PnemoniaDetector

December 9, 2025

1 Pneumonia Classification from Chest X-Ray Images Using FastAI Transfer Learning

1.1 Introduction

This project leverages deep learning to automate the detection of pneumonia in chest X-ray images and further classify it into bacterial or viral subtypes.

1.1.1 Data Sources

- Kaggle - Chest X-Ray Images (Pneumonia) dataset, containing approximately 5,800 chest X-ray images.

Important: Extract the downloaded `chest_xray` folder into the `data` directory with the following structure: `data/chest_xray/train|val|test/NORMAL|PNEUMONIA/`.

1.1.2 Methodology

We will use a **two-stage pipeline** designed for high-sensitivity screening followed by focused subtype classification:

- **Stage 1: Pneumonia Detection (Normal vs. Pneumonia)**
  - Utilizes a ResNet-50 model with transfer learning.
  - Prioritizes sensitivity (recall) to minimize missed cases.
- **Stage 2: Pneumonia Classification (Bacterial vs. Viral)**
  - Analyzes images flagged by Stage 1 to distinguish bacterial from viral pneumonia.
  - Reuses the trained backbone from Stage 1 via transfer learning.

Image Preprocessing and Evaluation:
We will evaluate both stages using two types of image inputs:

Image Preprocessing and Evaluation: We evaluate both stages using two input types: - Original images with grayscale conversion (baseline). - CLAHE-enhanced images with Hot colormap: - CLAHE boosts local contrast. - Hot colormap enhances feature visibility.

1.1.3 References

This project uses code and ideas from the following sources:

1. Howard, J., & Gugger, S. (2020). *Deep Learning for Coders with FastAI and PyTorch*. O'Reilly Media.

2. Waheed, S., Ghosh, S., & Gadekallu, T. R. (2022). Pre-processing methods in chest X-ray image classification. *Frontiers in Medicine*, 9, 898289. https://doi.org/10.3389/fmed.2022.898289

3. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 770-778.

4. Panwar, H. et al. (2020). A deep learning and grad-CAM based color visualization approach for fast detection of COVID-19 cases using chest X-ray and CT-Scan images. Chaos Solitons Fractals, 140, 110190. https://pmc.ncbi.nlm.nih.gov/articles/PMC7413068/

1.2 Setup

1.2.1 Load Libraries and Initial Environment Variables

[141]:
```
import pandas as pd
import numpy as np
import os
from pathlib import Path
from joblib import Parallel, delayed
import multiprocessing
import random

# Plots
import seaborn as sns
import matplotlib.pyplot as plt


# Image processing
from PIL import Image, ImageStat
from skimage import io, measure, exposure, img_as_ubyte
from skimage import color as skcolor
import cv2

# Machine learning
from fastai.vision.all import *
from fastcore.all import *
from fastai.metrics import *
import torch
from torchvision.ops import sigmoid_focal_loss
from sklearn.model_selection import StratifiedShuffleSplit, train_test_split
from sklearn.utils import class_weight, resample
from sklearn.metrics import accuracy_score,roc_auc_score, precision_score,␣
 ↪recall_score, f1_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.calibration import calibration_curve
```

```
# Pandas settings
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)

random.seed(42)
np.random.seed(42)
torch.manual_seed(42)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(42)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
```

[142]:
```
PROJECT_PATH = '../'
```

Mount Google drive in google colab (skip if running locally)

[143]:
```
from google.colab import drive

#Check if drive is already mounted
if not os.path.exists("/content/drive/MyDrive"):
  print("Mounting Google Drive...")
  drive.mount("/content/drive")
  print(" Drive mounted successfully!")
else:
  print(" Drive already mounted")

## Set project path for Github repo on Google Drive
PROJECT_PATH = '/content/drive/MyDrive/SeattleU/5100-DataIntro/Projects/
 ↪image-classification/'
```

```
 Drive already mounted
```

[144]:
```
#PROJECT_PATH = r'C:\path\to\your\project'  # Uncomment and set this path if␣
 ↪running locally
```

1.2.2 Define Common Variables

[145]:
```
import os
import sys

# Set up working directories

DATA_PATH = os.path.join(PROJECT_PATH, 'data')
MODEL_PATH = os.path.join(PROJECT_PATH, 'models')
CODE_PATH = os.path.join(PROJECT_PATH, 'code')

# Add CODE_PATH to sys.path if it's not already there
```

```
if CODE_PATH not in sys.path:
    sys.path.insert(0, CODE_PATH) # Insert at the beginning to prioritize it

print(f'Project path: {PROJECT_PATH}')
print(f'Data will be saved to: {DATA_PATH}')
print(f'Models will be saved to: {MODEL_PATH}')
print(f'Code modules will be loaded from: {CODE_PATH}')

images_original_path = DATA_PATH + '/chest_xray'

# Common variables
metrics = ['contrast', 'entropy', 'signal_noise_ratio', 'mean']
image_classes = ['normal', 'pneumonia']
image_subclasses = ['bacterial', 'viral']

num_cores = multiprocessing.cpu_count()
```

```
Project path: /content/drive/MyDrive/SeattleU/5100-DataIntro/Projects/image-
classification/
Data will be saved to:
/content/drive/MyDrive/SeattleU/5100-DataIntro/Projects/image-
classification/data
Models will be saved to:
/content/drive/MyDrive/SeattleU/5100-DataIntro/Projects/image-
classification/models
Code modules will be loaded from:
/content/drive/MyDrive/SeattleU/5100-DataIntro/Projects/image-
classification/code
```

1.2.3 Define Image Transformation Module

Define the image transformation functions and classes in `custom_transforms.py` for reusability.

[146]:
```
%%writefile {CODE_PATH}/custom_transforms.py

import numpy as np
import cv2
import random
import torch
from fastai.vision.all import PILImage, ItemTransform

# Image Transformation Settings
CLAHE_CLIP_LIMIT = 2.0
CLAHE_TILE_GRID_SIZE = (8, 8)
CLAHE_IMAGE_BLUR = 7

COLORMAP_SELECTION = 'HOT'
```

```
class EnsureGrayscale(ItemTransform):
  """Convert image to grayscale using pure numpy, then to 3-channel for␣
 ↪ResNet"""

  def __repr__(self):
      return f"{self.__class__.__name__}()"

  def encodes(self, x):
      is_tuple = isinstance(x, (tuple, list))
      img = x[0] if is_tuple else x
      label = x[1] if is_tuple and len(x) > 1 else None

      arr = np.array(img)

      # Convert to grayscale using numpy formula
      if len(arr.shape) == 2:
          # Already grayscale
          arr_gray = arr
      elif len(arr.shape) == 3:
          # RGB to grayscale: 0.299*R + 0.587*G + 0.114*B
          arr_gray = np.dot(arr[...,:3], [0.299, 0.587, 0.114]).astype(np.uint8)
      else:
          arr_gray = arr

      # Replicate to 3 channels for ResNet
      arr_3ch = np.stack([arr_gray, arr_gray, arr_gray], axis=-1)
      res = PILImage.create(arr_3ch)

      if label is not None:
          return (res, label)
      else:
          return (res,)


# Apply CLAHE to grayscale image
class CLAHETransform(ItemTransform):

    def __init__(self, p=1.0):
        self.clip_limit = CLAHE_CLIP_LIMIT
        self.tile_grid_size = CLAHE_TILE_GRID_SIZE
        self.medianBlur = CLAHE_IMAGE_BLUR
        self.p = p

    def __repr__(self):
      return (f"{self.__class__.__name__}("
              f"clip_limit={self.clip_limit}, "
              f"tile_grid_size={self.tile_grid_size}, "
```

```
            f"medianBlur={self.medianBlur}, "
            f"p={self.p})")

    def encodes(self, x):
        is_tuple = isinstance(x, (tuple, list))
        img = x[0] if is_tuple else x
        label = x[1] if is_tuple and len(x) > 1 else None

        if random.random() > self.p:
            return x

        arr = np.array(img)

        # Get grayscale
        if len(arr.shape) == 2:
            gray = arr
        elif len(arr.shape) == 3:
            gray = np.dot(arr[...,:3], [0.299, 0.587, 0.114]).astype(np.uint8)
        else:
            gray = arr

        gray = cv2.medianBlur(gray, self.medianBlur)
        # Apply CLAHE
        clahe = cv2.createCLAHE(clipLimit=self.clip_limit, tileGridSize=self.
 ↪tile_grid_size)

        gray = clahe.apply(gray)

        # Replicate to 3 channels
        arr_3ch = np.stack([gray, gray, gray], axis=-1)
        res = PILImage.create(arr_3ch)

        if label is not None:
            return (res, label)
        else:
            return (res,)

# Apply colormap to grayscale image
class ColormapTransform(ItemTransform):
    """Apply colormap to grayscale image"""
    def __init__(self, p=1.0, colormap=COLORMAP_SELECTION):
        self.colormap = colormap
        self.p = p
        self.cv2_colormaps = {
            'JET': cv2.COLORMAP_JET,
            'HOT': cv2.COLORMAP_HOT,
            'VIRIDIS': cv2.COLORMAP_VIRIDIS,
```

```
            'PLASMA': cv2.COLORMAP_PLASMA,
            'OCEAN': cv2.COLORMAP_OCEAN,
            'BONE': cv2.COLORMAP_BONE,
            'WINTER': cv2.COLORMAP_WINTER,
            'INFERNO': cv2.COLORMAP_INFERNO,
            'MAGMA': cv2.COLORMAP_MAGMA,
        }

    def encodes(self, x):
        is_tuple = isinstance(x, (tuple, list))
        img = x[0] if is_tuple else x
        label = x[1] if is_tuple and len(x) > 1 else None

        if random.random() > self.p:
            return x

        arr = np.array(img)

        # Get grayscale
        if len(arr.shape) == 2:
            gray = arr
        else:
            gray = cv2.cvtColor(arr, cv2.COLOR_RGB2GRAY)

        # Normalize to 0-255 if needed
        if gray.dtype != np.uint8:
            gray = ((gray - gray.min()) / (gray.max() - gray.min()) * 255).
↪astype(np.uint8)

        # Apply colormap
        if self.colormap in self.cv2_colormaps:
            colored = cv2.applyColorMap(gray, self.cv2_colormaps[self.colormap])
            colored = cv2.cvtColor(colored, cv2.COLOR_BGR2RGB)
        else:
            colored = np.stack([gray, gray, gray], axis=-1)

        res = PILImage.create(colored)

        if label is not None:
            return (res, label)
        else:
            return (res,)
    def __repr__(self):
      return (f"{self.__class__.__name__}("
              f"colormap='{self.colormap}', "
              f"p={self.p})")
```

```python
# Loss function with focus on most difficult images
class FastFocalLoss(torch.nn.Module):
    def __init__(self, alpha=1.0, gamma=2.0, reduction='mean'):
        """
        alpha: float or 1D tensor of shape [num_classes]
        gamma: focusing parameter
        """
        super().__init__()
        # register alpha as buffer so it moves with the module to cuda
        if isinstance(alpha, (list, tuple)):
            alpha = torch.tensor(alpha, dtype=torch.float)
        self.register_buffer('alpha', torch.tensor(alpha, dtype=torch.float))
        self.gamma = gamma
        self.reduction = reduction

    def forward(self, logits, targets):

        logp = torch.nn.functional.log_softmax(logits, dim=1)
        logp_t = logp.gather(1, targets.unsqueeze(1)).squeeze(1)
        p_t = logp_t.exp()

        # alpha per sample
        if self.alpha.ndim == 0:
            alpha_t = self.alpha
        else:
            alpha_t = self.alpha[targets]

        focal_loss = -alpha_t * (1 - p_t) ** self.gamma * logp_t

        if self.reduction == 'mean':
            return focal_loss.mean()
        if self.reduction == 'sum':
            return focal_loss.sum()
        return focal_loss

    def activation(self, x):
        return torch.nn.functional.softmax(x, dim=1)

    def decodes(self, x):
        return x.argmax(dim=1)
```

```
Overwriting /content/drive/MyDrive/SeattleU/5100-DataIntro/Projects/image-
classification/code/custom_transforms.py
```

1.2.4 Define Helper Functions for Image Categorization and Processing

[147]:
```
# Image classification functions
def image_class_function(orig_file_path):
  file_path_str = str(orig_file_path)
  if 'NORMAL' in file_path_str.upper():
    return 'normal'
  elif 'VIRUS' in file_path_str.upper() or 'BACTERIA' in file_path_str.upper() :
    return 'pneumonia'
  else:
    return None

def image_usage_function(orig_file_path) :
  file_path_str = str(orig_file_path).upper()
  if 'TEST' in file_path_str:
    return 'test'
  elif 'TRAIN' in file_path_str:
    return 'train'
  #merge with train
  elif 'VAL' in file_path_str:
    return 'train'
  else :
    return None

def image_subclass_function(orig_file_path):
  file_path_str = str(orig_file_path)
  if 'BACTERIA' in file_path_str.upper():
    return 'bacterial'
  elif 'VIRUS' in file_path_str.upper():
    return 'viral'
  else:
    return None

def image_absolute_path_function(orig_file_path):
  return os.path.join(PROJECT_PATH, orig_file_path)

# Visualization helper function

## Plot metrics by image classification (normal vs pneumonia) or by subclass␣
 ↪(bacterial vs viral)
def plot_metrics_by_class(df, metrics, classes, class_column='image_class',
                          title_prefix="", color_map=None):

    plt.figure(figsize=(15, 6))
    for i in range(len(metrics)):
        plt.subplot(1, 4, i + 1)
        for cls in classes:
```

```
            vals = df[df[class_column] == cls][metrics[i]].dropna()
            color = color_map.get(cls, None) if color_map else None
            sns.histplot(
                vals,
                bins=40,
                label=cls,
                alpha=0.6,
                linewidth=1,
                stat='density',
                common_norm=False,
                color=color
            )
        title = f'{title_prefix}{metrics[i].capitalize()} by {class_column.
 ↪replace("_", " ").title()}'
        plt.title(title, fontsize=14)
        plt.xlabel(metrics[i].capitalize(), fontsize=12)
        plt.ylabel('Density', fontsize=12)
        plt.legend()
    plt.tight_layout()
    plt.show()


def print_learner_config(learn):
    lf = learn.loss_func
    print("=== Loss ===")
    print(f"Loss func  : {lf.__class__.__name__}")

    if isinstance(lf, CrossEntropyLossFlat):
        if hasattr(lf.func, "weight"):
          print("raw weight:", lf.func.weight)
    elif isinstance(lf, FastFocalLoss):
        if lf.alpha is not None:
            a = lf.alpha.detach().cpu().numpy()
            print(f"  alpha          : {a}")
        if lf.gamma is not None:
            g = lf.gamma
            print(f"  gammma         : {g}")
    else:
        print("  (generic loss; params from __dict__)")
        for k, v in lf.__dict__.items():
            if isinstance(v, torch.Tensor):
                v = v.detach().cpu().numpy()
            print(f"    {k}: {v}")

    print("\n=== Optimizer & training ===")
    wd = learn.wd if learn.wd is not None else 1e-2
    print(f"Weight decay (wd): {wd}")
```

```
    print("\n=== Dropout in model head ===")
    for m in learn.model.modules():
        if isinstance(m, nn.Dropout):
            print(f"  Dropout p={m.p}")

    print("\n=== Data & augmentations ===")
    print(f"Batch size : {learn.dls.bs}")
    print(f"Train batches: {len(learn.dls.train)}, Valid batches: {len(learn.
 ↪dls.valid)}")
    print("Item tfms  :")
    for t in learn.dls.after_item:
        print(f"  - {t}")

    print("Batch tfms :")
    for t in learn.dls.after_batch:
        print(f"  - {t}")

from custom_transforms import EnsureGrayscale, CLAHETransform,␣
 ↪ColormapTransform, FastFocalLoss

def compute_metrics_with_transform(img_path, transforms=None):
    abs_path = os.path.join(PROJECT_PATH, img_path)
    img = PILImage.create(abs_path)

    width, height = img.size

    if transforms is None:
        transforms = []
    else:
        transforms = [transforms]

    lbl = ''
    for t in transforms:
        res_obj = t.encodes((img, lbl))
        if isinstance(res_obj, (tuple, list)) and len(res_obj) == 2:
            img, lbl = res_obj
        elif isinstance(res_obj, (tuple, list)) and len(res_obj) == 1:
            img = res_obj[0]
        else:
            img = res_obj


    arr = np.array(img)
    # If arr is RGB, convert to grayscale using the standard weights:
    if arr.ndim == 3:
          arr = np.dot(arr[...,:3], [0.299, 0.587, 0.114])
```

```
    arr = arr.astype(np.float64)  # prevent overflow

    mean = np.mean(arr)

    contrast = np.std(arr)

    # Entropy
    hist, bins = np.histogram(arr, bins=256, range=(0,255), density=True)
    hist = hist[hist > 0]  # remove zeros to avoid log(0)
    entropy = -np.sum(hist * np.log2(hist))

    signal_noise_ratio = mean / (contrast + 1e-8)

    return {
        'orig_file_path': img_path,
        'mean': mean,
        'contrast': contrast,
        'entropy': entropy,
        'signal_noise_ratio': signal_noise_ratio,
        'width': width,
        'height': height
    }
```

1.2.5 Change the Runtime

Change the Runtime to utilize the NVIDIA T4 GPU, if it is not already set to the GPU.

In the menu above click Runtime → Change runtime type and change Hardware accelerator to T4 GPU

1.3 Exploratory Data Analysis

Our exploratory data analysis examines sample images, analyzes statistics (contrast, entropy, signal-to-noise ratio), and identifies low-quality samples to ensure a clean dataset.

1.3.1 Load and Catalog Chest X-Ray Images

From the Kaggle Pneumonia dataset, we catalog all images by extracting metadata and computing metrics.

Get the file names for each image

[ ]:
```
allowed_extensions = ['.jpg', '.jpeg', '.png']

absolute_image_file_paths = get_image_files(DATA_PATH+'/chest_xray')

# Filter for specific image extensions
```

```
absolute_image_file_paths = [f for f in absolute_image_file_paths if f.suffix.
 ↪lower() in allowed_extensions]

image_file_paths = [str(f.relative_to(PROJECT_PATH)) for f in␣
 ↪absolute_image_file_paths]
print("Number of images",len(image_file_paths))
```

```
Number of images 5856
```

Build a pandas DataFrame to systematically organize all images with their metadata.

[ ]:
```
df = pd.DataFrame([str(f) for f in image_file_paths],␣
 ↪columns=['orig_file_path'])
df['file_name'] = df['orig_file_path'].apply(lambda x: os.path.basename(x))
df['image_class'] = df['orig_file_path'].apply(image_class_function)
df['image_subclass'] = df['orig_file_path'].apply(image_subclass_function)
df['usage_type'] = df['orig_file_path'].apply(image_usage_function)

# Filter out rows where image_class or usage_type is None
df.dropna(subset=['image_class', 'usage_type'], inplace=True)

# Group by usage_type, image_class, and image_subclass and count the filenames,␣
 ↪including NaN values in the grouping
df.groupby(['usage_type', 'image_class', 'image_subclass'],␣
 ↪dropna=False)['file_name'].count()
```

[ ]:
```
usage_type  image_class  image_subclass
test        normal       NaN              234
            pneumonia    bacterial        242
                         viral            148
train       normal       NaN             1349
            pneumonia    bacterial       2538
                         viral           1345
Name: file_name, dtype: int64
```

We can notice that we have almost 2x pneumonia-bacteria images than normal and pneumonia-virus. We will need to handle that discrepancy for training set.

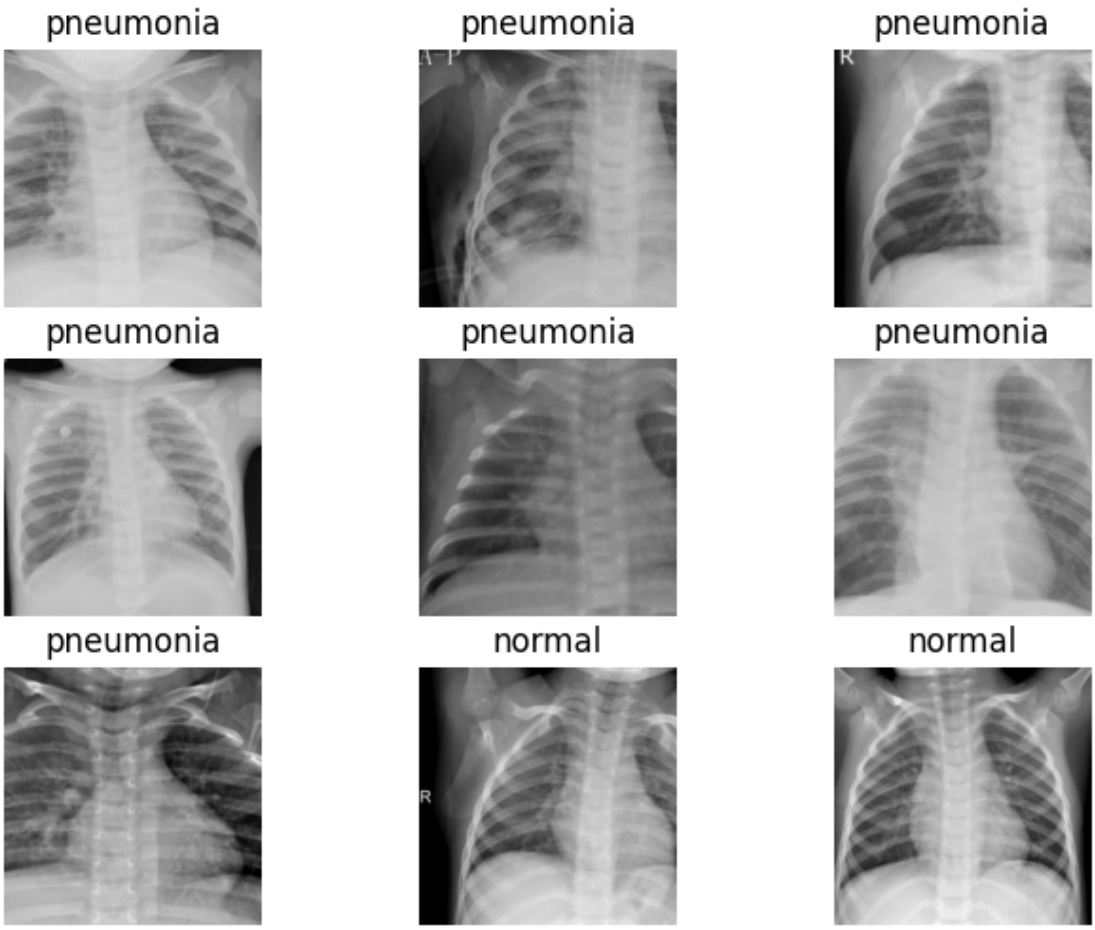
1.3.2 Show Sample Images

Sample images reveal clear visual differences: pneumonia cases show opacity, while normal lungs appear clear. Viral and bacterial pneumonia appear similar, with subtle differences.

[ ]:
```
from matplotlib.typing import path
# Set up a new DataBlock for 2-class classification (Normal, Pneumonia)
dblock = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    # This applies the lambda function to each DataFrame row
    get_x=ColReader('orig_file_path', pref=PROJECT_PATH + '/'),
```

```
    get_y=ColReader('image_class'),
    item_tfms=Resize(224)
)

# Create DataLoaders for the 2-class classification task
dls = dblock.dataloaders(df)
print("Image classification - pneumonia or normal")
dls.show_batch(figsize=(8, 6))
```

```
Image classification - pneumonia or normal
```

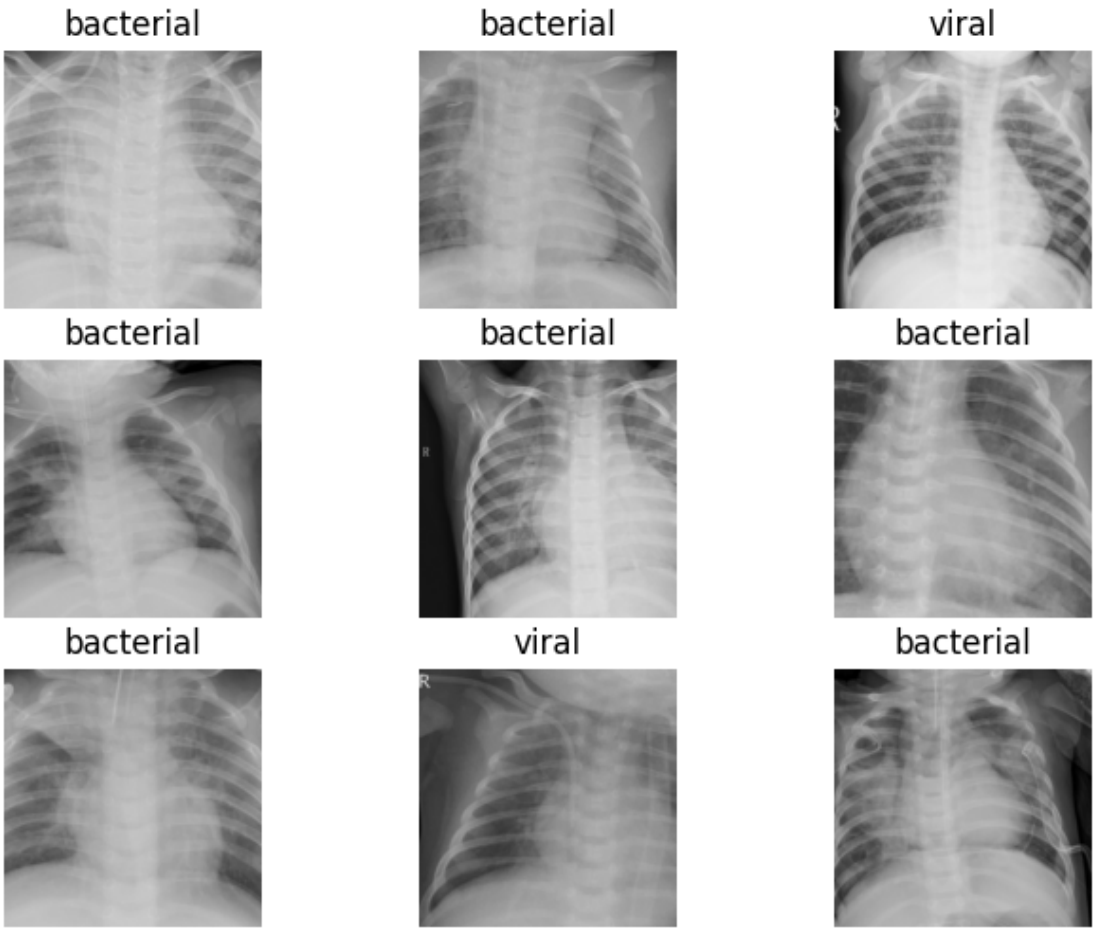


Sample images illustrate clear visual differences between normal lungs and those with pneumonia. Pneumonia images typically show areas of opacity or cloudiness, whereas normal images display clear lung fields.

