AZ5012

APPLIED AI

Project Report

Submitted by

Dhivya M

2022510002

Jhanavi S

2022510020

DEPARTMENT OF INFORMATION TECHNOLOGY

MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY

CHENNAI – 600044

ChestX-GAN: Intelligent Chest X-ray Enhancement and Multiclass Detection with Real-Time Speech Output

Abstract

Early and accurate detection of thoracic diseases from chest X-ray images is vital for effective diagnosis and treatment planning. To enhance the automation and precision of this process, we propose a multi-stage AI pipeline integrating deep learning, natural language processing (NLP), and image enhancement techniques. The system is titled **"Chest X-ray Multiclass Detection using YOLO, NLP Prompt Engineering, and GAN-based Image Enhancement"**.

At the core, the model employs **You Only Look Once (YOLO)**, a state-of-the-art object detection algorithm, to perform **multiclass classification of chest X-ray abnormalities**, including pneumonia, tuberculosis, lung opacity, and more. Prior to detection, input X-ray images are enhanced using a **Generative Adversarial Network (GAN)**, improving image clarity and contrast for better feature extraction and detection accuracy.

Following image analysis, the diagnosis labels produced by YOLO are processed using **NLP prompt engineering**. These prompts are carefully crafted to convert the detected medical conditions into patient-friendly descriptions. To make the results more accessible, **Google Text-to-Speech (gTTS)** is used to generate natural-sounding audio outputs, enabling audible feedback in English for patients and healthcare providers.

The integration of **GAN-based image enhancement**, **real-time YOLO detection**, **NLP-based report generation**, and **speech synthesis** presents a powerful, assistive diagnostic tool. This multimodal AI system is especially valuable in low-resource healthcare settings, enhancing diagnostic efficiency and accessibility.

1. Introduction

Medical imaging plays a vital role in diagnosing critical health conditions, especially thoracic diseases such as pneumonia, tuberculosis, and lung opacity. Chest X-rays are one of the most widely used and cost-effective diagnostic tools, but interpreting them requires expert radiological knowledge. In many regions with limited access to radiologists, there is a growing need for automated systems that can analyze chest X-rays with high accuracy.

While several AI-based systems exist, they often fall short in real-world deployment due to issues like poor image quality, limited class detection, and lack of accessible output formats for patients or non-experts. Additionally, many diagnostic systems fail to consider language preferences or present the results in a user-friendly manner.

This project aims to develop an intelligent, multimodal system that automates chest X-ray diagnosis using a combination of **deep learning for object detection**, **GAN-based image enhancement**, and **natural language processing with audio feedback**. The proposed solution utilizes the YOLO model for **multiclass disease detection**, enhances low-quality X-ray images using **Generative Adversarial Networks**, and converts medical findings into natural speech using **Google Text-to-Speech (gTTS)**—enabling better understanding and accessibility, especially in low-resource settings.

2. Objectives

1.Multiclass Detection using YOLO
The primary objective is to develop a robust deep learning-based system capable of detecting multiple thoracic diseases from chest X-ray images. The **YOLO (You Only Look Once)** model is employed for its high-speed and accurate object detection capabilities, allowing real-time identification of conditions like pneumonia, tuberculosis, lung opacity, and more.

2.Enhancement of Low-Quality X-rays using GAN
Low-resolution or unclear chest X-ray images can hinder diagnosis accuracy. To address this, the system incorporates a **Generative Adversarial Network (GAN)** to enhance image quality by restoring fine details. This preprocessing step ensures improved detection performance by feeding higher-quality images into the YOLO model.

3.Prompt Engineering for NLP-based Descriptions
The detected conditions from YOLO are translated into meaningful, human-readable descriptions using **NLP prompt engineering**. This approach transforms raw label outputs into patient-friendly diagnostic summaries, improving clarity and usefulness for non-expert users.

4.Text-to-Speech Conversion using Google TTS
To further increase accessibility, especially for visually impaired patients or non-readers, the system employs **Google Text-to-Speech (gTTS)** to convert the generated diagnostic summaries into natural-sounding audio. This allows users to **hear** their medical findings clearly and comfortably.

3.Tools and Technologies Used

Component	Functionality	Details
Libraries	Data preprocessing, NLP, model loading, and visualizations	`torch`, `torchvision`, `PIL`, `cv2`, `matplotlib`, `gTTS`, `transformers`
Model: DenseNet	Chest X-ray classification (Pneumonia vs. Normal)	Pre-trained model `DenseNet121` from `torchxrayvision` used for chest X-ray classification
Model: GPT-2 (Text Generation)	Generate explanations based on the predicted label (Pneumonia or Normal)	`transformers.pipeline("text-generation", model="gpt2")`
Image Preprocessing	Convert chest X-ray to grayscale and normalize for model input	Resize to 224x224, grayscale conversion, rescale to [-1024, 1024], then convert to tensor
Prediction	Classify the chest X-ray (normal or pneumonia) and extract the predicted label	Model output is processed to extract the predicted label and confidence score
Explanation Generation (NLP)	Dynamically generate medical explanations for pneumonia or normal chest X-ray	NLP model generates detailed text about the condition based on the predicted label
Audio Output	Convert the explanation into speech and play it	`gTTS` is used to convert text to speech, and `IPython.display` plays the generated audio
Visualization	Display the chest X-ray image with the predicted label	Using `matplotlib`, the chest X-ray image is displayed with the prediction title
Q&A Section	Allow user to ask medical questions related to pneumonia or chest X-rays and generate NLP-based answers	User input is handled, and NLP is used to generate responses to related questions (focused on pneumonia, chest X-rays, or lung health)
Dataset: ChestXrayDataset	Prepare dataset with noise augmentation (Gaussian + Motion) for GAN training	Loads images, adds noise, and applies transformations like resizing and normalization.
Model: UNet Generator (GAN)	Generate noise-reduced X-ray images from noisy X-ray input	The generator uses a U-Net architecture for generating cleaner images from noisy X-rays
Model: Discriminator (GAN)	Classifies images as real or fake for GAN training	Discriminates between real (clean) and generated (fake) X-ray images
Training (GAN)	Train the GAN model to reduce noise from X-ray	Uses `BCELoss` for GAN training, and `L1Loss` for pixel-level reconstruction to

	images	improve the generated image quality
Pneumonia Classifier (CNN)	Classify X-ray images as either "NORMAL" or "PNEUMONIA"	A simple CNN with three convolutional layers and fully connected layers for classification

