Graph-Based Approach for Pulmonary Diseases Classification

By Anuranj VV

INTRODUCTION

Accurate diagnosis of pulmonary diseases [1] is essential for effective treatment and management of the patient, given the significant impact these conditions have on public health. Medical imaging, particularly chest X-rays, plays a crucial role in detecting and classifying pulmonary diseases. However, due to the complexity of medical images, advanced methods are needed to ensure accurate diagnosis. Traditional approaches, such as CNNs[2], have been widely used for image classification tasks, including in the medical field. While CNNs have achieved notable success in various applications, they struggle to capture the intricate relationships within images. This limitation is especially relevant in medical imaging, where small variations and contextual information are critical for accurate diagnoses. CNNs work on fixed grid structures, which may miss essential details and relationships that are not easily captured in traditional formats.

GCNs[3], while effective for graph-based data, struggle with capturing fine-grained spatial details in X-ray images, which are critical for identifying subtle abnormalities. They also face issues with over smoothing, losing important local information in deeper layers, and difficulty handling dynamic or long-range relationships between distant regions in the image, which can be crucial for accurate diagnoses.

To overcome these challenges, there is growing interest in using graph-based representations and attention mechanisms. Graphs offer a flexible way to model complex relationships and structures inherent in medical images. Transforming X-ray images into graph structures leverages the strengths of graph-based methods, such as GAT[4][5], which primarily focus on the most important features within the graph that leads to improved classification of pulmonary diseases.

In this study, a novel graph-based approach has been proposed for classifying pulmonary diseases using chest X-ray images. In this method, the images are segmented into super pixels [6] using the SLIC algorithm[7] to capture both local and global patterns while retaining important structural information. A Region Adjacency Graph(RAG) [8] is built from super pixels, where nodes represent image regions and edges represent relationships between them.

The use of GAT is an advanced method that leverages attention mechanisms to focus on the most important parts of the image (super pixels) when learning features. The GAT is then applied to the RAG graphs to extract relevant features for predictive model training, thereby improving the model's ability to accurately classify diseases. Our approach seeks to address the limitations of CNNs and GCNs, providing a more robust solution for pulmonary disease classification.

LITERATURE REVIEW

The classification of pulmonary diseases using medical imaging, particularly chest X-ray images, has seen significant advancements through the application of DL techniques. Traditional CNNs have been widely utilized for image classification tasks; however, they often struggle to capture complex relationships within medical images, which is important for accurately diagnosing pulmonary diseases. Recent studies have explored various methodologies to enhance the performance of DL models in this domain.

In another study, a Hybrid Deep Learning Algorithm (HDLA) was proposed for automatic lung disease classification from chest X-ray images. This framework integrated various techniques, including anomaly detection and confidence prediction, to enhance the detection of conditions like pneumonia. The model demonstrated improved performance compared to traditional methods, highlighting the importance of combining different approaches to address the limitations of existing systems [10].

Moreover, the use of segmentation techniques prior to classification has shown promise in improving diagnostic accuracy. A study employed a two-stage pipeline that first segmented chest X-ray images before classifying them into pneumonia and tuberculosis. This method utilized three deep convolutional neural networks (VGG16, ResNet-50, and InceptionV3) pre-trained on the ImageNet dataset and assessed them in lung disease classification tasks using a transfer learning approach. [11]. The framework was validated on the publicly available Shenzhen and Montgomery lung datasets and compared its performance. The inceptionV3-based model outperformed the Shenzhen dataset despite being computationally less expensive.

One notable approach is the use of GCNs for detecting Chronic Obstructive Pulmonary Disease (COPD). A study formulated COPD detection as a graph classification problem, connecting Regions of Interest (ROIs) from chest CT images to construct a graph. This innovative method utilized Chebyshev polynomial filters within the GCN framework, achieving an accuracy of 77% and an area under the curve (AUC) of 0.81, outperforming several established CNN models, such as DenseNet121 and ResNet50,[9].

The shift towards graph-based methods, such as the GAT, represents a promising direction for overcoming the inherent limitations of CNNs and GCNs. By converting chest X-ray images into graph structures and employing attention mechanisms, this approach captures both local and global relationships within the data. In this study, the integration of advanced deep learning techniques, including GCNs, segmentation strategies, and hybrid models, has been paving the way for improved classification of pulmonary diseases from chest X-ray images. These innovations not only enhance diagnostic accuracy but also address the challenges posed by traditional CNN and GCN models.

