Variance Captured

Table 21

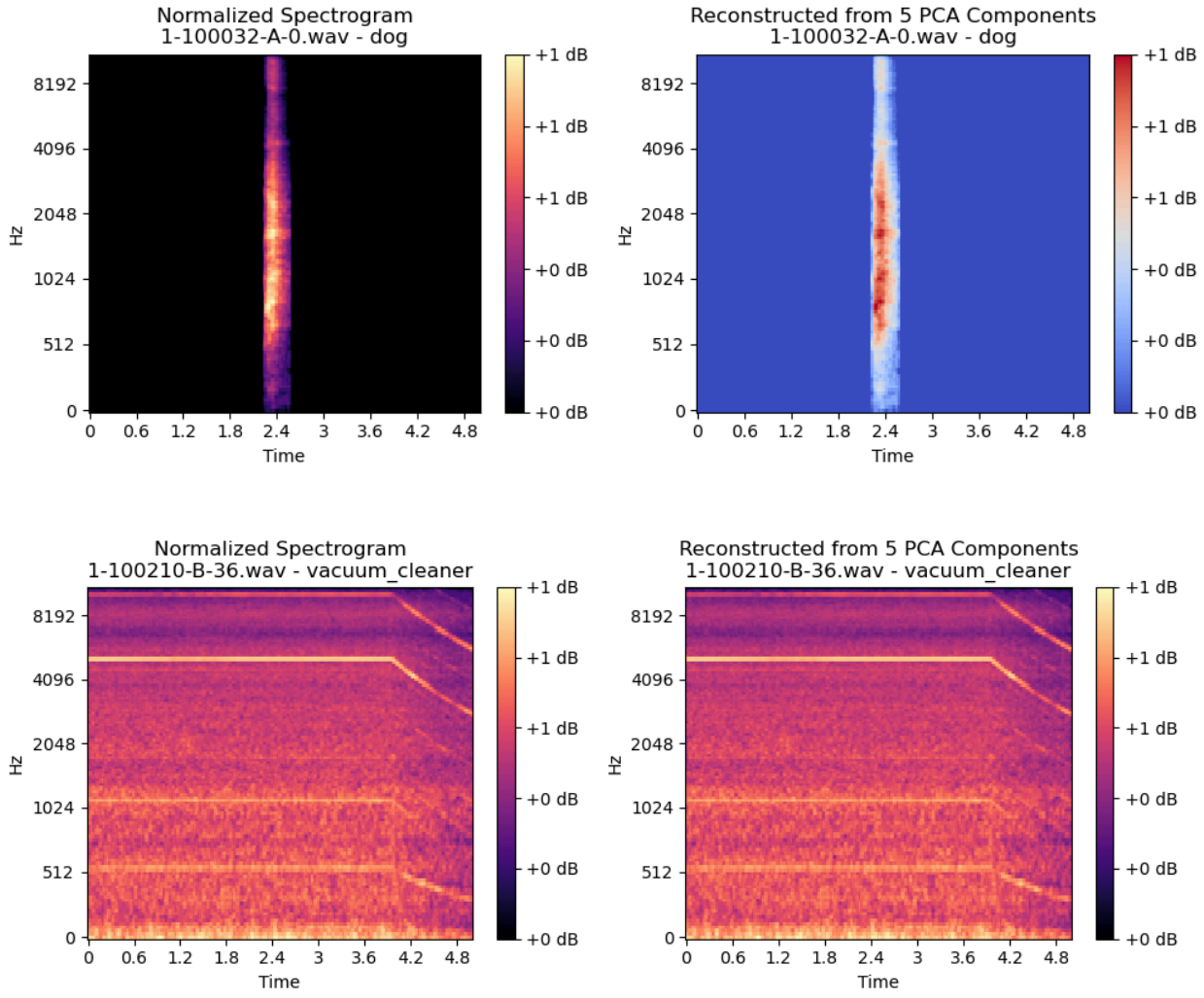
PCA Distribution of the Post Augmentation Data

Dataset Name	PCA components	Reduced PCA	% Variance Captured
UrbanSound8k	212,928	1,60,281	90%
ESC-50	50,600	31,300	90%

Through the variance against PCA components plot, the audio signal verification post PCA is done by re-constructing the mel-scaled spectrogram from the reduced components, and this is depicted below, for both UrbanSound8k and ESC-50 datasets. PCA is done on the whole dataset including Augmented mel-scaled spectrograms, and is applied to the test set separately while testing for the classification accuracies and other performance metrics. The reduced components here helped in decreasing the computational complexity of the training times of the proposed deep learning models and below Figure 92 shows the Pre and Post PCA Results.

Figure 92

Pre-PCA and Post-PCA Results for a Sample to Verify the Basic Audio Signal is Still Captured in ESC-50 and UrbanSound8k Respectively



From the above comparative analysis, it can be concluded that the PCA on the augmented datasets is a valid step as it captures the essential characteristics of the Urban audio sound. PCA is fitted after scaling the reduced components with standardscaler() method from sklearn.preprocessing. For PCA, sklearn.decomposition is utilized to fit the training, and testing data to PCA transform.

Vision Transformers

Baseline Model. The foundational model vision transformers architecture used in this study is "facebook/deit-base-patch16-224" from the Hugging Face Transformers library. The research primarily focuses on the ESC-50 and UrbanSound8k datasets. ESC-50 originally contains 2000 audio files divided into 50 categories, with each category having 40 files. In contrast, UrbanSound8k consists of 8,732 files categorized into 10 different classes based on Urban Taxonomy.

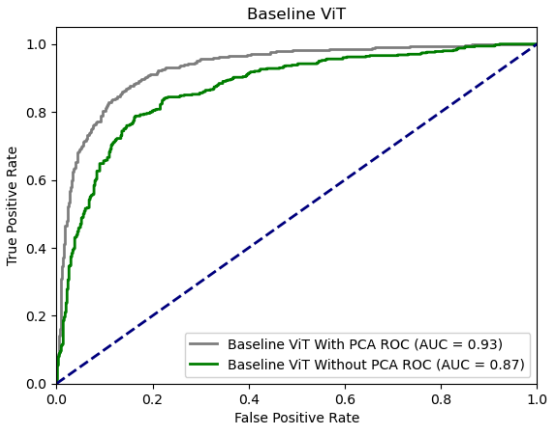
The project aims to evaluate the impact of Data Augmentation and Principal Component Analysis (PCA) on the Vision Transformer (ViT) model using these datasets. Given the limited number of audio files in both datasets, audio augmentation techniques such as SpecAugment, Noise Induction, and Time Stretching were employed. This resulted in the creation of 6,000 additional mel-scaled spectrograms, thereby expanding each of the 50 ESC-50 classes with 1,200 augmented spectrograms. Similarly, for the UrbanSound8k dataset, the 8,732 original files were augmented to yield 26,196 files, maintaining the distribution across the 10 classes.

These augmentations, applied to the mel-scaled spectrograms generated from the audio files, preserved the essential latent characteristics of the audio signals while varying aspects like amplitude, time, and frequency components. This approach facilitated the development of a more diverse dataset. The study also involves comparing the Receiver Operating Characteristic Area Under the Curve (ROC AUC) metric for the datasets, both before and after applying PCA. This comparison covers 8,000 spectrograms from ESC-50 and 34,928 from UrbanSound8k, including the augmented files. The results are depicted in Figure 4, illustrating how the ViT model yields different AUC scores for the pre- and post-PCA variants of the datasets. This figure provides a

comparative analysis of the AUC for both the datasets in Figure 93 and Figure 94. Figure 92 is a similar comparison between the pre-PCA and post-PCA variants of UrbanSound8k.

Figure 93

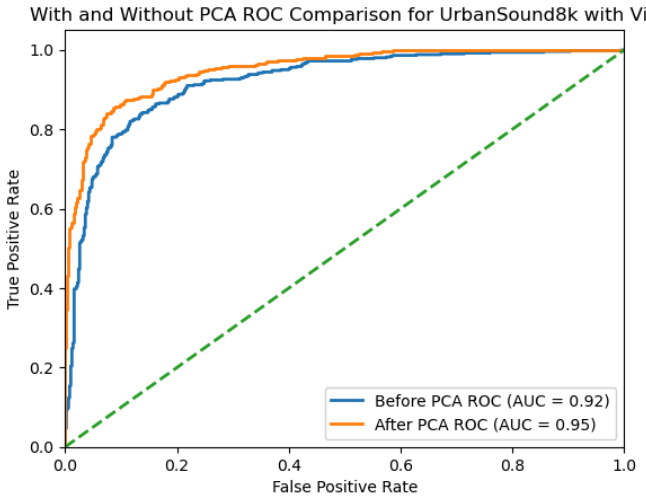
With And Without AUC With Baseline ViT For the ESC-50 Dataset (Including Augmented Files)



Clearly, PCA when employed with significant augmentation on ESC-50 provides a good result of AUC 0.93, improving from 0.87 which was still a better score, achieved through the significant generalization from the Augmentations applied.

Figure 94

Before and After PCA Comparison of AUC Metric for UrbanSound8k With ViT Architecture



From the above visualizations of AUC curves, and with the support of the below training time related metrics and performance metrics, it can be concluded that PCA definitely is compatible with the Urban Sound Classification scenario, and even better when supported with proper Augmentation techniques, improving performance as illustrated by Mushtaq et al. (2021). As ViT is very compute intensive, this was done on GPU through Google Collab pro subscription. The comparison results are shown in Table 22 below.

Table 22

Comparison of Results on Baseline ViT With and Without PCA in Validation Phases

	ESC-50		**UrbanSound8k**	
Evaluation KPI	**W/O PCA**	**With PCA**	**W/O PCA**	**With PCA**
Precision	0.84	0.91	0.90	0.94
F1-score	0.82	0.93	0.91	0.95
Accuracy Score	0.85	0.92	0.92	0.96
Recall	0.85	0.92	0.89	0.93
ROC AUC	0.87	0.93	0.92	0.95
Training Time	99 mins, 10 secs	80 mins, 15 secs	370 mins, 45 secs	330 mins, 42 secs

Hyperparameter Tuning. In the process of optimizing the Facebook/Deit-Base-Patch 16-224 vision transformers, several key hyperparameters were adjusted, resulting in significant improvements in the model's performance. The initial learning rate set at 0.001 was reduced to 0.0001, allowing for more precise fine-tuning of the model. To improve the accuracy of gradient estimation and hasten convergence, the batch size was doubled from 32 to 64. When signs of

overfitting appeared, weight decay, which serves as L2 regularization, was increased tenfold from 0.01 to 0.1. The number of training epochs was also expanded to 100, offering the model additional opportunities to learn, while the early_stopping hyperparameter was enabled to guard against overfitting. Additionally, both dropout and attention drop rates were raised from 0.1 to 0.2 to further mitigate overfitting. Directly below Table 23, there is a detailed summary of the specific hyperparameter values used for the Vision transformers.

Table 23

A Clear Summary of the Hyperparameters Used Upon Baseline ViT

Evaluation KPI	Default	Hypertuned
Learning Rate	0.001	0.0001
Batch Size	32	64
Number of Epochs	15	100 (early_stopping=true)
Weight Decay (L2 Regularization)	1e-2	1e-1
Optimizer	betas=(0.9, 0.999), eps=1e-8	betas=(0.89, 0.899), eps=1e-9
Dropout rate	0.1	0.2
Attention dropout	0.1	0.2

The performance metrics after applying the hyperparameter tuning to the With PCA, and W/O PCA variants is described in the below Table 24, where it can be clearly observed that the PCA with ViT is clearly a better combination in terms of performance for ESC-50 dataset.