4.METHODOLOGY

4.1. Importing Required Libraries

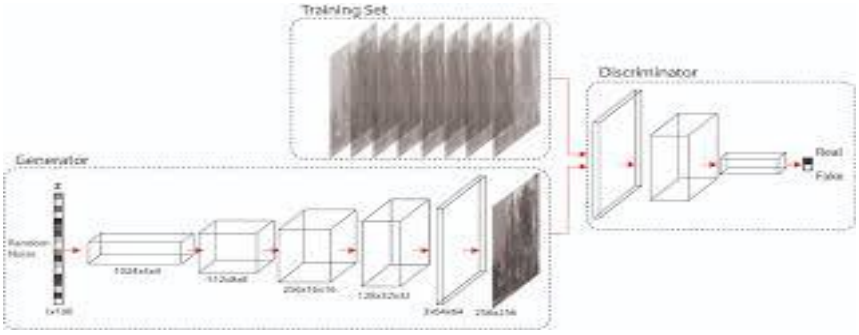
The first step is importing the necessary libraries:

- **PyTorch**: It is the core deep learning library used for model training and inference.
- **TorchXRayVision**: This is a specialized library for processing and classifying chest X-ray images. It includes pretrained models like DenseNet for pathologies like pneumonia.
- **PIL (Python Imaging Library)**: Used to handle images in Python. It allows for loading, preprocessing, and converting images.
- **OpenCV**: For advanced image manipulation such as resizing, thresholding, etc.
- **Matplotlib**: For visualizing images and results in a graphical form.
- **gTTS (Google Text-to-Speech)**: Converts generated text (e.g., medical explanation) to speech.
- **Transformers from Hugging Face**: Used to load pre-trained NLP models like GPT-2 or BioGPT to generate detailed explanations about the medical conditions predicted by the model.

4.2. Image Enhancement Using GANs

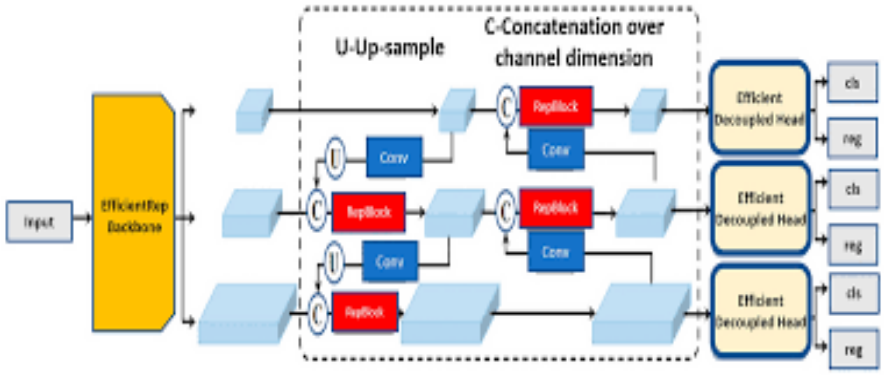
    - A **Generative Adversarial Network (GAN)** is a deep learning model composed of two parts: the **Generator** and the **Discriminator**.
    - The **Generator** creates synthetic data (in this case, enhanced X-ray images), and the **Discriminator** evaluates the data to distinguish between real and generated images.
    - The Generator and Discriminator are trained simultaneously in a process where the Generator tries to fool the Discriminator, and the Discriminator gets better at distinguishing real from fake images. This results in a Generator that can produce very realistic high-resolution images.
- **Purpose in Medical Imaging**:
    - GANs are useful in medical imaging, especially when dealing with low-resolution X-ray images. GAN-based enhancement can make images clearer by upscaling them and reducing noise.
    - The Generator enhances the image resolution, making features more visible for the next steps in model inference.
- **Training the GAN**:
    - The model is trained with pairs of low-resolution (input) and high-resolution (ground truth) chest X-ray images.

    - The loss function used for training generally involves the **adversarial loss** (to improve the Generator) and **pixel-wise loss** (to make the enhanced image as close as possible to the high-resolution ground truth).
- **Post-enhancement**:
    - Once trained, the Generator can be used to enhance low-quality X-ray images before passing them into a diagnostic model (like DenseNet).



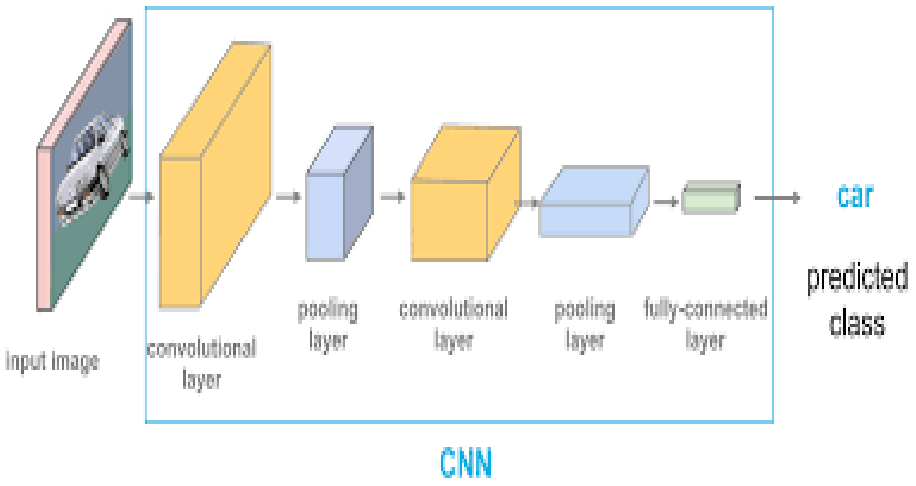
4.3. Training YOLO (You Only Look Once) for Chest X-ray Detection