[ ]:
```
# Create a DataFrame for pneumonia images
df_pneumonia = df[df['image_class'] == 'pneumonia']
```

```
# Set up a new DataBlock for 2-sub classification (Bacterial, Viral)
dblock_sub = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_x=ColReader('orig_file_path', pref=PROJECT_PATH + '/'),
    get_y=ColReader('image_subclass'),
    item_tfms=Resize(224)
)

# Create DataLoaders for the 3-class classification task
dls_sub = dblock_sub.dataloaders(df_pneumonia)

print("Pneumonia image sub classification: viral and bacterial")

dls_sub.show_batch(figsize=(8, 6))
```

```
Pneumonia image sub classification: viral and bacterial
```



Viral and bacterial pneumonia often look quite similar in X-ray images, with differences that are

subtle and harder to spot.

1.3.3 Calculate Image Statistics

We compute contrast, entropy, and signal-to-noise ratio for every X-ray to evaluate quality and compare classes.

[ ]:
```
from pathlib import Path
import seaborn as sns
import random
from collections import Counter
import os

image_classes = df['image_class'].unique()
image_subclasses = df['image_subclass'].dropna().unique()
metrics = ['contrast', 'entropy', 'signal_noise_ratio', 'mean']
metrics_file = os.path.join(DATA_PATH, 'xray_images_metrics.csv')

# Check if metrics file exists, and skip computation if it does
if os.path.exists(metrics_file):
    print(f"Loading metrics from {metrics_file}")
    df_metrics = pd.read_csv(metrics_file)
else:
    # Use all available CPUs

    print("\nNumber of available CPUs",num_cores)
    print("\n\n Compute Image Statistics...")
    results = Parallel(n_jobs=num_cores)(
        delayed(compute_metrics_with_transform)(f) for f in df['orig_file_path']
    )

    df_metrics = pd.DataFrame(results)
    #Save the DataFrame with metrics
    df_metrics.to_csv(metrics_file, index=False)
    print(f"Metrics computed and saved to {metrics_file}")

df = df.merge(df_metrics, on='orig_file_path', how='left')
df.head()
```

```
Loading metrics from
/content/drive/MyDrive/SeattleU/5100-DataIntro/Projects/image-
classification/data/xray_images_metrics.csv
```

[ ]:
```
                                orig_file_path          file_name  \
0  data/chest_xray/test/NORMAL/IM-0001-0001.jpeg  IM-0001-0001.jpeg
1  data/chest_xray/test/NORMAL/IM-0003-0001.jpeg  IM-0003-0001.jpeg
2  data/chest_xray/test/NORMAL/IM-0005-0001.jpeg  IM-0005-0001.jpeg
3  data/chest_xray/test/NORMAL/IM-0006-0001.jpeg  IM-0006-0001.jpeg
```

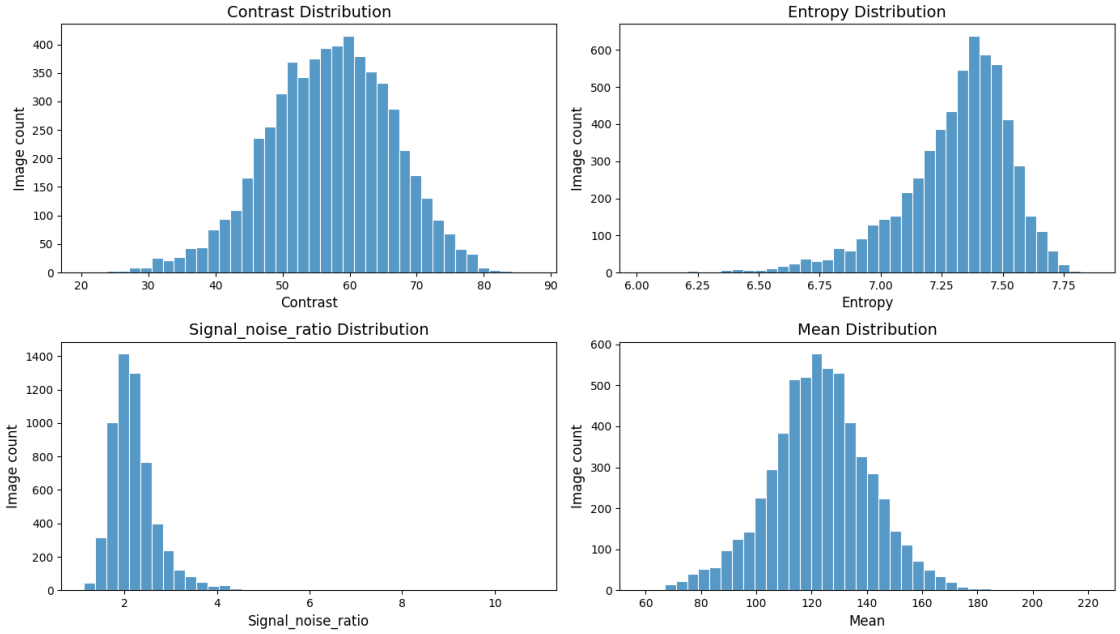
```
4  data/chest_xray/test/NORMAL/IM-0007-0001.jpeg  IM-0007-0001.jpeg

  image_class image_subclass usage_type        mean   contrast   entropy  \
0      normal           None       test  130.998608  57.102032  7.623622
1      normal           None       test  142.445303  60.668227  7.627586
2      normal           None       test  136.249559  55.404084  7.594997
3      normal           None       test  148.670080  42.324957  7.350563
4      normal           None       test  137.699779  53.966271  7.659481

   signal_noise_ratio  width  height
0            2.294115   1857    1317
1            2.347939   2111    1509
2            2.459197   2031    1837
3            3.512587   1663    1326
4            2.551590   2053    1818
```

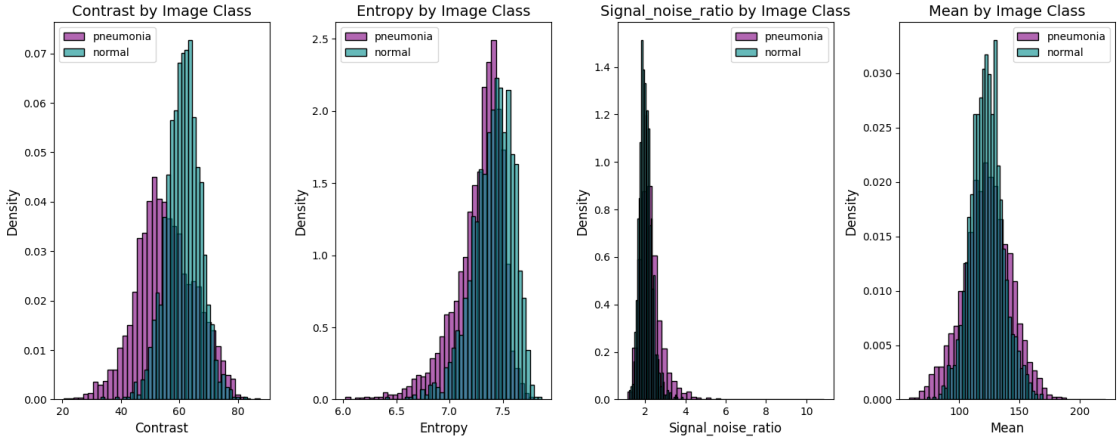
1.3.4 Plot Image Statistics

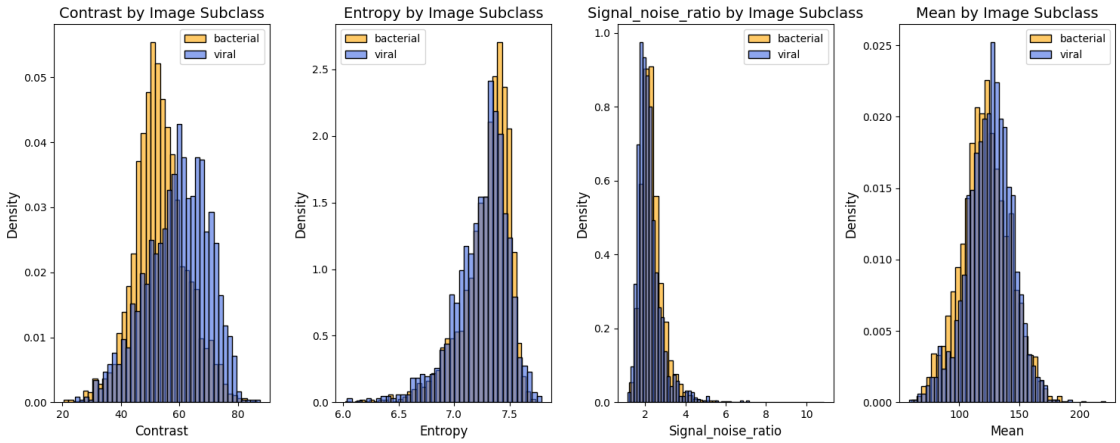
Visualize distributions to assess quality and identify class differences.

[ ]:
```python
#Plot diagrams for each metric
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 8))
for i in range(len(metrics)):
    plt.subplot(2, 2, i + 1)
    sns.histplot(
        df[metrics[i]].dropna(),
        bins=40,
        edgecolor='white'
    )
    plt.title(f'{metrics[i].capitalize()} Distribution', fontsize=14)
    plt.xlabel(metrics[i].capitalize(), fontsize=12)
    plt.ylabel('Image count', fontsize=12)
plt.tight_layout()
plt.show()


# Plot diagrams for each image-class (using helper function)
class_color_map = {
    'pneumonia': 'purple',
    'normal': 'darkcyan',
}

plot_metrics_by_class(df, metrics, ['pneumonia', 'normal'],
                      class_column='image_class',
```

```
                    color_map=class_color_map)

# Plot diagrams for each image-subclass (using helper function)
class_color_map = {
    'bacterial': 'orange',
    'viral': 'royalblue',
}

plot_metrics_by_class(df, metrics, image_subclasses,
                    class_column='image_subclass',
                    color_map=class_color_map)
```







Across image classes, the histograms show moderate contrast, entropy, and mean, while the signal-to-noise ratio is right-skewed, indicating a few exceptionally clean scans. Outliers with very low contrast or mean flag low-quality images. Comparing pneumonia to normal cases, pneumonia images exhibit slightly higher average contrast and entropy. Viral and bacterial pneumonia histograms look nearly identical across metrics, with only subtle differences.

1.3.5 Identify and Remove Low-Quality Images

Identify small images

Since our pre-trained model expects 224×224 inputs, we remove images smaller than this threshold.

[ ]:
```
## Identify smaller images than ImageNet(244)
size_threshold = 244
small_img = df[(df['width'] < size_threshold) | (df['height'] < size_threshold)]
small_img.describe()
```

[ ]:
```
              mean   contrast    entropy  signal_noise_ratio       width  \
count    52.000000  52.000000  52.000000           52.000000   52.000000
mean   132.656973  45.955423   7.002872            2.999987  470.711538
std     15.740903   8.647248   0.277183            0.780958   39.086298
min     71.142924  27.907264   6.362827            1.930794  384.000000
25%    122.345411  40.335480   6.856465            2.522469  445.000000
50%    134.517138  45.055093   7.034936            2.882159  464.500000
75%    142.962274  52.096492   7.203138            3.171104  492.500000
max    172.314810  67.581674   7.472790            5.536823  564.000000

            height
count    52.000000
mean   191.865385
std     30.943990
min    127.000000
25%    172.750000
```

```
50%    189.500000
75%    218.250000
max    242.000000
```

[ ]:
```
print("\nTotal number of images to remove",len(small_img))
df_clean = df[~df['orig_file_path'].isin(small_img['orig_file_path'])]
df_clean =  df_clean.reset_index(drop=True)
```

```
Total number of images to remove 52
```

Save dataset after removal of low quality images

[ ]:
```
df_clean.to_csv(os.path.join(DATA_PATH, 'df_clean.csv'),index=False)

df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5804 entries, 0 to 5803
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   orig_file_path      5804 non-null   object
 1   file_name           5804 non-null   object
 2   image_class         5804 non-null   object
 3   image_subclass      4221 non-null   object
 4   usage_type          5804 non-null   object
 5   mean                5804 non-null   float64
 6   contrast            5804 non-null   float64
 7   entropy             5804 non-null   float64
 8   signal_noise_ratio  5804 non-null   float64
 9   width               5804 non-null   int64
 10  height              5804 non-null   int64
dtypes: float64(4), int64(2), object(5)
memory usage: 498.9+ KB
```

[ ]:
```
df_clean.groupby(['usage_type', 'image_class', 'image_subclass'],␣
↪dropna=False)['orig_file_path'].count()
```

[ ]:
```
usage_type  image_class  image_subclass
test        normal       NaN               234
            pneumonia    bacterial         242
                         viral             148
train       normal       NaN              1349
            pneumonia    bacterial        2508
                         viral            1323
Name: orig_file_path, dtype: int64
```

1.4 Prepare and Analyze Training Data

All chest X-rays are converted to grayscale, normalized, and replicated across three channels for ResNet-50 compatibility. **Training Set 1** serves as a baseline. **Training Set 2** adds CLAHE and Hot colormap enhancements.

[ ]:
```
## Load cleaned images data
df_clean = pd.read_csv(os.path.join(DATA_PATH, 'df_clean.csv'))

# function to construct image path from image classification
def get_img_processed_dest_path(row, base_dir):
    parts = [
        base_dir,
        str(row['usage_type']),
        str(row['image_class']),
        str(row['file_name'])
    ]
    return os.path.join(*parts)
```

[ ]:
```
# Group by usage_type, image_class, and image_subclass and count the filenames,␣
 ↪including NaN values in the grouping
df_clean.groupby(['usage_type', 'image_class', 'image_subclass'],␣
 ↪dropna=False)['file_name'].count()
```

[ ]:
```
usage_type  image_class  image_subclass
test        normal       NaN               234
            pneumonia    bacterial         242
                         viral             148
train       normal       NaN              1349
            pneumonia    bacterial        2508
                         viral            1323
Name: file_name, dtype: int64
```

Note: Significant class imbalance exists and will be addressed during training.

1.4.1 Set 1 Preparation

Training Set 1 uses grayscale-converted images only, providing a baseline.

We start with clean dataset and drop metrics, which will be recalculated after grayscale conversion

[ ]:
```
df_training_set1 = df_clean.copy()

#drop metrics to recalculate them after transformation
df_training_set1.drop(columns=['contrast', 'entropy','mean',␣
 ↪'signal_noise_ratio', 'width', 'height'], inplace=True)

df_training_set1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5804 entries, 0 to 5803
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   orig_file_path  5804 non-null   object
 1   file_name       5804 non-null   object
 2   image_class     5804 non-null   object
 3   image_subclass  4221 non-null   object
 4   usage_type      5804 non-null   object
dtypes: object(5)
memory usage: 226.8+ KB
```

Compute Statistics Training for Set 1 with Grayscale Filter

[ ]:
```
df_training_set1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5804 entries, 0 to 5803
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   orig_file_path  5804 non-null   object
 1   file_name       5804 non-null   object
 2   image_class     5804 non-null   object
 3   image_subclass  4221 non-null   object
 4   usage_type      5804 non-null   object
dtypes: object(5)
memory usage: 226.8+ KB
```

Plot Training Set 1 Statistics

[ ]:
```
from pathlib import Path
import seaborn as sns
import random
from collections import Counter
import os

image_classes = df_training_set1['image_class'].unique()
image_subclasses = df_training_set1['image_subclass'].dropna().unique()
metrics = ['contrast', 'entropy', 'signal_noise_ratio', 'mean']
metrics_file_set1 = os.path.join(DATA_PATH, 'xray_images_metrics_set1.csv')

# Check if metrics file exists
if os.path.exists(metrics_file_set1):
    print(f"Loading metrics from {metrics_file_set1}")
    df_metrics_set1 = pd.read_csv(metrics_file_set1)
else:
```

```
    # Use all available CPUs

    print("\nNumber of available CPUs",num_cores)
    print("\n\n Compute Image Statistics...")
    tf = EnsureGrayscale()
    results = Parallel(n_jobs=num_cores)(
       delayed(compute_metrics_with_transform)(f, transforms=[tf])
        for f in df_training_set1['orig_file_path']
    )



    df_metrics_set1 = pd.DataFrame(results)

    # Save the DataFrame with metrics
    df_metrics_set1.to_csv(metrics_file_set1, index=False)
    print(f"Metrics computed and saved to {metrics_file_set1}")

df_training_set1 = df_training_set1.merge(df_metrics_set1, on='orig_file_path',␣
 ↪how='left')
df_training_set1.head()
```

```
Loading metrics from
/content/drive/MyDrive/SeattleU/5100-DataIntro/Projects/image-
classification/data/xray_images_metrics_set1.csv
```

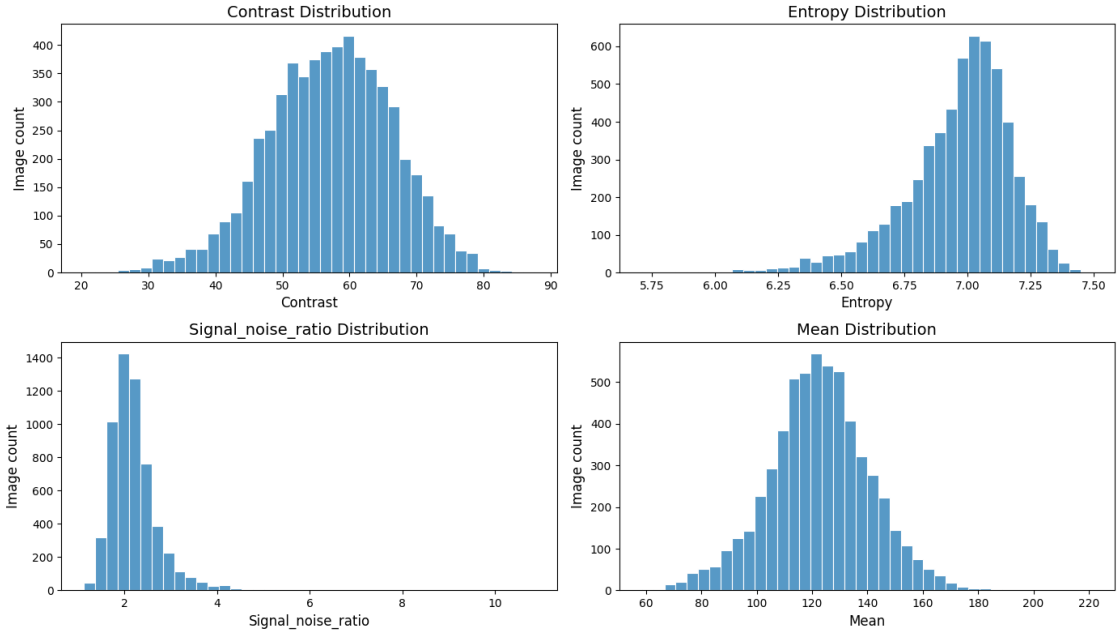
[ ]:

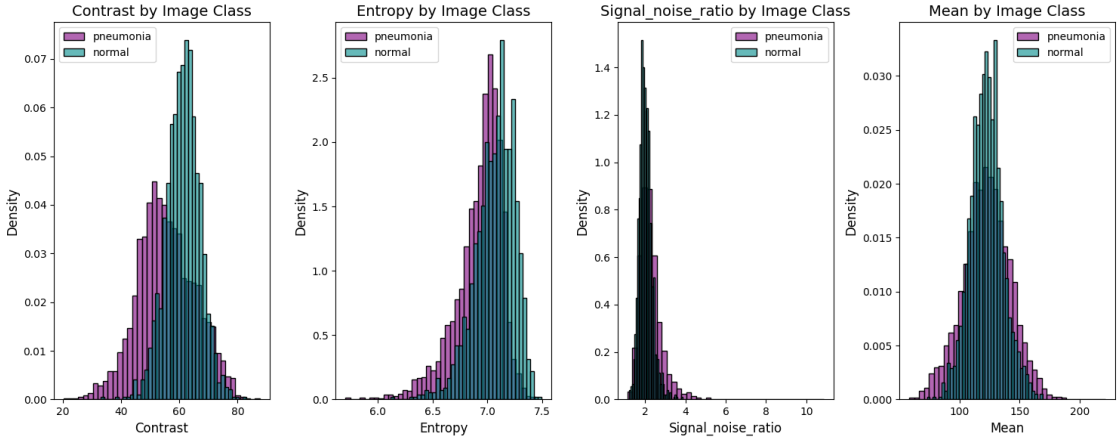
	orig_file_path	file_name	image_class	image_subclass	usage_type	mean	contrast	entropy	signal_noise_ratio	width	height
0	data/chest_xray/test/NORMAL/IM-0001-0001.jpeg	IM-0001-0001.jpeg	normal	NaN	test	130.735686	57.106981	7.265963	2.289312	1857	1317
1	data/chest_xray/test/NORMAL/IM-0003-0001.jpeg	IM-0003-0001.jpeg	normal	NaN	test	142.187266	60.665415	7.283907	2.343795	2111	1509
2	data/chest_xray/test/NORMAL/IM-0005-0001.jpeg	IM-0005-0001.jpeg	normal	NaN	test	135.989814	55.403034	7.246311	2.454555	2031	1837
3	data/chest_xray/test/NORMAL/IM-0006-0001.jpeg	IM-0006-0001.jpeg	normal	NaN	test	148.415087	42.322906	7.010260	3.506732	1663	1326
4	data/chest_xray/test/NORMAL/IM-0007-0001.jpeg	IM-0007-0001.jpeg	normal	NaN	test	137.443083	53.962186	7.308609	2.547026	2053	1818

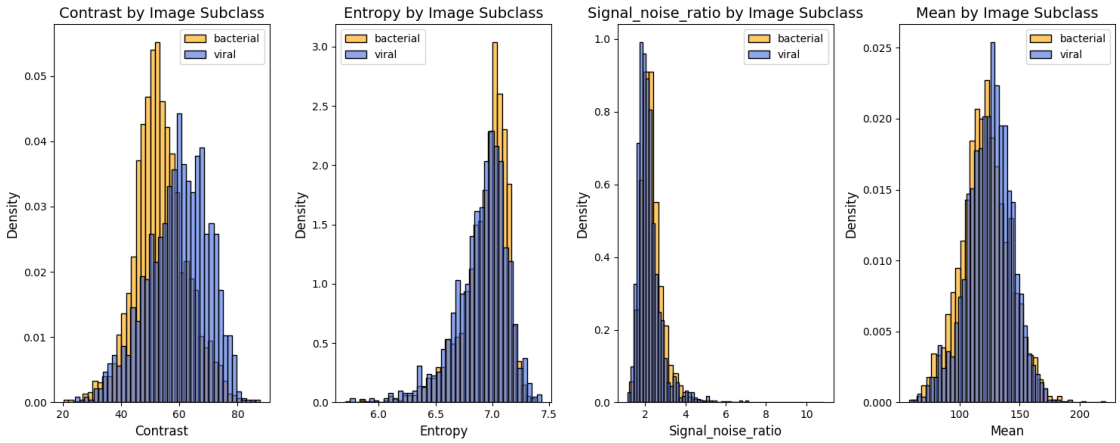
In this section, we visualize the distributions of key image metrics for Training Set 1.

[ ]:
```python
#Plot diagrams for each metric
import seaborn as sns
import matplotlib.pyplot as plt


print("Training Set 1")
plt.figure(figsize=(14, 8))
for i in range(len(metrics)):
    plt.subplot(2, 2, i + 1)
    sns.histplot(
        df_training_set1[metrics[i]].dropna(),
        bins=40,
        edgecolor='white'
    )
    plt.title(f'{metrics[i].capitalize()} Distribution', fontsize=14)
    plt.xlabel(metrics[i].capitalize(), fontsize=12)
    plt.ylabel('Image count', fontsize=12)
plt.tight_layout()
plt.show()


# Plot diagrams for each image-class (using helper function)
class_color_map = {
    'pneumonia': 'purple',
    'normal': 'darkcyan',
}

plot_metrics_by_class(df_training_set1, metrics, ['pneumonia', 'normal'],
                      class_column='image_class',
                      color_map=class_color_map)

# Plot diagrams for each image-subclass (using helper function)
class_color_map = {
    'bacterial': 'orange',
    'viral': 'royalblue',
}

plot_metrics_by_class(df_training_set1, metrics, image_subclasses,
                      class_column='image_subclass',
                      color_map=class_color_map)
```

```
Training Set 1
```


Contrast Distribution
Image count
Contrast
Entropy Distribution
Image count
Entropy
Signal_noise_ratio Distribution
Image count
Signal_noise_ratio
Mean Distribution
Image count
Mean


Contrast by Image Class
pneumonia
normal
Density
Contrast
Entropy by Image Class
pneumonia
normal
Density
Entropy
Signal_noise_ratio by Image Class
pneumonia
normal
Density
Signal_noise_ratio
Mean by Image Class
pneumonia
normal
Density
Mean



After converting to grayscale and removing low-quality scans, the class-level metric curves mostly overlap, with contrast standing out as the strongest differentiator.

Save Training Set 1 Data

```
[ ]: df_training_set1.to_csv(os.path.join(DATA_PATH, 'df_training_set1.
      ↪csv'),index=False)
```

1.4.2 Set 2 Preparation

Training Set 2 applies CLAHE for local contrast enhancement and Hot colormap to highlight patterns.