Table 24

Hyperparameter Tuning Impact on Performance Metrics for ESC-50 on ViT on Test Dataset

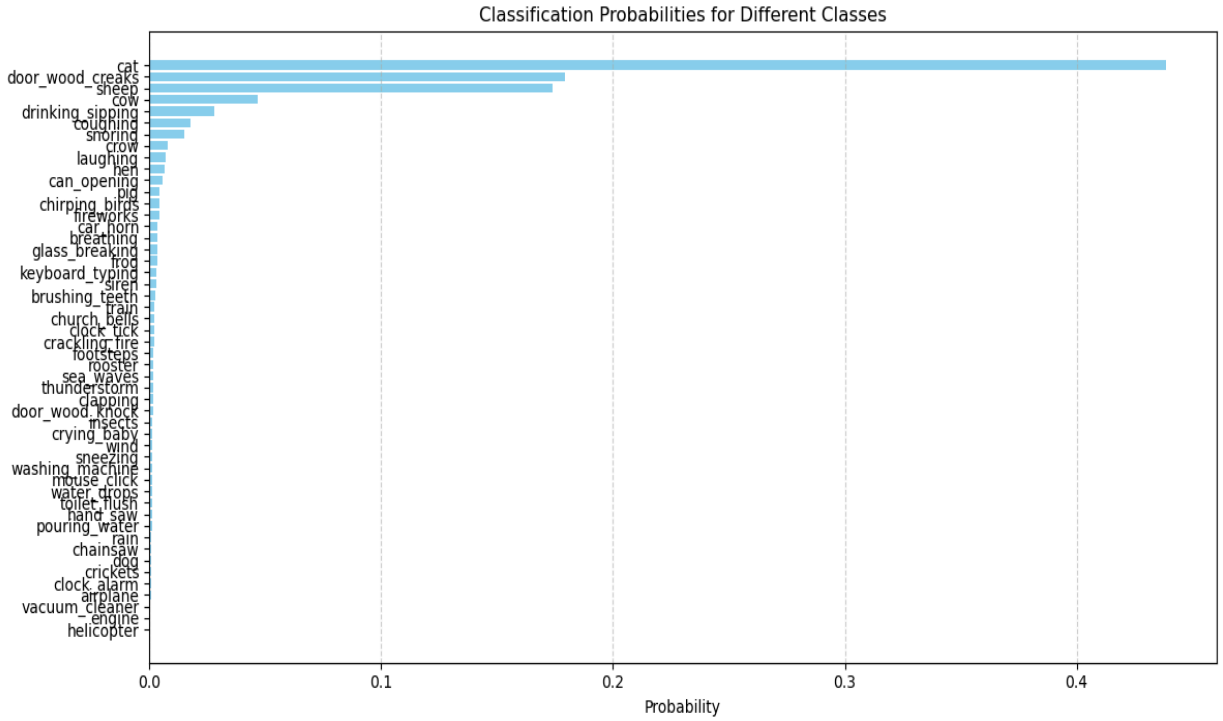
	ESC-50				UrbanSound8k			
	Validation		Testing		Validation		Testing	
KPI	W/O PCA	With PCA	W/O PCA	With PCA	W/O PCA	With PCA	W/O PCA	With PCA
Precision	0.84	0.91	0.90	0.93	0.90	0.94	0.94	0.96
F1-score	0.82	0.93	0.88	0.93	0.91	0.95	0.95	0.97
Accuracy Score	0.85	0.92	0.89	0.94	0.92	0.96	0.94	0.96
Recall	0.85	0.92	0.91	0.95	0.89	0.93	0.93	0.94
ROC AUC	0.87	0.93	0.92	0.98	0.92	0.95	0.95	0.98

The changes made to the hyperparameters of the Facebook/Deit-Base-Patch16-224 vision transformers have led to notable improvements in key performance indicators (KPIs) for the Urban Audio classification task on both the ESC-50 and UrbanSound8k datasets. The above comparison of performance metrics for both the ESC-50 and UrbanSound8k dataset concludes that the data engineering principles that were employed in this research Specaugment augmentation, feature reduction techniques like PCA on mel-scaled spectrograms. These adjustments collectively enhanced the model's ability to learn and generalize from the data, leading to the observed improvements in precision, F1-score, accuracy score, recall, and ROC AUC across both datasets and PCA configurations. The balanced approach to increasing learning opportunities (via more epochs and larger batch sizes) while simultaneously implementing

controls against overfitting (via increased weight decay and dropout rates) appears to have been particularly effective. Below figure 95 shows a classification result from the ESC-50 mel-scaled spectrograms dataset.

Figure 95

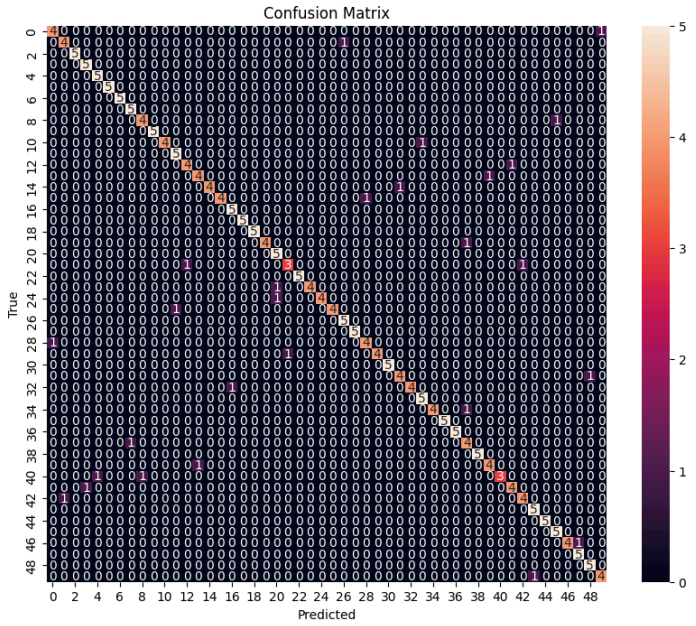
Classification Results for a Test Mel-Scale Spectrogram From ESC-50 Dataset.



The final soft-max layer in the ViT architecture takes care of computing the probabilities of the input test data belonging to all the classes, from the above visualization it's clear that the test spectrogram belongs to the class “cat” with the highest probability at 0.45. Below Figure 96 is a Confusion matrix from the ESC-50 classification result, the diagonal signifies that most of the classifications are accurate for all the classes, which is good news.

Figure 96

Confusion Matrix as a Classification Result for a ESC-50 Test Dataset



Majority of the values lying on the diagonal signifies that the model accurately classifies the True Positives, and True Negatives from the test dataset, in this case ESC-50, which is great!

CNN + BiGRU

Baseline Model. The initial baseline model CNN + BiGRU architecture was built using keras.layers library where required architecture is created by appening layers based on the kind of Neural network that is being employed. A baseline CNN + BiGRU is validated against ESC-50 and UrbanSound8k in validation phases. The project entailed managing a considerable amount of audio data, around 12 GB in size. To guarantee the security and availability of this data, it was stored and backed up using two cloud services: Google Cloud and AWS S3 Bucket. Below Figure 8 are the accuracy and other performance metrics with the baseline model against the Validation data for ESC-50 and UrbanSound8k datasets. Below Table 25 shows the Validation Performance.

Table 25

Validation Performance Metrics for a Baseline CNN + BiGRU Model for ESC and UrbanSound8k Datasets

	ESC-50		UrbanSound8k	
Evaluation KPI	**W/O PCA**	**With PCA**	**W/O PCA**	**With PCA**
Precision	0.68	0.78	0.85	0.90
F1-score	0.69	0.79	0.84	0.89
Accuracy Score	0.71	0.76	0.86	0.91
Recall	0.70	0.75	0.85	0.90
ROC AUC	0.75	0.80	0.80	0.90
Training Time	70 mins, 10 secs	50 mins, 15 secs	270 mins, 45 secs	220 mins, 42 secs

From the above comparison table, it can be observed that the impact of PCA when used with validation data of ESC-50 and Urbansound8k results in a positive impact on the training time and other performance metrics.

Hyperparameter tuning. Lowering the learning rate from 0.001 to 0.0001 leads to smaller updates in the model weights, which can enhance the model's ability to converge on a more accurate solution, albeit at a slower pace. This fine-tuning is especially beneficial in audio classification where subtle features might be significant like in Urban audio classification. Increasing the batch size from 32 to 64 allows the model to process more data before updating its weights. This can improve the stability of the learning process and provide a more accurate estimate of the gradient. Extending the number of epochs from 20 to 50, with early stopping

enabled, allows the model more opportunities to learn from the data. Early stopping ensures that training ceases when the model no longer improves, preventing overfitting and optimizing training time. Increasing weight decay from 1e-4 to 1e-3 adds a stronger penalty for large weights, which can help in reducing overfitting. This is particularly useful in complex models like CNN + BiGRU, as it encourages the model to learn more robust features. Adjusting the Adam optimizer's betas from (0.9, 0.999) to (0.85, 0.95) and eps from 1e-8 to 1e-9 alters the momentum and stabilization terms. This change can lead to a different optimization trajectory, potentially finding new and possibly better minima in the loss landscape. Increasing the dropout rate from 0.2 to 0.3 helps in preventing overfitting by randomly deactivating a proportion of neurons during training. This forces the network to learn more robust features, as it cannot rely on any single neuron. Doubling the GRU layer size from 128 to 256 increases the model's capacity to learn complex patterns in the data. This can be particularly advantageous for distinguishing between the nuanced sounds in urban audio classification. Increasing the number of GRU layers from 2 to 3 adds more depth to the model, allowing it to learn more complex, hierarchical representations of the audio data. Below is Table 26, which summarizes the hyperparameters and the changes for the CNN + BiGRU model, for both datasets.

Table 26

Detailed Hyperparameters Tuning for the CNN + BiGRU Architecture

Hyperparameter	Default	Hypertuned
Learning Rate	0.001	0.0001
Batch Size	32	64

Note. The Table 26 is continued on the next page

Hyperparameter	Default	Hypertuned
Number of Epochs	20	50 (early_stopping=true)
Weight Decay (L2 Regularization)	1e-4	1e-3
Optimizer	Adam (betas=(0.9, 0.999), eps=1e-8)	Adam (betas=(0.85, 0.95), eps=1e-9)
Dropout rate	0.2	0.3
GRU Layer Size	128	256
GRU Number of Layers	2	3

The above hyperparameters are found out by employing the GridSearchCV algorithm, where a set of hyperparameters are tested upon the model continuously, and the best score is captured. Below Table 27 is a summary of the performance metrics, and how they vary with Hyperparameter tuning, and for the test data from ESC-50 and UrbanSound8k.

Table 27

Summary of Performance Metrics and Impact of Hyperparameter Tuning with CNN + BiGRU on Test Data From UrbanSound8k and ESC-50 Datasets

	ESC-50				UrbanSound8k			
	Validation		Testing		Validation		Testing	
KPI	W/O PCA	With PCA	W/O PCA	With PCA	W/O PCA	With PCA	W/O PCA	With PCA
Precision	0.68	0.78	0.81	0.88	0.85	0.90	0.86	0.90
F1-score	0.69	0.79	0.83	0.86	0.84	0.89	0.88	0.90

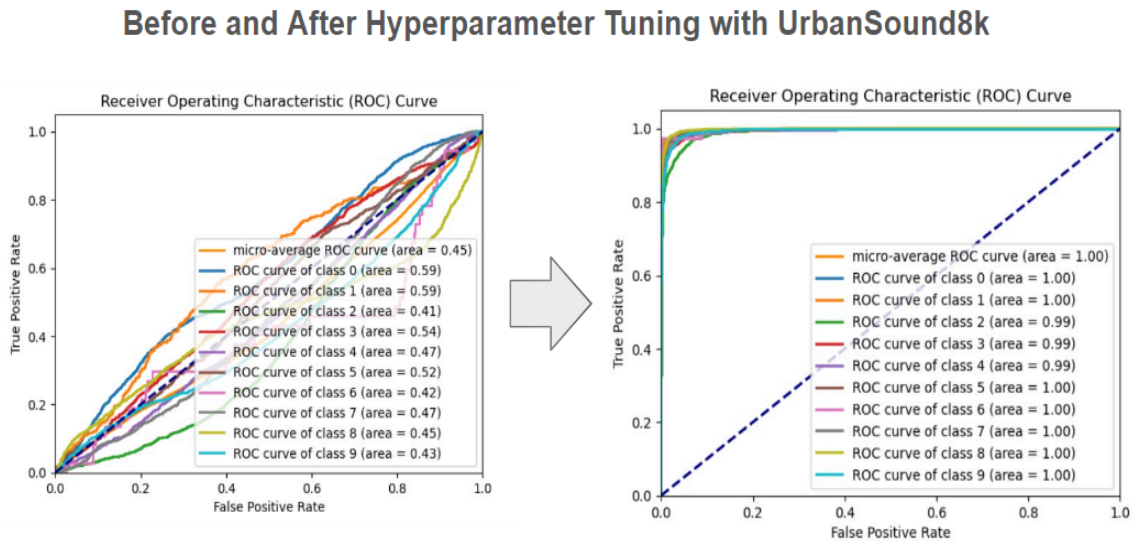
Note. The Table 27 is continued on the next page

	ESC-50				UrbanSound8k			
	Validation		Testing		Validation		Testing	
KPI	W/O PCA	With PCA	W/O PCA	With PCA	W/O PCA	With PCA	W/O PCA	With PCA
Accuracy Score	0.71	0.76	0.84	0.89	0.86	0.91	0.89	0.92
Recall	0.70	0.75	0.82	0.88	0.85	0.90	0.89	0.91
ROC AUC	0.75	0.80	0.85	0.88	0.80	0.90	0.85	0.95

From the above comparative analysis, it can be concluded that the hyperparameter actually worked with the CNN + BiGRU, while employing PCA and data augmentation techniques. Below is figure 97 which visualizes the class level ROC curves for UrbanSound8k, before and after Hyperparameter tuning.