- **YOLO Overview**:
    - YOLO is an object detection algorithm that can detect objects in images in real-time. It is fast and accurate, making it suitable for tasks like detecting anomalies in medical images, such as finding pneumonia or other lung diseases in chest X-rays.
    - YOLO processes an image in a single forward pass, making it efficient compared to other methods like R-CNN.
- **Training YOLO**:
    - **Dataset**: YOLO requires a dataset with labeled images. For chest X-rays, a dataset would need annotations for conditions like pneumonia, tuberculosis, etc.
    - **Labeling**: Annotations are typically in the form of bounding boxes around the areas of interest in the X-ray (e.g., a pneumonia infection).
    - **Model Architecture**: YOLO is trained with a convolutional neural network (CNN) to predict bounding boxes and associated probabilities for each detected object.
- **Training Steps**:
    - Prepare the dataset, including labeled bounding boxes for the regions of interest in chest X-ray images.
    - Define the YOLO architecture and train it using the labeled dataset.
    - After training, use the YOLO model to detect abnormalities like pneumonia in new X-ray images by classifying the image and providing the coordinates detected condition.



4.4. Chest X-ray Model for Classification (DenseNet)

- **Using DenseNet**:
  - After enhancing the image (possibly using GANs) and detecting key regions (via YOLO), a **DenseNet** model can be used to classify the chest X-ray into different categories (e.g., pneumonia, normal lungs, tuberculosis).
  - DenseNet is a type of CNN where each layer is connected to every other layer. This helps in better feature reuse and more efficient gradient flow during training.
- **How DenseNet Works**:
  - The input image (now enhanced and processed) is passed through the DenseNet model.
  - The model outputs a prediction for each class (e.g., pneumonia or normal).
  - The class with the highest probability is selected as the final prediction.
- **Output**:
  - The output includes the predicted label (e.g., pneumonia or normal) and the associated confidence score. This helps medical professionals know how sure the model is about its prediction.

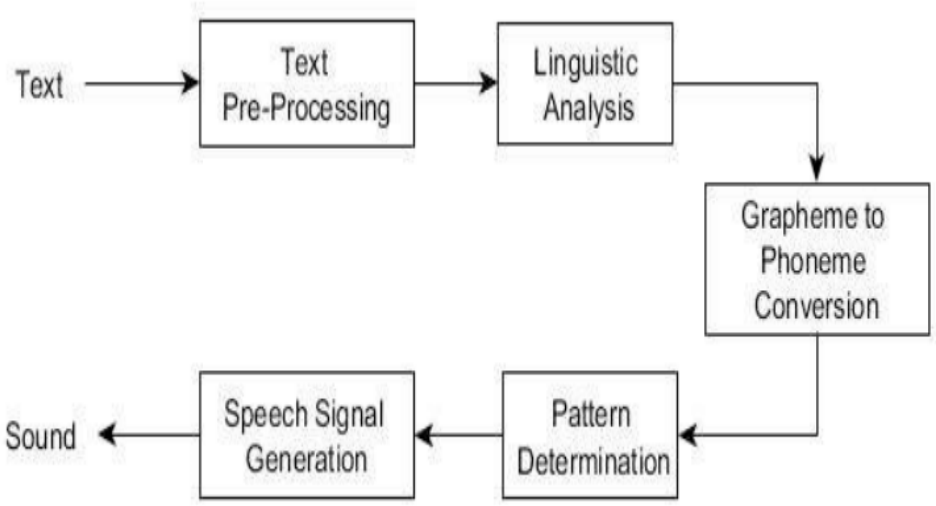


4.5. Natural Language Processing (NLP) for Generating Explanations

- **NLP for Medical Explanations**:
  - After making a prediction (e.g., pneumonia or normal), it is useful to generate a medical explanation that helps the user understand the result.
  - This is achieved by using pre-trained NLP models, like **GPT-2** or **BioGPT**, which can generate natural language explanations based on medical conditions.
- **Generating Explanations**:
  - Based on the predicted label (e.g., pneumonia), the system sends a prompt to the NLP model to generate a medical explanation.
  - For example, for pneumonia, the input could be: "Explain what pneumonia is and how it can be detected from a chest X-ray."
  - The NLP model generates a detailed response that explains the medical condition in layman's terms.
- **Customization**:
  - The generated text is tailored to sound like a doctor's explanation. It describes symptoms, diagnostic tests, and treatment options.