Compute Statistics with Applied CLAHE and Hot Colormap

```
[ ]: #configure image transformation classes

df_training_set2 = df_clean.copy()
metrics_file_set2 = os.path.join(DATA_PATH, 'xray_images_metrics_set2.csv')

#drop metrics to replace them with metrics for clahe
df_training_set2.drop(columns=['contrast', 'entropy','mean',␣
  ↪'signal_noise_ratio', 'width', 'height'], inplace=True)

if os.path.exists(metrics_file_set2):
    print(f"Loading metrics from {metrics_file_set2}")
    df_metrics_set2 = pd.read_csv(metrics_file_set2)
else:

    print(f"\nCPUs available: {num_cores}")

    print("\nCompute image statistics with CLAHE...")

    results = Parallel(n_jobs=num_cores)(
```

```
        delayed(compute_metrics_with_transform)(
            f,
            transforms=[EnsureGrayscale(), CLAHETransform(),␣
 ↪ColormapTransform()]
        )
        for f in df_training_set2['orig_file_path']
    )

    df_metrics_set2 = pd.DataFrame(results)
    df_metrics_set2.to_csv(metrics_file_set2, index=False)
    print(f"Metrics saved to {metrics_file_set2}")

df_training_set2 = df_training_set2.merge(df_metrics_set2, on='orig_file_path',␣
 ↪how='left')
df_training_set2.head()
```

```
Loading metrics from
/content/drive/MyDrive/SeattleU/5100-DataIntro/Projects/image-
classification/data/xray_images_metrics_set2.csv
```

[ ]:

	orig_file_path	file_name	\
0	data/chest_xray/test/NORMAL/IM-0001-0001.jpeg	IM-0001-0001.jpeg	
1	data/chest_xray/test/NORMAL/IM-0003-0001.jpeg	IM-0003-0001.jpeg	
2	data/chest_xray/test/NORMAL/IM-0005-0001.jpeg	IM-0005-0001.jpeg	
3	data/chest_xray/test/NORMAL/IM-0006-0001.jpeg	IM-0006-0001.jpeg	
4	data/chest_xray/test/NORMAL/IM-0007-0001.jpeg	IM-0007-0001.jpeg	

	image_class	image_subclass	usage_type	mean	contrast	entropy	\
0	normal	NaN	test	117.961912	70.935172	7.535515	
1	normal	NaN	test	128.261761	72.839031	7.532657	
2	normal	NaN	test	124.704067	71.393380	7.560612	
3	normal	NaN	test	136.341304	64.090955	7.405120	
4	normal	NaN	test	124.624939	70.550699	7.554565	

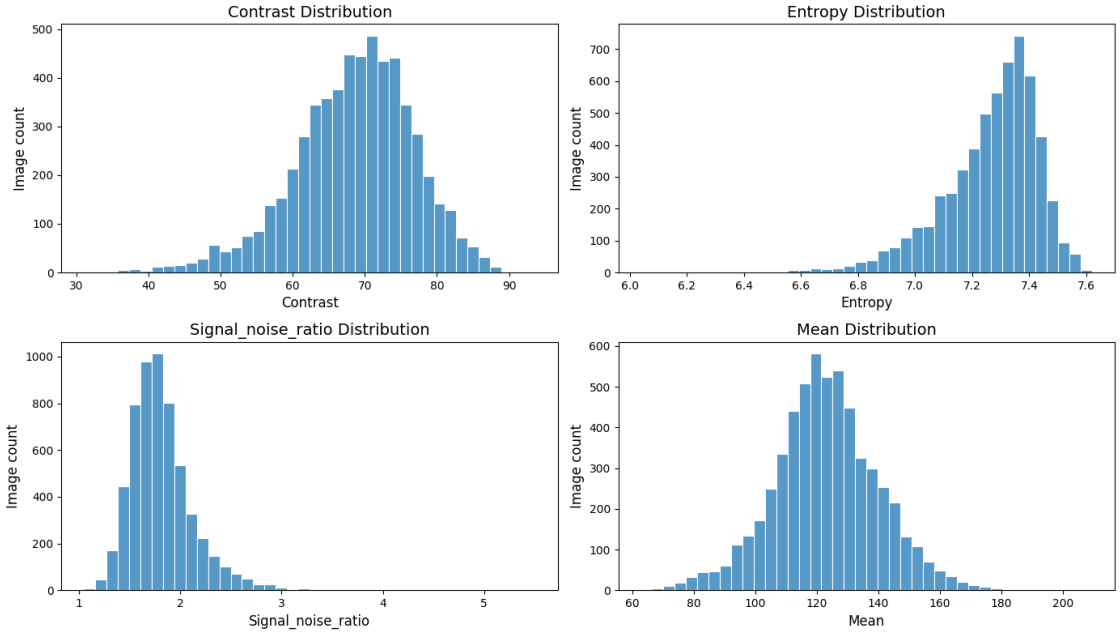
	signal_noise_ratio	width	height
0	1.662954	1857	1317
1	1.760893	2111	1509
2	1.746718	2031	1837
3	2.127310	1663	1326
4	1.766459	2053	1818

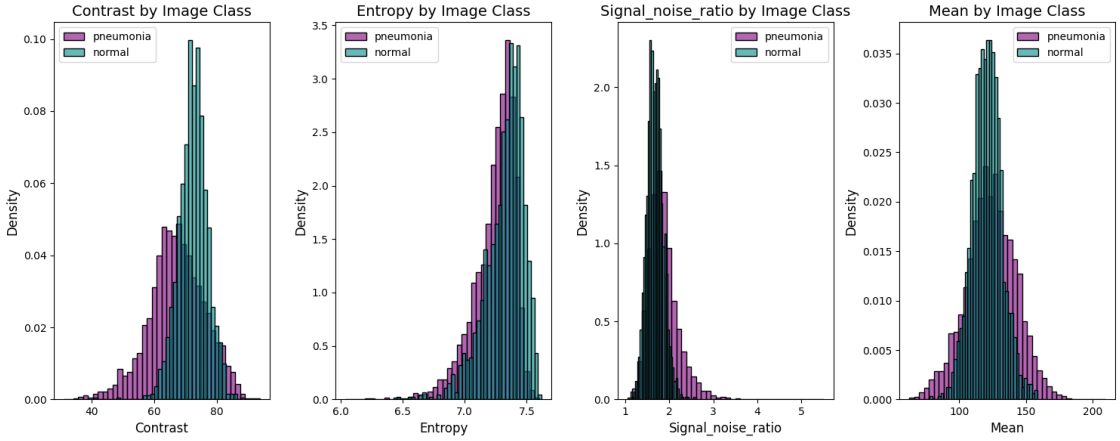
Plot Training Set 2 with Applied Enhancements Plot image metrics distributions with comparison between image classifications

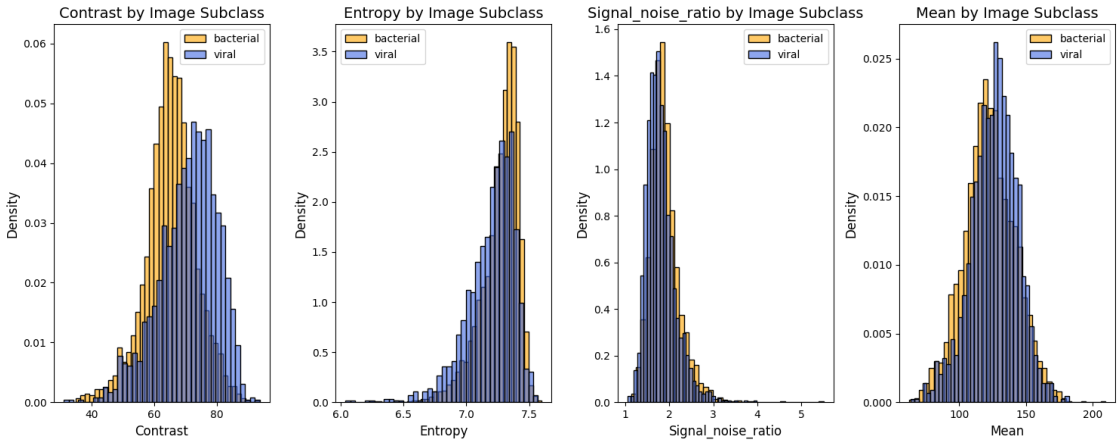
[ ]:
```
#Plot diagrams for each metric
import seaborn as sns
import matplotlib.pyplot as plt
```

```
print("Training Set 2")
plt.figure(figsize=(14, 8))
for i in range(len(metrics)):
    plt.subplot(2, 2, i + 1)
    sns.histplot(
        df_training_set2[metrics[i]].dropna(),
        bins=40,
        edgecolor='white'
    )
    plt.title(f'{metrics[i].capitalize()} Distribution', fontsize=14)
    plt.xlabel(metrics[i].capitalize(), fontsize=12)
    plt.ylabel('Image count', fontsize=12)
plt.tight_layout()
plt.show()


# Plot diagrams for each image-class (using helper function)
class_color_map = {
    'pneumonia': 'purple',
    'normal': 'darkcyan',
}

plot_metrics_by_class(df_training_set2, metrics, ['pneumonia', 'normal'],
                      class_column='image_class',
                      color_map=class_color_map)

# Plot diagrams for each image-subclass (using helper function)
class_color_map = {
    'bacterial': 'orange',
    'viral': 'royalblue',
}

plot_metrics_by_class(df_training_set2, metrics, image_subclasses,
                      class_column='image_subclass',
                      color_map=class_color_map)
```

```
Training Set 2
```


Contrast Distribution
Image count
Contrast
Entropy Distribution
Image count
Entropy
Signal_noise_ratio Distribution
Image count
Signal_noise_ratio
Mean Distribution
Image count
Mean


Contrast by Image Class
pneumonia
normal
Density
Contrast
Entropy by Image Class
pneumonia
normal
Density
Entropy
Signal_noise_ratio by Image Class
pneumonia
normal
Density
Signal_noise_ratio
Mean by Image Class
pneumonia
normal
Density
Mean



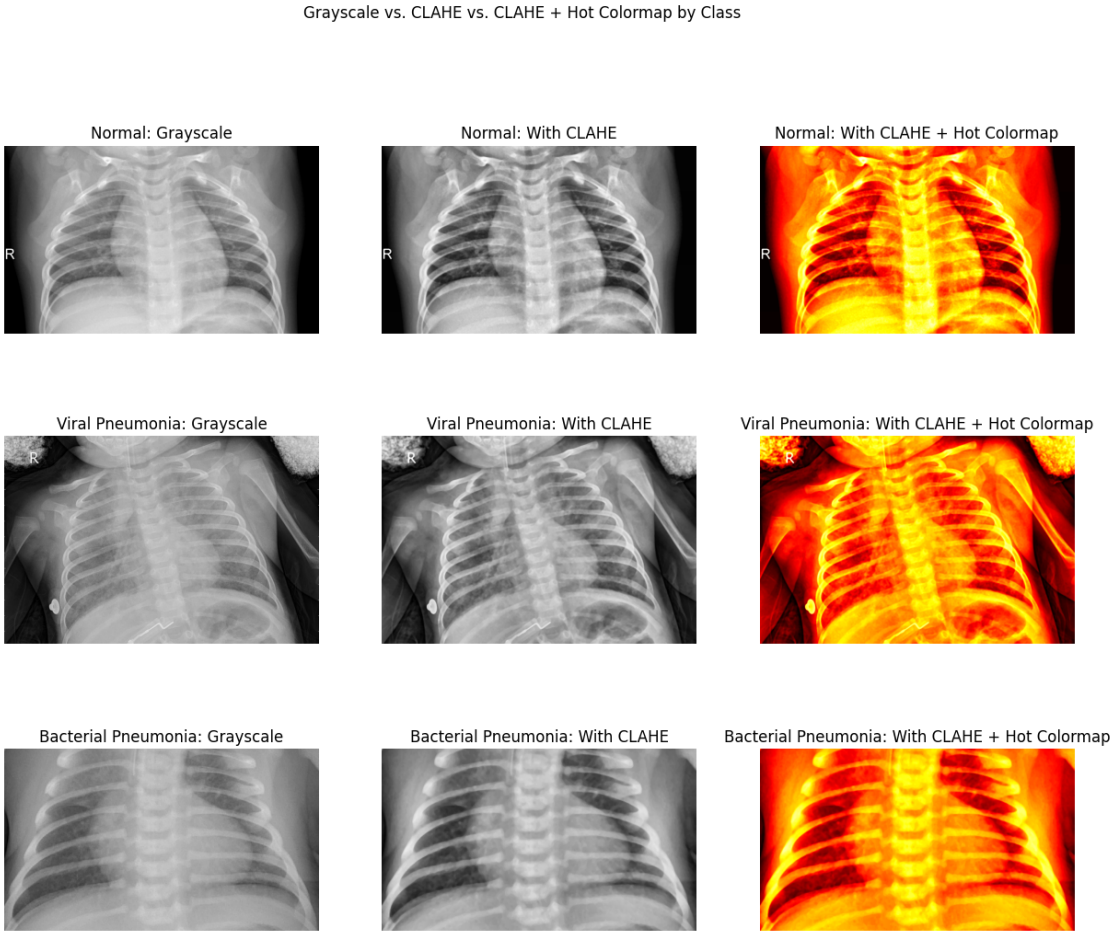
After applying CLAHE and Hot Colormap we can notice enhanced local contrast. The entropy distribution becomes slightly more concentrated and shifts toward higher values (~7.0-7.2), suggesting more uniform information content across the enhanced images.

Show Sample Images Enhanced with CLAHE and Hot Colormap CLAHE sharpens structures like ribs. Hot colormap creates a heat-map effect, highlighting abnormalities to aid feature extraction.

[ ]:
```
# Select one random normal image
sample_normal = df_training_set2[df_training_set2['image_class'] == 'normal'].
  ↪sample(1).iloc[0]

# Select one random bacterial image from pneumonia class
sample_bacterial = df_training_set2[
    (df_training_set2['image_class'] == 'pneumonia') &
    (df_training_set2['image_subclass'] == 'bacterial')
].sample(1).iloc[0]

# Select one random viral image from pneumonia class
sample_viral = df_training_set2[
    (df_training_set2['image_class'] == 'pneumonia') &
    (df_training_set2['image_subclass'] == 'viral')
].sample(1).iloc[0]

samples = [sample_normal, sample_viral, sample_bacterial]
class_names = ['Normal', 'Viral Pneumonia', 'Bacterial Pneumonia']

fig, axs = plt.subplots(3, 3, figsize=(15, 12))
fig.suptitle("Grayscale vs. CLAHE vs. CLAHE + Hot Colormap by Class")

for i, sample in enumerate(samples):
    img_path = image_absolute_path_function(sample['orig_file_path'])
```

```
    img_pil = Image.open(img_path)
    # Grayscale (EnsureGrayscale)
    img_gray, =  EnsureGrayscale().encodes((img_pil,))
    axs[i, 0].imshow(img_gray, cmap='gray')
    axs[i, 0].set_title(f"{class_names[i]}: Grayscale")
    axs[i, 0].axis('off')


    # CLAHE
    img_clahe, = CLAHETransform().encodes((img_gray,))
    axs[i, 1].imshow(img_clahe, cmap='gray')
    axs[i, 1].set_title(f"{class_names[i]}: With CLAHE")
    axs[i, 1].axis('off')

    # CLAHE + Colormap (Hot)
    img_colormap, = ColormapTransform().encodes((img_clahe,))
    axs[i, 2].imshow(img_colormap)
    axs[i, 2].set_title(f"{class_names[i]}: With CLAHE + Hot Colormap")
    axs[i, 2].axis('off')

plt.tight_layout
```

[ ]: <function matplotlib.pyplot.tight_layout(*, pad: 'float' = 1.08, h_pad: 'float | None' = None, w_pad: 'float | None' = None, rect: 'tuple[float, float, float, float] | None' = None) -> 'None'>



The visual comparison shows that CLAHE enhances local contrast, making rib structures and lung tissue boundaries more distinct across all cases. The Hot Colormap transformation provides intuitive heat-map visualization where brighter regions highlight abnormalities—the diffuse infiltrates in viral pneumonia and consolidated regions in bacterial pneumonia, improving feature extraction for deep learning models.

Save Training Set 2 Data Save training set information, so metrics can used without recalculation

```
[ ]: df_training_set2.to_csv(os.path.join(DATA_PATH, 'df_training_set2.
     ↪csv'),index=False)
```

1.5 Model Preparation and Training

We apply deep learning to classify chest X-ray images as normal, viral pneumonia, or bacterial pneumonia. Rather than building a neural network from scratch, we leverage **transfer learning** using the ResNet-50 architecture, which has been pretrained on the extensive ImageNet dataset.

This approach allows us to reuse model image classification capabilities and efficiently fine-tune the model with our x-ray images.

We train model version of ResNet-50 to distinguish both general pneumonia and its subtypes using chest X-ray images from the Kaggle Chest X-Ray Images (Pneumonia) dataset.

Our image sets are prepared with the following enhancements: - **Grayscale conversion** to standardize input. - **CLAHE preprocessing** (method) is applied to improve local contrast. - **Hot colormap application** (details) further enhances feature differentiation, making anatomical and pathological regions more visible.

We will use a two-stage ResNet-50 model pretrained on ImageNet.

- **Stage 1** detects pneumonia vs normal cases.
- **Stage 2** classifies pneumonia cases as viral or bacterial.

Training uses transfer learning, the F1-score metric, and both standard grayscale images and CLAHE-preprocessed images with Hot colormap enhancement for best results.

Note:This section uses the saved training sets created in the previous section. If your training sets are already prepared, you can start from here after running the setup

1.5.1 Training Set 1: Two-Stage Training

Set 1 uses minimally processed grayscale images for hierarchical classification.

Load Training Images Data

[ ]:
```
## Load cleaned images data
df_clean = pd.read_csv(os.path.join(DATA_PATH, 'df_clean.csv'))
df_training_set1 = pd.read_csv(os.path.join(DATA_PATH, 'df_training_set1.csv'))
```

Stage 1 Training: Normal vs. Pneumonia Binary classification optimizing F1-score to prioritize recall and minimize missed cases.

Model Training Create stratified splits for balance, then fine-tune pretrained ResNet-50 with weighted cross-entropy loss.

[ ]:
```
df_set1_stage1_train = df_training_set1[df_training_set1['usage_type'] ==␣
↪'train'].copy()
df_set1_stage1_train = df_set1_stage1_train.reset_index(drop=True)

# split train and validation data proportionally across classes with shuffle
splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
train_idx, val_idx = next(splitter.split(df_set1_stage1_train,␣
↪df_set1_stage1_train['image_class']))

df_set1_stage1_train['is_validation'] = False
df_set1_stage1_train.loc[val_idx, 'is_validation'] = True

dls_set1_stage1 = ImageDataLoaders.from_df(
```

```
    df_set1_stage1_train,
    path=PROJECT_PATH,
    fn_col='orig_file_path',
    label_col='image_class',
    valid_col='is_validation',
    item_tfms=[EnsureGrayscale(), Resize(224)],
    batch_tfms=[*aug_transforms(
        do_flip=True, max_rotate=3,
        max_zoom=1,
        max_lighting=0,
        max_warp =0.0,
        p_affine = 0.5
        ),
        Normalize.from_stats(*imagenet_stats)],
    num_workers=num_cores,
    bs=32
)

# Class weights: [normal: 1.5, pneumonia: 1.0] to handle class imbalance
weights = torch.tensor([1.2, 0.8], device=dls_set1_stage1.device)
loss_func = CrossEntropyLossFlat(
     weight=weights
     )

learn_set1_stage1 = vision_learner(
    dls_set1_stage1,
    resnet50,
    pretrained=True,
    loss_func=loss_func,
    metrics=[error_rate, F1Score(average='binary'),␣
 ↪Precision(average='binary'), Recall(average='binary')],
    cbs=[
        SaveModelCallback(monitor='f1_score', comp=np.greater,␣
 ↪with_opt=True,fname='tmp_set1_stage1_model'),
        EarlyStoppingCallback(monitor='valid_loss', patience=3) #avoid over␣
 ↪training
    ]
)

learn_set1_stage1.path = Path(MODEL_PATH)
learn_set1_stage1.model_dir = '.'
```

```
Downloading: "https://download.pytorch.org/models/resnet50-11ad3fa6.pth" to
/root/.cache/torch/hub/checkpoints/resnet50-11ad3fa6.pth

100%|      | 97.8M/97.8M [00:00<00:00, 237MB/s]
```

Here we train a ResNet-50 model pretrained on ImageNet: first 4 epochs with only the head, then

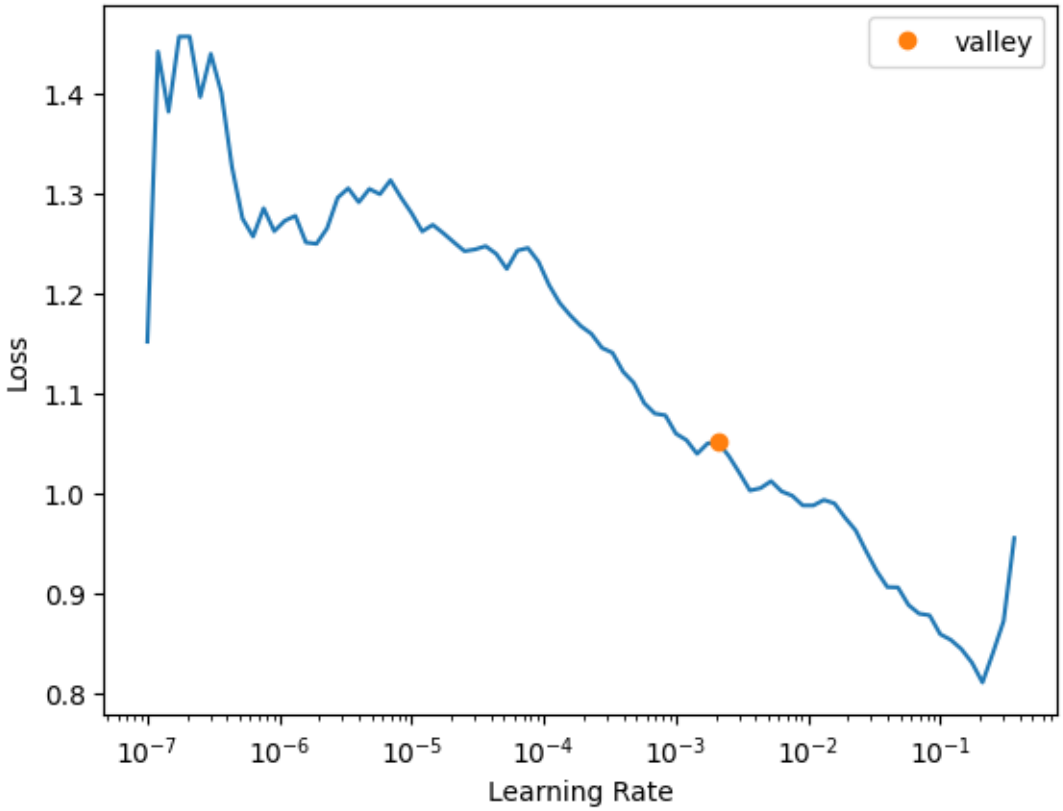
12 epochs after unfreezing the backbone.

[ ]:
```
learn_set1_stage1.lr_find()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
SuggestedLRs(valley=0.0020892962347716093)
```



[ ]:
```
#found using lr_find()
lr = 1e-4

print_learner_config(learn_set1_stage1)

# STAGE 1 - Normal vs Pneumonia
learn_set1_stage1.fit_one_cycle(4,lr)
learn_set1_stage1.unfreeze()
learn_set1_stage1.fit_one_cycle(12,lr)
```

```
=== Loss ===
Loss func   : CrossEntropyLossFlat
weight: None (no class weights set)
```

```
=== Optimizer & training ===
Weight decay (wd): None

=== Dropout in model head ===
  Dropout p=0.25
  Dropout p=0.5

=== Data & augmentations ===
Batch size : 32
Train batches: 129, Valid batches: 33
Item tfms  :
  - EnsureGrayscale(enc:1,dec:0)
  - Resize -- {'size': (224, 224), 'method': 'crop', 'pad_mode': 'reflection',
'resamples': (<Resampling.BILINEAR: 2>, <Resampling.NEAREST: 0>), 'p': 1.0}
(enc:1,dec:0)
  - ToTensor(enc:2,dec:0)
Batch tfms :
  - IntToFloatTensor -- {'div': 255.0, 'div_mask': 1}
(enc:2,dec:1)
  - Flip -- {'size': None, 'mode': 'bilinear', 'pad_mode': 'reflection',
'mode_mask': 'nearest', 'align_corners': True, 'p': 0.5}
(enc:3,dec:0)
  - Normalize -- {'mean': tensor([[[[0.4850]],

         [[0.4560]],

         [[0.4060]]]], device='cuda:0'), 'std': tensor([[[[0.2290]],

         [[0.2240]],

         [[0.2250]]]], device='cuda:0'), 'axes': (0, 2, 3)}
(enc:2,dec:2)

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Better model found at epoch 0 with f1_score value: 0.9110169491525424.
Better model found at epoch 1 with f1_score value: 0.9415807560137457.
Better model found at epoch 2 with f1_score value: 0.958922558922559.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Better model found at epoch 0 with f1_score value: 0.9544527532290958.
Better model found at epoch 1 with f1_score value: 0.9776609724047306.
Better model found at epoch 2 with f1_score value: 0.9823413996075867.
Better model found at epoch 3 with f1_score value: 0.9881889763779528.
Better model found at epoch 6 with f1_score value: 0.9960732984293194.
```

The results demonstrated high performance with an F1-score ~ 0.97.

Evaluate Training Results Reload the best model and recompute validation metrics. Final validation: F1 0.99.

