MSc in Artificial Intelligence and Machine Learning

CS6482 – Deep Reinforcement Learning

Assignment 1: Sem2 AY 24/25 - Convolutional Neural Networks (CNNs)

Thendral Balakrishnan– 24128902

Table of contents

1. Introduction..... 3
2. Dataset..... 3
    2.1 Visualisation..... 3
3. Data Pre-processing..... 4
    3.1 Loading and Extracting Features from Audio Data..... 4
    3.2 Label Encoding..... 4
4. CNN Architectures..... 5
    4.1 One-Dimensional CNN..... 5
    4.2 1D AlexNet Architecture..... 6
    4.3 1D GoogLeNet Architecture:..... 7
5. Data Augmentation..... 9
6. Loss Function..... 10
7. Optimizer..... 10
8. Experiments..... 11
9. Results..... 12
10. Model Performance Comparison..... 16
11. References..... 19

1. Introduction

Convolutional Neural Networks (CNNs), widely used for image classification, are also highly effective in audio classification. While images are represented as two-dimensional signals, audio signals exist as one-dimensional waveforms, allowing CNNs to extract meaningful patterns from both domains using similar principles. This project investigates the use of CNNs for classifying bird genera based on their birdsong audio. The British Bird Songs dataset from Kaggle was selected, and the Librosa library is utilized for audio feature extraction, converting sound waves into 1D numerical arrays for processing. Although the dataset is relatively small, model performance could be enhanced by experimenting with different CNN architectures and implementing data augmentation to improve generalization and classification accuracy.

Code inspiration for this project was primarily taken from a machine learning blog titled *An Introduction to Audio Classification with Keras* (Ibrahim, 2023). The original implementation was applied to the UrbanSounds dataset from Kaggle, whereas this project adapts and extends the approach to the Bird Songs dataset. Additional modifications include testing different CNN architectures and applying data augmentation techniques to enhance model performance.

2. Dataset

The British Birdsongs Dataset is a curated collection of bird songs, featuring 264 recordings from 66 bird genera commonly found in the United Kingdom. Sourced from the Xeno Canto collection, it serves as a valuable resource for studying bird calls and songs. Each recording is stored in FLAC format, ensuring high-quality sound preservation.

2.1 Visualisation

The histogram plot of the dataset shows that the number of recordings per genus is quite low. Most genera have only 3 recordings, while a few have slightly more, with the maximum number of recordings per genus being 12. This indicates that each class has a very limited number of samples, which may impact the ability to capture variations in bird vocalizations effectively.

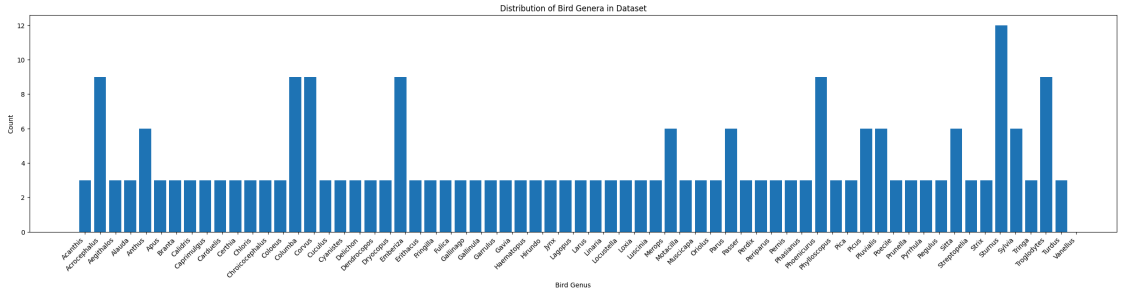


Fig 2.1: Distribution of Bird Genera in Dataset (Bird Genus vs Count)

3. Data Pre-processing

3.1 Loading and Extracting Features from Audio Data

The Librosa library is a widely used Python package for analyzing and processing audio signals. It provides tools for loading, resampling, feature extraction, and transformation of audio data, making it highly suitable for machine learning applications in sound classification. In this project, Librosa is used to load bird song recordings from the dataset while ensuring a consistent sample rate of 22,050 Hz across all audio files. This standardization is essential to ensure consistency in the input data, allowing deep learning models to learn and generalize effectively.

```
# Load the audio file and resample it
target_sr = 22050
audio, sample_rate = librosa.load(file_path, sr=target_sr)

# Extract MFCC features
mfccs = librosa.feature.mfcc(y=audio, sr=target_sr, n_mfcc=40)
mfccs_scaled = np.mean(mfccs.T, axis=0)
```

Fig 3.1: Feature Extraction from Audio Data

Once the audio is loaded, the Mel-Frequency Cepstral Coefficient (MFCCs) are extracted using Librosa's librosa.feature.mfcc() function. MFCCs are widely used in speech and bioacoustic analysis because they effectively capture the spectral properties of audio signals, mimicking how the human ear perceives sound. Since MFCCs generate a time-independent matrix, their mean values are computed to produce fixed-length feature vectors, ensuring consistency across recordings of varying durations. These extracted feature vectors, along with their corresponding genus labels, are then stored as NumPy arrays.

