JSS MAHAVIDYAPEETHA

JSS SCIENCE AND TECHNOLOGY UNIVERSITY

JSS TECHNICAL INSTITUTIONS CAMPUS

MYSURU – 570006 KARNATAKA, INDIA

Image Processing and Pattern Recognition - 22CB673

DICOM Preprocessing Pipeline for Mammography Imaging

A Project Report submitted to

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

JSS Science and Technology University

in partial fulfillment of the curriculum requirements for the Internal

Assessment- Event 2 as part of the undergraduate program

Bachelor of Engineering

In

INFORMATION SCIENCE AND ENGINEERING

By

Kiran Seenivasan 01JST22UCB019

(Name) (USN)

(Name) (USN)

DEPARTMENT OF INTORMATION SCIENCE AND ENGINEERING

JSS Science and Technology University, Mysuru

2025

Abstract

This project presents a robust and scalable image preprocessing pipeline designed specifically for breast cancer DICOM images, such as those used in the RSNA Breast Cancer Detection dataset. The objective was to enhance medical image quality and consistency prior to downstream tasks like classification or diagnosis. The preprocessing pipeline includes critical steps such as VOI LUT-based intensity transformation for windowing, inversion for MONOCHROME1 images, adaptive contrast enhancement using CLAHE and histogram equalization, and intelligent cropping based on pixel distribution to isolate the region of interest. Additionally, the pipeline supports aspect-ratio-preserving padding and resizing to standardized input dimensions. Outcomes include significantly cleaner, contrast-enhanced, and size-normalized mammogram images, making the data more suitable for machine learning models and improving interpretability for clinicians. This modular pipeline can be applied to large-scale datasets, deployed in production environments, or integrated into annotation and visualization tools.

1. Introduction

1.1 About the Project Work

This project focuses on the preprocessing of mammographic DICOM images for use in breast cancer detection workflows. Before such images can be utilized in diagnostic tools or machine learning models, they require extensive cleaning, enhancement, and normalization to ensure consistency and clarity. The work involves applying VOI LUT windowing, handling different photometric interpretations, enhancing contrast, cropping irrelevant regions, and resizing the images to standard dimensions. This preprocessing pipeline is modular and efficient, and it produces uniform, high-quality outputs suitable for real-world clinical or research applications.

1.2 Motivation

Medical DICOM images, particularly mammograms, often vary greatly in resolution, contrast, orientation, and formatting. These inconsistencies present major challenges for automated diagnostic systems. A reliable preprocessing pipeline is essential to extract only the relevant anatomical region (typically the breast tissue), remove unnecessary borders and labels, enhance important structures, and format the images consistently. The motivation behind this work was to build a reusable, automated tool that addresses these challenges systematically and prepares the dataset for accurate analysis or AI-based classification of Breast Cancer Detection.

1.2 Challenges

1. **Inconsistent Image Formats:** DICOM files come from different imaging machines and may vary in resolution, bit-depth, and layout.
2. **Photometric Interpretation Variance:** Some images use MONOCHROME1, where brightness levels are inverted, requiring additional handling.
3. **Presence of Noise and Borders:** Images often contain black borders, labels, and irrelevant annotations that interfere with model training.
4. **Maintaining Aspect Ratio:** Resizing must be done carefully to preserve anatomical correctness and prevent distortion.
5. **Contrast and Windowing:** Raw pixel values require conversion using DICOM windowing functions (VOI LUT) to become visually meaningful.

1.4 Problem Definition

To build an automated preprocessing system for mammographic DICOM images that:

- Enhances image quality through proper windowing and contrast adjustment.
- Crops out irrelevant regions while preserving the anatomical region of interest.
- Normalizes the image to a fixed resolution and format.
- Prepares the data in a consistent structure for downstream medical or AI applications.

1.5 Aim and Objectives

Aim:

To develop a robust and scalable image preprocessing pipeline tailored for mammographic DICOM images used in breast cancer detection.

Objectives:

- Implement VOI LUT-based pixel intensity transformation.
- Handle photometric variations like MONOCHROME1 inversion.
- Automatically crop and align breast tissue using image statistics.
- Normalize and resize images while preserving important spatial features.
- Enable optional contrast enhancement (CLAHE, histogram equalization).
- Output uniformly formatted, high-quality images ready for analysis or model inference.

