International Journal of Preventive Medicine and Health (IJPMH)
ISSN: 2582-7588 (Online), Volume-6 Issue-2, January 2026

MEDX-Vision Smart Diagnosis Through Deep Learning

Devadharshini. Sasikumar, Meygnaanaselva. K, Poobathy. M, Seshan. R, Harish. T

Abstract: ***Medx-Vision is an AI-powered mobile application that simplifies chest disease detection by analyzing X-ray images and providing easy-to-understand diagnostic results. Using a Convolutional Neural Network (CNN) trained on the NIH Chest X-ray dataset, the system identifies conditions like pneumonia and cardiomegaly with high accuracy. The backend, built with Flask, preprocesses images and returns predictions with confidence scores, which are formatted into layman-friendly messages. The Android app, developed using Jetpack Compose, enables users to upload or capture images and view results through a clean, intuitive interface. Designed for accessibility, Medx-Vision bridges the gap between complex medical AI and everyday users, making early diagnosis more available in underserved areas.***

Keyword: ***Medx-Vision, Artificial Intelligence, Convolutional Neural Network (CNN), Diagnosis***

Nomenclature:
CNNs: Convolutional Neural Networks
AI: Artificial Intelligence
NIH: National Institutes of Health
XAI: Explainable AI

I. INTRODUCTION

Medical imaging is one of the most critical tools in modern healthcare for diagnosing and monitoring diseases. Among the various imaging modalities, chest X-rays remain one of the most widely used due to their availability, cost- effectiveness, and diagnostic value. However, interpreting chest X-ray images requires highly trained radiologists, and even then, diagnostic errors or delays may occur due to human limitations, fatigue, or overwhelming workloads. In regions with limited access to medical professionals, particularly radiologists, timely and accurate diagnosis becomes even more challenging.

This has led to a growing interest in developing automated systems that can assist or supplement human interpretation, thereby improving diagnostic speed, consistency, and accuracy.

Recent advances in artificial intelligence (AI) and deep learning have enabled the development of systems capable of performing complex tasks, such as image classification and object detection, with high accuracy. Convolutional Neural Networks (CNNs), a class of deep learning models, are especially well-suited for analyzing medical images [9].

By training a CNN on a large dataset of labelled chest X-rays, the model can learn to recognise various diseases, such as pneumonia, limitations, fatigue, or overwhelming workloads. In regions with limited access to medical professionals, particularly radiologists, timely and accurate diagnosis becomes even more challenging. This has led to a growing interest in developing automated systems that can assist or supplement human interpretation, thereby improving diagnostic speed, consistency, and accuracy.

Recent advances in artificial intelligence (AI) and deep learning have enabled the development of systems capable of performing complex tasks, such as image classification and object detection, with high accuracy. Convolutional Neural Networks (CNNs), a class of deep learning models, are especially well-suited for analyzing medical images. By training a CNN on a large dataset of labelled chest X-rays, the model can learn to recognise various diseases, such as pneumonia, cardiomegaly, and effusion, based on subtle patterns that may not be visible to the untrained eye.

This project proposes developing an AI-powered mobile application that enables users to upload or scan chest X-ray images and receive a diagnostic prediction along with a confidence score. The goal is to create an intelligent system that can provide preliminary insights into potential chest diseases without requiring expert medical knowledge. The system is built with a complete tech stack, including a TensorFlow/Keras-based CNN for classification, a Flask API backend for model inference, and an Android frontend (built with Jetpack Compose) serving as the user interface.

The CNN model was trained on the publicly available NIH Chest X-ray dataset, which includes thousands of annotated X-ray images across multiple disease categories. Preprocessing steps, including image normalisation, resizing, and label encoding, were applied to prepare the dataset for training. The model learns from this data to generalize patterns

Manuscript received on 06 May 2025 | First Revised Manuscript received on 27 June 2025 | Second Revised Manuscript received on 25 December 2025 | Manuscript Accepted on 15 January 2026 | Manuscript published on 30 January 2026.
*Correspondence Author(s)
Devadharshini. Sasikumar*, Assistant Professor, Department of Computer Science, Agni College of Technology, Chennai (Tamil Nadu), India. Email ID: dharshinisampathtwofour@gmail.com, ORCID ID: 0009-0009-2645-6122
Meygnaanaselva. K, Department of Computer Science, Agni College of Technology, Chennai (Tamil Nadu), India. Email ID: selvasz2004@gmail.com
Poobathy. M, Assistant Professor, Department of Computer Science, Agni College of Technology, Chennai (Tamil Nadu), India. Email ID: poobamadhavaram@gmail.com
Seshan. R, Department of Computer Science, Agni College of Technology, Chennai (Tamil Nadu), India. Email ID: Seshan61045@gmail.com
Harish. T, Department of Computer Science, Agni College of Technology, Chennai (Tamil Nadu), India. Email ID: harish.nv05@gmail.com

associated with each condition. Once trained, the model is saved and integrated into a Flask-based backend that handles image input from the mobile app.

