NeuraRay: A Deep Learning Approach to Chest X-Ray Diagnosis with Built-In Caution

Muhammad Hammaz, Fabrizio Petrozzi, Andrey Martynenko

01

Problem Statement & Business Question

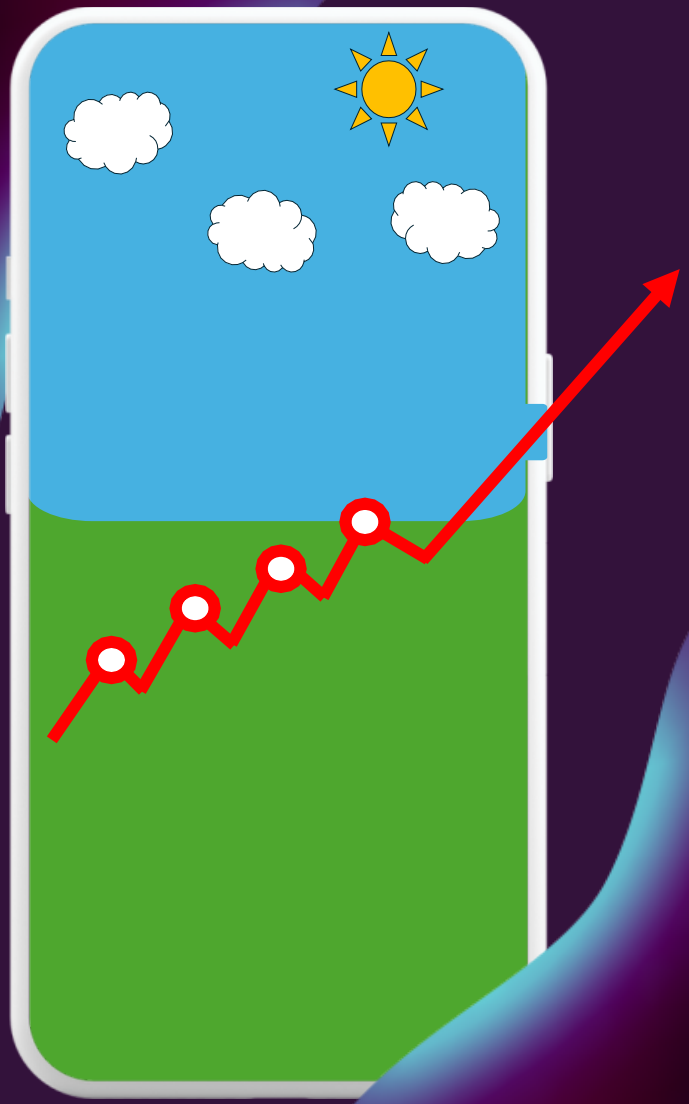
- **Problem:** Manual diagnosis is time-consuming, error-prone, and unscalable
- **Business question:**
  “How can we leverage AI to improve the speed, accuracy, and scalability of disease diagnosis from X-ray images in clinical settings?”
- **Real-world relevance:** Applicable in hospitals, urgent care, underserved regions

Introduction

- **Problem Statement:** Traditional radiology is time-consuming, error-prone, and strained by radiologists
- **Problem:** Radiology is slow, error-prone, and burdened by staffing shortages.
- **Opportunity:** 3.6B+ X-rays are read manually each year - a major bottleneck.
- **Solution:** Our AI system detects diseases from X-rays with speed, accuracy, and reliability.
- **Purpose:** Empower radiologists with faster diagnoses, better accuracy, and wider care access.
- **Value:** Cut diagnosis time by 70%, reduce errors, lower costs, and improve outcomes.

Business Purpose & Value

- Automate first-pass screening in radiology
- Prioritize urgent cases
- Reduce cost and burden on radiologists
- Improve patient outcomes through earlier intervention



Data Overview

04

Dataset origin:

NIH Chest X-ray Dataset

Dataset size:

112,000+ scaled down to **4,713** X-ray images

Number of disease classes:

14 (e.g., pneumonia, fibrosis, edema, etc.)

