Clini Scan: Lung-Abnormality Detection on Chest X-rays using AI

P PRIYA KAVYA SUDHA

Table of Contents

8.Light denoising- Gaussian blur, Median filtering

9. Crop borders

10.Padding for uniform shape-for consistent height/width

PROJECT DESCRIPTION:

The project aims to develop an AI-powered system capable of automatically detecting and localizing lung abnormalities from chest X-ray images using deep learning techniques. Chest X-rays are one of the most commonly used diagnostic tools in healthcare, but interpreting them requires high expertise and can be time-consuming. By building an automated detection system, the project supports radiologists in identifying key pathological findings such as opacities, consolidations, fibrosis, nodules, and masses. The system can also be extended to classify disease conditions like pneumonia and tuberculosis, making it highly relevant for real-world clinical applications.

The model will be trained using the VinDr-CXR dataset, which contains high-quality chest X-ray images with detailed expert annotations. Before training, the images undergo preprocessing steps such as DICOM loading, contrast enhancement, resizing, and normalization to ensure consistent quality. Annotation files are converted into training-compatible formats like YOLO to enable efficient object detection.

A deep learning model—such as YOLOv8 or YOLOv9—is then trained to learn the patterns of abnormalities from the dataset. The system is evaluated using metrics like mAP, precision, recall, and F1-score to ensure high diagnostic accuracy. Once trained, the model can automatically detect abnormalities in new chest X-rays and display bounding boxes with confidence scores, making the results interpretable and clinically useful.

Finally, the system is optimized for deployment so that it can be integrated into hospital workflows, web interfaces, or desktop applications. With a user-friendly interface, healthcare professionals can upload an X-ray and instantly receive AI-generated insights, ultimately improving diagnostic efficiency and patient care.

dataset used:

VinBigData-chest-xray-abnormalities-detection

Environment setup in Kaggel

To begin working on lung abnormality detection in Kaggle, you first need to set up the environment properly so that all required libraries and data are available for processing. Kaggle provides a Jupyter Notebook interface with preinstalled machine learning libraries such as PyTorch, TensorFlow, NumPy, OpenCV, Matplotlib, Scikit-Learn, and pydicom, so no manual installation is required. The first step is to enable **Internet Off** mode (default) and attach the **VinDr-CXR dataset** or any required dataset from Kaggle. This is done by going to the “Add Data” option on the right side of the notebook and selecting the dataset like *vinbigdata-chest-xray-abnormalities-detection*. After attaching the dataset, it becomes accessible under the directory /kaggle/input/. You can verify the dataset by listing the files using Python commands such as os.listdir("/kaggle/input"). Next, import the essential libraries including NumPy, Pandas, OpenCV, Matplotlib, and pydicom to handle DICOM images. If you plan to train YOLO models, you can install ultralytics inside the notebook with pip install ultralytics, which Kaggle allows even without internet. Once the libraries are imported and the dataset path is confirmed, you are ready to perform preprocessing, annotation conversion, model training, and evaluation directly in the notebook. This setup ensures a smooth and consistent environment for developing the entire AI pipeline within Kaggle.

Convert DICOM->PNG

1. Importing Libraries

- pydicom → To load and read DICOM medical images.
- cv2 (OpenCV) → For image processing and pseudocoloring.
- numpy → For array and numerical operations.
- os → To access files inside the input folder.
- matplotlib.pyplot → To display images in the notebook.

2. Setting Input Folder

```
input_folder = "/kaggle/input/vinbigdata-chest-xray-abnormalities-detection/test"
```

3. Collecting DICOM Files

```
dicom_files = [f for f in os.listdir(input_folder) if f.endswith(".dicom")]
```

- Lists only files ending with .dicom.
- Filters out any non-DICOM files.

4. Looping Through the First 10 Images

```
for i, filename in enumerate(dicom_files[:10]):
```

- Processes only the first 10 images.
- enumerate gives index + filename.

5. Reading Each DICOM Image

```
dicom = pydicom.dcmread(path)
image = dicom.pixel_array
```

- Loads the DICOM file.
- Extracts the pixel data from the DICOM.

6. Normalizing Pixel Values (0–255)

```
image = image.astype(np.float32)
image = 255 * (image - np.min(image)) / (np.max(image) - np.min(image))
image = image.astype(np.uint8)
```

- Converts pixel range to standard 0–255.
- Essential for visualization.
- Converts image to uint8 format.

7.Applying Pseudocolor

```
pseudo_rgb = cv2.applyColorMap(image, cv2.COLORMAP_JET)
```

- Applies JET colormap.
- Highlights intensity differences in different colors.
- Makes abnormalities more visible.

8. Saving Output Image

```
output_path = f"/kaggle/working/pseudo_{i+1}.png"
cv2.imwrite(output_path, pseudo_rgb)
```

- Saves each processed image in Kaggle working directory.
- Names like pseudo_1.png, pseudo_2.png, etc.

9. Displaying Each Image

```
plt.imshow(cv2.cvtColor(pseudo_rgb, cv2.COLOR_BGR2RGB))
plt.title(f"Pseudocolor Image {i+1}")
plt.axis("off")
plt.show()
```

- Converts BGR → RGB for correct display.
- Shows each pseudocolored image in the notebook.

Convert to grayscale images

1. Extract Pixel Array

```
image = dicom.pixel_array
```

- Reads the raw pixel values from the DICOM file.
- These values are originally stored in grayscale but not in a fixed 0–255 range.

2. Convert to Float for Processing

```
image = image.astype(np.float32)
```

- Converts pixel values to float so mathematical operations can be applied safely.

