CSF425 Deep Learning Project: Audio Classification using Mel spectrograms

Yash Pandey 2021A7PS0661P
Vikram Komperla 2021A4PS1427P
Adhvik Ramesh 2021AA2465P

March 25, 2024

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
March, 2024

Contents

1 Introduction

We have approached the task of audio-classification by using Mel-Spectrograms for an array-representation of the audio files. The first step was to handle the diverse dataset provided to us using augmentation and pre-processing techniques. We made the following observations about the data-set:

1. Audio files provided in various formats (.wav, .aiff etc.)
2. A huge class-imbalance exists in number of data-points as the largest class contains 640 data points (Dog barking) while the smallest class consists of 168 data points (Knocking).
3. The audio files are not of uniform length but are all under 4 seconds in duration.
4. Some audios consist of background noise and vary in volume & intensity of the labelled sound.

Based on the above-mentioned observations, we have conducted preprocessing of the audios as well as the generated spectrograms as well as augmenting the diverse dataset to equalise the number of data-points in each class. Secondly, we have trained a number of different models on non-augmented and augmented data, drawing inspiration from existing architectures such as VGG and ResNets to arrive at the most suitable model for the task.

2 Preprocessing

Due to the presence of various formats of audio files and differing durations with background noise present, we preprocessed all the audio files (including augmented files) by applying techniques such as normalization and trimming & padding as described below.

Normalisation

- Load the audio file using `librosa.load()`, which results in a NumPy array.
- Since different file formats like (.wav, .aiff, .mp3, .flac, etc) have different min and max values, we normalise the array to ensure all audio files are on the same scale.
  - Normalize all values in the array to range between -1 to 1.

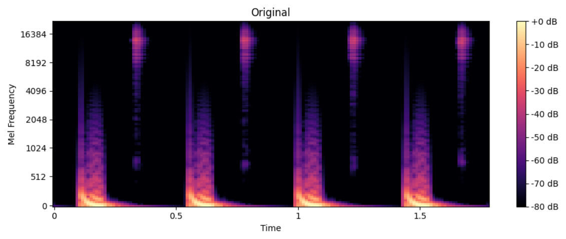
Trimming and Padding

- Detect silence in the audio clip:
  - Trim leading and trailing silent sections.
  - Flatten intermittent values to remove background noise below 60 decibels.
    - 60 dB is a conservative estimate for background noise when recording the audio clip.
- Check the length of the audio clip:
  - If it's less than 4 seconds, pad the clip with silence to make it exactly 4 seconds long. This not only makes all the audio clips uniform but also makes sure that the size of the numpy/pytorch arrays generated are of the same size.

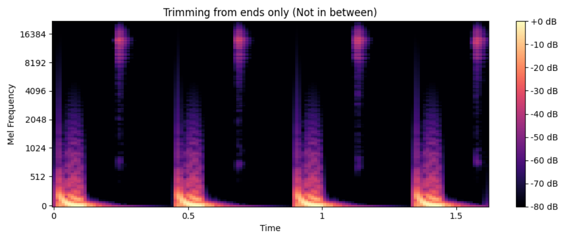
3 Generation of spectrograms

After augmentation and pre-processing of audio files, we generated the mel spectrograms using the librosa library. Each spectrogram was converted to a 3-channel array and saved as a Numpy array (.npz) so as to utilise them for training later.
Given below are 4 mel-spectrograms corresponding to different stages of pre-processing to help visualize how pre-processing makes changes to an audio file. Fig 1.(a) corresponds to the original audiofile. Fig 1.(b) corresponds to the

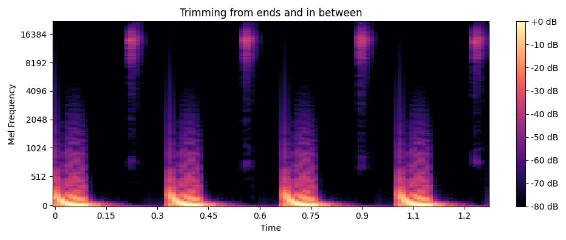
audio file after removing sound lower than 60db from its edges. Fig 1.(c) corresponds to the audio file after removing sound lower than 60db from in between. Fig 1.(d) corresponds to the final pre-processed audio file after padding has been added to make the audio clip 4 seconds long.



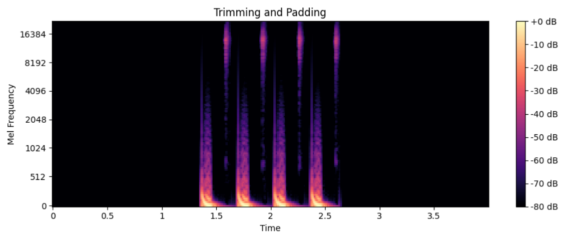
(a) Original Melspectrogram



(b) Trimmed from ends only (Not in between)



(c) Trimmed from ends and in between



(d) Final Pre-processed version

Figure 1: Stages of pre-processing

NOTE: Normalisation is done before the trimming process. The process of normalising doesn't cause changes to the mel-spectrogram. Also, Fig 1.(b) may seem stretched as compared to Fig 1.(a) but this just seems that way because of representaion effects while scaling the image size for display

4 Augmentation

We decided to augment the training data based on the following observations:

1. There was an imbalance in the number of images for different classes as some images had only 140 images while a couple classes like the class 'siren' had 640 images.
2. The audios in a class varied widely with respect to duration, background noise etc.

