01_exploration

February 15, 2026

1 Chest X-Ray Pneumonia Detection - Data Exploration

1.1 Mini Project 5: CNN Image Classifier

Dataset: Chest X-Ray Pneumonia Detection
Source: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

This notebook explores the dataset to understand: - Dataset structure and size - Class distribution and imbalance - Image properties (dimensions, color channels) - Visual differences between classes - Data quality issues

[1]:
```
#pip install opencv-python
```

[2]:
```
#pip install opencv-python seaborn
```

[3]:
```
# Import libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
import cv2
from collections import Counter
import warnings
warnings.filterwarnings('ignore')

# Set style
plt.style.use('default')
sns.set_palette("husl")

print("Libraries imported successfully!")
```

```
Libraries imported successfully!
```

1.2 1. Dataset Structure

First, let's understand how the dataset is organized.

[4]:
```
# Define dataset path - UPDATE THIS PATH TO YOUR LOCAL DIRECTORY
# Example: DATA_DIR = Path('/path/to/chest_xray')
DATA_DIR = Path('../data/chest_xray')

# Check if directory exists
if not DATA_DIR.exists():
    print(f" ERROR: Directory not found at {DATA_DIR}")
    print("\nPlease update DATA_DIR to point to your chest_xray folder.")
    print("Expected structure:")
    print("chest_xray/")
    print("   train/")
    print("   val/")
    print("   test/")
else:
    print(f" Dataset found at: {DATA_DIR}")
    print("\nDirectory structure:")
    for item in sorted(DATA_DIR.iterdir()):
        print(f"  - {item.name}")
```

```
 Dataset found at: ../data/chest_xray

Directory structure:
  - .DS_Store
  - test
  - train
  - val
```

1.3 2. Count Images Per Split and Class

Let's count how many images we have in each dataset split and class.

[5]:
```
def count_images(data_dir):
    """
    Count images in each split (train/val/test) and class (NORMAL/PNEUMONIA)
    """
    results = {}

    for split in ['train', 'val', 'test']:
        split_path = data_dir / split
        if not split_path.exists():
            print(f" Warning: {split} directory not found")
            continue

        split_counts = {}
        for class_name in ['NORMAL', 'PNEUMONIA']:
            class_path = split_path / class_name
            if class_path.exists():
                # Count image files (jpg, jpeg, png)
```

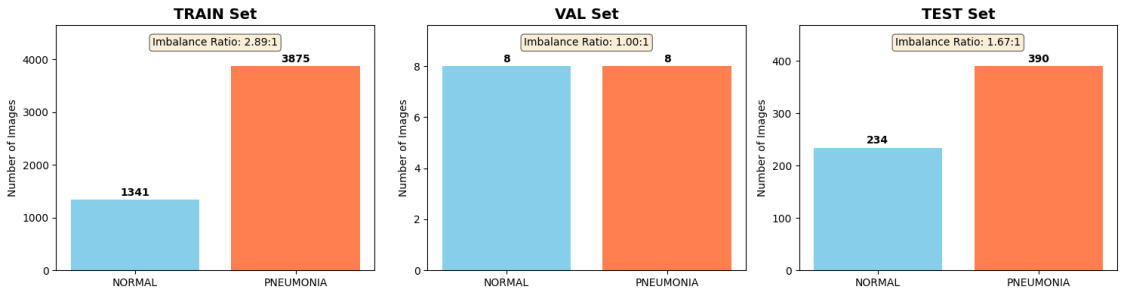
```
                images = list(class_path.glob('*.jpeg')) + \
                        list(class_path.glob('*.jpg')) + \
                        list(class_path.glob('*.png'))
                split_counts[class_name] = len(images)
            else:
                split_counts[class_name] = 0

        results[split] = split_counts

    return results

# Count images
image_counts = count_images(DATA_DIR)

# Display as DataFrame
df_counts = pd.DataFrame(image_counts).T
df_counts['Total'] = df_counts.sum(axis=1)
df_counts.loc['Total'] = df_counts.sum()

print("\n  Image Count Summary:")
print("=" * 50)
print(df_counts)
print("=" * 50)
```

```
  Image Count Summary:
==================================================
       NORMAL  PNEUMONIA  Total
train    1341       3875   5216
val         8          8     16
test      234        390    624
Total    1583       4273   5856
==================================================
```