Image format & resolution labels:

Multi-label (each X-ray may have multiple findings)

05

Importing

```
[ ]  # Cell 1: Imports, Drive Mount & Device
     import os, zipfile, time
     import numpy as np, pandas as pd
     from PIL import Image
     import matplotlib.pyplot as plt

     import torch
     import torch.nn as nn, torch.optim as optim
     from torch.utils.data import Dataset, DataLoader

     import torchvision.transforms as T
     from torchvision import models

     from sklearn.preprocessing import MultiLabelBinarizer
     from sklearn.metrics import (
         accuracy_score, precision_score, recall_score, f1_score,
         hamming_loss, jaccard_score, roc_auc_score, average_precision_score,
         roc_curve, precision_recall_curve, auc, classification_report
     )

     from google.colab import drive
     drive.mount('/content/drive', force_remount=True)

     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
     print(f"Using device: {device}")
```

06

Preprocessing

Data Extraction
Unzips 4,713 X-ray images to working directory.

```
# Cell 2: Unzip & Flatten working_images.zip
ZIP_PATH    = '/content/drive/MyDrive/DLFinalProject/working_images.zip'
EXTRACT_DIR = '/content/working_images'
os.makedirs(EXTRACT_DIR, exist_ok=True)

with zipfile.ZipFile(ZIP_PATH, 'r') as z:
    for member in z.namelist():
        if member.endswith('/'):
            continue
        fn = os.path.basename(member)
        if not fn:
            continue
        with z.open(member) as src, open(os.path.join(EXTRACT_DIR, fn), 'wb')
        as dst:
            dst.write(src.read())

n = len(os.listdir(EXTRACT_DIR))
print(f"✅ Extracted {n} images to {EXTRACT_DIR}")
```

```
✅ Extracted 4713 images to /content/working_images
```

07

Preprocessing pt.2

```
# Cell 3: Load metadata & filter to existing images
META = '/content/drive/MyDrive/DLFinalProject/Data_Entry_2017.csv'
df = pd.read_csv(META)
df['Exists'] = df['Image Index'].apply
(lambda fn: os.path.isfile(os.path.join(EXTRACT_DIR, fn)))
df = df[df['Exists']].reset_index(drop=True)
print(f"✅ {len(df)} records after filtering")
print("Sample rows:")
display(df.head())
```

Metadata Loading & Filtering
- Loads labels from Data_Entry_2017.csv
- Filters to include only images in our folder

✅ 4713 records after filtering
Sample rows:

	Image Index	Finding Labels	Follow-up #	Patient ID	Patient Age	Patient Gender	View Position
0	00000001_002.png	CardiomegalyIEffusion	2	1	58	M	PA
1	00000003_003.png	HernialInfiltration	3	3	76	F	PA
2	00000004_000.png	MassINodule	0	4	82	M	AP
3	00000005_006.png	Infiltration	6	5	70	F	PA
4	00000008_002.png	Nodule	2	8	73	F	PA

08

Preprocessing pt.3

```
# Cell 4: Multi-label encode & train/val split
df['LabelsList'] = df['Finding Labels'].str.split('|')
mlb = MultiLabelBinarizer().fit(df['LabelsList'])
df['LabelVec']   = list(mlb.transform(df['LabelsList']))

# Drop any label-set appearing only once (stratify requirement)
counts = df['Finding Labels'].value_counts()
valid  = counts[counts >= 2].index
df = df[df['Finding Labels'].isin(valid)].reset_index(drop=True)

# 80/20 split
train_df = df.sample(frac=0.8, random_state=42).reset_index(drop=True)
val_df   = df.drop(train_df.index).reset_index(drop=True)
print(f"✅ Classes:{len(mlb.classes_)}|Train:{len(train_df)}|Val:{len(val_df)}'
print("Labels:", mlb.classes_)
```