The following two augmentation processes were used to address the aforementioned problems

4.1 Audio Augmentation

For creating more audio files, we decided to equalise the number of files in each class to 800, a number above the maximum number of files present in any class(640) as even the classes with a large number of images as compared to the others require augmentation. The techniques used are as follows:

1. Pitch-Shift
2. Time-Shift
3. Noise-Injection
4. Speed-Change

These techniques were used as merited by the results in:

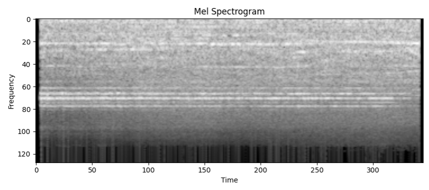
Process of augmentation: For each class, we calculate the number of files in the class folder. If this number is less than the maximum allowed samples (800), we determine the number of additional files needed to reach the maximum. We then evenly distribute these additional files among the existing files to determine the augmentation factor per file. Any remaining augmentations are distributed to the files randomly. We apply four types of augmentations: time shift, speed change, pitch shift, and noise injection. Each file is augmented based on the determined augmentation factor chosen randomly for each file.

4.2 spectrogram Augmentation

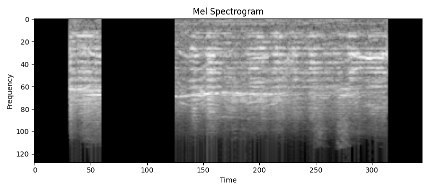
We have used the following two techniques for spectrogram augmentation, where after augmenting audio, we make the number of spectrograms/data-points for each class to 1000:

1. Frequency-Masking
2. Time-Masking

Masking for augmentation of spectrograms makes the model more robust to noise and allows it to focus on different frequency bands in the spectrogram or be invariant to irrelevant features such as background noise. This allows the model to generalise better.



(a) Non-Augmented Spectrogram



(b) Spectrogram after applying frequency masking

Figure 2: Augmentation of spectrograms

Process of augmentation: After audio augmentation, preprocessing and creation of Mel spectrograms, we randomly sampled 200 spectrograms for each class and applied frequency masking and time masking to create 200 new spectrograms. This results in a total of 1000 data-points for each class.

5 Training

We have experimented with a variety of architectures to arrive at the best possible model for the audio classification task that can handle the diverse dataset. Some techniques followed for the training of various models are as follows:

1. All models were trained for 10 epochs.
2. Adam optimiser with a learning rate of 0.001 was chosen based on experiments described below
3. An Exponential LR Scheduler was used for learning rate decay
4. The validation accuracy was checked at each epoch and the model with highest validation accuracy was saved and returned at the end of training.

The following models are a few amongst the ones we trained & collected results for:

1. VGG10
2. VGG16
3. VGGish

4. ResNet CNN
5. Simplified ResNet CNN
6. Basic CNN Model
7. Attention-based CNN

The final statistics for these models are as follows (the results are further described in later tables and text.

Table 1: Model Performance Comparison

Model Type	Best Training Accuracy (%)	Best Validation Accuracy (%)
Basic CNN	98.66	89.91
VGG10	98.57	89.16
VGGish	98.08	90.49
Simple ResNet (10 epochs)	96.35	94.38
Simple ResNet (25 epochs)	98.52	95.86
Full ResNet	98.55	94.46
Attention Based CNN	98.35	89.5
LSTM-Based CNN	98.61	90.41

The two chosen model architectures are as follows:

1. **Simplified ResNet CNN (25 epochs, Exponential Scheduler, lr = 0.001, gamma = 0.9)**
2. **VGGish (10 epochs, Exponential Scheduler, lr = 0.001, gamma = 0.9)**

Although the metric results of VGGish and the next best, LSTM CNN and Attention-Based CNN were close, we chose the VGGish as we were told to avoide RNNs and we gave priority to the metric results over the Attention-Based CNN's novelty.

5.1 Choice of Optimizer: Adam

Throughout the training processes, we have used the Adam optimizer due to its ability to adaptively adjust learning rates for each parameter, making it well-suited for a wide range of deep learning tasks. Adam combines the benefits of both momentum and RMSprop algorithms, allowing for faster convergence and improved generalization compared to other optimizers like SGD or Adagrad. Additionally, Adam requires less manual tuning of learning rates and momentum parameters, making it more convenient and efficient for training deep neural networks

5.2 Choice of model

We observed generally good performance from the above models on both the validation & testing set. The figures and tables in this report illustrate various results obtained from these models and we have thoroughly described why we chose our model architectures based on experimental results. The ResNet and Attention-based CNN architectures gave the highest validation accuracies.

5.2.1 VGG10

The VGG10 architecture consists of 3 convolutional layers followed by max-pooling layers for feature extraction, resulting in spatial reduction. These convolutional layers use 3x3 kernels with padding to preserve spatial dimensions. After feature extraction, the output is flattened and passed through two fully connected layers with ReLU activation functions for classification. The architecture follows a pattern of increasing feature map depth while decreasing spatial dimensions until the fully connected layers for classification. As shown in figure 3, the VGG10 model showed signs of over-fitting as the training accuracy was significantly higher than the validation accuracy at the point where it's validation accuracy was at its highest.