2. Implementation

2.1 Algorithm / Model / Method Explanation

The system processes and prepares DICOM (Digital Imaging and Communications in Medicine) medical images for potential diagnostic or ML-based tasks. The core methods involved are:

2.11 VOI LUT (Value of Interest Look-Up Table)

Purpose: Enhances image contrast to highlight the most clinically relevant pixel intensities.
How it works:
DICOM files often include WindowCenter and WindowWidth tags, which define the range of pixel intensities that should be visible.
VOI LUT maps raw pixel values to this visible range using either:
Linear Transformation (most common): Scales and clamps pixel intensities to a visible range.
Sigmoid Transformation (if specified): Applies a sigmoid curve to emphasize mid-range values and compress extremes, useful for subtle contrasts.
Calculations:
For linear: $Values\ below\ center\ -\ width/2\ are\ mapped\ to\ black, above\ center\ +\ width/2\ to\ white, and\ in\ between\ linearly\ scaled.$
For sigmoid: $pixel\ =\ range\ /\ (1\ +\ exp(-4\ *\ (pixel\ -\ center)\ /\ width))$
Why it's important: DICOMs often store values in a wide range (e.g., 12–16 bit), and VOI LUT ensures only the most meaningful ranges are displayed for human and machine interpretation.

2.12 Normalization

Purpose: Scales image pixel intensities to a consistent range for processing.

Steps:

- **Min-max normalization**:

$$\text{normalized\_pixel} = \frac{\text{pixel} - \min(\text{pixel})}{\max(\text{pixel}) - \min(\text{pixel})}$$

  Now the pixel values lie in [0, 1].

- **Scaling to 0–255**:
  After normalization, pixels are multiplied by 255.
- **Type conversion**:
  Image is cast to `uint8` to make it compatible with:

  - OpenCV functions (like resizing and CLAHE)
  - Display tools like `matplotlib.imshow`
  - Memory-efficient representation

Why:

- Many ML frameworks (and OpenCV) expect `uint8` inputs.
- Standardized pixel ranges allow for model generalization across datasets.

2.13 Auto Flip (Horizontal)

Purpose: Ensures consistent left-right orientation across images.
Logic:
It checks whether the **right 10%** of image width is **brighter** (higher sum of pixel values) than the **left 10%**.

$$\text{Flip if: } \sum \text{pixels in right } 10\% > \sum \text{pixels in left } 10\%$$

The image is then flipped horizontally using:

$$image = np.flip(image, axis = 1)$$

Prevents inconsistent orientations which may affect model predictions.

2.14 Cropping

Purpose: Removes black borders or irrelevant regions around the main content.
How it works:

- **X-axis Cropping**:

Computes $sum * std$ across columns to detect intensity variation.
Smooths this signal with a 1% convolution kernel.
Finds the first column (after max activity) where this value drops below 5% of the peak → crops everything after this.

- **Y-axis Cropping**:

Similar method across rows (top and bottom).
Threshold is 10% of the max $sum * std$.
Returns offset_top, offset_bottom to slice the vertical section.

$$\text{Heuristic Score} = \text{smooth}(\sum \text{pixels}) \times \text{smooth}(\text{std deviation})$$

Why:
Helps remove noisy blank regions which can distort model learning.
Retains only informative central content.

2.15 Padding & Resizing

Purpose: Standardizes all images to a fixed shape: **768 × 1344** (Width × Height) without distorting content.
Steps:
3 After cropping, checks the aspect ratio of the image.
4 Adds **padding** (black pixels) to either:
5 Width (if image is too tall),
6 Height (if image is too wide).
7 Resizes the padded image to the final size using OpenCV:

$$\text{If } r > r_t: \quad \text{Pad width} \Rightarrow w' = h/r_t$$
$$\text{Else:} \quad \text{Pad height} \Rightarrow h' = w \cdot r_t$$

2.16 Contrast Enhancements

`CLAHE` (Contrast Limited Adaptive Histogram Equalization) or `Histogram Equalization` can be applied for sharper details.

$$\text{CLAHE}(p) = \text{cv2.createCLAHE().apply}(p)$$
$$\text{HistEq}(p) = \text{cv2.equalizeHist}(p)$$

