CENG 407

Innovative System Design and Development I

2025-2026 Fall

MULTI-LABEL CHEST X-RAY DISEASE CLASSIFICATION AND EXPLAINABILITY WITH DEEP LEARNING

METHODOLOGY

YAREN AKDOĞAN 202111046

SEZİN KÖROĞLU 202211046

İBRAHİM TAŞPINAR 202111072

CEVDET ALP ÖZGÜN 202211406

İPEK BAŞ 202211013

SALİH BARKIN AKKAYA 202211004

Abstract

This methodology presents an end-to-end system for multi-label thoracic disease classification from chest X-ray images, coupled with explainability and deployable software integration. The study uses the NIH ChestX-ray14 dataset and addresses key data challenges, including multi-label co-occurrence and severe class imbalance via a weighted learning strategy. To improve standardization and model robustness, images are filtered to Posterior–Anterior (PA) views and processed under multiple preprocessing pipelines (contrast/intensity enhancement, noise reduction, bone suppression, and frequency-domain filtering). All images are then standardized to a 224×224, three-channel format to support transfer learning with common pretrained backbones. The core model employs a DenseNet-121 backbone augmented with Convolutional Block Attention Modules (CBAM) inserted after each dense block to strengthen sensitivity to subtle radiographic patterns while maintaining computational practicality. For interpretability, Grad-CAM is integrated to generate class-specific heatmaps highlighting image regions that drive predictions. The system is operationalized through a RESTful backend that orchestrates preprocessing, inference, and explainability, and persists traceable outputs (predictions, metadata, model versions, and explanation artifacts) in a MongoDB-based schema with external file storage for large media. A Flutter frontend consumes the API to deliver user-friendly upload, results visualization, and history retrieval.

1. Dataset & Preprocessing Methodology

1.1. NIH ChestX-ray14

Selection and the analysis of the dataset for this project, we selected the NIH ChestX-ray14 dataset, which is a standard for thoracic disease classification. We chose this dataset because it provides a sufficient amount of collection (112,120 chest X-ray images) which are labeled for 14 different conditions. During our analysis, we observed that the dataset has a multi-label structure. That means a patient could have multiple diseases at the same time.

We also identified a significant class imbalance issue. While common diseases like Infiltration had over 20,000 examples. Rare diseases like Hernia had very few samples. To prevent the model from ignoring these rare classes, we decided to create a weighted learning strategy for our training process. [1]

1.2. Dataset Filtering (PA view only)

Before starting the training process, we decided to filter the dataset based on the position of X-Ray views. The original dataset contains both Anterior-Posterior (AP) and Posterior-Anterior (PA) views. We chose to limit our dataset to PA views only, as they usually offer better standardization for deep learning models. We implemented a script that scanned the metadata and selected only the rows where the position of data was labeled as "PA". Based on this filtered list, we recreated the file directory and updated the training/testing split to ensure that no AP images were included in our final dataset.

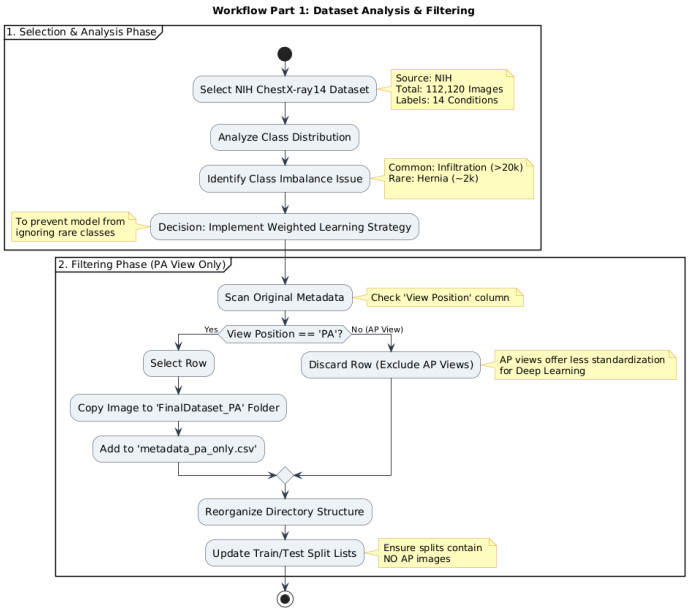

Figure-1: Dataset Analysis and Filtering Workflow Diagram

1.3. Preprocessing Pipelines

Since unprocessed X-ray images generally contain noise or varying lighting conditions, we designed four different preprocessing scenarios to improve the model's performance.

