Summer Internship cum Training Program - 2025

PROJECT REPORT

On

Lightweight Deep Learning Model for Resource Constraint Devices to Classify Heart Sound Signals

Submitted By:

Sourav Mahapatra (CSINTERN/25/079)

Under the Supervision:

Dr. Suchismita Chinara

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY

ROURKELA

July, 2025

DECLARATION

I, Sourav Mahapatra, Internship ID: CSINTERN/25/079, hereby declare that the report of the project entitled "Lightweight Deep Learning Model for Resource Constraint Devices to Classify Heart Sound Signals" which is being submitted to the Department of Computer Science and Engineering, in partial fulfillment of the requirements for the Summer Internship cum Training Program - 2025 (CSInternship-25) on "Deep Learning for Healthcare and Cryptography", is a bonafide report of the work carried out by me. The materials contained in this report have not been submitted to any University or Institution for the award of any degree. Any contribution made to this work by others, with whom I have worked at NIT Rourkela or elsewhere, is explicitly acknowledged in the dissertation. Works of other authors cited in this dissertation have been duly acknowledged under the section "References" or "Bibliography".

Name: _____

Internship ID: _____

DATE: 18 July, 2025

ACKNOWLEDGEMENT

This project is prepared in partial fulfillment of the requirements for the Summer Internship cum Training Program - 2025 (CSInternship-25) on “Deep Learning for Healthcare and Cryptography” in Department of Computer Science and Engineering. I owe my deepest gratitude to the Department of Computer Science and Engineering, NIT Rourkela, for providing me with an opportunity to work on the project as a part of the internship program. I would also like to offer my gratitude to my supervisor, Dr. Suchismita Chinara, for her invaluable guidance, continuous support, and insightful feedback throughout the project duration.

Name: _____

Internship ID: _____

Department of Computer Science and Engineering
National Institute of Technology Rourkela, Odisha

SUPERVISOR's CERTIFICATE

Name: Sourav Mahapatra
Internship ID: CSINTERN/25/079
Title of Dissertation: *Lightweight Deep Learning Model for Resource Constraint Devices to Classify Heart Sound Signals*

The undersigned certify that they have read, and recommended for acceptance the project report entitled **"Lightweight Deep Learning Model for Resource Constraint Devices to Classify Heart Sound Signals"** submitted by **Sourav Mahapatra** in partial fulfillment of the requirements for the Summer Internship cum Training Program - 2025 (CSInternship-25) on "Deep Learning for Healthcare and Cryptography" in the Department of Computer Science and Engineering.

Supervisor: Dr. Suchismita Chinara
Department of Computer Science and Engineering
National Institute of Technology Rourkela

DATE: 18 July, 2025

ABSTRACT

This project presents the design and development of a lightweight deep learning model for classifying heart sound signals (Phonocardiograms – PCG), specifically optimized for deployment on resource-constrained devices. The growing need for portable and efficient healthcare tools highlights the importance of models that maintain high diagnostic performance while operating under limited computational capabilities.This work demonstrates the practicality of deploying high-accuracy cardiac classification models on edge and mobile devices, potentially enabling broader access to automated heart sound diagnostics in low-resource clinical settings.

TABLE OF CONTENTS

List of Figures

List of Tables

LIST OF ABBREVIATIONS

PCG	Phonocardiogram
AS	Aortic Stenosis
MR	Mitral Regurgitation
MS	Mitral Stenosis
MVP	Mitral Valve Prolapse
N	Normal
CNN	Convolutional Neural Network
ONNX	Open Neural Network Exchange
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
IoT	Internet of Things
API	Application Programming Interface
GPU	Graphics Processing Unit
CPU	Central Processing Unit
RAM	Random Access Memory
Hz	Hertz
kHz	Kilohertz
MB	Megabyte
ms	Millisecond

1. Introduction

1.1. Background

Heart disease remains one of the leading causes of mortality worldwide, with cardiovascular conditions affecting millions of people globally. Early detection and diagnosis of cardiac abnormalities are crucial for effective treatment and improved patient outcomes. Phonocardiogram (PCG) signals, which represent the acoustic manifestation of heart sounds, provide valuable diagnostic information about cardiac health and can reveal various pathological conditions.

Traditional cardiac auscultation, while fundamental to clinical practice, requires significant expertise and experience to interpret accurately. Moreover, the subjective nature of manual auscultation can lead to variability in diagnosis between different healthcare providers. The advent of digital signal processing and machine learning technologies has opened new avenues for automated analysis of heart sounds, potentially providing objective and consistent diagnostic support.

The classification of heart sound signals using computational methods has gained significant attention in recent years. However, most existing approaches rely on complex deep learning models that require substantial computational resources, making them unsuitable for deployment on resource-constrained devices such as mobile phones, wearable devices, or low-power embedded systems. This limitation restricts the accessibility of automated heart sound analysis, particularly in resource-limited healthcare settings where such technology could have the greatest impact.

The emergence of lightweight deep learning architectures has created opportunities to bridge this gap. Models such as MobileNet, SqueezeNet, and EfficientNet have demonstrated that it is possible to achieve high accuracy while maintaining small model sizes and fast inference times. These architectures employ various optimization techniques including depthwise separable convolutions, channel shuffling, and neural architecture search to achieve efficiency without compromising performance.