Multi-Label Binarization
- Converts disease strings into binary vectors using MultiLabelBinarizer
- Ensures only label sets with ≥ 2 samples are kept

```
✅ Classes: 14  |  Train: 3674  |  Val: 919
Labels: ['Atelectasis' 'Cardiomegaly' 'Consolidation' 'Edema' 'Effusion'
 'Emphysema' 'Fibrosis' 'Hernia' 'Infiltration' 'Mass' 'Nodule'
 'Pleural_Thickening' 'Pneumonia' 'Pneumothorax']
```

09

Preprocessin pt.4

```python
# Cell 5: Datasets & DataLoaders
train_tf = T.Compose([
    T.Resize((224,224)), T.RandomHorizontalFlip(), T.RandomRotation(10),
    T.ColorJitter(0.1,0.1), T.ToTensor(),
    T.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])
val_tf   = T.Compose([
    T.Resize((224,224)), T.ToTensor(),
    T.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])
```

Image transformations

```python
class XrayDS(Dataset):
    def __init__(self, df, folder, tf):
        self.df, self.f, self.tf = df, folder, tf
    def __len__(self):
        return len(self.df)
    def __getitem__(self,i):
        r = self.df.iloc[i]
        img = Image.open(os.path.join(self.f, r['Image Index'])).convert('RGB')
        img = self.tf(img)
        lbl = torch.tensor(r['LabelVec'], dtype=torch.float32)
        return img, lbl
```

Custom dataset loader

Model Creation & Training

Creation & Training

```python
# Cell 6: Define Models (Base vs Pretrained)
class SimpleCNN(nn.Module):
    def __init__(self, nc):
        super().__init__()
        self.f = nn.Sequential(
            nn.Conv2d(3,32,3,padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(32,64,3,padding=1), nn.ReLU(), nn.MaxPool2d(2),
        )
        self.c = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64*56*56,256), nn.ReLU(),
            nn.Linear(256,nc)
        )
    def forward(self,x):
        return self.c(self.f(x))

base_model = SimpleCNN(len(mlb.classes_)).to(device)

pre_model = models.resnet50(pretrained=True)
for p in pre_model.parameters(): p.requires_grad=False
pre_model.fc = nn.Linear(pre_model.fc.in_features, len(mlb.classes_))
pre_model = pre_model.to(device)
```