4.6. Text-to-Speech (gTTS) for Audio Output

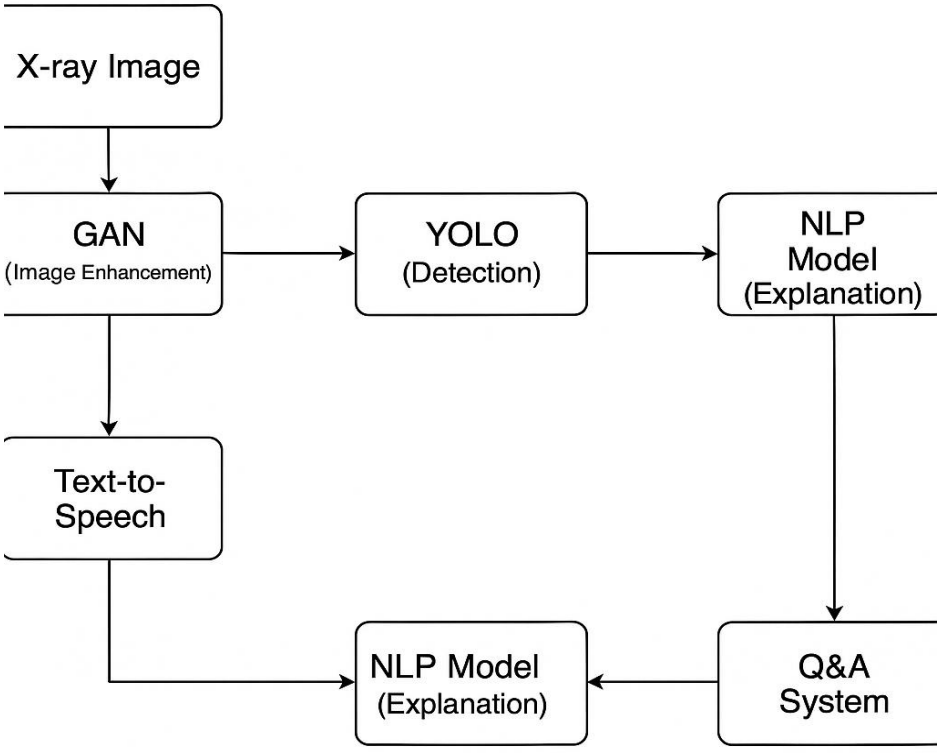
- **Text-to-Speech**:
  - After generating the medical explanation, you can use the **gTTS** (Google Text-to-Speech) library to convert the text explanation into an audio file.
  - This makes the explanation accessible even to patients who may have difficulty reading or prefer to listen to the explanation.
- **Audio Playback**:
  - The audio file is saved (e.g., as "explanation.mp3") and can be played back within a web interface or Jupyter notebook for instant audio feedback.



4.7. Q&A System for Interaction

- **User Interaction**:
  - The system can offer a Q&A feature, where users can ask questions about their diagnosis (e.g., “What causes pneumonia?” or “What treatments are available for pneumonia?”).
  - These questions are answered by querying the same NLP model, which generates professional responses based on the question and the previously predicted condition.
- **Functionality**:
  - A loop is set up where the user can type questions, and the NLP model responds with an explanation. If the question is unrelated to the prediction (e.g., general health questions), the system prompts the user to ask questions related to the X-ray diagnosis.

5.ARCHITECTURE



6.RESULTS

1.Image Enhancement with GAN
The use of GAN (Generative Adversarial Networks) successfully enhanced the quality of chest X-ray images. The low-resolution input images were improved, providing more clarity and detail, which contributed to better feature extraction and detection performance.

2.Multiclass Detection with YOLO
The YOLO (You Only Look Once) object detection model effectively detected multiple classes within the chest X-ray images. By leveraging the enhanced images from the GAN, the model's accuracy in detecting various anomalies, such as tumors or fractures, was significantly improved.

3.Speech Output using Gtts
The integration of gTTS (Google Text-to-Speech) enabled the conversion of recognized anomalies in the chest X-ray images into speech. This provided users with auditory feedback in both English and Tamil, making the system accessible to a wider audience, including non-technical users.

4.Question Answering using NLP
The implementation of a Natural Language Processing (NLP) model facilitated a simple and intuitive question-answering feature. Users could ask questions related to the chest X-ray analysis, and the system provided accurate responses, further enhancing the user experience.

7. Future Scope

1. **Enhanced Image Resolution and Data Diversity**
   Future improvements can include exploring more advanced image enhancement techniques, such as using pretrained super-resolution models for better results. Additionally, increasing the diversity of the dataset by incorporating more chest X-ray images from various sources could further improve the model’s robustness and accuracy.
2. **Integration with Real-Time Medical Systems**
   The system could be expanded to integrate with hospital diagnostic systems for real-time analysis of patient X-rays. This would provide immediate feedback to medical professionals, aiding in faster diagnosis and treatment.
3. **Multimodal Medical Assistance**
   Expanding the speech output to include more medical insights, such as explanations of detected conditions or guidance on the next steps in patient care, could further enhance the system's utility. Adding voice interaction features for user input could also improve accessibility and usability.
4. **Cross-Platform Deployment**
   The current implementation could be extended to work across multiple platforms (e.g., mobile apps, web interfaces), making it more accessible for healthcare

professionals in various settings. This would allow for remote diagnosis and telemedicine applications.

5. **Incorporating Other Medical Imaging Modalities**
   The future scope of this project can be broadened to include other medical imaging modalities such as CT scans or MRIs, providing a more comprehensive tool for medical diagnostics.

8. Conclusion

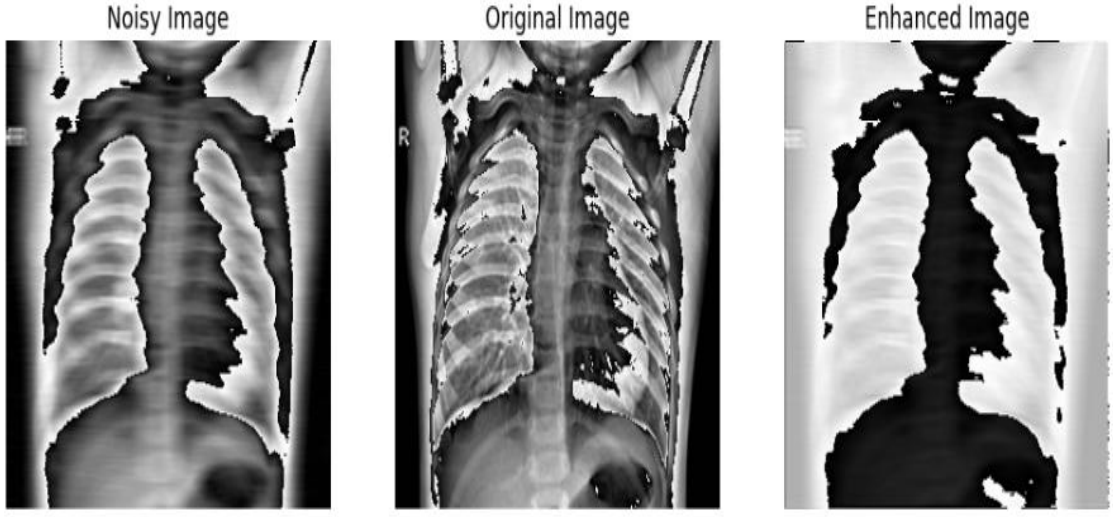
In conclusion, the integration of GAN-based image enhancement, YOLO object detection, and gTTS speech output has led to the development of a robust and accessible medical image analysis system. By enhancing chest X-ray images, the system significantly improved the accuracy of anomaly detection, enabling better identification of medical conditions. The use of YOLO for multiclass detection further strengthened the system's diagnostic capabilities, while the incorporation of gTTS allowed for bilingual auditory feedback, making the system user-friendly for a wider audience, including non-technical users.