Figure 97

Positive Impact of Hyperparameter Tuning on the UrbanSound8k Using Class Wise ROC

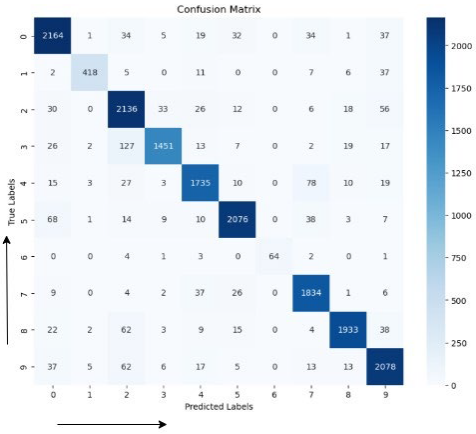


A very clear improvement from the baseline model on the class level accuracy after the hyperparameter tuning using the specific hyperparameters on UrbanSound8k.

Below Figure 98 is a visualization of the confusion matrix post Hyperparameter tuning, considering this confusion matrix as final prediction results on the UrbanSound8k dataset.

Figure 98

Confusion Matrix with Classification Results for UrbanSound8k Test Data



CNN + LSTM

Baseline Model. The RCNN architecture here is CNN+LSTM architecture, this architecture has been implemented with Keras.layers, adding layers of Neural networks based on the architecture. The construction of the CNN + LSTM architecture for urban audio classification combines the feature extraction capabilities of Convolutional Neural Networks (CNNs) with the sequential data processing strengths of Long Short-Term Memory (LSTM) networks. Initially, the CNN layers extract relevant features from the audio spectrograms, efficiently capturing both temporal and frequency domain characteristics. These extracted features are then fed into the LSTM layers, which are adept at handling time-series data, allowing the model to understand and model the temporal dynamics of urban sounds. This hybrid architecture leverages the strengths of both CNNs and LSTMs, making it particularly effective for complex audio classification tasks

like distinguishing various urban sounds. The baseline model of CNN + LSTM, initially evaluated upon validation datasets of ESC-50 and UrbanSound8k and compared in the below Table 28.

Table 28

Below is the Comparative Analysis of the PCA Impact on Both ESC-50 & UrbanSound8k

	ESC-50		UrbanSound8k	
Evaluation KPI	**W/O PCA**	**With PCA**	**W/O PCA**	**With PCA**
Precision	0.67	0.71	0.84	0.90
F1-score	0.69	0.74	0.83	0.92
Accuracy Score	0.71	0.76	0.86	0.93
Recall	0.70	0.74	0.88	0.91
ROC AUC	0.73	0.80	0.90	0.96
Training Time	80 m, 56 s	68 m, 24 s	260 m, 36 s	240 m, 46 s

Hyperparameter Tuning. Reducing the learning rate from 0.001 to 0.0001 in the CNN + LSTM model can significantly enhance the precision of model training. A lower rate allows for finer adjustments to the model weights, potentially leading to better performance, especially in distinguishing subtle features in urban audio data. However, the trade-off is longer training times.Increasing the batch size from 32 to 64 means the model will process a larger set of data before updating its weights. This can lead to more stable gradient estimates and potentially smoother convergence. Extending the training duration from 20 to 50 epochs, while incorporating early stopping, gives the model more opportunity to learn from the dataset.

Early stopping is a key addition as it prevents overfitting by halting training when the model ceases to improve. Increasing the weight decay helps in regularizing the model. A higher decay rate (1e-3) penalizes large weights more, encouraging the model to find simpler patterns, which might be more generalizable and less prone to overfitting. Tweaking the parameters of the Adam optimizer can affect the model's convergence behavior. The adjusted betas and eps values may enable the model to explore the loss landscape differently, possibly leading to improved performance in classifying audio data. Increasing the dropout rate from 0.2 to 0.3 helps in reducing overfitting by randomly dropping units in the neural network. This forces the model to learn more robust and redundant representations of the data. Upscaling the LSTM layer size from 128 to 256 enhances the model's capacity to capture complex features in the data. This can be especially beneficial for processing the intricate patterns found in urban soundscapes. Adding an extra layer (going from 2 to 3) allows the model to learn more complex, hierarchical representations of the data. While this can improve the model's learning capability, it also increases computational complexity and the risk of overfitting. Below Table 29 is a summary of the values of specific hyperparameters and how the default values have been changed as part of the hyperparameter tuning process.

Table 29

Hyperparameters for CNN + LSTM Architecture And the Change in Values

Hyperparameter	Default	Hypertuned
Learning Rate	0.001	0.0001
Batch Size	32	64

Note. The Table 29 is continued on the next page

Hyperparameter	Default	Hypertuned
Number of Epochs	20	50 (early_stopping=true)
Weight Decay (L2 Regularization)	1e-4	1e-3
Optimizer	Adam (betas=(0.9, 0.999), eps=1e-8)	Adam (betas=(0.85, 0.95), eps=1e-9)
Dropout rate	0.2	0.3
LSTM Layer Size	128	256
LSTM Number of Layers	2	3

Upon tuning the hyperparameters with the above values for the CNN + LSTM architecture with ESC-50 and UrbanSound8k dataset, there have been little differences in the changes of the Performance metrics, but there is a slight improvement which is shown in the below Table 30, it shows the final comparative analysis with and without PCA, before and after hyperparameter tuning for both the ESC-50 and UrbanSound8k datasets.

Table 30

Comparative Analysis of the Performance Metrics for CNN + LSTM Model, With ESC-50 and UrbanSound8k Dataset.

	ESC-50				**UrbanSound8k**			
	Validation		**Testing**		**Validation**		**Testing**	
KPI	**W/O PCA**	**With PCA**	**W/O PCA**	**With PCA**	**W/O PCA**	**With PCA**	**W/O PCA**	**With PCA**
Precision	0.67	0.71	0.75	0.80	0.84	0.90	0.91	0.93

Note. The Table 30 is continued on the next page

	ESC-50				UrbanSound8k			
	Validation		Testing		Validation		Testing	
KPI	W/O PCA	With PCA	W/O PCA	With PCA	W/O PCA	With PCA	W/O PCA	With PCA
F1-score	0.69	0.74	0.74	0.81	0.83	0.92	0.89	0.92
Accuracy Score	0.71	0.76	0.78	0.82	0.86	0.92	0.91	0.93
Recall	0.70	0.74	0.78	0.83	0.88	0.91	0.90	0.94
ROC AUC	0.73	0.80	0.83	0.90	0.90	0.94	0.93	0.96

From the above metrics, it can be observed that the performance metrics indicate that both hypertuning and PCA are effective strategies for improving the performance of a CNN + LSTM model in audio classification tasks, with a more pronounced effect observed on the UrbanSound8k dataset. Hypertuning the CNN + LSTM model consistently improves its performance across both datasets and all metrics. This indicates that optimizing hyperparameters is crucial for maximizing model effectiveness CA, when applied, generally leads to better model performance. This suggests that reducing feature dimensionality while retaining essential information can significantly enhance the model's ability to classify audio data accurately. The model seems more adept with the UrbanSound8k dataset, as indicated by higher scores across all metrics, compared to the ESC-50 dataset. This could be due to inherent differences in the datasets or their suitability to the CNN + LSTM architecture. These results can signify the importance of a high amount of data for training purposes, which is a need for the deep learning models to perform.

CNN

Baseline model. This research studies and compares the performance of CNN, and its variants against Urban Audio classification tasks. CNN's were always effective in Image classification tasks, so experimenting with the audio image classification task made sense, and demonstrated in the work of Massoudi et al. (2021), while validating the ESC-50 and Urbansound8k datasets against the baseline CNN model, the initial accuracies on ESC-50 validation dataset showed low accuracies as the number of input files are close to ~7000, while the validation accuracies with UrbanSound8k (including augmented) start off at a good note. Below Table 31 is a comparison of the validation accuracies with and without PCA.

Table 31

In the Table Below, the Validation Performance Metrics are Shown for UrbanSound8k and ESC-50 Dataset With and Without PCA

	ESC-50		UrbanSound8k	
Evaluation KPI	**W/O PCA**	**With PCA**	**W/O PCA**	**With PCA**
Precision	0.63	0.71	0.83	0.89
F1-score	0.62	0.74	0.81	0.90
Accuracy Score	0.63	0.69	0.82	0.91
Recall	0.63	0.68	0.84	0.92
ROC AUC	0.60	0.70	0.84	0.91
Training Time	75 m, 26 s	67 m, 34 s	144 m, 36 s	120 m, 46 s

The Sheer amount of training data is the reason behind such huge differences in the performance metrics for UrbanSound8k, when compared to the ESC-50 dataset.

Hyperparameter Tuning. Decreasing the learning rate to 0.0001 allows the CNN model to make more granular updates to its weights, enhancing its ability to fine-tune the extraction of audio features. This careful adjustment can lead to improved accuracy. Increasing the batch size to 64 allows the model to process more data at once, which can aid in stabilizing the gradient descent process and potentially speed up convergence. However, this comes at the cost of increased computational load and memory usage. Extending the number of epochs to 50, along with implementing early stopping, provides the model with more iterations to learn from the data, while preventing overfitting by terminating training when no improvement is observed. A higher weight decay rate (1e-3) introduces a more substantial penalty for large weights, encouraging the model to maintain simpler and more generalizable features, which can be crucial in complex audio environments.

Modifying the Adam optimizer's parameters can significantly impact the model's optimization path. The hypertuned settings might help the model in navigating the loss landscape more effectively, potentially discovering better minima. Raising the dropout rate to 0.3 increases the model's resilience against overfitting by randomly deactivating a portion of neurons during training. This encourages the model to develop more robust feature representations. Increasing from 4 to 6 layers allows the CNN to capture a wider range of features at different levels of abstraction, which is beneficial for analyzing complex urban soundscapes. The hypertuned larger filter sizes (ranging up to 1024) enable the model to capture more nuanced features within the audio data, potentially improving its ability to differentiate between various urban sounds.

These hyperparameter adjustments for the CNN model are targeted towards enhancing its performance in classifying urban audio data. Below Table 32 shows the specific hyperparameters that were tuned for the CNN model for both ESC-50 and UrbanSound8k datasets.

Table 32

Comparative Analysis of The Specific Hyperparameters for ESC-50 and UrbanSound8k

Hyperparameter	Default	Hypertuned
Learning Rate	0.001	0.0001
Batch Size	32	64
Number of Epochs	20	50 (early_stopping=true)
Weight Decay (L2 Regularization)	1e-4	1e-3
Optimizer	Adam (betas=(0.9, 0.999), eps=1e-8)	Adam (betas=(0.85, 0.95), eps=1e-9)
Dropout rate	0.2	0.3
Number of Convolutional Layers	4	6
Filter Sizes	[64, 128, 256, 512]	[128, 256, 512, 1024]

The testing performance metrics with the hyperparameter tuning implemented yields better results when validated using the classification performance metrics. The hyperparameters are found out by employing GridSearchCV method, the hyperparameters when used to tune the model, are meant to improve the performance and generalizing capabilities of the CNN model.