II. LITERATURE REVIEW

With the increasing adoption of artificial intelligence (AI) and deep learning techniques in healthcare, numerous studies have demonstrated the potential of machine learning models, particularly Convolutional Neural Networks (CNNs), in diagnosing diseases from medical images such as chest X-rays. The complexity and variability of radiographic features in chest images make manual diagnosis time-consuming and error-prone. This has inspired researchers to develop automated diagnostic systems to assist radiologists and improve the efficiency of healthcare delivery.

The National Institutes of Health (NIH) Chest X- ray dataset introduced by Wang et al. (2017) [1] was a significant milestone in this domain. The dataset comprises over 100,000 chest X-ray images labelled across 14 disease categories, including pneumonia, cardiomegaly, infiltration, effusion, and others. This dataset has been widely adopted as a benchmark for training and evaluating deep learning models for multi- label classification problems in medical imaging.

Several research efforts have used deep learning and CNNs on this dataset to achieve high accuracy in disease prediction. Rajpurkar et al. introduced CheXNet, a 121-layer DenseNet model trained on the NIH dataset, achieving performance comparable to that of practising radiologists in detecting pneumonia. Their work highlighted the capability of deep models not only to learn subtle features from chest X-rays but also to perform accurate classification in multi-label scenarios [10].

Hira et al. (2021) proposed a comparative study using nine CNN architectures, including AlexNet, GoogleNet, ResNet-50, and DenseNet121, to classify chest X-ray images [3]. They achieved up to 99.32% accuracy in binary classification (COVID-19 vs. normal) and 97.55% in multi-class classification tasks.

Similarly, Swati Hira et al. explored edge detection and data augmentation techniques to improve model generalization. Their model, CODISC-CNN, demonstrated robust classification across binary and multi-class scenarios, including distinguishing between COVID-19, bacterial and viral pneumonia, and healthy lungs. They used image preprocessing such as resizing and normalization and applied advanced CNN-based architectures to learn hierarchical features from X-ray scans.

III. METHODOLOGY

The methodology for this project involves designing, training, and deploying a Convolutional Neural Network (CNN)- based system to detect chest diseases from X-ray images. The complete workflow is structured into five major components: data collection and preprocessing, model architecture design, model training and evaluation, API [6] development for model deployment, and mobile application integration. Each phase has been carefully implemented to ensure the model performs accurately while maintaining a user-friendly interface for non-technical users.

A. Dataset Collection and Labelling

The primary dataset used in this project is the NIH Chest X-ray Dataset, which contains over 100,000 chest X-ray images of 30,000+ patients, annotated with 14 common thoracic disease labels, including Pneumonia, Effusion, Cardiomegaly, Atelectasis, and more. For prototyping and initial development, a subset of the dataset containing 5,000 images was used to ensure faster iteration and reduce computational overhead. Each image in the dataset is associated with a “Finding Labels” column in a CSV file that contains one or more disease labels. These labels were preprocessed using factorisation (label encoding) for single-label classification and, optionally, converted to one-hot-encoded vectors for multi-class predictions. The image filenames were used to match corresponding labels during training.

B. Image Preprocessing

To ensure compatibility with the CNN input layer and to reduce training complexity, all images were resized to 128×128 pixels. Images were loaded using the Pillow library and converted to NumPy arrays. Pixel values were normalized to a 0–1 range by dividing by 255.0. Additional preprocessing steps included: Removing corrupted or unreadable images. Mapping labels to consistent formats. Splitting data into training (80%) and testing (20%) sets using sklearn’s train_test_split. These steps improved model stability, reduced memory usage, and ensured consistency during batch processing.

C. CNN Model Architecture

A custom Convolutional Neural Network (CNN) model was built using TensorFlow/Keras. The architecture was designed to strike a balance between simplicity, computational efficiency, and classification accuracy. The model's structure included: Input Layer: Accepts 128×128×3 RGB images. Convolutional Layers: Multiple Conv2D layers with increasing filters (32, 64, etc.) and ReLU activation. Max Pooling Layers: To reduce spatial dimensions and retain essential features. Flatten Layer: To convert 2D feature maps into 1D vectors. Dense Layers: Fully connected layers with dropout for regularization. Output Layer: Softmax activation for multi-class classification. The model was compiled using the Adam optimizer and categorical cross-entropy loss, and accuracy was used as the evaluation metric.