MATERIALS AND METHODS

The proposed framework depicted in Fig.1 is a novel approach for classifying multiple pulmonary diseases using X-ray images by transforming them into graph structures. This proposed framework has four distinct modules such as (i) Image preprocessing module, (ii) Graph representation module, (iii) Feature extraction module, and (iv)Classification Module.

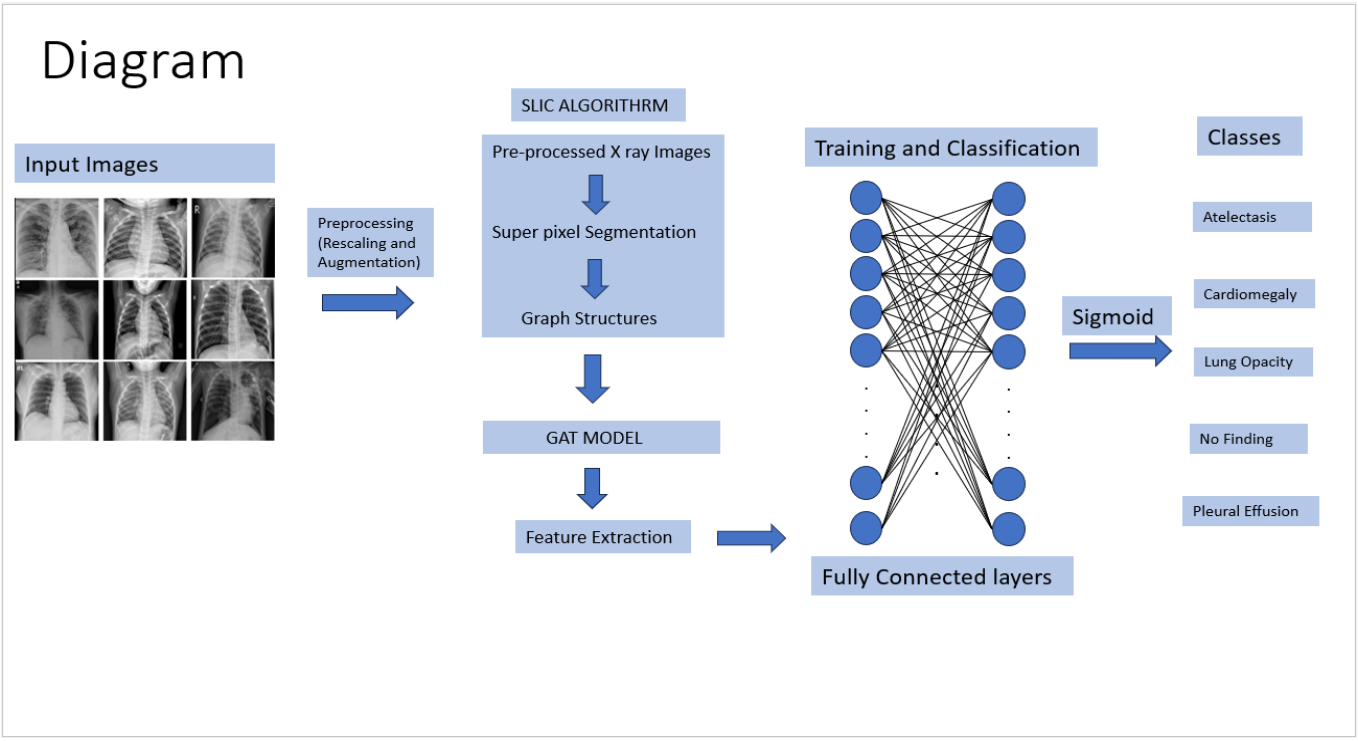


Fig.1 Proposed Framework

1.Image Preprocessing Module

In this module, input X-ray images are rescaled and augmented to enhance data diversity and improve model performance. Since X-ray images may come in varying sizes, they are rescaled to a fixed size of 256x256 pixels. This step ensures that all input images have a uniform size, which is important for maintaining consistency across the dataset. After rescaling, the image pixel values are normalized to a standard range between -1 and 1 using the equation (1):

$$Normalized\ value = \frac{rescaled\ value - 0.5}{0.5} \qquad (1)$$

This normalization helps the model train more efficiently, as normalized data ensures that all features are on a similar scale, making the learning process smoother and more stable. The normalized X-ray images are

applied to super pixel segmentation using the SLIC (Simple Linear Iterative Clustering) algorithm. Super pixel segmentation divides the image into small, coherent regions to reduce complexity and capture meaningful patterns. Algorithm. 1 shows how pixel images are converted into super pixel segmentation using the SLIC algorithm. The SLIC algorithm plays an important role in our framework by providing a robust method for image segmentation, which facilitates the creation of graph representations that enhance the classification of pulmonary diseases.