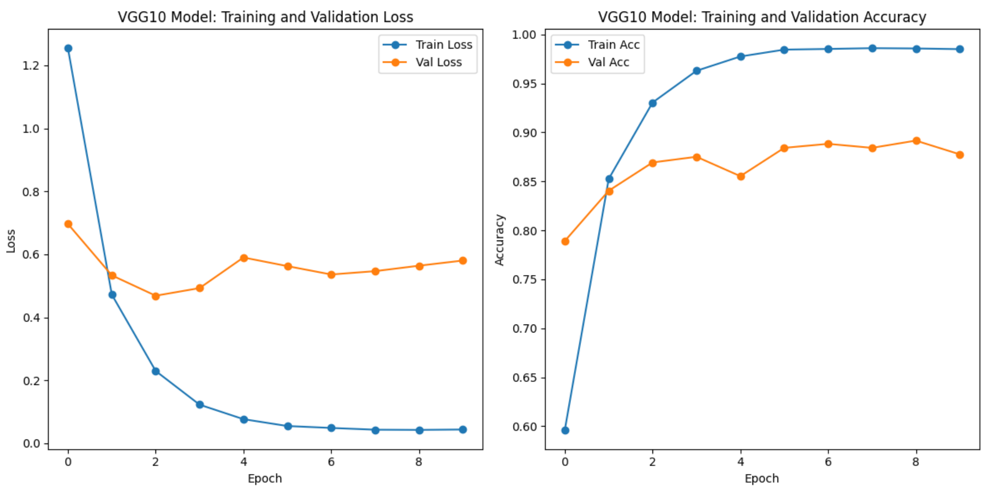


Figure 3: Training and Validation Accuracies for VGG10

Table 2: Training & Validation Accuracies per Epoch for VGG10

Epoch	Train Accuracy	Validation Accuracy
0	0.5962	0.7891
1	0.8525	0.8404
2	0.9305	0.8693
3	0.9629	0.8751
4	0.9775	0.8553
5	0.9845	0.8842
6	0.9852	0.8883
7	0.9860	0.8842
8	0.9857	0.8916
9	0.9850	0.8776

Table 3: Classification Metrics for Each Class for VGG10

Class	Precision	Recall	F1-score
0	0.9213	0.9647	0.9425
1	0.8797	0.8688	0.8742
2	0.9420	0.9286	0.9353
3	0.7794	0.7361	0.7571
4	0.9353	0.9559	0.9455
5	0.9266	0.9018	0.9140
6	0.8605	0.8810	0.8706
7	0.8537	0.8537	0.8537
8	0.8750	0.8630	0.8690
9	0.8000	0.8302	0.8148
10	0.9624	0.9143	0.9377
11	0.9630	0.9286	0.9455
12	0.5714	0.7442	0.6465

5.2.2 Basic CNN Model

The CNNModel architecture comprises three convolutional layers followed by max pooling operations for downsampling. Each convolutional layer is activated by ReLU. Subsequently, the feature maps are flattened and passed through two fully connected layers with ReLU activation, culminating in a final linear layer for classification. This model architecture is designed for image classification tasks, where it learns hierarchical features from input images and makes predictions based on learned representations. The CNN model, being relatively less heavy was trained for 20 epochs. We obtained a highest validation accuracy of 89.5% along with a highest training accuracy of 98.7%. However, it's accuracy on both the training set and validation set plateaued near the 7th epoch and showed no significant improvement in further epochs. Moreover, the significant 9.2% difference between it's best validation and testing accuracies indicated overfitting which is why we didn't consider it as a sole model architecture.

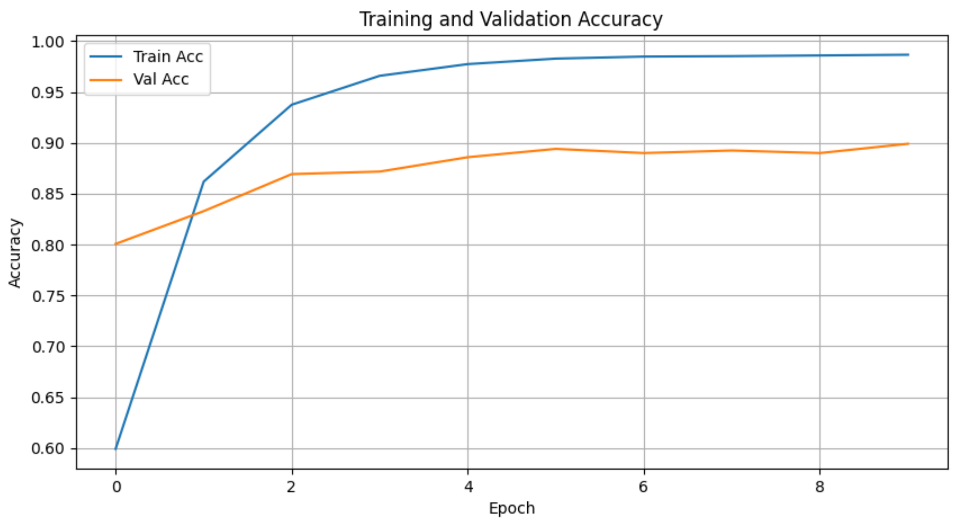


Figure 4: Training and Validation Accuracies for Basic CNN Model

Table 4: Training and validation accuracy for each epoch for Basic CNN

Epoch	Train Acc	Val Acc
0	0.5993	0.8007
1	0.8619	0.8329
2	0.9375	0.8693
3	0.9659	0.8718
4	0.9774	0.8859
5	0.9828	0.8941
6	0.9848	0.8900
7	0.9852	0.8925
8	0.9859	0.8900
9	0.9866	0.8991

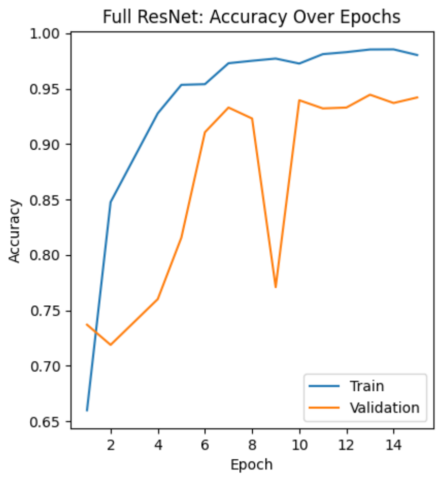
Table 5: Classification Metrics for Each Class for Basic CNN

