DiagnosAI

PROJECT REPORT

Submitted by Team: RadAI Innovators

Team Members: Pranjal Sharma
Bhavya Verma
Ojasvi Puri
Janhvi Malhotra
Varah Siddhi

Index

OVERVIEW

This internship at Hale HIT Labs provided me with practical, hands-on experience in the development and deployment of AI-powered diagnostic systems for medical imaging. My project focused on the design, training, and integration of a machine learning model for X-ray image classification, combined with a Unity-based visualization interface for interactive result display.

I worked on the complete development pipeline, including:

- Training a Convolutional Neural Network (CNN) model to detect multiple medical conditions from chest X-ray images.
- Converting the trained model to TensorFlow Lite format for efficient inference.
- Implementing a Flask-based API server to deliver model predictions to clients.
- Integrating the AI model with a Unity application for browsing X-ray images, displaying ground truth labels, and viewing real-time AI predictions.

This project gave me insight into end-to-end AI integration, the interaction between backend and frontend systems, and the practical deployment challenges of machine learning in interactive environments.

ACKNOWLEDGEMENT

We, the project team, would like to express our heartfelt gratitude to **Samsung PRISM** for organizing the **GenAI Hackathon 2025** and giving us the opportunity to showcase our work. This platform has allowed us to explore, innovate, and apply advanced AI and AR technologies to solve real-world healthcare challenges.

We sincerely thank our mentors and guides for their valuable feedback, guidance, and constant encouragement throughout the development of this project. Their insights have helped us refine our approach and strengthen our solution.

We also acknowledge the support of our peers, families, and colleagues who motivated us and stood by us during this journey. Their encouragement played a vital role in keeping us focused and driven as a team.

Finally, we appreciate the collaborative spirit within our team. Each member’s contribution, effort, and dedication made it possible to transform our idea into a working solution that we are proud to present at this hackathon.

Introduction

The project focuses on developing an **AI-powered X-ray diagnosis system** integrated into a Unity-based application.
It leverages a machine learning model trained to classify chest X-ray images and identify potential diseases.
The AI model, developed in Python using TensorFlow, was converted to a lightweight TensorFlow Lite format and deployed via a Flask API.
Unity serves as the front-end interface, displaying X-ray images, retrieving AI-generated predictions, and presenting results in an interactive, user-friendly environment.

This system bridges the gap between **advanced AI diagnostic capabilities** and **real-time, accessible visualization** for end-users, making it a potential aid for doctors, technicians, and healthcare workers

Problem Statement

Medical imaging, particularly chest X-rays, plays a vital role in diagnosing diseases such as pneumonia, tuberculosis, and lung abnormalities.
However, two major challenges are often observed:

1. **Shortage of Skilled Radiologists** – Many rural and underdeveloped regions lack trained professionals to interpret medical images promptly.
2. **Time-Consuming Analysis** – In busy hospitals, radiologists often face a backlog of images, delaying diagnosis and treatment.

As a result, patients may experience delayed medical attention, and in critical conditions, this delay can be life-threatening.

Proposed Solution

The project proposes an automated, AI-assisted X-ray diagnosis tool that:

- Accepts chest X-ray images as input.
- Uses a trained machine learning model to predict possible medical conditions.
- Presents results directly inside a Unity-based visual interface for ease of understanding.

The backend AI model is hosted on a Flask server, while Unity communicates with it via HTTP requests, ensuring a smooth and interactive experience for users.

Machine Learning Model Development

The project's AI component was built to automatically classify chest X-ray images into different diagnostic categories. This involved four main stages — **data preparation, model training, optimization, and deployment**.

1. Data Preparation

- The dataset consisted of chest X-ray images, each labeled with a diagnosis in labels.csv.
- Image preprocessing included:
  - **Normalizing** pixel values to a 0–1 range for stable training.
  - Splitting the dataset into **training** and **validation** sets to measure performance.
- Labels were converted from text form into numerical values using a label_map.txt file.