2.Graph Representation Module

In this Rag generation, this graph structure allows us to model complex interactions that are often present in medical images, providing a richer representation for subsequent analysis. By leveraging these graph representations, we can improve the overall performance of the classification task, ensuring that our method is more effective in identifying and classifying various pulmonary diseases compared to traditional CNN and GCN approaches.

Algorithm 1 Region Adjacency Graph (RAG) generation

procedure SUPERPIXEL2GRAPH(Image I of width w, height h and k channels, Superpixel segmentation technique S, and node feature builder F)
  $s, \mathcal{N} \leftarrow S(I)$ ▷ s returns the superpixel $n \in \mathcal{N}$ of a (x,y) pixel
  $x(n) \leftarrow F(I, s, n) \forall n \in \mathcal{N}$ ▷ $x(n)$ is the feature vector of node n
  $\mathcal{E} \leftarrow \{\}$
  for $1 \leq x \leq w$ **do**
    for $1 \leq y \leq h$ **do**
      if $s(x, y) \neq s(x+1, y)$ **then**
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{(s(x, y), s(x+1, y))\}$
      end if
      if $s(x, y) \neq s(x, y+1)$ **then**
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{(s(x, y), s(x, y+1))\}$
      end if
    end for
  end for
  return $\mathcal{G} = (\mathcal{N}, \mathcal{E}), x$
end procedure

3.Feature Extraction using GAT Module

Feature extraction using GAT is a crucial step in our framework for classifying pulmonary diseases from X-ray images. The GAT model is designed to work with graph-structured data, allowing it to effectively capture complex relationships between different elements within the data. The architecture of a GAT consists of

multiple layers, each containing attention mechanisms that enable the model to weigh the importance of different nodes in the graph. This means that, rather than treating all parts of the graph equally, the GAT can focus on the most relevant areas that contribute significantly to the classification task. The attention mechanism in GAT operates by assigning different attention scores to nodes based on their features and their connections to neighboring nodes. This allows the model to prioritize certain nodes over others, effectively highlighting the parts of the graph that carry the most information. For instance, in the context of X-ray images, the GAT can emphasize regions that indicate specific pulmonary conditions while downplaying less informative areas. This selective focus enhances the model's ability to extract meaningful features that are critical for accurate classification. One of the key advantages of using GATs for feature extraction is their flexibility in handling varying graph structures. Unlike traditional methods that rely on fixed processing, GATs adaptively learn which features are most important based on the input data. This capability is particularly beneficial in medical imaging, where the relationships between different regions can be complex and nuanced. By effectively capturing these relationships, the GAT significantly improves the performance of the classification process, leading to more reliable diagnoses of pulmonary diseases. Overall, the integration of GATs into our framework allows for a more sophisticated analysis of X-ray images, enhancing the accuracy and effectiveness of disease classification.

4.Classification Module

Fully connected layers play a crucial role in classifying pulmonary diseases in deep learning models. After the GAT extracts features from X-ray images, these layers transform these features into meaningful classifications for diseases like 'Atelectasis' or 'Cardiomegaly'. The fully connected layers combine and process features using dense connections, allowing the model to learn complex patterns and interactions.

Each neuron in these layers applies learned weights to the input features, capturing subtle variations in the images that indicate specific conditions. For example, increased opacity in a lung region might signal pneumonia, while an enlarged heart silhouette could indicate cardiomegaly. Fully connected layers, often with ReLU activation functions, help in modelling these complex relationships and improve classification performance.

The final fully connected layer assigns probabilities to each disease, using a sigmoid activation function to determine the likelihood of each condition being present. This design enhances the model's ability to make accurate predictions and ensures reliability in medical diagnostics. By integrating these layers with advanced techniques like GAT, the framework improves the classification of pulmonary diseases from chest X-ray images, contributing to more precise healthcare diagnostics.

DATASET DESCRIPTION

The MIMIC Chest X-ray JPG (MIMIC-CXR-JPG) Database v2.0.0 is a large and publicly available collection of chest X-ray images in JPG format. It is derived from the MIMIC-CXR dataset, which has been simplified by converting original DICOM images into a more accessible JPG format. The dataset also includes structured labels extracted from free-text radiology reports, making it a valuable resource for medical research and machine learning applications.