This fine-tuning process is crucial for optimizing the model to suit the specific challenges presented by urban audio

Below Table 33 is a comprehensive analysis of the performance metrics when tuned using the hyperparameters found out by employing the gridsearchcv on the CNN model.

Table 33

Below Table Compares the Performance Metrics for Different Variants of the ESC-50 and UrbanSound8k Datasets Before and after Hyperparameter Tuning

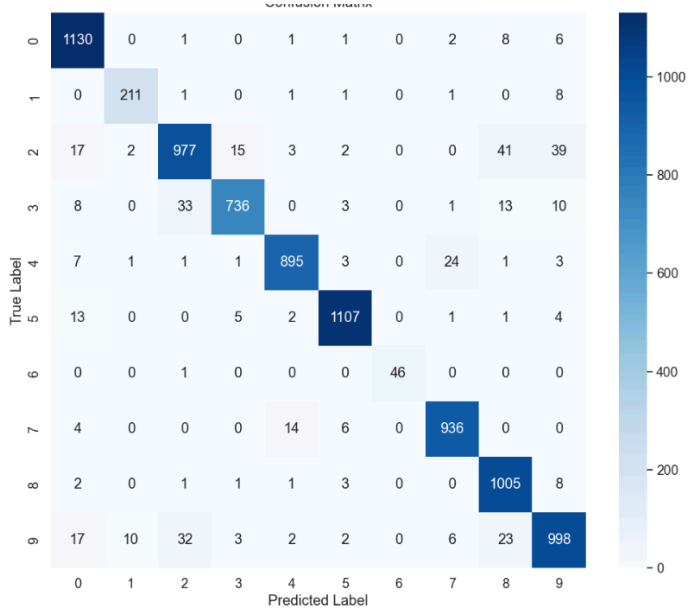
	ESC-50				**UrbanSound8k**			
	Validation		**Testing**		**Validation**		**Testing**	
KPI	**W/O PCA**	**With PCA**	**W/O PCA**	**With PCA**	**W/O PCA**	**With PCA**	**W/O PCA**	**With PCA**
Precision	0.63	0.71	0.75	0.80	0.83	0.89	0.91	0.93
F1-score	0.62	0.74	0.74	0.81	0.81	0.90	0.89	0.92
Accuracy Score	0.63	0.69	0.78	0.82	0.82	0.91	0.91	0.93
Recall	0.63	0.68	0.78	0.83	0.84	0.92	0.90	0.94
ROC AUC	0.60	0.70	0.83	0.90	0.84	0.91	0.93	0.96

It can be observed that the Hyperparameter tuning had a significant impact on the ESC-50 dataset, however the scores with Augmented PCA reduced UrbanSound8k are pretty good. The other performance metrics suggest that the CNN model, especially when augmented with PCA, is effective in urban audio classification tasks. The improvements observed with PCA

indicate its significant role in enhancing the model's ability to discern and classify complex urban sounds, particularly in the more diverse UrbanSound8k dataset. In both datasets, there is generally an improvement in metrics from the validation to the testing phase, indicating that the model is likely generalizing well and not overfitting. Below Figure 99 is an example of the confusion matrix which acts like a classification result for the testing done with the UrbanSound8k dataset.

Figure 99

Confusion Matrix on Testing Data from UrbanSound8k Dataset



Above confusion matrix further adds to the existing understanding of the results by confirming the class-wise predicted labels are predicted accurately for the majority of the part. As the majority values are all falling on the diagonal, this means that the classification was right for the most part. Below is Figure 100, which shows the class wise accuracy on the Urbansound8k test data.

Figure 100

Class Wise Performance Metrics on Test Data of UrbanSound8k

```
Class 0 - Precision: 0.9432, Recall: 0.9835, F1 Score: 0.9629
Class 1 - Precision: 0.9420, Recall: 0.9462, F1 Score: 0.9441
Class 2 - Precision: 0.9331, Recall: 0.8914, F1 Score: 0.9118
Class 3 - Precision: 0.9671, Recall: 0.9154, F1 Score: 0.9406
Class 4 - Precision: 0.9739, Recall: 0.9562, F1 Score: 0.9650
Class 5 - Precision: 0.9814, Recall: 0.9771, F1 Score: 0.9792
Class 6 - Precision: 1.0000, Recall: 0.9787, F1 Score: 0.9892
Class 7 - Precision: 0.9640, Recall: 0.9750, F1 Score: 0.9694
Class 8 - Precision: 0.9203, Recall: 0.9843, F1 Score: 0.9513
Class 9 - Precision: 0.9275, Recall: 0.9131, F1 Score: 0.9202
```

The above class wise performance metrics confirm that the hypertuned CNN model performs consistently well on the class level classification as well.

Final Results Comparison

Comparing the PCA reduced variants of the augmented ESC-50 and UrbanSound8k datasets, for understanding the overall results of the performance of the models on the Urban Audio Classification task using ViT, CNN+BiGRU, CNN + LSTM, and CNN. Below Table 34 compares the results of the various deep learning models employed in this research for ESC-50.

Table 34

Comparison of the Performance Metrics across Different models for ESC-50

	Validation					Testing				
Model Name	**Precision**	**F1-score**	**Accuracy**	**Recall**	**AUC ROC**	**Precision**	**F1-score**	**Accuracy**	**Recall**	**AUC ROC**
ViT	0.91	0.93	0.92	0.92	0.93	0.93	0.93	0.94	0.95	0.98

Note. The Table 34 is continued on the next page

Validation					Testing					
Precision	**F1-score**	**Accuracy**	**Recall**	**AUC ROC**	**Precision**	**F1-score**	**Accuracy**	**Recall**	**AUC ROC**	
CNN + BiGRU	0.78	0.79	0.76	0.75	0.80	0.88	0.86	0.89	0.88	0.88
CNN + LSTM	0.71	0.74	0.76	0.74	0.80	0.80	0.81	0.82	0.83	0.89
CNN	0.71	0.74	0.69	0.68	0.70	0.80	0.81	0.82	0.83	0.90

Below Table 35 similarly gives the performance metrics of deep learning models on PCA reduced Augmented data from UrbanSound8k dataset

Table 35

Comparison of the Performance Metrics Across Different Models for UrbanSoun8k

	Validation					Testing				
Model Name	**Precision**	**F1-score**	**Accuracy**	**Recall**	**AUC ROC**	**Precision**	**F1-score**	**Accuracy**	**Recall**	**AUC ROC**
ViT	0.94	0.95	0.96	0.93	0.95	0.96	0.97	0.96	0.94	0.98
CNN + BiGRU	0.90	0.89	0.91	0.90	0.90	0.90	0.90	0.92	0.91	0.95
CNN + LSTM	0.90	0.92	0.92	0.91	0.94	0.93	0.92	0.93	0.94	0.96
CNN	0.89	0.90	0.91	0.92	0.91	0.93	0.92	0.93	0.94	0.96

Conclusion

ViT stands at the forefront when compared with the performance metrics of all other models. Since Facebook's ViT is pre-trained on ImageNet and the architecture is specifically built for image classification, it achieves an accuracy of 0.93 with ESC-50 testing data. This suggests that ViT requires a significant amount of pre-training or a large training dataset. When evaluating the models on UrbanSound8k, the accuracies and other performance metrics of all deep learning models are satisfactory. This is partly because CNNs are effective image classification models, and their extension with BiGRU or LSTM layers, while sometimes increasing complexity and time, can be managed through proper hyperparameter tuning, such as employing methods like GridSearchCV.

In conclusion, the data engineering techniques employed in this research were instrumental in enhancing classification results and performance metrics. The initial preprocessing steps, including smoothing, making durations consistent through padding with silence, converting audio files into mel-scaled spectrograms, and applying SpecAugment augmentation, diversified the dataset. This ensured sufficient training data for the deep learning architectures. Employing PCA on the augmented dataset proved beneficial, as it reduced training duration by approximately 10-15 minutes in all cases. This reduction indicates that PCA effectively minimized non-essential features while retaining key characteristics for classification. The input features for all models were 16x16 patches, a standard size in image classification problems.

Overall, with appropriate data engineering processes, ViT and other CNN variants provide good overall test accuracy, reaffirming that CNN and its variants are well-suited for Audio Image Classification. Transformers, adapted to vision tasks, continue to be popular

architectures in the field and hold significant potential for experimentation in other domains, such as audio. This research validates the effectiveness of ViT against the ESC-50 and UrbanSound8k datasets, confirming the versatility and robustness of these architectures in audio classification tasks.

Limitations

There were quite a lot of difficulties while working on this research project, some of the most significant ones were to deal with the very minimally available labeled audio data, like ESC-50 and UrbanSound8k datasets. The choice of data augmentation was also tricky as the team had to experiment with various available techniques, and based on the accuracies the choice was decided to be SpecAugment, because of its simplicity and applications to the audio field. The biggest limitation was the non-availability of the high-computer resources, Google Collab Pro subscription and the high performance PC mitigated these effects, but it was still a significant limitation for this project.

Hyperparameter tuning techniques such as GridSearchCV can be computationally intensive, making it challenging to retrain models like ViT with various hyperparameters due to high computational demands. ViT is prone to overfitting, particularly with smaller datasets, highlighting the importance of data augmentation and feature reduction in this context. These steps are crucial for preventing overfitting, but they can also pose limitations when it comes to generalizing the model for new, unseen urban audio data.

Future Scope

Currently, this research employs augmentation methods like SpecAugment, Time Stretching, and Noise Induction. These could be enhanced using advanced techniques such as CycleGAN or other GAN architectures to generate augmented data. Such methods could better

preserve critical characteristics of audio files and offer more flexibility in adjusting augmentation parameters. The potential for experimentation is vast, and the study could be expanded to include more complex, overlapping sounds, like a scenario where a jackhammer is operating simultaneously with a car horn.

There's also room to explore various other feature engineering methods, such as Chromagrams, MFCC (Mel-frequency cepstral coefficients) vectors, waveforms, and constant-Q transform (CQT) waveforms. Customized experimentation at every stage is feasible for this research topic, opening up a broad and diverse range of future possibilities. The direction of future research can be highly varied and will largely depend on the specific use cases being targeted.

Adding to this, integrating advanced signal processing techniques and exploring different types of neural network architectures could further enhance model performance. Investigating the impact of different audio preprocessing methods, like filter banks or advanced noise reduction techniques, could also be insightful. Furthermore, the integration of multi-modal data, combining audio with other sensory inputs like visual or textual data, could lead to more robust and versatile models, particularly useful in complex urban environments. The development of real-time processing models for urban sound classification is another exciting avenue, with potential applications in smart city initiatives and environmental monitoring.

Appendix B

Preprocessing Code

Pre-processing of the audio files using Resampling at 44.1 khz, Smoothing using Spectral Gating, Padding the audio to make it consistent as 5 seconds audio file, then apply threshold level noise reduction to the audio files, this is done for all the audio files for ESC-50 and UrbanSound8k dataset. This is done for all the files in the directory and the process is mentioned in the below images of code snippets starting from Figure B1, B2, B3, B4, B5, B6, B7, B8.

Figure B1

Audio File Directory along Processing The Audio to Reduce The Noise by Padding

```
import os
import librosa
import numpy as np
import noisereduce as nr
import soundfile as sf

def read_and_process_audio(file_path, target_sr=44100):
    # Load and resample the audio file at 44.1 kHz
    audio, sr = librosa.load(file_path, sr=target_sr)

    # Noise reduction using spectral gating
    noise_clip = audio[:int(1.5 * sr)]
    audio = nr.reduce_noise(audio_clip=audio, noise_clip=noise_clip, verbose=False)

    # Padding the audio to 5 seconds if it's shorter
    target_length = 5 * sr  # 5 seconds at the sampling rate of the audio
    if len(audio) < target_length:
        padding_length = target_length - len(audio)
        audio = np.pad(audio, (0, padding_length), mode='constant')

    # Noise reduction using a threshold
    noise_threshold = 0.02  # This value might need tweaking
    audio = np.where(np.abs(audio) < noise_threshold, 0, audio)

    return audio, sr

def process_directory(input_dir, output_dir, target_sr=44100):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for file_name in os.listdir(input_dir):
        if file_name.lower().endswith('.wav'):
            file_path = os.path.join(input_dir, file_name)
            audio_processed, sampling_rate = read_and_process_audio(file_path, target_sr)

            # Save the processed audio
            output_file_path = os.path.join(output_dir, file_name)
            sf.write(output_file_path, audio_processed, sampling_rate)

# Example usage
input_directory = 'raw_audio'
output_directory = 'preprocessed_audio'
process_directory(input_directory, output_directory)
```

The next step is to use the preprocessed_audio files and extract mel-scaled spectrograms and then apply SpecAugment augmentations, with Time stretching, and noise inclusion. Then we apply Min-max normalization.