[ ]:
```
interp = ClassificationInterpretation.from_learner(learn_set1_stage1)
results = learn_set1_stage1.validate()
val_loss, val_error, val_f1, val_precision, val_recall = results
print(f"Validation Accuracy: {1 - val_error:.4f}")
print(f"Validation metrics: Loss={val_loss:.4f}, F1={val_f1:.4f},␣
↪Precision={val_precision:.4f}, Recall={val_recall:.4f}")
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Better model found at epoch 0 with f1_score value: 0.9986876640419947.
Validation Accuracy: 0.9942
Validation metrics: Loss=0.0313, F1=0.9961, Precision=0.9987, Recall=0.9935
```

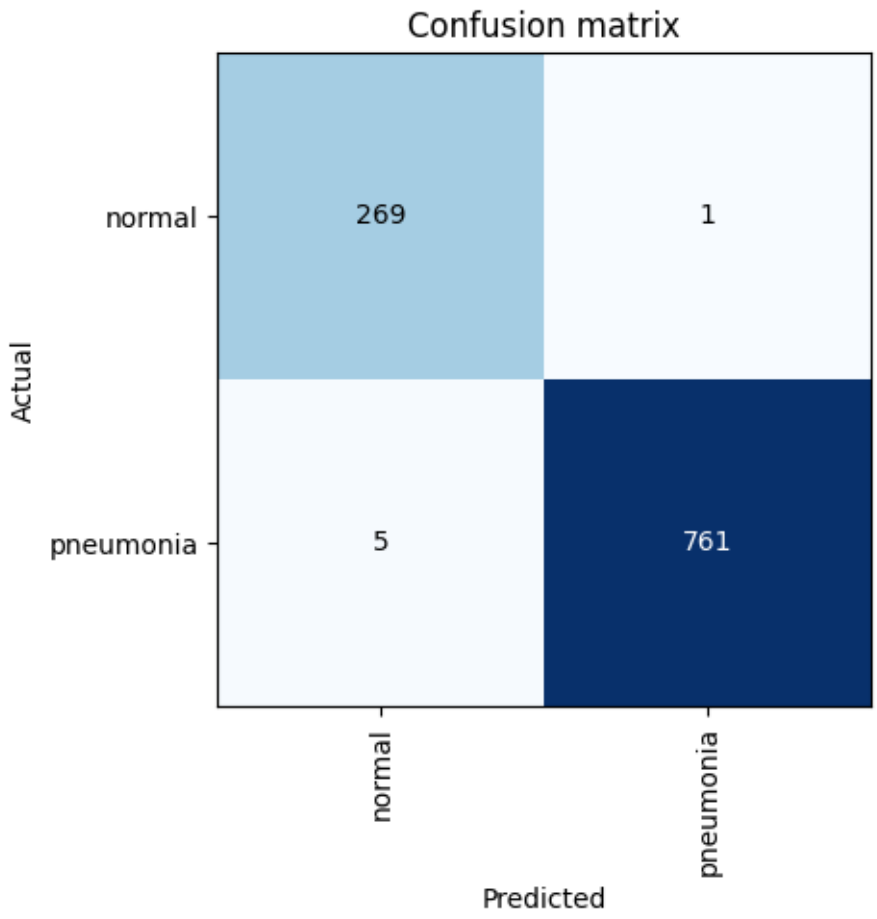
The final validation results are calculated using the best saved model weights. This approach produced F1 0.99, precision 0.99, and recall 0.99.

Confusion Matrix

Here we show how many images were correctly classified along with the false positives and false negatives in the classification.

[ ]:
```
interp.plot_confusion_matrix()
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```



The confusion matrix shows that only a handful number of images were misclassified during training and validation.

Top Losses Analysis

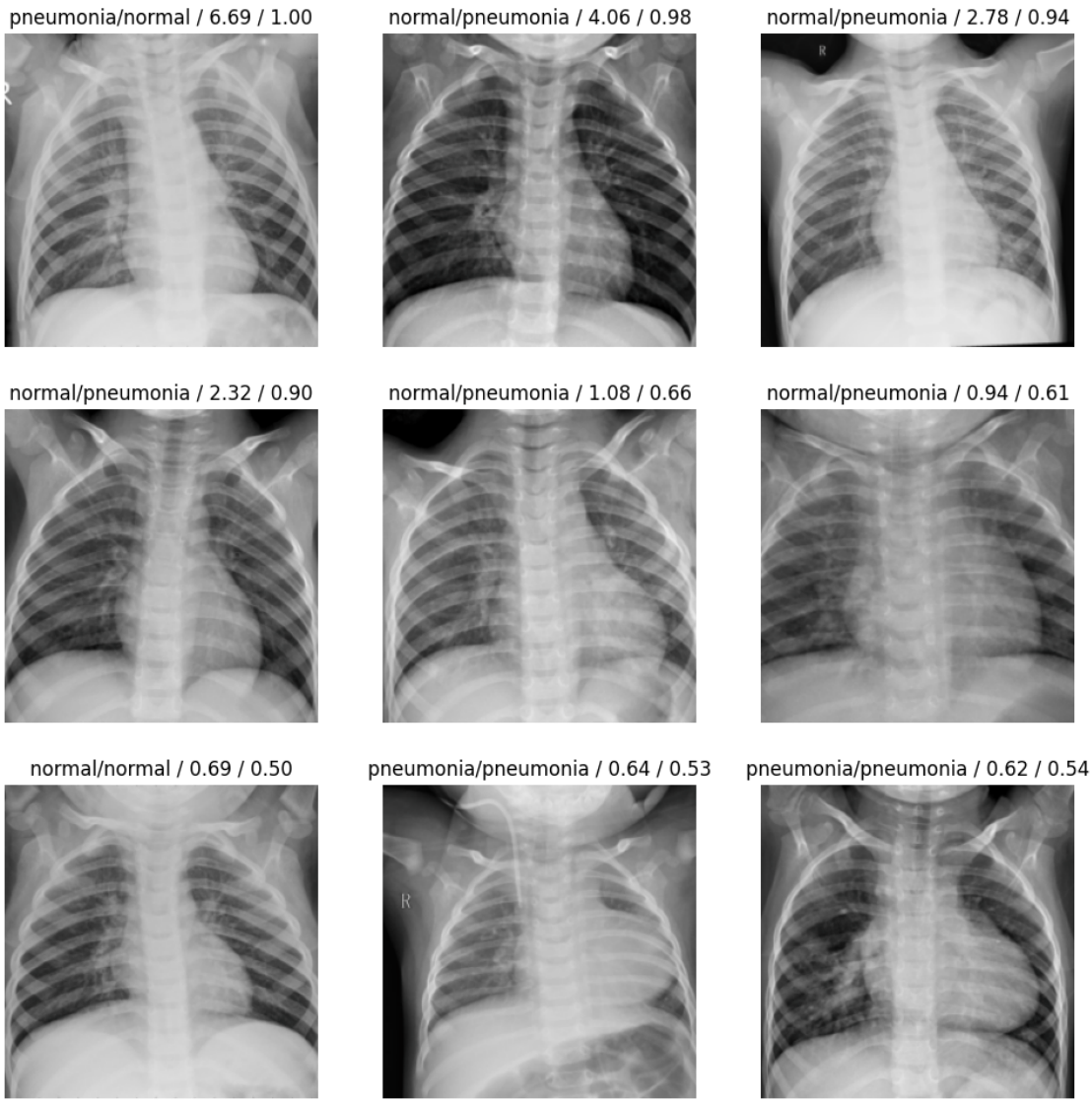
We will show misclassified images along with their predicted probabilities to determine which image types are causing problems.

[ ]:
```
plt.tight_layout()
interp.plot_top_losses(9, figsize=(12,12))
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<Figure size 640x480 with 0 Axes>
```



Some misclassifications occur with high confidence (>0.9).

Save Trained Model After training, we reload the best model checkpoint and export the trained model. Additionally, the backbone weights are saved separately for use in stage 2, allowing transfer learning for pneumonia subtype classification.

[ ]:
```
learn_set1_stage1.load('tmp_set1_stage1_model')  # Loads the best checkpoint
# Save model for inference
learn_set1_stage1.export(os.path.join(MODEL_PATH, 'set1_pneumonia_detector.
↪pkl'))
```

```
# Save backbone only for stage 2 training
torch.save(
    learn_set1_stage1.model[0].state_dict(),
    Path(MODEL_PATH) / 'set1_pneumonia_detector_backbone.pth'
)
```

Stage 2 Training: Viral vs. Bacterial Pneumonia Focuses on pneumonia images only, using Stage 1 backbone for subtype classification.

Model Training Here, we use the pretrained pneumonia detection backbone to classify pneumonia cases into bacterial or viral subtypes. Training uses a narrow range of learning rates for this more detailed classification.

[ ]:
```
# Filter dataframes for pneumonia images only
df_set1_stage2_train =␣
 ↪df_set1_stage1_train[df_set1_stage1_train['image_class']=='pneumonia'].copy()
df_set1_stage2_train = df_set1_stage2_train.reset_index(drop=True)


# split train and validation data proportionally across classes with shuffle
splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
train_idx, val_idx = next(splitter.split(df_set1_stage2_train,␣
 ↪df_set1_stage2_train['image_subclass']))

df_set1_stage2_train['is_validation'] = False
df_set1_stage2_train.loc[val_idx, 'is_validation'] = True

#prioritize hardest images
focal_loss = FastFocalLoss(alpha=0.25, gamma=2.0)

# DataLoaders for subclassification (bacterial vs viral)
dls_set1_stage2 = ImageDataLoaders.from_df(
    df_set1_stage2_train,
    path=PROJECT_PATH,
    fn_col='orig_file_path',
    label_col='image_subclass',
    valid_col='is_validation',
    item_tfms=[EnsureGrayscale(),Resize(224)],
    loss_func=focal_loss,
    batch_tfms=[*aug_transforms(
        do_flip=True, max_rotate=3,
        max_zoom=1,
        max_lighting=0,
        max_warp =0.0,
        p_affine = 0.5
        ),
```

```
        Normalize.from_stats(*imagenet_stats)],
    num_workers=num_cores,
    vocab= image_subclasses,
    bs=64
)

learn_set1_stage2 = vision_learner(
    dls_set1_stage2,
    resnet50,
    pretrained = False,
    metrics=[error_rate, F1Score(average='macro'), Precision(average='macro'),␣
 ↪Recall(average='macro')],
    cbs=[
        SaveModelCallback(monitor='f1_score', comp=np.greater,␣
 ↪with_opt=True,fname='tmp_set1_stage2_model'),
        EarlyStoppingCallback(monitor='valid_loss', patience=5)
    ]
)

learn_set1_stage2.path = Path(MODEL_PATH)
learn_set1_stage2.model_dir = Path(MODEL_PATH)

# Load the pneumonia trained model
learn_set1_stage2.model[0].load_state_dict(
    torch.load(Path(MODEL_PATH) / 'set1_pneumonia_detector_backbone.pth')
)
```

[ ]:
```
<All keys matched successfully>
```

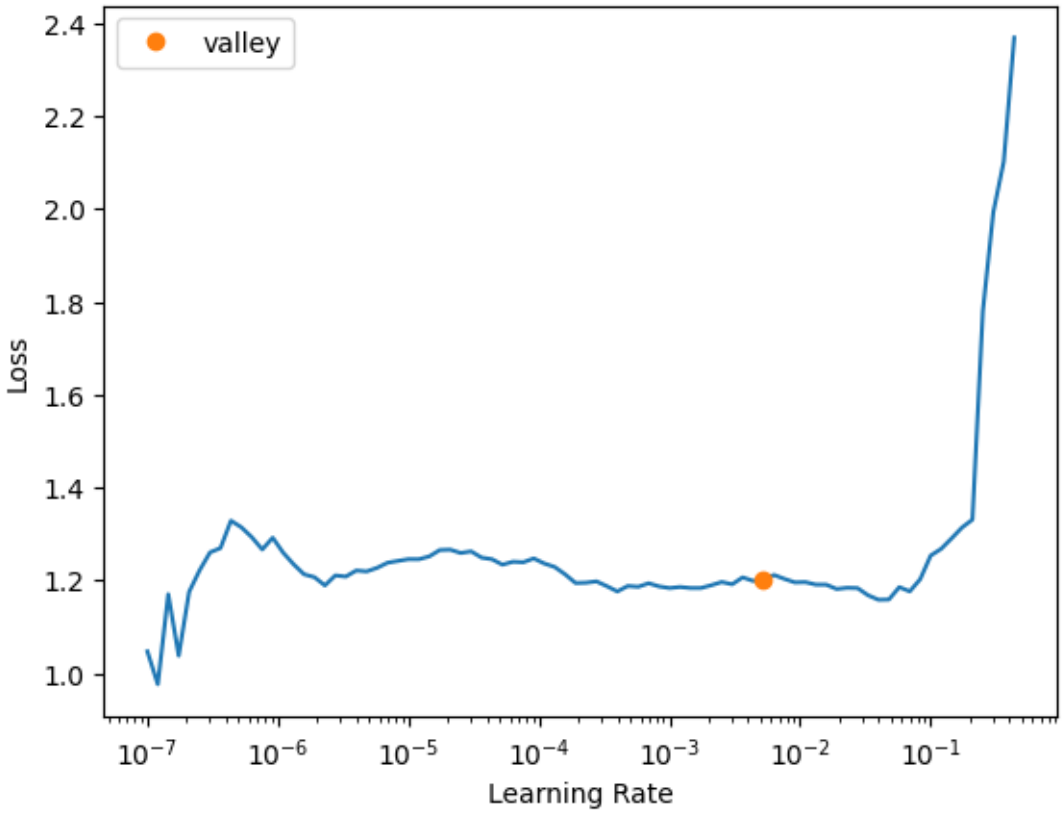
This block first probes FastAI's LR finder to grab recommended learning rate, then fine-tunes only the classifier head for six epochs at half that rate to stabilize the new layers. After unfreezing the pretrained backbone, it runs a second 12-epoch fit_one_cycle with a discriminative LR slice (from valley/100 up to valley/10) so low-level filters adapt cautiously while higher layers adjust faster.

[ ]:
```
learn_set1_stage2.lr_find()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
SuggestedLRs(valley=0.005248074419796467)
```



[ ]:
```
# Train the model
lr = 1e-3
print_learner_config(learn_set1_stage2)

learn_set1_stage2.fit_one_cycle(6, lr_max=lr)  # Fine-tune the head
learn_set1_stage2.unfreeze()
learn_set1_stage2.fit_one_cycle(12, lr_max=slice(lr/100.0, lr/10.0))  # Refine␣
↪features
```

```
=== Loss ===
Loss func   : CrossEntropyLossFlat
weight: None (no class weights set)

=== Optimizer & training ===
Weight decay (wd): None

=== Dropout in model head ===
  Dropout p=0.25
  Dropout p=0.5

=== Data & augmentations ===
```

```
Batch size : 32
Train batches: 95, Valid batches: 24
Item tfms  :
  - EnsureGrayscale(enc:1,dec:0)
  - Resize -- {'size': (224, 224), 'method': 'crop', 'pad_mode': 'reflection',
'resamples': (<Resampling.BILINEAR: 2>, <Resampling.NEAREST: 0>), 'p': 1.0}
(enc:1,dec:0)
  - ToTensor(enc:2,dec:0)
Batch tfms :
  - IntToFloatTensor -- {'div': 255.0, 'div_mask': 1}
(enc:2,dec:1)
  - Flip -- {'size': None, 'mode': 'bilinear', 'pad_mode': 'reflection',
'mode_mask': 'nearest', 'align_corners': True, 'p': 0.5}
(enc:3,dec:0)
  - Normalize -- {'mean': tensor([[[[0.4850]],

         [[0.4560]],

         [[0.4060]]]], device='cuda:0'), 'std': tensor([[[[0.2290]],

         [[0.2240]],

         [[0.2250]]]], device='cuda:0'), 'axes': (0, 2, 3)}
(enc:2,dec:2)

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Better model found at epoch 0 with f1_score value: 0.5813468277976443.
Better model found at epoch 4 with f1_score value: 0.6810203739605597.
Better model found at epoch 5 with f1_score value: 0.6830531974920973.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Better model found at epoch 0 with f1_score value: 0.6865250522288817.
Better model found at epoch 1 with f1_score value: 0.6876932025926541.
Better model found at epoch 2 with f1_score value: 0.703838852690674.
Better model found at epoch 3 with f1_score value: 0.7069023166945307.
Better model found at epoch 6 with f1_score value: 0.713081261510129.
No improvement since epoch 6: early stopping
```

The bacterial vs. viral pneumonia classifier, trained with balanced class weights and F1-score monitoring, achieved moderate performance, with best validation F1-score reaching ~0.76 and recall up to ~0.76 during training.

Evaluate Training Results After training, we reload the best saved model and recalculate metrics on the validation set for an accurate assessment.

[ ]:
```
interp = ClassificationInterpretation.from_learner(learn_set1_stage2)
results = learn_set1_stage2.validate()
val_loss, val_error, val_f1, val_precision, val_recall = results
print(f"Validation Accuracy: {1 - val_error:.4f}")
print(f"Validation metrics: Loss={val_loss:.4f}, F1={val_f1:.4f},␣
 ↪Precision={val_precision:.4f}, Recall={val_recall:.4f}")
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Better model found at epoch 0 with f1_score value: 0.7503742795119395.
Validation Accuracy: 0.7627
Validation metrics: Loss=0.5157, F1=0.7131, Precision=0.7504, Recall=0.7011
```

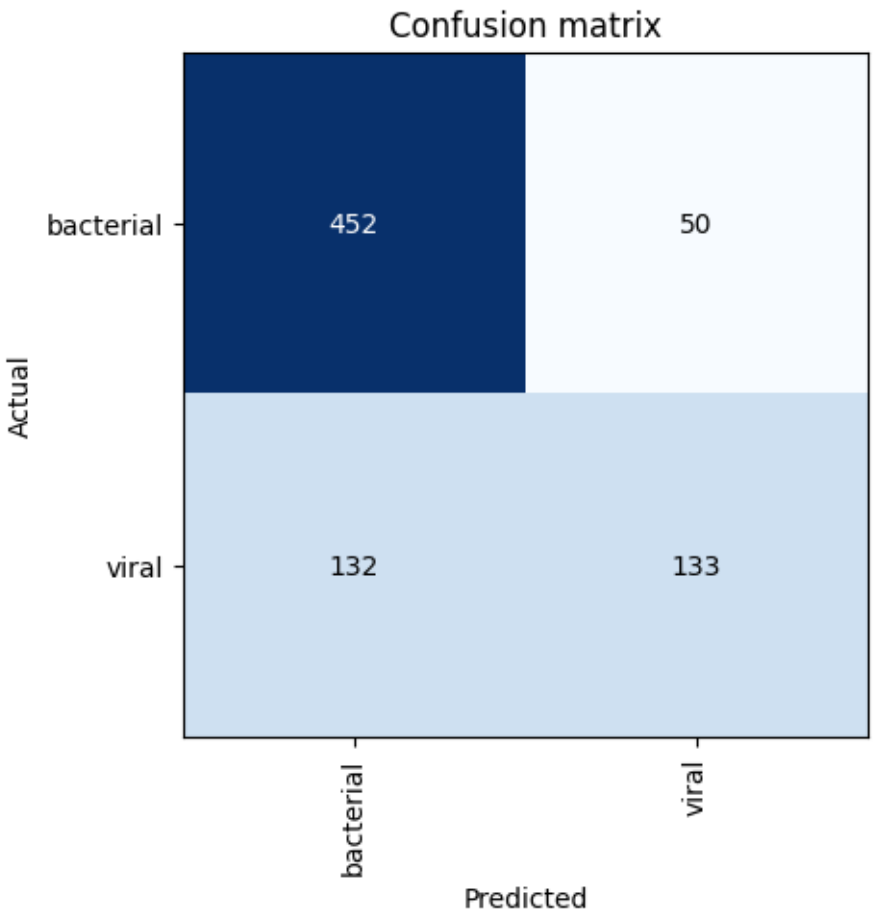
The final validation results are calculated using the best saved model weights for subtype classification. This approach produced an F1-score of approximately 0.76, with precision and recall 0.76, confirming moderate—performance for bacterial vs. viral pneumonia detection.

Confusion Matrix

Here we show how many images were correctly classified along with the false positives and false negatives for viral and bacterial pneumonia.

[ ]:
```
interp.plot_confusion_matrix()
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```



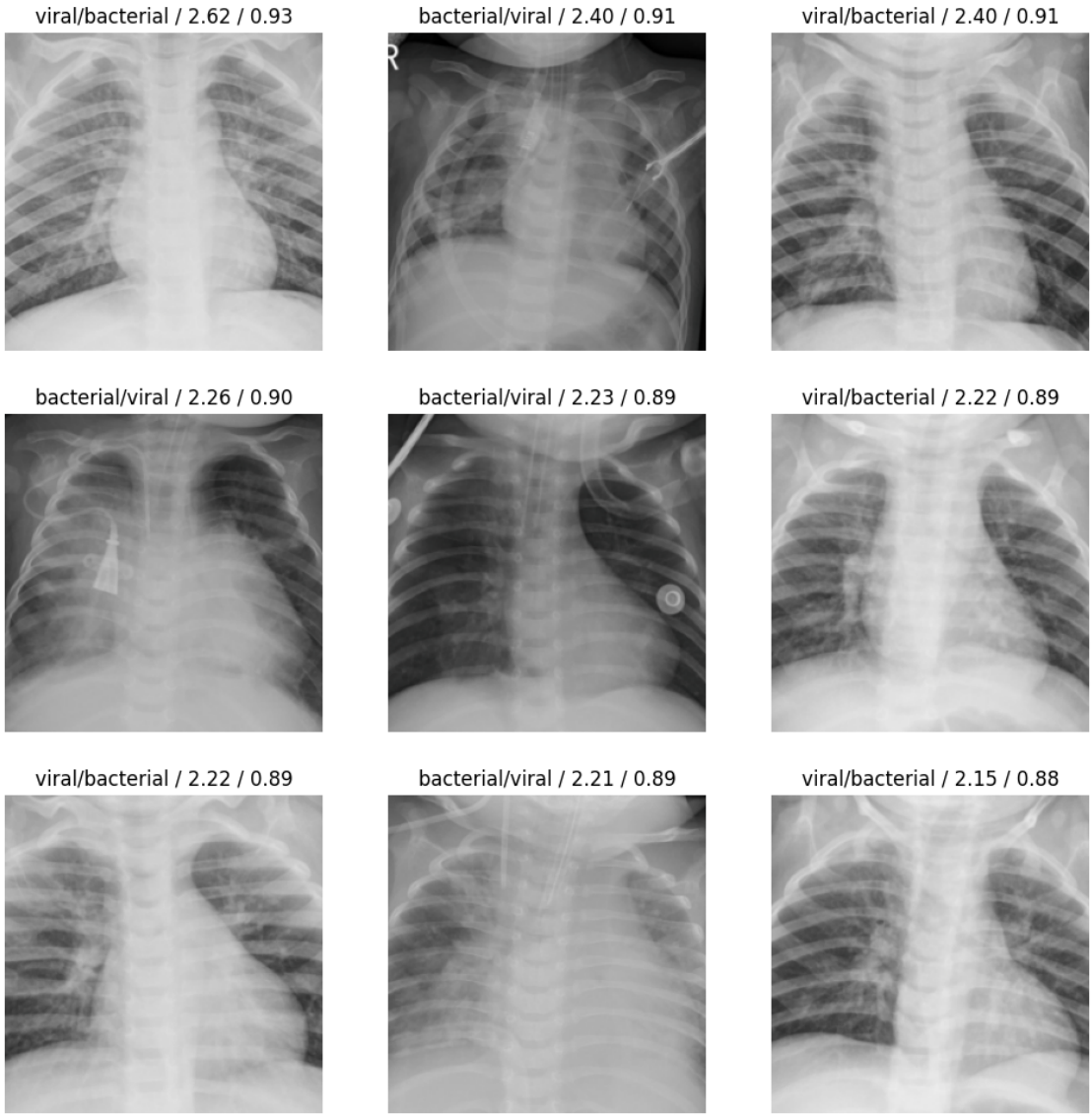
The model more reliably detects bacterial pneumonia (percentage wise), but confuses 76 viral cases as bacterial and 90 bacterial cases as viral.

Top Losses Analysis

We will show misclassified viral and bacterial pneumonia images along with their predicted probabilities to determine which image types are causing problems.

[ ]:
```
plt.tight_layout()
interp.plot_top_losses(9, figsize=(12,12))
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<Figure size 640x480 with 0 Axes>
```



Several concerning examples were misclassified with high confidence (probabilities above 0.9). These errors are most common in zoomed-in, low-contrast images.

Save Trained Model We will save trained model for deployment.

[ ]:
```
learn_set1_stage2.load('tmp_set1_stage2_model')  # Loads the best checkpoint
# Save Stage 2 model for iference
learn_set1_stage2.export(os.path.join(MODEL_PATH,␣
 ↪'set1_stage2_bacterial_viral_detector.pkl'))
```

1.5.2 Training Set 2 - Two-Stage Classification with CLAHE and Colormap Enhancements

In Training Set 2, we enhanced chest X-ray images using Contrast Limited Adaptive Histogram Equalization (CLAHE). This set will use the same hierarchical two-stage classification pipeline as Set 1.

Load training images data

```
[ ]: df_clean = pd.read_csv(os.path.join(DATA_PATH, 'df_clean.csv'))
df_training_set2 = pd.read_csv(os.path.join(DATA_PATH, 'df_training_set2.csv'))
df_set2_stage1_train =␣
  ↪df_training_set2[df_training_set2['usage_type']=='train'].copy()
df_set2_stage1_train = df_set2_stage1_train.reset_index(drop=True)
```

Stage 1 Training - Normal vs Pneumonia For the first-stage model using Training Set 2, we prioritize recall to avoid missing pneumonia cases, as pneumonia images are more common in the dataset.