- **Intensity Enhancement:** In the first scenario, we focused on improving contrast. We applied Gamma Correction and Contrast Limited Adaptive Histogram Equalization (CLAHE) to make the details of lungs more visible. We also added an unsharp mask to make clear the edges [3],[5].
- **Noise Reduction:** In the second scenario, we decided to reduce noise. We used a median filter to make the image smoother while preserving edges. Also, by CLAHE and a sharpening kernel, we highlighted the desired structures [3],[4].
- **Bone Suppression:** We experimented with a bone suppression technique to prevent the ribs from hiding lung abnormalities. We achieved this by subtracting a blurred version of the image from the original. That caused reducing the brightness of the bone structures and made the soft tissue clearer [3].
- **Frequency Domain Filtering:** Finally, we tested a method combining CLAHE with a Butterworth band-pass filter. This allowed us to emphasize specific structural details in the frequency domain while removing high-frequency noise [2],[3].

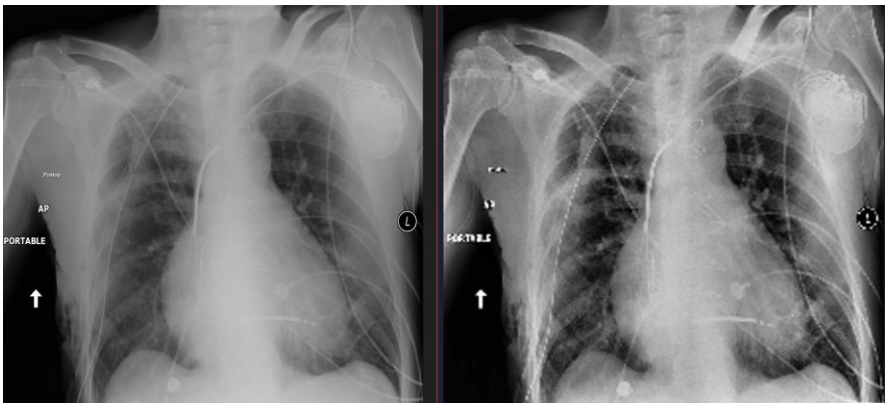

Figure-2: Intensity Enhancement

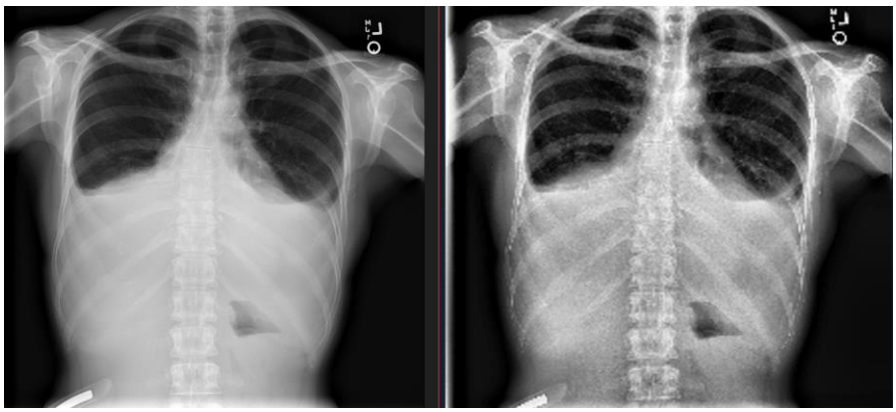

Figure-3: Noise Reduction

Figure-4: Bone Suppression

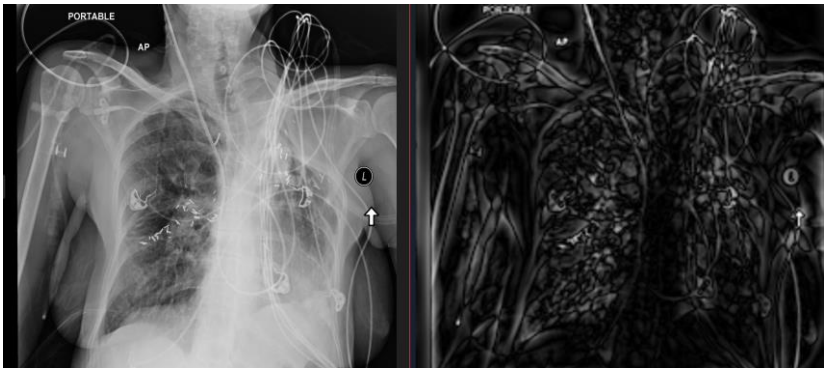

Figure-5: Frequency Domain Filtering

1.4. Standardization for the Backend