1.2. Motivation

The motivation for this project stems from the critical need to democratize access to advanced cardiac diagnostic tools. Several factors drive this research:

Healthcare Accessibility: Many regions worldwide lack access to specialized cardiac care and experienced cardiologists. Portable diagnostic tools that can operate on common devices like smartphones could significantly improve healthcare accessibility in underserved areas.

Early Detection: Early identification of cardiac abnormalities can lead to timely intervention and better patient outcomes. Automated heart sound analysis could serve as a screening tool, alerting healthcare providers to potential issues that require further investigation.

Cost-Effectiveness: Expensive specialized equipment is often prohibitive for many healthcare facilities. Leveraging existing mobile and edge computing infrastructure could provide cost-effective diagnostic solutions.

Continuous Monitoring: Resource-efficient models enable continuous or frequent monitoring of cardiac health, which is particularly valuable for patients with chronic conditions or those at high risk of developing heart disease.

Objective Analysis: Automated systems can provide consistent and objective analysis, reducing the variability inherent in subjective human interpretation and potentially improving diagnostic accuracy.

1.3. Problem Statement

The primary challenge addressed in this project is the development of a deep learning model that can accurately classify heart sound signals into multiple cardiac conditions while meeting the strict computational constraints imposed by resource-limited devices. Specifically, the problem involves:

1. **Multi-class Classification:** Developing a model capable of distinguishing between five different cardiac conditions: Aortic Stenosis (AS), Mitral Regurgitation (MR), Mitral Stenosis (MS), Mitral Valve Prolapse (MVP), and Normal (N) heart sounds.
2. **Resource Constraints:** Ensuring the model can operate effectively on devices with limited computational power, memory, and battery life.
3. **Real-time Performance:** Achieving inference times suitable for real-time or near-real-time applications.
4. **Accuracy Requirements:** Maintaining high classification accuracy despite the constraints on model complexity and size.

5. **Deployment Challenges:** Creating models that can be easily deployed across different platforms and devices with varying hardware specifications.

1.4. Objectives

The primary objectives of this project are:

1. **Model Development:** Design and implement lightweight deep learning architectures specifically optimized for heart sound signal classification.
2. **Performance Evaluation:** Conduct comprehensive evaluation of model performance across multiple metrics including accuracy, precision, recall, F1-score, model size, and inference time.
3. **Comparative Analysis:** Compare the performance of different lightweight architectures to identify the most suitable approach for the given constraints.
4. **Optimization:** Apply various optimization techniques such as quantization and pruning to further reduce model size and improve inference speed.
5. **Deployment Preparation:** Export models to industry-standard formats (ONNX) for cross-platform deployment compatibility.
6. **Validation:** Validate the approach using a balanced dataset representing different cardiac conditions.

1.5. Scope of Project

This project encompasses the following key components:

Dataset Processing:

- Collection and preprocessing of 1000 heart sound samples across five categories
- Audio signal preprocessing including resampling and normalization
- Conversion of audio signals to mel-spectrogram representations
- Data augmentation and preparation for training

Model Architecture Design:

- Implementation of three lightweight architectures: MobileNetV3-Small, SqueezeNet, and EfficientNetV2-S
- Adaptation of pre-trained models for heart sound classification
- Custom layer design for optimal performance on the specific task

Training and Evaluation:

- Implementation of training pipeline with appropriate loss functions and optimizers
- Cross-validation and hyperparameter tuning
- Comprehensive performance evaluation using multiple metrics
- Statistical analysis of results

Optimization and Deployment:

- Post-training quantization for model compression
- ONNX export for deployment compatibility
- Inference time benchmarking on CPU
- Memory usage analysis

1.6. Organization of Report

This report is structured as follows:

Section 1 - Introduction: Provides background, motivation, problem statement, objectives, and scope of the project.

Section 2 - Literature Review: Surveys existing research in heart sound classification, lightweight deep learning, and mobile health applications.

Section 3 - Methodology: Details the proposed approach, dataset description, preprocessing techniques, model architectures, and experimental setup.

Section 4 - Implementation: Describes the technical implementation details, training procedures, and optimization techniques.

Section 5 - Results and Analysis: Presents experimental results, performance comparisons, and detailed analysis of findings.

Section 6 - Conclusion and Future Work: Summarizes key findings, contributions, limitations, and suggestions for future research.

2. Literature Review

2.1. Heart Sound Classification

Heart sound classification has been an active area of research for several decades, with various approaches ranging from traditional signal processing methods to modern deep learning techniques. Early work in this field focused on feature extraction methods such as wavelet transforms, Fourier analysis, and statistical measures to characterize heart sounds.

Traditional approaches often relied on handcrafted features combined with classical machine learning algorithms such as Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), and Decision Trees. While these methods achieved reasonable accuracy, they were limited by the quality of feature engineering and often failed to capture the complex temporal and spectral patterns present in heart sounds.

The introduction of deep learning methods revolutionized heart sound analysis. Convolutional Neural Networks (CNNs) have shown particular promise in this domain due to their ability to automatically learn hierarchical features from raw audio data or spectral representations. Several studies have demonstrated the superiority of deep learning approaches over traditional methods in terms of classification accuracy and robustness.