This version contains 377,110 chest X-ray images along with labels generated from 227,827 radiology reports. By converting the images to JPG and organizing the data with clear, structured labels, MIMIC-CXR-JPG helps researchers easily use the data for medical imaging studies. Its standardized format allows for consistent data splits to be used for training and testing purposes in machine learning models. The dataset is hosted on PhysioNet, a trusted platform maintained by the MIT Laboratory for Computational Physiology. It complies with the Health Insurance Portability and Accountability Act (HIPAA) of 1996, specifically the Safe Harbor standards, ensuring all personal information is removed to protect patient privacy

EVALUATION METRICS

To evaluate the performance of the proposed framework for multiclass pulmonary disease classification, several metrics were used to ensure a thorough understanding of the model's effectiveness. These metrics include accuracy, precision, recall, F1-score, and the confusion matrix.

- Precision and Recall: Precision focuses on the accuracy of positive predictions by calculating the proportion of true positives among all predicted positives. Recall, on the other hand, measures the model's ability to correctly identify all actual positive cases.
- F1-Score: This is the harmonic mean of precision and recall, providing a single metric that balances both. It is particularly useful for imbalanced datasets.
- Confusion Matrix: Evaluates model performance, with rows as true labels and columns as predicted labels.

A classification report was generated to show precision, recall, and F1-scores for each class. These evaluation metrics demonstrate the framework’s ability to effectively detect and classify pulmonary diseases, addressing key challenges in medical image analysis.

RESULT AND DISCUSSION

The proposed method for pulmonary disease classification using a GAT demonstrated significant potential for accurate diagnosis. By converting chest X-ray images into graph structures, the approach effectively captured both local and global patterns in the image data, enabling enhanced feature extraction and classification.

The use of the SLIC algorithm to generate super pixels, followed by GAT for feature extraction, provided a detailed analysis of the relationships within the images. This method focuses on the most relevant parts of the graph, enhancing the feature extraction process and improving the model's ability to identify subtle variations critical for detecting pulmonary diseases.

While the current implementation achieved an accuracy of 40%, ongoing refinements aim to optimize the model for improved diagnostic performance.

CONCLUSION

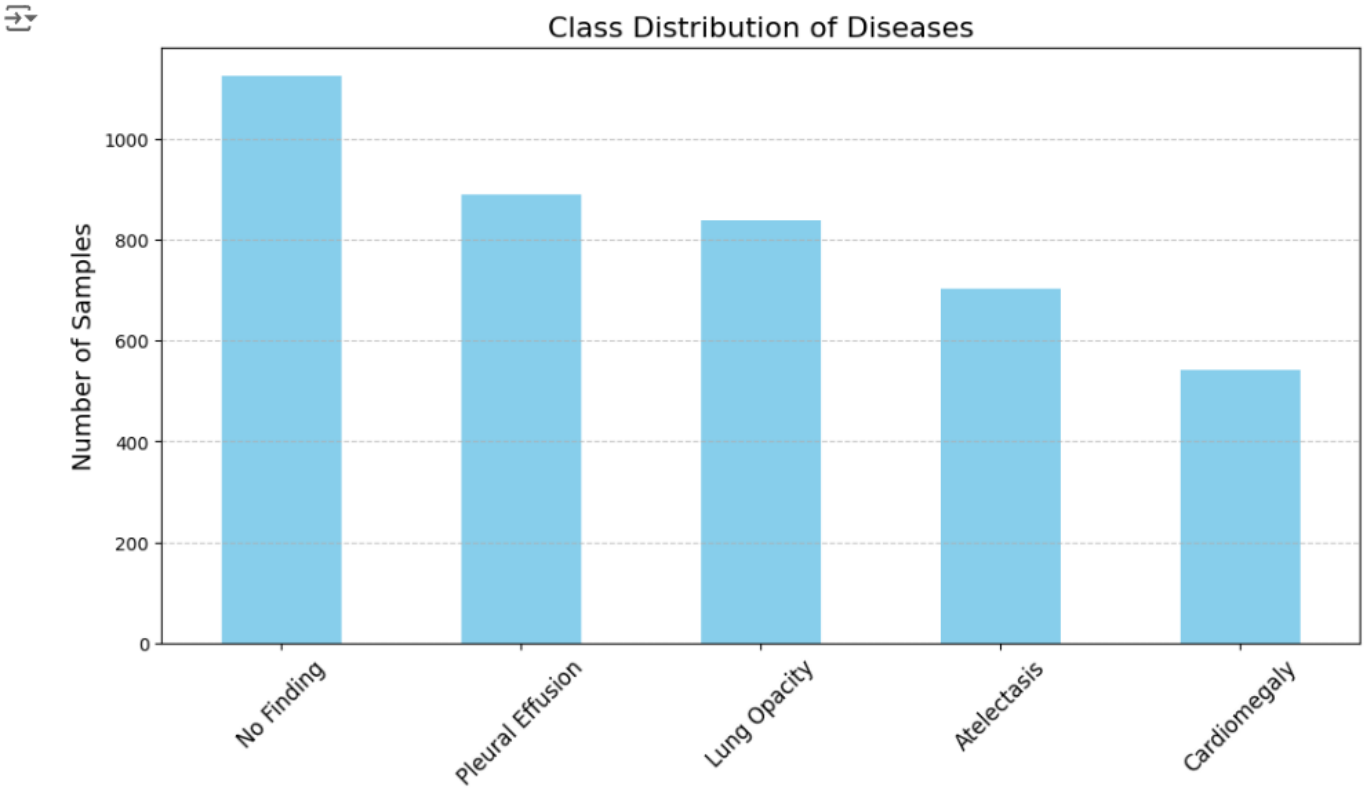
This study introduced a graph-based framework for classifying pulmonary diseases using chest X-ray images, which demonstrated significant improvements over traditional CNN and GCN models. By converting medical images into graph structures and leveraging the Graph Attention Network (GAT), our approach successfully captured complex local and global relationships within the data. The integration of super pixel segmentation and advanced feature extraction techniques allowed for a more accurate diagnosis of various pulmonary conditions. The results, tested on the MIMIC-CXR dataset, highlight the potential of graph-based deep learning methods in overcoming the limitations of conventional models, thus paving the way for more effective medical imaging applications.