Class	Precision	Recall	F1-score
Car horn	85.26	95.29	90.00
Dog barking	91.61	88.75	90.16
Drilling	95.62	93.57	94.58
Fart	78.57	76.39	77.46
Guitar	93.48	94.85	94.16
Gunshot and gunfire	87.39	92.86	90.04
Hi-hat	92.68	90.48	91.57
Knock	83.33	85.37	84.34
Laughter	92.75	87.67	90.14
Shatter	86.54	84.91	85.71
Siren	97.01	92.86	94.89
Snare drum	88.89	92.86	90.83
Splash and splatter	72.50	67.44	69.88

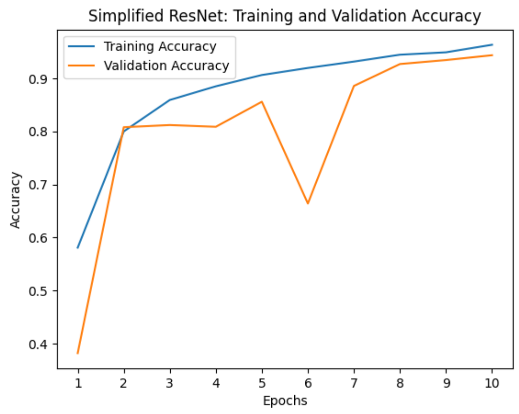
5.2.3 Simplified ResNet - (our first chosen model)

The Simplified ResNet architecture is a modified version of the original ResNet, designed for simpler implementation and reduced computational complexity. It consists of multiple ResNet blocks, each containing two convolutional layers with batch normalization and ReLU activation functions. These blocks introduce skip connections, allowing the gradients to flow more directly during training and mitigating the vanishing gradient problem. The downsampling of feature maps is achieved through strided convolutions or pooling layers. Finally, a global average pooling layer aggregates features, followed by a fully connected layer for classification. This architecture enables effective training of deep neural networks with improved performance and faster convergence. In comparison to other models, as illustrated in figure 4, the simple ResNet performed significantly better than the VGG-10 & Basic-CNN. It obtained a highest validation accuracy of 94.38% and a highest training accuracy of 96.35%. It did not seem to over-fit as it's validation and training accuracies were both high and very close to each other at the point where it's validation accuracy was highest. Moreover, based on popular research, temporal features are of significance for audio classification tasks, allowing CNNs to perform better. Hence, we decided to use the Simple ResNet as our first model architecture.

Why we chose Simplified ResNet over a complex version of ResNet CNN To validate if a more complex ResNet performed better than our simplified version, we trained a complex "Full ResNet" for 15 epochs using the same exponential scheduler with learning rate = 0.001. The results were worse than what the simplified ResNet attained over 25 epochs, with a highest validation accuracy of 94.46% and a highest training accuracy of 98.55%. Preferring the lighter architecture and more consistent results of the simplified ResNet, we chose that as our first architecture.



(a) Full ResNet with Exponential Scheduler



(b) Simplified ResNet with Exponential Scheduler, 10 epochs

Figure 5: Training and Validation Accuracies

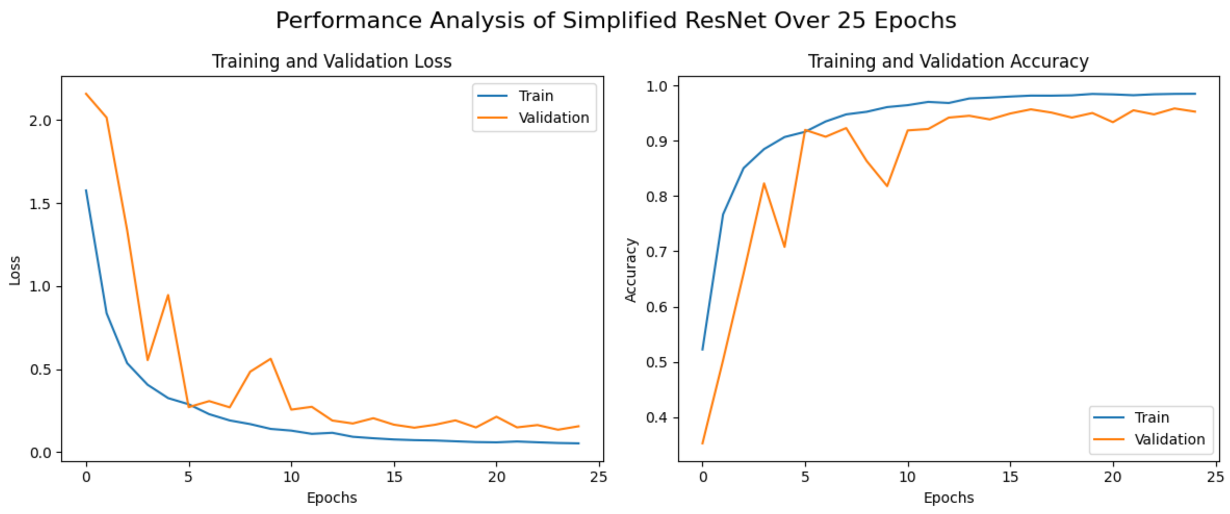


Figure 6: Training and Validation Accuracies for Simplified ResNet with Exponential Scheduler, 25 epochs

Table 6: Training and Validation Accuracy, Simplified Resnet, 25 Epochs

Epoch	Training Accuracy	Validation Accuracy
1	52.24	35.24
2	76.68	50.29
3	85.06	66.00
4	88.51	82.30
5	90.69	70.80
6	91.63	91.98
7	93.51	90.74
8	94.79	92.31
9	95.26	86.35
10	96.11	81.80
11	96.46	91.89
12	97.05	92.14
13	96.85	94.21
14	97.67	94.54
15	97.82	93.88
16	98.02	94.95
17	98.19	95.70
18	98.18	95.12
19	98.24	94.21
20	98.49	95.04
21	98.41	93.38
22	98.26	95.53
23	98.43	94.79
24	98.50	95.86
25	98.52	95.29