Despite challenges such as data quality, model training, and real-time processing, the project has successfully demonstrated the potential of combining image enhancement, machine learning, and natural language processing to support healthcare professionals in diagnosing and understanding medical conditions. The future scope of the project includes expanding its capabilities by integrating with real-time medical systems, enhancing its multilingual support, and broadening the dataset to further improve accuracy.

9.APPENDICES

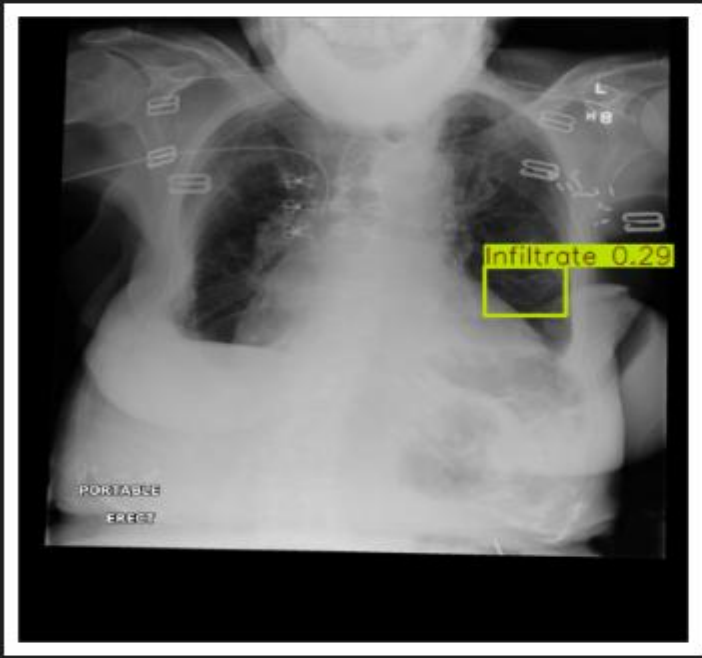
9.1 SCREENSHOTS

GAN:

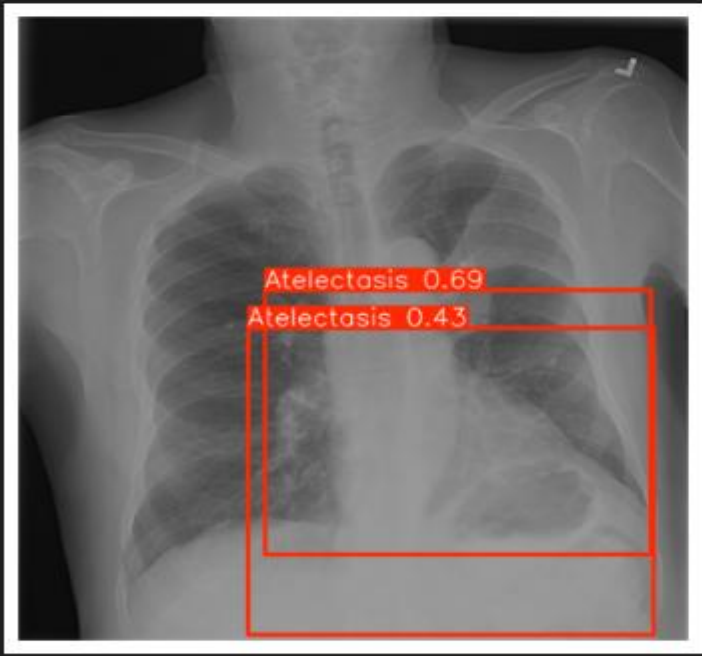


YOLO:

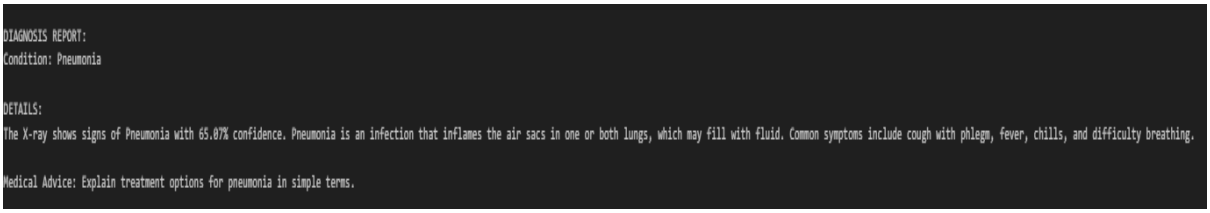
```
image 1/1 /content/chestXray8_512/val/images/00023325_019.png: 512x512 1 Infiltrate, 11.1ms
Speed: 1.5ms preprocess, 11.1ms inference, 2.7ms postprocess per image at shape (1, 3, 512, 512)
```

