Software Design Document

Project: Breast Cancer Detection System
Team: Woroma Dimkpa, Kahlel Cardona, Taratong Dolinsky
Date: 02/17/2026

1. Introduction

1.1 Purpose

This Software Design Document (SDD) describes the technical architecture and detailed design of the Breast Cancer Detection System defined in the Software Requirements Specification (SRS). While the SRS specifies what the system must accomplish, this document explains how the system is structured and implemented to satisfy those requirements.

The system is designed to automatically classify mammogram images as benign or malignant using convolutional neural networks (CNNs). The design focuses on modularity, scalability, reproducibility, and maintainability to ensure that the system can support experimentation with different model architectures and preprocessing strategies.

This document provides architectural diagrams, module descriptions, data flow explanations, class structures, deployment architecture, and design rationales. It ensures traceability between requirements and implementation components, thereby supporting verification, testing, and future system enhancements.

1.2 Design Goals

The primary design goals of the system are:

Modularity: Each functional component (data loading, preprocessing, training, evaluation) is implemented as an independent module.
Scalability: The system must support both small experimental subsets and the full CBIS-DDSM dataset.
Reproducibility: Fixed random seeds and controlled data splits ensure consistent experimental results.
Maintainability: The architecture allows easy replacement or extension of CNN models.
Performance Efficiency: GPU acceleration is leveraged to reduce training time.

These goals directly support the non-functional requirements specified in the SRS.

2. Overall System Architecture

2.1 Architectural Style

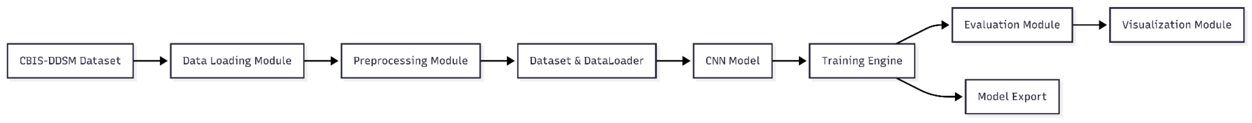
The system follows a layered modular architecture combined with a machine learning pipeline pattern. Each layer is responsible for a specific aspect of processing, and communication between layers occurs through well-defined interfaces.

The architecture consists of the following layers:

1. **Data Layer** – Handles dataset ingestion and validation.
2. **Processing Layer** – Performs image preprocessing and augmentation.
3. **Model Layer** – Defines CNN architectures and transfer learning models.
4. **Training Layer –** Executes forward and backward propagation.
5. **Evaluation Layer** – Computes performance metrics.
6. **Visualization and Persistence Layer** – Displays results and exports trained models.

This layered separation improves maintainability and supports future system extensions without modifying core components.

Figure 1: High-Level System Architecture



3. Component-Level Design

3.1 Data Loading Module

The Data Loading Module is responsible for importing mammogram images and associated metadata from the CBIS-DDSM dataset. It reads CSV files containing image paths and class labels, validates file availability, and handles exceptions such as missing or corrupted files.

Images may be stored in DICOM or JPEG format. When DICOM files are used, pixel arrays are extracted and converted into normalized floating-point arrays. Error handling mechanisms ensure that corrupted images are logged rather than causing system termination, thereby satisfying reliability requirements.

This module produces structured dataset objects that are passed to the preprocessing layer.

3.2 Preprocessing Module

The Preprocessing Module transforms raw mammogram images into standardized input suitable for CNN models. Given that mammogram images vary significantly in resolution and contrast, preprocessing ensures consistency and computational efficiency.

Key preprocessing steps include resizing images to 128×128 for baseline models or 224×224 for transfer learning architectures. Pixel intensities are normalized to a standardized range to stabilize model training. Optional augmentation techniques such as horizontal flipping and brightness adjustments are applied during training to improve generalization performance.

This module ensures compatibility with multiple CNN architectures and contributes to improved robustness and model performance.