Table 7: Classification Metrics for Each Class for Simplified ResNet [25 epochs, Exp Scheduler]

Class	Precision	Recall	F1-score
Car horn	96.59	100.00	98.27
Dog barking	97.42	94.38	95.87
Drilling	97.86	97.86	97.86
Fart	96.97	88.89	92.75
Guitar	97.08	97.79	97.44
Gunshot and gunfire	97.32	97.32	97.32
Hi-hat	81.25	92.86	86.67
Knock	88.64	95.12	91.76
Laughter	95.77	93.15	94.44
Shatter	96.23	96.23	96.23
Siren	96.48	97.86	97.16
Snare drum	96.36	94.64	95.50
Splash and splatter	93.02	93.02	93.02

5.2.4 VGGish - (our second chosen model)

The VGGish model is a variant of the VGG architecture, which is known for its simplicity and effectiveness in image classification tasks. (This model is not the same as the Official VGGish model developed by google) This model consists of six convolutional layers with max-pooling layers interspersed between them, followed by two fully connected layers. Each convolutional layer is followed by a ReLU activation function to introduce non-linearity.

The architecture follows a Conv:pool–Conv:pool–Conv:Conv-pool–Conv:Conv:pool followed by 2 fully connected layers. The first 2 convolution give 64 out features, the next 2 give 128 out features while the last 2 give 256 out features. This provides a hierarchial structure of feature extracting by making the complex as it goes deeper so as to extract the finer features. At the same time downsampling is done to reduce dimensions. The last 2 fully connected layers output 512 and 13 classes respectively.

Results from experiments with the VGGish model showed promising performance in terms of both training and validation accuracy. However, despite its effectiveness the simplified ResNet model demonstrated comparable or even superior performance in terms of validation accuracy while being computationally less intensive. Therefore, we chose the simplified ResNet model as the first architecture over the VGGish model for its balance between performance and computational efficiency. However, as this model gave the 2nd best accuracy and exceptionally good classification metrics (table 9) we chose this as our second model

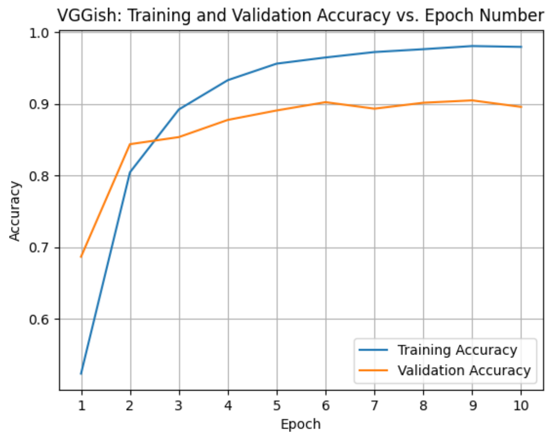


Figure 7: Training and Validation Accuracies for VGGish

Epoch	Training Accuracy (%)	Validation Accuracy (%)
1	52.32	68.65
2	80.44	84.37
3	89.22	85.36
4	93.31	87.76
5	95.62	89.08
6	96.48	90.24
7	97.24	89.33
8	97.64	90.16
9	98.08	90.49
10	97.96	89.58

Table 8: Training and Validation Accuracy for VGGish

Table 9: Classification Report for VGGish Model

Class	**Precision (%)**	**Recall (%)**	**F1-score**
Car horn	86.52	90.59	88.51
Dog barking	91.36	92.50	91.92
Drilling	95.68	95.00	95.34
Fart	84.13	73.61	78.52
Guitar	96.12	91.18	93.58
Gunshot and gunfire	91.67	88.39	89.66
Hi-hat	88.37	90.48	89.42
Knock	92.50	90.24	91.36
Laughter	77.17	97.26	86.06
Shatter	83.33	84.91	84.11
Siren	98.52	95.00	96.72
Snare drum	93.04	95.54	94.29
Splash and splatter	72.50	67.44	69.88

5.2.5 Attention-Based CNN

The AttentionCNN model is a variant of Convolutional Neural Networks (CNNs) that incorporates an attention mechanism. The model starts with three convolutional layers of increasing out features. Following each convolutional layer, proceeds a max pooling operation to down sample the feature space. After flattening the output from the last convolutional layer, it is passed through the attention layer which computes attention weights using softmax, focusing on key parts of the feature space. The output from this is then sent to two fully-connected linear layers that converge to classify into 13 categories as required. Essentially, the model increases in complexity while extracting features while at the same time reducing dimensions. In between it is designed to focus on relevant parts of the image.

As seen from the data, both the AttentionCNN and the VGGish are provide very similar performance, especially in terms of validation accuracy. Validation Accuracy of the AttentionCNN model steadily increased, reaching a peak of 89.50% at epoch 8. The VGGish model shows a decent increase in Validation Accuracy, with a peak of 90.49% at epoch 9. These results show that the VGGish model is better than the AttentionCNN, however we must note that the VGGish model is slightly more complex than the AttentionCNN.