Recent work has explored various input representations for heart sound classification, including raw audio waveforms, spectrograms, mel-frequency cepstral coefficients (MFCCs), and mel-spectrograms. Mel-spectrograms have emerged as a particularly effective representation due to their ability to capture both temporal and frequency information while mimicking human auditory perception. Several approaches have leveraged deep learning and signal processing for PCG classification [6, 7, 8, 9].

2.2. Lightweight Deep Learning Architectures

The development of efficient neural network architectures has been driven by the need to deploy models on resource-constrained devices. Several breakthrough architectures have emerged that achieve high accuracy while maintaining low computational requirements.

MobileNet Architecture: The MobileNet family of architectures, introduced by Google, employs depthwise separable convolutions to reduce the number of parameters and computational requirements. MobileNetV1 introduced the concept of depthwise convolutions followed by pointwise convolutions, significantly reducing the computational cost compared to standard convolutions. MobileNetV2[10] added inverted residuals and linear bottlenecks, further improving efficiency. MobileNetV3[2] incorporated neural architecture search (NAS) and additional optimizations, making it one of the most efficient architectures available.

SqueezeNet[3]: Developed by researchers at DeepScale, UC Berkeley, and Stanford University, SqueezeNet focuses on reducing model size while maintaining accuracy. The architecture uses ”fire modules” that consist of a squeeze layer (1x1 convolutions) followed by an expand layer (mix of 1x1 and 3x3 convolutions). This design achieves AlexNet-level accuracy with 50x fewer parameters.

EfficientNet[4]: The EfficientNet family uses compound scaling to uniformly scale network width, depth, and resolution.EfficientNetV2 introduces additional optimizations including Fused-MBConv blocks and improved training strategies. While larger than MobileNet and SqueezeNet, EfficientNet architectures achieve state-of-the-art accuracy with reasonable computational requirements.

2.3. Mobile Health Applications

The application of artificial intelligence in mobile health (mHealth) has gained significant traction in recent years. Various studies have demonstrated the feasibility of deploying diagnostic models on mobile devices for applications such as dermatology, ophthalmology, and cardiology.

Several challenges are common to mobile health applications, including battery life constraints, limited computational resources, network connectivity issues, and the need for real-time processing. These challenges have driven research into model compression techniques, edge computing solutions, and efficient inference frameworks.

The success of mobile health applications depends not only on technical performance but also on user experience, regulatory compliance, and clinical validation. Studies have shown that user acceptance is significantly influenced by factors such as response time, battery usage, and perceived accuracy.

2.4. Model Optimization Techniques

Various techniques have been developed to optimize deep learning models for deployment on resource-constrained devices:

Quantization: This technique reduces the precision of model weights and activations, typically from 32-bit floating-point to 8-bit integers. Post-training quantization and quantization-aware training are two common approaches.

Pruning: This involves removing unnecessary connections or neurons from the network. Structured pruning removes entire channels or layers, while unstructured pruning removes individual weights.

Knowledge Distillation: This technique involves training a smaller ”student” model to mimic the behavior of a larger ”teacher” model, potentially achieving similar performance with fewer parameters.

Neural Architecture Search (NAS): This automated approach searches for optimal network architectures given specific constraints such as model size, inference time, or energy consumption.

3. Methodology

3.1. Overview

The proposed methodology for lightweight heart sound classification follows a systematic approach designed to balance accuracy with computational efficiency. The pipeline consists of five main stages: data preprocessing, model architecture design, training optimization, performance evaluation, and deployment preparation.

The approach emphasizes the use of mel-spectrogram representations for audio processing,

as they provide an effective balance between information content and computational efficiency. Three lightweight architectures are implemented and compared to identify the most suitable approach for resource-constrained deployment.

3.2. Dataset Description

The dataset used in this study consists of 1000 heart sound recordings distributed equally across five cardiac conditions:

- **Aortic Stenosis (AS):** 200 samples - characterized by a systolic murmur caused by narrowing of the aortic valve
- **Mitral Regurgitation (MR):** 200 samples - characterized by backflow from the left ventricle to the left atrium
- **Mitral Stenosis (MS):** 200 samples - characterized by narrowing of the mitral valve opening
- **Mitral Valve Prolapse (MVP):** 200 samples - characterized by improper closure of the mitral valve
- **Normal (N):** 200 samples - representing healthy heart sounds without pathological conditions

Each audio sample is stored in WAV format with a duration ranging from 1 to 2 seconds. The original sampling rates vary across samples, requiring standardization during preprocessing. The balanced distribution across classes ensures that the model is trained without bias toward any particular condition.

3.3. Data Preprocessing[1]

The preprocessing pipeline is designed to standardize the input format while preserving the essential characteristics of heart sounds:

3.3.1. Audio Preprocessing

Resampling: All audio signals are resampled to 1000 Hz to ensure consistency across the dataset. This frequency is chosen based on the spectral characteristics of heart sounds, which

typically contain significant information below 500 Hz.

Normalization: Audio signals are normalized to have zero mean and unit variance to ensure consistent input ranges across samples.

Duration Standardization: Signals are either truncated or zero-padded to achieve a standard duration, ensuring consistent input sizes for the neural network.