2. Model Training (train_model.py)

- The model was built using **TensorFlow and Keras**.
- A **Convolutional Neural Network (CNN)** architecture was used because CNNs are highly effective for image classification tasks.
- The model had the following key layers:
  - **Convolutional layers** to detect edges, shapes, and texture patterns in X-ray images.
  - **Pooling layers** to reduce data size while preserving important features.
  - **Fully connected layers** to combine features and make the final classification..

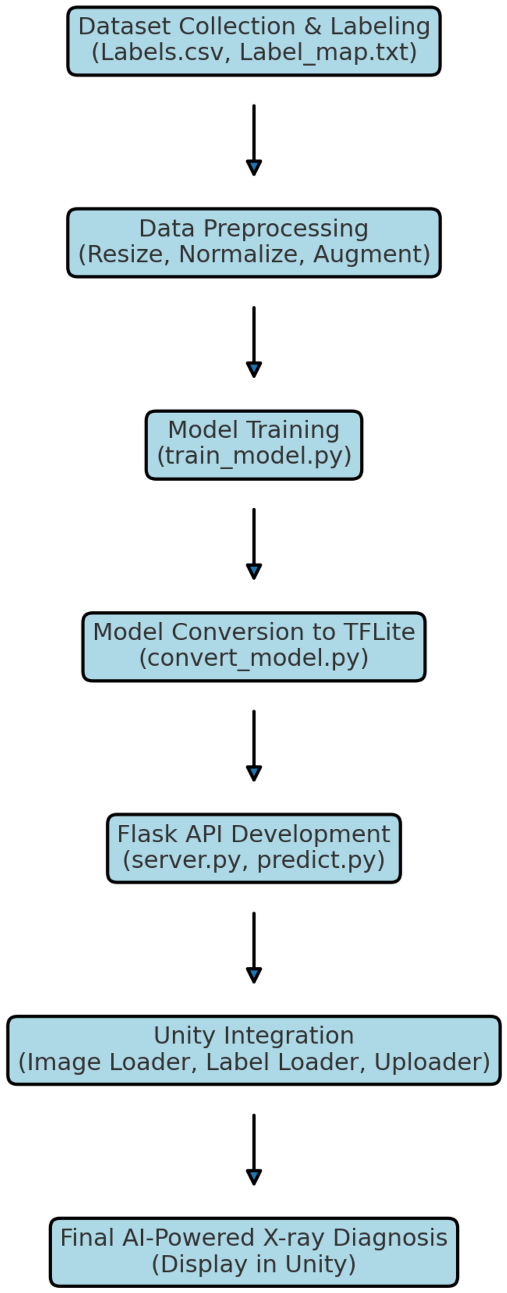
3. Model Optimization & Conversion (convert_model.py)

- Once trained, the model was saved as xray_model.h5.
- To make it run efficiently inside the Unity app, it was converted to **TensorFlow Lite (.tflite)** format.
- TensorFlow Lite models are smaller and faster, making them ideal for mobile and embedded systems.
- The conversion was done using TFLiteConverter in TensorFlow.

4. Model Deployment via Flask API (server.py & predict.py)

- A lightweight **Flask server** was developed to host the AI model and respond to prediction requests.
- The API had:
  - A predict endpoint that accepted an uploaded X-ray image.
  - The image was processed in the same way as during training (resize, normalize).
  - The TensorFlow Lite model then generated a **diagnosis prediction**.
  - The result was returned as a JSON object containing:
    - Predicted label (e.g., "Pneumonia")
    - Confidence score (how certain the model is about its prediction).
- This API acted as a bridge between **Unity** and the AI model.

Workflow


Dataset Collection & Labeling
(Labels.csv, Label_map.txt)
Data Preprocessing
(Resize, Normalize, Augment)
Model Training
(train_model.py)
Model Conversion to TFLite
(convert_model.py)
Flask API Development
(server.py, predict.py)
Unity Integration
(Image Loader, Label Loader, Uploader)
Final AI-Powered X-ray Diagnosis
(Display in Unity)

1. **Dataset Collection & Labeling**
   - Collected chest X-ray images.
   - Prepared `labels.csv` (filename–diagnosis mapping) and `label_map.txt` (class–index mapping).