Independent of the chosen preprocessing strategy, we decided to put our processed dataset in a standard format. This method ensured the compatibility with our backend system. For every image in our dataset, we resized the resolution to 224x224 pixels. Even though the X-rays images in our dataset are originally in grayscale format we converted them into a 3-channel RGB format by repeating the grayscale channel three times. This decision was made to give us the opportunity to use pre-trained deep learning architectures (DenseNet, ResNet, etc.) that expect 3-channel input by default.

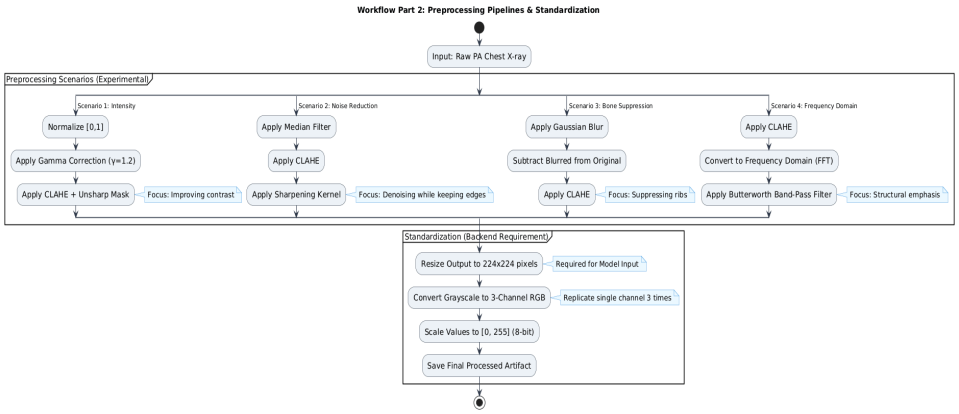

Figure-6: Preprocessing Pipelines and Workflow Diagram

2. Model Architecture

2.1. DenseNet-121 Backbone

We have chosen DenseNet as the backbone of our model due to its proven success in medical image classification tasks. The dense connectivity technique used in DenseNet allows the outputs of previous layers to be utilized in all future layers within a dense block in the model, which mitigates the possible vanishing gradient issues we may encounter [6]. The pretrained ImageNet weights of DenseNet also allow for faster convergence and stable training. The segmented structure of DenseNet allows for the insertion of different modules between its dense blocks, which is perfect for our project. The dense blocks are made up of bottleneck layers with dense connectivity and make up a large part of the model. Bottleneck layers start with batch normalization and ReLU followed by a 1x1 convolutional layer, after which another batch normalization and ReLU, and end with one final 3x3 convolutional layer [6].