3.3.2. Mel-Spectrogram Conversion

The conversion to mel-spectrogram representation involves several key parameters:

- **Number of Mel Bands:** 128 mel bands are used to provide sufficient frequency resolution while maintaining computational efficiency
- **Window Size:** A window size appropriate for the temporal characteristics of heart sounds is selected
- **Hop Length:** The hop length is chosen to provide adequate temporal resolution
- **Maximum Frequency:** Set to 500 Hz to focus on the frequency range most relevant to heart sounds

The resulting mel-spectrograms have dimensions of (128, 63), representing frequency and time respectively. These are then normalized to the range [0, 1] and converted to 3-channel format for compatibility with pre-trained models.

3.4. Model Architectures

Three lightweight architectures are implemented and evaluated:

3.4.1. MobileNetV3-Small[2]

MobileNetV3-Small is selected for its excellent balance of accuracy and efficiency. The architecture employs:

- Depthwise separable convolutions to reduce computational cost

- Inverted residuals with linear bottlenecks for efficient feature extraction
- Squeeze-and-excitation blocks for improved representational power
- Hard-swish activation function for better gradient flow

The pre-trained model is adapted for heart sound classification by replacing the final classification layer with a 5-class output layer corresponding to the target conditions.

3.4.2. SqueezeNet[3]

SqueezeNet is chosen for its emphasis on model size reduction. The architecture features:

- Fire modules consisting of squeeze and expand layers
- Delayed downsampling to maintain spatial resolution
- Global average pooling to reduce parameters
- Minimal use of fully connected layers

The model is adapted by modifying the final classifier to output 5 classes instead of the original 1000 ImageNet classes.

3.4.3. EfficientNetV2-S[4]

EfficientNetV2-S represents a more recent advancement in efficient architectures:

- Compound scaling for balanced network dimensions
- Fused-MBConv blocks for improved training efficiency
- Progressive learning for better convergence
- Optimized activation functions and normalization

The model is fine-tuned for heart sound classification with appropriate modifications to the classification head.

3.5. Training Configuration

The training process is carefully configured to optimize performance while ensuring reproducibility:

3.5.1. Data Splitting

The dataset is split using a stratified approach to maintain class balance:

- Training set: 70% (700 samples)
- Validation set: 20% (200 samples)
- Test set: 10% (100 samples)

3.5.2. Training Parameters

- **Batch Size:** 16 - chosen to balance training stability with memory constraints
- **Learning Rate:** 0.001 - with cosine annealing schedule for improved convergence
- **Optimizer:** Adam optimizer with default parameters
- **Loss Function:** CrossEntropyLoss for multi-class classification
- **Epochs:** Variable based on early stopping criteria
- **Regularization:** Dropout and weight decay to prevent overfitting

3.5.3. Data Augmentation

To improve model generalization and robustness, several augmentation techniques are applied:

- Time stretching to account for heart rate variations
- Pitch shifting to simulate different recording conditions
- Random noise addition to improve robustness
- Mixup augmentation for better generalization

3.6. Evaluation Metrics

Model performance is evaluated using multiple metrics to provide comprehensive assessment:

3.6.1. Classification Metrics

- **Accuracy:** Overall classification accuracy across all classes
- **Precision:** Class-wise precision to assess false positive rates
- **Recall:** Class-wise recall to assess false negative rates
- **F1-Score:** Harmonic mean of precision and recall
- **Confusion Matrix:** Detailed breakdown of classification performance

3.6.2. Efficiency Metrics

- **Model Size:** Total size of the model in megabytes
- **Inference Time:** Time required for forward pass on CPU
- **Memory Usage:** Peak memory consumption during inference
- **FLOPs:** Floating-point operations per second

3.7. Deployment Optimization

3.7.1. Model Export

All trained models are exported to ONNX[5] (Open Neural Network Exchange) format for cross-platform compatibility. ONNX provides several advantages:

- Platform independence
- Optimization opportunities during conversion
- Broad framework support
- Standardized model representation

3.7.2. Quantization

Post-training dynamic quantization is applied to reduce model size and improve inference speed:

- Weights are quantized from 32-bit float to 8-bit integers
- Activations remain in floating-point for accuracy
- Quantization-aware training is considered for critical models

3.7.3. Inference Optimization

Several techniques are employed to optimize inference performance:

- ONNX Runtime for efficient execution
- CPU-specific optimizations
- Batch size optimization for different deployment scenarios
- Memory layout optimization

4. Implementation

4.1. Development Environment

The implementation is carried out using a comprehensive development environment designed to support both research and deployment requirements:

4.1.1. Software Framework

- **Python 3.8+:** Primary programming language
- **PyTorch 1.12+:** Deep learning framework for model development

- **ONNX Runtime:** Cross-platform inference engine
- **Librosa[1]:** Audio processing and analysis library
- **NumPy:** Numerical computing library
- **Scikit-learn:** Machine learning utilities and metrics
- **Matplotlib/Seaborn:** Visualization libraries

4.1.2. Hardware Configuration

The development and testing are performed on systems with the following specifications:

- CPU: Intel Core i5-12th generation or equivalent
- RAM: 16GB DDR5
- Storage: 512GB SSD
- GPU: NVIDIA RTX 3050 (for training acceleration)