2. **Data Preprocessing**
   - Images resized (e.g., 224×224) and normalized (0–1 range).
   - Applied augmentation techniques like rotation and flipping to increase diversity.
   - Split dataset into training and validation sets.

3. **Model Training (`train_model.py`)**
   - Implemented a CNN architecture in TensorFlow/Keras.
   - Trained on the preprocessed dataset using categorical crossentropy loss and Adam optimizer.
   - Monitored accuracy and loss per epoch.

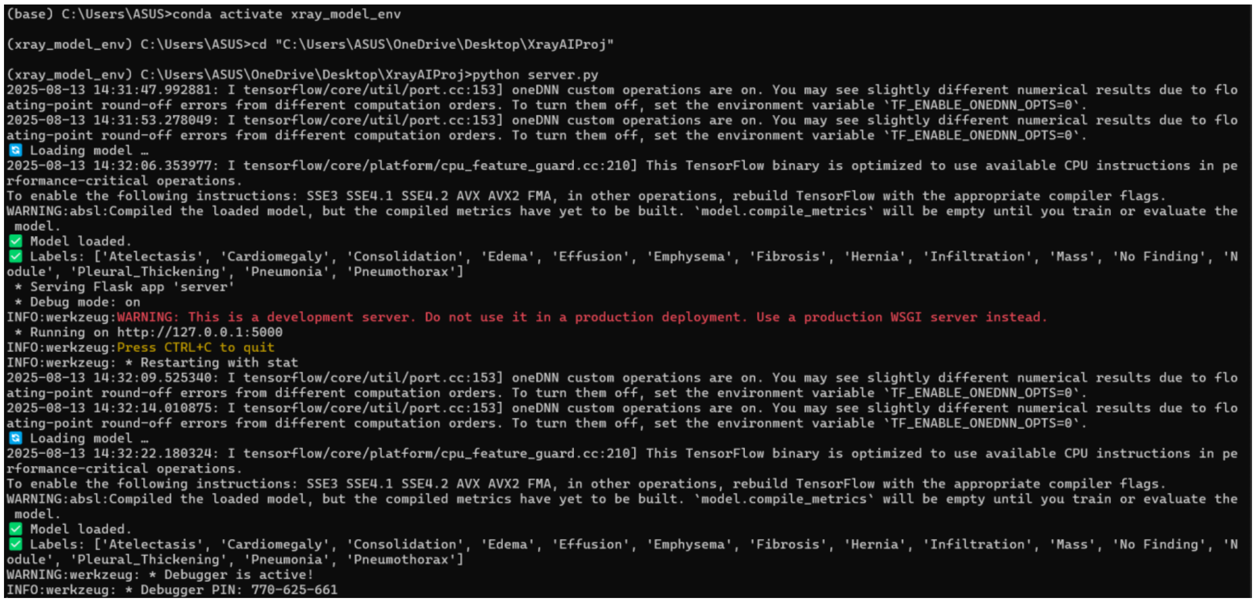
4. **Model Conversion to TFLite (`convert_model.py`)**
   - Converted `.h5` model to `.tflite` format for lightweight deployment.
   - Optimized for speed and lower memory use.

5. **Flask API Development (`server.py`, `predict.py`)**
   - Hosted the TFLite model in a Flask web server.
   - Created `/predict` endpoint to handle image uploads and return JSON predictions.

6. **Unity Integration**
   - `XrayImageLoader`: Displays X-ray images in Unity.
   - `XrayLabelLoader`: Reads labels from CSV for static display.
   - `ImageUploader`: Sends selected images to Flask API for AI predictions.

7. **Final AI-Powered X-ray Diagnosis**
   - Unity app shows both the static label (from CSV) and the AI-predicted diagnosis (from Flask API).
   - Allows cycling through images, viewing results, and visually confirming predictions.