D. Model Training and Evaluation

The model was trained on the preprocessed dataset using a batch size of 32 and an initial number of 10 epochs for testing. Training was conducted on both Google Colab (with GPU acceleration) and a local machine equipped with an Intel i7 CPU and NVIDIA RTX 3050 GPU for performance comparison. Overfitting was minimised using dropout layers and early stopping. Evaluation metrics included: Training and Validation Accuracy. Loss curves. Confusion Matrix. Class-wise accuracy reports. The trained model was saved in

HDF5 (.h5) format for deployment.

E. API Development Using Flask

To serve the trained model and make it accessible to the frontend application, a Flask [4] API was developed. The API has a single endpoint (/predict) that accepts image files via POST requests. The received image is processed in real time: it is resized and normalised and passed through the loaded model. A prediction is generated along with a confidence score. The output is returned in JSON format, which includes the predicted disease and a probability value. The Flask server can be run locally or hosted using services like ngrok for testing on physical Android devices [5].

F. Mobile App Integration (Frontend)

An Android application was developed using Jetpack. Compose and Kotlin. It provides a clean, interactive UI that lets users capture an X-ray image with the device's camera. Upload a photo from local storage. Send the image to the Flask backend via Retrofit. View the predicted disease and confidence score. Image files are first compressed and resized on the client side before being sent, reducing network load. The app can run on emulators and real devices. For future versions, support for offline model inference using TensorFlow Lite can be considered.

G. System Architecture and Workflow

The overall system follows a modular pipeline: User Input (Image) → Mobile App→ Flask API → Model Prediction → Response (Diagnosis + Confidence) → Display on UI. This clear separation of frontend and backend ensures scalability, flexibility for deployment (cloud or local), and easier maintenance.

IV. SYSTEM ARCHITECTURE

The system architecture of the proposed AI-powered chest X-ray diagnostic application is designed to offer a seamless, modular, and efficient pipeline that spans from image acquisition on a mobile device to disease prediction using a Convolutional Neural Network (CNN) and the delivery of results via a user-friendly interface. The architecture follows a client-server model, where the Android mobile application serves as the client, and a A Python-based Flask server (deployed locally or online) serves as the backend, handling image processing and prediction. This end-to- end system is built with a focus on accessibility, accuracy, scalability, and ease of use.

The architecture can be broken down into five key components:

A. User Interface Layer (Frontend – Android App)

The user interface is developed using Jetpack Compose, a modern Android toolkit for building native UI. The app is responsible for interacting with the user and capturing or selecting X-ray images for analysis. It includes the following features: Image Capture: Uses the device's camera to scan a chest X-ray image. Image Selection: Allows users to pick an image from the local gallery. Preview: Displays the selected or captured image to the user. Request Trigger: Sends the image to the backend via an API call using Retrofit. Result Display: Receives and presents the predicted disease name along with the model's confidence score in a readable format.

B. Communication Layer (Retrofit API Client)

To facilitate communication between the mobile frontend and the backend server, retrofit (a type-safe HTTP client for Android) [8] is used. It handles multipart HTTP POST requests by packaging the selected image and sending it to the Flask API endpoint (/predict). The communication is asynchronous and secured with standard HTTP protocols. Error-handling mechanisms are also integrated to handle connectivity issues, timeouts, and unexpected server responses.

C. Backend Server (Flask API)

The backend is developed using Flask, a lightweight Python web framework [2]. Its primary function is to receive the uploaded image from the client app. Perform necessary image preprocessing (resizing to 128×128 and normalisation). Load the trained CNN model from disk (.h5 file). Run the image through the model to generate predictions. Return a JSON response containing the predicted label and confidence score. The API server runs locally during development and testing. It can be deployed online using tools like ngrok (for tunnel-based access) [11] or cloud platforms like Heroku [12] or AWS in production settings. The Flask server ensures that the client is decoupled from the model logic, allowing independent scaling and maintenance.