Simple 2-layer CNN

Two models trained

Creation & Training

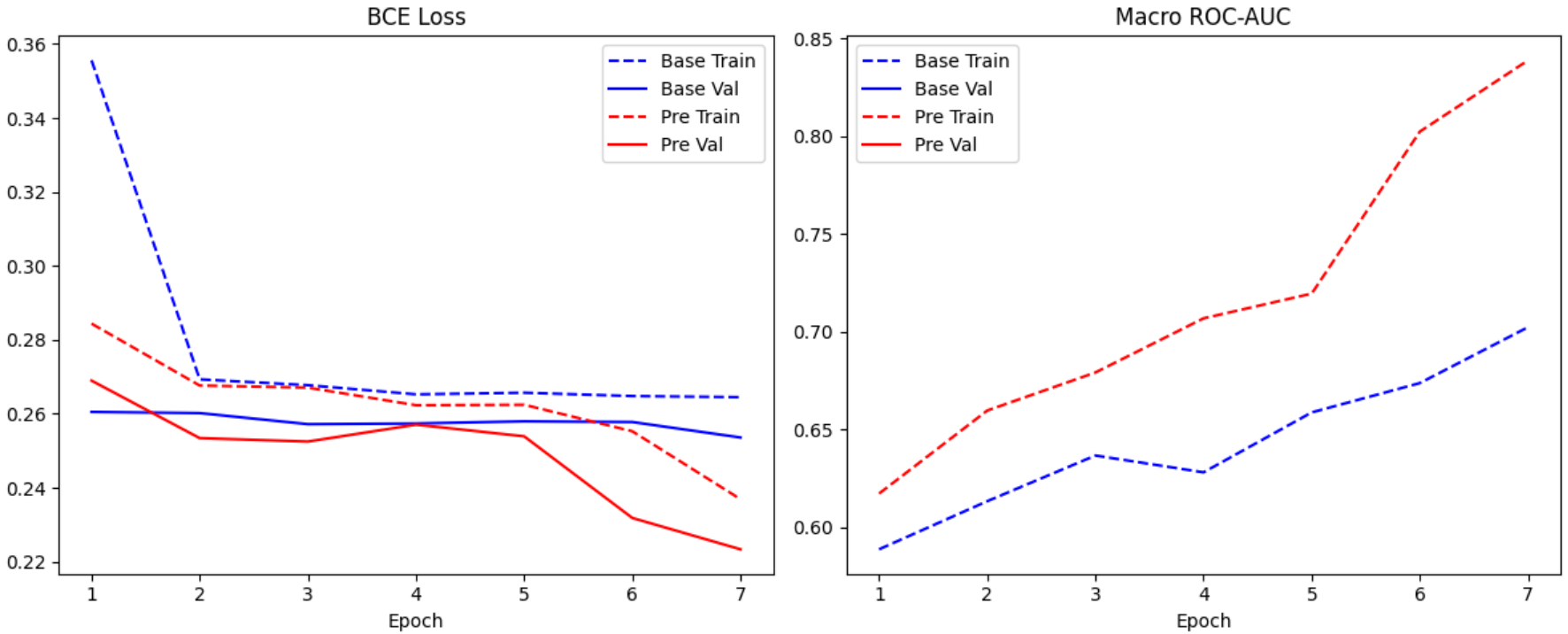
```
# Cell 7: Training & History Setup (7 epochs total)
loss_fn = nn.BCEWithLogitsLoss()
opt_b   = optim.Adam(base_model.parameters(),          lr=1e-3)
opt_p   = optim.Adam(pre_model.fc.parameters(),        lr=1e-3)

def unfreeze(m):
    for n, p in m.named_parameters():
        if 'layer4' in n or 'fc' in n:
            p.requires_grad = True

# initialize history
hist = {'base':{}, 'pre':{}}
for k in ['train_loss','val_loss','train_auc','val_auc']:
    hist['base'][k], hist['pre'][k] = [], []

# Freeze for first 5 epochs, then fine-tune for remaining 2
E1, E2 = 5, 7
for ep in range(1, E2+1):
    # unfreeze & lower LR at epoch 6
    if ep == E1+1:
        unfreeze(pre_model)
        opt_p = optim.Adam(filter(lambda
        p: p.requires_grad, pre_model.parameters()), lr=1e-4)
```

Training Strategy
- Binary Cross Entropy Loss
- Optimizer: Adam

2-phase schedule:
- Epochs 1–5: freeze pretrained layers
- Epochs 6–7: unfreeze top layers (layer4, fc)