Figure B2

Mel-Spectrogram Creation Using Audio Files

```
import os
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import random

def create_mel_spectrogram(file_path, save_path):
    """
    Creates and saves a mel-scaled spectrogram from an audio file.

    Args:
    file_path (str): Path to the audio file.
    save_path (str): Path to save the spectrogram image.
    """
    # Load the audio file
    audio, sr = librosa.load(file_path, sr=None)

    # Create a mel-scaled spectrogram
    S = librosa.feature.melspectrogram(y=audio, sr=sr)
    S_DB = librosa.power_to_db(S, ref=np.max)

    # Convert to RGB and save the image
    plt.figure(figsize=(3, 3))
    librosa.display.specshow(S_DB, sr=sr, x_axis='time', y_axis='mel')
    plt.axis('off')
    plt.savefig(save_path, bbox_inches='tight', pad_inches=0)
    plt.close()
```

Figure B3

Applying Masking and Frequency to The Audio for Data Augmentation

```
def apply_time_masking(spectrogram, time_mask_param):
    """
    Applies time masking to a spectrogram.

    Args:
    spectrogram (numpy.ndarray): The spectrogram to which time masking will be applied.
    time_mask_param (int): The parameter for time masking.

    Returns:
    numpy.ndarray: The time-masked spectrogram.
    """
    num_masks = random.randint(1, 2)  # Number of masks to apply
    for _ in range(num_masks):
        t = random.randrange(0, time_mask_param)
        t_zero = random.randrange(0, spectrogram.shape[1] - t)
        spectrogram[:, t_zero:t_zero + t] = 0
    return spectrogram

def apply_frequency_masking(spectrogram, frequency_mask_param):
    """
    Applies frequency masking to a spectrogram.

    Args:
    spectrogram (numpy.ndarray): The spectrogram to which frequency masking will be applied.
    frequency_mask_paramb (int): The parameter for frequency masking.

    Returns:
    numpy.ndarray: The frequency-masked spectrogram.
    """
    num_masks = random.randint(1, 2)  # Number of masks to apply
    for _ in range(num_masks):
        f = random.randrange(0, frequency_mask_param)
        f_zero = random.randrange(0, spectrogram.shape[0] - f)
        spectrogram[f_zero:f_zero + f, :] = 0
    return spectrogram
```

These augmentation techniques are all employed on the preprocessed audio files, converting them into mel-scaled spectrograms and augmented mel-scaled spectrograms, this is done for all the files in the directory.

Figure B4

Saving MelScaleSpectogram from The Original Audio and Augmented Audio

```
def process_audio_directory(input_dir, output_dir_original, output_dir_augmented, time_mask_param, frequency_mask_param):
    """
    Processes all audio files in a directory, creating and saving both original and augmented mel-scaled spectrograms.

    Args:
    input_dir (str): Directory containing the audio files.
    output_dir_original (str): Directory to save original spectrograms.
    output_dir_augmented (str): Directory to save augmented spectrograms.
    time_mask_param (int): Parameter for time masking.
    frequency_mask_param (int): Parameter for frequency masking.
    """
    # Create output directories if they don't exist
    if not os.path.exists(output_dir_original):
        os.makedirs(output_dir_original)
    if not os.path.exists(output_dir_augmented):
        os.makedirs(output_dir_augmented)

    for file_name in os.listdir(input_dir):
        if file_name.lower().endswith('.wav'):
            file_path = os.path.join(input_dir, file_name)

            # Create and save original mel-scaled spectrogram
            original_spec_path = os.path.join(output_dir_original, file_name.replace('.wav', '.png'))
            create_mel_spectrogram(file_path, original_spec_path)

            # Create and save augmented mel-scaled spectrogram
            augmented_spec_path = os.path.join(output_dir_augmented, file_name.replace('.wav', '_augmented.png'))
            augment_spectrogram(file_path, augmented_spec_path, time_mask_param, frequency_mask_param)

# Example usage
input_directory = 'preprocessed_audio'
output_directory_original = 'mel_spectrograms'
output_directory_augmented = 'augmented_mel_spectrograms'
time_mask_param = 20  # This can be tweaked
frequency_mask_param = 20  # This can be tweaked
process_audio_directory(input_directory, output_directory_original, output_directory_augmented, time_mask_param,
                        frequency_mask_param)
```

Normalizing the augmented spectrogram with min-max normalization and storing the augmented normalized files.

Figure B5

Fetching the Stored Augmented Mel-Spectrograms to Normalize the Image Vector Values Without Changing the Audio Characteristics.

```python
from PIL import Image
import numpy as np
import os

def normalize_image(image_path, save_path):
    """
    Applies min-max normalization to an image and saves it.

    Args:
    image_path (str): Path to the input image.
    save_path (str): Path to save the normalized image.
    """
    # Open the image
    img = Image.open(image_path)
    img_np = np.array(img)

    # Apply min-max normalization
    norm_img_np = (img_np - img_np.min()) / (img_np.max() - img_np.min()) * 255
    norm_img = Image.fromarray(norm_img_np.astype('uint8'), 'RGB')

    # Save the normalized image
    norm_img.save(save_path)

def process_images_directory(input_dir, output_dir):
    """
    Processes all images in a directory, applying min-max normalization.

    Args:
    input_dir (str): Directory containing the images.
    output_dir (str): Directory to save normalized images.
    """
    # Create output directory if it doesn't exist
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for file_name in os.listdir(input_dir):
        if file_name.lower().endswith('.png'):
            file_path = os.path.join(input_dir, file_name)
            normalized_img_path = os.path.join(output_dir, file_name)
            normalize_image(file_path, normalized_img_path)

# Example usage
augmented_directory = 'augmented_mel_spectrograms'
normalized_directory = 'normalized_augmented_mel_spectrograms'
process_images_directory(augmented_directory, normalized_directory)
```

Now that the augmented Normalized mel-scaled spectrograms are ready, we apply stratified sampling to divide the datasets into valid and test.

Figure B6

Splitting the Audio Data for Training and Testing

```
import os
from sklearn.model_selection import train_test_split
import shutil

def split_dataset(input_dir, output_dir_train, output_dir_val, test_size=0.2):
    """
    Splits the dataset into training and validation sets using stratified sampling.

    Args:
    input_dir (str): Directory containing the dataset, organized by class.
    output_dir_train (str): Directory to save the training set.
    output_dir_val (str): Directory to save the validation set.
    test_size (float): Proportion of the dataset to include in the validation set.
    """
    # Create output directories if they don't exist
    if not os.path.exists(output_dir_train):
        os.makedirs(output_dir_train)
    if not os.path.exists(output_dir_val):
        os.makedirs(output_dir_val)

    # Gather all file paths and their corresponding class labels
    file_paths = []
    labels = []
    for class_dir in os.listdir(input_dir):
        class_dir_path = os.path.join(input_dir, class_dir)
        if os.path.isdir(class_dir_path):
            for file_name in os.listdir(class_dir_path):
                if file_name.lower().endswith('.png'):
                    file_paths.append(os.path.join(class_dir_path, file_name))
                    labels.append(class_dir)

    # Split the dataset into training and validation sets
    train_paths, val_paths, train_labels, val_labels = train_test_split(
        file_paths, labels, test_size=test_size, stratify=labels, random_state=42)

    # Function to copy files to their new location
    def copy_files(file_paths, labels, output_dir):
        for path, label in zip(file_paths, labels):
            class_dir = os.path.join(output_dir, label)
            if not os.path.exists(class_dir):
                os.makedirs(class_dir)
            shutil.copy2(path, class_dir)

    # Copy files to training and validation directories
    copy_files(train_paths, train_labels, output_dir_train)
    copy_files(val_paths, val_labels, output_dir_val)

# Example usage
input_directory = 'normalized_spectrograms'
train_directory = 'train_set'
val_directory = 'val_set'
split_dataset(input_directory, train_directory, val_directory, test_size=0.2)
```

The next step is to apply PCA dimension reduction techniques, and then re-construct the Mel-scaled spectrograms from the reduced PCA components.

Figure B7

Loading and Flattens the Image by Using Principal Component Analysis From the Directory

```python
import os
import numpy as np
from PIL import Image
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

def load_images_from_directory(directory):
    """
    Loads and flattens all images from a directory.

    Args:
    directory (str): The path to the directory containing images.

    Returns:
    numpy.ndarray: Array of flattened images.
    list: List of original image shapes.
    """
    images = []
    original_shapes = []
    for class_dir in os.listdir(directory):
        class_dir_path = os.path.join(directory, class_dir)
        if os.path.isdir(class_dir_path):
            for file_name in os.listdir(class_dir_path):
                if file_name.lower().endswith('.png'):
                    file_path = os.path.join(class_dir_path, file_name)
                    with Image.open(file_path) as img:
                        img_array = np.array(img)
                        original_shapes.append(img_array.shape)
                        images.append(img_array.flatten())
    return np.array(images), original_shapes
```

Figure B8

Reshaping the image according to the model and storing back to the directory in separate folder with the names of [reconstructed_FileName.png]

```python
def reconstruct_images(pca_data, original_shapes):
    """
    Reconstructs images from PCA-transformed data.

    Args:
    pca_data (numpy.ndarray): PCA-transformed data.
    original_shapes (list): List of original image shapes.

    Returns:
    list: Reconstructed images.
    """
    reconstructed = []
    for data, shape in zip(pca_data, original_shapes):
        img_array = data.reshape(shape)
        reconstructed.append(img_array)
    return reconstructed

def save_reconstructed_images(images, directory):
    """
    Saves reconstructed images to a directory.

    Args:
    images (list): List of reconstructed images.
    directory (str): The directory to save images.
    """
    if not os.path.exists(directory):
        os.makedirs(directory)

    for idx, image in enumerate(images):
        plt.imsave(os.path.join(directory, f'reconstructed_{idx}.png'), image)

# Load and flatten images
train_directory = 'train_set'
flattened_images, original_shapes = load_images_from_directory(train_directory)

# Apply PCA
pca = PCA(0.90)  # Retain 90% of variance
pca_images = pca.fit_transform(flattened_imagaes)

# Reconstruct images from PCA data
reconstructed_images = reconstruct_images(pca_images, original_shapes)

# Save reconstructed images
output_directory = 'pca_reconstructed_images'
save_reconstructed_images(reconstructed_images, output_directory)
```

Figure B9

One More Approach to Fetch the Mel-Spectogram Features From Audio Files

```
#D = [] # Dataset
count = len(D)  # Initialize count with the current length of the dataset D

# Loop through each row in the 'valid_data' DataFrame
for row in valid_data.itertuples():
    # Load the audio file specified in 'row.path' from the 'UrbanSound8K/audio/' directory
    # The audio is trimmed or padded to 2.97 seconds
    y, sr = librosa.load('UrbanSound8K/audio/' + row.path, duration=2.97)

    # Generate a Mel-scaled power (energy-squared) spectrogram for the audio signal
    ps = librosa.feature.melspectrogram(y=y, sr=sr)

    # Check if the spectrogram shape is 128x128. If not, skip the current iteration
    if ps.shape != (128, 128): continue

    count += 1  # Increment count for each valid spectrogram

    # Append a tuple of the spectrogram and its corresponding class ID to the dataset 'D'
    D.append((ps, row.classID))

    # Print the current count
    print(count)
```

The next step is to use these re-constructed images into Deep learning models.