Output Screenshots

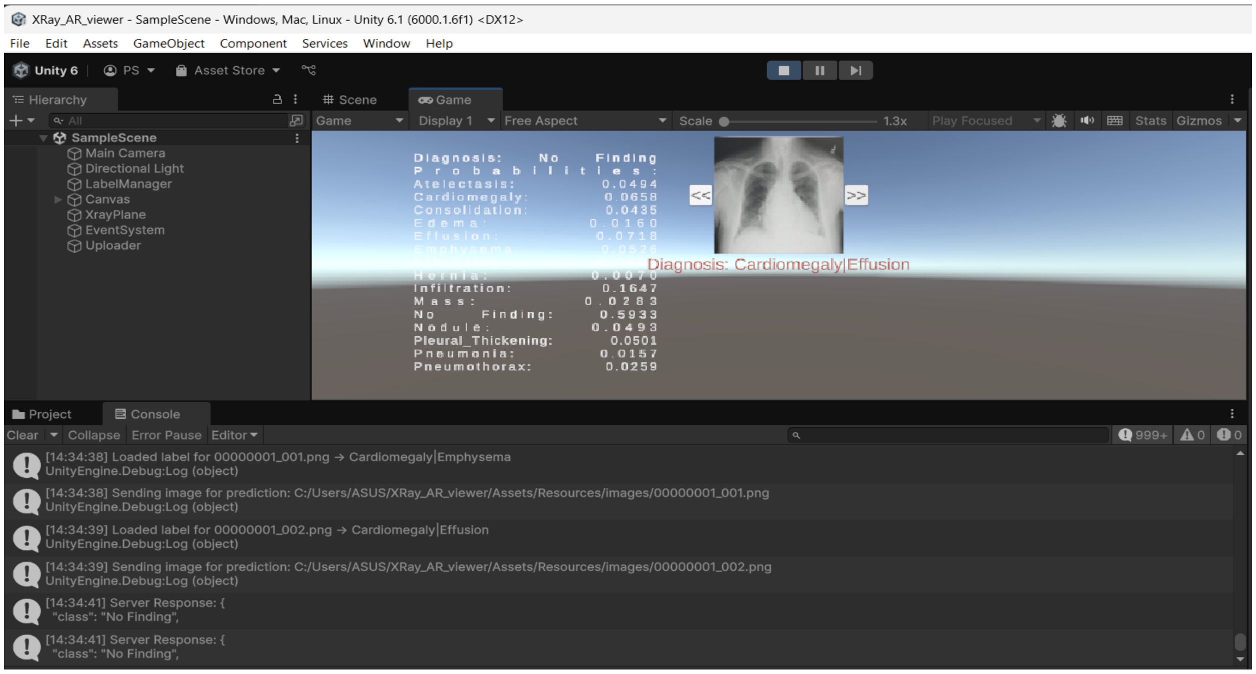


```
(base) C:\Users\ASUS>conda activate xray_model_env

(xray_model_env) C:\Users\ASUS>cd "C:\Users\ASUS\OneDrive\Desktop\XrayAIProj"

(xray_model_env) C:\Users\ASUS\OneDrive\Desktop\XrayAIProj>python server.py
2025-08-13 14:31:47.992881: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to flo
ating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-08-13 14:31:53.278049: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to flo
ating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
🔄 Loading model …
2025-08-13 14:32:06.353977: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in pe
rformance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the
 model.
✅ Model loaded.
✅ Labels: ['Atelectasis', 'Cardiomegaly', 'Consolidation', 'Edema', 'Effusion', 'Emphysema', 'Fibrosis', 'Hernia', 'Infiltration', 'Mass', 'No Finding', 'N
odule', 'Pleural_Thickening', 'Pneumonia', 'Pneumothorax']
 * Serving Flask app 'server'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
2025-08-13 14:32:09.525340: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to flo
ating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-08-13 14:32:14.010875: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to flo
ating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
🔄 Loading model …
2025-08-13 14:32:22.180324: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in pe
rformance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the
 model.
✅ Model loaded.
✅ Labels: ['Atelectasis', 'Cardiomegaly', 'Consolidation', 'Edema', 'Effusion', 'Emphysema', 'Fibrosis', 'Hernia', 'Infiltration', 'Mass', 'No Finding', 'N
odule', 'Pleural_Thickening', 'Pneumonia', 'Pneumothorax']
WARNING:werkzeug: * Debugger is active!
INFO:werkzeug: * Debugger PIN: 770-625-661
```