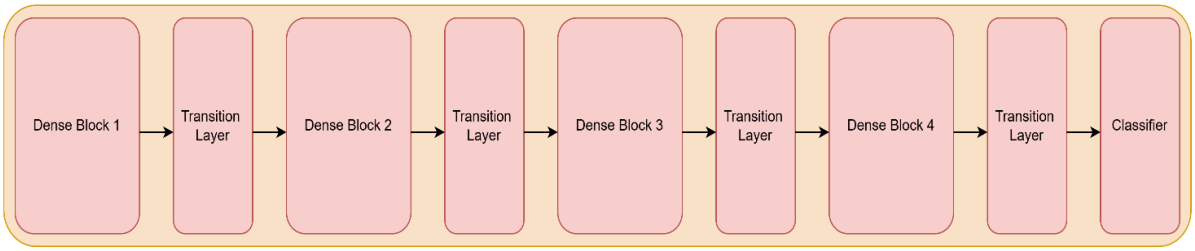

Figure-7: High Level Architecture of DenseNet-121

2.2. Convolutional Block Attention Module (CBAM)

CBAM is an attention module comprised of two smaller components called channel attention module and spatial attention module respectively [7]. Neither component affects the input or output sizes of the layers before and after them thus allowing for easy insertion into our model.

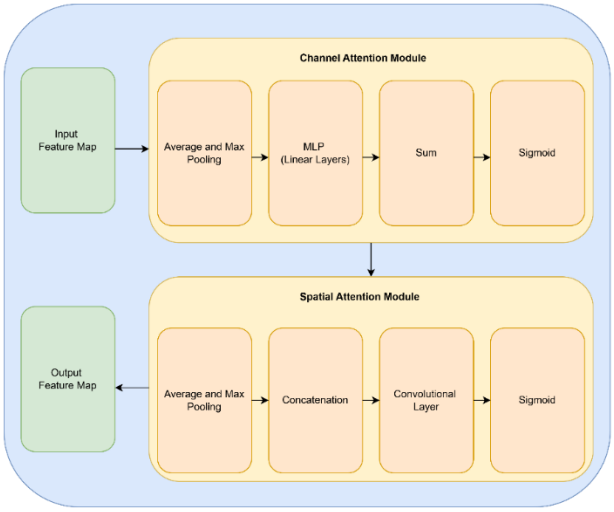

Figure-8: High Level Architecture of CBAM

2.2.1. Channel Attention Module

The channel attention module decides what features are important for a given task. The module applies average and max pooling to model the current features and passes them to an MLP. The sigmoid of the sum of the resulting features is multiplied with the current model features to update the feature map of the model [7].

2.2.2. Spatial Attention Module

The spatial attention module focuses on where important regions are on the feature maps. Similarly to the channel attention module, it applies average and max pooling across channels to generate maps that highlight where important features are. These feature maps are concatenated to make a feature map with two channels which is passed through a convolutional layer without changing its size while reducing the channel count back down to one. The sigmoid of the resulting map is multiplied with the current feature map to highlight important spatial regions [7].

2.2.3. Overall Proposed Architecture

Our model adds a CBAM module after each dense block without changing the feature map size to avoid compatibility issues with subsequent dense blocks. By adding this module after each dense block, we aim to create a model that is more capable of detecting the small differences present in X-ray images without causing an excessive processing load on devices. Due to the precise nature of diagnosing thoracic diseases from X-ray images, we aim to optimize our model's confidence and interpretability. We are planning on using F1 and F1 macro as our metrics during development to evaluate the performance of our model.