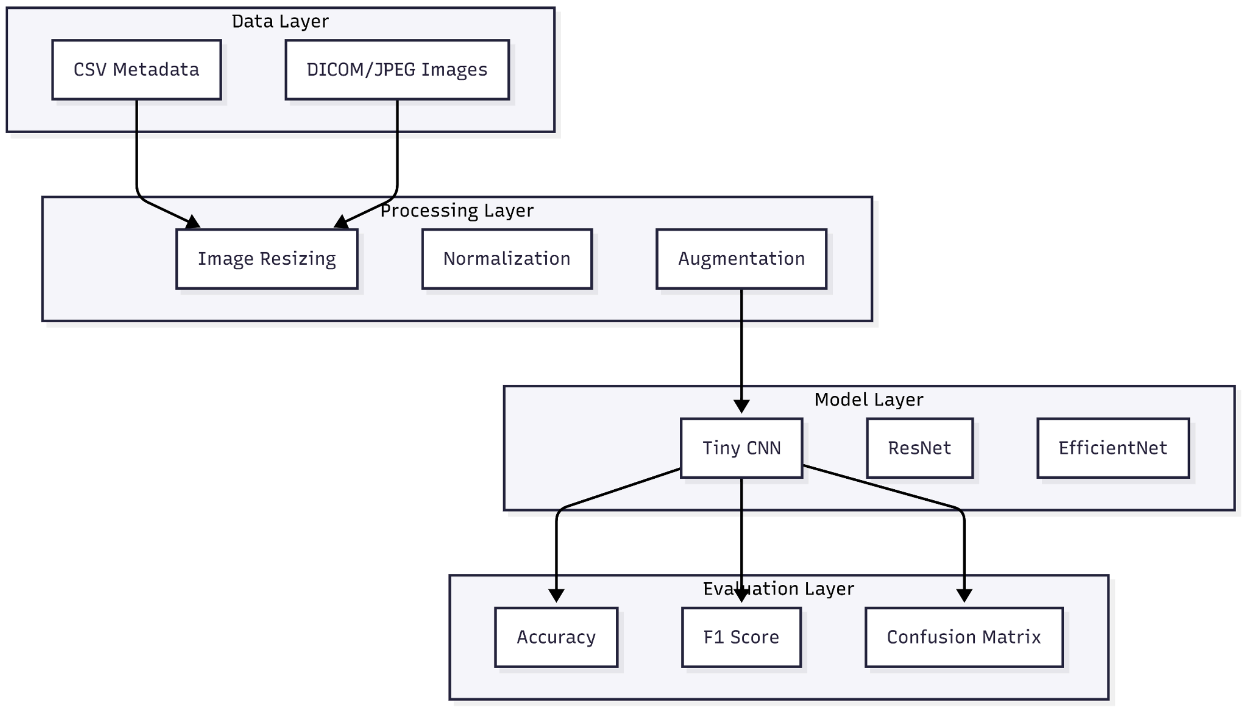
3.3 Dataset and DataLoader Module

The Dataset and DataLoader module bridges preprocessing and model training. It implements a structured dataset class responsible for retrieving individual samples and applying preprocessing transformations dynamically.

The DataLoader organizes samples into batches, enables shuffling during training, and supports stratified splitting into training, validation, and test subsets. Stratified splitting ensures class distribution consistency across subsets, mitigating bias caused by class imbalance.

Batch processing significantly improves GPU utilization and training efficiency.

Figure 2: Component Interaction Diagram



3.4 Model Architecture Module

The Model Architecture Module defines the deep learning models used for classification. The system supports both a baseline Tiny CNN and transfer learning models such as ResNet and EfficientNet.

The Tiny CNN consists of two convolutional layers followed by max-pooling operations and fully connected layers. This architecture is sufficient for initial experimentation and rapid validation.

Transfer learning models leverage pre-trained weights from large-scale datasets. The first convolutional layer may be adapted for single-channel input, and the final classification layer is modified to output two classes. Transfer learning accelerates convergence and often yields improved classification accuracy compared to training from scratch.