2.2 Tools / Technologies / Platforms Used

Category	Tool / Technology	Purpose
Programming Language	Python 3.x	Core language for implementation of all image preprocessing operations.
Medical Image Handling	pydicom	To read and interpret DICOM files including metadata (e.g., WindowWidth, WindowCenter, VOILUTFunction).
Image Processing	OpenCV (cv2)	Applied for CLAHE, histogram equalization, resizing, flipping, and padding operations.
Numerical Computation	NumPy	Used for efficient matrix and vector computations, normalization, padding, and intensity operations.
Visualization	Matplotlib	Used to display image transformations step by step in debug mode.
Environment Management	.env + python-dotenv	Environment variable management (e.g., data paths, flags) for flexible deployment.
Development Environment	Google Colab	Easy testing and rapid prototyping of image preprocessing pipelines.
Image Format Handling	DICOM (Digital Imaging and Communications in Medicine)	Medical standard for handling, storing, printing, and transmitting information in medical imaging.

2.3 Steps Followed / Execution Details

Below is a detailed step-by-step execution flow of how each medical image is processed in the pipeline:

Step 1: Read the DICOM File

$dicom\ =\ pydicom.dcmread(file\_path)$
$image = dicom.pixel_array$

```
def make_prediction():
    #path = os.path.join(BASE_DIR,file_name)
    #path ='/Users/kiranseenivasan/Documents/Cancer Project/AI--CANCER-PROJECT-
py/uploads/0a395ff2-d281-4927-ae5f-d52407298ccf.dcm'
    path = "/content/68070693.dcm"
    print("predict path is" + path)
    final_img = process(path, crop_image=True, size=(TARGET_WIDTH,
TARGET_HEIGHT), debug=False, save=False)
    final_img = np.expand_dims(final_img, axis=-1)  # Add channel dimension
    final_img = np.expand_dims(final_img, axis=0)
    print("Processed image shape:", final_img.shape)

    # Show the image using matplotlib
    plt.imshow(final_img[0, :, :, 0], cmap='gray')  # Extract single image and
channel
    plt.title("Processed Image")
    plt.axis('off')
    plt.show()
```

- Loads the DICOM file from the provided path.
- Extracts the raw pixel array for processing.
- Metadata (like WindowWidth, WindowCenter, PhotometricInterpretation) is also extracted.

Step 2: Apply VOI LUT Transformation

```
if voi_lut_function == 'SIGMOID':
  image = sigmoid transformation
else:
  image = linear scaling transformation
```

```
def voi_lut(image, dicom):
    # Additional Checks
    if 'WindowWidth' not in dicom.getPixelDataInfo() or 'WindowWidth' not in
dicom.getPixelDataInfo():
        return image

    # Load only the variables we need
    center = dicom['WindowCenter']
    width = dicom['WindowWidth']
    bits_stored = dicom['BitsStored']
    voi_lut_function = dicom['VOILUTFunction']

    # For sigmoid it's a list, otherwise a single value
    if isinstance(center, list):
        center = center[0]
    if isinstance(width, list):
        width = width[0]
    # Set y_min, max & range
    y_min = 0
    y_max = float(2**bits_stored - 1)
    y_range = y_max

    # Function with default LINEAR (so for Nan, it will use linear)
    if voi_lut_function == 'SIGMOID':
        image = y_range / (1 + np.exp(-4 * (image - center) / width)) + y_min
    else:
        # Checks width for < 1 (in our case not necessary, always >= 750)
        center -= 0.5
        width -= 1

        below = image <= (center - width / 2)
        above = image > (center + width / 2)
        between = np.logical_and(~below, ~above)

        image[below] = y_min
        image[above] = y_max
        if between.any():
            image[between] = (
                ((image[between] - center) / width + 0.5) * y_range + y_min
            )

    return image
```

Step 3: Invert Pixel Values (if MONOCHROME1)

$if\ dicom.PhotometricInterpretation\ ==\ 'MONOCHROME1':$

$image\ =\ np.max(image)\ -\ image$

- In medical imaging, MONOCHROME1 means that **higher pixel values represent darker areas**, so the image is inverted to match MONOCHROME2 format.

Step 4: Normalize Pixel Intensities

$image\ =\ (image\ -\ image.min())\ /\ (image.max()\ -\ image.min())$
$image\ =\ (image\ *\ 255).astype(np.uint8)$

- Scales the pixel values from their original range to **[0, 1]**, then to **[0, 255]**.
- Converted to uint8 for compatibility with OpenCV functions like resize, CLAHE, etc.