1.4 3. Class Distribution Visualization

[6]:
```
# Visualize class distribution
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

for idx, split in enumerate(['train', 'val', 'test']):
    if split in image_counts:
        counts = image_counts[split]
        axes[idx].bar(counts.keys(), counts.values(), color=['skyblue',␣
 ↪'coral'])
        axes[idx].set_title(f'{split.upper()} Set', fontsize=14,␣
 ↪fontweight='bold')
        axes[idx].set_ylabel('Number of Images')
```

```
        axes[idx].set_ylim(0, max(counts.values()) * 1.2)

        # Add value labels on bars
        for i, (k, v) in enumerate(counts.items()):
            axes[idx].text(i, v + max(counts.values())*0.02, str(v),
                           ha='center', fontweight='bold')

        # Calculate imbalance ratio
        if counts['NORMAL'] > 0:
            ratio = counts['PNEUMONIA'] / counts['NORMAL']
            axes[idx].text(0.5, max(counts.values())*1.1,
                           f'Imbalance Ratio: {ratio:.2f}:1',
                           ha='center', transform=axes[idx].transData,
                           bbox=dict(boxstyle='round', facecolor='wheat',␣
 ↪alpha=0.5))

plt.tight_layout()
plt.savefig('../results/class_distribution.png', dpi=300, bbox_inches='tight')
plt.show()

print("\n Class Imbalance Analysis:")
for split in ['train', 'val', 'test']:
    if split in image_counts and image_counts[split]['NORMAL'] > 0:
        ratio = image_counts[split]['PNEUMONIA'] / image_counts[split]['NORMAL']
        print(f"  {split.upper()}: {ratio:.2f} pneumonia images per 1 normal␣
 ↪image")
```




```
 Class Imbalance Analysis:
  TRAIN: 2.89 pneumonia images per 1 normal image
  VAL: 1.00 pneumonia images per 1 normal image
  TEST: 1.67 pneumonia images per 1 normal image
```

1.5 4. Image Properties Analysis

Let's examine the dimensions and properties of our images.