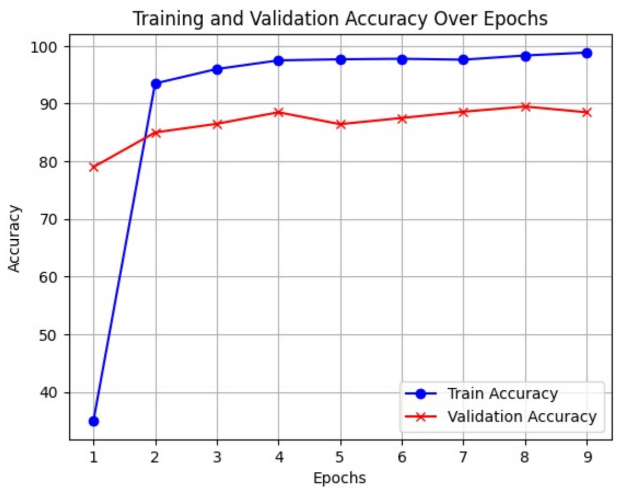


Figure 8: Training and Validation Accuracies for Attention-Based CNN

Table 10: Training & Validation Accuracy of AttentionCNN Model

Epoch	Train Accuracy (%)	Validation Accuracy (%)
1	35.00	79.00
2	93.50	85.00
3	96.00	86.50
4	97.50	88.50
5	97.69	86.44
6	97.78	87.51
7	97.62	88.60
8	98.35	89.50
9	98.85	88.50

Table 11: Classification Report for Attention CNN

Class	Precision (%)	Recall (%)	F1-score
Car horn	83.33	94.12	88.40
Dog barking	88.41	90.62	89.51
Drilling	94.81	91.43	93.09
Fart	80.26	84.72	82.43
Guitar	93.43	94.12	93.77
Gunshot and gunfire	84.87	90.18	87.45
Hi-hat	86.96	95.24	90.91
Knock	81.40	85.37	83.33
Laughter	87.14	83.56	85.31
Shatter	83.33	84.91	84.11
Siren	97.78	94.29	96.00
Snare drum	95.41	92.86	94.12
Splash and splatter	88.00	51.16	64.71

5.2.6 LSTM CNN

The LSTM-CNN model is designed for audio classification tasks. It consists of three convolutional layers followed by max-pooling layers for feature extraction. The convolutional layers utilize kernel sizes of 3x3 with ReLU activation functions to capture spatial features from the input audio spectrograms. Max-pooling layers with kernel sizes of 2x2 are employed to downsample the feature maps. The output of the convolutional layers is flattened and fed into an LSTM (Long Short-Term Memory) layer with a specified number of hidden units and layers. The LSTM layer learns temporal dependencies in the extracted features. Finally, a fully connected layer with softmax activation is used to predict the audio class labels.

The model was trained on a dataset with 10 epochs using the Adam optimizer with a learning rate of 0.001. During training, the model achieved a best training accuracy of 98.61% and a best validation accuracy of 90.41% over the 10 epochs. The training accuracy steadily increased with each epoch, indicating effective learning of the audio features by the model. The validation accuracy also showed improvement over the epochs, reaching a peak at epoch 9 before slightly dropping in epoch 10. Overall, the LSTM-CNN model demonstrates strong performance in audio classification tasks, achieving high accuracy rates on both training and validation datasets.

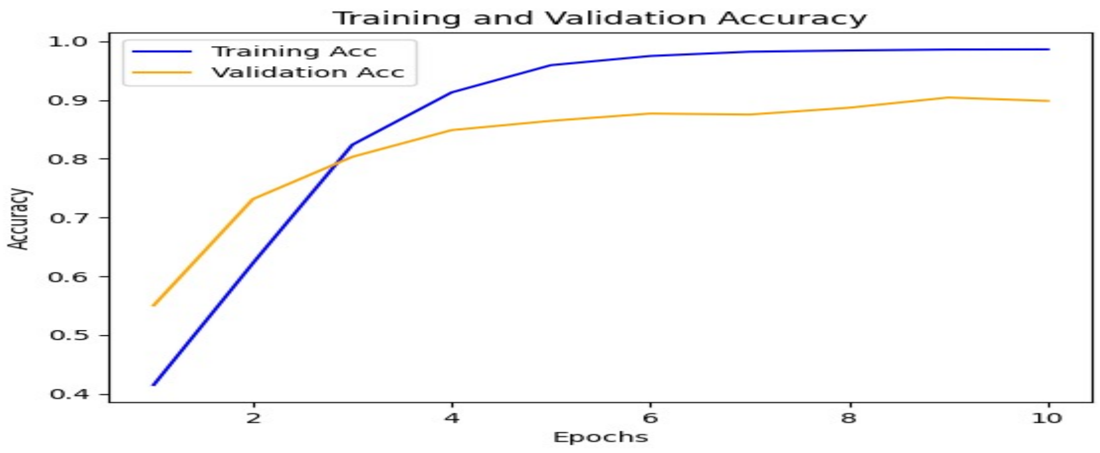


Figure 9: Training and Validation Accuracies for LSTM CNN

Table 12: Training and Validation Accuracy

Epoch	Training Accuracy (%)	Validation Accuracy (%)
1	41.46	55.00
2	62.21	73.12
3	82.36	80.31
4	91.28	84.86
5	95.92	86.44
6	97.48	87.68
7	98.21	87.51
8	98.42	88.67
9	98.58	90.41
10	98.61	89.83

6 Choice of Scheduler

For choosing whether or not and which scheduler to use, we conducted the following three experiments with our simplified ResNet model:

1. Simplified ResNet using exponential scheduler with gamma = 0.9
2. Simplified ResNet using step scheduler with step size = 0.7 & gamma = 0.1
3. Simplified ResNet with no scheduler