Step 5: Auto-Flip Horizontally (if needed)

$if\ image[:, right\_10\%].sum()\ >\ image[:, left\_10\%].sum():$

$image = np.flip(image, axis = 1)$

Step 6: Cropping

$image\ =\ crop(image)$

```
def crop(image, size=None, debug=False):
    # Image dimensions
    H, W = image.shape
    # Compute x/bottom/top offsets
    x_offset = get_x_offset(image, debug=debug)
    offset_bottom, offset_top = get_y_offsets(image[:,:x_offset], debug=debug)
    # Crop Height and Width
    h_crop = offset_top - offset_bottom
    w_crop = x_offset

    # Pad crop offsets to target aspect ratio
    if size is not None:
        # Height too large, pad x offset
        if (h_crop / w_crop) > TARGET_HEIGHT_WIDTH_RATIO:
            x_offset += int(h_crop / TARGET_HEIGHT_WIDTH_RATIO - w_crop)
        else:
            # Height too small, pad bottom/top offsets
            offset_bottom -= int(0.50 * (w_crop * TARGET_HEIGHT_WIDTH_RATIO - 
h_crop))
            offset_bottom_correction = max(0, -offset_bottom)
            offset_bottom += offset_bottom_correction

            offset_top += int(0.50 * (w_crop * TARGET_HEIGHT_WIDTH_RATIO - 
h_crop))
            offset_top += offset_bottom_correction

    # Crop Image
    image = image[offset_bottom:offset_top:,:x_offset]

    return image
```

```
# X Crop offset based on first column with sum below 5% of maximum column 
sums*std
def get_x_offset(image, max_col_sum_ratio_threshold=0.05, debug=None):
    # Image Dimensions
    H, W = image.shape
    # Percentual margin added to offset
    margin = int(image.shape[1] * 0.00)
    # Threshold values based on smoothed sum x std to capture varying intensity 
columns
```

```
    vv = smooth(image.sum(axis=0).squeeze()) *
smooth(image.std(axis=0).squeeze())
    # Find maximum sum in first 75% of columns
    vv_argmax = vv[:int(image.shape[1] * 0.75)].argmax()
    # Threshold value
    vv_threshold = vv.max() * max_col_sum_ratio_threshold

    # Find first column after maximum column below threshold value
    for offset, v in enumerate(vv):
        # Start searching from vv_argmax
        if offset < vv_argmax:
            continue

        # Column below threshold value found
        if v < vv_threshold:
            offset = min(W, offset + margin)
            break

    if isinstance(debug, np.ndarray):
        #debug[1].imshow(image)
        debug[1].set_title('X Offset')
        vv_scale = H / vv.max() * 0.90
        # Values
        debug[1].plot(H - vv * vv_scale , c='red', label='vv')
        # Threshold
        debug[1].hlines(H - vv_threshold * vv_scale, 0, W -1, colors='orange',
label='threshold')
        # Max Value
        debug[1].scatter(vv_argmax, H - vv[vv_argmax] * vv_scale, c='blue',
s=100, label='Max', zorder=np.PINF)
        # First Column Below Threshold
        debug[1].scatter(offset, H - vv[offset] * vv_scale, c='purple', s=100,
label='Offset', zorder=np.PINF)
        debug[1].set_ylim(H, 0)
        debug[1].legend()
        debug[1].axis('off')

    return offset

def get_y_offsets(image, max_row_sum_ratio_threshold=0.10, debug=None):
    # Image Dimensions
    H, W = image.shape
    # Margin to add to offsets
    margin = 0
    # Threshold values based on smoothed sum x std to capture varying intensity
columns
    vv = smooth(image.sum(axis=1).squeeze()) *
smooth(image.std(axis=1).squeeze())
    # Find maximum sum * std row in inter quartile rows
    vv_argmax = int(image.shape[0] * 0.25) + vv[int(image.shape[0] *
0.25):int(image.shape[0] * 0.75)].argmax()
    # Threshold value
    vv_threshold = vv.max() * max_row_sum_ratio_threshold
    # Default crop offsets
    offset_bottom = 0
    offset_top = H

    # Bottom offset, search from argmax to bottom
    for offset in reversed(range(0, vv_argmax)):
        v = vv[offset]
        if v < vv_threshold:
            offset_bottom = offset
            break

    # Top offset, search from argmax to top
    for offset in range(vv_argmax, H):
        v = vv[offset]
        if v < vv_threshold:
            offset_top = offset
            break

    return max(0, offset_bottom - margin), min(image.shape[0], offset_top +
margin)
```

- Uses a **content-aware heuristic** based on the **product of sum and standard deviation** along rows and columns.
- Removes noisy or blank borders to focus on the medically relevant part of the image.