Model training

```
[ ]: # Stratified split for validation
splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
train_idx, val_idx = next(splitter.split(df_set2_stage1_train,␣
  ↪df_set2_stage1_train['image_class']))

df_set2_stage1_train['is_validation'] = False
df_set2_stage1_train.loc[val_idx, 'is_validation'] = True

dblock = DataBlock(
    blocks=(ImageBlock, CategoryBlock(vocab=image_classes)),
    get_x=ColReader('orig_file_path', pref=PROJECT_PATH + '/'),
    get_y=ColReader('image_class'),
    splitter=ColSplitter('is_validation'),
    item_tfms=[EnsureGrayscale(), CLAHETransform(), ColormapTransform(),␣
  ↪Resize(224, method=ResizeMethod.Pad, pad_mode=PadMode.Zeros)],
      batch_tfms=[*aug_transforms(
        do_flip=False,
        max_rotate=0.0,
        max_zoom=1.05,
        max_lighting=0.00,
        p_lighting=0.75,
        max_warp=0.0,
        pad_mode=PadMode.Zeros,
        p_affine=0.75
    ),
    Normalize()]
)
```

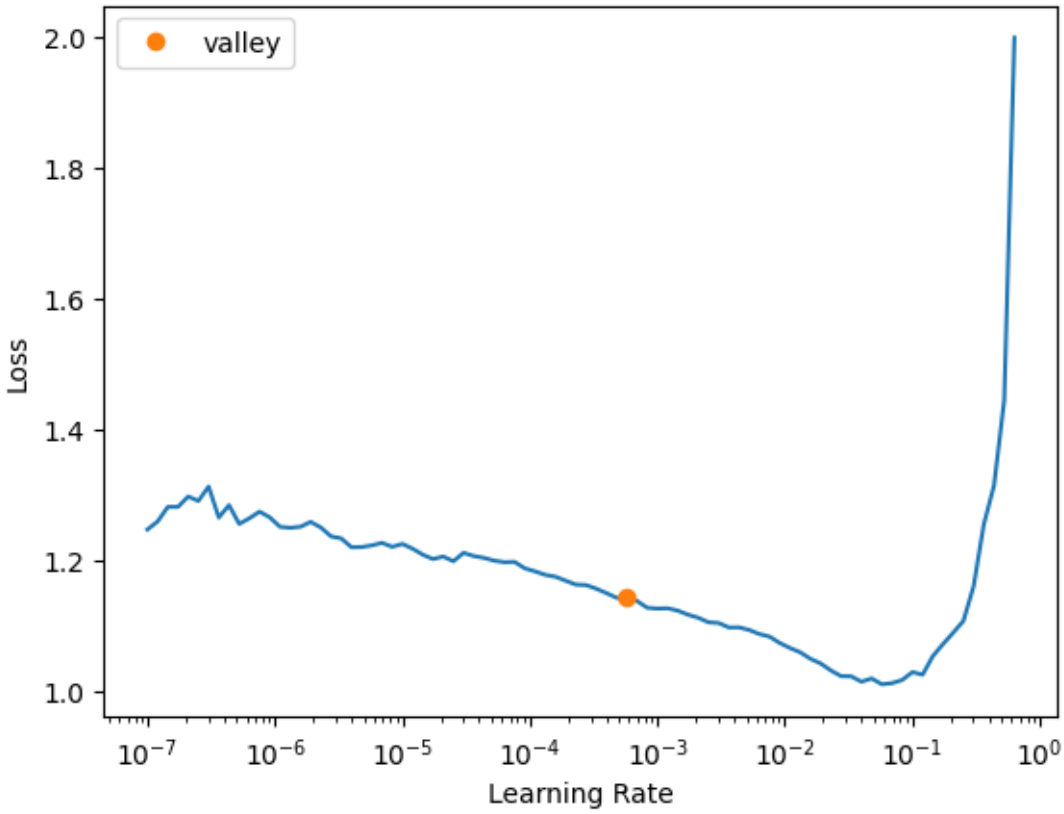
```
dls_set2_stage1 = dblock.dataloaders(
    df_set2_stage1_train,
    bs=64,
    num_workers=num_cores,
    path=PROJECT_PATH
)


weights = torch.tensor([1.0, 0.5], device=dls_set2_stage1.device)
loss_func = CrossEntropyLossFlat(
    weight=weights
)

loss_function = LabelSmoothingCrossEntropyFlat(eps=0.20)

learn_set2_stage1 = vision_learner(
    dls_set2_stage1,
    resnet50,
    pretrained=True,
    loss_func=loss_function,
    metrics=[
      error_rate,
      F1Score(average='binary'),
      Precision(average='binary'),
      Recall(average='binary'),
      RocAucBinary()
    ],
    wd=0.0005,
   # ps=0.45,
    cbs=[
        SaveModelCallback(monitor='f1_score', comp=np.greater,␣
 ↪with_opt=True,fname='tmp_set2_stage1_model'),
        EarlyStoppingCallback(monitor='valid_loss', patience=3)
    ]
)

learn_set2_stage1.model_dir = MODEL_PATH
```

[ ]:
```
learn_set2_stage1.lr_find()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
SuggestedLRs(valley=0.0005754399462603033)
```



Here we will train ResNet-50 model with CLAHE + Colormap (Hot) adjusted images

[ ]:
```
  # Based on Learning Rate Finder
lr = 5e-4

print_learner_config(learn_set2_stage1)

# STAGE 1 - Normal vs Pneumonia
learn_set2_stage1.fit_one_cycle(4,lr)
learn_set2_stage1.unfreeze()
learn_set2_stage1.fit_one_cycle(8,lr)
```

```
=== Loss ===
Loss func  : LabelSmoothingCrossEntropyFlat
  (generic loss; params from __dict__)
    __stored_args__: {'eps': 0.2, 'weight': None, 'reduction': 'mean'}
    axis: -1
    flatten: True
    floatify: False
    is_2d: True
    func: LabelSmoothingCrossEntropy()
    __module__: fastai.losses
```

```
    __doc__: None
    __annotations__: {'dump_patches': <class 'bool'>, '_version': <class 'int'>,
'training': <class 'bool'>, '_parameters': dict[str,
typing.Optional[torch.nn.parameter.Parameter]], '_buffers': dict[str,
typing.Optional[torch.Tensor]], '_non_persistent_buffers_set': set[str],
'_backward_pre_hooks': dict[int, typing.Callable], '_backward_hooks': dict[int,
typing.Callable], '_is_full_backward_hook': typing.Optional[bool],
'_forward_hooks': dict[int, typing.Callable], '_forward_hooks_with_kwargs':
dict[int, bool], '_forward_hooks_always_called': dict[int, bool],
'_forward_pre_hooks': dict[int, typing.Callable],
'_forward_pre_hooks_with_kwargs': dict[int, bool], '_state_dict_hooks':
dict[int, typing.Callable], '_load_state_dict_pre_hooks': dict[int,
typing.Callable], '_state_dict_pre_hooks': dict[int, typing.Callable],
'_load_state_dict_post_hooks': dict[int, typing.Callable], '_modules': dict[str,
typing.Optional[fastai.torch_core.Module]], 'call_super_init': <class 'bool'>,
'_compiled_call_impl': typing.Optional[typing.Callable], 'forward':
typing.Callable[…, typing.Any], '__call__': typing.Callable[…, typing.Any]}
    training: True
    _parameters: {}
    _buffers: {}
    _non_persistent_buffers_set: set()
    _backward_pre_hooks: OrderedDict()
    _backward_hooks: OrderedDict()
    _is_full_backward_hook: None
    _forward_hooks: OrderedDict()
    _forward_hooks_with_kwargs: OrderedDict()
    _forward_hooks_always_called: OrderedDict()
    _forward_pre_hooks: OrderedDict()
    _forward_pre_hooks_with_kwargs: OrderedDict()
    _state_dict_hooks: OrderedDict()
    _state_dict_pre_hooks: OrderedDict()
    _load_state_dict_pre_hooks: OrderedDict()
    _load_state_dict_post_hooks: OrderedDict()
    _modules: {}
    eps: 0.2
    weight: None
    reduction: mean
    __wrapped__: LabelSmoothingCrossEntropy()

=== Optimizer & training ===
Weight decay (wd): 0.0005

=== Dropout in model head ===
  Dropout p=0.25
  Dropout p=0.5

=== Data & augmentations ===
Batch size : 64
```

```
Train batches: 64, Valid batches: 17
Item tfms  :
  - EnsureGrayscale()
  - CLAHETransform(clip_limit=2.0, tile_grid_size=(8, 8), medianBlur=7, p=1.0)
  - ColormapTransform(colormap='HOT', p=1.0)
  - Resize -- {'size': (224, 224), 'method': 'pad', 'pad_mode': 'zeros',
'resamples': (<Resampling.BILINEAR: 2>, <Resampling.NEAREST: 0>), 'p': 1.0}
(enc:1,dec:0)
  - ToTensor(enc:2,dec:0)
Batch tfms :
  - IntToFloatTensor -- {'div': 255.0, 'div_mask': 1}
(enc:2,dec:1)
  - Zoom -- {'size': None, 'mode': 'bilinear', 'pad_mode': 'zeros', 'mode_mask':
'nearest', 'align_corners': True, 'p': 1.0}
(enc:3,dec:0)
  - Normalize -- {'mean': None, 'std': None, 'axes': (0, 2, 3)}
(enc:2,dec:2)
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Better model found at epoch 0 with f1_score value: 0.9134615384615384.
Better model found at epoch 1 with f1_score value: 0.9392117568470274.
Better model found at epoch 2 with f1_score value: 0.9581151832460733.
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Better model found at epoch 0 with f1_score value: 0.9553805774278216.
Better model found at epoch 1 with f1_score value: 0.973185088293002.
Better model found at epoch 2 with f1_score value: 0.9741207697412076.
Better model found at epoch 3 with f1_score value: 0.9774535809018567.
Better model found at epoch 4 with f1_score value: 0.9845360824742269.
Better model found at epoch 5 with f1_score value: 0.9875735775016351.
```

On training data we reached ~0.99 `F1_Score`, and ~0.08 `error_rate`.

Evaluate Training Results After training, we reload the best saved model and recalculate metrics on the validation set for an accurate assessment. We then use the confusion matrix to examine how many images are misclassified.

[ ]:
```
interp = ClassificationInterpretation.from_learner(learn_set2_stage1)
results = learn_set2_stage1.validate()
val_loss, val_error, val_f1, val_precision, val_recall,roc_acc = results
print(f"Validation Accuracy: {1 - val_error:.4f}")
print(f"Validation metrics: Loss={val_loss:.4f}, F1={val_f1:.4f},␣
  ↪Precision={val_precision:.4f}, Recall={val_recall:.4f}")
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Better model found at epoch 0 with f1_score value: 0.9895150720838795.
Validation Accuracy: 0.9817
Validation metrics: Loss=0.3563, F1=0.9876, Precision=0.9895, Recall=0.9856
```

Validation of the model reconfirms its strong results, with an F1-score, precision, and recall ~ 0.99.

Confusion Matrix

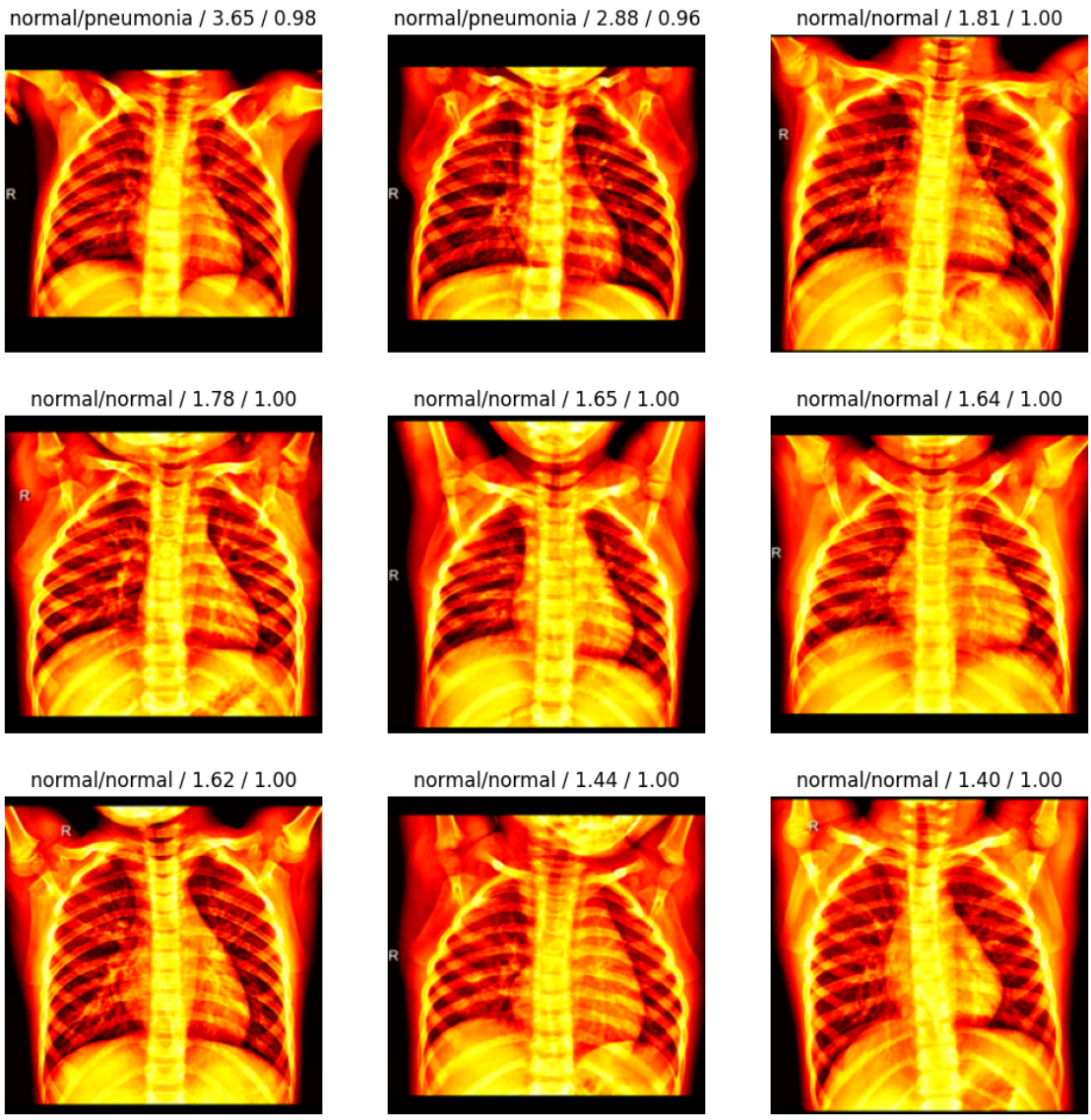
[ ]:
```
interp.plot_confusion_matrix()
```

Only 5 pneumonia case misclassified as normal (false-negative) and 3 normal cases misclassified as pneumonia.

Top Losses Analysis

[ ]:
```
plt.tight_layout()
interp.plot_top_losses(9, figsize=(12,12))
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

<Figure size 640x480 with 0 Axes>
```



The model tends to be confident in some misclassifications, but with lower loss than training set 1.

Save Trained Model After training, we reload the best model checkpoint and export the trained model. Additionally, the backbone weights are saved separately for use in stage 2, allowing transfer learning for pneumonia subtype classification.

[ ]:
```
learn_set2_stage1.load('tmp_set2_stage1_model')  # Loads the best checkpoint
# Save Stage 1 weights
```

```
learn_set2_stage1.export(os.path.join(MODEL_PATH, 'set2_pneumonia_detector.
 ↪pkl'))

# Save backbone for stage 2 training
torch.save(
    learn_set2_stage1.model[0].state_dict(),
    Path(MODEL_PATH) / 'set2_pneumonia_detector_backbone.pth'
)
```

Stage 2 Training - Pneumonia Viral vs Pneumonia Bacterial For stage 2, the model focuses on distinguishing bacterial vs. viral pneumonia among images already identified as pneumonia in stage 1. We leverage the trained backbone from stage 1. To address the substantial class imbalance between bacterial and viral images, we apply class weights in the loss function. We also use our dataset statistics for training normalization and apply minimal augumentation.

Model Training

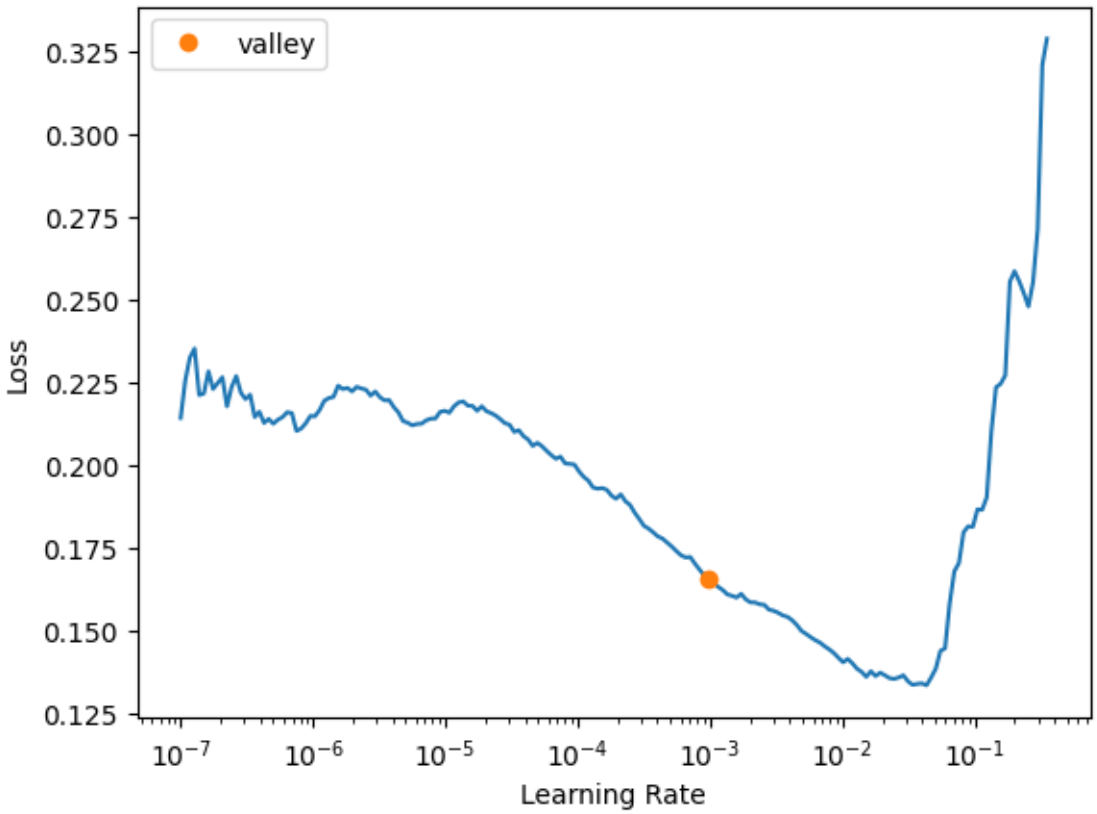
[ ]:
```
# Filter dataframes for pneumonia images only
df_set2_stage2_train =␣
 ↪df_set2_stage1_train[df_set2_stage1_train['image_class']=='pneumonia'].copy()
df_set2_stage2_train = df_set2_stage2_train.reset_index(drop=True)

# split train and validation data proportionally across subclasses with shuffle
splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
train_idx, val_idx = next(splitter.split(df_set2_stage2_train,␣
 ↪df_set2_stage2_train['image_subclass']))

df_set2_stage2_train['is_validation'] = False
df_set2_stage2_train.loc[val_idx, 'is_validation'] = True

#prioritize hardest images
focal_loss = FastFocalLoss(0.25, gamma=2.0)

# DataLoaders for subclassification (bacterial vs viral)
dblock = DataBlock(
    blocks=(ImageBlock, CategoryBlock(vocab=image_subclasses)),
    get_x=ColReader('orig_file_path', pref=PROJECT_PATH + '/'),
    get_y=ColReader('image_subclass'),
    splitter=ColSplitter('is_validation'),
    item_tfms=[EnsureGrayscale(), CLAHETransform(), ColormapTransform(),␣
 ↪Resize(224,method='pad',pad_mode='zeros')],
      batch_tfms=[*aug_transforms(
        do_flip=False,
        max_rotate=0,
        max_zoom=1.05,
        max_lighting=0.0,
        p_lighting=0.75,
```

```
        max_warp=0.0,
        pad_mode='zeros'
    ),
    Normalize.from_stats(*imagenet_stats)],

)

dls_set2_stage2 = dblock.dataloaders(
    df_set2_stage2_train,
    bs=64,
    drop_last=False,
    num_workers=num_cores,
    path=PROJECT_PATH
)

learn_set2_stage2 = vision_learner(
    dls_set2_stage2,
    resnet50,
    wd = 0.1,
    ps = 0.40,
    loss_func=focal_loss,
    pretrained=False,
    metrics=[error_rate, F1Score(average='macro'), Precision(average='macro'),␣
 ↪Recall(average='macro')],
    cbs=[
        GradientAccumulation(n_acc=2),
        SaveModelCallback(monitor='f1_score', comp=np.greater,␣
 ↪with_opt=True,fname='tmp_set2_stage2_model'),
        EarlyStoppingCallback(monitor='valid_loss', patience=5)
    ]
)

# Load the pneumonia trained model
learn_set2_stage2.model[0].load_state_dict(
    torch.load(Path(MODEL_PATH) / 'set2_pneumonia_detector_backbone.pth')
)

learn_set2_stage2.path = Path(MODEL_PATH)
learn_set2_stage2.model_dir = '.'
```

[ ]:
```
suggestion = learn_set2_stage2.lr_find(
    start_lr=1e-7,        # Start very low for focal loss
    end_lr=1,             # End lower than default (10) - your focal loss is␣
 ↪sensitive
    num_it=200,           # Double default for smoother curve
    show_plot=True
)
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```



Here, we use the pretrained pneumonia detection backbone to classify pneumonia cases into bacterial or viral subtypes. Training uses a narrow range of learning rates for this more detailed classification.

[ ]:
```
#Bacterial vs Viral Training
lr = 1e-3
print("Learning rate  ",lr)

print_learner_config(learn_set2_stage2)

learn_set2_stage2.fit_one_cycle(6, lr_max=lr/2)
learn_set2_stage2.unfreeze()
learn_set2_stage2.fit_one_cycle(12, lr_max=slice(lr/10, lr/100))
```

```
Learning rate   0.001
=== Loss ===
Loss func   : FastFocalLoss
  alpha          : 0.25
```

```
  gammma         : 2.0

=== Optimizer & training ===
Weight decay (wd): 0.1

=== Dropout in model head ===
  Dropout p=0.2
  Dropout p=0.4

=== Data & augmentations ===
Batch size : 64
Train batches: 48, Valid batches: 12
Item tfms  :
  - EnsureGrayscale()
  - CLAHETransform(clip_limit=2.0, tile_grid_size=(8, 8), medianBlur=7, p=1.0)
  - ColormapTransform(colormap='HOT', p=1.0)
  - Resize -- {'size': (224, 224), 'method': 'pad', 'pad_mode': 'zeros',
'resamples': (<Resampling.BILINEAR: 2>, <Resampling.NEAREST: 0>), 'p': 1.0}
(enc:1,dec:0)
  - ToTensor(enc:2,dec:0)
Batch tfms :
  - IntToFloatTensor -- {'div': 255.0, 'div_mask': 1}
(enc:2,dec:1)
  - Zoom -- {'size': None, 'mode': 'bilinear', 'pad_mode': 'zeros', 'mode_mask':
'nearest', 'align_corners': True, 'p': 1.0}
(enc:3,dec:0)
  - Normalize -- {'mean': tensor([[[[0.4850]],

         [[0.4560]],

         [[0.4060]]]], device='cuda:0'), 'std': tensor([[[[0.2290]],

         [[0.2240]],

         [[0.2250]]]], device='cuda:0'), 'axes': (0, 2, 3)}
(enc:2,dec:2)
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
Better model found at epoch 0 with f1_score value: 0.6165307048368014.
Better model found at epoch 2 with f1_score value: 0.7045052470171067.
Better model found at epoch 3 with f1_score value: 0.7311572065291183.
Better model found at epoch 4 with f1_score value: 0.7464292318295669.
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
Better model found at epoch 0 with f1_score value: 0.7376804957365037.
```

```
Better model found at epoch 2 with f1_score value: 0.7408950758914536.
Better model found at epoch 5 with f1_score value: 0.743884411627872.
Better model found at epoch 7 with f1_score value: 0.744716987232521.
No improvement since epoch 3: early stopping
```

The bacterial vs. viral classifier stalled at F1-score ~ 0.74, with error rate below 0.23, indicating balanced and moderately strong discrimination between the two pneumonia subtypes. For bacterial vs viral classifier with CLAHE, we can see slightly better detection than without it.

Evaluate Training Results

[ ]:
```
interp = ClassificationInterpretation.from_learner(learn_set2_stage2)
results = learn_set2_stage2.validate()
val_loss, val_error, val_f1, val_precision, val_recall = results
print(f"Validation Accuracy: {1 - val_error:.4f}")
print(f"Validation metrics: Loss={val_loss:.4f}, F1={val_f1:.4f},␣
↪Precision={val_precision:.4f}, Recall={val_recall:.4f}")
```

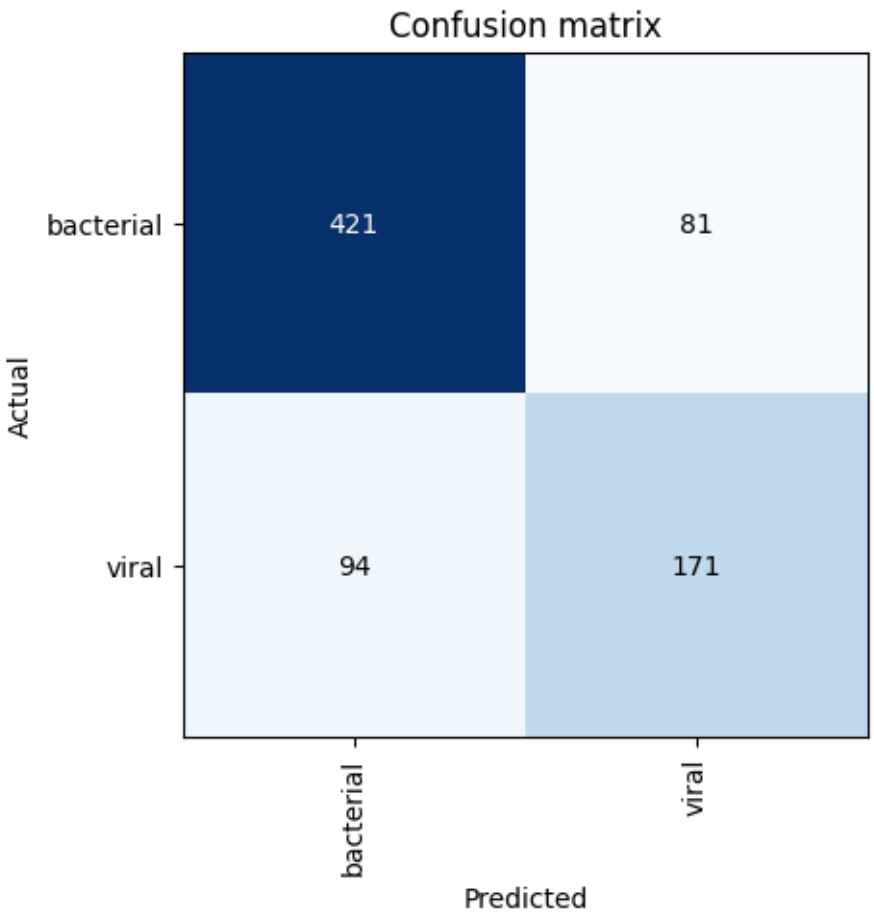
```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Better model found at epoch 0 with f1_score value: 0.7480235783633842.
Validation Accuracy: 0.7718
Validation metrics: Loss=0.0578, F1=0.7447, Precision=0.7480, Recall=0.7420
```

Validation reconfirms strong training results with `F1-score` reaching 75%.

Confusion Matrix

[ ]:
```
interp.plot_confusion_matrix()
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