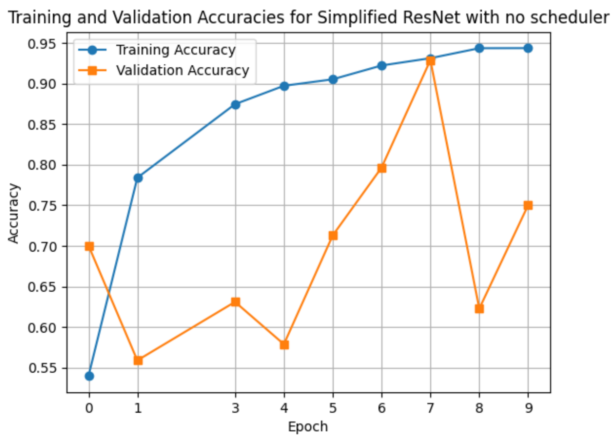


Figure 10: Training and Validation Accuracies for Simplified ResNet CNN with no scheduler

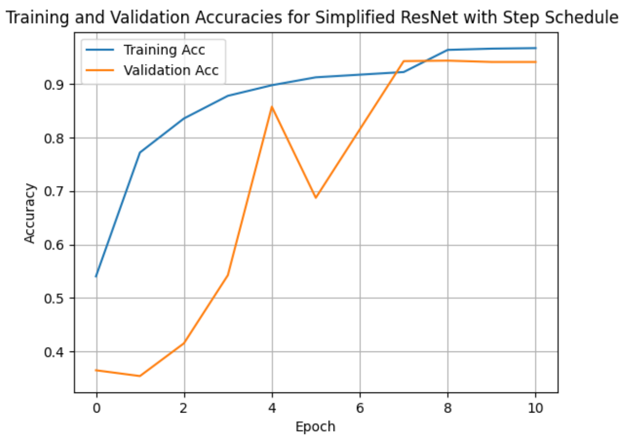


Figure 11: Training and Validation Accuracies for Simplified ResNet CNN with step scheduler

The first experiment's results are described in table 15 with a highest validation accuracy of 94.38%. As inferred clearly from table 14, the model performs very poorly with no scheduler. Moreover, as indicated by table 13, the model's training accuracy generally increases per epoch and the validation accuracy increases till epoch 6 but then essentially plateaus. The model attained 94.3% validation accuracy with the step scheduler which is the same as what it obtained with the exponential scheduler. However, due to a more regular performance with the exponential scheduler, we choose to use that for our models.

Table 13: Training and Validation Results for Simplified ResNet with Step Scheduler

Epoch	Train Acc	Validation Acc
0	0.5404	0.3648
1	0.7720	0.3540
2	0.8356	0.4152
3	0.8780	0.5426
4	0.8978	0.8577
5	0.9126	0.6873
7	0.9225	0.9429
8	0.9639	0.9438
9	0.9662	0.9413
10	0.9672	0.9413

Table 14: Training and Validation Results for simplified ResNet with no scheduler

Epoch	Train Loss	Train Accuracy	Validation Accuracy
0	1.5182	0.5398	0.7006
1	0.7635	0.7840	0.5591
3	0.4193	0.8746	0.6311
4	0.3552	0.8972	0.5790
5	0.3153	0.9052	0.7130
6	0.2654	0.9222	0.7965
7	0.2316	0.9312	0.9289
8	0.1906	0.9435	0.6228
9	0.1863	0.9436	0.7502

Table 15: Training and Validation Accuracies per Epoch for Simplified ResNet (10 epochs, Exp Scheduler)

Epoch	Train Accuracy	Validation Accuracy
0	0.5810	0.3821
1	0.8001	0.8081
2	0.8594	0.8122
3	0.8852	0.8089
4	0.9065	0.8561
5	0.9198	0.6642
6	0.9318	0.8859
7	0.9448	0.9272
8	0.9493	0.9347
9	0.9635	0.9438

7 Hyperparameter Tuning for Learning Rate & Gamma

We utilized Optuna, a hyperparameter optimization framework, to determine the optimal learning rate (lr) and gamma for our models. The experiment was conducted under the following conditions:

1. **Objective Function Definition:** We defined an objective function responsible for evaluating the model's validation accuracy. This function utilized a set of hyperparameters sampled from specified search spaces using Optuna's `trial.suggest_loguniform` and `trial.suggest_uniform` methods.

2. **Hyperparameter Search Spaces:** The search spaces for lr and gamma were defined to cover a range of values. `trial.suggest_loguniform` and `trial.suggest_uniform` methods were used to sample hyperparameters from these spaces.

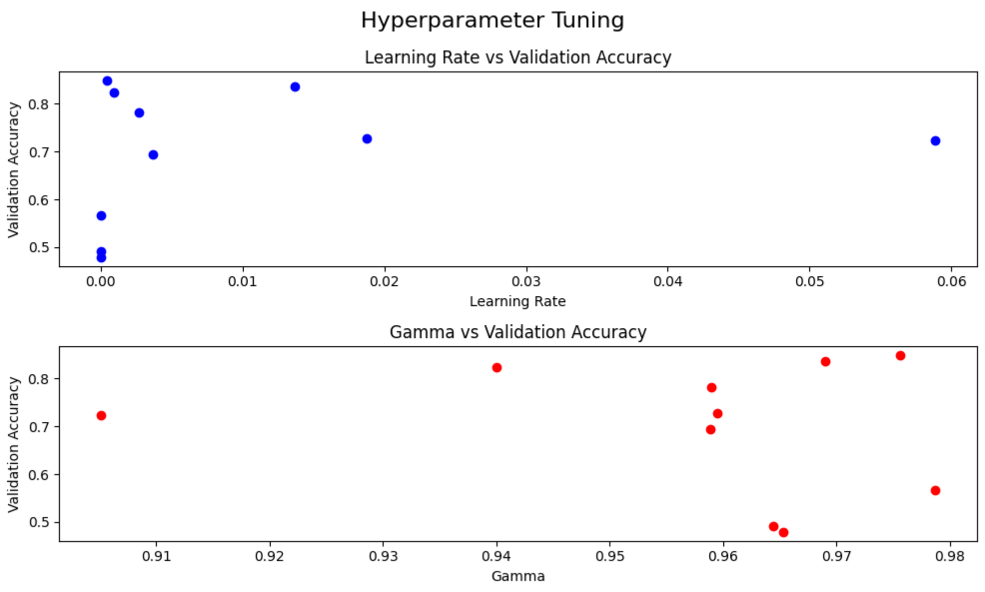


Figure 12: Hyperparamter Tuning Results

3. **Model Architecture:** Within the objective function, we instantiated the Simplified ResNet model with the sampled hyperparameters. This model was then trained for a fixed number of epochs (5).