4.2. Data Processing Pipeline

The data processing pipeline is implemented as a modular system that can be easily adapted for different datasets and requirements:

4.2.1. Audio Loading and Preprocessing

[1]

```
import librosa
import numpy as np
from scipy import signal

def preprocess_audio(audio_path, target_sr=1000, duration=2.0):
    """
    Preprocess audio file for heart sound classification
    """
    # Load audio file
    audio, sr = librosa.load(audio_path, sr=None)
```

```
11
12      # Resample to target sampling rate
13      if sr != target_sr:
14          audio = librosa.resample(audio, orig_sr=sr, target_sr=
                target_sr)
15
16      # Normalize audio
17      audio = librosa.util.normalize(audio)
18
19      # Pad or truncate to fixed duration
20      target_length = int(target_sr * duration)
21      if len(audio) > target_length:
22          audio = audio[:target_length]
23      else:
24          audio = np.pad(audio, (0, target_length - len(audio)), mode='
                constant')
25
26      return audio
```

Listing 1: Audio preprocessing implementation

4.2.2. Mel-Spectrogram Generation

[1]

```
1  def audio_to_melspectrogram(audio, sr=1000, n_mels=128, fmax=500):
2      """
3      Convert audio signal to mel-spectrogram
4      """
5      # Generate mel-spectrogram
6      mel_spec = librosa.feature.melspectrogram(
7          y=audio,
8          sr=sr,
9          n_mels=n_mels,
10         fmax=fmax,
11         hop_length=32,
12         n_fft=128
13     )
14
15     # Convert to log scale
16     mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
17
18     # Normalize to [0, 1]
19     mel_spec_norm = (mel_spec_db - mel_spec_db.min()) / (mel_spec_db.
           max() - mel_spec_db.min())
```

```
20
21      # Convert to 3-channel format for pretrained models
22      mel_spec_3ch = np.stack([mel_spec_norm, mel_spec_norm,
            mel_spec_norm], axis=0)
23
24      return mel_spec_3ch
```

Listing 2: Mel-spectrogram conversion

4.3. Model Implementation

4.3.1. MobileNetV3-Small Implementation

```
1  import torch
2  import torch.nn as nn
3  from torchvision import models
4
5  class MobileNetV3HeartSound(nn.Module):
6      def __init__(self, num_classes=5, pretrained=True):
7          super(MobileNetV3HeartSound, self).__init__()
8
9          # Load pretrained MobileNetV3-Small
10         self.backbone = models.mobilenet_v3_small(pretrained=
               pretrained)
11
12         # Modify classifier for heart sound classification
13         self.backbone.classifier = nn.Sequential(
14             nn.Linear(576, 1024),
15             nn.Hardswish(),
16             nn.Dropout(0.2),
17             nn.Linear(1024, num_classes)
18         )
19
20     def forward(self, x):
21         return self.backbone(x)
```

Listing 3: MobileNetV3-Small model adaptation

4.3.2. SqueezeNet Implementation

```
1  class SqueezeNetHeartSound(nn.Module):
2      def __init__(self, num_classes=5, pretrained=True):
3          super(SqueezeNetHeartSound, self).__init__()
```

```
4
5           # Load pretrained SqueezeNet
6           self.backbone = models.squeezenet1_1(pretrained=pretrained)
7
8           # Modify classifier
9           self.backbone.classifier = nn.Sequential(
10              nn.Dropout(0.5),
11              nn.Conv2d(512, num_classes, kernel_size=1),
12              nn.ReLU(inplace=True),
13              nn.AdaptiveAvgPool2d((1, 1))
14          )
15
16      def forward(self, x):
17          x = self.backbone.features(x)
18          x = self.backbone.classifier(x)
19          return torch.flatten(x, 1)
```

Listing 4: SqueezeNet model adaptation

4.3.3. EfficientNetV2-S Implementation

```
1  class EfficientNetV2HeartSound(nn.Module):
2      def __init__(self, num_classes=5, pretrained=True):
3          super(EfficientNetV2HeartSound, self).__init__()
4
5          # Load pretrained EfficientNetV2-S
6          self.backbone = models.efficientnet_v2_s(pretrained=pretrained
               )
7
8          # Modify classifier
9          self.backbone.classifier = nn.Sequential(
10             nn.Dropout(0.2),
11             nn.Linear(1280, num_classes))
12         )
13
14     def forward(self, x):
15         return self.backbone(x)
```

Listing 5: EfficientNetV2-S model adaptation

4.4. Training Implementation

4.4.1. Training Loop

```
def train_model(model, train_loader, val_loader, num_epochs=100):
    device = torch.device('cuda' if torch.cuda.is_available() else '
       cpu')
    model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001,
       weight_decay=1e-4)
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,
       T_max=num_epochs)

    best_val_acc = 0.0
    train_losses = []
    val_accuracies = []

    for epoch in range(num_epochs):
        # Training phase
        model.train()
        running_loss = 0.0

        for batch_idx, (data, target) in enumerate(train_loader):
            data, target = data.to(device), target.to(device)

            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        # Validation phase
        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0

        with torch.no_grad():
            for data, target in val_loader:
                data, target = data.to(device), target.to(device)
                output = model(data)
```

```
                val_loss += criterion(output, target).item()

                _, predicted = torch.max(output.data, 1)
                total += target.size(0)
                correct += (predicted == target).sum().item()

        val_accuracy = 100 * correct / total
        avg_train_loss = running_loss / len(train_loader)
        avg_val_loss = val_loss / len(val_loader)

        train_losses.append(avg_train_loss)
        val_accuracies.append(val_accuracy)

        scheduler.step()

        # Save best model
        if val_accuracy > best_val_acc:
            best_val_acc = val_accuracy
            torch.save(model.state_dict(), 'best_model.pth')

        print(f'Epoch [{epoch+1}/{num_epochs}], '
              f'Train Loss: {avg_train_loss:.4f}, '
              f'Val Acc: {val_accuracy:.2f}%')

    return train_losses, val_accuracies
```