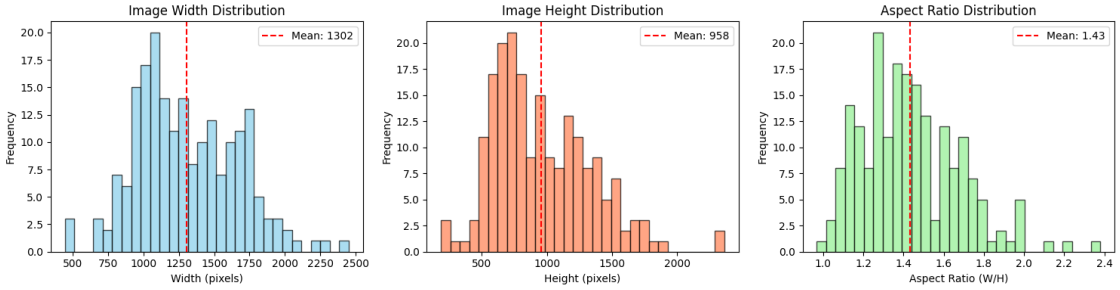
[7]:
```
def analyze_image_properties(data_dir, split='train', sample_size=100):
    """
    Analyze image dimensions and properties
    """
    properties = {
        'widths': [],
        'heights': [],
        'channels': [],
        'aspect_ratios': []
    }

    split_path = data_dir / split

    # Collect all image paths
    all_images = []
    for class_name in ['NORMAL', 'PNEUMONIA']:
        class_path = split_path / class_name
        if class_path.exists():
            all_images.extend(list(class_path.glob('*.jpeg')))
            all_images.extend(list(class_path.glob('*.jpg')))
            all_images.extend(list(class_path.glob('*.png')))

    # Sample images
    if len(all_images) > sample_size:
        sample_images = np.random.choice(all_images, sample_size, replace=False)
    else:
        sample_images = all_images

    print(f"Analyzing {len(sample_images)} images from {split} set...")

    for img_path in sample_images:
        img = cv2.imread(str(img_path))
        if img is not None:
            h, w = img.shape[:2]
            properties['heights'].append(h)
            properties['widths'].append(w)
            properties['channels'].append(img.shape[2] if len(img.shape) == 3␣
 ↪else 1)
            properties['aspect_ratios'].append(w / h)

    return properties

# Analyze training set
props = analyze_image_properties(DATA_DIR, 'train', sample_size=200)

# Display statistics
print("\n  Image Dimension Statistics:")
```

```
print("=" * 50)
print(f"Width:  min={min(props['widths'])}, max={max(props['widths'])},␣
 ↪mean={np.mean(props['widths']):.1f}")
print(f"Height: min={min(props['heights'])}, max={max(props['heights'])},␣
 ↪mean={np.mean(props['heights']):.1f}")
print(f"Channels: {Counter(props['channels'])}")
print(f"Aspect Ratio: min={min(props['aspect_ratios']):.2f},␣
 ↪max={max(props['aspect_ratios']):.2f}")
print("=" * 50)
```

```
Analyzing 200 images from train set…

  Image Dimension Statistics:
==================================================
Width:  min=445, max=2450, mean=1302.5
Height: min=187, max=2363, mean=957.7
Channels: Counter({3: 200})
Aspect Ratio: min=0.97, max=2.38
==================================================
```

[8]:
```
# Visualize dimension distributions
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

# Width distribution
axes[0].hist(props['widths'], bins=30, color='skyblue', edgecolor='black',␣
 ↪alpha=0.7)
axes[0].set_xlabel('Width (pixels)')
axes[0].set_ylabel('Frequency')
axes[0].set_title('Image Width Distribution')
axes[0].axvline(np.mean(props['widths']), color='red', linestyle='--',
                label=f'Mean: {np.mean(props["widths"]):.0f}')
axes[0].legend()

# Height distribution
axes[1].hist(props['heights'], bins=30, color='coral', edgecolor='black',␣
 ↪alpha=0.7)
axes[1].set_xlabel('Height (pixels)')
axes[1].set_ylabel('Frequency')
axes[1].set_title('Image Height Distribution')
axes[1].axvline(np.mean(props['heights']), color='red', linestyle='--',
                label=f'Mean: {np.mean(props["heights"]):.0f}')
axes[1].legend()

# Aspect ratio distribution
axes[2].hist(props['aspect_ratios'], bins=30, color='lightgreen',␣
 ↪edgecolor='black', alpha=0.7)
axes[2].set_xlabel('Aspect Ratio (W/H)')
```

```
axes[2].set_ylabel('Frequency')
axes[2].set_title('Aspect Ratio Distribution')
axes[2].axvline(np.mean(props['aspect_ratios']), color='red', linestyle='--',
                label=f'Mean: {np.mean(props["aspect_ratios"]):.2f}')
axes[2].legend()

plt.tight_layout()
plt.savefig('../results/image_properties.png', dpi=300, bbox_inches='tight')
plt.show()
```

