```
pip install numpy pandas matplotlib seaborn scikit-learn tensorflow torch torchvision torchaudio opencv-python pillow albumentat
```

```
Requirement already satisfied: numpy in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (1.2
Requirement already satisfied: pandas in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (2.
Requirement already satisfied: matplotlib in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages
Requirement already satisfied: seaborn in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (0
Requirement already satisfied: scikit-learn in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packag
Requirement already satisfied: tensorflow in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages
Requirement already satisfied: torch in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (2.5
Requirement already satisfied: torchvision in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-package
Requirement already satisfied: torchaudio in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages
Requirement already satisfied: opencv-python in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packa
Requirement already satisfied: pillow in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (11
Requirement already satisfied: albumentations in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pack
Requirement already satisfied: grad-cam in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/jishnunarasimhamoorthy/Library/Python/3.12/lib/python/site-p
Requirement already satisfied: pytz>=2020.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packag
Requirement already satisfied: tzdata>=2022.7 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pack
Requirement already satisfied: contourpy>=1.0.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pa
Requirement already satisfied: cycler>=0.10 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packag
Requirement already satisfied: fonttools>=4.22.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-p
Requirement already satisfied: kiwisolver>=1.3.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-p
Requirement already satisfied: packaging>=20.0 in /Users/jishnunarasimhamoorthy/Library/Python/3.12/lib/python/site-packages
Requirement already satisfied: pyparsing>=2.3.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pa
Requirement already satisfied: scipy>=1.6.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packag
Requirement already satisfied: joblib>=1.2.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packa
Requirement already satisfied: threadpoolctl>=3.1.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/sit
Requirement already satisfied: absl-py>=1.0.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pack
Requirement already satisfied: astunparse>=1.6.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-p
Requirement already satisfied: flatbuffers>=24.3.25 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/sit
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib
Requirement already satisfied: google-pasta>=0.1.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site
Requirement already satisfied: libclang>=13.0.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pa
Requirement already satisfied: opt-einsum>=2.3.2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-p
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /Library/
Requirement already satisfied: requests<3,>=2.21.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site
Requirement already satisfied: setuptools in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages
Requirement already satisfied: six>=1.12.0 in /Users/jishnunarasimhamoorthy/Library/Python/3.12/lib/python/site-packages (fr
Requirement already satisfied: termcolor>=1.1.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pa
Requirement already satisfied: typing-extensions>=3.6.6 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12
Requirement already satisfied: wrapt>=1.11.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packa
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site
Requirement already satisfied: tensorboard<2.19,>=2.18 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
Requirement already satisfied: keras>=3.5.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packag
Requirement already satisfied: h5py>=3.11.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packag
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
Requirement already satisfied: filelock in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (
Requirement already satisfied: networkx in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (
Requirement already satisfied: jinja2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (fr
Requirement already satisfied: fsspec in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (fr
Requirement already satisfied: sympy==1.13.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packa
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
Requirement already satisfied: PyYAML in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (fr
Requirement already satisfied: pydantic>=2.9.2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pac
Requirement already satisfied: albucore==0.0.21 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pa
Requirement already satisfied: eval-type-backport in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
Requirement already satisfied: opencv-python-headless>=4.9.0.80 in /Library/Frameworks/Python.framework/Versions/3.12/lib/py
Requirement already satisfied: stringzilla>=3.10.4 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site
Requirement already satisfied: simsimd>=5.9.2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pack
Requirement already satisfied: ttach in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (fro
```

```
# Core Libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning Libraries
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix, classification_report

# Deep Learning Libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Visualization
```

```
import cv2
from tensorflow.keras.utils import plot_model
import albumentations as A

# Grad-CAM
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet import preprocess_input
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/albumentations/check_version.py:51: UserWarn
  data = fetch_version_info()
```

```
def check_gpu():
    print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
    if tf.test.gpu_device_name():
        print("GPU is enabled:", tf.test.gpu_device_name())
    else:
        print("GPU is not enabled.")

check_gpu()
```

```
Num GPUs Available:  0
GPU is not enabled.
```

```
# Define paths to the dataset
base_path = "archive/sample"
labels_path = os.path.join(base_path, "sample_labels.csv")
images_path = os.path.join(base_path, "images")
```