Step 7: Aspect Ratio Padding and Resizing

$image = np.pad(\ldots)$ # *if needed*
$image = cv2.resize(image, (768, 1344))$

```
h, w = image.shape
    if debug:
        print("STEP 8: Padding to maintain aspect ratio")
    if (h / w) > TARGET_HEIGHT_WIDTH_RATIO:
        pad = int(h / TARGET_HEIGHT_WIDTH_RATIO - w)
        image = np.pad(image, ((0,0),(0,pad)), constant_values=0)
        if debug:
            print(f"       Padded width by {pad} pixels")
    else:
        pad = int(0.50 * (w * TARGET_HEIGHT_WIDTH_RATIO - h))
        image = np.pad(image, ((pad,pad),(0,0)), constant_values=0)
        if debug:
            print(f"       Padded height by {2*pad} pixels")
    if debug:
        show_step(image, "After Padding")
```

- Padding is applied to maintain the **target aspect ratio** of 1344:768 before resizing.
- Ensures input dimensions are uniform for further model or diagnostic tasks.

Step 8: Optional Contrast Enhancement (CLAHE or Histogram Equalization)

```
if apply_clahe:
  $image = CLAHE.apply(image)$
elif apply_eq_hist:
  $image = cv2.equalizeHist(image)$
```

- **CLAHE (Contrast Limited Adaptive Histogram Equalization)** is used to improve local contrast in soft tissues.
- **Histogram Equalization** is used to redistribute pixel intensity for global contrast improvement.

Step 9: Return or Visualize the Final Image

3. Results and Discussions

3.1 Key Findings or Learning

During the implementation of the DICOM medical image preprocessing pipeline, several important technical and practical learnings were gathered:

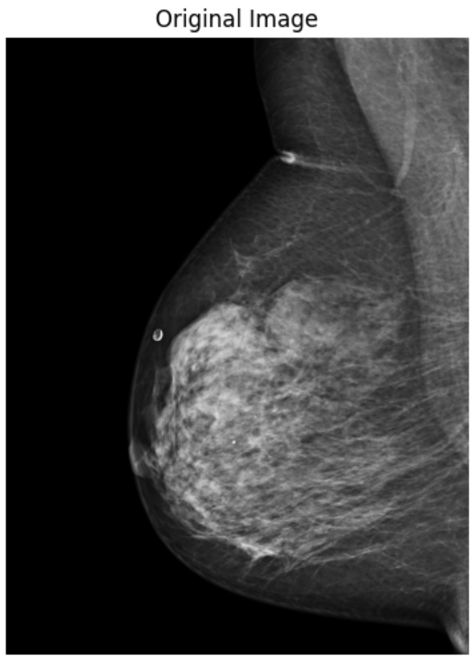
- **DICOM Handling Complexity**: Medical DICOM images come with varied metadata formats (e.g., list vs. scalar values for `WindowCenter`), requiring careful parsing for robust VOI LUT application.
- **VOI LUT Impact**: Applying the VOI LUT significantly enhanced the visibility of medically relevant structures by adjusting pixel intensities based on DICOM window settings. It was evident that VOI LUT must be handled before normalization to preserve contrast fidelity.
- **Normalization Importance**: Normalizing pixel values between [0, 1], then scaling to [0, 255], was crucial for standardizing the image format for downstream processing (especially for OpenCV compatibility and visualization).
- **Orientation Handling**: The horizontal flip check using summed pixel intensities of the left vs. right 10% revealed that inconsistent orientation is common in DICOM datasets. The auto-flip step ensures spatial consistency across the dataset.
- **Dynamic Cropping Strategy**: The cropping mechanism using `sum × std` heuristics effectively removed empty margins and focused the image on regions of interest without needing manual annotation.
- **Resizing and Padding**: Automatically padding before resizing preserved the image aspect ratio while ensuring consistent input dimensions for potential ML models (e.g., 768×1344).
- **Debug Visualization**: Using `matplotlib` to visualize each step (original, VOI LUT applied, cropped, padded/resized, CLAHE/hist equalized) helped in understanding how each transformation modifies the image.