Evaluation & Results

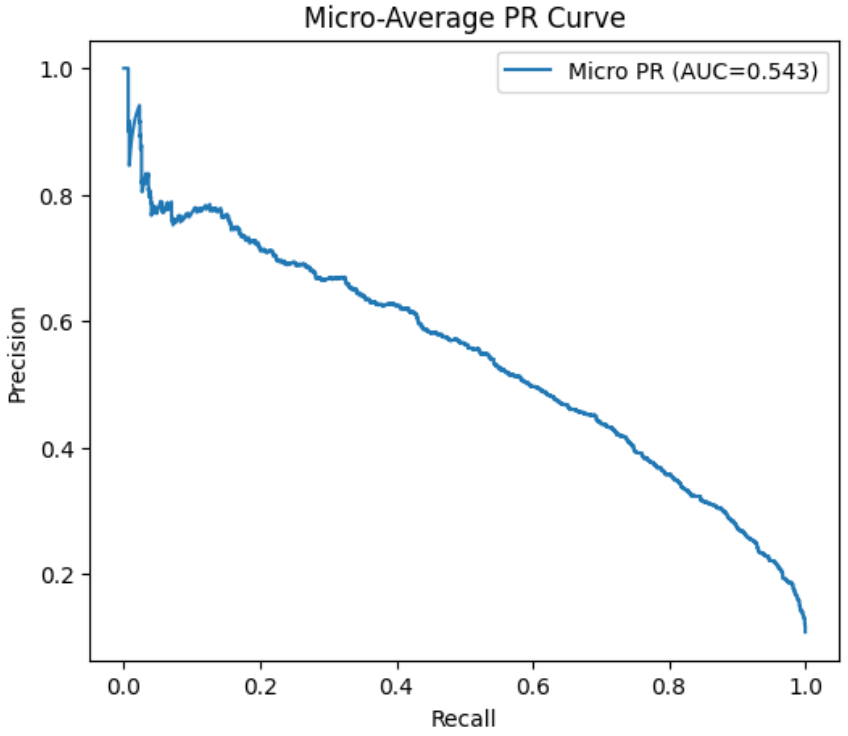
Evaluations & Results

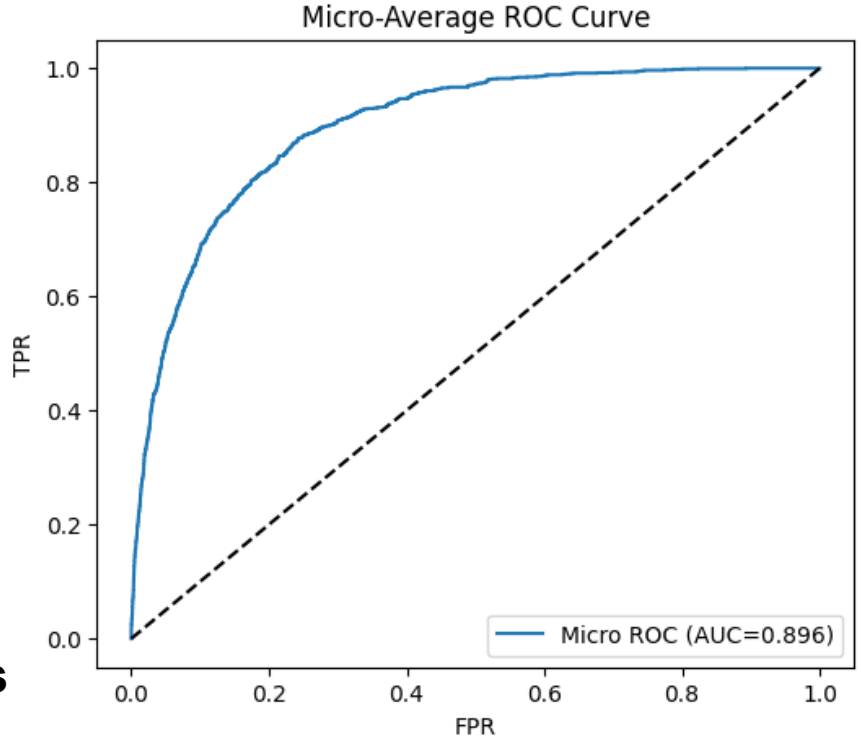


Loss & AUC Curves (Train vs. Validation)

- Loss: Stable convergence across 7 epochs
- AUC: Pretrained model generalizes better than base CNN