Data Cleaning and Loading

```
# Load the CSV file
data = pd.read_csv(labels_path)

# Add image paths to the dataframe
data['image_path'] = data['Image Index'].apply(lambda x: os.path.join(images_path, x))

# Clean Patient Age
# Extract numeric values and convert to integer
data['Patient Age'] = data['Patient Age'].str.extract('(\d+)').astype(float).astype(int)

# Replace missing labels with an empty string
data['Finding Labels'] = data['Finding Labels'].fillna("")

# List of disease classes
disease_classes = [
    'Atelectasis', 'Cardiomegaly', 'Effusion', 'Infiltration',
    'Mass', 'Nodule', 'Pneumonia', 'Pneumothorax',
    'Consolidation', 'Edema', 'Emphysema', 'Fibrosis',
    'Pleural_Thickening', 'Hernia'
]

# Generate binary columns for each disease class
for disease in disease_classes:
    data[disease] = data['Finding Labels'].apply(lambda x: 1 if disease in x else 0)

# Check the processed data
print(data.head())
```

```
         Image Index                                   Finding Labels  \
0  00000013_005.png  Emphysema|Infiltration|Pleural_Thickening|Pneu...
1  00000013_026.png                         Cardiomegaly|Emphysema
2  00000017_001.png                                     No Finding
3  00000030_001.png                                     Atelectasis
4  00000032_001.png                   Cardiomegaly|Edema|Effusion

   Follow-up #  Patient ID  Patient Age Patient Gender View Position  \
0            5          13           60              M            AP
1           26          13           57              M            AP
2            1          17           77              M            AP
3            1          30           79              M            PA
4            1          32           55              F            AP
```

```
   OriginalImageWidth  OriginalImageHeight  OriginalImagePixelSpacing_x  ...  \
0                3056                 2544                        0.139  ...
1                2500                 2048                        0.168  ...
2                2500                 2048                        0.168  ...
3                2992                 2991                        0.143  ...
4                2500                 2048                        0.168  ...

   Mass Nodule  Pneumonia  Pneumothorax  Consolidation  Edema  Emphysema  \
0     0      0          0             1              0      0          1
1     0      0          0             0              0      0          1
2     0      0          0             0              0      0          0
3     0      0          0             0              0      0          0
4     0      0          0             0              0      1          0

   Fibrosis  Pleural_Thickening  Hernia
0         0                   1       0
1         0                   0       0
2         0                   0       0
3         0                   0       0
4         0                   0       0

[5 rows x 26 columns]
<>:9: SyntaxWarning: invalid escape sequence '\d'
<>:9: SyntaxWarning: invalid escape sequence '\d'
/var/folders/wn/n2fx_jqn07z9z11jmc33c0000000gn/T/ipykernel_62307/505373478.py:9: SyntaxWarning: invalid escape sequence '\d'
  data['Patient Age'] = data['Patient Age'].str.extract('(\d+)').astype(float).astype(int)
```

Checking for Null Values

```
print(data.isnull().sum())
```

```
Image Index                   0
Finding Labels                0
Follow-up #                   0
Patient ID                    0
Patient Age                   0
Patient Gender                0
View Position                 0
OriginalImageWidth            0
OriginalImageHeight           0
OriginalImagePixelSpacing_x   0
OriginalImagePixelSpacing_y   0
image_path                    0
Atelectasis                   0
Cardiomegaly                  0
Effusion                      0
Infiltration                  0
Mass                          0
Nodule                        0
Pneumonia                     0
Pneumothorax                  0
Consolidation                 0
Edema                         0
Emphysema                     0
Fibrosis                      0
Pleural_Thickening            0
Hernia                        0
dtype: int64
```

Checking for Label Consistency

```
unique_labels = set('|'.join(data['Finding Labels']).split('|'))
print("Unique labels in dataset:", unique_labels)
```

```
Unique labels in dataset: {'Pleural_Thickening', 'Emphysema', 'Pneumothorax', 'Pneumonia', 'Infiltration', 'Mass', 'Atelecta
```

Making sure columns only contain 1 and 0

```
print(data[disease_classes].sum())
print(data[disease_classes].head())
```

```
Atelectasis          508
Cardiomegaly         141
Effusion             644
Infiltration         967
```

```
Mass                284
Nodule              313
Pneumonia            62
Pneumothorax        271
Consolidation       226
Edema               118
Emphysema           127
Fibrosis             84
Pleural_Thickening  176
Hernia               13
dtype: int64
   Atelectasis  Cardiomegaly  Effusion  Infiltration  Mass  Nodule  Pneumonia  \
0            0             0         0             1     0       0          0
1            0             1         0             0     0       0          0
2            0             0         0             0     0       0          0
3            1             0         0             0     0       0          0
4            0             1         1             0     0       0          0

   Pneumothorax  Consolidation  Edema  Emphysema  Fibrosis  \
0             1              0      0          1         0
1             0              0      0          1         0
2             0              0      0          0         0
3             0              0      0          0         0
4             0              0      1          0         0

   Pleural_Thickening  Hernia
0                   1       0
1                   0       0
2                   0       0
3                   0       0
4                   0       0
```