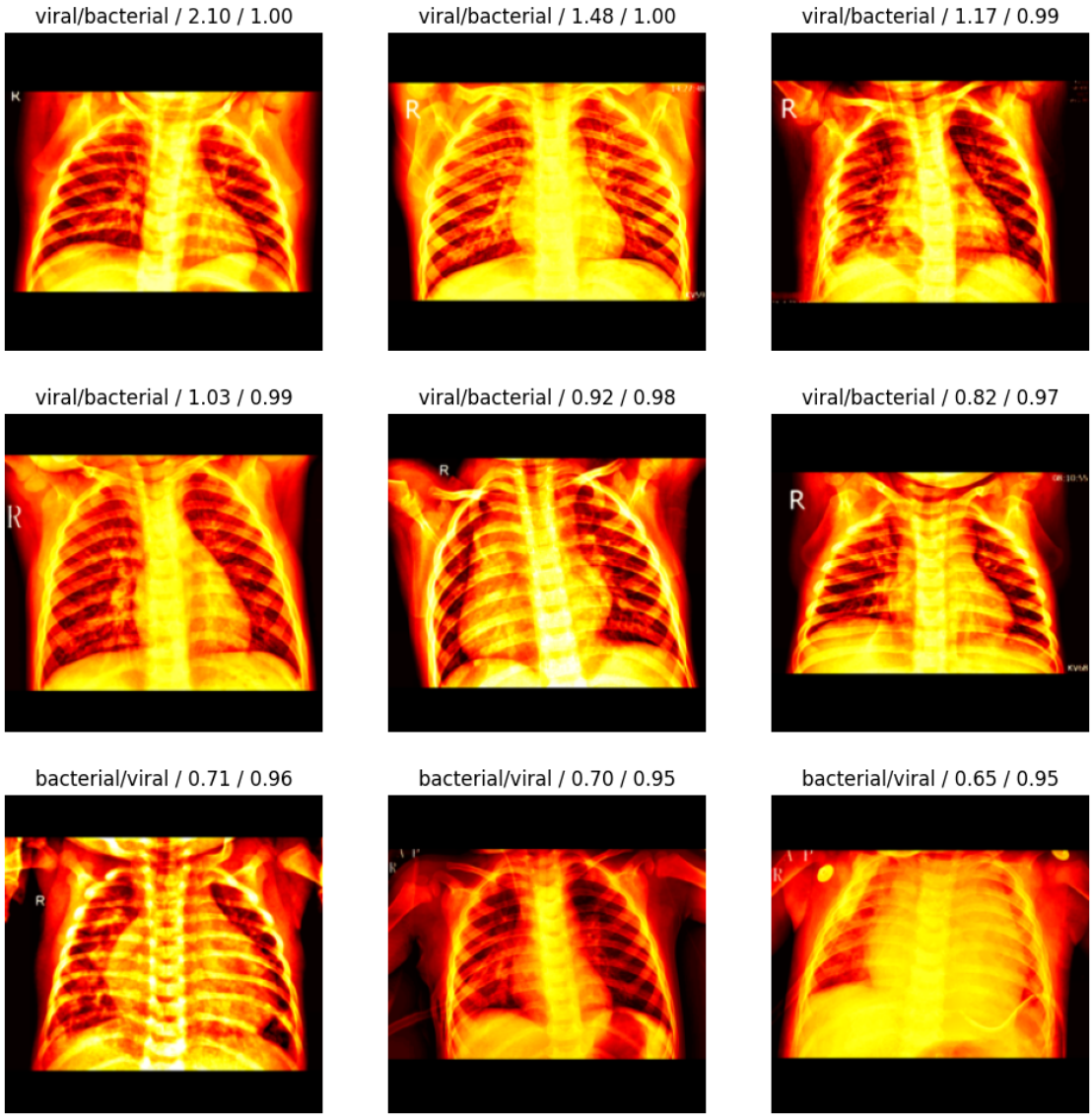


With CLAHE preprocessing, the model is better at identifying both bacterial and viral pneumonia. The number of correctly classified viral cases increased compared to no CLAHE.

Top Losses Analysis

[ ]:
```
plt.tight_layout()
interp.plot_top_losses(9, figsize=(12,12))
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<Figure size 640x480 with 0 Axes>
```



We can observe that several images are incorrectly classified with strong confidence, which means that CLAHE preprocessing negatively impacts certain images.

Save Trained Model We will save trained model, so we can test it with other images.

[ ]:
```
learn_set2_stage2.load('tmp_set2_stage2_model')  # Loads the best checkpoint
# Save Stage 2 weights
learn_set2_stage2.export(os.path.join(MODEL_PATH,␣
 ↪'set2_stage2_bacterial_viral_detector.pkl'))
```

1.6 Test Models on the Testing Sets

1.6.1 Load Training Sets data

[73]:
```
df_training_set1 = pd.read_csv(os.path.join(DATA_PATH, 'df_training_set1.csv'))
df_training_set2 = pd.read_csv(os.path.join(DATA_PATH, 'df_training_set2.csv'))
```

1.6.2 Model 1 Testing

Test Pneumonia vs Normal Model

[74]:
```
df_test1_stage1 = df_training_set1[
    (df_training_set1['usage_type'] == 'test')].copy()

# Load trained bacterial/viral model
learn_set1_stage1 = load_learner(os.path.join(MODEL_PATH,␣
 ↪'set1_pneumonia_detector.pkl'))

test_dl_set1_stage1 = learn_set1_stage1.dls.test_dl(df_test1_stage1)

all_preds, _ = learn_set1_stage1.get_preds(dl=test_dl_set1_stage1)

pred_labels = [learn_set1_stage1.dls.vocab[i] for i in all_preds.argmax(dim=1)]
max_confidence = all_preds.max(dim=1).values.cpu().numpy()
labels_set1_stage1_test = df_test1_stage1['image_class'].tolist()


# Build results DataFrame
df_test1_stage1_results = df_test1_stage1.copy()
df_test1_stage1_results['predicted_class'] = pred_labels
df_test1_stage1_results['true_class'] = labels_set1_stage1_test
df_test1_stage1_results['confidence'] = max_confidence
df_test1_stage1_results['correct'] =␣
 ↪(df_test1_stage1_results['predicted_class'] ==␣
 ↪df_test1_stage1_results['true_class']).astype(int)

# Save results
df_test1_stage1_results.to_csv(DATA_PATH+"/results_set1_stage1_test.csv",␣
 ↪index=False)
```

```
/usr/local/lib/python3.12/dist-packages/fastai/learner.py:455: UserWarning:
load_learner` uses Python's insecure pickle module, which can execute malicious
arbitrary code when loading. Only load files you trust.
If you only need to load model weights and optimizer state, use the safe
`Learner.load` instead.
  warn("load_learner` uses Python's insecure pickle module, which can execute
malicious arbitrary code when loading. Only load files you trust.\nIf you only
```

```
need to load model weights and optimizer state, use the safe `Learner.load`
instead.")
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

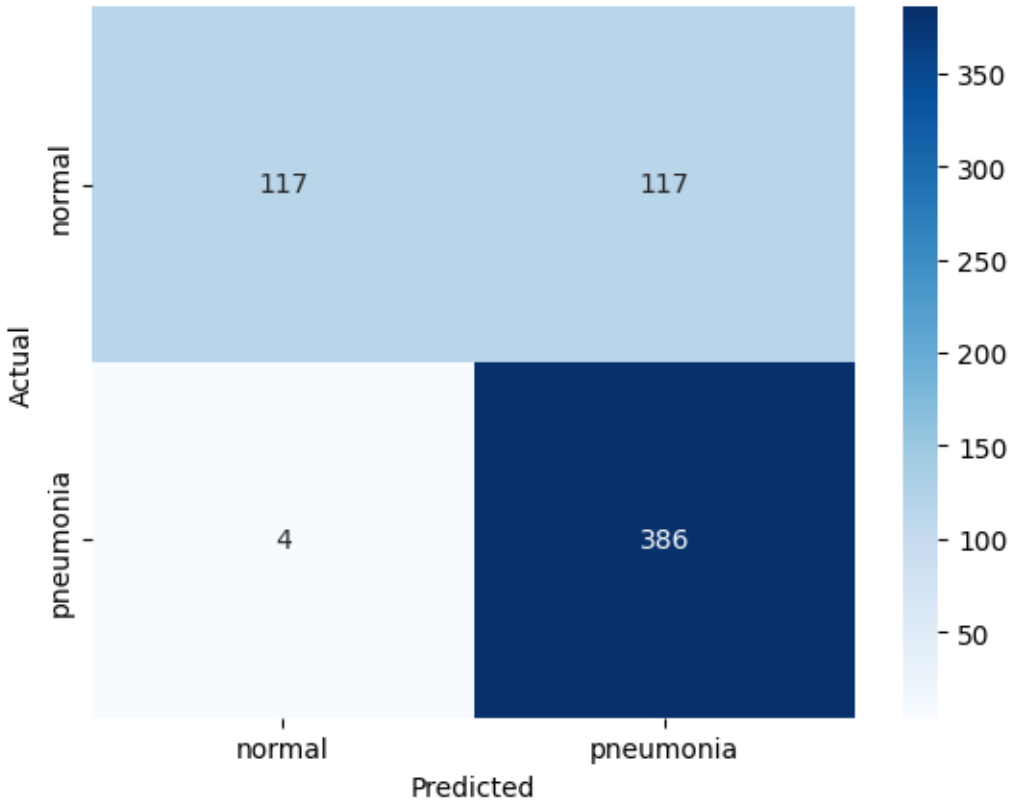
Testing Results Analysis

[75]:
```
actual_classes = df_test1_stage1_results['true_class']
predicted_classes = df_test1_stage1_results['predicted_class']
labels = ['normal','pneumonia']

accuracy = accuracy_score(actual_classes, predicted_classes)
precision = precision_score(actual_classes, predicted_classes,␣
 ↪average='binary', pos_label='pneumonia')
recall = recall_score(actual_classes, predicted_classes, average='binary',␣
 ↪pos_label='pneumonia')
f1 = f1_score(actual_classes, predicted_classes, average='binary',␣
 ↪pos_label='pneumonia')

print(f"Accuracy: {accuracy:.3f}")
print(f"Binary Precision: {precision:.3f}")
print(f"Binary Recall: {recall:.3f}")
print(f"Binary F1-score: {f1:.3f}")
```

```
Accuracy: 0.806
Binary Precision: 0.767
Binary Recall: 0.990
Binary F1-score: 0.865
```

Confusion Matrix

[76]:
```
labels = image_classes

cm = confusion_matrix(actual_classes, predicted_classes,␣
 ↪labels=df_test1_stage1_results['true_class'].unique())
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=df_test1_stage1_results['true_class'].unique(),
            yticklabels=df_test1_stage1_results['true_class'].unique())
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

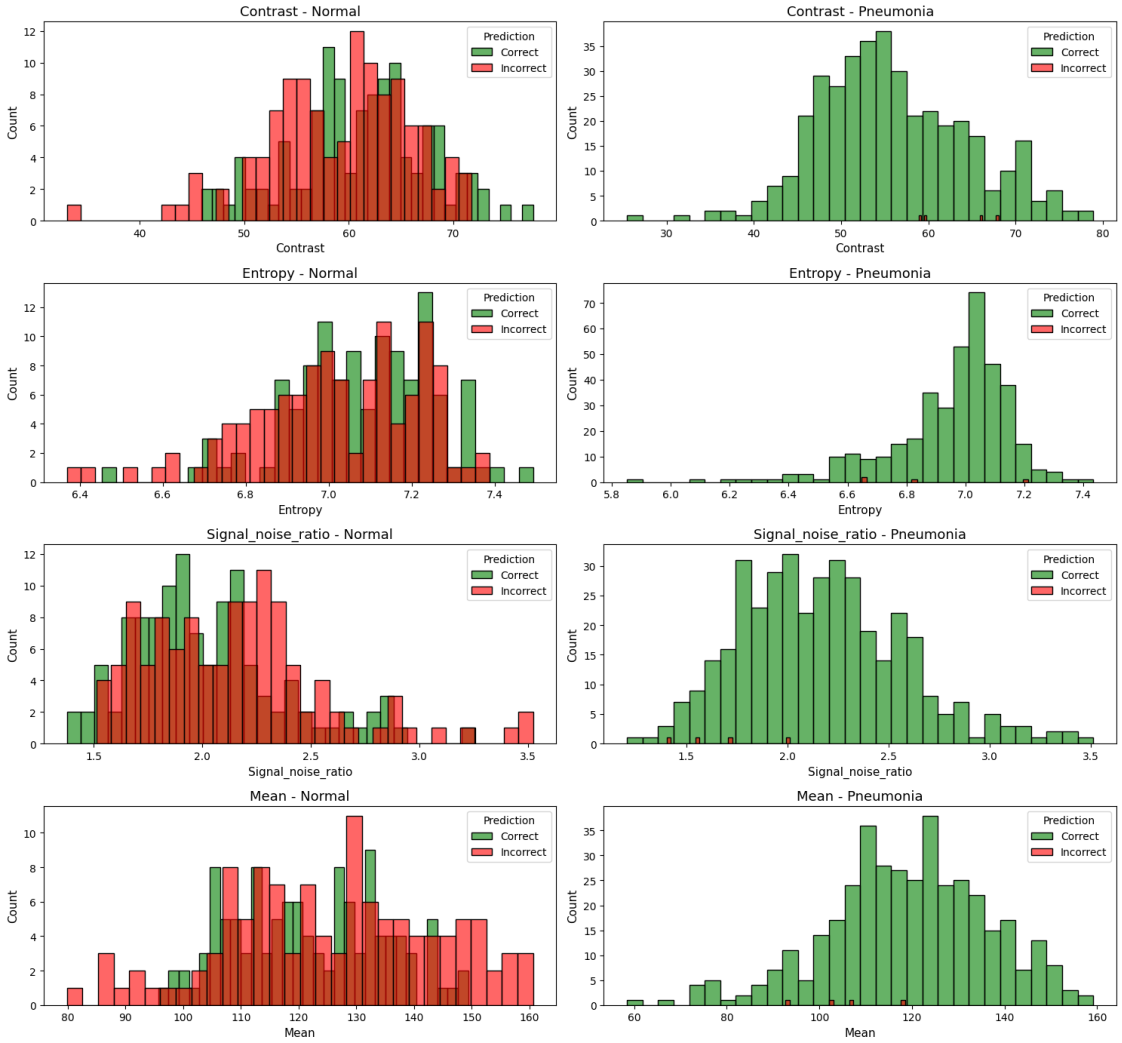


50% of normal x-rays were misclassified as pneumonia showing strong over confidence on pneumonia. In case of pneumonia cases, around 99% were correctly classified.

Evaluate Normal and Pneumonia Testing Results

[77]:
```
import seaborn as sns
import matplotlib.pyplot as plt

group_names = {1: 'Correct', 0: 'Incorrect'}
group_colors = {1: 'green', 0: 'red'}

fig, axes = plt.subplots(len(metrics), len(image_classes), figsize=(15, 14),␣
 ↪sharex=False)

for i, metric in enumerate(metrics):
    for j, cls in enumerate(image_classes):
        ax = axes[i, j]
        for corr in [1, 0]:  # correct, incorrect
            vals = df_test1_stage1_results[
                (df_test1_stage1_results['image_class'] == cls) &
                (df_test1_stage1_results['correct'] == corr)
            ][metric].dropna()
```

```
            sns.histplot(
                vals,
                bins=30,
                ax=ax,
                color=group_colors[corr],
                alpha=0.6,
                linewidth=1,
                label=group_names[corr]
            )
        ax.set_title(f"{metric.capitalize()} - {cls.capitalize()}", fontsize=13)
        ax.set_xlabel(metric.capitalize(), fontsize=11)
        ax.set_ylabel('Count', fontsize=11)
        ax.legend(title='Prediction')

plt.tight_layout()
plt.show()
```



Test Viral vs Bacterial Model

[78]:
```
from fastai.vision.all import *
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

df_test1_stage2 = df_training_set1[
    (df_training_set1['usage_type'] == 'test') &
    (df_training_set1['image_class'] == 'pneumonia')
].copy()

# Load trained bacterial/viral model
learn_set1_stage2 = load_learner(os.path.join(MODEL_PATH,␣
 ↪'set1_stage2_bacterial_viral_detector.pkl'))

test_dl_set1_stage2 = learn_set1_stage2.dls.test_dl(df_test1_stage2)

all_preds, _ = learn_set1_stage2.get_preds(dl=test_dl_set1_stage2)

pred_labels = [learn_set1_stage2.dls.vocab[i] for i in all_preds.argmax(dim=1)]
max_confidence = all_preds.max(dim=1).values.cpu().numpy()
labels_set1_stage2_test = df_test1_stage2['image_subclass'].tolist()  #␣
 ↪bacterial/viral

# Build results DataFrame
df_test1_stage2_results = df_test1_stage2.copy()
df_test1_stage2_results['predicted_class'] = pred_labels
df_test1_stage2_results['true_class'] = labels_set1_stage2_test
df_test1_stage2_results['confidence'] = max_confidence
df_test1_stage2_results['correct'] =␣
 ↪(df_test1_stage2_results['predicted_class'] ==␣
 ↪df_test1_stage2_results['true_class']).astype(int)

# Save results
df_test1_stage2_results.to_csv(DATA_PATH+"/results_set1_stage2_test.csv",␣
 ↪index=False)
```

```
/usr/local/lib/python3.12/dist-packages/fastai/learner.py:455: UserWarning:
load_learner` uses Python's insecure pickle module, which can execute malicious
arbitrary code when loading. Only load files you trust.
If you only need to load model weights and optimizer state, use the safe
`Learner.load` instead.
  warn("load_learner` uses Python's insecure pickle module, which can execute
malicious arbitrary code when loading. Only load files you trust.\nIf you only
need to load model weights and optimizer state, use the safe `Learner.load`
instead.")
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

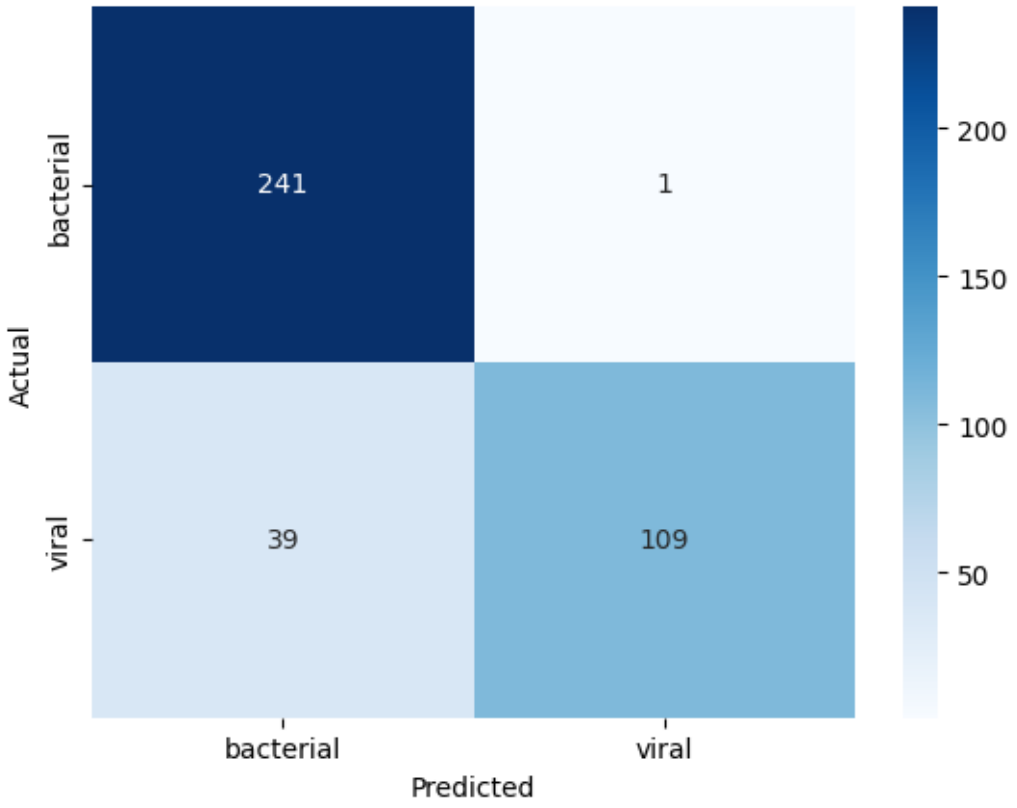
Testing Results Analysis

[79]:
```
actual_classes = df_test1_stage2_results['true_class']
predicted_classes = df_test1_stage2_results['predicted_class']
labels = sorted(list(set(actual_classes) | set(predicted_classes)))  # ensures␣
 ↪all unique labels are used

accuracy = accuracy_score(actual_classes, predicted_classes)
precision = precision_score(actual_classes, predicted_classes, average='macro',␣
 ↪labels=labels)
recall = recall_score(actual_classes, predicted_classes, average='macro',␣
 ↪labels=labels)
f1 = f1_score(actual_classes, predicted_classes, average='macro', labels=labels)

print(f"Accuracy: {accuracy:.3f}")
print(f"Macro Precision: {precision:.3f}")
print(f"Macro Recall: {recall:.3f}")
print(f"Macro F1-score: {f1:.3f}")
```

```
Accuracy: 0.897
Macro Precision: 0.926
Macro Recall: 0.866
Macro F1-score: 0.884
```

Confusion Matrix

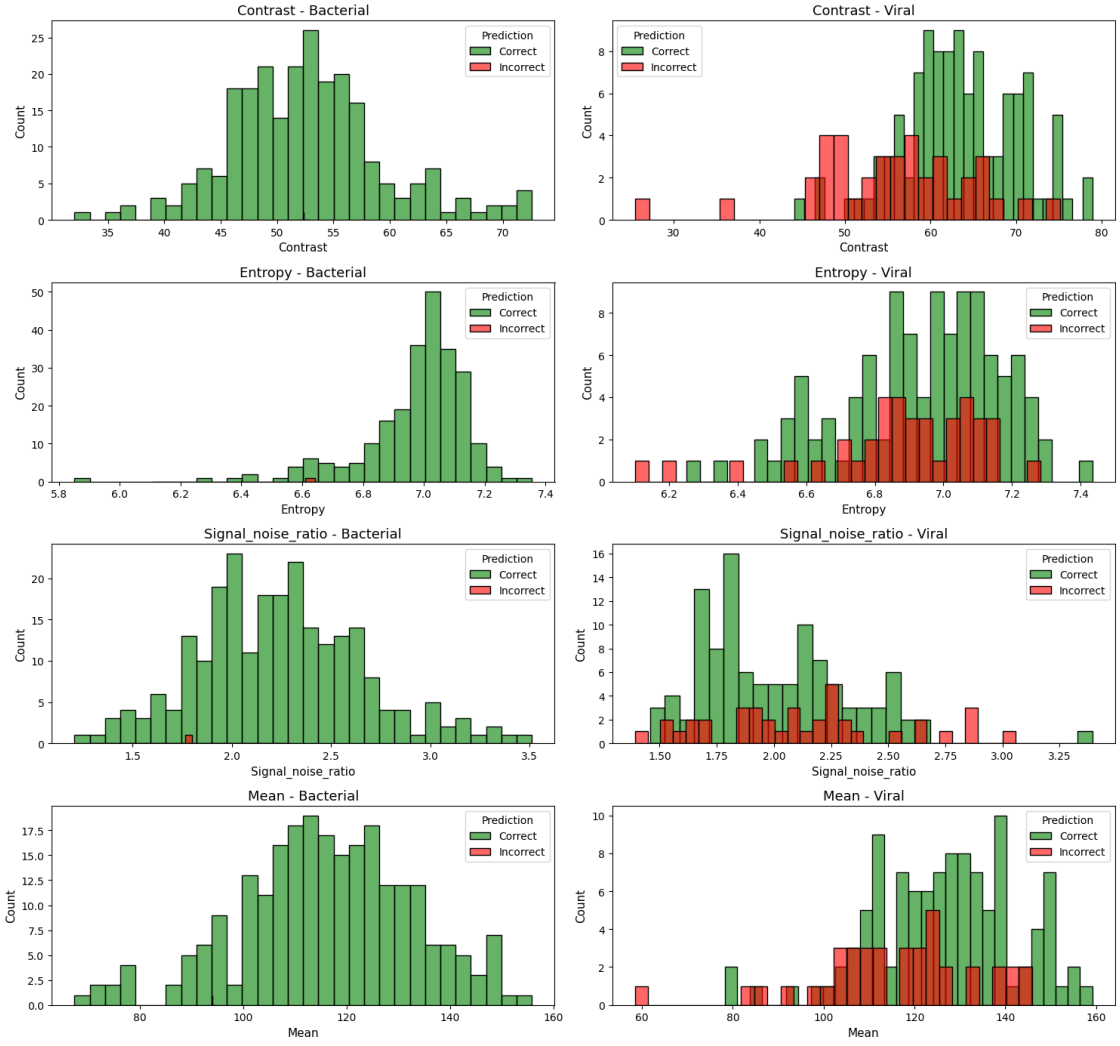
[80]:
```
# Get all present labels
labels = image_classes

cm = confusion_matrix(actual_classes, predicted_classes,␣
 ↪labels=df_test1_stage2_results['true_class'].unique())
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=df_test1_stage2_results['true_class'].unique(),
            yticklabels=df_test1_stage2_results['true_class'].unique())
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