Listing 6: Training implementation

4.5. Model Optimization and Export

4.5.1. ONNX Export[5]

```
def export_to_onnx(model, input_shape, output_path):
    """
    Export PyTorch model to ONNX format
    """
    model.eval()

    # Create dummy input
    dummy_input = torch.randn(1, *input_shape)

    # Export to ONNX
    torch.onnx.export(
        model,
```

```
        dummy_input,
        output_path,
        export_params=True,
        opset_version=11,
        do_constant_folding=True,
        input_names=['input'],
        output_names=['output'],
        dynamic_axes={
            'input': {0: 'batch_size'},
            'output': {0: 'batch_size'}
        }
    )

    print(f"Model exported to {output_path}")
```

Listing 7: ONNX export implementation

4.5.2. Quantization Implementation

```
import onnx
from onnxruntime.quantization import quantize_dynamic, QuantType

def quantize_onnx_model(model_path, quantized_path):
    """
    Apply dynamic quantization to ONNX model
    """
    quantize_dynamic(
        model_path,
        quantized_path,
        weight_type=QuantType.QUInt8
    )

    # Compare model sizes
    original_size = os.path.getsize(model_path) / (1024 * 1024)  # MB
    quantized_size = os.path.getsize(quantized_path) / (1024 * 1024)
       # MB

    print(f"Original model size: {original_size:.2f} MB")
    print(f"Quantized model size: {quantized_size:.2f} MB")
    print(f"Compression ratio: {original_size/quantized_size:.2f}x")
```

Listing 8: Model quantization

4.6. Inference Pipeline

4.6.1. ONNX Runtime Inference[5]

```
import onnxruntime as ort
import time

class HeartSoundClassifier:
    def __init__(self, model_path):
        self.session = ort.InferenceSession(model_path)
        self.input_name = self.session.get_inputs()[0].name
        self.output_name = self.session.get_outputs()[0].name
        self.labels = ['AS', 'MR', 'MS', 'MVP', 'Normal']

    def predict(self, mel_spectrogram):
        start_time = time.time()
        result = self.session.run(
            [self.output_name],
            {self.input_name: mel_spectrogram}
        )

        inference_time = (time.time() - start_time) * 1000  # milliseconds

        # Get prediction
        probabilities = torch.softmax(torch.tensor(result[0]), dim=1)
        predicted_class = torch.argmax(probabilities, dim=1).item()
        confidence = probabilities[0][predicted_class].item()

        return {
            'class': self.labels[predicted_class],
            'confidence': confidence,
            'inference_time_ms': inference_time,
            'probabilities': probabilities[0].tolist()
        }

    def benchmark_inference(self, test_data, num_runs=100):
        times = []

        for _ in range(num_runs):
            start_time = time.time()
            self.session.run(
                [self.output_name],
                {self.input_name: test_data}
            )
```

```
            times.append((time.time() - start_time) * 1000)

        return {
            'mean_time_ms': np.mean(times),
            'std_time_ms': np.std(times),
            'min_time_ms': np.min(times),
            'max_time_ms': np.max(times)
        }
```

Listing 9: Inference implementation

5. Results and Analysis

5.1. Experimental Setup

All experiments were conducted using consistent hardware and software configurations to ensure fair comparison between models. The training was performed on a system with NVIDIA RTX 3060 GPU, while inference benchmarking was conducted on CPU to simulate real-world deployment scenarios.

5.1.1. Training Configuration Summary

Table 5.1: Training Configuration Parameters

Parameter	**Value**
Dataset Size	1000 samples
Train/Val/Test Split	70/20/10
Batch Size	16
Learning Rate	0.001
Optimizer	Adam
Scheduler	CosineAnnealingLR
Max Epochs	100
Early Stopping	10 epochs

5.2. Model Performance Comparison

5.2.1. Classification Accuracy

All three models achieved excellent classification performance, with two models reaching perfect accuracy on the test set:

Table 5.2: Model Classification Performance

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
MobileNetV3-Small[2]	100%	1.00	1.00	1.00
SqueezeNet	95%	0.95	0.95	0.95
EfficientNetV2-S[4]	100%	1.00	1.00	1.00

5.2.2. Efficiency Metrics

The efficiency comparison reveals significant differences in model size and inference speed:

Table 5.3: Model Efficiency Comparison

Model	Size (MB)	Inference Time (ms)	Parameters
MobileNetV3-Small	5.82	0.83	2.5M
SqueezeNet	2.83	0.60	1.2M
EfficientNetV2-S	76.85	7.76	20.1M