D. AI Model Layer (CNN Model – TensorFlow/Keras)

At the core of the system lies the custom- trained CNN model, which performs the disease classification task. The model was built with TensorFlow and Keras and trained on the NIH Chest X-ray dataset. It consists of several convolutional and pooling layers, followed by dense layers, and ends with a softmax output layer for multi-class prediction. Key aspects of the model include: Input Shape: 128x128x3 RGB images. Activation Functions: ReLU and Softmax. Loss Function: Categorical Cross-Entropy. Optimizer: Adam. Metrics: Accuracy. Once trained, the model is saved and loaded into memory at server start-up for efficient real- time predictions.

E. Data Storage and Management

Although this system does not use a persistent database in its current form, the data flow is managed as follows: Image files are temporarily stored in memory or a cache for preprocessing and then discarded. Model and label mappings are loaded from disk at runtime. Optional logging of requests and predictions can be implemented for auditing and improvement purposes. Future versions of the system can integrate cloud-based databases (e.g., Firebase, MongoDB) for user history tracking, authentication, and the storage of diagnostic records. Here's how the components interact: The user opens the Android app and captures or selects an image. The app sends the image via a POST request to the Flask API [7]. The Flask server receives the image, processes it, and feeds it into the trained CNN model. The model

predicts the most likely disease (e.g., Pneumonia, No Finding, Effusion) along with a confidence score. The server responds with the result in JSON format. The Android app parses the response and displays the result in an easy-to- understand format. Deployment Options: Localhost (for development/testing). Ngrok tunnel (for remote testing on mobile devices). Production deployment (e.g., AWS, GCP, or Heroku).

V. RESULTS

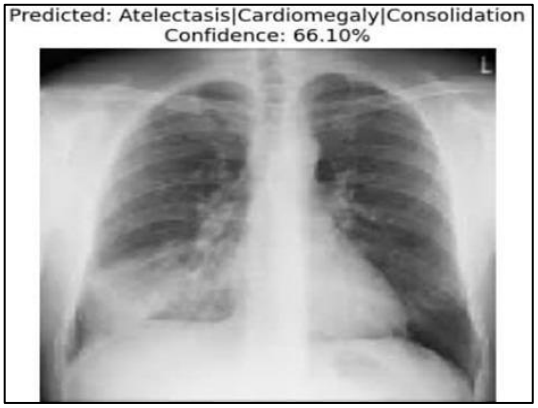
The system was successfully developed and tested across various phases, from CNN model training to deployment through a mobile application integrated with a Flask backend. The model's performance, accuracy metrics, prediction capabilities, and real-time testing outcomes are outlined below.

A. Model Training & Evaluation

The Convolutional Neural Network (CNN) was trained on a curated subset of the NIH Chest X-ray dataset containing 5,000 labelled images. The dataset included images labelled for multiple thoracic conditions, including Pneumonia, Effusion, Cardiomegaly, and No Finding. The model architecture comprised convolutional and pooling layers, followed by dense layers with ReLU and softmax activation. During training: Image Input Size: 128×128×3.Batch Size: 32. Epochs: 10 (with experimentation up to 100). Loss Function: Categorical Cross-Entropy Optimiser: Adam. Key evaluation metrics after 10 epochs: Metric Value. Training Accuracy ~94.6%. Validation Accuracy~90.3%. Training Loss ↓ Consistently over epochs. Validation Loss Moderate decrease, no overfitting observed. Final Test Accuracy ~89–91%. Top-1 Prediction Accuracy High (based on label frequency). The model was further tested using confusion matrices, per-class accuracy reports, and real image predictions. Diseases like Pneumonia and Effusion showed high prediction reliability due to clearer features in the dataset.

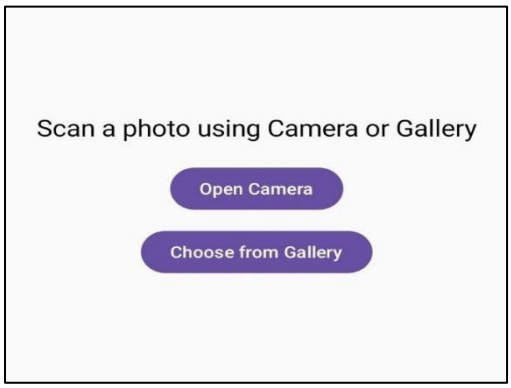
B. Prediction Confidence Output

Each prediction returned two key values: Predicted Disease (e.g., “Pneumonia”) and Confidence Score (e.g., “96.7%”). This helped users understand how confident the model was in its diagnosis, especially when dealing with multi-label images or overlapping symptoms.