3.2 Label Encoding

Deep learning models require numerical input, so the categorical bird genus labels need to be converted into a machine-readable format. This is done using label encoding, where each unique genus is assigned a corresponding integer value using scikit-learn's LabelEncoder(). However, treating categorical labels as ordinal numbers could mislead the model, as it may assume an inherent order in classes. To prevent this, the encoded labels are further converted into a one-hot encoded representation. This transformation ensures that each genus is represented by a binary vector, where only one position is active (set to 1), eliminating any unintended hierarchical relationships between classes.

Once the MFCC feature vectors and one-hot encoded labels are prepared, they serve as the final input dataset for training the Convolutional Neural Network (CNN). These preprocessed features

provide a structured and standardized format, allowing the model to effectively learn the distinct patterns in bird vocalizations for genus classification.

4. CNN Architectures

The first CNN architecture follows the framework outlined by Ibrahim (2023). Building upon this foundation, 1D adaptations of AlexNet and GoogLeNet are optimized specifically for sequential audio data processing.

4.1 One-Dimensional CNN

The 1D CNN model uses a series of convolutional, pooling, dropout, and dense layers. It consists of two 1D convolutional layers with 64 and 128 feature maps, respectively, each followed by max pooling to reduce dimensionality and dropout layers to prevent overfitting. The extracted features are then flattened and passed through a fully connected layer with 512 neurons, ensuring effective feature abstraction. A final dense layer with 66 output units is used for classification. The choice of parameters balances model complexity and computational efficiency, with a total of 714,690 trainable parameters.

Layer	**Type**	**# Maps**	**Output Shape**
1	Input Layer	-	(40,1)
2	Convolution	64	(40,64)
3	Max Pooling	64	(20,64)
4	Dropout	64	(20,64)
5	Convolution	128	(20,128)
6	Max Pooling	128	(10,128)
7	Dropout	128	(10,128)
8	Flatten	-	1280
9	Dense	-	512
10	Dropout	-	512
11	Output	-	66

Table 4.1.1: 1D CNN Architecture

4.2 1D AlexNet Architecture

AlexNet, originally introduced by Krizhevsky et al. (2012) for image classification, has been adapted in this project for audio classification by modifying its convolutional layers to process sequential data. A key modification in this adaptation is the use of stride = 1 throughout the network to prevent the loss of important information, as a higher stride value would excessively reduce the sequence length. Additionally, “same” padding is applied to all convolutional layers.

The architecture consists of five convolutional layers, each followed by batch normalization for stabilizing training. MaxPooling1D layers are strategically placed to downsample the feature maps while retaining crucial information. Finally, fully connected layers with dropout regularization are used before the softmax output layer to classify bird species based on their vocal characteristics.

The 1D AlexNet model has a total of 6,416,840 parameters, out of which 2,138,178 are trainable, while 2,304 are non-trainable. The optimizer parameters account for 4,276,358, reflecting the computational overhead required for optimizing the deep network.

Layer	**Type**	**# Maps**	**Output Shape**
1	Input Layer	-	(40,1)
2	Convolution	128	(36,128)
3	Batch Normalization	-	(36,128)
4	Max Pooling	-	(35,128)
5	Convolution	256	(35,256)
6	Batch Normalization	-	(35,256)
7	Max Pooling	-	(17,256)
8	Convolution	256	(17,256)
9	Batch Normalization	-	(17,256)
10	Convolution	256	(17,256)
11	Batch Normalization	256	(17,256)
12	Convolution	256	(17,256)
13	Batch Normalization	256	(17,256)

14	Max Pooling	256	(16,256)
15	Flatten	-	4096
16	Dense	-	1024
17	Dropout	-	1024
18	Dense	-	1024
19	Dropout	-	1024
20	Output	-	66

Table 4.2.1: 1D AlexNet Architecture

4.3 1D GoogLeNet Architecture:

GoogleNet, also known as the Inception architecture, was introduced by Szegedy et al. (2015) as a deep convolutional network designed to optimize computational efficiency while maintaining high classification accuracy. The architecture is built upon Inception modules, which allow multiple convolutional operations with different kernel sizes to be performed in parallel. This enables the network to learn features at multiple scales simultaneously.