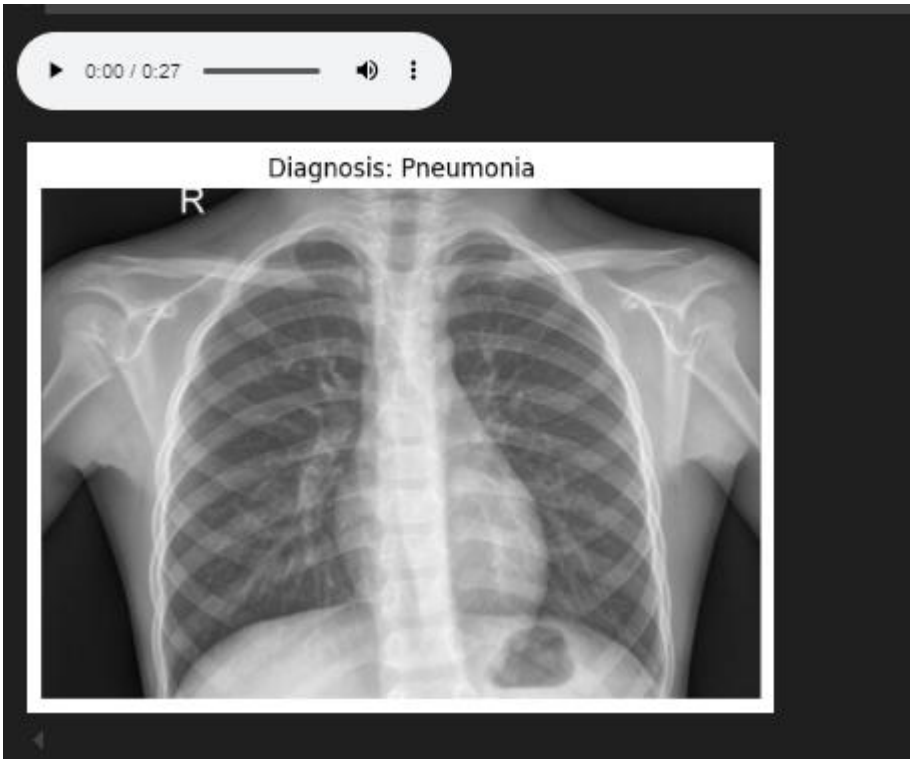



```
image 1/1 /content/chestXray8_512/val/images/00030674_000.png: 512x512 2 Atelectasiss, 12.2ms
Speed: 1.8ms preprocess, 12.2ms inference, 2.4ms postprocess per image at shape (1, 3, 512, 512)
```



NLP:





9.2 SOURCE CODE

GAN:

```
import os

import cv2

import numpy as np

from PIL import Image

import torch

import torch.nn as nn

import torch.optim as optim
```

```
from torch.utils.data import Dataset, DataLoader

from torchvision import transforms

from torchvision.utils import save_image

import matplotlib.pyplot as plt




class ChestXrayDataset(Dataset):

    def __init__(self, root_dir, mode="train", noise_type="gaussian+motion",
target_size=(256, 256)):

        self.root_dir = os.path.join(root_dir, mode)

        self.classes = ["NORMAL", "PNEUMONIA"]

        self.noise_type = noise_type

        self.target_size = target_size

        self.image_paths = []


        # Verify root directory exists

        if not os.path.exists(self.root_dir):

            raise FileNotFoundError(f"Directory not found: {self.root_dir}")


        # Load image paths

        for cls in self.classes:

            cls_dir = os.path.join(self.root_dir, cls)

            if os.path.exists(cls_dir):

                for img_name in os.listdir(cls_dir):

                    if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):

                        self.image_paths.append((os.path.join(cls_dir, img_name), cls))
```

```
    # Verify we found images
    if len(self.image_paths) == 0:
        raise RuntimeError(f"No images found in {self.root_dir}")

    print(f"Loaded {len(self.image_paths)} images from {mode} set")

    self.transform = transforms.Compose([
        transforms.Grayscale(),
        transforms.Resize(self.target_size),
        transforms.ToTensor(),
        transforms.Normalize(0.5, 0.5)
    ])

def __len__(self):
    """Returns the total number of images"""
    return len(self.image_paths)

def add_medical_noise(self, img_np):
    """Add realistic noise to chest X-rays"""
    # Gaussian noise (sensor noise)
    if "gaussian" in self.noise_type:
        row, col = img_np.shape
        mean = 0
        var = 0.1 * img_np.max()
```

```
            sigma = var ** 0.5
            gauss = np.random.normal(mean, sigma, (row, col))
            noisy = img_np + gauss

        # Motion blur (patient movement)
        if "motion" in self.noise_type:
            size = 15
            kernel = np.zeros((size, size))
            kernel[int((size-1)/2), :] = np.ones(size)
            kernel /= size
            noisy = cv2.filter2D(noisy, -1, kernel)

        return np.clip(noisy, 0, 255).astype(np.uint8)

    def __getitem__(self, idx):
        img_path, label = self.image_paths[idx]
        try:
            img = Image.open(img_path).convert('L')
            img = img.resize(self.target_size)
            img_np = np.array(img)

            # Add noise
            noisy_img = self.add_medical_noise(img_np)
            noisy_img = Image.fromarray(noisy_img)
```

```
            # Apply transforms
            clean = self.transform(img)
            noisy = self.transform(noisy_img)

            class_idx = self.classes.index(label)
            return noisy, clean, class_idx

        except Exception as e:
            print(f"Error loading {img_path}: {str(e)}")
            # Return zero tensors if image fails to load
            dummy = torch.zeros(1, *self.target_size)
            return dummy, dummy, 0

class UNetGenerator(nn.Module):
    def __init__(self, in_channels=1, out_channels=1):
        super().__init__()

        # Downsampling
        self.down1 = nn.Sequential(
            nn.Conv2d(in_channels, 64, 4, 2, 1),
            nn.LeakyReLU(0.2)
        )
        self.down2 = nn.Sequential(
            nn.Conv2d(64, 128, 4, 2, 1),
            nn.BatchNorm2d(128),
```

```
            nn.LeakyReLU(0.2)
        )
        self.down3 = nn.Sequential(
            nn.Conv2d(128, 256, 4, 2, 1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2)
        )

        # Upsampling
        self.up1 = nn.Sequential(
            nn.ConvTranspose2d(256, 128, 4, 2, 1),
            nn.BatchNorm2d(128),
            nn.ReLU()
        )
        self.up2 = nn.Sequential(
            nn.ConvTranspose2d(256, 64, 4, 2, 1),
            nn.BatchNorm2d(64),
            nn.ReLU()
        )
        self.up3 = nn.Sequential(
            nn.ConvTranspose2d(128, out_channels, 4, 2, 1),
            nn.Tanh()
        )

    def forward(self, x):
```

```
        # Encoder
        d1 = self.down1(x)  # 256x256 -> 128x128
        d2 = self.down2(d1)  # 128x128 -> 64x64
        d3 = self.down3(d2)  # 64x64 -> 32x32

        # Decoder with skip connections
        u1 = self.up1(d3)  # 32x32 -> 64x64
        u2 = self.up2(torch.cat([u1, d2], 1))  # 64x64 -> 128x128
        u3 = self.up3(torch.cat([u2, d1], 1))  # 128x128 -> 256x256

        return u3

class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Conv2d(1, 64, 4, 2, 1),
            nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, 4, 2, 1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),
            nn.Conv2d(128, 256, 4, 2, 1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),
            nn.Conv2d(256, 1, 4, 1, 1),
```

```
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)


def train_gan(train_loader, epochs=10, device="cuda"):
    G = UNetGenerator().to(device)
    D = Discriminator().to(device)

    # Loss functions
    criterion_GAN = nn.BCELoss()
    criterion_pixel = nn.L1Loss()
    lambda_pixel = 100

    # Optimizers
    opt_G = optim.Adam(G.parameters(), lr=0.0002, betas=(0.5, 0.999))
    opt_D = optim.Adam(D.parameters(), lr=0.0002, betas=(0.5, 0.999))

    for epoch in range(epochs):
        for i, (noisy, clean, _) in enumerate(train_loader):
            real = clean.to(device)
            noisy = noisy.to(device)
```

```
        # Get discriminator output size
        with torch.no_grad():
            out_size = D(real).shape[2:]

        # Train Discriminator
        D.zero_grad()
        fake = G(noisy)
        real_loss = criterion_GAN(D(real), torch.ones(real.size(0), 1, *out_size).to(device))
        fake_loss = criterion_GAN(D(fake.detach()), torch.zeros(fake.size(0), 1, *out_size).to(device))
        d_loss = (real_loss + fake_loss) / 2
        d_loss.backward()
        opt_D.step()

        # Train Generator
        G.zero_grad()
        g_loss = criterion_GAN(D(fake), torch.ones(fake.size(0), 1, *out_size).to(device))
        pixel_loss = criterion_pixel(fake, real)
        total_loss = g_loss + lambda_pixel * pixel_loss
        total_loss.backward()
        opt_G.step()

    print(f"Epoch {epoch}: D_loss={d_loss.item():.3f}, G_loss={total_loss.item():.3f}")

return G
```

```
def train_gan(train_loader, epochs=10, device="cuda"):
    G = UNetGenerator().to(device)
    D = Discriminator().to(device)

    criterion_GAN = nn.BCELoss()
    criterion_pixel = nn.L1Loss()
    lambda_pixel = 100

    opt_G = optim.Adam(G.parameters(), lr=0.0002, betas=(0.5, 0.999))
    opt_D = optim.Adam(D.parameters(), lr=0.0002, betas=(0.5, 0.999))

    for epoch in range(epochs):
        for i, (noisy, clean, _) in enumerate(train_loader):
            real = clean.to(device)
            noisy = noisy.to(device)

            # Train Discriminator
            D.zero_grad()
            fake = G(noisy)
            real_pred = D(real)
            fake_pred = D(fake.detach())

            # Create properly sized targets
            real_target = torch.ones_like(real_pred)
```

```
            fake_target = torch.zeros_like(fake_pred)

            real_loss = criterion_GAN(real_pred, real_target)
            fake_loss = criterion_GAN(fake_pred, fake_target)
            d_loss = (real_loss + fake_loss) / 2
            d_loss.backward()
            opt_D.step()

            # Train Generator
            G.zero_grad()
            fake_pred = D(fake)
            g_loss = criterion_GAN(fake_pred, real_target)
            pixel_loss = criterion_pixel(fake, real)
            total_loss = g_loss + lambda_pixel * pixel_loss
            total_loss.backward()
            opt_G.step()

        print(f"Epoch {epoch}: D_loss={d_loss.item():.3f}, G_loss={total_loss.item():.3f}")

    return G

class PneumoniaClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
```

```
            nn.Conv2d(1, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(128*32*32, 256),
            nn.ReLU(),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)


if __name__ == "__main__":
    # Initialize
    device = "cuda" if torch.cuda.is_available() else "cpu"
    print(f"Using device: {device}")
```

```
# 1. Prepare Data
train_dataset = ChestXrayDataset("chest_xray/chest_xray/chest_xray", mode="train")
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
# Test the dimensions
test_input = torch.randn(1, 1, 256, 256).to(device)
G = UNetGenerator().to(device)
D = Discriminator().to(device)

print("Generator output size:", G(test_input).shape)  # Should be [1, 1, 256, 256]
print("Discriminator output size:", D(test_input).shape)  # Should be [1, 1, 30, 30]
   # 2. Train GAN
print("Training GAN for image enhancement...")
generator = train_gan(train_loader, epochs=10, device=device)
torch.save(generator.state_dict(), "generator.pth")

# 3. Train Classifier on Enhanced Images
print("\nTraining Classifier...")
classifier = PneumoniaClassifier().to(device)
criterion = nn.BCELoss()
optimizer = optim.Adam(classifier.parameters(), lr=0.001)

for epoch in range(10):
    for noisy, _, labels in train_loader:
        noisy = noisy.to(device)
        labels = labels.float().unsqueeze(1).to(device)
```

```
            # Enhance images
            with torch.no_grad():
                enhanced = generator(noisy)

            # Classify
            optimizer.zero_grad()
            outputs = classifier(enhanced)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

        print(f"Epoch {epoch}: Loss={loss.item():.4f}")

    torch.save(classifier.state_dict(), "classifier.pth")
```