Appendix C

Code for Implementing Models

The implementation of each ViT, CNN with BiGRU , CNN, CNN with LSTM are in the Figures C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16,C17,C18,C19

Figure C1

ViT Implementation

```
#this step is to check for the GPU, use it if available or move on

import requests
import torch
from PIL import Image
from transformers import *
from tqdm import tqdm

device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
import requests
from tqdm import tqdm
import zipfile
import os


def get_file(url):
  response = requests.get(url, stream=True)
  total_size = int(response.headers.get('content-length', 0))
  filename = None
  content_disposition = response.headers.get('content-disposition')
  if content_disposition:
      parts = content_disposition.split(';')
      for part in parts:
          if 'filename' in part:
              filename = part.split('=')[1].strip('"')
  if not filename:
      filename = os.path.basename(url)
  block_size = 1024 # 1 Kibibyte
  tqdm_bar = tqdm(total=total_size, unit='iB', unit_scale=True)
  with open(filename, 'wb') as file:
      for data in response.iter_content(block_size):
          tqdm_bar.update(len(data))
          file.write(data)
  tqdm_bar.close()
  print(f"Downloaded {filename} ({total_size} bytes)")
  return filename

def download_and_extract_dataset():
  # dataset from https://github.com/udacity/dermatologist-ai
  # 5.3GB
  train_url = "https://s3-us-west-1.amazonaws.com/udacity-dlnfd/datasets/skin-cancer/train.zip"
  # 824.5MB
  valid_url = "https://s3-us-west-1.amazonaws.com/udacity-dlnfd/datasets/skin-cancer/valid.zip"
  # 5.1GB
  test_url  = "https://s3-us-west-1.amazonaws.com/udacity-dlnfd/datasets/skin-cancer/test.zip"
  for i, download_link in enumerate([valid_url, train_url, test_url]):
    data_dir = get_file(download_link)
    print("Extracting", download_link)
    with zipfile.ZipFile(data_dir, "r") as z:
      z.extractall("data")
    # remove the temp file
    os.remove(data_dir)

# comment the below line if you already downloaded the dataset
# download_and_extract_dataset()
```

Figure C2

Initial Data loading to ViT Model

```
# the model name
# model_name = "google/vit-base-patch16-224"
# model_name = "google/vit-base-patch16-224-in21k"
model_name= 'facebook/deit-base-patch16-224'
# load the image processor
image_processor = ViTImageProcessor.from_pretrained(model_name)
# loading the pre-trained model
model = ViTForImageClassification.from_pretrained(model_name).to(device)
```

```
from datasets import load_dataset

# load the custom dataset
ds = load_dataset("imagefolder", data_dir="/content/drive/Shareddrives/DATA 270/ESC Cross-Validation Dataset/ESC-50-master/data_
```

```
ds
labels = ds["train"].features["label"]
labels
testlabels = ds["test"].features["label"]
testlabels
```

Figure C3

Extraction the Labels for Our Dataset

```
def transform(examples):
  # convert all images to RGB format, then preprocessing it
  # using our image processor
  inputs = image_processor([img.convert("RGB") for img in examples["image"]], return_tensors="pt")
  # we also shouldn't forget about the labels
  inputs["labels"] = examples["label"]
  return inputs

# use the with_transform() method to apply the transform to the dataset on the fly during training
dataset = ds.with_transform(transform)

for item in dataset["train"]:
  print(item["pixel_values"].shape)
  print(item["labels"])
  break

import torch

def collate_fn(batch):
  return {
      "pixel_values": torch.stack([x["pixel_values"] for x in batch]),
      "labels": torch.tensor([x["labels"] for x in batch]),
  }

# extract the labels for our dataset
labels = ds["train"].features["label"].names
print(len(labels))
labels
```

Figure C4

Classification Metrics - Confusion Matrix

```
#Defiinting the classification metrics
```

```
from evaluate import load
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Load the metrics
accuracy = load("accuracy")
f1 = load("f1")
precision = load("precision")
recall = load("recall")

def compute_metrics(eval_pred):
    predictions = np.argmax(eval_pred.predictions, axis=1)
    references = eval_pred.label_ids

    # Compute scores
    accuracy_score = accuracy.compute(predictions=predictions, references=references)
    f1_score = f1.compute(predictions=predictions, references=references, average="macro")
    precision_score = precision.compute(predictions=predictions, references=references, average="macro")
    recall_score = recall.compute(predictions=predictions, references=references, average="macro")

    # Confusion matrix
    cm = confusion_matrix(references, predictions)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='g')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()

    return {**accuracy_score, **f1_score, **precision_score, **recall_score}

# Example usage
# results = compute_metrics(eval_pred)
```

Figure C5

Load ViT Model and Initializing parameters for the model

```
# load the ViT model
model = ViTForImageClassification.from_pretrained(
    model_name,
    num_labels=len(labels),
    id2label={str(i): c for i, c in enumerate(labels)},
    label2id={c: str(i) for i, c in enumerate(labels)},
    ignore_mismatched_sizes=True,
)
```

```
#initializing the training parameters for the model
from transformers import TrainingArguments

training_args = TrainingArguments(
  # output_dir="./vit-base-food", # output directory
  output_dir="./vit-base-audio-classify",
  per_device_train_batch_size=32, # batch size per device during training
  evaluation_strategy="epoch",    # evaluation strategy to adopt during training
  num_train_epochs=15,             # total number of training epochs
  # fp16=True,                    # use mixed precision
  # save_steps=1000,                # number of update steps before saving checkpoint
  # eval_steps=1000,                # number of update steps before evaluating
  # logging_steps=1000,             # number of update steps before logging
  save_steps=100,
  eval_steps=100,
  logging_steps=100,
  save_total_limit=2,             # limit the total amount of checkpoints on disk
  remove_unused_columns=False,    # remove unused columns from the dataset
  push_to_hub=False,              # do not push the model to the hub
  report_to='tensorboard',        # report metrics to tensorboard
  load_best_model_at_end=True,    # load the best model at the end of training
  save_strategy = 'epoch',
)
```

```
# start training
trainer.train()
```

```
# trainer.evaluate(dataset["test"])
trainer.evaluate()
```

```
trainer.evaluate(dataset["test"])
```

Checking for the inference against a test spectrogram.

Figure C6

Loading the Image and Get Prediction From the Pretrained Model

```
# load the best model, change the checkpoint number to the best checkpoint
# if the last checkpoint is the best, then ignore this cell
best_checkpoint = 400
# best_checkpoint = 150
# model = ViTForImageClassification.from_pretrained(f"./vit-base-food/checkpoint-{best_checkpoint}").to(device)
model = ViTForImageClassification.from_pretrained(f"./vit-base-audio-classify/checkpoint-{best_checkpoint}").to(device)

def load_image(image_path):
  if os.path.exists(image_path):
    return Image.open(image_path).convert("RGB")

def get_prediction(model, url_or_path):
  # load the image
  img = load_image(url_or_path)
  print(type(img))
  # preprocessing the image
  pixel_values = image_processor(img, return_tensors="pt")["pixel_values"].to(device)
  # perform inference
  output = model(pixel_values)
  # get the label id and return the class name
  return model.config.id2label[int(output.logits.softmax(dim=1).argmax())]

def get_prediction_probs(model, url_or_path, num_classes=50):
    # load the image
    img = load_image(url_or_path)
    # preprocessing the image
    pixel_values = image_processor(img, return_tensors="pt")["pixel_values"].to(device)
    # perform inference
    output = model(pixel_values)
    # get the top k classes and probabilities
    probs, indices = torch.topk(output.logits.softmax(dim=1), k=num_classes)
    # get the class labels
    id2label = model.config.id2label
    classes = [id2label[idx.item()] for idx in indices[0]]
    # convert the probabilities to a list
    probs = probs.squeeze().tolist()
    # create a dictionary with the class names and probabilities
    results = dict(zip(classes, probs))
    return results

# example 1
get_prediction_probs(model,"/content/audio_data/data/test/crying_baby/1-22694-B-20.png")
```

Figure C7

CNN With BiGRU Implementation

```
# CNN with BiGRU Implementation

# Import necessary modules from TensorFlow's Keras API
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Bidirectional, GRU, Dense, Activation, Flatten, Dropout, BatchNormaliz
from tensorflow.keras.layers import Reshape

# Create a sequential model
model = Sequential()

# Define the input shape assuming grayscale images of size 128x128 pixels
input_shape = (128, 128, 1)

# Convolutional layers to extract spatial features
model.add(Conv2D(24, (5, 5), strides=(1, 1), input_shape=input_shape))
# Batch normalization and max pooling to enhance training stability and reduce spatial dimensions
# model.add(BatchNormalization())
model.add(MaxPooling2D((4, 2), strides=(4, 2)))
model.add(Activation('relu'))

model.add(Conv2D(48, (5, 5), padding="valid"))
# Batch normalization and max pooling
# model.add(BatchNormalization())
model.add(MaxPooling2D((4, 2), strides=(4, 2)))
model.add(Activation('relu'))

# Reshape for Bidirectional GRU layer, assuming 48 channels
model.add(Reshape((-1, 48)))

# Bidirectional GRU layer for capturing temporal dependencies
model.add(Bidirectional(GRU(48, return_sequences=True)))
# Batch normalization for normalization between GRU layers
# model.add(BatchNormalization())
model.add(Activation('relu'))

# Flatten the output for fully connected layers
model.add(Flatten())

# Dropout for regularization to prevent overfitting
model.add(Dropout(rate=0.5))

# Fully connected layers for classification
model.add(Dense(64))
# Batch normalization for normalization between fully connected layers
# model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(rate=0.5))

# Output layer with softmax activation for multiclass classification
model.add(Dense(10))
model.add(Activation('softmax'))
```

Below Figure C8 shows the reshape the data for the model to train and test

Figure C8

Splitting the Data For The Model

```
#Model Training & Testing :
dataset = D
random.shuffle(dataset)

train = dataset[:41790]
test = dataset[41790:]

X_train, y_train = zip(*train)
X_test, y_test = zip(*test)

X_train = np.array([x.reshape( (128, 128, 1) ) for x in X_train])
X_test = np.array([x.reshape( (128, 128, 1) ) for x in X_test])

y_train = np.array(keras.utils.to_categorical(y_train, 10))
y_test = np.array(keras.utils.to_categorical(y_test, 10))
```

Figure C9

Model Compilation and Evaluation

```
model.compile(
    optimizer="Adam",  #Change
    loss="categorical_crossentropy",
    metrics=['accuracy'])

model.fit(
    x=X_train,
    y=y_train,
    epochs=25,
    batch_size=128,
    validation_data= (X_test, y_test))

score = model.evaluate(
    x=X_test,
    y=y_test)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Figure C10

Model EarlyStopping and checkpoint

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# Define ModelCheckpoint callback
checkpoint_callback = ModelCheckpoint(
    filepath='best_model_weights_2.h5',  # Filepath to save the best weights
    monitor='val_accuracy',  # Monitor validation accuracy
    save_best_only=True,  # Save only the best weights
    mode='max',  # Mode can be 'max' or 'min' depending on what you are monitoring
    verbose=1  # Display messages about the saving process
)

# Define EarlyStopping callback
early_stopping_callback = EarlyStopping(
    monitor='val_accuracy',
    patience=5,  # Number of epochs with no improvement after which training will be stopped
    verbose=1,
    restore_best_weights=True
)

# Compile the model
model.compile(
    optimizer="Adam",  # Change
    loss="categorical_crossentropy",
    metrics=['accuracy']
)

# Train the model with callbacks
history = model.fit(
    x=X_train,
    y=y_train,
    epochs=50,
    batch_size=128,
    validation_data=(X_test, y_test),
    callbacks=[checkpoint_callback, early_stopping_callback]  # Add the callbacks here
)

# Evaluate the model
score = model.evaluate(
    x=X_test,
    y=y_test
)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Figure C11

Performing the GridSearchCV for HyperParameter tuning similar approach can be used to CNN