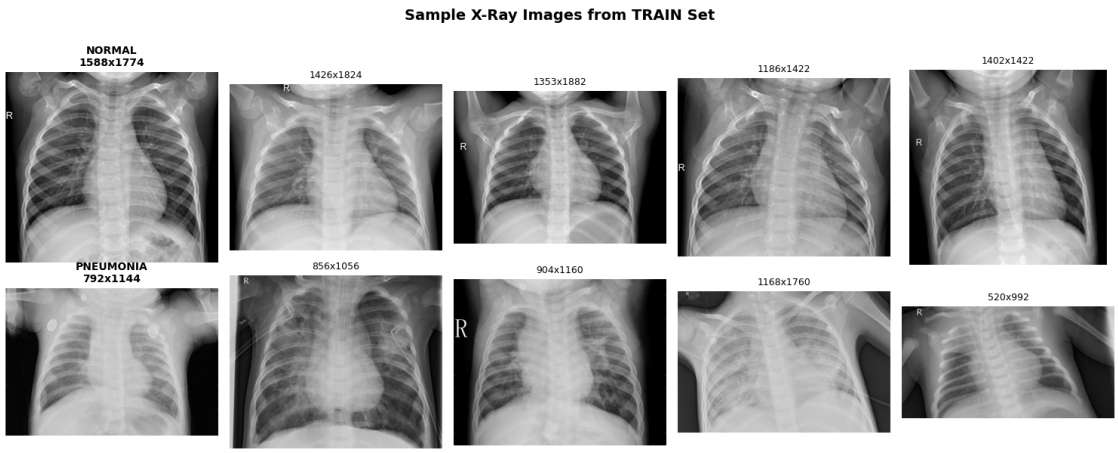


1.6 5. Visual Inspection: Sample Images

Let's look at actual X-ray images from both classes.

[9]:
```
def display_sample_images(data_dir, split='train', n_samples=5):
    """
    Display random sample images from each class
    """
    fig, axes = plt.subplots(2, n_samples, figsize=(15, 6))

    for row, class_name in enumerate(['NORMAL', 'PNEUMONIA']):
        class_path = data_dir / split / class_name

        # Get random images
        image_files = list(class_path.glob('*.jpeg')) + \
                      list(class_path.glob('*.jpg')) + \
                      list(class_path.glob('*.png'))

        if len(image_files) >= n_samples:
            sample_files = np.random.choice(image_files, n_samples,␣
↪replace=False)
        else:
            sample_files = image_files

        for col, img_path in enumerate(sample_files):
            img = cv2.imread(str(img_path))
```

```
            if img is not None:
                # Convert BGR to RGB
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

                axes[row, col].imshow(img, cmap='gray')
                axes[row, col].axis('off')

                if col == 0:
                    axes[row, col].set_title(f'{class_name}\n{img.
↪shape[0]}x{img.shape[1]}',
                                             fontweight='bold', fontsize=10)
                else:
                    axes[row, col].set_title(f'{img.shape[0]}x{img.shape[1]}',␣
↪fontsize=9)

    plt.suptitle(f'Sample X-Ray Images from {split.upper()} Set',
                 fontsize=14, fontweight='bold', y=1.02)
    plt.tight_layout()
    plt.savefig('../results/sample_images.png', dpi=300, bbox_inches='tight')
    plt.show()

# Display samples
display_sample_images(DATA_DIR, 'train', n_samples=5)
```



1.7 6. Pixel Intensity Analysis

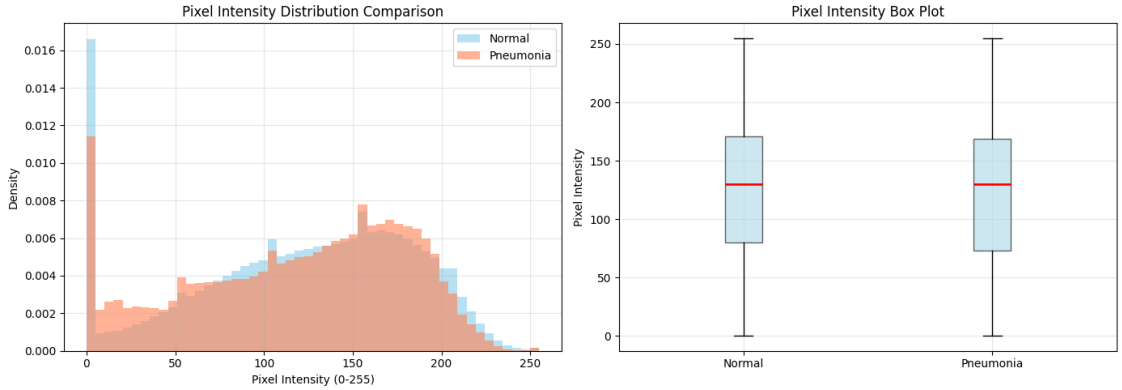
Let's analyze the pixel intensity distributions to understand the contrast and brightness characteristics.