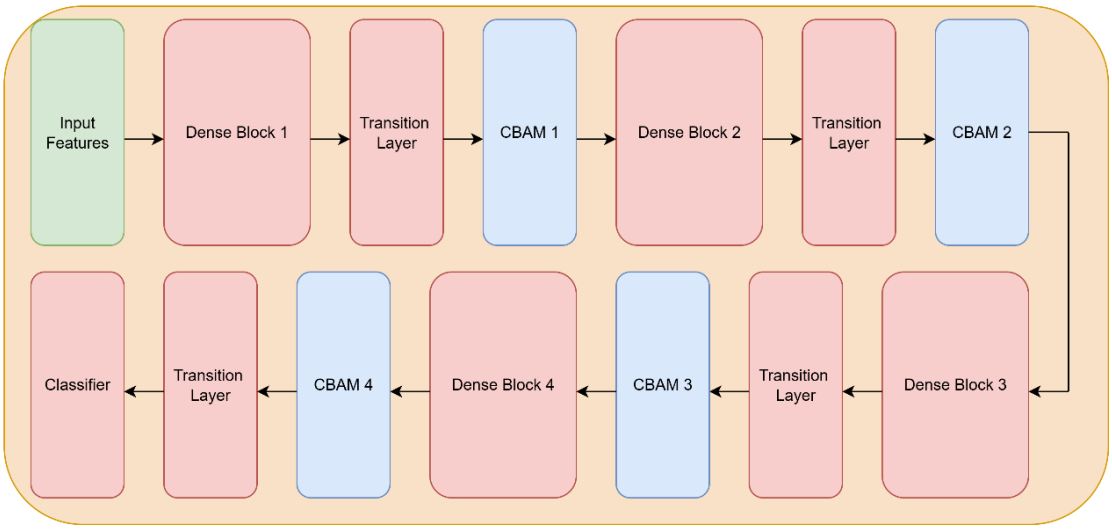

Figure-9: High Level Architecture of Our Model

2.3. Model Explainability Using Grad-CAM

Grad-CAM will be integrated into the model outputs to improve model interpretability. Grad-CAM will be used to visualize the areas that are relevant to the predictions our model makes by generating individual heatmaps over the original X-ray image for each possible disease class [8]. Through this technique, we aim to increase the trust of users by giving them the ability to see why the model came to a decision.

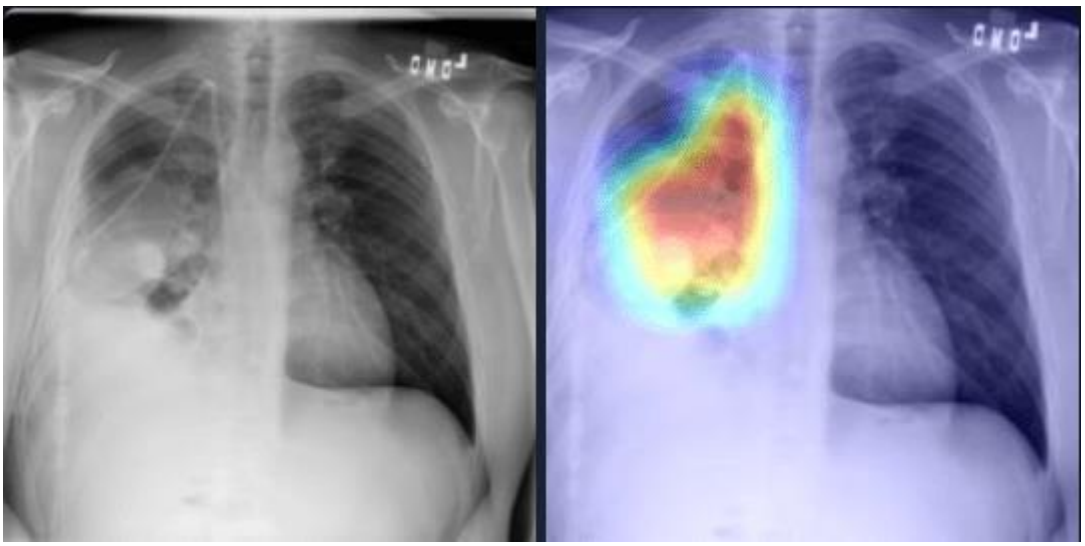

Figure-10: Original X-ray Image and The Grad-CAM Heatmap of The Same Image

3. Backend & Database Methodology with MongoDB

3.1. Backend Role in the System

In this project, the backend will be designed to manage the machine learning pipeline in a controlled manner and to provide a clean API for the system. After reviewing similar medical imaging projects, the system architecture will be designed to separate the model code from the application logic. This design will prevent the frontend from directly interacting with complex training scripts or local file structures. Instead, the backend will handle the entire workflow, including request handling, data preparation, model inference, and the delivery of consistent JSON responses.