Check for Multi-Label Rows : Making sure there are rows with multiple classifications.

```
multi_label_rows = data[disease_classes].sum(axis=1)
print(f"Number of multi-label rows: {sum(multi_label_rows > 1)}")
```

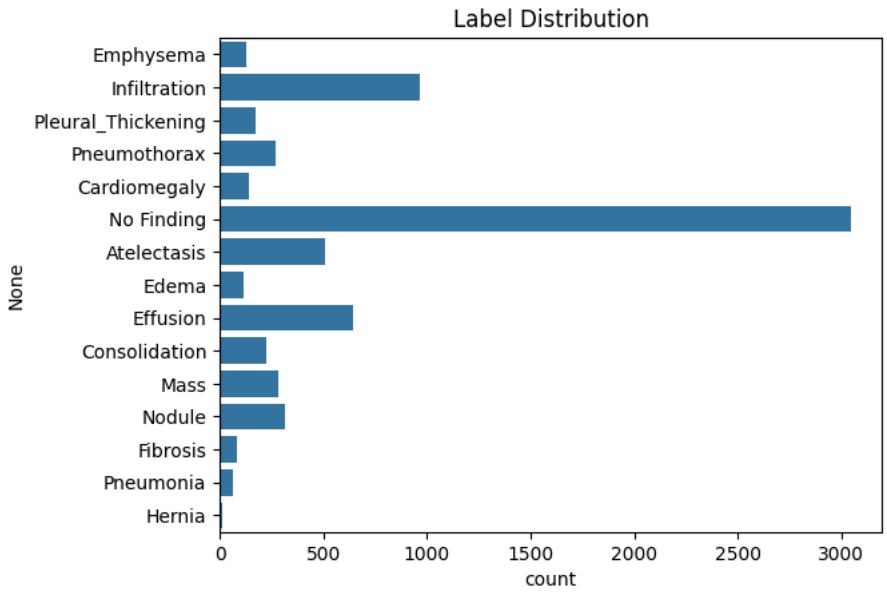
```
Number of multi-label rows: 980
```

```
print(f"Dataset size after cleaning: {len(data)}")
```

```
Dataset size after cleaning: 5606
```

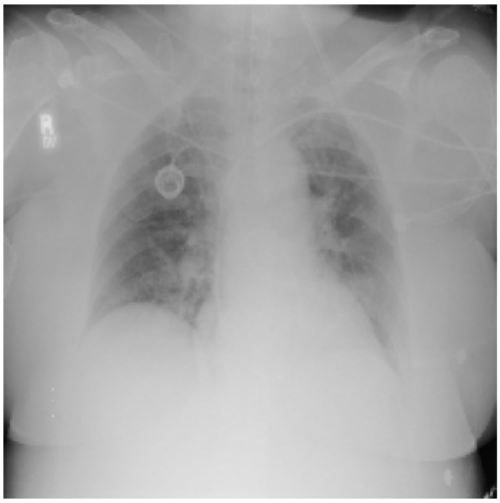
EDA

```
def visualize_label_distribution(data):
    """
    Visualize the distribution of labels.
    """
    all_labels = [label for sublist in data['Finding Labels'].str.split('|') for label in sublist]
    sns.countplot(y=pd.Series(all_labels))
    plt.title("Label Distribution")
    plt.show()


visualize_label_distribution(data)
```




```
def display_one_image_per_class(data):
    """
    Display one random image for each of the 14 unique disease classes in the dataset.
    """
    # List of all 14 possible disease classes
    all_classes = [
        'Atelectasis', 'Cardiomegaly', 'Effusion', 'Infiltration',
        'Mass', 'Nodule', 'Pneumonia', 'Pneumothorax',
        'Consolidation', 'Edema', 'Emphysema', 'Fibrosis',
        'Pleural_Thickening', 'Hernia'
    ]

    for disease in all_classes:
        # Filter data to get rows that include the current disease
        class_data = data[data['Finding Labels'].str.contains(disease, na=False)]

        if not class_data.empty:
            # Select one random image from this class
            sample = class_data.sample(1).iloc[0]

            # Load and display the image
            img = load_img(sample['image_path'], target_size=(224, 224))
            plt.imshow(img)
            plt.title(f"Class: {disease}")
            plt.axis('off')
            plt.show()
        else:
            print(f"No images found for class: {disease}")

# Call the function
display_one_image_per_class(data)
```