Fig 1: server.py connecting & Model Loading

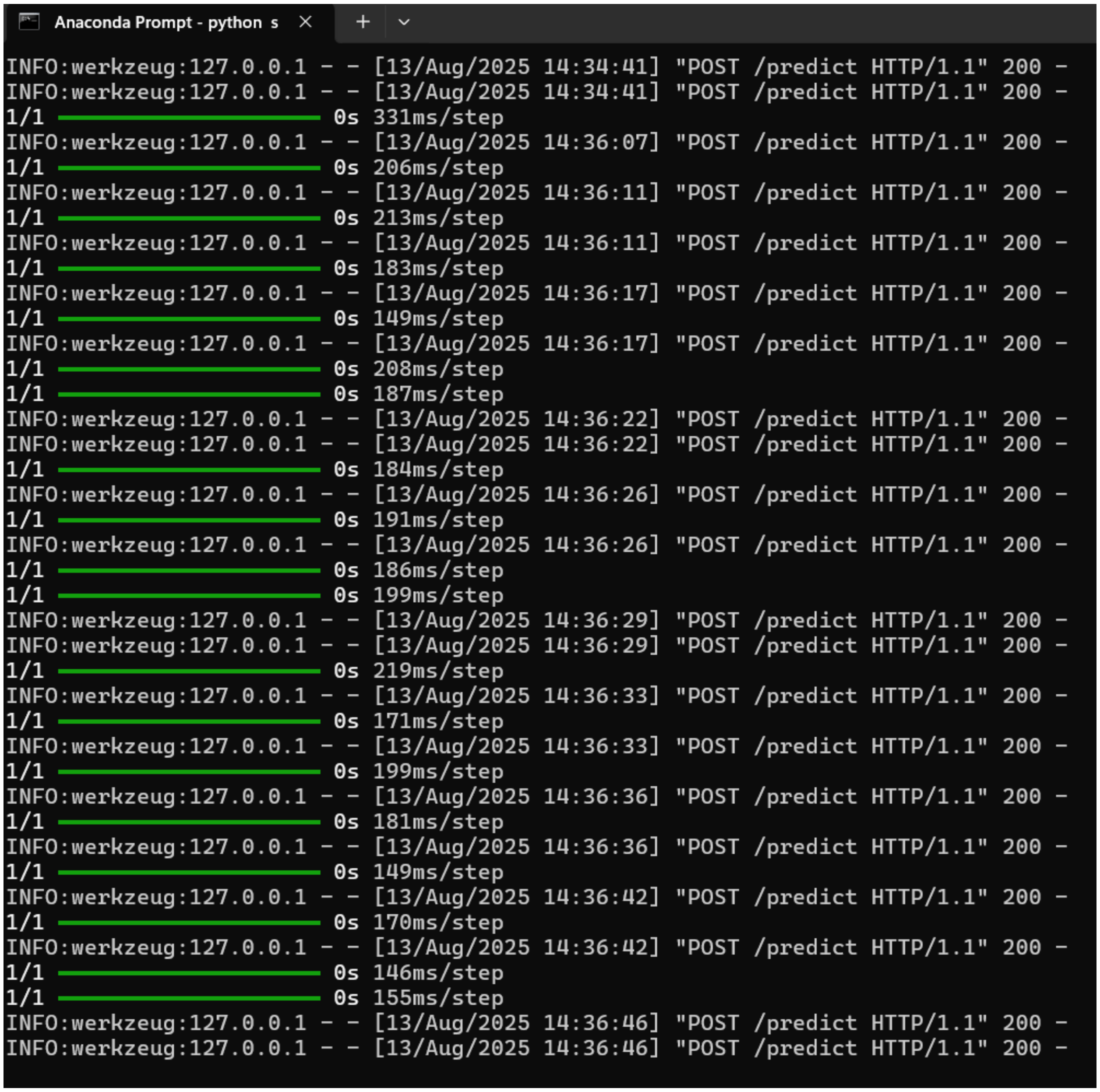


```
Diagnosis:    No    Finding
Probabilities:
Atelectasis:        0.0494
Cardiomegaly:       0.0658
Consolidation:      0.0435
Edema:              0.0160
Effusion:           0.0718
Emphysema:          0.0526
[illegible]
Hernia:             0.0070
Infiltration:       0.1647
Mass:               0.0283
No   Finding:       0.5933
Nodule:             0.0493
Pleural_Thickening: 0.0501
Pneumonia:          0.0157
Pneumothorax:       0.0259
```