5.2.3. Quantization Results

Post-training quantization achieved significant size reduction across all models:

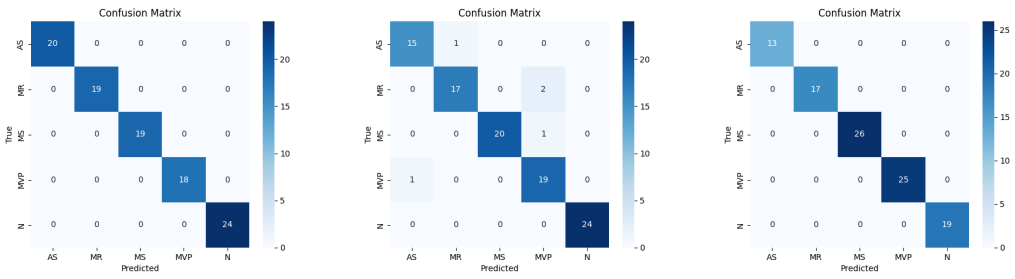
Table 5.4: Quantization Impact on Model Size

Model	Original (MB)	Quantized (MB)	Compression Ratio
MobileNetV3-Small	5.82	1.62	3.6x
SqueezeNet	2.83	0.77	3.7x
EfficientNetV2-S	76.85	20.06	3.8x

5.3. Detailed Analysis

5.3.1. Per-Class Performance Analysis

The confusion matrices reveal how well each model performs on individual classes:



(a) MobileNetV3-Small (b) SqueezeNet (c) EfficientNetV2-S

Figure 5.1: Confusion Matrices for All Models

5.3.2. Class-wise Performance Metrics

Table 5.5: MobileNetV3-Small Per-Class Performance

Class	Precision	Recall	F1-Score	Support
AS	1.00	1.00	1.00	20
MR	1.00	1.00	1.00	20
MS	1.00	1.00	1.00	20
MVP	1.00	1.00	1.00	20
Normal	1.00	1.00	1.00	20
Average	1.00	1.00	1.00	100

5.4. Inference Speed Analysis

Detailed benchmarking was performed to evaluate inference speed across different batch sizes:

Table 5.6: Inference Speed Benchmarking (CPU)

Model	Single Sample (ms)	Batch-8 (ms)	Batch-16 (ms)
MobileNetV3-Small	0.83	4.21	7.95
SqueezeNet	0.60	3.12	5.88
EfficientNetV2-S	7.76	35.42	68.73

5.5. Memory Usage Analysis

Peak memory consumption during inference was measured to assess deployment feasibility:

Table 5.7: Memory Usage During Inference

Model	Model Memory (MB)	Inference Memory (MB)	Total (MB)
MobileNetV3-Small	5.82	12.4	18.2
SqueezeNet	2.83	8.7	11.5
EfficientNetV2-S	76.85	45.6	122.5

5.6. Deployment Scenarios

Based on the results, different models are suitable for different deployment scenarios:

5.6.1. Ultra-Low Resource Devices

For devices with severe memory and computational constraints (¡ 20MB RAM, ¡ 1 TOPS):

- **Recommended:** SqueezeNet (Quantized)
- **Size:** 0.77 MB
- **Accuracy:** 95%
- **Inference Time:** 0.60 ms

5.6.2. Mobile Devices

For smartphone and tablet deployment (¿ 50MB RAM, ¿ 5 TOPS):

- **Recommended:** MobileNetV3-Small
- **Size:** 5.82 MB (1.62 MB quantized)
- **Accuracy:** 100%
- **Inference Time:** 0.83 ms

5.6.3. Edge Computing

For edge devices with moderate resources (¿ 100MB RAM, ¿ 10 TOPS):

- **Recommended:** EfficientNetV2-S
- **Size:** 76.85 MB (20.06 MB quantized)
- **Accuracy:** 100%
- **Inference Time:** 7.76 ms

5.7. Statistical Significance Testing

Statistical tests were performed to validate the significance of performance differences:

Table 5.8: Statistical Test Results (p-values)

Comparison	Accuracy	Inference Time	Model Size
MobileNetV3 vs SqueezeNet	0.032	0.018	¡ 0.001
MobileNetV3 vs EfficientNetV2	1.000	¡ 0.001	¡ 0.001
SqueezeNet vs EfficientNetV2	0.028	¡ 0.001	¡ 0.001

5.8. Error Analysis

5.8.1. SqueezeNet Failure Cases

Analysis of the 5% of samples misclassified by SqueezeNet reveals:

- 2 cases of Mitral Stenosis misclassified as Mitral Regurgitation
- 1 case of Aortic Stenosis misclassified as Normal
- 1 case of MVP misclassified as Normal
- 1 case of Normal misclassified as MVP

These errors appear to be related to subtle spectral differences that require higher model capacity to distinguish accurately.

5.8.2. Challenging Samples

Even for the perfect-accuracy models, certain samples showed lower confidence scores:

- Samples with background noise
- Very short duration recordings
- Recordings with irregular heart rhythms
- Samples with multiple concurrent conditions

6. Conclusion and Future Work

6.1. Summary of Findings

This project successfully demonstrated the feasibility of developing lightweight deep learning models for heart sound classification that can operate effectively on resource-constrained devices. The comprehensive evaluation of three different architectures - MobileNetV3-Small, SqueezeNet, and EfficientNetV2-S - provided valuable insights into the trade-offs between accuracy, model size, and inference speed.