4. **Training Configuration:** During training, we utilized the Adam optimizer and a cross-entropy loss function. The model was trained on the training dataset and evaluated on the validation dataset.

5. **Hyperparameter Optimization Process:** Optuna's `create_study` function initiated the hyperparameter optimization process. The objective was to maximize validation accuracy by exploring different combinations of lr and gamma.

Table 16: Hyperparameter Tuning Results

Trial	**Learning Rate (LR)**	**Gamma**	**Validation Accuracy**
0	0.00272	0.959	0.782
1	0.01369	0.969	0.835
2	1.91×10^{-5}	0.979	0.566
3	0.00366	0.959	0.694
4	0.00045	0.976	0.849
5	1.42×10^{-5}	0.964	0.490
6	0.05887	0.905	0.723
7	0.00096	0.940	0.824
8	0.01873	0.960	0.727
9	1.17×10^{-5}	0.965	0.479
Best	0.00045	0.976	0.849

6. **Best Hyperparameters Determination:** After a specified number of trials, Optuna identified the best hyperparameters along with the corresponding best validation accuracy. This automated the optimization of model performance without the need for manual parameter tuning.

By employing Optuna in this manner, we aimed to efficiently explore the hyperparameter space and identify the configuration that maximized the model's performance on the validation dataset. However, upon running the simplified ResNet for 25 epochs with these parameters, we got highest validation accuracy of which is less than what it attained for learning rate = 0.001 and gamma = 0.9, which indicates that the hyperparameter tuning needs to run each trial for a larger number of epochs than 5, however, due to a shortage of time, we did not run a further experiment with larger number of epochs.

8 Comparison of results with Original Dataset vs Augmented Dataset

The original dataset provided contains 4,861 files for training with each class containing varying number of files. After augmentation we increased that to 13,000 files with 1,000 files for each class. This made the validation accuracy increase. We concluded this by running the Simplified Resnet with the original and then the augmented dataset and compared the results.
Without augmentation - Highest validation accuracy = 91.32%

With augmentation - Highest validation accuracy = 95.86%
Below are the corresponding observations after training the Simplified Resnet with the original Dataset

Table 17: Training and Validation Results without augmentation for Simplified ResNet

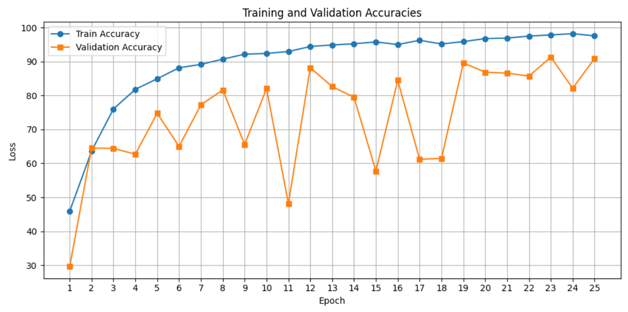


Figure 15: Training and Validation Accuracies for Simplified ResNet CNN without augmentation

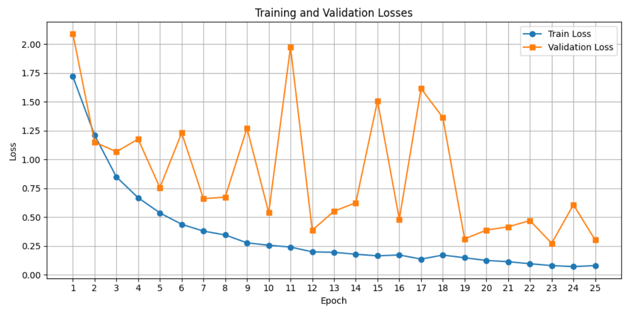


Figure 16: Training and Validation Loss for Simplified ResNet CNN without augmentation

Epoch	**Train Loss**	**Train Accuracy (%)**	**Validation Loss**	**Validation Accuracy (%)**
1	1.7201	45.91	2.0915	29.61
2	1.2107	63.64	1.1501	64.52
3	0.8484	76.03	1.0672	64.43
4	0.6680	81.81	1.1763	62.70
5	0.5361	84.92	0.7555	74.77
6	0.4373	88.17	1.2296	65.01
7	0.3784	89.16	0.6595	77.25
8	0.3449	90.70	0.6732	81.64
9	0.2767	92.16	1.2715	65.59
10	0.2556	92.39	0.5389	82.05
11	0.2400	92.94	1.9729	48.14
12	0.1985	94.42	0.3862	88.17
13	0.1943	94.88	0.5508	82.63
14	0.1776	95.23	0.6242	79.49
15	0.1640	95.76	1.5085	57.73
16	0.1712	95.00	0.4811	84.45
17	0.1355	96.26	1.6156	61.21
18	0.1704	95.16	1.3631	61.46
19	0.1471	95.86	0.3106	89.58
20	0.1238	96.75	0.3872	86.85
21	0.1131	96.91	0.4153	86.60
22	0.0952	97.49	0.4701	85.69
23	0.0794	97.84	0.2727	91.32
24	0.0705	98.20	0.6070	82.13
25	0.0792	97.55	0.3008	90.90