[10]:
```
def analyze_pixel_intensity(data_dir, split='train', n_samples=50):
    """
```

```
    Analyze pixel intensity distributions for each class
    """
    intensity_data = {'NORMAL': [], 'PNEUMONIA': []}

    for class_name in ['NORMAL', 'PNEUMONIA']:
        class_path = data_dir / split / class_name
        image_files = list(class_path.glob('*.jpeg')) + \
                      list(class_path.glob('*.jpg')) + \
                      list(class_path.glob('*.png'))

        if len(image_files) > n_samples:
            sample_files = np.random.choice(image_files, n_samples,␣
 ↪replace=False)
        else:
            sample_files = image_files

        for img_path in sample_files:
            img = cv2.imread(str(img_path), cv2.IMREAD_GRAYSCALE)
            if img is not None:
                intensity_data[class_name].extend(img.flatten())

    return intensity_data

# Analyze intensities
print("Analyzing pixel intensities...")
intensities = analyze_pixel_intensity(DATA_DIR, 'train', n_samples=50)

# Plot intensity distributions
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Histogram comparison
axes[0].hist(intensities['NORMAL'], bins=50, alpha=0.6, label='Normal',␣
 ↪color='skyblue', density=True)
axes[0].hist(intensities['PNEUMONIA'], bins=50, alpha=0.6, label='Pneumonia',␣
 ↪color='coral', density=True)
axes[0].set_xlabel('Pixel Intensity (0-255)')
axes[0].set_ylabel('Density')
axes[0].set_title('Pixel Intensity Distribution Comparison')
axes[0].legend()
axes[0].grid(alpha=0.3)

# Box plot comparison
data_to_plot = [intensities['NORMAL'], intensities['PNEUMONIA']]
axes[1].boxplot(data_to_plot, labels=['Normal', 'Pneumonia'], patch_artist=True,
               boxprops=dict(facecolor='lightblue', alpha=0.6),
               medianprops=dict(color='red', linewidth=2))
axes[1].set_ylabel('Pixel Intensity')
```

```
axes[1].set_title('Pixel Intensity Box Plot')
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.savefig('../results/pixel_intensity_analysis.png', dpi=300,␣
 ↪bbox_inches='tight')
plt.show()

# Print statistics
print("\n Pixel Intensity Statistics:")
print("=" * 50)
for class_name in ['NORMAL', 'PNEUMONIA']:
    data = intensities[class_name]
    print(f"{class_name}:")
    print(f"  Mean: {np.mean(data):.2f}")
    print(f"  Std:  {np.std(data):.2f}")
    print(f"  Min:  {np.min(data):.2f}")
    print(f"  Max:  {np.max(data):.2f}")
print("=" * 50)
```

```
Analyzing pixel intensities…
```




```
 Pixel Intensity Statistics:
==================================================
NORMAL:
  Mean: 121.21
  Std:  62.01
  Min:  0.00
  Max:  255.00
PNEUMONIA:
  Mean: 119.08
  Std:  61.06
  Min:  0.00
```

```
  Max:  255.00
==================================================
```

1.8 7. Key Findings Summary

Let's summarize the important findings from our exploration.