Class: Atelectasis

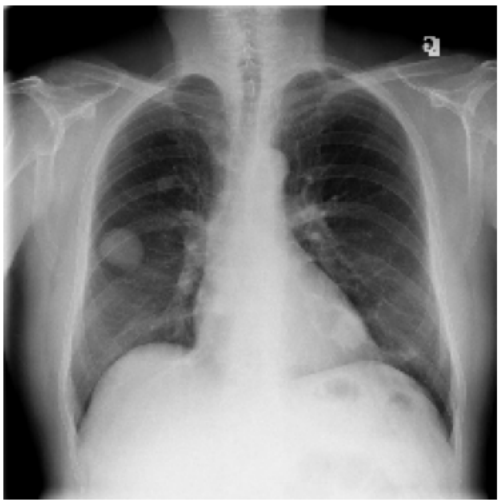


Class: Cardiomegaly

Class: Effusion

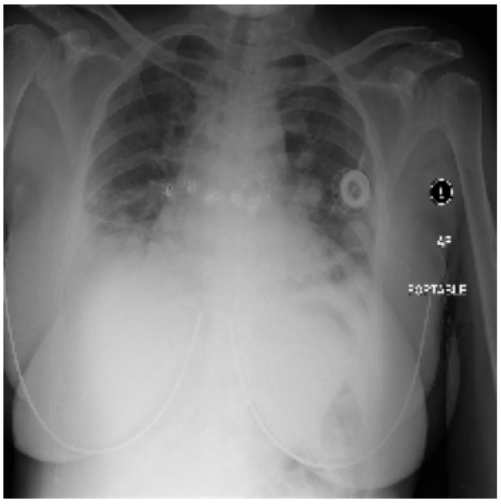
Class: Infiltration

Class: Mass



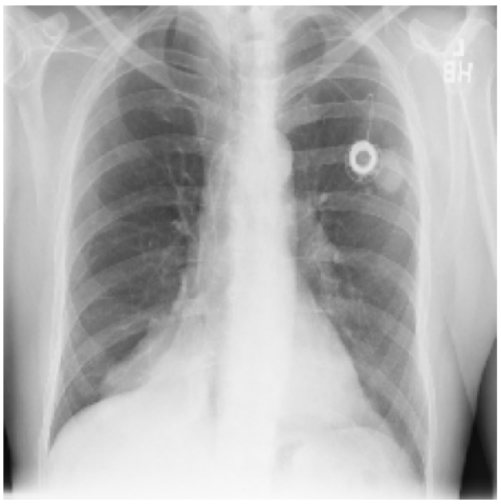
Class: Nodule

Class: Pneumonia

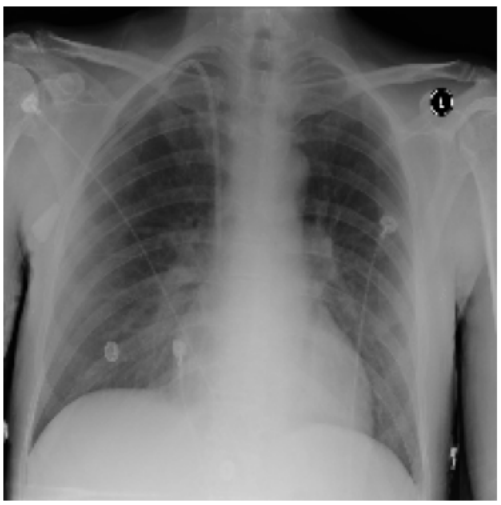


Class: Pneumothorax

Class: Consolidation



Class: Edema



Class: Emphysema

Class: Fibrosis

Class: Pleural_Thickening

Class: Hernia

Train-Test Split

```
from sklearn.model_selection import train_test_split

# Split the dataset into train and test sets (80% train, 20% test)
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)

print(f"Training set size: {len(train_data)}")
print(f"Testing set size: {len(test_data)}")
```

```
Training set size: 4484
Testing set size: 1122
```