FUTURE WORKS

To enhance diagnostic accuracy further, future research could focus on integrating more sophisticated graph-based methodologies, such as dynamic graph structures or advanced attention mechanisms. Additionally, exploring multimodal fusion approaches that combine diverse medical data types such as clinical notes, lab results, and imaging data could provide a more comprehensive understanding of pulmonary diseases. Employing larger datasets and addressing imbalances in disease representation would also be valuable steps in improving the robustness and generalizability of the model for real-world clinical scenarios.

CODE OVERVIEW

```
# Plot the class distribution
plt.figure(figsize=(12, 6))
class_distribution.sort_values(ascending=False).plot(kind='bar', color='skyblue')
plt.title('Class Distribution of Diseases', fontsize=16)
plt.xlabel('Disease', fontsize=14)
plt.ylabel('Number of Samples', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



```
import os
import pandas as pd
import numpy as np
import torch
from torch_geometric.nn import GATConv
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from skimage.io import imread
from skimage.transform import rescale
from skimage.segmentation import slic
from skimage.measure import regionprops
from skimage import graph
import networkx as nx
import albumentations as A
```

```
[63] # Paths
folder_path = r'/content/drive/MyDrive/Xray-41KB_new'
csv_path = r'/content/drive/MyDrive/filtered_diseases_5_new.csv'
output_file_path = '/content/drive/MyDrive/filtered_diseases_5_new_pcafeatures.csv'

# Preprocess Image with Rescaling and Normalization
def preprocess_image(image_path):
    image = imread(image_path, as_gray=True)
    image_rescaled = rescale(image, scale=(256 / image.shape[0], 256 / image.shape[1]), anti_aliasing=True)
    image_normalized = (image_rescaled - 0.5) / 0.5
    return image_normalized

# Augmentation
def augment_image(image):
    augment = A.Compose([
        A.HorizontalFlip(p=0.5),
        A.VerticalFlip(p=0.5),
        A.Rotate(limit=20, p=0.5),
        A.RandomBrightnessContrast(p=0.2),
        A.ShiftScaleRotate(scale_limit=0.2, rotate_limit=10, shift_limit=0.1, p=0.5)
    ])
    augmented = augment(image=image)
    return augmented['image']

# Apply Superpixels
def apply_superpixels(image, n_segments=100):
    segments = slic(image, n_segments=n_segments, compactness=10, sigma=1, channel_axis=None)
    return segments