[11]:
```
print("\n" + "="*60)
print("KEY FINDINGS & RECOMMENDATIONS")
print("="*60)

# Calculate key metrics
if 'train' in image_counts:
    train_total = sum(image_counts['train'].values())
    train_ratio = image_counts['train']['PNEUMONIA'] /␣
 ↪image_counts['train']['NORMAL']

    print("\n1  DATASET SIZE:")
    print(f"   Total training images: {train_total}")
    print(f"   Validation set size: {sum(image_counts.get('val', {}).
 ↪values())}")
    print(f"   Test set size: {sum(image_counts.get('test', {}).values())}")

    print("\n2  CLASS IMBALANCE:")
    print(f"   Training set ratio: {train_ratio:.2f}:1 (Pneumonia:Normal)")
    if train_ratio > 2:
        print("    SIGNIFICANT IMBALANCE DETECTED")
        print("   → Recommendation: Use class_weight in model training")
        print(f"   → Suggested weight for NORMAL: {train_ratio:.2f}")
        print(f"   → Suggested weight for PNEUMONIA: 1.0")

    print("\n3  IMAGE PROPERTIES:")
    print(f"   Average dimensions: {np.mean(props['widths']):.0f} x {np.
 ↪mean(props['heights']):.0f}")
    print(f"   Size variation: Width range [{min(props['widths'])},␣
 ↪{max(props['widths'])}]")
    print("   → Recommendation: Resize all images to consistent size (e.g.,␣
 ↪224x224 or 150x150)")

    most_common_channels = Counter(props['channels']).most_common(1)[0][0]
    if most_common_channels == 3:
        print("    Images have 3 channels (RGB)")
        print("   → Recommendation: Convert to grayscale OR keep RGB (X-rays␣
 ↪are typically grayscale)")
    else:
        print("    Images are grayscale (1 channel)")
```

```
    print("\n4  VALIDATION SET:")
    val_total = sum(image_counts.get('val', {}).values())
    if val_total < 100:
        print(f"     Validation set is very small ({val_total} images)")
        print("   → Recommendation: Consider creating new train/val split from␣
 ↪training data")
        print("   → Suggested: 80/20 or 85/15 split")

    print("\n5  RECOMMENDED PREPROCESSING PIPELINE:")
    print("   1. Resize to 224x224 or 150x150")
    print("   2. Convert to grayscale (if needed)")
    print("   3. Normalize pixel values to [0, 1]")
    print("   4. Use class weights to handle imbalance")

    print("\n6  AUGMENTATION SUGGESTIONS (Medical Imaging Safe):")
    print("    Rotation (±10-15 degrees)")
    print("    Width/Height shift (±10%)")
    print("    Zoom (0.9-1.1)")
    print("    Brightness adjustment")
    print("    Horizontal flip (heart position matters!)")
    print("    Vertical flip (not realistic)")

print("\n" + "="*60)
print("NEXT STEPS:")
print("="*60)
print("1. Decide on image size (224x224 recommended)")
print("2. Consider re-splitting train/val if validation set is too small")
print("3. Build baseline CNN with 3-4 conv blocks")
print("4. Calculate and use class weights during training")
print("5. Start simple, then add complexity")
print("="*60)
```

```
============================================================
KEY FINDINGS & RECOMMENDATIONS
============================================================

1  DATASET SIZE:
   Total training images: 5216
   Validation set size: 16
   Test set size: 624

2  CLASS IMBALANCE:
   Training set ratio: 2.89:1 (Pneumonia:Normal)
     SIGNIFICANT IMBALANCE DETECTED
   → Recommendation: Use class_weight in model training
   → Suggested weight for NORMAL: 2.89
   → Suggested weight for PNEUMONIA: 1.0
```

```
3  IMAGE PROPERTIES:
   Average dimensions: 1302 x 958
   Size variation: Width range [445, 2450]
   → Recommendation: Resize all images to consistent size (e.g., 224x224 or
150x150)
    Images have 3 channels (RGB)
   → Recommendation: Convert to grayscale OR keep RGB (X-rays are typically
grayscale)

4  VALIDATION SET:
    Validation set is very small (16 images)
   → Recommendation: Consider creating new train/val split from training data
   → Suggested: 80/20 or 85/15 split

5  RECOMMENDED PREPROCESSING PIPELINE:
   1. Resize to 224x224 or 150x150
   2. Convert to grayscale (if needed)
   3. Normalize pixel values to [0, 1]
   4. Use class weights to handle imbalance

6  AUGMENTATION SUGGESTIONS (Medical Imaging Safe):
    Rotation (±10-15 degrees)
    Width/Height shift (±10%)
    Zoom (0.9-1.1)
    Brightness adjustment
    Horizontal flip (heart position matters!)
    Vertical flip (not realistic)

============================================================
NEXT STEPS:
============================================================
1. Decide on image size (224x224 recommended)
2. Consider re-splitting train/val if validation set is too small
3. Build baseline CNN with 3-4 conv blocks
4. Calculate and use class weights during training
5. Start simple, then add complexity
============================================================
```

1.9 8. Data Quality Check

Let's check for any corrupted or problematic images.