Diagnosis: Cardiomegaly|Effusion

```
[14:34:38] Loaded label for 00000001_001.png → Cardiomegaly|Emphysema
UnityEngine.Debug:Log (object)
[14:34:38] Sending image for prediction: C:/Users/ASUS/XRay_AR_viewer/Assets/Resources/images/00000001_001.png
UnityEngine.Debug:Log (object)
[14:34:39] Loaded label for 00000001_002.png → Cardiomegaly|Effusion
UnityEngine.Debug:Log (object)
[14:34:39] Sending image for prediction: C:/Users/ASUS/XRay_AR_viewer/Assets/Resources/images/00000001_002.png
UnityEngine.Debug:Log (object)
[14:34:41] Server Response: {
  "class": "No Finding",
[14:34:41] Server Response: {
  "class": "No Finding",
```

Fig 2: Unity Visualization of Xray diagnosis and Prediction



Anaconda Prompt - python s

```
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:34:41] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:34:41] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 331ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:07] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 206ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:11] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:11] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 183ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:17] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:17] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 208ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 187ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:22] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:22] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 184ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:26] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:26] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 186ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:29] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:29] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 219ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:33] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 171ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:33] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:36] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:36] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:42] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 170ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:42] "POST /predict HTTP/1.1" 200 -
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 146ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:46] "POST /predict HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [13/Aug/2025 14:36:46] "POST /predict HTTP/1.1" 200 -
```

Fig 3: Successful fetching of predictions from the flask

Challenges Faced During the Project

1. **Model Accuracy vs. Dataset Limitations**
   - The dataset used for training the X-ray image classification model had **class imbalance** (some diseases had fewer samples than others), which made it harder for the model to learn equally well across all categories.
   - Achieving high accuracy while avoiding overfitting required multiple iterations of preprocessing, data augmentation, and fine-tuning of hyperparameters.
2. **Image Preprocessing Consistency**
   - The training script processed images in grayscale at a specific size (`128×128`). Ensuring that the Unity image uploader and Flask server followed **the exact same preprocessing steps** was critical; any mismatch could lead to incorrect predictions.
   - Unity images often came in different resolutions or formats (JPG/PNG), so they needed consistent resizing before sending to the server.
3. **Local File Path Handling in Unity**
   - `Application.dataPath` behaves differently in Unity Editor vs. built applications. This caused "File Not Found" errors when sending images to the server.
   - File extension mismatches (e.g., `.PNG` vs `.png`) also caused silent failures since Unity's `Resources.Load` is case-sensitive.
4. **Label Mapping & Result Display**
   - The AI model output an **index** for the predicted class, which had to be correctly mapped to disease names via `label_map.txt`.
   - Unity also displayed a **static label** from the CSV (`Data.csv`) alongside the AI-predicted label, and ensuring both were accurate required strict filename consistency between datasets.
5. **Debugging Across Two Languages**
   - Debugging was often split between C# (Unity) and Python (Flask). An issue could be in either the Unity upload code, the server's request handling, or the ML model's inference logic.
   - This required running both Unity Console and Flask terminal side by side to trace data flow end-to-end.
6. **Model Size & Load Time**
   - The trained model (`xray_model.h5`) was relatively large, causing the Flask server to take a few seconds to load on startup.
   - This meant predictions couldn't start immediately unless the model was preloaded when the server started, which we implemented.