YOLO:

```
train_images_path = '/content/chestXray8_512/train/images'
val_images_path = '/content/chestXray8_512/val/images'

class_names = [
    "Atelectasis",
    "Cardiomegaly",
    "Effusion",
    "Infiltrate",
    "Nodule",
    "Mass",
    "Pneumonia",
```

```
    "Pneumothorax",
]

num_classes = len(class_names)
yaml_file_path = 'data.yaml'

with open(yaml_file_path, 'w') as yaml_file:
    yaml_file.write(yaml_content)
from ultralytics import YOLO
model = YOLO("yolov8s.pt")
model.train(data="/content/data.yaml", epochs=40, imgsz=512)
best='/content/runs/detect/train/weights/best.pt'
model=YOLO(best)
image_dir = "/content/chestXray8_512/val/images"

all_images = os.listdir(image_dir)
selected_images = all_images[:50]

for img_name in selected_images:
    img_path = os.path.join(image_dir, img_name)
    results = model.predict(img_path)
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    for result in results:
        plotted_img = result.plot()
        plt.figure(figsize=(8, 6))
        plt.imshow(plotted_img)
        plt.axis('off')
        plt.show()
```

NLP:

```
import torch
import torchvision.transforms as transforms
import torchxrayvision as xrv
from PIL import Image
import numpy as np
import cv2
import matplotlib.pyplot as plt
from gtts import gTTS
import IPython.display as ipy_display
from transformers import pipeline
import os


model = xrv.models.DenseNet(weights="densenet121-res224-all")
model.eval()

nlp_model = pipeline("text-generation", model="gpt2")


img_path = "/content/chestXray8_512/val/images/00000048_000.png"
assert os.path.exists(img_path), f"Image not found: {img_path}"

img = Image.open(img_path).convert("L")
img_np = np.array(img).astype(np.float32)
img_np = (img_np / 255.0) * 2048 - 1024
img_resized = cv2.resize(img_np, (224, 224))
img_tensor = torch.tensor(img_resized).unsqueeze(0).unsqueeze(0)


output = model(img_tensor)
```

```
pred_idx = output[0].argmax().item()
pred_label = model.pathologies[pred_idx]
confidence = torch.sigmoid(output[0])[pred_idx].item()



if pred_label.lower() == "pneumonia":
    prompt = "Explain what pneumonia is and how it can be detected from a chest X-ray."
elif pred_label.lower() == "normal":
    prompt = "Explain what a normal chest X-ray looks like and what it indicates about lung health."
else:
    prompt = f"Explain what {pred_label} is in a medical context related to chest X-rays."


explanation = nlp_model(prompt, max_length=200)[0]['generated_text']


doctor_response = f"As a medical professional, I can tell you that based on your chest X-ray, the condition is diagnosed as {pred_label}. "


if pred_label.lower() == "pneumonia":
    doctor_response += (
        "Pneumonia is a serious lung condition that requires immediate attention. "
        "The X-ray suggests inflammation or infection in the lungs, which could be bacterial, viral, or caused by other pathogens. "
        "It is critical to follow up with further diagnostic tests, such as blood work or sputum culture, and to begin a course of antibiotics or antiviral medications. "
        "Rest and proper hydration are also crucial for recovery."
    )
elif pred_label.lower() == "normal":
    doctor_response += (
        "The X-ray appears to show healthy lungs with no signs of infection or inflammation. "
        "Your lungs are well-aerated and functioning as expected. "
```

```
        "It is important to maintain a healthy lifestyle, avoid smoking, and engage in regular physical activity to keep your lungs in good condition."
    )
else:
    doctor_response += explanation



print("Prediction:", pred_label)
print("Confidence: {:.2f}".format(confidence))
print("Doctor's Explanation:\n", doctor_response)

tts = gTTS(text=doctor_response, lang='en')
tts.save("explanation.mp3")
ipy_display.display(ipy_display.Audio("explanation.mp3", autoplay=True))



plt.figure(figsize=(6, 6))
plt.imshow(img, cmap='gray')
plt.title(f"Prediction: {pred_label} (Confidence: {confidence:.2f})")
plt.axis("off")
plt.show()
ipy_display.display(plt.gcf())

def ask_questions():
    print("\nYou can now ask questions related to the prediction. Type 'no more questions' to stop.")
    while True:
        question = input("Ask a question: ")
        if question.lower() == "no more questions":
            print("Thank you for your questions. If you need further assistance, feel free to ask again!")
```

```
            break
        if any(kw in question.lower() for kw in ["pneumonia", "lungs", "chest x-ray",
pred_label.lower()]):
            answer_prompt = f"Provide a professional medical explanation for: {question}"
            answer = nlp_model(answer_prompt, max_length=150)[0]['generated_text']
            print("Answer:", answer)



            tts = gTTS(text=answer, lang='en')
            tts.save("answer.mp3")
            ipy_display.display(ipy_display.Audio("answer.mp3", autoplay=True))
        else:
            print("Please ask questions related to the diagnosis or lung health.")



ask_questions()
```