Inspiration for the code taken from Lekhuyen (2020), which was used for image classification. The network starts with an initial Conv1D layer (64 filters, 5×1 kernel, stride 1), followed by batch normalization and max pooling to stabilize training and reduce feature map dimensions. A second convolutional stage refines feature extraction using 1×1 and 3×1 convolutions, helping to maintain computational efficiency. The core of the model consists of multiple Inception blocks, where each block processes the input using a combination of 1×1, 3×1, and 5×1 convolutions, along with a parallel max-pooling path, capturing short- and long-range dependencies in the birdsong patterns.

To enhance feature representation, pooling layers are applied after certain Inception blocks to progressively reduce the temporal dimensions while preserving key information. After passing through a series of Inception modules, the feature maps are flattened and processed through a dropout layer (0.4 dropout rate) to mitigate overfitting. The final classification is performed using a softmax-activated dense layer, mapping the extracted features to the respective bird species.

Layer	Type	# Maps	Output Shape
1	Input Layer	1	(40,1)
2	Convolution	64	(40,64)
3	Batch Normalization	-	(40,64)
4	Max Pooling	-	(40,64)
5	Convolution	64	(40,64)
6	Convolution	192	(40,192)
7	Batch Normalization	-	(40,192)
8	Max Pooling	-	(40,192)
9	Inception Block	64, 96, 128, 16, 32, 32	(40,256)
10	Inception Block	128, 128, 192, 32, 96, 64	(40,480)
11	Max Pooling	-	(40,480)
12	Inception Block	192, 96, 208, 16, 48, 64	(40,512)
13	Inception Block	160, 112, 224, 24, 64, 64	(40,512)
14	Inception Block	128, 128, 256, 24, 64, 64	(40,512)
15	Inception Block	112, 144, 288, 32, 64, 64	(40,528)
16	Inception Block	256, 160, 320, 32, 128, 128	(40,832)
17	Max Pooling	-	(40,832)
18	Inception Block	256, 160, 320, 32, 128, 128	(40,832)
19	Inception Block	384, 192, 384, 48, 128, 128	(40,1024)
20	Flatten	-	40960
21	Dropout	-	40960
22	Output	-	66

Table 4.3.1: 1D GoogLeNet Architecture

Why 1D GoogLeNet Has More Parameters Than 1D AlexNet

The GoogLeNet model contains a total of 10,467,928 parameters, out of which 3,489,138 are trainable, and 512 are non-trainable. In traditional 2D architectures, GoogLeNet is significantly more parameter-efficient than AlexNet due to its extensive use of 1×1 convolutions, which reduce the number of feature maps before applying larger kernel convolutions, and the elimination of large fully connected layers, which account for most of AlexNet's parameters. However, in this 1D adaptation, the opposite is observed—GoogLeNet has 10,467,928 parameters, while AlexNet has only 6,416,840 parameters. This occurs because 1×1 convolutions in 1D do not reduce feature map depth as effectively as in 2D, leading to a higher overall parameter count in the Inception blocks. Additionally, GoogLeNet's Inception architecture involves multiple parallel convolutional paths (1×1, 3×1, and 5×1 convolutions, along with pooling operations), which, when concatenated, increase the number of output channels at each stage, further contributing to a higher parameter count. In contrast, AlexNet follows a sequential architecture with fewer parallel branches, making it more parameter-efficient in 1D than GoogLeNet. The result is a reversal of the usual 2D trend, where GoogLeNet is typically lighter than AlexNet, demonstrating how architectural choices affect parameter count differently when transitioning from 2D to 1D applications.

5. Data Augmentation

Since the initial dataset was too small with only 264 samples, it resulted in poor performance for all the CNN models due to the lack of sufficient training data. To overcome this limitation and improve model generalization, data augmentation was applied to expand the dataset to 792 samples, effectively tripling its size.