3.2 Outcomes Achieved

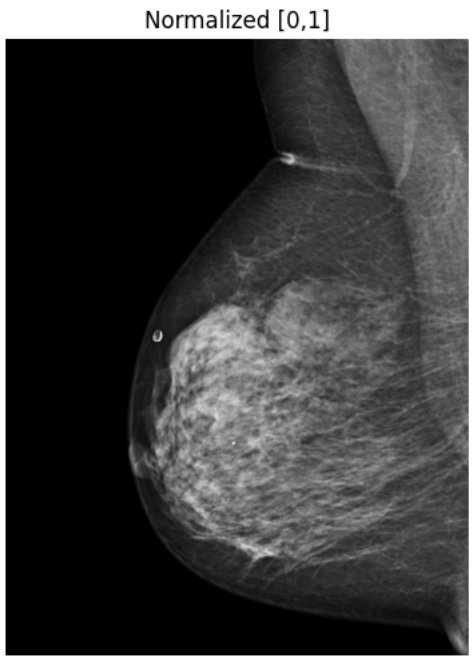
Goal	Achieved Outcome
Read and visualize DICOM images	Successfully read .dcm files and displayed using matplotlib.
VOI LUT transformation	Correct application of WindowCenter, WindowWidth, and VOILUTFunction. Enhanced region visibility.
Normalize & prepare image data	Transformed DICOM data to consistent [0, 255] uint8 format.
Handle flipped orientations	Automatically detected and corrected horizontal misalignments.
Remove irrelevant borders	Cropped images based on intensity variation along axes.
Resize while preserving aspect ratio	Resized all images to 768×1344 with proper padding.
Optional contrast enhancement	Successfully tested CLAHE and Histogram Equalization.
Flexible, modular pipeline	All preprocessing steps integrated with debug visualization and toggles (e.g., apply_clahe, debug).

3.3 Screenshots / Sample Outputs

3.31 Original DICOM Image



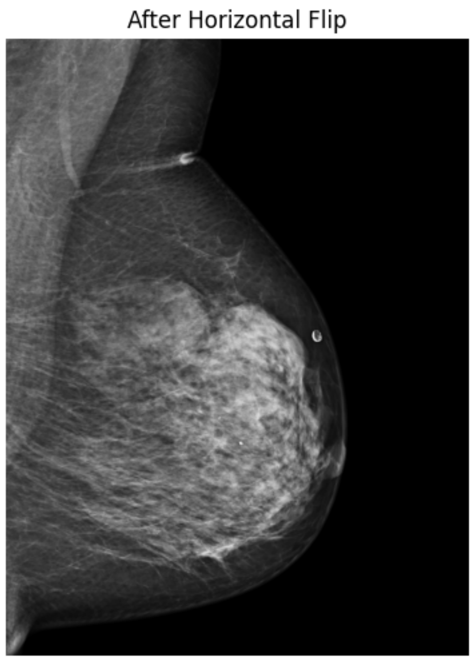
3.32 Normalized Image + Normalized Matrix



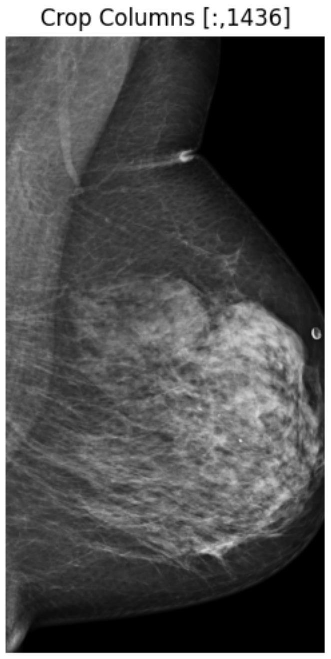
```
Original IMG Matrix:[[   0    0    0 ... 1626 1618 1448]
 [   0    0    0 ... 1788 1805 1634]
 [   0    0    0 ... 1743 1707 1678]
 ...
 [   0    0    0 ... 1448 1456 1414]
 [   0    0    0 ... 1388 1358 1336]
 [   0    0    0 ... 1377 1365 1377]]
```

```
 New Img Matrix[[0.         0.         0.         ... 0.39745783 0.39550232 0.35394769]
 [0.         0.         0.         ... 0.43705695 0.44121242 0.39941335]
 [0.         0.         0.         ... 0.4260572  0.41725739 0.41016866]
 ...
 [0.         0.         0.         ... 0.35394769 0.3559032  0.34563676]
 [0.         0.         0.         ... 0.33928135 0.33194818 0.32657052]
 [0.         0.         0.         ... 0.33659252 0.33365925 0.33659252]]
```