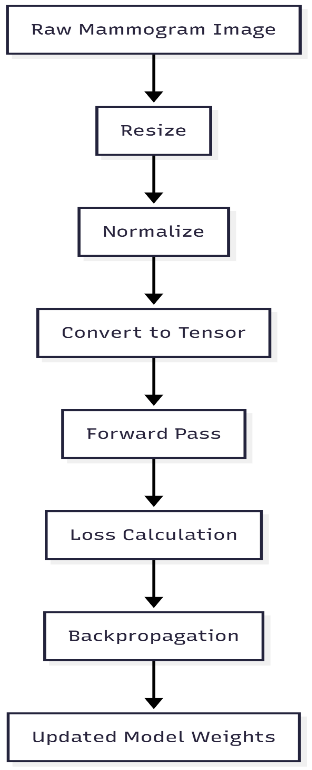
3.5 Training Engine Module

The Training Engine coordinates the optimization process. During each training epoch, batches of images are passed through the model to compute predictions. The loss function quantifies prediction error, and gradients are computed via backpropagation.

The optimizer updates model parameters iteratively to minimize classification error. Class-weighted loss functions are used to address class imbalance between benign and malignant samples.

Training progress is monitored using validation accuracy and loss metrics to detect potential overfitting.

Figure 3: Data Flow During Training Phase



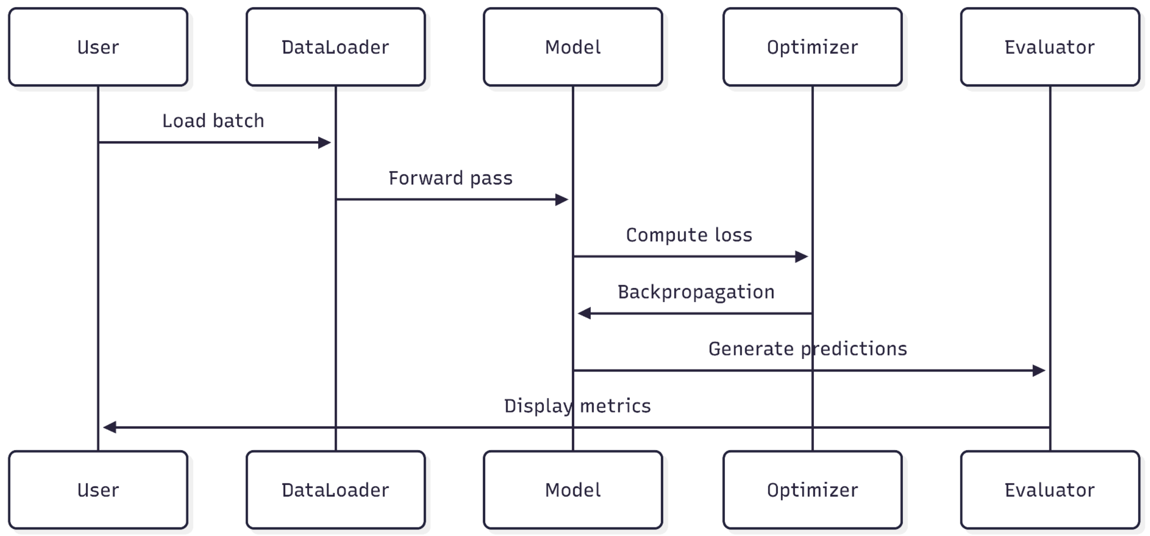
4. Data Flow Design

The system's data flow begins with ingestion of raw mammogram images and metadata. After preprocessing, data is converted into tensor representations and batched for efficient computation.

During training, the model performs forward propagation to generate predictions. The loss function computes classification error, and gradients are propagated backward through the network. Updated weights improve classification performance iteratively across epochs.

Evaluation metrics are computed using validation or test sets to assess generalization capability.