Pre-Processing for Neural Network

```
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)


# Training generator
train_gen = datagen.flow_from_dataframe(
    train_data,
    x_col='image_path',
    y_col=disease_classes,
    target_size=(224, 224),
    color_mode='grayscale',  # Load images as grayscale
    class_mode='raw',  # Multi-label classification
    batch_size=32
)

# Testing generator
test_gen = datagen.flow_from_dataframe(
    test_data,
    x_col='image_path',
    y_col=disease_classes,
    target_size=(224, 224),
    color_mode='grayscale',  # Load images as grayscale
    class_mode='raw',  # Multi-label classification
    batch_size=32
)
```

```
Found 4484 validated image filenames.
Found 1122 validated image filenames.
```

Base CNN Model

```
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Model

# Define the Functional API Model
input_layer = Input(shape=(224, 224, 1), name='input_layer')
x = Conv2D(32, (3, 3), activation='relu', name='conv2d_1')(input_layer)
x = MaxPooling2D((2, 2), name='max_pooling2d_1')(x)
x = Conv2D(64, (3, 3), activation='relu', name='conv2d_2')(x)
x = MaxPooling2D((2, 2), name='max_pooling2d_2')(x)
x = Conv2D(128, (3, 3), activation='relu', name='conv2d_3')(x)
x = MaxPooling2D((2, 2), name='max_pooling2d_3')(x)
x = Flatten(name='flatten')(x)
x = Dense(128, activation='relu', name='dense_1')(x)
x = Dropout(0.5, name='dropout')(x)
output_layer = Dense(len(disease_classes), activation='sigmoid', name='output_layer')(x)

CNN_model = Model(inputs=input_layer, outputs=output_layer, name='CNN_Model')
```

```
# Compile the model
CNN_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Summary of the model
CNN_model.summary()
```

Model: "CNN_Model"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 1)	0
conv2d_1 (Conv2D)	(None, 222, 222, 32)	320
max_pooling2d_1 (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_2 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_3 (Conv2D)	(None, 52, 52, 128)	73,856
max_pooling2d_3 (MaxPooling2D)	(None, 26, 26, 128)	0
flatten (Flatten)	(None, 86528)	0
dense_1 (Dense)	(None, 128)	11,075,712
dropout (Dropout)	(None, 128)	0
output_layer (Dense)	(None, 14)	1,806