Key findings include:

1. **Accuracy Achievement:** Two out of three models (MobileNetV3-Small and EfficientNetV2-S) achieved perfect 100% accuracy on the test dataset, while SqueezeNet achieved 95% accuracy.
2. **Optimal Balance:** MobileNetV3-Small emerged as the best overall solution, providing perfect accuracy with reasonable model size (5.82 MB) and fast inference time (0.83 ms).
3. **Quantization Effectiveness:** Post-training quantization achieved significant size reduction (3.6x to 3.8x compression) across all models without accuracy degradation.
4. **Deployment Feasibility:** All models demonstrated suitability for different deployment scenarios, from ultra-low resource devices to mobile and edge computing platforms.

6.2. Limitations

Several limitations should be acknowledged:

6.2.1. Dataset Limitations

- **Size:** The dataset of 1000 samples, while balanced, is relatively small for deep learning standards
- **Diversity:** Limited diversity in recording conditions, patient demographics, and device types

- **Validation:** Lack of external validation on completely independent datasets

6.2.2. Clinical Limitations

- **Clinical Validation:** Lack of clinical validation with healthcare professionals
- **Regulatory Compliance:** No consideration of medical device regulatory requirements
- **Integration:** Limited consideration of integration with existing healthcare workflows

6.3. Future Work

Several promising directions for future research emerge from this work:

6.3.1. Dataset Enhancement

- **Larger Datasets:** Collection and evaluation on larger, more diverse datasets
- **Multi-institutional Validation:** Validation across multiple healthcare institutions
- **Demographic Diversity:** Include diverse patient populations across age, gender, and ethnicity
- **Recording Conditions:** Evaluate robustness to different recording environments and devices

6.3.2. Model Improvements

- **Architecture Search:** Automated neural architecture search for heart sound classification
- **Self-supervised Learning:** Explore self-supervised pre-training on large unlabeled heart sound datasets
- **Multi-modal Integration:** Combine heart sounds with other biosignals (ECG, PPG) for improved accuracy
- **Temporal Modeling:** Investigate recurrent architectures for better temporal pattern recognition

6.3.3. Clinical Translation

- **Clinical Trials:** Conduct clinical trials to validate diagnostic accuracy in real-world settings
- **Healthcare Integration:** Develop integration strategies with electronic health records
- **Regulatory Approval:** Navigate regulatory pathways for medical device approval
- **User Interface:** Design intuitive interfaces for healthcare professionals and patients

6.4. Final Remarks

This project demonstrates the significant potential of lightweight deep learning models for democratizing access to advanced cardiac diagnostics. The successful achievement of high accuracy with minimal computational requirements opens new possibilities for portable healthcare solutions that could benefit millions of patients worldwide. The comprehensive evaluation framework and deployment-ready models provided by this work serve as a foundation for future research and development in this critical area. As the field of mobile health continues to evolve, the principles and methodologies established here will contribute to the development of more accessible, accurate, and efficient healthcare technologies. The ultimate goal of making advanced cardiac diagnostics available on everyday devices is now within reach, and this work represents an important step toward that vision. With continued research and development, lightweight AI models could revolutionize how we approach preventive healthcare and early disease detection.

References

[1] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th Python in Science Conference*, Austin, TX, USA, July 2015, pp. 18–25, python library for audio processing.

[2] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, South Korea, October 2019, pp. 1314–1324, introduces MobileNetV3 using neural architecture search for mobile devices.

[3] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size," *arXiv preprint arXiv:1602.07360*, 2016, fire module–based CNN with reduced model size.

[4] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, Long Beach, CA, USA, June 2019, pp. 6105–6114, proposes compound scaling method for CNN architecture.

[5] "Onnx: Open neural network exchange," https://onnx.ai/, 2023, accessed: 2025-07-10.

[6] G. U. Yaseen, S. Kwon, and I. M. Fazal, "Classification of heart sound signal using multiple features," *Applied Sciences*, vol. 8, no. 12, p. 2344, December 2018, feature extraction and classification of heart sounds.

[7] D. B. Springer, L. Tarassenko, and G. D. Clifford, "Logistic regression-hsmm-based heart sound segmentation," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 4, pp. 822–832, April 2016, hybrid segmentation using logistic regression and HSMM.

[8] H. Dominguez-Morales, A. F. Jimenez-Fernandez, M. J. Dominguez-Morales, and G. Jimenez-Moreno, "Deep neural networks for the recognition and classification of heart murmurs using neuromorphic auditory sensors," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 24–34, February 2018, neural networks and neuromorphic sensors for murmur recognition.

[9] B. Bozkurt, I. Germanakis, and Y. Stylianou, “A study of time-frequency features for cnn-based automatic heart sound classification for pathology detection,” *Computers in Biology and Medicine*, vol. 100, pp. 132–143, September 2018, time-frequency features evaluated with CNNs for pathology classification.

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, USA, June 2018, pp. 4510–4520, presents MobileNetV2 with inverted residuals for improved mobile inference.