This approach supports a workflow that includes steps such as preprocessing and explainability and it ensures that every output remains traceable. Managing these steps manually in medical imaging can lead to mistakes or inconsistent results [9], [11]. By positioning the backend as a central coordinator, each request follows the same execution logic, which increases overall reliability.

3.2. API Design Decisions

The backend will be implemented as a RESTful API service. During planning, the API will focus on the endpoints required by the user—such as uploading an image or retrieving a prediction—rather than exposing internal training details. This design will keep the communication between the frontend and backend simple and clear.

We also plan to standardize model outputs before sending them to the UI. Since the task is multi-label, the model will produce a probability value for each label [9], [11]. These raw probability scores will remain the primary stored result. A thresholding strategy will be used to convert probabilities into final predicted labels when required, and the selected threshold strategy will be stored so there is no confusion during later review.

3.3. Why MongoDB?

The database layer will use MongoDB. This choice aligns with the structure of the generated outputs: for each inference run, the system produces a “package” that includes metadata, predictions, and links to images. This naturally fits MongoDB’s document-based storage model.

Medical image ML pipelines change frequently during development—preprocessing steps may change, and model versions may be updated [10]. MongoDB’s flexible schema supports adding new fields or adjusting the stored structure without repeatedly redesigning strict relational tables, which supports faster iteration.

3.4. Data Stored in MongoDB

The system does not lock into an overly rigid schema early, but it defines the core record categories required for organization and traceability:

- **Image Records:** Unique IDs, basic metadata, and the path to where the file is stored.
- **Preprocessing Records:** Details on which filters or methods are used for repeatability [12].
- **Model Records:** Version information and checkpoint locations to identify which model produces each prediction.
- **Prediction Records:** Multi-label probability scores and the final labels based on the threshold strategy [9], [11].
- **Explanation Records:** Grad-CAM settings (e.g., target layer) and links to generated heatmap files [8].

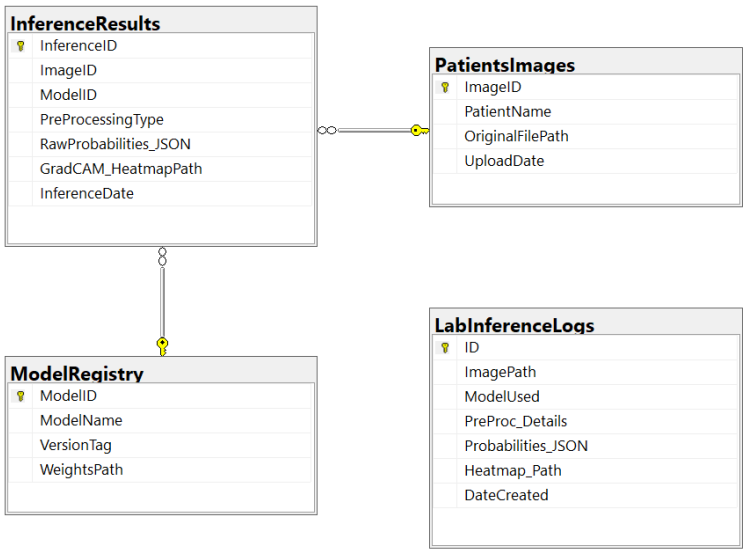

Figure-11: This table shows the logical structure of our database and how we link images to their specific model predictions and preprocessing steps.

3.5. File Storage Strategy

We plan to keep large files such as X-ray images and Grad-CAM overlays as regular files on the storage layer rather than storing them inside the database. In MongoDB, we will store only the file path (URI). This strategy will keep the database responsive and will allow the frontend to fetch images directly using the provided links.

3.6. Inference Workflow

When the backend receives an inference request, it will follow a consistent pipeline:

1. **Input preparation:** The backend will load the selected image artifact (original or preprocessed), apply the required resizing and channel formatting expected by the model, and normalize the input according to the model configuration.
2. **Model selection:** Inference will run using a specific model version (checkpoint). The backend will load the weights and maintain label ordering consistent with training [9], [11].
3. **Forward pass and outputs:** The model will produce a probability value per label (multi-label setup). These probabilities will be stored as the main prediction output.
4. **Decision logic (thresholding):** When needed, a threshold strategy will convert probabilities into a predicted label set, and the applied threshold strategy will be recorded to avoid later ambiguity.