The augmentation process involved the following transformations:

- Additive Noise: Gaussian noise with mean of 0 and a standard deviation of 0.01 was added to simulate real-world environmental variations in birdsong recordings.
- Time Stretching: The audio was randomly stretched and compressed by a factor between 0.9 and 1.1 using Librosa's time-stretching function. The modified signal is then converted back to MFCC features to maintain consistency with the original dataset.

```
# Add noise
noise = np.random.normal(0, 0.01, feature.shape[0])
augmented_features.append(feature + noise)
augmented_labels.append(label)

# Time stretching
stretch_factor = random.uniform(0.9, 1.1)
n_fft_value = 2048
```

Fig 5.1: Code snippet from notebook showing data augmentation

6. Loss Function

Categorical cross-entropy is defined as the negative log likelihood of the true class, penalizing incorrect classifications more heavily when the predicted probability for the class is low. It is chosen as the loss function because the project involves multi-class classification, where each audio sample belongs to a single bird genus out of 66 possible categories. Since the models' final layer uses a softmax activation function, which outputs a probability distribution across all classes, categorical cross-entropy effectively measures how well the predicted probabilities align with the true class labels.

This loss function ensures that the model learns to assign higher confidence to the correct class while minimizing errors across all categories. Given the small dataset and the potential for class imbalances, categorical cross-entropy is an optimal choice as it works well with probability-based models and ensures effective learning of discriminative features from the spectrogram-based MFCC inputs.

7. Optimizer

The Adaptive Moment Estimation (Adam) optimizer was chosen due to its adaptive learning rate capabilities, making it suitable for training deep learning models on datasets with complex features. Adam combines the benefits of both SGD with momentum and RMSprop, adjusting learning rates dynamically for each parameter based on first-moment (mean) and second-moment (variance) estimates of gradients. This prevents the model from getting stuck in local minima while ensuring faster convergence compared to traditional optimizers like vanilla SGD.

While Adam was used predominantly, the RMSprop optimizer was used with GoogLeNet due to its effectiveness in training deep architectures. As a deep network, GoogLeNet benefits from RMSprop's ability to stabilize gradient updates and reduce oscillations, ensuring smoother

learning. While Adam is an adaptive optimizer, it incorporates momentum, which may not always be ideal for very deep networks, sometimes leading to slower convergence compared to RMSprop. In contrast, RMSprop dynamically adjusts learning rates per parameter more aggressively, helping the model converge faster. Although Adam combines elements of RMSprop and momentum-based updates, it can sometimes be too conservative in modifying learning rates, making it slightly less efficient in optimizing deep networks like GoogLeNet.

8. Experiments

The experimental setup was designed to evaluate the performance of three different CNN architectures—1D CNN, AlexNet, and GoogLeNet—under varying training conditions. Initially, these models were trained without data augmentation to establish a baseline for comparison. Next, data augmentation was applied, expanding the dataset from 264 to 792 samples, after which the same models were retrained under identical settings to assess the impact of augmentation on model performance. Additionally, AlexNet and GoogLeNet were trained for 50 epochs with data augmentation, using a learning rate of 0.0001 with both the Adam optimizer and RMSprop, to analyze the effect of different optimization techniques on model convergence and accuracy. This experimental setup provided insights into the role of data augmentation, training duration, and optimization strategies in improving classification performance.

Condition	Models Trained	Epochs	Data Augmentation	Learning Rate	Optimizer(s)
Baseline Training	1D CNN, AlexNet, GoogLeNet	30	No	0.001	Adam
With Data Augmentation	1D CNN, AlexNet, GoogLeNet	30	Yes	0.001	Adam
Extended Training	AlexNet, GoogLeNet	50	Yes	0.0001	Adam, RMSprop

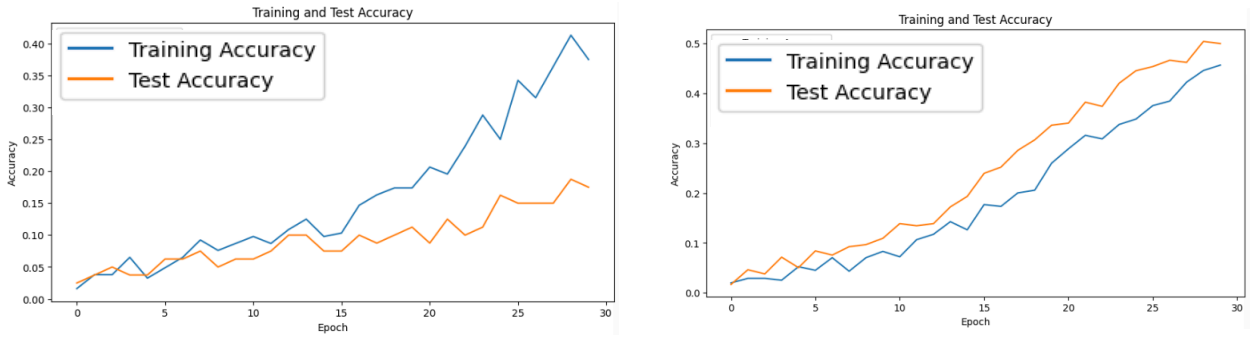
Fig 8.1.: List of Experiments

9. Results

9.1 1D CNN

The 1D CNN model exhibited significant overfitting when trained without data augmentation. While it achieved a training accuracy of 36.8%, its test accuracy was only 17.5%, indicating that the model failed to generalize well to unseen data. This suggests that the dataset size was insufficient, causing the model to memorize training patterns rather than learning meaningful representations.