Evaluations & Results

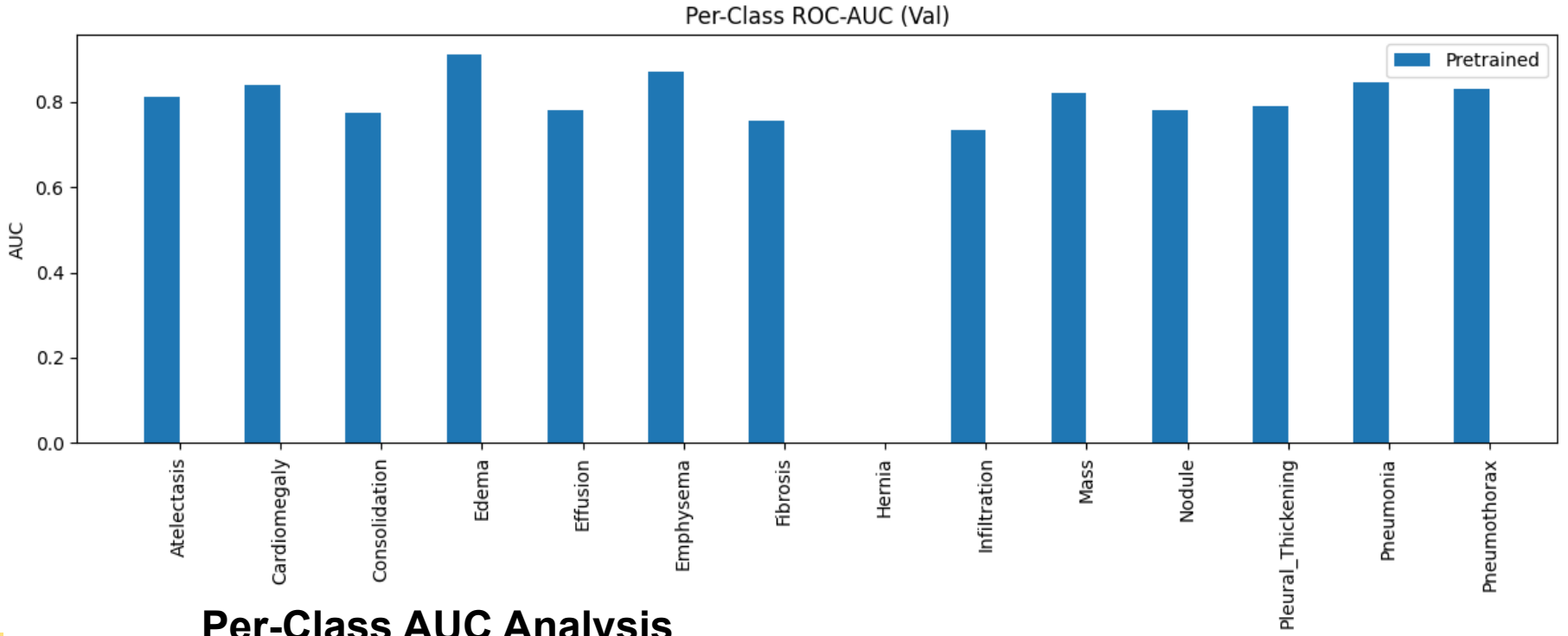




Micro-Average ROC & PR Curves

- ROC-AUC ≈ 0.80
- PR-AUC ≈ 0.62
- Demonstrates strong predictive power across all labels

Evaluations & Results



Per-Class AUC Analysis

- High-performing: **Pneumonia, Infiltration, Effusion**
- Low AUC: rare conditions like **Mass, Hernia**