A key decision is that prediction results will not only be displayed once; they will be stored as a structured record linked to the image reference and the model version that produces them. This will enable later retrieval (e.g., showing the latest prediction for a model) without rerunning inference.

3.7. Explainability Workflow (Grad-CAM)

We plan to integrate Grad-CAM directly into the backend so that explainability functions as a core capability rather than an external script [12]. This will help users understand which parts of the X-ray contribute to the model's decision.

We plan to treat Grad-CAM as a separate action because it typically requires additional computation compared to a standard prediction. The backend will generate the heatmap, save it as an image, and store the corresponding settings in the database. This will allow later inspection of how a specific model version produces a specific explanation for a given case.

```json
{
  "inference_id": "INF_7721",
  "patient_data": {
    "image_id": 1,
    "path": "/storage/xray_001.png"
  },
  "model_info": {
    "name": "DenseNet121",
    "version": "v1.2"
  },
  "results": {
    "probabilities": {
      "Effusion": 0.82,
      "Atelectasis": 0.15
    },
    "threshold": 0.5,
    "final_labels": ["Effusion"]
  },
  "explainability": {
    "method": "Grad-CAM",
    "heatmap_url": "/outputs/cam_001.png"
  },
  "timestamp": "2025-12-18T21:20:00Z"
}
```


Figure -12: An example of a JSON document from our database. It shows how we store the model's probability scores and the file paths for the Grad-CAM outputs in one place.

4. Backend

4.1. Frontend Interaction and API Integration

This section of our report explains the technical details of how our deep learning model will reach the end user and how data will be transferred.

4.1.1. API Design and Endpoints

We plan to use a modular RESTful API architecture on the server side to handle Flutter/Web (client) requests. Of course, endpoints such as POST/analyze, GET/results/{id}, and GET/history will be defined for basic functions.

The POST/analyze endpoint will be used as the gateway to initiate the analysis process when the user uploads the x-ray image. The GET/results/{id} endpoint will be where the results of the analysis and the outputs obtained through Grad-CAM are queried. GET/history is planned to be the structure that lists the user's past queries[13].

4.1.2. Data Format and Communication Protocol

Due to platform independence and lightness, it has been decided that the format for all data exchanges on both the client and server sides will be JSON (JavaScript Object Notation). While this methodology and visual data are sent in the body of HTTP requests,

the model's results, disease probabilities, and metadata will be communicated to the interface as JSON objects.

4.1.3. Integration Flow and Grad-CAM Visualization

The file referencing method will be the most useful bridge for integrating the Grad-CAM heat maps generated by our model into the interface. This method will work as follows: the heatmap to be generated will be saved to a directory on the server side, and the URL link for this image will be shared in the JSON response returned to the interface. As a result, instead of large image data being pulled directly into the database, it will be loaded as a media object via the server.

Analysis results and Grad-CAM outputs transmitted via the API are permanently stored on the backend for later access and traceability. This enables the system to support both real-time analysis and the management of past results.

4.1.4. Error Handling and Status Codes

System resilience is important in this project, as it is in every project. For this reason, we are considering developing an integrated structure with HTTP Status Codes for potential risks.

If an invalid file is uploaded (anything other than an x-ray), the user will be notified with a 400 Bad Request error and an explanatory text. Any issues that may occur on the server or model side will be prevented with a 500 Internal Server Error. We will enhance the resilience of our system with numerous alerts and informative notifications for other potential issues of this nature.

5. Frontend

5.1. Frontend Interaction and User Interface Design

This section of our report explains how the user will interact with our system through the Flutter-based frontend and how the results produced by the deep learning model will be presented to the user in a clear and understandable way.

The Flutter application will act as the client side of the system and will be responsible only for user interaction and visualization. All heavy operations such as image preprocessing, model inference, and Grad-CAM generation will be handled on the backend side [13],[14].