[12]:
```python
def check_image_quality(data_dir, split='train'):
    """
    Check for corrupted or problematic images
    """
    issues = []
```

```
    split_path = data_dir / split

    print(f"Checking image quality in {split} set...")

    for class_name in ['NORMAL', 'PNEUMONIA']:
        class_path = split_path / class_name
        if not class_path.exists():
            continue

        image_files = list(class_path.glob('*.jpeg')) + \
                      list(class_path.glob('*.jpg')) + \
                      list(class_path.glob('*.png'))

        for img_path in image_files:
            try:
                img = cv2.imread(str(img_path))
                if img is None:
                    issues.append(f"Cannot read: {img_path.name}")
                elif img.size == 0:
                    issues.append(f"Empty image: {img_path.name}")
                elif img.shape[0] < 50 or img.shape[1] < 50:
                    issues.append(f"Too small: {img_path.name} ({img.shape})")
            except Exception as e:
                issues.append(f"Error loading {img_path.name}: {str(e)}")

    return issues

# Check quality
quality_issues = check_image_quality(DATA_DIR, 'train')

if quality_issues:
    print(f"\n Found {len(quality_issues)} potential issues:")
    for issue in quality_issues[:10]:  # Show first 10
        print(f"  - {issue}")
    if len(quality_issues) > 10:
        print(f"  ... and {len(quality_issues) - 10} more")
else:
    print("\n No quality issues detected!")
```

```
Checking image quality in train set…

 No quality issues detected!
```

1.10 9. Export Summary Report

[13]:
```
# Create summary report
summary = {
    'Dataset': 'Chest X-Ray Pneumonia Detection',
    'Source': 'Kaggle - Paul Mooney',
    'Total Images': sum(sum(counts.values()) for counts in image_counts.
 ↪values()),
    'Training Images': sum(image_counts.get('train', {}).values()),
    'Validation Images': sum(image_counts.get('val', {}).values()),
    'Test Images': sum(image_counts.get('test', {}).values()),
    'Classes': 2,
    'Average Width': int(np.mean(props['widths'])),
    'Average Height': int(np.mean(props['heights'])),
    'Imbalance Ratio (train)': f"{image_counts['train']['PNEUMONIA'] /␣
 ↪image_counts['train']['NORMAL']:.2f}:1"
}

# Save as DataFrame
df_summary = pd.DataFrame([summary]).T
df_summary.columns = ['Value']
df_summary.to_csv('../results/dataset_summary.csv')

print("\n Dataset Summary:")
print(df_summary)
print("\n Summary saved to 'dataset_summary.csv'")
```

```
 Dataset Summary:
                                                Value
Dataset                 Chest X-Ray Pneumonia Detection
Source                              Kaggle - Paul Mooney
Total Images                                        5856
Training Images                                     5216
Validation Images                                     16
Test Images                                          624
Classes                                                2
Average Width                                       1302
Average Height                                       957
Imbalance Ratio (train)                          2.89:1

 Summary saved to 'dataset_summary.csv'
```

1.11 Exploration Complete!

What we've learned: - Dataset structure and size - Class distribution and imbalance - Image dimensions and properties - Visual characteristics of both classes - Potential data quality issues

Next steps: 1. Review the key findings above 2. Decide on preprocessing strategy 3. Move on to building the baseline CNN

Files generated: - `class_distribution.png` - `image_properties.png` - `sample_images.png` - `pixel_intensity_analysis.png` - `dataset_summary.csv`