With data augmentation, the 1D CNN model's performance improved significantly, achieving a training accuracy of 44.9% and a test accuracy of 50%. This indicates that the augmented dataset helped the model generalize better to unseen data. Interestingly, the test accuracy plot remained higher than the training accuracy plot for most of the training process, suggesting that data augmentation introduced variability that made the model more robust. This could also indicate that regularization effects from augmentation prevented overfitting, allowing the model to learn more meaningful patterns rather than memorizing the training data.

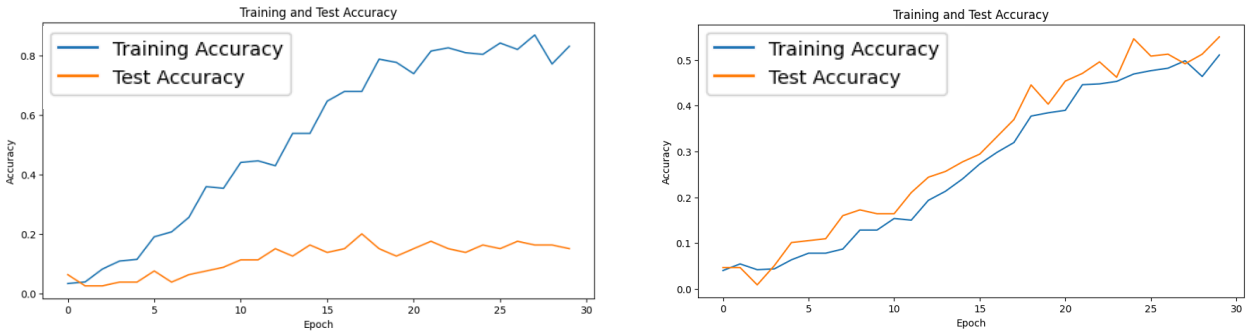


(a) (b)

Fig 9.1.: (a) Training and Test Accuracy Trend for 1D CNN Over 30 Epochs
(b)Training and Test Accuracy Trend for 1D CNN Over 30 Epochs after Data Augmentation

9.2 1D AlexNet

Without data augmentation, AlexNet exhibited significant overfitting, achieving a high training accuracy of 84.9% while the test accuracy remained very low at 15%. This suggests that the model was learning patterns specific to the training data but failed to generalize to unseen data. After applying data augmentation, the training accuracy dropped to 55.2%, but the test accuracy improved significantly to 55%, indicating better generalization.



(a) (b)

Fig 9.2.1: (a) Training and Test Accuracy Trend for AlexNet Over 30 Epochs
(b)Training and Test Accuracy Trend for AlexNet Over 30 Epochs after Data Augmentation

To further analyze the performance of the AlexNet model, extended experiments were conducted over multiple configurations. The primary objective was to observe how variations in training duration, learning rate, and optimizer choice impacted model accuracy.

Initially, the model was trained for 50 epochs without modifying any hyperparameters. The training accuracy reached 61.93%, while the test accuracy was 55.4%. This performance was slightly better than the results obtained when the model was trained for 30 epochs, indicating that the additional training helped the model learn better representations. However, the increase in accuracy was not substantial, suggesting that merely increasing the number of epochs was not sufficient for significant performance improvement.