Total params: 11,170,190 (42.61 MB)
Trainable params: 11,170,190 (42.61 MB)
Non-trainable params: 0 (0.00 B)

```
from tensorflow.keras.callbacks import EarlyStopping

# Define the EarlyStopping callback
early_stopping = EarlyStopping(
    monitor='val_loss',  # Metric to monitor
    patience=3,          # Number of epochs with no improvement
    restore_best_weights=True  # Restore model weights from the best epoch
)

# Train the model
history = CNN_model.fit(
    train_gen,            # Training data generator
    validation_data=test_gen,  # Validation data generator
    epochs=5,             # Adjust the number of epochs
    callbacks=[early_stopping],  # Early stopping
    verbose=1
)
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/keras/src/trainers/data_adapters/py_dataset_
  self._warn_if_super_not_called()
Epoch 1/5
141/141 ━━━━━━━━━━━━━━━━━━━━ 107s 754ms/step - accuracy: 0.1662 - loss: 0.2607 - val_accuracy: 0.1176 - val_loss: 0.1819
Epoch 2/5
141/141 ━━━━━━━━━━━━━━━━━━━━ 101s 717ms/step - accuracy: 0.1710 - loss: 0.1972 - val_accuracy: 0.1176 - val_loss: 0.1797
Epoch 3/5
141/141 ━━━━━━━━━━━━━━━━━━━━ 105s 740ms/step - accuracy: 0.1833 - loss: 0.1965 - val_accuracy: 0.1176 - val_loss: 0.1796
Epoch 4/5
141/141 ━━━━━━━━━━━━━━━━━━━━ 108s 762ms/step - accuracy: 0.1630 - loss: 0.1891 - val_accuracy: 0.1176 - val_loss: 0.1786
Epoch 5/5
141/141 ━━━━━━━━━━━━━━━━━━━━ 106s 758ms/step - accuracy: 0.1781 - loss: 0.1858 - val_accuracy: 0.1194 - val_loss: 0.1790
```

```
def generate_gradcam_heatmap(model, last_conv_layer_name, img_array, class_index=None):
    grad_model = tf.keras.models.Model(
        [model.input],
        [model.get_layer(last_conv_layer_name).output, model.output]
    )

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        if class_index is None:
            class_index = tf.argmax(predictions[0])
        class_channel = predictions[:, class_index]
```

```
    grads = tape.gradient(class_channel, conv_outputs)
    if grads is None:
        raise ValueError("Gradients could not be computed. Check model compatibility.")

    # Debug gradients
    print("Gradients shape:", grads.shape)
    print("Gradients min:", tf.reduce_min(grads).numpy())
    print("Gradients max:", tf.reduce_max(grads).numpy())

    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    conv_outputs = conv_outputs[0]
    heatmap = tf.reduce_sum(tf.multiply(pooled_grads, conv_outputs), axis=-1)

    # Normalize the heatmap
    heatmap = np.maximum(heatmap, 0)
    heatmap = heatmap / np.max(heatmap) if np.max(heatmap) != 0 else np.zeros_like(heatmap)
    return heatmap


# heatmap = np.maximum(heatmap, 0)
# max_value = np.max(heatmap)
# heatmap = heatmap / max_value if max_value > 1e-10 else np.zeros_like(heatmap)


import os
import random
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# Specify the image directory
image_dir = "archive/sample/images"  # Replace with your actual directory

# List all image files in the directory
image_files = [os.path.join(image_dir, f) for f in os.listdir(image_dir) if f.endswith((".png", ".jpg", ".jpeg"))]

# Randomly select an image
img_path = random.choice(image_files)
print(f"Selected Image: {img_path}")
```

```
Selected Image: archive/sample/images/00001301_021.png
```

```
# Load the selected image
img = load_img(img_path, target_size=(224, 224), color_mode="grayscale")

# Convert the image to an array and normalize it
img_array = np.expand_dims(img_to_array(img) / 255.0, axis=0)  # Normalize to [0, 1]
print("Image shape:", img_array.shape)  # Should print (1, 224, 224, 1) for grayscale
```

```
Image shape: (1, 224, 224, 1)
```

```
# Predict the class for the selected image using the model
predictions = CNN_model.predict(img_array)
predicted_class_index = np.argmax(predictions[0])  # Get the index of the predicted class
predicted_class = disease_classes[predicted_class_index]  # Use the index to find the class name

print(f"Predicted Class: {predicted_class}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
Predicted Class: Infiltration
```

```
for idx, layer in enumerate(CNN_model.layers):
    if hasattr(layer, 'output_shape'):
        output_shape = layer.output_shape
    elif hasattr(layer, 'batch_input_shape'):  # Handle InputLayer specifically
        output_shape = layer.batch_input_shape
    else:
        output_shape = "Unknown"
    print(f"Layer {idx}: {layer.name}, Output shape: {output_shape}")
```

```
Layer 0: input_layer, Output shape: Unknown
Layer 1: conv2d_1, Output shape: Unknown
Layer 2: max_pooling2d_1, Output shape: Unknown
Layer 3: conv2d_2, Output shape: Unknown
Layer 4: max_pooling2d_2, Output shape: Unknown
Layer 5: conv2d_3, Output shape: Unknown
Layer 6: max_pooling2d_3, Output shape: Unknown
```

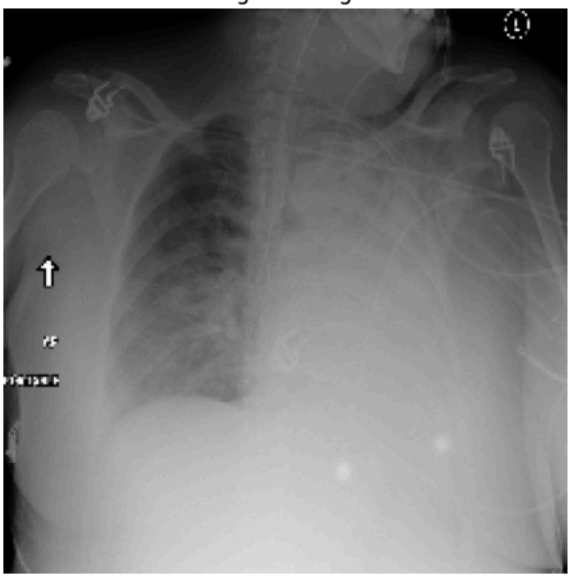
```
Layer 7: flatten, Output shape: Unknown
Layer 8: dense_1, Output shape: Unknown
Layer 9: dropout, Output shape: Unknown
Layer 10: output_layer, Output shape: Unknown
```