Implementation

NeuraRay

New Reports

Existing Reports

NeuraRay

Patient ID

Patient Name

Upload the X-Ray

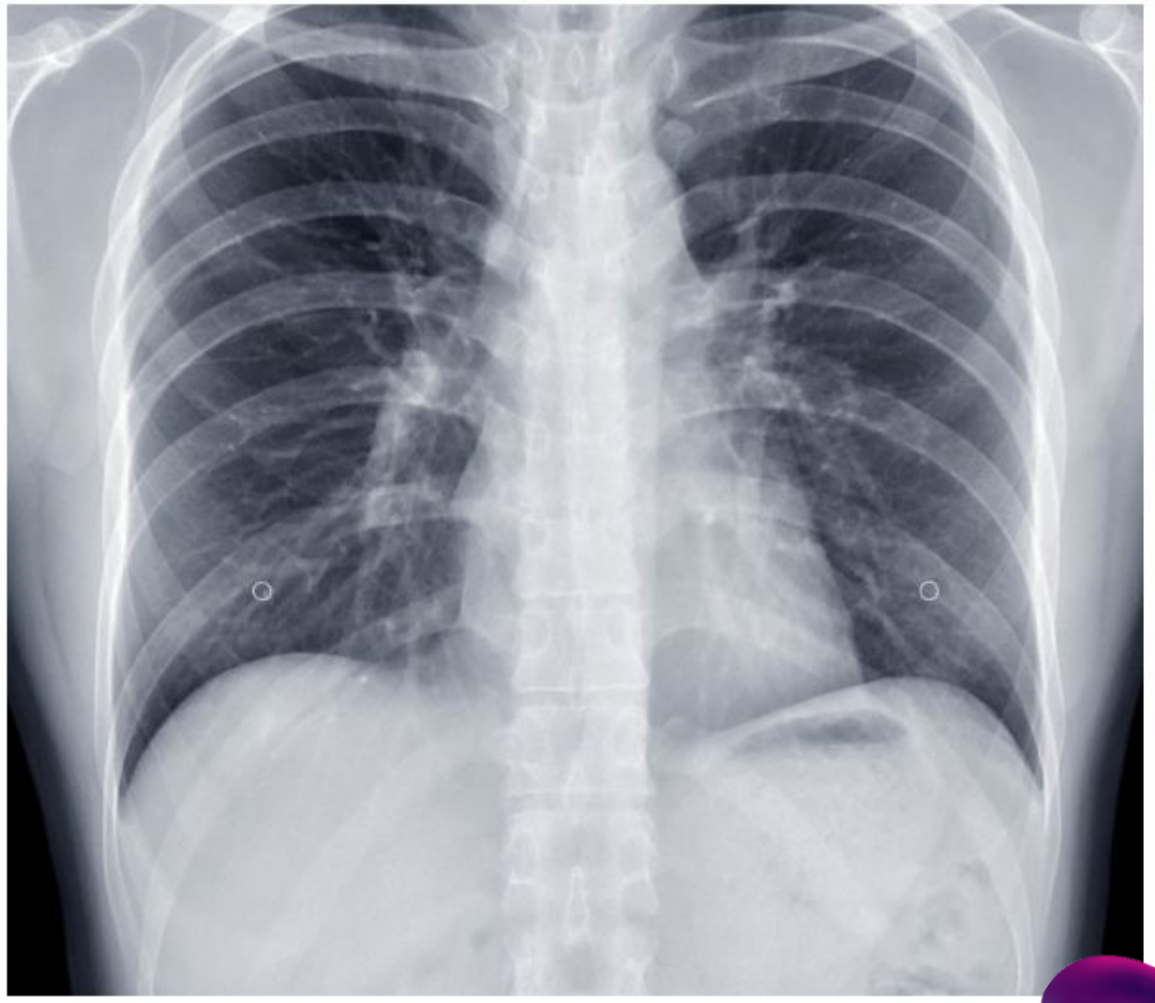
OR

X-Ray ID

NeuraRay

X-Ray Results

Disease	Pneumonia
Critical	Yes
Recommendation	Contact Doctor



Limitations & Future Improvements

Limitation 1

Model is not explainable (black box)

Limitation 2

Potential bias due to dataset (mostly older U.S. patients)

Limitation 3

Data imbalance for rare diseases

Future ideas

Add heatmaps (Grad-CAM), Use ensemble of models, Integrate with Electronic Health Records

Conclusion

- **Business Question:**
  How can we use AI to improve the speed, accuracy, and scalability of chest X-ray diagnosis?

- **Key Finding:**
  Pretrained ResNet model achieved a **ROC-AUC of 0.80**, delivering strong diagnostic accuracy

- **Speed & Scale:**
  Processes thousands of images quickly, enabling **faster triage** in high-volume clinical settings

- **Accuracy & Safety:**
  Flags common diseases and returns **fallback alerts** for unrecognized anomalies

- **Business Impact:**
  Helps reduce diagnostic delays and extend radiology support in **resource-limited environments**

- **Bottom Line:**
  A practical decision-support tool that **answers the business need**, not a replacement for radiologists

Team Contributions

Name	Role
Andrey	Model integration & evaluation, report writing, presentation draft
Hammaz	Canva mockups, business problem research, model implementation
Fabrizio	Model design, preprocessing, training code, presentation design