```
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Bidirectional, GRU, Dense, Activation, Flatten, Dropout, BatchNormaliza
from tensorflow.keras.optimizers import Adam
# Import scikeras instead of tensorflow.keras.wrappers
from scikeras.wrappers import KerasClassifier
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.model_selection import StratifiedKFold
import numpy as np
# Function to create the model with hyperparameters
def create_model(optimizer='adam', dropout_rate_cnn=0.5, dropout_rate_dense=0.5, gru_units=48):
    model = Sequential()

    input_shape = (128, 128, 1)

    # Convolutional layers
    model.add(Conv2D(24, (5, 5), strides=(1, 1), input_shape=input_shape))
    model.add(MaxPooling2D((4, 2), strides=(4, 2)))
    model.add(Activation('relu'))

    model.add(Conv2D(48, (5, 5), padding="valid"))
    model.add(MaxPooling2D((4, 2), strides=(4, 2)))
    model.add(Activation('relu'))

    # Reshape for Bidirectional GRU layer
    model.add(Reshape((-1, 48)))

    # Bidirectional GRU layer
    model.add(Bidirectional(GRU(gru_units, return_sequences=True)))
    model.add(Activation('relu'))

    model.add(Flatten())
    model.add(Dropout(rate=dropout_rate_cnn))

    # Fully connected layers
    model.add(Dense(64))
    model.add(Activation('relu'))
    model.add(Dropout(rate=dropout_rate_dense))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

Figure C12

Hyperparameter Continuation

```
# Convert labels to one-hot encoding
#y_train_onehot = np.eye(10)[y_train.astype(int)] # Assuming there are 10 classes

# Use StratifiedKFold for better cross-validation with classification problems
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# Create KerasClassifier
model = KerasClassifier(build_fn=create_model, epochs=50, batch_size=128, verbose=0)

# Define the hyperparameters to search
# Add the prefix "model__" to the parameters that belong to the Keras model
param_grid = {
    'model__optimizer': ['adam', 'rmsprop'],
    'model__dropout_rate_cnn': [0.3, 0.5, 0.7],
    'model__dropout_rate_dense': [0.3, 0.5, 0.7],
    'model__gru_units': [32, 48, 64]
}
# Create GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=cv, scoring='accuracy', n_jobs=-1)
grid_result = grid.fit(X_train, y_train)

# Print the best results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

# Get the best model
best_model = grid_result.best_estimator_.model

# Now you can train this best model on your entire training set or use it for predictions
best_model.fit(X_train, y_train_onehot)

# Evaluate the best model on the test set
test_score = best_model.evaluate(X_test, y_test)
print('Test loss:', test_score[0])
print('Test accuracy:', test score[1])
```

Figure C13

Finding the Confusion Matrix at the eEnd

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Predictions on test set
y_pred = model.predict(X_test)

# Convert one-hot encoded labels back to integers
y_test_int = np.argmax(y_test, axis=1)
y_pred_int = np.argmax(y_pred, axis=1)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test_int, y_pred_int)
print('Confusion Matrix:')
print(conf_matrix)

# Visualize Confusion Matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=range(10), yticklabels=range(10))
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
from sklearn.metrics import roc_auc_score, f1_score, confusion_matrix, precision_score, mean_squared_error
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
from scipy import interp

# Predictions on test set
y_pred = model.predict(X_test)

# Convert one-hot encoded labels back to integers
y_test_int = np.argmax(y_test, axis=1)
y_pred_int = np.argmax(y_pred, axis=1)

# ROC AUC
roc_auc = roc_auc_score(y_test, y_pred, average='macro')
print('ROC AUC:', roc_auc)

# F1 Score
f1 = f1_score(y_test_int, y_pred_int, average='macro')
print('F1 Score:', f1)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test_int, y_pred_int)
print('Confusion Matrix:')
print(conf_matrix)

# Precision Matrix
precision = precision_score(y_test_int, y_pred_int, average=None)
print('Precision Matrix:')
print(precision)

# MSE and RMSE
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)

# Plot ROC Curve
n_classes = 10  # Number of classes
fpr = dict()
tpr = dict()
roc_auc = dict()

# Binarize the labels
y_test_bin = label_binarize(y_test_int, classes=list(range(n_classes)))

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_pred.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

Figure C14

Finding the ROC Curve

```
# Plot ROC curves
plt.figure()
lw = 2
plt.plot(fpr["micro"], tpr["micro"], color='darkorange', lw=lw, label='ROC curve (area = {:.2f})'.format(roc_auc["micro"]))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=lw, label='ROC curve of class {} (area = {:.2f})'.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

Figure C15

Finding the Accuracy, Precision, Recall and F1 Score

```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Evaluate the model
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Print confusion matrix
conf_mat = confusion_matrix(y_true, y_pred_classes)
print("Confusion Matrix:")
print(conf_mat)

# Print classification report
class_report = classification_report(y_true, y_pred_classes)
print("Classification Report:")
print(class_report)

# Extract metrics
accuracy = accuracy_score(y_true, y_pred_classes)
precision = precision_score(y_true, y_pred_classes, average='weighted')
recall = recall_score(y_true, y_pred_classes, average='weighted')
f1 = f1_score(y_true, y_pred_classes, average='weighted')

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

Figure C16

KFold Cross-Verification k=10

```python
from sklearn.model_selection import KFold
import numpy as np
import keras.utils

# Assign the previously prepared dataset 'D' to a new variable 'dataset'
dataset = np.array(D)  # Convert dataset to a NumPy array for easier handling

# Create a KFold object for 10-fold cross-validation
kf = KFold(n_splits=10, shuffle=False)

# Initialize a list to store accuracy for each fold
fold_accuracies = []

# Iterate over each fold
for train_index, test_index in kf.split(dataset):
    # Split the dataset into training and testing sets for the current fold
    train, test = dataset[train_index], dataset[test_index]

    # Unzip the train and test datasets into features (X) and labels (y)
    X_train, y_train = zip(*train)
    X_test, y_test = zip(*test)

    # Reshape the feature data for training and testing
    X_train = np.array([x.reshape((128, 128, 1)) for x in X_train])
    X_test = np.array([x.reshape((128, 128, 1)) for x in X_test])

    # Convert the training and testing label data into categorical format
    y_train = np.array(keras.utils.to_categorical(y_train, 10))
    y_test = np.array(keras.utils.to_categorical(y_test, 10))

    # Here, you would train your model using X_train and y_train
    # model.fit(X_train, y_train)

    # And then test the model using X_test and y_test
    # accuracy = model.evaluate(X_test, y_test)

    # Store the accuracy for each fold
    fold_accuracies.append(accuracy)

# After all folds are processed, fold_accuracies will contain the accuracy of each fold
```

Figure C17

CNN+LSTM Implementation

```python
class CNNWithLSTM(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=2, out_channels=24, kernel_size=(6,6), stride=1),
            nn.BatchNorm2d(24),
            nn.ReLU(),
            nn.Dropout(0.2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=24, out_channels=24, kernel_size=(6,6), stride=1),
            nn.BatchNorm2d(24),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.2),
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=24, out_channels=48, kernel_size=(5,5), stride=(2,2)),
            nn.BatchNorm2d(48),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.2),
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=48, out_channels=48, kernel_size=(5,5), stride=(2,2)),
            nn.BatchNorm2d(48),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.2),
        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(in_channels=48, out_channels=64, kernel_size=(4,4), stride=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.2),
        )
        self.conv6 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=(4,4), stride=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.2),
        )
        self.connected_layer = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64*4*43, 200),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(200, 50),
            nn.Softmax(),
        )

        ## ading a LSTM here

        self.lstm = nn.LSTM(input_size=64, hidden_size=128, num_layers=2, batch_first=True, dropout=0.2)


    def forward(self, input_data):
        input_data = input_data.to('cpu')
        x = self.conv1(input_data)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.conv6(x)
        # Flatten before LSTM
        x = x.view(x.size(0), -1)

        # Pass through LSTM
        lstm_out, _ = self.lstm(x.view(len(x), 1, -1))

        # # Flatten before fully connected layers
        lstm_out = lstm_out[:, -1, :]  # Taking the last timestep output

        # # Reshape the output of LSTM to match the input size of fully connected layers
        lstm_out = lstm_out.view(lstm_out.size(0), -1)
        x = self.connected_layer(x)
        return x
```

Figure C18

CNN Model Implementation

```
# Import necessary modules from Keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense

# Initialize a Sequential model
model = Sequential()
# Define the input shape for the first layer
input_shape=(128, 216, 1)

# Add a 2D Convolutional layer with 24 filters, each with a size of 5x5
# Stride is set to (1, 1), and input shape is specified for the first layer
model.add(Conv2D(24, (5, 5), strides=(1, 1), input_shape=input_shape))
# Add a Max Pooling layer that reduces the spatial dimensions (height and width) of the output
# Pooling size is set to (4, 2) with strides of (4, 2)
model.add(MaxPooling2D((4, 2), strides=(4, 2)))
# Add an Activation layer with 'relu' activation function (Rectified Linear Unit)
model.add(Activation('relu'))

# Add another 2D Convolutional layer, this time with 48 filters, each of size 5x5
# Padding is set to "valid", which means no padding (i.e., border pixels are not considered)
model.add(Conv2D(48, (5, 5), padding="valid"))
# Add another Max Pooling layer with the same pooling size and strides as before
model.add(MaxPooling2D((4, 2), strides=(4, 2)))
# Add another 'relu' activation layer
model.add(Activation('relu'))

# Add a third 2D Convolutional layer with 48 filters and "valid" padding
model.add(Conv2D(48, (5, 5), padding="valid"))
# Add another 'relu' activation layer
model.add(Activation('relu'))

# Flatten the output of the previous layer
# This is necessary to transition from 2D layers to 1D layers
model.add(Flatten())
# Add a Dropout layer to prevent overfitting by randomly setting a fraction of inputs to 0
# Dropout rate is set to 50%
model.add(Dropout(rate=0.5))

# Add a Dense (fully connected) layer with 64 units
model.add(Dense(64))
# Add another 'relu' activation layer
model.add(Activation('relu'))
# Add another Dropout layer with a rate of 50%
model.add(Dropout(rate=0.5))

# Add a Dense layer with 50 units
model.add(Dense(50))
# Add a softmax activation layer
# Softmax is often used in the output layer for multi-class classification
model.add(Activation('softmax'))
```