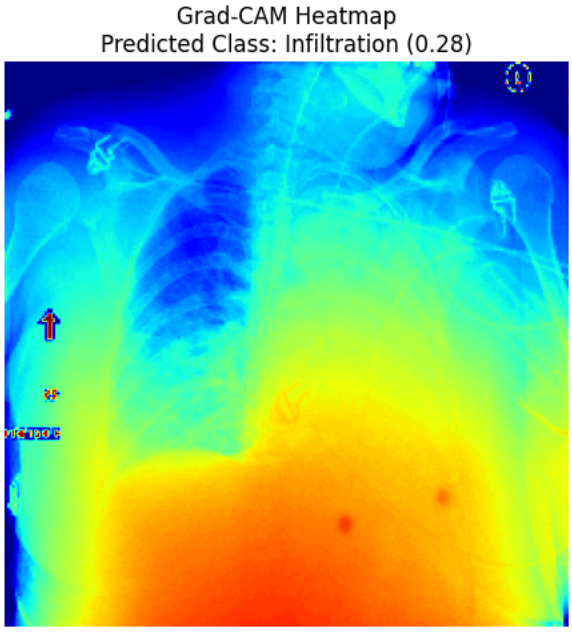
```
# Generate and display the heatmap
last_conv_layer_name = "conv2d_3"  # Update this based on your model
heatmap = generate_gradcam_heatmap(CNN_model, last_conv_layer_name, img_array)
```

```
Gradients shape: (1, 52, 52, 128)
Gradients min: -0.022241646
Gradients max: 0.029697776
```

```
# Resize the heatmap to match the image size
heatmap_resized = cv2.resize(heatmap, (224, 224))

# Overlay the heatmap on the original image
superimposed_img = cv2.addWeighted(img_to_array(img) / 255.0, 0.6, heatmap_resized, 0.4, 0)

# Get prediction probabilities for each class
predicted_probabilities = predictions[0]

# Print predicted probabilities for all classes
print("Prediction Probabilities:")
for i, class_name in enumerate(disease_classes):
    print(f"{class_name}: {predicted_probabilities[i]:.2f}")

# Get the predicted class label
predicted_class_index = np.argmax(predicted_probabilities)  # Index of the predicted class
predicted_class = disease_classes[predicted_class_index]    # Class label from disease_classes list
predicted_probability = predicted_probabilities[predicted_class_index]  # Probability of the predicted class

# Plot the original image and Grad-CAM heatmap side by side
plt.figure(figsize=(12, 6))

# Original Image
plt.subplot(1, 2, 1)
plt.imshow(img_to_array(img) / 255.0, cmap='gray')  # Display the original image in grayscale
plt.axis('off')
plt.title("Original Image")

# Grad-CAM Heatmap Overlayed Image
plt.subplot(1, 2, 2)
plt.imshow(superimposed_img, cmap='jet')  # Display the heatmap overlay
plt.axis('off')
plt.title(f'Grad-CAM Heatmap\nPredicted Class: {predicted_class} ({predicted_probability:.2f})')

plt.show()
```

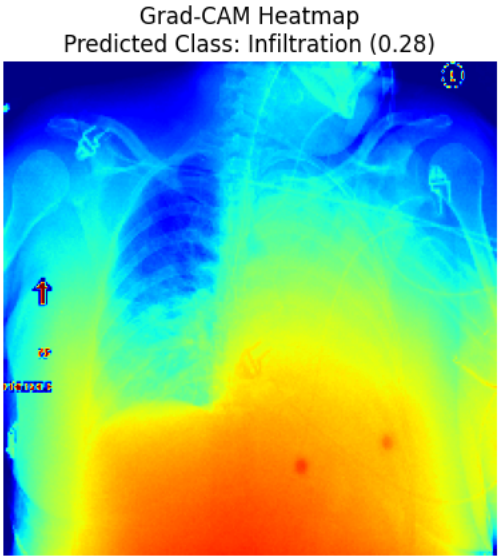
```
Prediction Probabilities:
Atelectasis: 0.15
Cardiomegaly: 0.05
Effusion: 0.19
Infiltration: 0.28
Mass: 0.09
Nodule: 0.08
Pneumonia: 0.02
Pneumothorax: 0.08
Consolidation: 0.07
Edema: 0.04
Emphysema: 0.05
Fibrosis: 0.03
Pleural_Thickening: 0.06
Hernia: 0.01
```






```
# # Resize the heatmap to match the image size
# heatmap_resized = cv2.resize(heatmap, (224, 224))

# # Overlay the heatmap on the original image
# superimposed_img = cv2.addWeighted(img_to_array(img) / 255.0, 0.6, heatmap_resized, 0.4, 0)

# # Get prediction probabilities for each class
# predicted_probabilities = predictions[0]

# # Print predicted probabilities for all classes
# print("Prediction Probabilities:")
# for i, class_name in enumerate(disease_classes):
#     print(f"{class_name}: {predicted_probabilities[i]:.2f}")

# # Get the predicted class label
# predicted_class_index = np.argmax(predicted_probabilities)  # Index of the predicted class
# predicted_class = disease_classes[predicted_class_index]    # Class label from disease_classes list
# predicted_probability = predicted_probabilities[predicted_class_index]  # Probability of the predicted class

# # Display the image with heatmap, predicted class, and its probability
# plt.imshow(superimposed_img, cmap='jet')
# plt.axis('off')
# plt.title(f'Grad-CAM Heatmap\nPredicted Class: {predicted_class} ({predicted_probability:.2f})')
# plt.show()
```

```
Prediction Probabilities:
Atelectasis: 0.15
Cardiomegaly: 0.05
Effusion: 0.19
Infiltration: 0.28
Mass: 0.09
Nodule: 0.08
Pneumonia: 0.02
Pneumothorax: 0.08
Consolidation: 0.07
Edema: 0.04
Emphysema: 0.05
Fibrosis: 0.03
Pleural_Thickening: 0.06
Hernia: 0.01
```