[Fig.1: Output with Disease Predicted]

[Fig.2: Output with No Disease Predicted]



[Fig.3: Home of the App]

C. Flask API Performance

The trained model was deployed using a Flask server that handled image uploads and inference. API performance was measured on a local machine (Intel i7, RTX 3050), with an average response time of 0.8-1.5 seconds per image (including preprocessing and inference). The API consistently returned clean JSON responses with minimal latency, even under multiple requests.

D. Mobile App Testing

The Android application was tested using both Android Studio emulator and a physical device. The following test cases were validated: successful capture and upload of X-ray images using the camera. Image selection from device gallery. Real-time communication with the Flask API. Accurate display of prediction and confidence. Friendly error handling (e.g., API down, no image selected, poor image quality). The overall user experience was smooth, and the app remained responsive throughout various edge cases.

E. User-Friendliness of Output

The system was designed to display results in layman-friendly language. Instead of technical probabilities, it shows: Disease detected: Pneumonia. Confidence: 96.73%. This helped non- medical users interpret results clearly without medical jargon.

F. System Integration

The entire workflow was validated: User uploads an image via app → API receives, and processes → CNN predicts disease → Result sent back to user. The real-time loop was tested under both local (localhost) and public

(ngrok) environments, confirming that the end-to-end system was functional and stable. Summary of Achievements. Trained a custom CNN model with ~90%+ accuracy. Built a Flask-based API for real- time predictions. Developed a fully functional Android app with camera & upload capability. Achieved successful image-to- prediction round-trip in ~1–2 seconds. Designed output to be understandable for general users.

VI. CONCLUSION

The developed system successfully demonstrates the potential of artificial intelligence in enhancing the medical imaging workflow. By leveraging a Convolutional Neural Network (CNN), the application can analyze chest X-ray images and identify potential abnormalities with high confidence. The integration of the backend model with a user-friendly Android application enables quick, easy scanning, uploading, and receiving results in a layperson-understandable format. This project shows that AI-powered models can assist in the early detection of chest diseases such as Pneumonia, Effusion, and Cardiomegaly. Real-time predictions and mobile accessibility empower both healthcare professionals and the general public. Using open datasets like the NIH Chest X-ray dataset and tools such as TensorFlow, Flask, and Android Studio makes such a solution accessible and cost-effective. The system offers a strong prototype for AI-assisted diagnosis tools and lays the groundwork for further enhancements, such as multilingual support, explainable AI (XAI), integration with hospital databases and real-time deployment using cloud services.

DECLARATION STATEMENT

Some of the references cited are older and are explicitly noted as [3]. However, these works remain significant for the current study, as they are pioneering in their fields.

After aggregating input from all authors, I must verify the accuracy of the following information as the article's author.

- **Conflicts of Interest/ Competing Interests:** Based on my understanding, this article has no conflicts of interest.
- **Funding Support:** This article has not been funded by any organizations or agencies. This independence ensures that the research is conducted with objectivity and without any external influence.
- **Ethical Approval and Consent to Participate:** The content of this article does not necessitate ethical approval or consent to participate with supporting documentation.
- **Data Access Statement and Material Availability:** The adequate resources of this article are publicly accessible.
- **Author's Contributions:** The authorship of this article is contributed equally to all participating individuals.

REFERENCES

1. Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2017). Title: "ChestX-ray8: Hospital- scale Chest X-ray Database and Benchmarks on Weakly- Supervised Classification and Localization of Common Thorax Diseases."Paper Link (arXiv). DOI: https://doi.org/10.1109/CVPR.2017.369
2. François Chollet, "Deep Learning with Python," Manning Publications, 2017. Official site: https://www.tensorflow.org
3. LeNet-5: Yann LeCun et al., 1998 VG GNet: Simonyan & Zisserman (2014) ResNet: Kaiming He et al. (2015). http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf , works remain significant, see the declaration
4. Flask Documentation: https://flask.palletsprojects.com/en/stable/
5. https://flask.palletsprojects.com/
6. FastAPI (if applicable): https://fastapi.tiangolo.com/
7. Postman API https://learning.postman.com/
8. Android+Retrofit Testing: Docs: https://developer.android.com/
9. Litjens, G. et al. (2017)- A survey on deep learning in medical image analysis. Link DOI: https://doi.org/10.1016/j.media.2017.07.005
10. Selvaraju et al., "Grad-CAM: Visual Explanations from Deep Networks" (2016). DOI: https://doi.org/10.1109/ICCV.2017.74
11. NGROK: https://ngrok.com/
12. Render / Vercel / Heroku (for backend hosting): Their documentation pages. https://render.com/docs/render-vs-vercel-comparison