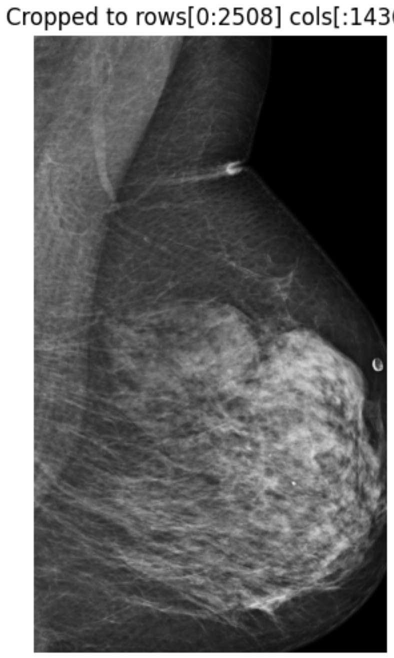
3.33 Flipped Horizontally (Right Side Brighter)



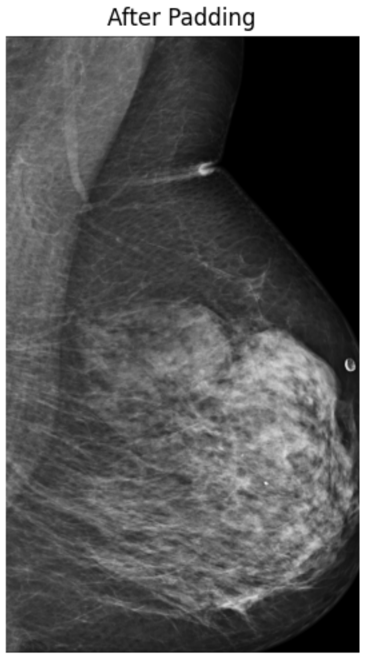
3.34 Compute X_offset and crop columns (X_offset = 1436)



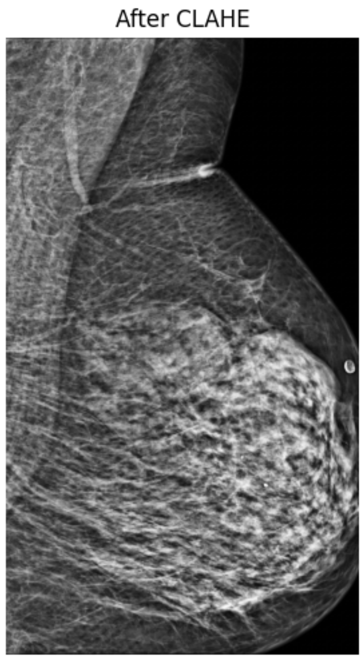
3.35 Compute Y_offset and crop rows (Y_offset = 2508)

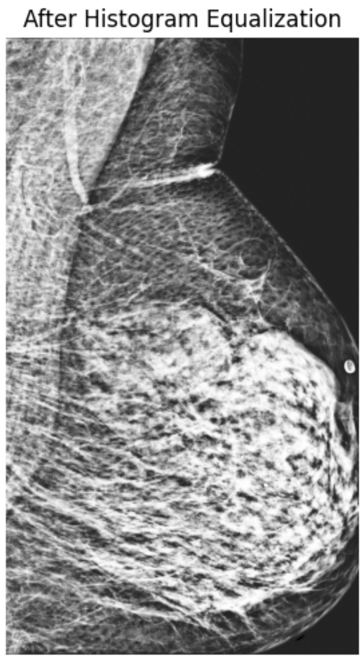


3.36 Added Padding to Maintain Aspect Ratio(4 pixels) and Resizing to (768, 1344)

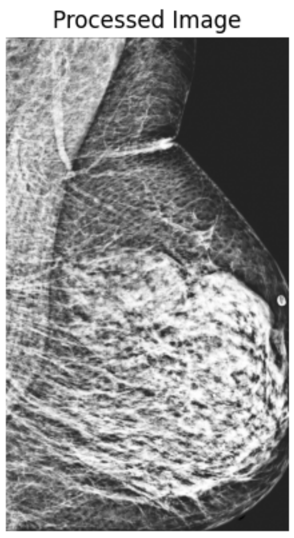


3.36 After CLAHE and Histogram Equalization





3.36 Final Processed Image



• Conclusion

4.1 Summary of Work

This project aimed to develop a robust preprocessing pipeline for medical imaging data stored in the DICOM format. The objectives were to:

- Correctly load `.dcm` files using `pydicom`.
- Apply essential transformations like VOI LUT, normalization, and format conversion for model compatibility.
- Automate horizontal flipping to address orientation inconsistencies.
- Remove irrelevant black borders or noise using intensity-based cropping.
- Resize images to a standardized resolution (768×1344) while preserving aspect ratio.
- Optionally enhance contrast using advanced techniques like CLAHE and histogram equalization.
- Allow modular toggling (e.g., apply_clahe, debug) for flexibility during development.

Each step was carefully implemented and verified using visualization (matplotlib) and debug flags to ensure that transformations were accurately applied and interpretable. The final image outputs are normalized, cropped, aligned, and formatted in a consistent shape, making them directly suitable for use in deep learning models or medical image analysis workflows.

4.2 Personal Takeaways & Scope for Improvement

Key Learnings:

- Deepened understanding of DICOM metadata: Learned how tags like WindowCenter, WindowWidth, and PhotometricInterpretation influence pixel interpretation.
- Image preprocessing in the medical domain is much more sensitive than general image processing due to its direct impact on diagnostics.
- Automation of flipping and cropping showed the value of data standardization before feeding into models.
- Visual debugging (via matplotlib) proved essential in interpreting changes and verifying correctness across steps.

Scope for Improvement:

- More robust flipping logic: The current logic assumes brightness asymmetry across the image, which may fail on symmetric or multi-view images. Incorporating anatomical landmarks or metadata-based orientation tags would improve accuracy.
- Adaptive parameter tuning: Currently, thresholds used in cropping (sum * std) are hardcoded. These could be dynamically computed based on dataset-wide statistics or be learned during training.
- Support for multiple modalities: The pipeline currently assumes grayscale single-channel images. Expanding support for multi-frame DICOMs (like ultrasound cine loops or MRI slices) would broaden applicability.
- Batch Processing & Logging: Adding support for batch preprocessing with logs per image (e.g., time taken, applied transformations) can enhance pipeline traceability and scaling.
- GUI or Web Interface: A simple UI to toggle preprocessing options and preview results (using Streamlit or Gradio) could make it user-friendly for medical professionals without programming skills.

REFERENCES

[1] **Mark Wijkhuizen** (2023). *RSNA ConvNeXtV2 Inference [TensorFlow]*.
Kaggle. https://www.kaggle.com/code/markwijkhuizen/rsna-convnextv2-inference-tensorflow

→ A comprehensive and modular inference notebook for RSNA chest X-ray preprocessing and model inference, serving as a practical base reference for this project's DICOM handling and image preparation pipeline.

[2] **Pianykh, O. S.** (2008). *Digital Imaging and Communications in Medicine (DICOM): A Practical Introduction and Survival Guide*. Springer. https://doi.org/10.1007/978-3-540-74571-2

→ A foundational reference for understanding DICOM structure, tags like `WindowCenter`, `WindowWidth`, and the overall medical imaging pipeline.

[3] **Zuiderveld, K.** (1994). *Contrast Limited Adaptive Histogram Equalization*. In P. Heckbert (Ed.), *Graphics Gems IV*(pp. 474–485). Academic Press. https://doi.org/10.1016/B978-0-08-050755-2.50065-3
→ The original algorithm for CLAHE, used optionally in this preprocessing pipeline for contrast enhancement.

[4] **Bob de Graaf** (2020). *DICOM SDL + VOI LUT Preprocessing – Kaggle Notebook*.
Kaggle. https://www.kaggle.com/code/bobdegraaf/dicomsdl-voi-lut
→ Used for implementing and adapting the VOI LUT function to adjust image contrast based on DICOM metadata.

[5] **OpenCV Development Team**. *OpenCV: Open Source Computer Vision Library*. https://opencv.org/
→ Library used for applying various image preprocessing techniques like resizing, histogram equalization, and CLAHE.

[6] **NumPy Developers**. *NumPy: The fundamental package for scientific computing with Python*. https://numpy.org/
→ Used for fast matrix operations, normalization, cropping, and data manipulation.

[7] **pydicom Contributors**. *pydicom Documentation*. https://pydicom.github.io/
→ Primary library used to load and parse DICOM (.dcm) files including metadata access.