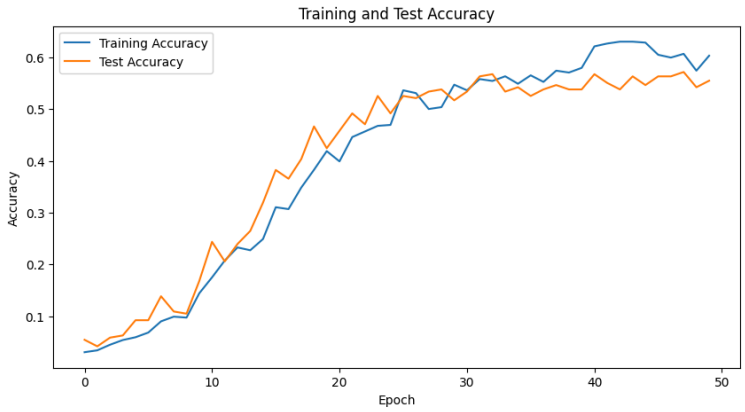
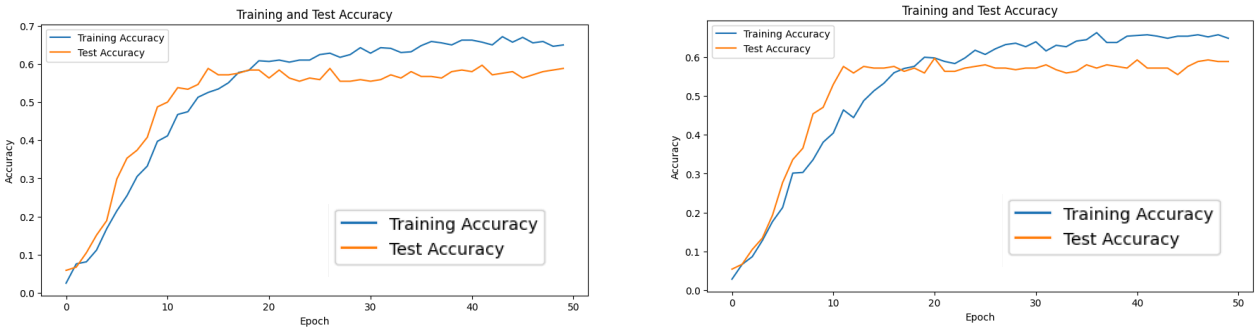


Fig 9.2.2 : (a) Training and Test Accuracy Trend for AlexNet Over 50 Epochs

Following this, the learning rate was reduced to 0.0001, keeping all other parameters constant. With this modification, the training accuracy improved to 66%, and the test accuracy increased to 58.8%. This result indicates that reducing the learning rate allowed the model to converge more effectively, leading to better generalization and improved test performance. A smaller learning rate typically helps in refining the weight updates, thereby stabilizing training and reducing the risk of overshooting optimal parameters.

In the next experiment, the optimizer was changed from Adam to RMSprop, while keeping the learning rate at 0.0001. This modification resulted in a training accuracy of 64.2% and a test accuracy of 58.8%, showing that the test accuracy remained unchanged compared to the previous experiment. The slight drop in training accuracy suggests that RMSprop might have provided a different update pattern, which did not significantly enhance generalization beyond what was already achieved with Adam.



(a) (b)

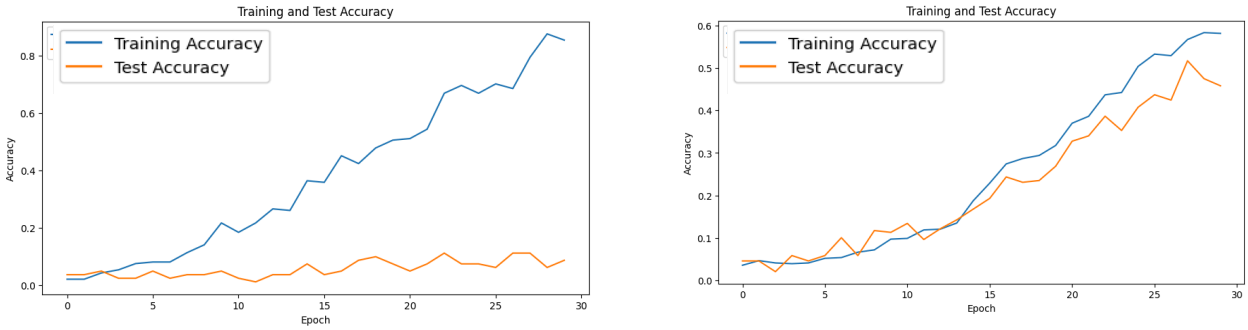
Fig 9.2.3: Training and Test Accuracy Trend for AlexNet Over 50 Epochs, learning rate = 0.0001
(a)With Adam Optimizer (b) With RMSProp Optimizer

From these experiments, several key observations were made. Firstly, extending the training duration improved accuracy, but the gains were not substantial. Secondly, reducing the learning rate had a more pronounced effect, leading to better training and test performance by ensuring smoother and more stable convergence. Lastly, switching to RMSprop did not yield significant improvements over Adam, suggesting that Adam was already well-suited for optimizing this model.