Color Mapping: • The colors in the heatmap correspond to the importance of specific regions of the image: • Red/Hot Colors (e.g., Red, Yellow, Orange): These are areas of high importance or attention for the model. They strongly influenced the model's decision. • Blue/Cool Colors (e.g., Blue, Green): These regions have little to no influence on the model's prediction.

Evaluation

```
# Evaluate the model on the test data
test_loss, test_accuracy = CNN_model.evaluate(test_gen, verbose=1)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")

# Access ground truth labels
y_true = test_gen.labels  # Correct attribute for ground truth labels

# Generate predictions
y_pred_probs = CNN_model.predict(test_gen, verbose=0)  # Predicted probabilities
y_pred_classes = np.argmax(y_pred_probs, axis=1)  # Predicted class labels
```

```
36/36 ━━━━━━━━━━━━━━━━━━━━ 19s 516ms/step - accuracy: 0.1196 - loss: 0.1777
Test Loss: 0.1793
Test Accuracy: 0.1176
```

```
# Use disease_classes directly for target names
print("\nClassification Report:\n")
print(classification_report(y_true.argmax(axis=1), y_pred_classes, target_names=disease_classes))
```

```
Classification Report:

                    precision    recall  f1-score   support

       Atelectasis       0.00      0.00      0.00       726
      Cardiomegaly       0.00      0.00      0.00        20
         Effusion        0.00      0.00      0.00        86
      Infiltration       0.12      1.00      0.21       132
              Mass       0.00      0.00      0.00        35
```

```
                  Nodule       0.00      0.00      0.00        36
               Pneumonia       0.00      0.00      0.00         6
            Pneumothorax       0.00      0.00      0.00        26
           Consolidation       0.00      0.00      0.00        19
                  Edema       0.00      0.00      0.00        12
               Emphysema       0.00      0.00      0.00        10
                Fibrosis       0.00      0.00      0.00         9
      Pleural_Thickening       0.00      0.00      0.00         4
                  Hernia       0.00      0.00      0.00         1

                accuracy                           0.12      1122
               macro avg       0.01      0.07      0.02      1122
            weighted avg       0.01      0.12      0.02      1122

/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1531: Und
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1531: Und
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1531: Und
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Ensure y_true and y_pred_probs are arrays
y_true = test_gen.labels  # True labels
y_pred_probs = CNN_model.predict(test_gen, verbose=0)  # Predicted probabilities

# Plot ROC curves for each class
plt.figure(figsize=(10, 8))
for i, class_name in enumerate(disease_classes):
    # Compute ROC curve and AUC for each class
    fpr, tpr, _ = roc_curve(y_true[:, i], y_pred_probs[:, i])
    roc_auc = auc(fpr, tpr)

    # Plot the ROC curve
    plt.plot(fpr, tpr, label=f"{class_name} (AUC = {roc_auc:.2f})")

# Plot the diagonal (random guess line)
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")

# Customize the plot
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for Each Class")
plt.legend(loc="lower right")
plt.show()
```