5.2. User Flow and Application Structure

We plan to design the Flutter application with a simple and intuitive user flow. First, the user will be able to select a chest X-ray image from their device. After the image is selected, it will be displayed on the screen so that the user can confirm the correct image before analysis.

Once the user starts the analysis, the selected image will be sent to the backend through the defined API endpoint. During this process, a loading indicator will be shown to inform the user that the analysis is in progress. After the backend completes the inference, the results will be sent back to the Flutter application and displayed to the user [15].

5.3. Result Visualization and Grad-CAM Integration

The results page of the application will display the predicted diseases together with their probability values. These values will help the user understand how confident the model is about each prediction.

For model explainability, Grad-CAM heatmaps generated on the backend will be visualized in the Flutter interface.

5.4. API Communication and Data Handling

Image uploads will be handled using multipart requests, while prediction results and metadata will be transferred in JSON format. [16],[17].

On the client side, the received JSON responses will be parsed and mapped to internal data structures. This structure will allow the application to safely display disease probabilities, Grad-CAM image references, and possible error messages returned by the backend. [18].

5.5. Error Handling and User Feedback

User experience and system robustness are important aspects of the frontend design. For this reason, the Flutter application will handle different error scenarios and provide informative feedback to the user.

If no image is selected or an unsupported file is uploaded, the user will be warned before the request is sent. In case of network problems or server-side errors, meaningful messages will be displayed and the user will be allowed to retry the operation [17].

References

[1] National Institutes of Health, “NIH Chest X-rays Dataset,” Kaggle, [Online].

Available: https://www.kaggle.com/datasets/nih-chest-xrays/data.

Accessed: 19 Dec. 2025.

[2] A. Giełczyk, A. Marciniak, M. Tarczewska & Z. Lutowski, “Pre-processing methods in chest X-ray image classification,” PLOS ONE, vol. 17, no. 4, e0265949, Apr. 2022.

Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0265949

Accessed: 19 Dec. 2025.

[3] IEEE Xplore, “Article (document no. 10955260),” [Online].

Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10955260.

Accessed: 19 Dec. 2025.

[4] “Median Filter – an overview,” ScienceDirect Topics, Elsevier, [Online].

Available: https://www.sciencedirect.com/topics/engineering/median-filter.

Accessed: 19 Dec. 2025.

[5] N. F. Sahib and Z. A. Hashim, “Contrast Image Enhancement by Gamma Correction,” Computer Engineering and Intelligent Systems, vol. 9, no. 7, 2018.

Available: https://scispace.com/pdf/contrast-image-enhancement-by-gamma-correction-3upr34p2k1.pdf.

Accessed: 19 Dec. 2025.

[6] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, [Online].
Available: https://arxiv.org/abs/1608.06993
Accessed: 20 Dec. 2025.

[7] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “CBAM: Convolutional Block Attention Module,” *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, [Online].
Available: https://arxiv.org/abs/1807.06521
Accessed: 20 Dec. 2025.

[8] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization,” *IEEE*

International Conference on Computer Vision (ICCV), 2017, [Online].
Available: https://arxiv.org/abs/1610.02391
Accessed: 19 Dec. 2025.

[9] X. Wang et al., "ChestX-ray8: Hospital-scale chest X-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," *CVPR*, 2017.
[10] I. E. Ihongbe et al., "Evaluating Explainable Artificial Intelligence (XAI) techniques in chest radiology imaging through a human-centered lens," *PLOS ONE*, 2024.
[11] J. Irvin et al., "CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison," *AAAI*, 2019.

[12] J. Han et al., "BO-CLAHE enhancing neonatal chest X-ray image quality for improved lesion classification," *Scientific Reports*, 2025.

[13] Flutter Team. Flutter Documentation.

https://docs.flutter.dev

[14] Google Developers. Flutter Architectural Overview.

https://docs.flutter.dev/resources/architectural-overview

[15] ISO/IEC 25010:2011. Systems and software Quality Requirements and Evaluation (SQuaRE).

[16] ECMA International. (2017). The JSON Data Interchange Syntax (ECMA-404).

[17] Mozilla Developer Network (MDN). HTTP Overview and Status Codes.