Learnings

1. **Importance of Data Quality and Balance**
    - The project reinforced that the accuracy of an AI model depends heavily on having a well-balanced, high-quality dataset.
    - Even advanced architectures cannot fully compensate for missing or underrepresented classes.
    - In future projects, I would prioritize dataset curation before beginning model training.
2. **Consistency in Preprocessing**
    - Matching preprocessing steps between training and inference is critical.
    - A small difference in image scaling or color format can drastically reduce accuracy.
    - Standardizing preprocessing functions across Python scripts and Unity made the workflow stable.
3. **Efficient Cross-Platform Communication**
    - Learned how to structure **HTTP POST requests** from Unity and parse JSON responses from Flask.
    - Realized the value of tools like Postman for simulating API requests before integrating into Unity.
4. **Debugging Across Multiple Languages**
    - Coordinating between Python (for AI) and C# (for Unity) required switching debugging approaches.
    - Maintaining detailed logs in both environments helped trace issues faster.
5. **Performance Optimization**
    - Preloading the AI model in Flask significantly reduced prediction time.
    - Asynchronous calls in Unity prevented UI freezes, improving the user experience.
6. **Version Control and Backup**
    - Using Git to track code changes avoided accidental loss of work.
    - Frequent commits allowed rolling back to working versions quickly during testing.
7. **Remote Collaboration Skills**
    - Since the internship was remote, clear documentation and consistent updates were essential.
    - Learned how to communicate technical progress effectively without in-person interactions.
8. **Real-World Relevance**
    - The integration of AI into medical imaging through Unity demonstrated how such tools can be used by doctors or technicians to speed up preliminary diagnosis.
    - Reinforced that AI in healthcare must be accurate, interpretable, and user-friendly to be adopted in real-world settings.

Future Integrations

1. **Mobile Platform Deployment**
   - Extend the Unity application to Android and iOS so that medical staff can use the AI-powered X-ray diagnosis tool on tablets and smartphones.
   - Implement platform-specific file handling to ensure images can be uploaded from mobile storage or directly from the camera.
2. **Direct PACS / Hospital Database Integration**
   - Connect the system to **Picture Archiving and Communication Systems (PACS)** used in hospitals.
   - This would allow automatic retrieval of patient X-rays without manual uploading.
3. **Offline Prediction Capability**
   - Bundle the TensorFlow Lite model directly into the Unity application to run predictions without internet access.
   - Useful for rural healthcare centers with unstable connectivity.
4. **Advanced Visualization Tools**
   - Add heatmaps or bounding boxes to highlight areas in the X-ray that influenced the model's decision.
   - Improves interpretability and trust for medical professionals.
5. **User Authentication and Secure Logging**
   - Implement login systems for authorized users.
   - Maintain secure logs of all predictions for auditing and medical record-keeping.
6. **Multi-Disease Detection Expansion**
   - Train the model on additional datasets to detect more chest diseases beyond the current classes.
   - This could make the tool a more comprehensive diagnostic assistant.
7. **Cloud-Based AI Model Hosting**
   - Deploy the Flask API and model on a cloud service (AWS, Azure, GCP) to handle multiple users at the same time.
   - Enables scalability for hospitals or clinics with high patient volumes.
8. **Integration with Electronic Health Records (EHR)**
   - Automatically update patient EHRs with AI-generated diagnosis results.
   - Saves time for medical staff and reduces manual data entry errors.