9.3 1D GoogLeNet

Initially, the model was trained for 30 epochs without data augmentation, achieving a high training accuracy of 87.14% but a very low test accuracy of 8.7%, indicating severe overfitting. To address this, data augmentation was applied, which significantly reduced training accuracy to 57.2% but improved test accuracy to 45.8%, demonstrating better generalization.

The experiments on the GoogLeNet model demonstrate the critical role of data augmentation in reducing overfitting, as evidenced by the significant improvement in test accuracy from 8.7% to 45.8% when augmentation was introduced. Extending training to 50 epochs further enhanced performance, with Adam at a 0.001 learning rate achieving 56.7% test accuracy, compared to 53.4% with RMSprop, highlighting Adam's superior generalization ability. The results suggest that while longer training and optimizer selection impact performance, overfitting remains an issue, as indicated by the persistent gap between training and test accuracy

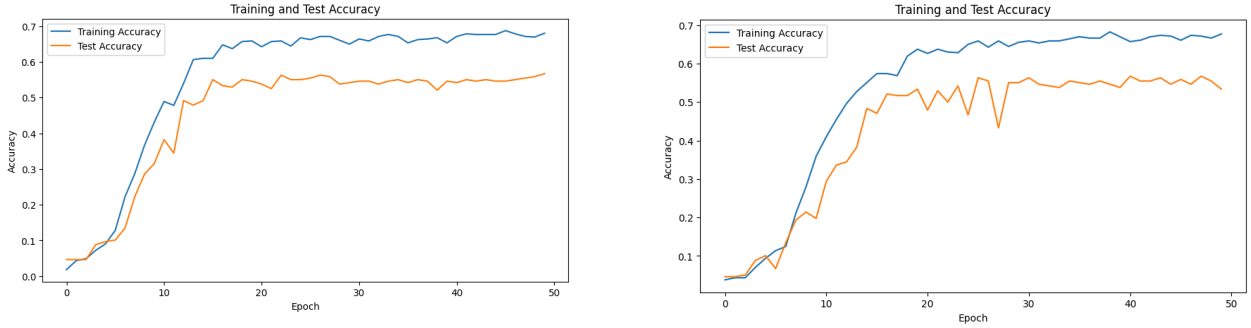


(a) (b)

Fig 9.3.1: Training and Test Accuracy Trend for GoogLeNet Over 30 Epochs

(a) without Data Augmentation (b) with Data Augmentation

.



(a) (b)

Fig 9.3.2: Training and Test Accuracy Trend for GoogLeNet Over 50 Epochs (learning rate = 0.0001)

(a) with Adam Optimizer (b) with RMSProp Optimizer

10. Model Performance Comparison

The experiments conducted on the three CNN architectures—1D CNN, AlexNet, and GoogLeNet—revealed distinct performance trends based on data augmentation, training duration, learning rate tuning, and optimizer selection. Initially, all models suffered from poor generalization due to the limited dataset size, with GoogLeNet showing the worst test accuracy. The introduction of data augmentation significantly improved test accuracy, particularly for 1D CNN and AlexNet, demonstrating its effectiveness in reducing overfitting.

Among the models, AlexNet exhibited strong performance with a balanced training-test accuracy trend, though its improvement plateaued beyond 30 epochs. GoogLeNet required extended training and hyperparameter tuning to show improvements, benefiting from 50 epochs of training with Adam (56.7% test accuracy) and RMSprop (53.4% test accuracy). The learning rate reduction to 0.0001 further improved accuracy, reinforcing the importance of fine-tuning hyperparameters. RMSprop performed better for deeper architectures, as observed in GoogLeNet, while Adam provided better stability for AlexNet.

Overall, the results highlight that AlexNet performed more consistently across configurations, while GoogLeNet showed greater sensitivity to training conditions. The 1D CNN was simpler and more efficient, making it an attractive choice for computationally constrained environments, though its baseline performance was lower. The findings emphasize that model complexity alone does not determine performance—training strategies such as data augmentation, learning rate tuning, and optimizer selection are crucial for achieving optimal results.