3. Normalize Pixel Intensities (Min–Max Scaling)

```
image = 255 * (image - np.min(image)) / (np.max(image) - np.min(image))
```

- Ensures all pixel values fall between **0 and 255**.
- Helps standardize brightness and contrast across different DICOM files.

4. Convert Back to 8-bit Grayscale

```
gray_image = image.astype(np.uint8)
```

- Converts normalized values into **standard grayscale image format** (0–255 range).

5. Save Grayscale Image

```
cv2.imwrite(output_path, gray_image)
```

- Stores the normalized grayscale image as a .png file.

6. Display Grayscale Image

```
plt.imshow(gray_image, cmap="gray")
```

- Displays the image using a gray colormap so it appears properly in grayscale.

Resize

1. Define Resize Dimensions

```
resize_height = 256
resize_width = 256
```

Sets the target output size for all images.

Ensures every X-ray image becomes 256 × 256 pixels, which is required for deep learning models that expect uniform input dimensions.

2. Resize the Grayscale Image

resized_image = cv2.resize(gray_image, (resize_width, resize_height),

interpolation=cv2.INTER_AREA)

Converts the original DICOM image into the new fixed size.

cv2.INTER_AREA interpolation is used:

Best for shrinking images.

Reduces noise.

Preserves important medical structures.

3. Save the Resized Image

cv2.imwrite(output_path, resized_image)

Stores the resized version in the working directory for later processing.

4. Display the Resized Image

plt.imshow(resized_image, cmap="gray")

Visualizes the resized output to confirm correctness.

Intensity normalization-minmax or z-score

Min–Max Normalization (Function Explanation)

Code:

minmax_norm = (image - img_min) / (img_max - img_min + 1e-8)

minmax_norm = (minmax_norm * 255).astype(np.uint8)

Explanation:

This method scales pixel values into a fixed range, usually 0 to 1.

img_min = smallest pixel value

img_max = largest pixel value

The formula:

$$\text{normalized} = \frac{\boldsymbol{pixel - min}}{\boldsymbol{max - min}}$$

After scaling to 0–1, the values are multiplied by 255 to convert into grayscale image format.

Purpose:

Makes all images have a uniform brightness range.

Useful when original DICOM images have different exposure levels.

Z-Score Normalization (Function Explanation)

Code:

zscore_norm = (image - img_mean) / (img_std + 1e-8)

Explanation:

Z-Score normalization standardizes pixel values.

img_mean = average brightness

img_std = standard deviation of brightness

The formula:

$$\boldsymbol{z} = \frac{\boldsymbol{pixel - mean}}{\boldsymbol{std}}$$

This converts the image so that:

Mean becomes 0

Standard deviation becomes 1

After this, the values are again scaled to 0–255 for display.

Purpose:

Removes lighting variations.

Makes pixel distribution consistent across all images.

Helpful for ML models that prefer normalized distributions.

Light contrast enhancement-CLAHE Method

1. Divides the image into small tiles

The image is split into small 8×8 regions (tileGridSize=(8,8) in your code).

Contrast enhancement is applied separately for each region.

✓ 2. Limits contrast to avoid noise

The parameter clipLimit=2.0 ensures that:

Very bright or dark pixels are not overly enhanced.

Noise is not amplified too much.

This is important for medical images where over-contrast can hide important details.

✓ 3. Applies histogram equalization within each tile

Each tile is enhanced so that hidden structures such as:

lung boundaries

rib details

soft tissues

become more visible.

✓ 4. Smooths the tiles together

After enhancing each tile, CLAHE merges them smoothly to avoid blocky artifacts.

Code Line Explained

clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

clahe_image = clahe.apply(image)

✓ createCLAHE() → Creates the CLAHE processor

✓ apply(image) → Enhances light + contrast of your grayscale DICOM image

Light denoising- Gaussian blur, Median filtering

1. Gaussian Blur (cv2.GaussianBlur)

Gaussian Blur is a linear smoothing filter that reduces image noise by convolving the image with a Gaussian kernel. The kernel values follow a Gaussian (bell-shaped) distribution, giving more weight to pixels near the center and less weight to pixels farther away.

Technical behavior

It performs weighted averaging, where weights come from the Gaussian function:

$$\boldsymbol{G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}}$$

(5×5) kernel defines the neighborhood size.

σ = 1 controls the spread of Gaussian distribution.

Because of weighted averaging, the filter smooths image intensity variations gradually.

. Crop borders

A typical cropping function follows these steps:

1. Find Non-Black Pixels

It scans the image to identify pixels that have intensity values greater than 0 (or above a low threshold).
These represent actual X-ray content.

2. Determine Bounding Box

Once useful pixels are detected, it calculates:

- minimum row index
- maximum row index
- minimum column index
- maximum column index

These four values define the **tightest bounding rectangle** around the important region.

3. Crop the Image

The function extracts the region inside this bounding box:

cropped = image[min_row:max_row, min_col:max_col]

This removes:

- Empty black borders
- Machine-generated padding
- Irrelevant surrounding area

Padding for uniform shape-for consistent height/width

After cropping, every image has **different height and width**.

But deep learning models require **all images to have the same size**.
So we **pad** the smaller images with black pixels (0 value) to make them equal to the **largest height and width**.

1 **Find current image height & width**

```
h, w = img.shape
```

2 **Compute padding size for top and bottom**

- If image height is smaller than target height:
- pad_top = (target_h - h) // 2
- pad_bottom = target_h - h - pad_top

3 **Compute padding size for left and right**

```
pad_left = (target_w - w) // 2

pad_right = target_w - w - pad_left
```

4 **Add black padding**

```
padded = cv2.copyMakeBorder(
   img,
   pad_top, pad_bottom,
   pad_left, pad_right,
   cv2.BORDER_CONSTANT,
   value=0
)
```