Figure 4: Training Sequence Diagram

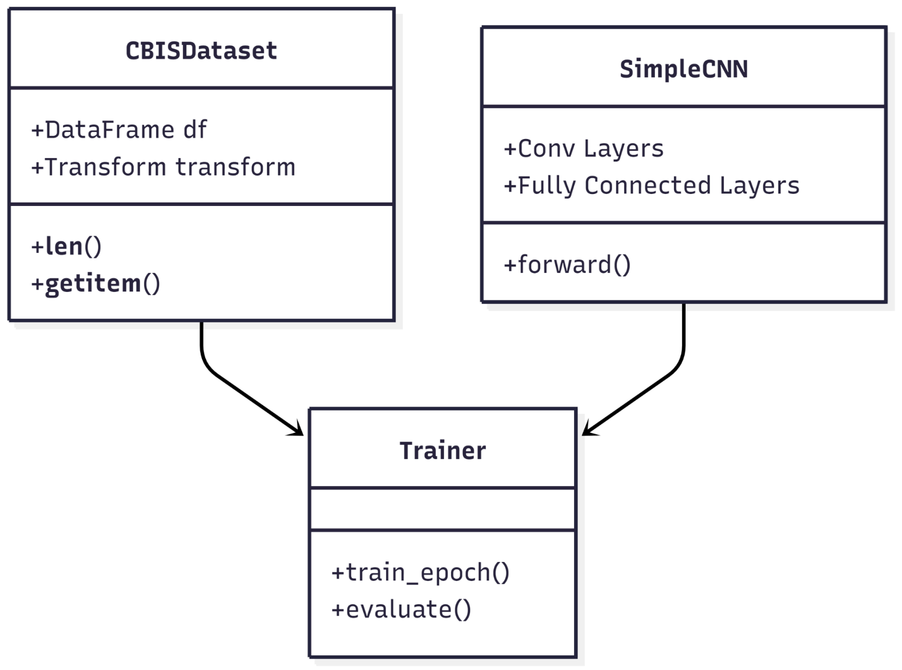


5. Class Design

The system implements object-oriented principles to structure core components. The dataset class encapsulates data retrieval logic, the CNN class encapsulates model structure and forward propagation logic, and the trainer class coordinates optimization and evaluation processes.

This separation of concerns improves maintainability and allows independent testing of modules.

Figure 5: Class Diagram

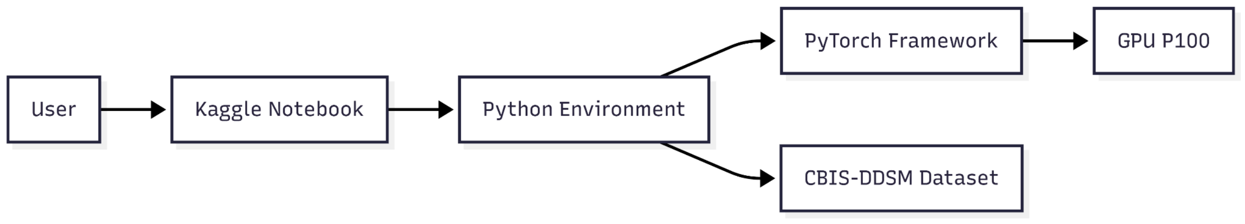


6. Deployment Architecture

The system is designed for execution in cloud-based environments such as Kaggle notebooks with GPU acceleration. The deployment architecture includes a Python runtime environment, deep learning frameworks, and access to the CBIS-DDSM dataset.

GPU resources such as Tesla T4 or P100 accelerate matrix computations required for CNN training. Dependency management is handled through pip or conda environments to ensure portability across Windows, macOS, and Linux systems.

Figure 6: Deployment Architecture



7. Design Rationale

PyTorch was selected due to its dynamic computation graph and flexibility for research-oriented development. CNN architectures were chosen because of their demonstrated success in medical image analysis. Transfer learning was incorporated to improve performance while reducing training time.

The modular design ensures scalability and maintainability, allowing additional architectures or evaluation methods to be integrated without significant restructuring.

8. Traceability to SRS

SRS Requirement	**Description (from SRS)**	**Design Implementation**
FR1	Load mammogram images and metadata	Data Loading Module
FR2	Preprocess images before training	Preprocessing Module
FR3	Train a CNN model	Training Engine Module
FR4	Support multiple CNN architectures	Model Architecture Module
FR5	Implement transfer learning	Model Architecture Module (Pretrained Weights Adaptation)
FR6	Handle class imbalance	Training Engine (Class-Weighted Loss Function)
FR7	Evaluate model performance	Evaluation Module
FR8	Visualize training results	Visualization Module
FR9	Export trained model	Model Export Function (torch.save)
FR10	Perform stratified data splitting	Dataset & DataLoader Module