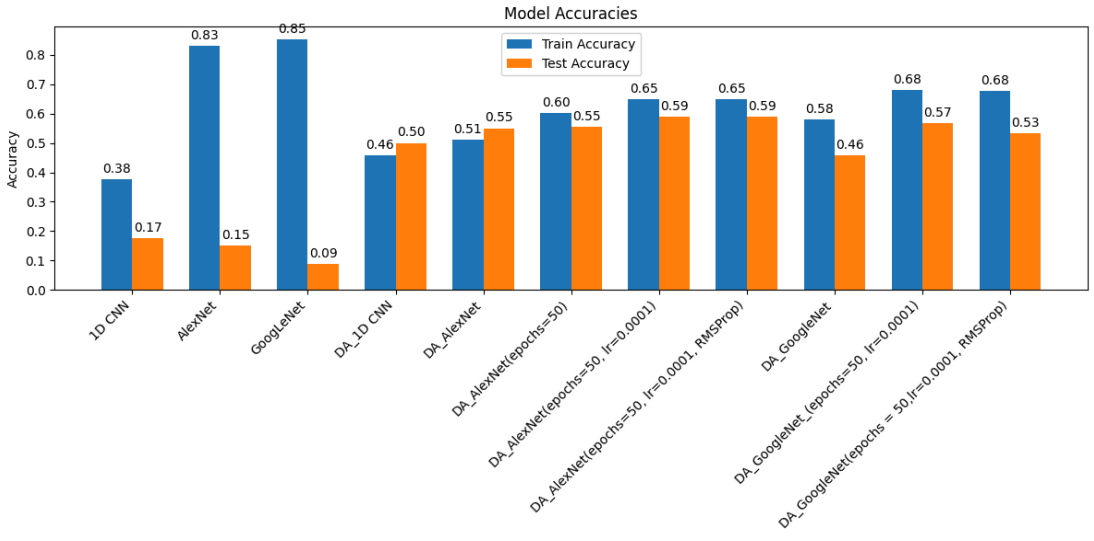


Fig 10.1: Model Performance Comparison

10.1 Learnings from the Project

This project provided valuable insights into the application of CNN architectures for audio classification, particularly in handling limited datasets. A key takeaway was the impact of data augmentation in mitigating overfitting, as models trained without it suffered from extremely low test accuracy. Extending training epochs and lowering the learning rate contributed to better model convergence, but gains were not always substantial beyond a certain point, highlighting the need for hyperparameter optimization.

The comparison between Adam and RMSprop showed that while Adam provided stable optimization across models, RMSprop was beneficial for deeper architectures like GoogLeNet. This suggests that optimizer selection should be architecture-dependent. Additionally, the study confirmed that AlexNet was more parameter-efficient in 1D than GoogLeNet, which contrasts with their typical performance in 2D applications, demonstrating the importance of tailoring architectures for specific input types.

Another key learning was that training complexity should be balanced with computational efficiency. While GoogLeNet had the highest parameter count, its performance gains were only marginal compared to AlexNet, indicating diminishing returns for deeper architectures when working with small datasets. These findings reinforce the necessity of systematic experimentation and fine-tuning to optimize model performance.

10.2 Scope for Improvement

Increasing dataset size is a primary improvement area, as deep learning models typically require large amounts of data to generalize effectively. Additional data augmentation techniques, such as pitch shifting and frequency masking, could further enhance variability in the training set.

Hyperparameter tuning can also be refined, particularly by implementing learning rate schedulers to dynamically adjust the learning rate based on training progress. Exploring alternative optimizers like SGD with momentum may provide further stability and generalization benefits.

From a model architecture perspective, transfer learning from pretrained CNN models designed for 1D time-series data could be explored to leverage existing knowledge rather than training from scratch. Additionally, attention mechanisms or hybrid CNN-RNN architectures could be tested to enhance temporal feature extraction.

Another improvement would be to evaluate model interpretability, using techniques such as Grad-CAM to visualize how models focus on different parts of the audio input. This could

provide insights into whether models are learning meaningful patterns or relying on spurious correlations.

Overall, by addressing these areas, the classification performance of CNNs on audio data could be further improved, making them more robust and effective for real-world applications.

11. References

1. Ibrahim, M. (2023) An Introduction to Audio Classification with Keras, Weights & Biases. Available at: https://wandb.ai/mostafaibrahim17/ml-articles/reports/An-Introduction-to-Audio-Classification-with-Keras--Vmlldzo0MDQzNDUy#audio-classification-made-easy:-the-best-libraries-for-the-job (Accessed: [16 March 2025]).
2. Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012) 'ImageNet Classification with Deep Convolutional Neural Networks', Advances in Neural Information Processing Systems, 25(2). DOI:10.1145/3065386.
3. Lekhuyen (2020) Implementation of GoogleNet on Keras, Medium. Available at: https://lekhuyen.medium.com/implementation-of-googlenet-on-keras-d9873aeed83c (Accessed: [16 March 2025]).
4. C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.