Evaluate Viral and Bacterial Test Results We will evaluate misclassified image characteristics to identify trends

[81]:
```
import seaborn as sns
import matplotlib.pyplot as plt

group_names = {1: 'Correct', 0: 'Incorrect'}
group_colors = {1: 'green', 0: 'red'}

fig, axes = plt.subplots(len(metrics), len(image_classes), figsize=(15, 14),␣
 ↪sharex=False)

for i, metric in enumerate(metrics):
    for j, cls in enumerate(image_subclasses):
        ax = axes[i, j]
        for corr in [1, 0]:  # correct, incorrect
            vals = df_test1_stage2_results[
                (df_test1_stage2_results['image_subclass'] == cls) &
                (df_test1_stage2_results['correct'] == corr)
            ][metric].dropna()
            sns.histplot(
```

```
                vals,
                bins=30,
                ax=ax,
                color=group_colors[corr],
                alpha=0.6,
                linewidth=1,
                label=group_names[corr]
            )
        ax.set_title(f"{metric.capitalize()} - {cls.capitalize()}", fontsize=13)
        ax.set_xlabel(metric.capitalize(), fontsize=11)
        ax.set_ylabel('Count', fontsize=11)
        ax.legend(title='Prediction')

plt.tight_layout()
plt.show()
```



1.6.3 Model 2 Testing

Test Pneumonia vs Normal Model

[82]:
```
# Define the test set using df_training_set2
df_test_set2_stage1 = df_training_set2[df_training_set2['usage_type'] ==␣
 ↪'test'].copy()

# Load the previously saved stage 1 model
learn_set2_stage1 = load_learner(os.path.join(MODEL_PATH,␣
 ↪'set2_pneumonia_detector.pkl'))

# Create test dataloader
test_dl = learn_set2_stage1.dls.test_dl(df_test_set2_stage1)

# Get predictions
preds, _ = learn_set2_stage1.get_preds(dl=test_dl)
pred_labels = [learn_set2_stage1.dls.vocab[i] for i in preds.argmax(dim=1)]
max_confidence = preds.max(dim=1).values.cpu().numpy()
labels_set2_stage1_test = df_test_set2_stage1['image_class'].tolist()


# Build results DataFrame
df_test_set2_stage1_results = df_test_set2_stage1.copy()
df_test_set2_stage1_results['predicted_class'] = pred_labels
df_test_set2_stage1_results['true_class'] = labels_set2_stage1_test
df_test_set2_stage1_results['confidence'] = max_confidence
df_test_set2_stage1_results['correct'] =␣
 ↪(df_test_set2_stage1_results['predicted_class']
                                        ==␣
 ↪df_test_set2_stage1_results['true_class']).astype(int)

# Save results
df_test_set2_stage1_results.to_csv(DATA_PATH+"/results_set2_stage1_test.csv",␣
 ↪index=False)
```

```
/usr/local/lib/python3.12/dist-packages/fastai/learner.py:455: UserWarning:
load_learner` uses Python's insecure pickle module, which can execute malicious
arbitrary code when loading. Only load files you trust.
If you only need to load model weights and optimizer state, use the safe
`Learner.load` instead.
  warn("load_learner` uses Python's insecure pickle module, which can execute
malicious arbitrary code when loading. Only load files you trust.\nIf you only
need to load model weights and optimizer state, use the safe `Learner.load`
instead.")
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

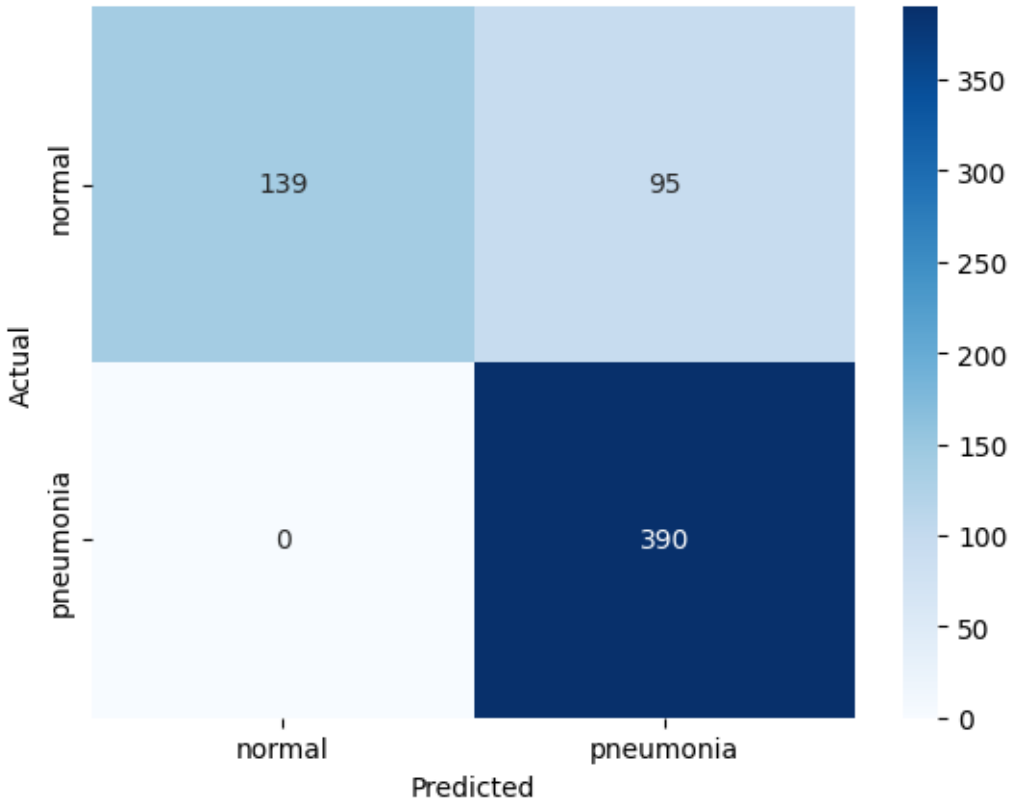
Testing Results Analysis

[83]:
```
# Calculate Metrics

accuracy = accuracy_score(labels_set2_stage1_test, pred_labels)
precision = precision_score(labels_set2_stage1_test,␣
 ↪pred_labels,average='binary', pos_label='pneumonia')
recall = recall_score(labels_set2_stage1_test, pred_labels,average='binary',␣
 ↪pos_label='pneumonia')
f1 = f1_score(labels_set2_stage1_test, pred_labels,average='binary',␣
 ↪pos_label='pneumonia')

print(f"Set 2 Stage 1 (Pneumonia Detection) Results:")
print(f"Accuracy: {accuracy:.3f}")
print(f"Binary Precision: {precision:.3f}")
print(f"Binary Recall: {recall:.3f}")
print(f"Binary F1-score: {f1:.3f}")
```

```
Set 2 Stage 1 (Pneumonia Detection) Results:
Accuracy: 0.848
Binary Precision: 0.804
Binary Recall: 1.000
Binary F1-score: 0.891
```

Confusion Matrix

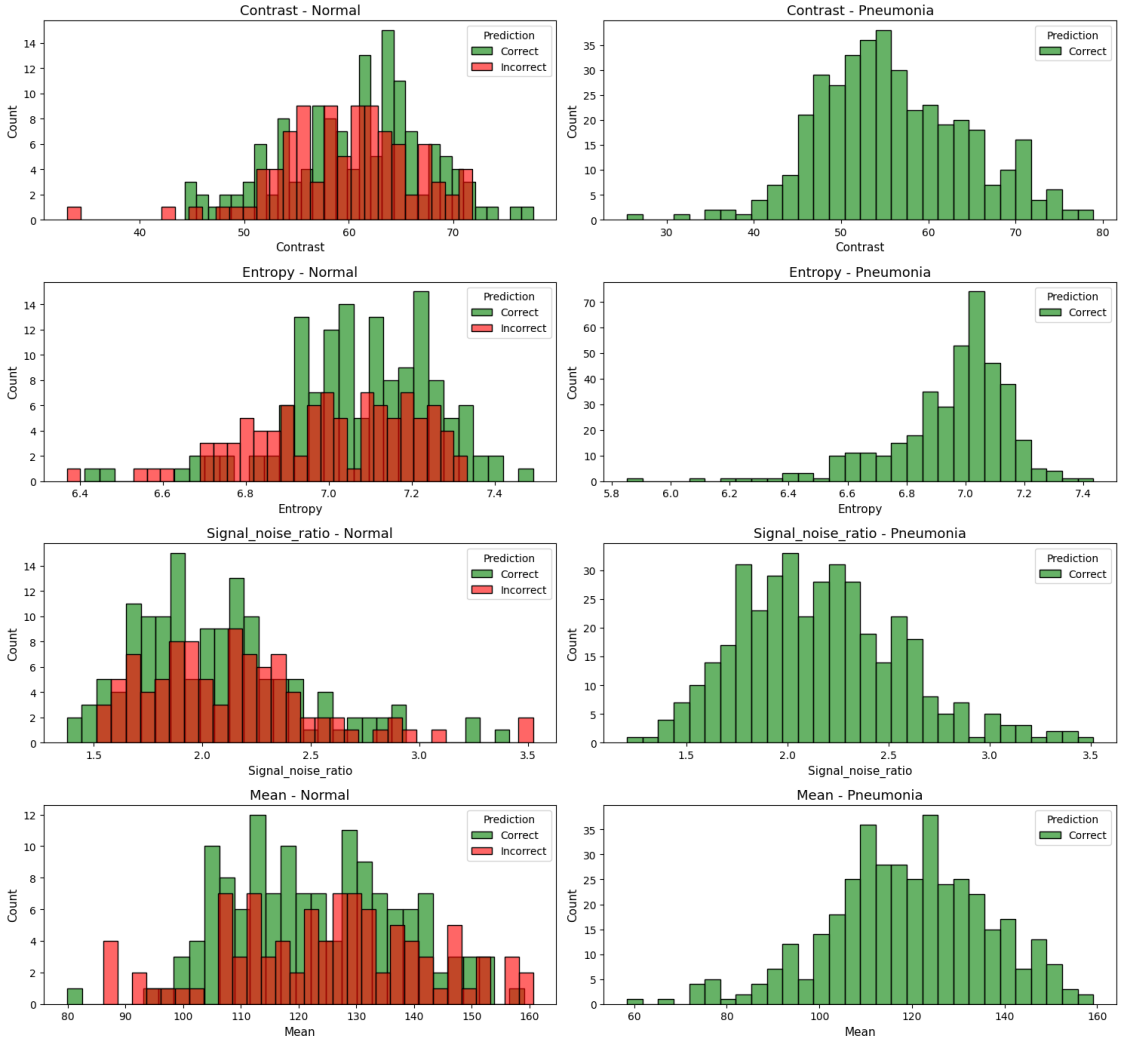
[84]:
```
labels = image_classes

cm = confusion_matrix(labels_set2_stage1_test, pred_labels, labels=
                      df_test_set2_stage1_results['true_class'].unique())
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=df_test_set2_stage1_results['true_class'].unique(),
            yticklabels=df_test_set2_stage1_results['true_class'].unique())
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



Evaluate Normal and Pneumonia Testing Results

[85]:
```
import seaborn as sns
import matplotlib.pyplot as plt

group_names = {1: 'Correct', 0: 'Incorrect'}
group_colors = {1: 'green', 0: 'red'}

fig, axes = plt.subplots(len(metrics), len(image_classes), figsize=(15, 14),␣
 ↪sharex=False)

for i, metric in enumerate(metrics):
    for j, cls in enumerate(image_classes):
        ax = axes[i, j]
        for corr in [1, 0]:  # correct, incorrect
            vals = df_test1_stage1_results[
                (df_test_set2_stage1_results['image_class'] == cls) &
                (df_test_set2_stage1_results['correct'] == corr)
            ][metric].dropna()
            sns.histplot(
                vals,
```

```
                bins=30,
                ax=ax,
                color=group_colors[corr],
                alpha=0.6,
                linewidth=1,
                label=group_names[corr]
            )
        ax.set_title(f"{metric.capitalize()} - {cls.capitalize()}", fontsize=13)
        ax.set_xlabel(metric.capitalize(), fontsize=11)
        ax.set_ylabel('Count', fontsize=11)
        ax.legend(title='Prediction')

plt.tight_layout()
plt.show()
```



Test Viral vs Bacterial Model

[86]:
```
# Define the test set for Set 2 Stage 2 (Pneumonia only)
df_test_set2_stage2 = df_training_set2[
    (df_training_set2['usage_type'] == 'test') &
    (df_training_set2['image_class'] == 'pneumonia')
].copy()

# Load the previously saved stage 2 model
learn_set2_stage2 = load_learner(os.path.join(MODEL_PATH,␣
 ↪'set2_stage2_bacterial_viral_detector.pkl'))

# Create test dataloader
test_dl = learn_set2_stage2.dls.test_dl(df_test_set2_stage2)

# Get predictions
preds, _ = learn_set2_stage2.get_preds(dl=test_dl)
pred_labels = [learn_set2_stage2.dls.vocab[i] for i in preds.argmax(dim=1)]
labels_set2_stage2_test = df_test_set2_stage2['image_subclass'].tolist()
max_confidence = preds.max(dim=1).values.cpu().numpy()
print(pred_labels)

# Build results DataFrame
df_test_set2_stage2_results = df_test_set2_stage2.copy()
df_test_set2_stage2_results['predicted_class'] = pred_labels
df_test_set2_stage2_results['true_class'] = labels_set2_stage2_test
df_test_set2_stage2_results['confidence'] = max_confidence
df_test_set2_stage2_results['correct'] =␣
 ↪(df_test_set2_stage2_results['predicted_class']
                                        ==␣
 ↪df_test_set2_stage2_results['true_class']).astype(int)

# Save results
df_test_set2_stage2_results.to_csv(DATA_PATH+"/results_set2_stage2_test.csv",␣
 ↪index=False)
```

```
/usr/local/lib/python3.12/dist-packages/fastai/learner.py:455: UserWarning:
load_learner` uses Python's insecure pickle module, which can execute malicious
arbitrary code when loading. Only load files you trust.
If you only need to load model weights and optimizer state, use the safe
`Learner.load` instead.
  warn("load_learner` uses Python's insecure pickle module, which can execute
malicious arbitrary code when loading. Only load files you trust.\nIf you only
need to load model weights and optimizer state, use the safe `Learner.load`
instead.")
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
['bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'viral', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'viral', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'viral', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'viral',
'bacterial', 'bacterial', 'viral', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'viral', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'viral', 'viral', 'viral', 'bacterial', 'viral',
'bacterial', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral',
'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral',
'viral', 'viral', 'viral', 'viral', 'bacterial', 'bacterial', 'bacterial',
'viral', 'viral', 'viral', 'bacterial', 'bacterial', 'bacterial', 'viral',
'viral', 'bacterial', 'bacterial', 'viral', 'viral', 'viral', 'bacterial',
'viral', 'bacterial', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral',
'viral', 'viral', 'bacterial', 'viral', 'viral', 'viral', 'bacterial', 'viral',
```

```
'viral', 'viral', 'bacterial', 'viral', 'viral', 'viral', 'viral', 'viral',
'bacterial', 'bacterial', 'viral', 'viral', 'bacterial', 'bacterial',
'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial', 'bacterial',
'bacterial', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral',
'viral', 'viral', 'viral', 'viral', 'bacterial', 'bacterial', 'viral', 'viral',
'viral', 'viral', 'viral', 'bacterial', 'bacterial', 'bacterial', 'viral',
'viral', 'viral', 'viral', 'bacterial', 'viral', 'viral', 'viral', 'bacterial',
'viral', 'viral', 'viral', 'bacterial', 'bacterial', 'viral', 'viral', 'viral',
'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral',
'bacterial', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral',
'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral', 'viral',
'bacterial', 'viral', 'viral', 'bacterial', 'viral']
```

Testing Results Analysis

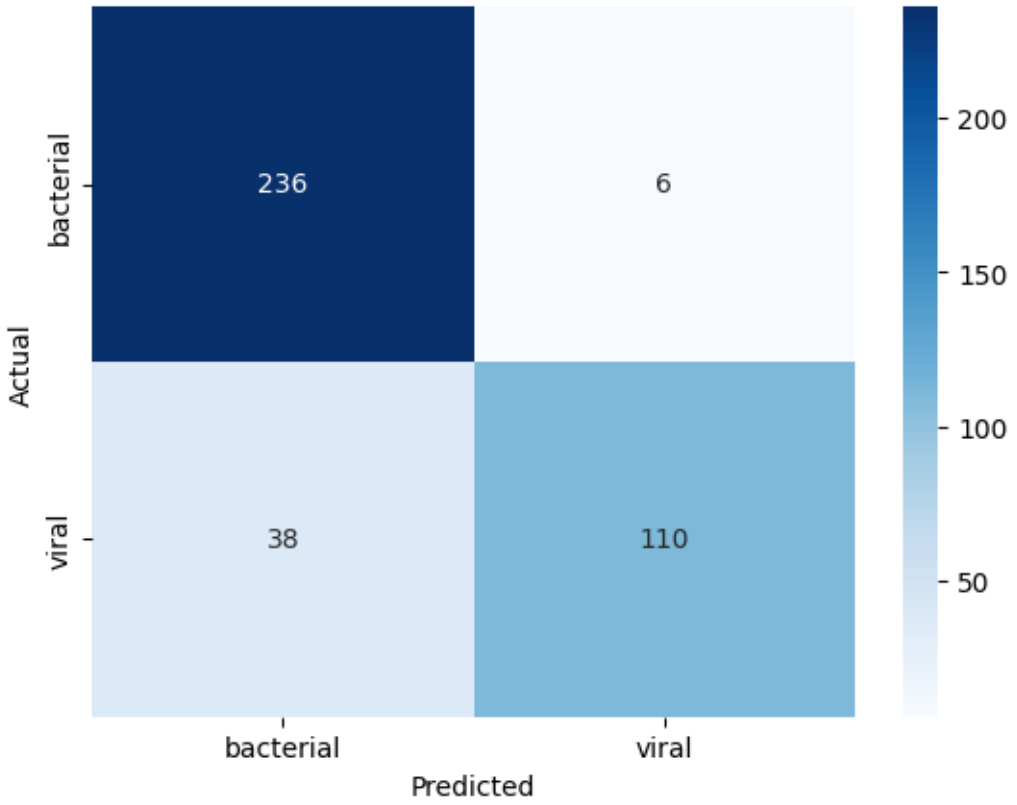
[87]:
```
# Metrics
actual_classes = df_test_set2_stage2_results['true_class']
predicted_classes = df_test_set2_stage2_results['predicted_class']


accuracy = accuracy_score(actual_classes, predicted_classes)
precision = precision_score(actual_classes, predicted_classes, average='macro',␣
 ↪labels=image_subclasses)
recall = recall_score(actual_classes, predicted_classes, average='macro',␣
 ↪labels=image_subclasses)
f1 = f1_score(actual_classes, predicted_classes, average='macro',␣
 ↪labels=image_subclasses)

print(f"Accuracy: {accuracy:.3f}")
print(f"Macro Precision: {precision:.3f}")
print(f"Macro Recall: {recall:.3f}")
print(f"Macro F1-score: {f1:.3f}")
```

```
Accuracy: 0.887
Macro Precision: 0.905
Macro Recall: 0.859
Macro F1-score: 0.874
```

Confusion Matrix

[88]:
```
labels = image_classes

cm = confusion_matrix(labels_set2_stage2_test, pred_labels,␣
 ↪labels=image_subclasses)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=image_subclasses,
            yticklabels= image_subclasses)
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

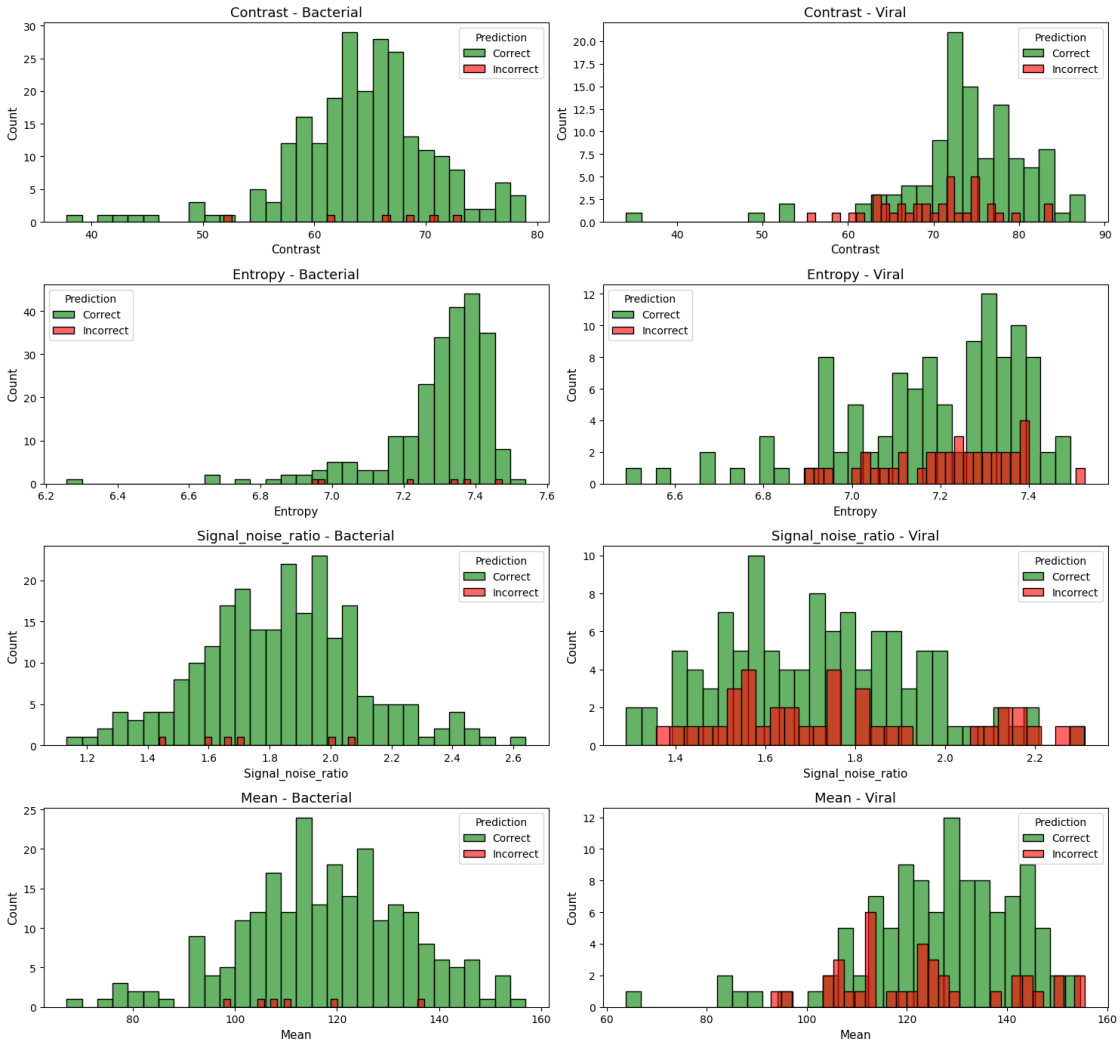
```
plt.show()
```



Evaluate Viral and Bacterial Test Results We will evaluate misclassified image characteristics to identify trends

[89]:
```
import seaborn as sns
import matplotlib.pyplot as plt

group_names = {1: 'Correct', 0: 'Incorrect'}
group_colors = {1: 'green', 0: 'red'}

fig, axes = plt.subplots(len(metrics), len(image_classes), figsize=(15, 14),␣
 ↪sharex=False)

for i, metric in enumerate(metrics):
    for j, cls in enumerate(image_subclasses):
        ax = axes[i, j]
        for corr in [1, 0]:  # correct, incorrect
            vals = df_test_set2_stage2_results[
                (df_test_set2_stage2_results['image_subclass'] == cls) &
```

```
                (df_test_set2_stage2_results['correct'] == corr)
            ][metric].dropna()
            sns.histplot(
                vals,
                bins=30,
                ax=ax,
                color=group_colors[corr],
                alpha=0.6,
                linewidth=1,
                label=group_names[corr]
            )
        ax.set_title(f"{metric.capitalize()} - {cls.capitalize()}", fontsize=13)
        ax.set_xlabel(metric.capitalize(), fontsize=11)
        ax.set_ylabel('Count', fontsize=11)
        ax.legend(title='Prediction')

plt.tight_layout()
plt.show()
```



1.6.4 Comparative Analysis and Model Selection

Here we will compare both models performance and select best one for our pneumonia diagnosis

Show Sample Predictions Visualize predictions for randomly selected images for Model 1 (Grayscale only) and Model 2 (Clahe + Colormap)

[90]:
```
# Diagnose image using stage 1 and stage 2 model
def run_pipeline_check(img_orig, learn_stage1, learn_stage2):
    """
    Performs a two-stage classification on an image:
    1. Classifies as 'normal' or 'pneumonia' using learn_stage1.
    2. If 'pneumonia', classifies as 'bacterial' or 'viral' using learn_stage2.
    """
```