Figure C19

CNN Model Training and Evaluation

```
model.compile(
    optimizer="Adam",
    loss="categorical_crossentropy",
    metrics=['accuracy'])

model.fit(
    x=X_train,
    y=y_train,
    epochs=20,
    batch_size=128,
    validation_data= (X_test, y_test))

score = model.evaluate(
    x=X_test,
    y=y_test)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

References

Abdel-Hamid, O., Mohamed, A. R., Jiang, H., Deng, L., Penn, G., & Yu, D. (2014, October 1). *Convolutional neural networks for speech recognition*. IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/document/6857341

Laith,A., Zhang, J., Humaidi, A. J., Al-Dujaili, A. Q., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1). https://doi.org/10.1186/s40537-021-00444-8

Arnault, A. (2020, October 22). *Urban Sound Classification : striving towards a fair comparison*. arXiv.org. https://arxiv.org/abs/2010.11805

Arnault, A., Hanssens, B., & Riche, N. (2022). Urban Sound Classification : striving towards a fair comparison. *Arxiv*, arXiv:2010.11805v1.

Ashraf, M., Abid, F., Din, I. U., Rasheed, J., Yeşiltepe, M., Yeo, S. F., & Ersoy, M. T. (2023). A hybrid CNN and RNN variant model for music classification. *Applied Sciences*, *13*(3), 1476. https://doi.org/10.3390/app13031476

Bahmei, B., Birmingham, E., & Arzanpour, S. (2022). CNN-RNN and data augmentation using deep convolutional generative adversarial network for environmental sound classification. *IEEE Signal Processing Letters*, *29*, 682–686. https://doi.org/10.1109/lsp.2022.3150258

Barchiesi, D., Giannoulis, D., Stowell, D., & Plumbley, M. D. (2015). Acoustic Scene Classification: Classifying environments from the sounds they produce. *IEEE Signal Processing Magazine*, *32*(3), 16–34. https://doi.org/10.1109/msp.2014.2326181

Bubashait, M., & Hewahi, N. M. (2021). Urban Sound Classification Using DNN, CNN & LSTM a Comparative Approach. *2021 International Conference on Innovation and*

Intelligence for Informatics, Computing, and Technologies (3ICT). https://doi.org/10.1109/3ict53449.2021.9581339

Chhikara, J. (n.d.). Transfer Learning Models Based Environment Audio Classification. *International Journal of Emerging Technologies in Engineering Research (IJETER)*, https://www.ijeter.everscience.org/Manuscripts/Volume-9/Issue-1/Vol-9-issue-1-M-01.

Das, J. K., Ghosh, A., Pal, A. K., Dutta, S., & Chakrabarty, A. (2020). Urban Sound Classification Using Convolutional Neural Network and Long Short Term Memory Based on Multiple Features. *IEEEXPLORE*. https://doi.org/10.1109/icds50568.2020.9268723

Demir, F., Türkoğlu, M., Aslan, M., & Şengür, A. (2020). A new pyramidal concatenated CNN approach for environmental sound classification. *Applied Acoustics*, *170*, 107520. https://doi.org/10.1016/j.apacoust.2020.107520

Díez, I., Saratxaga, I., Salegi, U., Navas, E., & Hernáez, I. (2023). NoiSenseDB: An Urban Sound Event database to develop neural classification systems for Noise-Monitoring Applications. *Applied Sciences*, *13*(16), 9358. https://doi.org/10.3390/app13169358

Domazetovska, S. (2022). *Urban sound recognition using different feature extraction techniques*. https://scholar.google.com/citations?view_op=view_citation&hl=en&user=pj5KjnIAAAAJ&citation_for_view=pj5KjnIAAAAJ:Zph67rFs4hoC

Dosovitskiy, A. (2020, October 22). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv.org. https://arxiv.org/abs/2010.11929

Fonseca, E., Favory, X., Pons, J., Font, F., & Serra, X. (2022). FSD50K: an open dataset of Human-Labeled Sound Events. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *30*, 829–852. https://doi.org/10.1109/taslp.2021.3133208

Gao, Z., Liu, T., Zhu, M., Li, J., Ning, Y., & Wang, Z. (2023). Environmental Sound Classification Using CNN Based on Mel-spectogram. *IEEEXPLORE*. https://doi.org/10.1109/aibt57480.2023.00015

Gazneli, A. (2022, April 25). *End-to-End audio strikes back: boosting augmentations towards an efficient audio classification network*. arXiv.org. https://arxiv.org/abs/2204.11479

Gong, Y., Chung, Y., & Glass, J. (2021). AST: Audio Spectrogram Transformer. *Arxiv*. https://doi.org/10.21437/interspeech.2021-698

Jain, A., Samala, P. R., Mittal, D., Jyothi, P., & Singh, M. (2022). SPLICEOUT: a simple and efficient audio augmentation method. *Interspeech 2022*. https://doi.org/10.21437/interspeech.2022-572

Kadhum, M., Manaseer, S., & Dalhoum, A. L. A. (2021a). Evaluation Feature Selection Technique on Classification by Using Evolutionary ELM Wrapper Method with Features Priorities. *Journal of Advances in Information Technology*, *12*(1), 21–28. https://doi.org/10.12720/jait.12.1.21-28

Kadhum, M., Manaseer, S., & Dalhoum, A. L. A. (2021b). Evaluation Feature Selection Technique on Classification by Using Evolutionary ELM Wrapper Method with Features Priorities. *Journal of Advances in Information Technology*, *12*(1), 21–28. https://doi.org/10.12720/jait.12.1.21-28

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., & Shah, M. (2022). Transformers in Vision: a survey. *ACM Computing Surveys*, *54*(10s), 1–41. https://doi.org/10.1145/3505244

Koutini, K., Schlüter, J., Eghbal-Zadeh, H., & Widmer, G. (2022). Efficient Training of Audio Transformers with Patchout. *Interspeech 2022*. https://doi.org/10.21437/interspeech.2022-227

Lamrini, M., Chkouri, M. Y., & Touhafi, A. (2023). Evaluating the performance of Pre-Trained Convolutional Neural Network for audio classification on embedded systems for anomaly detection in smart cities. *Sensors*, *23*(13), 6227. https://doi.org/10.3390/s23136227

Li, J. B. (2022, March 25). *AudioTagging Done Right: 2nd comparison of deep learning methods for environmental sound classification*. arXiv.org. https://arxiv.org/abs/2203.13448

Lim, K. M., Lee, C. P., Lee, Z., & Alqahtani, A. (2023). EnViTSA: Ensemble of Vision Transformer with SpecAugment for Acoustic Event Classification. *Sensors*, *23*(22), 9084. https://doi.org/10.3390/s23229084

Liu, X., Lu, H., Yuan, J., & Li, X. (2023). CAT: Causal Audio Transformer for Audio Classification. *IEEEXPLORE*. https://doi.org/10.1109/icassp49357.2023.10096787

Luz, J. S., Oliveira, M. C., Araújo, F. H. D., & Magalhães, D. (2021). Ensemble of handcrafted and deep features for urban sound classification. *Applied Acoustics*, *175*, 107819. https://doi.org/10.1016/j.apacoust.2020.107819

Massoudi, M., Verma, S., & Jain, R. (2021). Urban Sound Classification using CNN. *International Conference on Inventive Computation Technologies (ICICT 2021)*. https://doi.org/10.1109/icict50816.2021.9358621

Mekruksavanich, S., & Jitpattanakul, A. (2021). Deep Convolutional Neural Network with RNNs for Complex Activity Recognition Using Wrist-Worn Wearable Sensor Data. *Electronics*, *10*(14), 1685. https://doi.org/10.3390/electronics10141685

Mkrtchian, G., & Furletov, Y. (2022). Classification of Environmental Sounds Using Neural Networks. *IEEEXPLORE*. https://doi.org/10.1109/synchroinfo55067.2022.9840922

Mo, R., & Lam, A. Y. S. (2020). Augmenting Telephony Audio Data using Robust Principal Component Analysis. *IEEEXPLORE*. https://doi.org/10.1109/ssci47803.2020.9308406

Mushtaq, Z., Su, S., & Tran, Q. (2021). Spectral images based environmental sound classification using CNN with meaningful data augmentation. *Applied Acoustics*, *172*, 107581. https://doi.org/10.1016/j.apacoust.2020.107581

Nogueira, A. F. R., Oliveira, H. S., Machado, J., & Tavares, J. M. R. S. (2022a). Transformers for Urban Sound Classification—A Comprehensive Performance Evaluation. *Sensors*, *22*(22), 8874. https://doi.org/10.3390/s22228874

Nogueira, A. F. R., Oliveira, H. S., Machado, J., & Tavares, J. M. R. S. (2022b). Transformers for Urban Sound Classification—A Comprehensive Performance Evaluation. *Sensors*, *22*(22), 8874. https://doi.org/10.3390/s22228874

Park, D., Chan, W., Zhang, Y., Chiu, C., Zoph, B., Cubuk, E. D., & Le, Q. V. (2019). SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. *Arxiv*. https://doi.org/10.21437/interspeech.2019-2680

Piczak, K. J. (2015). ESC: Dataset for Environmental Sound Classification. *ACM Digital Library*. https://doi.org/10.1145/2733373.2806390

Rahman, F., Mubarek, Ö., & Kira, Z. (2022). On the Surprising Effectiveness of Transformers in Low-Labeled Video Recognition. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2209.07474

Saeed, N., Nyberg, R. G., Alam, M., Dougherty, M., Jooma, D., & Rebreyend, P. (2021a). Classification of the acoustics of loose gravel. *Sensors*, *21*(14), 4944. https://doi.org/10.3390/s21144944

Saeed, N., Nyberg, R. G., Alam, M., Dougherty, M., Jooma, D., & Rebreyend, P. (2021b). Classification of the acoustics of loose gravel. *Sensors*, *21*(14), 4944. https://doi.org/10.3390/s21144944

Sainath, T. N., Vinyals, O., W, A., Senior, & Sak, H. (2015). Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks. *IEEEXPLORE*. https://doi.org/10.1109/icassp.2015.7178838

Sak, H., W, A., Senior, & Beaufays, F. (2014). Long Short-Term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv (Cornell University)*. http://export.arxiv.org/pdf/1402.1128

Salamon, J., & Bello, J. P. (2017). Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal Processing Letters*, *24*(3), 279–283. https://doi.org/10.1109/lsp.2017.2657381

Salamon, J., Jacoby, C., & Bello, J. P. (2014). A Dataset and Taxonomy for Urban Sound Research. *ACM Digital Library*. https://doi.org/10.1145/2647868.2655045

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958. https://jmlr.csail.mit.edu/papers/volume15/srivastava14a/srivastava14a.pdf

Toth, L. (2014). Combining time- and frequency-domain convolution in convolutional neural network-based phone recognition. *IEEEXPLORE*. https://doi.org/10.1109/icassp.2014.6853584

Turab, M., Kumar, T. A., Bendechache, M., & Saber, T. (2022). Investigating multi-feature selection and ensembling for audio classification. *International Journal of Artificial Intelligence & Applications*, *13*(3), 69–84. https://doi.org/10.5121/ijaia.2022.13306

Vaswani, A. (2017, June 12). *Attention is all you need*. arXiv.org. https://arxiv.org/abs/1706.03762

Wang, C., Song, Y., Liu, H., Liu, H., Liu, J., Li, B., & Yuan, X. (2022). Real-Time vehicle sound detection system based on depthwise separable convolution neural network and spectrogram augmentation. *Remote Sensing*, *14*(19), 4848. https://doi.org/10.3390/rs14194848

Xueli, M., Ablimit, M., & Hamdulla, A. (2021, July 16). *Multiclass Language Identification Using CNN-Bigru-Attention Model on Spectrogram of Audio Signals*. IEEE Conference Publication | IEEE Xplore. Retrieved December 8, 2023, from https://ieeexplore.ieee.org/document/9520702/authors#authors

Yadav, H., Shah, P., Gandhi, N. R., Vyas, T., Nair, A., Desai, S., Gohil, L., Tanwar, S., Sharma, R., Verdeș, M., & Răboacă, M. S. (2023). CNN and Bidirectional GRU-Based Heartbeat Sound Classification Architecture for Elderly People. *Mathematics*, *11*(6), 1365. https://doi.org/10.3390/math11061365

Yazgaç, B. G., & Kırcı, M. (2022). Fractional-Order Calculus-Based Data Augmentation Methods for Environmental Sound Classification with Deep Learning. *Fractal and Fractional*, *6*(10), 555. https://doi.org/10.3390/fractalfract6100555

Yuyu, M., Chang, C., Huo, J., Zhang, Y., Al-Neshmi, H. M. M., Xu, J., & Xie, T. (2022). Research on Ultra-Short-Term prediction model of wind power based on attention mechanism and CNN-BIGRU combined. *Frontiers in Energy Research*, *10*. https://doi.org/10.3389/fenrg.2022.920835

Zeng, C., Zhu, D., Wang, Z., Wu, M., Xiong, W., & Zhao, N. (2021). Spatial and temporal learning representation for end-to-end recording device identification. *EURASIP Journal on Advances in Signal Processing*, *2021*(1). https://doi.org/10.1186/s13634-021-00763-1

Zhang, Z., Xu, S., Zhang, S., Qiao, T., & Cao, S. (2021). Attention based convolutional recurrent neural network for environmental sound classification. *Neurocomputing*, *453*, 896–903. https://doi.org/10.1016/j.neucom.2020.08.069

Zhao, C., Wang, J., Li, L., Qu, X., & Xiao, J. (2022). Adaptive Few-Shot Learning Algorithm for rare sound event detection. *2022 International Joint Conference on Neural Networks (IJCNN)*. https://doi.org/10.1109/ijcnn55064.2022.9892604

Zhao, X., Shao, Y., Mai, J., Yin, A., & Xu, S. (2020, December 16). *Respiratory Sound Classification Based on BiGRU-Attention Network with XGBoost*. IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9313506/authors#authors

Zhao, Y., Li, Y., & Wu, N. (2021). Data augmentation and its application in distributed acoustic sensing data denoising. *Geophysical Journal International*, *228*(1), 119–133. https://doi.org/10.1093/gji/ggab345