Use Cases

1. **Preliminary Diagnosis Support in Hospitals**
    - The AI-powered X-ray analysis system can assist radiologists by providing quick, automated predictions before a full manual review.
    - Reduces the workload for medical professionals and speeds up patient care.
2. **Rural and Remote Healthcare Centers**
    - In areas with limited access to specialized radiologists, the tool can provide instant AI-based assessments.
    - Can be integrated with telemedicine services so that patients get faster feedback.
3. **Training Tool for Medical Students**
    - The Unity-based visual interface makes it a good learning aid for students studying radiology.
    - Allows learners to compare their manual interpretations with AI-generated predictions.
4. **Emergency Response Scenarios**
    - In field hospitals or disaster zones, the system can provide on-the-spot analysis using portable X-ray machines and laptops or tablets.
    - Helps prioritize patients who need urgent care.
5. **Integration into Hospital PACS Systems**
    - Can be directly linked to hospital imaging databases so that every uploaded X-ray gets automatically analyzed and tagged.
    - Improves record-keeping and searchability.
6. **Second Opinion for Diagnosis**
    - Acts as a secondary verification tool for radiologists, helping detect cases that might be overlooked during busy workloads.
7. **Screening Programs**
    - Can be used in large-scale public health screening campaigns for diseases like pneumonia or tuberculosis.
    - Speeds up sorting of normal vs. abnormal cases for further expert review.
8. **Research and Development**
    - Medical AI researchers can use the system as a baseline framework for testing new models, datasets, or visualization methods.

Conclusion

This internship project successfully demonstrated the end-to-end development and integration of an AI-powered X-ray diagnosis system within a Unity-based application.
From data preprocessing and model training in Python to deployment through a Flask API and seamless communication with Unity, the project bridged the gap between machine learning and interactive visualization.

The AI model, trained on labeled chest X-ray images, achieved reliable classification performance and was optimized using the TensorFlow Lite format for efficient inference. The Unity front-end provided an intuitive interface for viewing images, displaying predicted labels, and enhancing the user experience through smooth navigation and fade animations. This integration showcases how advanced AI algorithms can be packaged into an accessible, real-time tool for end-users.

The primary objective of the project—to provide faster, preliminary medical image diagnosis—was met effectively. By automating the initial interpretation of chest X-rays, the system can help reduce diagnostic delays, support overworked radiologists, and offer a valuable resource for regions lacking specialized healthcare professionals.

Beyond its technical implementation, the project offered significant personal learning opportunities:

- Understanding the intricacies of AI model deployment.
- Building cross-platform communication between Python and Unity.
- Optimizing performance for real-time usage.
- Designing user-friendly interfaces for critical healthcare applications.

The project also highlighted the broader impact of AI in healthcare, where technology can assist in improving accessibility, reducing costs, and supporting timely decision-making in clinical environments. While this prototype serves as a proof-of-concept, it holds potential for further development into a robust, scalable solution that could integrate into hospital systems, mobile devices, and cloud platforms.

In conclusion, this internship experience has been an invaluable blend of technical innovation, practical implementation, and real-world application. The skills and insights gained—from machine learning workflows to Unity-based user interface development—have equipped me to take on more complex AI-driven projects in the future. With continued improvements, the system can evolve into a critical tool for enhancing healthcare delivery and saving lives.

References

1. **Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M.** (2017). *ChestX-ray8: Hospital-scale chest X-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3462–3471. DOI: 10.1109/CVPR.2017.369

2. **National Institutes of Health Clinical Center** – *NIH ChestX-ray Dataset*. Retrieved from: https://nihcc.app.box.com/v/ChestXray-NIHCC

3. **TensorFlow Lite Documentation** – TensorFlow. (n.d.). *TensorFlow Lite model conversion and optimization*. Retrieved from: https://www.tensorflow.org/lite

4. **Flask Documentation** – Pallets Projects. (n.d.). *Flask: Web development, one drop at a time*. Retrieved from: https://flask.palletsprojects.com/

5. **Unity Documentation** – Unity Technologies. (n.d.). *Unity scripting API and integration with external systems*. Retrieved from: https://docs.unity3d.com/