```
    # Ensure grayscale transformation for Set 1 models
    img_s1_processed, = EnsureGrayscale().encodes((img_orig,))

    # Stage 1: Normal vs Pneumonia
    pred_1, _, probs_1 = learn_stage1.predict(img_s1_processed)
    conf_1 = probs_1.max().item()

    if pred_1 == 'normal':
        return f"Normal (Conf: {conf_1:.4f})"
    else:
        # Stage 2: Bacterial vs Viral (only for pneumonia cases)
        pred_2, _, probs_2 = learn_stage2.predict(img_s1_processed)
        conf_2 = probs_2.max().item()
        return f"{pred_2.capitalize()} (Conf: {conf_2:.4f})"

tfm_s1 = EnsureGrayscale()
tfm_s2 = Pipeline([EnsureGrayscale(), CLAHETransform(), ColormapTransform()])

n = 3
samples_norm = df_training_set2[df_training_set2['image_class'] == 'normal'].
 ↪sample(n, random_state=42).to_dict('records')
samples_bact = df_training_set2[(df_training_set2['image_class'] ==␣
 ↪'pneumonia') & (df_training_set2['image_subclass'] == 'bacterial')].
 ↪sample(n, random_state=42).to_dict('records')
samples_vir = df_training_set2[(df_training_set2['image_class'] == 'pneumonia')␣
 ↪& (df_training_set2['image_subclass'] == 'viral')].sample(n,␣
 ↪random_state=42).to_dict('records')

all_samples = [samples_norm, samples_bact, samples_vir]
cat_names = ['Normal', 'Bacterial', 'Viral']

fig, axes = plt.subplots(6, 3, figsize=(16, 24))
plt.subplots_adjust(hspace=0.3, wspace=0.1)
fig.suptitle(f"Model Inputs & Predictions: Grouped by Class", fontsize=20, y=0.
 ↪92)

for class_idx, class_samples in enumerate(all_samples):

    row_s1_idx = class_idx * 2
    row_s2_idx = class_idx * 2 + 1

    for sample_idx in range(n):

        sample = class_samples[sample_idx]
        img_path = os.path.join(PROJECT_PATH, sample['orig_file_path'])
        img_orig = PILImage.create(img_path)
```

```
        # --- ROW A: Set 1 (Grayscale) ---
        ax_s1 = axes[row_s1_idx, sample_idx]

        # Transform & Predict
        img_disp_s1, = tfm_s1(img_orig)
        res_1 = run_pipeline_check(img_orig, learn_set1_stage1,␣
↪learn_set2_stage1)

        pred_lbl = res_1.split(' ')[0].lower()
        true_lbl = sample['image_subclass'] if sample['image_class'] ==␣
↪'pneumonia' else 'normal'
        bg_color = '#d6f5d6' if pred_lbl == true_lbl else '#f5d6d6'

        # Plot
        ax_s1.imshow(img_disp_s1, cmap='gray')
        ax_s1.axis('off')
        ax_s1.set_title(f"{cat_names[class_idx]} Sample {sample_idx+1} (Set␣
↪1)", fontsize=10, fontweight='bold')
        ax_s1.text(0.5, -0.1, f"Pred: {res_1}", transform=ax_s1.transAxes,␣
↪ha='center', va='top',
                   bbox=dict(boxstyle='round', facecolor=bg_color, alpha=0.9))

        ax_s2 = axes[row_s2_idx, sample_idx]

         # Transform & Predict
        img_disp_s2, = tfm_s2(img_orig)
        res_2 = run_pipeline_check(img_orig, learn_set2_stage1,␣
↪learn_set2_stage2)

        # Parse Result
        pred_lbl = res_2.split(' ')[0].lower()
        bg_color = '#d6f5d6' if pred_lbl == true_lbl else '#f5d6d6'

        # Plot
        ax_s2.imshow(img_disp_s2)
        ax_s2.axis('off')
        ax_s2.set_title(f"{cat_names[class_idx]} Sample {sample_idx+1} (Set␣
↪2)", fontsize=10, fontweight='bold')
        ax_s2.text(0.5, -0.1, f"Pred: {res_2}", transform=ax_s2.transAxes,␣
↪ha='center', va='top',
                   bbox=dict(boxstyle='round', facecolor=bg_color, alpha=0.9))

plt.show()
```

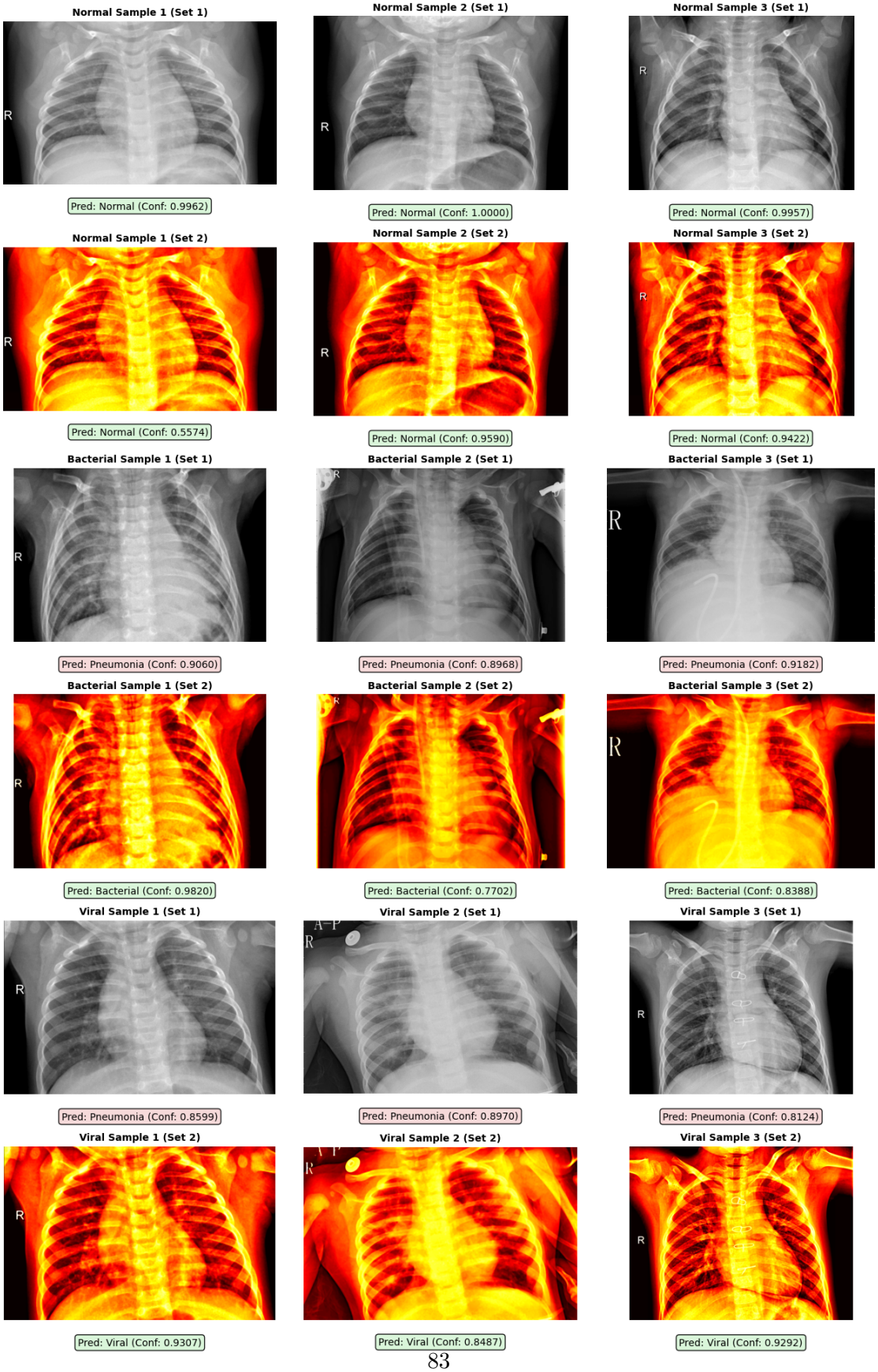
```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>

```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

Model Inputs & Predictions: Grouped by Class



Compare Training Models Accuracy In below tables we will compare models accuracy across stages. **Stage 1** performs binary discrimination between Normal and Pneumonia cases, while **Stage 2** further separates Pneumonia cases into Viral and Bacterial subclasses. **Set 1** operates on baseline preprocessed images, whereas **Set 2** uses an enhanced preprocessing strategy designed to emphasize lung structures and pathological patterns.

Stage 1: Normal vs Pneumonia

Model set	Stage	Accuracy	Precision (Pneumonia)	Recall (Pneumonia)	F1-score (Pneumonia)	Confusion matrix (TN, FP / FN, TP)
Set 1	Stage 1	0.806	0.767	0.990	0.865	117, 117 / 4, 386
Set 2	Stage 1	0.848	0.804	1.000	0.891	139, 95 / 0, 390

Stage 2: Viral vs Bacterial

Model set	Stage	Accuracy	Macro precision	Macro recall	Macro F1-score	Confusion matrix (TN, FP / FN, TP)
Set 1	Stage 2	0.897	0.926	0.866	0.884	241, 1 / 39, 109
Set 2	Stage 2	0.887	0.905	0.859	0.874	236, 6 / 38, 110

Across **Stage 1** (Normal vs Pneumonia), **Set 2** achieved higher test accuracy and F1-score than Set 1, while also reaching perfect recall for the Pneumonia class, indicating fewer missed pneumonia cases at the cost of slightly more false positives.

In **Stage 2** (Viral vs Bacterial), **Set 1** retained a small advantage in overall accuracy and macro precision/recall, although both model sets produced very similar macro `F1-scores`, suggesting broadly comparable performance when averaging over the two pneumonia subtypes.

Taken together, these results show that the enhanced preprocessing in **Set 2** is particularly beneficial for sensitive pneumonia detection in **Stage 1**, whereas the baseline configuration in **Set 1** remains slightly more stable for the more challenging Viral vs Bacterial discrimination in **Stage 2**.

Model Selection The overall error rate is calculated as the total number of errors (complete misses plus subtype misclassifications) divided by the total number of pneumonia cases, giving approximately 11.3% for both model sets.

Model Set	Total Pneumonia Cases	Total Errors	Error Rate Calculation		Error Rate (%)
Set 1	390	44	44 / 390	0.1128	11.3
Set 2	390	44	44 / 390	0.1128	11.3

In summary, while both model sets demonstrate comparable overall error rates (11.3%), the enhanced preprocessing in Model Set 2 (CLAHE + Hot colormap) proves clinically superior by achieving better sensitivity in Stage 1, ensuring no pneumonia cases are missed and all patients receive timely treatment. In contrast, Model Set 1's baseline approach results in critical misses that could lead to severe outcomes. For pneumonia screening where minimizing false negatives is paramount, Set 2's configuration offers a more reliable and safer pipeline, balancing detection accuracy with practical clinical impact.

1.7 Post Training Model Calibration

Next we can further improve accuracy of the model by post-training model calibration using basic thresholding to establish a straightforward baseline for decision-making. We will focus on Stage 1 only, which shown significant `Pneumonia` label over confidence

1.7.1 Load Models

[94]:
```
learn_stage1 = load_learner(MODEL_PATH + "/set2_pneumonia_detector.pkl")

df_test_set2_stage1_results = pd.read_csv(DATA_PATH+"/results_set2_stage1_test.
 ↪csv")
```

```
/usr/local/lib/python3.12/dist-packages/fastai/learner.py:455: UserWarning:
load_learner` uses Python's insecure pickle module, which can execute malicious
arbitrary code when loading. Only load files you trust.
If you only need to load model weights and optimizer state, use the safe
`Learner.load` instead.
  warn("load_learner` uses Python's insecure pickle module, which can execute
malicious arbitrary code when loading. Only load files you trust.\nIf you only
need to load model weights and optimizer state, use the safe `Learner.load`
instead.")
```

1.7.2 Calculate Optimal Threshold

[103]:
```
import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score

def pick_threshold(df_results, positive_label, metric="f1"):
    y_true_str = df_results["true_class"].to_numpy()
    probs      = df_results["confidence"].to_numpy()
    y_true = (y_true_str == positive_label).astype(int)
    results = []
    thresholds = np.linspace(0.1, 0.9, 17)
```

```
    for t in thresholds:
        y_pred = (probs >= t).astype(int)
        prec = precision_score(y_true, y_pred, zero_division=0)
        rec  = recall_score(y_true, y_pred, zero_division=0)
        f1   = f1_score(y_true, y_pred, zero_division=0)
        results.append((t, prec, rec, f1))
    arr = np.array(results, dtype=[("t", float), ("prec", float), ("rec",␣
↪float), ("f1", float)])
    best = arr[arr["f1"].argmax()]
    t = float(best["t"])
    print(f"Best threshold: {t:.2f} (precision={best['prec']:.3f},␣
↪recall={best['rec']:.3f}, F1={best['f1']:.3f})")
    return t, results

# Stage 1: Normal vs Pneumonia
best_t_stage1, stage1_results = pick_threshold(
    df_test_set2_stage1_results, positive_label="pneumonia", metric="f1"
)
```

```
Best threshold: 0.80 (precision=0.760, recall=0.956, F1=0.847)
```

Model metrics after calibrating threshold

1.7.3 Normal vs Pneumonia (Model 2) Calibration

Threshold setting	Precision (Pneumonia)	Recall (Pneumonia)	F1-score (Pneumonia)
Before (t = 0.50)	0.625	1.000	0.769
After (t = 0.80)	0.760	0.956	0.847

Threshold calibration for the selected Model 2 improved the balance of accuracy. Precision increased from 0.625 to 0.760 and F1-score from 0.769 to 0.847, with recall remaining above 0.95.

[102]:
```
t1 = best_t_stage1
y_true1_str = df_test_set2_stage1_results["true_class"].to_numpy()
probs1      = df_test_set2_stage1_results["confidence"].to_numpy()

y_true1 = (y_true1_str == "pneumonia").astype(int)
y_pred1 = (probs1 >= t1).astype(int)

cm1 = confusion_matrix(y_true1, y_pred1)
tn1, fp1, fn1, tp1 = cm1.ravel()

cm1_df = pd.DataFrame(
    [[tn1, fp1],
     [fn1, tp1]],
    index   = ["Actual Normal", "Actual Pneumonia"],
    columns = ["Predicted Normal", "Predicted Pneumonia"],
```

```
)
print("Confusion Matrix After Calibrations")
cm1_df
```

```
Confusion Matrix After Calibrations
```

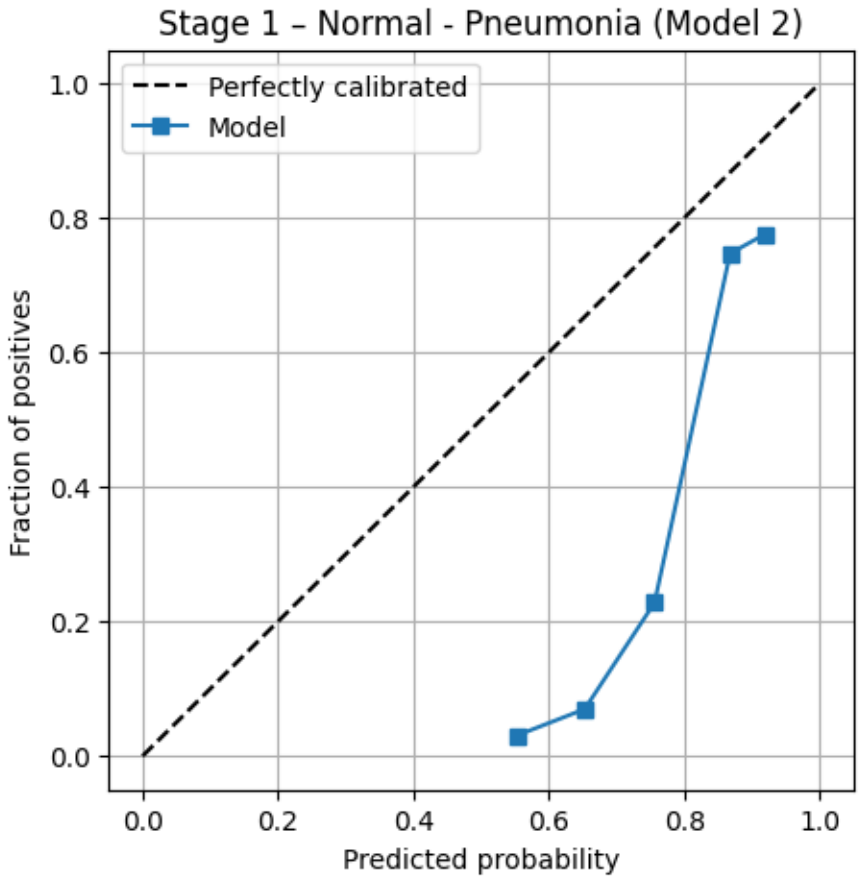
[102]:

	Predicted Normal	Predicted Pneumonia
Actual Normal	116	118
Actual Pneumonia	17	373

Model Calibration Plot Below we will plot calibration curve to show how well predicted probabilities match the true observed frequencies of the classes.

[101]:
```
def plot_calibration(df_results, positive_label, n_bins=10, title=""):
    """
    df_results:
      - 'true_class'   (string labels)
      - 'confidence'   (probability for positive_label)
    """
    y_true_str = df_results["true_class"].to_numpy()
    probs      = df_results["confidence"].to_numpy()

    y_true = (y_true_str == positive_label).astype(int)  # 1 = positive

    frac_pos, mean_pred = calibration_curve(
        y_true,
        probs,
        n_bins=n_bins,
        strategy="uniform"
    )

    plt.figure(figsize=(5,5))
    plt.plot([0, 1], [0, 1], "k--", label="Perfectly calibrated")
    plt.plot(mean_pred, frac_pos, "s-", label="Model")
    plt.xlabel("Predicted probability")
    plt.ylabel("Fraction of positives")
    plt.title(title or f"Calibration: {positive_label}")
    plt.legend()
    plt.grid(True)
    plt.show()

    return frac_pos, mean_pred


    # Stage 1: Normal vs Pneumonia
frac1, mean1 = plot_calibration(
    df_test_set2_stage1_results,
    positive_label="pneumonia",
```

```
    n_bins=10,
    title="Stage 1 - Normal - Pneumonia (Model 2)"
)
```



Based on the curve, we can see that our probabilities does not match well actual, which means that model is a good candidate for furhter calibration.

1.8 Deployment

We can use the model to predict diagnosis for an example x-ray image.

Test the Model with New X-ray Images

[129]:
```
img = PILImage.create(PROJECT_PATH+'deployment/sample-images'+ '/gr1.jpeg')
img.to_thumb(300)
```

[129]:

Predict the image class and plot the prediction

[135]:
```
def run_pipeline_check(img_orig, learn_stage1, learn_stage2, thresh_stage1=0.
↪80, thresh_stage2=0.65):

    # Ensure grayscale transformation (assuming your models expect it)
    img_processed = PILImage.create(img_orig).convert('L')  # Convert to␣
↪grayscale PILImage

    # Stage 1: Normal vs Pneumonia
    _, _, probs_1 = learn_stage1.predict(img_processed)  # Get probabilities

    pneumonia_idx = learn_stage1.dls.vocab.o2i['pneumonia']
    prob_pneumonia = probs_1[pneumonia_idx].item()
    conf_1 = prob_pneumonia
    if prob_pneumonia >= thresh_stage1:
        pred_1 = 'pneumonia'
    else:
        pred_1 = 'normal'
        return pred_1.capitalize(), conf_1, probs_1, None  # Early return for␣
↪normal

    # Stage 2: Viral vs Bacterial (only for pneumonia cases)
    _, _, probs_2 = learn_stage2.predict(img_processed)
    viral_idx = learn_stage2.dls.vocab.o2i['viral']
    prob_viral = probs_2[viral_idx].item()
    conf_2 = max(prob_viral, 1 - prob_viral)  # Max confidence for the␣
↪predicted class
```

```
    if prob_viral >= thresh_stage2:
        pred_2 = 'viral'
    else:
        pred_2 = 'bacterial'
    return pred_2.capitalize(), conf_2, probs_1, probs_2

# Load models
learn_stage1 = load_learner(f"{MODEL_PATH}/set2_pneumonia_detector.pkl")
learn_stage2 = load_learner(f"{MODEL_PATH}/set2_stage2_bacterial_viral_detector.
 ↪pkl")


# Run the pipeline
prediction, conf, probs_stage1, probs_stage2 = run_pipeline_check(
    img, learn_stage1, learn_stage2, thresh_stage1=0.80, thresh_stage2=0.65
)
print(f"Final Prediction: {prediction} (Conf: {conf:.4f})")

# Optional: Plot the image and probabilities (for both stages if applicable)
fig, axes = plt.subplots(1, 3 if probs_stage2 is not None else 2, figsize=(12,␣
 ↪4))

# Display the image
axes[0].imshow(img, cmap='gray')
axes[0].axis('off')
axes[0].set_title(f"Image\nPrediction: {prediction}")

# Stage 1 probabilities
sns.barplot(x=probs_stage1, y=learn_stage1.dls.vocab, orient='h', ax=axes[1])
axes[1].set_xlabel("Probability")
axes[1].set_ylabel("Class")
axes[1].set_title("Stage 1: Normal vs Pneumonia")

# Stage 2 probabilities (if pneumonia)
if probs_stage2 is not None:
    sns.barplot(x=probs_stage2, y=learn_stage2.dls.vocab, orient='h',␣
 ↪ax=axes[2])
    axes[2].set_xlabel("Probability")
    axes[2].set_ylabel("Class")
    axes[2].set_title("Stage 2: Bacterial vs Viral")

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/fastai/learner.py:455: UserWarning:
load_learner` uses Python's insecure pickle module, which can execute malicious
arbitrary code when loading. Only load files you trust.
```

```
If you only need to load model weights and optimizer state, use the safe
`Learner.load` instead.
  warn("load_learner` uses Python's insecure pickle module, which can execute
malicious arbitrary code when loading. Only load files you trust.\nIf you only
need to load model weights and optimizer state, use the safe `Learner.load`
instead.")
```

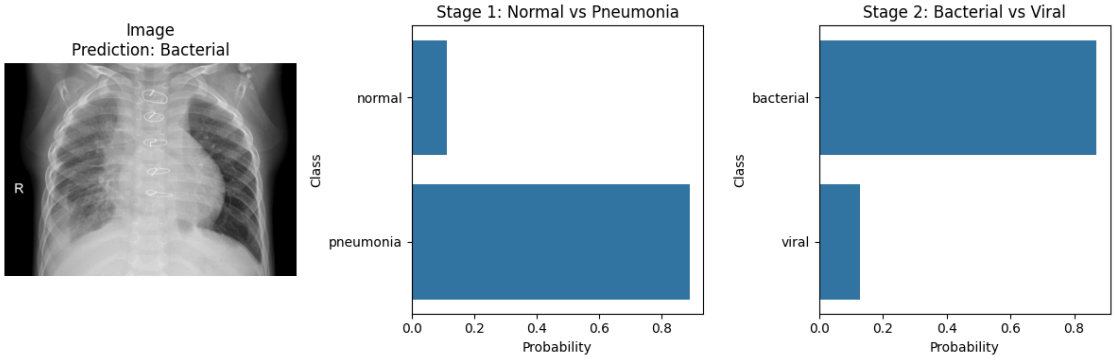
```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Final Prediction: Bacterial (Conf: 0.8702)
```



Reasoning: Filter the comparative DataFrame to focus on cases where Dynamic CLAHE changed the outcome (improved or worsened), and visualize the distribution of image metrics for these groups using box plots to understand the impact of image characteristics.

[154]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Filter for differences
df_diff = df_clean[df_clean['Outcome'].isin(['Dynamic Improved', 'Dynamic␣
  ↪Worsened'])].copy()

metrics = ['contrast', 'entropy', 'signal_noise_ratio', 'mean']

plt.figure(figsize=(14, 10))
for i, metric in enumerate(metrics):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(data=df_diff, x='Outcome', y=metric)
    plt.title(f'{metric.capitalize()} Distribution by Outcome')

plt.tight_layout()
```

```
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.12/dist-packages/pandas/core/indexes/base.py in␣
 ↪get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'Outcome'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/tmp/ipython-input-1604719651.py in <cell line: 0>()
      3
      4 # Filter for differences
----> 5 df_diff = df_clean[df_clean['Outcome'].isin(['Dynamic Improved',␣
 ↪'Dynamic Worsened'])].copy()
      6
      7 metrics = ['contrast', 'entropy', 'signal_noise_ratio', 'mean']

/usr/local/lib/python3.12/dist-packages/pandas/core/frame.py in␣
 ↪__getitem__(self, key)
   4100             if self.columns.nlevels > 1:
   4101                 return self._getitem_multilevel(key)
-> 4102             indexer = self.columns.get_loc(key)
   4103             if is_integer(indexer):
   4104                 indexer = [indexer]

/usr/local/lib/python3.12/dist-packages/pandas/core/indexes/base.py in␣
 ↪get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
```

```
    3814             # If we have a listlike key, _check_indexing_error will rai e

KeyError: 'Outcome'
```

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Filter the comparative dataframe for changed outcomes
# We focus on where the model changed its mind: Improved vs Worsened
df_pairplot = df_clean[df_clean['Outcome'].isin(['Dynamic Improved', 'Dynamic␣
 ↪Worsened'])].copy()

# 2. Define the metrics to analyze
metrics = ['contrast', 'entropy', 'signal_noise_ratio', 'mean']

# 3. Generate PairPlot
# This helps visualize if a combination of features separates the two groups
plt.figure(figsize=(12, 12))
g = sns.pairplot(
    df_pairplot,
    vars=metrics,
    hue='Outcome',
    palette={'Dynamic Improved': 'green', 'Dynamic Worsened': 'red'},
    diag_kind='kde',
    height=2.5
)

g.fig.suptitle('Multivariate Analysis: Image Characteristics vs. Model␣
 ↪Outcome', y=1.02)
plt.show()
```