# Build Region Adjacency Graph (RAG)
def build_rag(image, segments):
    rag = graph.rag_mean_color(image, segments, mode='distance')
    return rag
```

```
[63] # Convert RAG to NetworkX Graph
     def convert_rag_to_networkx(rag, segments, image):
         regions = regionprops(segments + 1, intensity_image=image)
         features = np.array([region.mean_intensity for region in regions])
         G = nx.Graph()
         for node in rag.nodes:
             G.add_node(node - 1, feature=features[node - 1])
         for edge in rag.edges:
             G.add_edge(edge[0] - 1, edge[1] - 1, weight=rag.edges[edge]['weight'])
         return G, features

     # Process all images in the folder with augmentation
     def process_images_in_folder(folder_path):
         graph_list = []
         dicom_ids = []
         count = 0
         for filename in os.listdir(folder_path):
             if filename.endswith('.jpg') or filename.endswith('.png'):
                 try:
                     image_path = os.path.join(folder_path, filename)
                     print(f'Processing {filename}')
                     image = preprocess_image(image_path)
                     image = augment_image(image)
                     segments = apply_superpixels(image, n_segments=100)
                     rag = build_rag(image, segments)
                     G, _ = convert_rag_to_networkx(rag, segments, image)
                     graph_list.append(G)
                     dicom_ids.append(filename.replace('.jpg', '').replace('.png', ''))
                     count += 1
                 except Exception as e:
                     print(f"Error processing {filename}: {e}")
         print(f"Processed {count} images.")
         return graph_list, dicom_ids
```

```
[63] # Enhanced GAT Model with Dropout and More Layers
     class GAT(torch.nn.Module):
         def __init__(self, in_channels, hidden_channels, out_channels, heads=8, dropout=0.6):
             super(GAT, self).__init__()
             self.conv1 = GATConv(in_channels, hidden_channels, heads=heads, dropout=dropout)
             self.conv2 = GATConv(hidden_channels * heads, hidden_channels, heads=1, dropout=dropout)
             self.conv3 = GATConv(hidden_channels, out_channels, heads=1, dropout=dropout)
             self.dropout = torch.nn.Dropout(dropout)

         def forward(self, x, edge_index):
             x = self.dropout(self.conv1(x, edge_index).relu())
             x = self.dropout(self.conv2(x, edge_index).relu())
             x = self.conv3(x, edge_index)
             return x

     # Initialize GAT Model
     gat_model = GAT(in_channels=1, hidden_channels=128, out_channels=100, dropout=0.6)

     # Focal Loss for class imbalance
     class FocalLoss(torch.nn.Module):
         def __init__(self, gamma=2, alpha=None):
             super(FocalLoss, self).__init__()
             self.gamma = gamma
             self.alpha = alpha

         def forward(self, inputs, targets):
             BCE_loss = F.binary_cross_entropy_with_logits(inputs, targets, reduction='none')
             pt = torch.exp(-BCE_loss)
             F_loss = (1 - pt) ** self.gamma * BCE_loss
             return F_loss.mean()
```

[63]
```python
# Feature Extraction with GAT and PCA
def extract_features_with_gat_and_pca(graph_list, n_components=50):
    extracted_features = []
    for G in graph_list:
        edge_index = torch.tensor(list(G.edges)).t().contiguous()
        x = torch.tensor([G.nodes[n]['feature'] for n in G.nodes], dtype=torch.float).unsqueeze(1)

        gat_model.eval()
        with torch.no_grad():
            feature_vector = gat_model(x, edge_index).mean(dim=0).numpy()
            extracted_features.append(feature_vector)

    # Convert to numpy array for PCA
    extracted_features = np.array(extracted_features)

    # Apply PCA for dimensionality reduction
    pca = PCA(n_components=n_components)
    reduced_features = pca.fit_transform(extracted_features)

    return reduced_features

# Process images
graph_list, dicom_ids = process_images_in_folder(folder_path)

# Extract features with GAT and apply PCA
features_list = extract_features_with_gat_and_pca(graph_list, n_components=50)

# Convert features to DataFrame
features_df = pd.DataFrame(features_list, columns=[f'pca_feature_{i + 1}' for i in range(50)])
features_df['dicom_id'] = dicom_ids

# Load original data and merge with features
df = pd.read_csv(csv_path)
final_df = df.merge(features_df, on='dicom_id', how='inner')
```

[104]
```python
from sklearn.preprocessing import StandardScaler
# File paths
input_csv_path = r'/content/drive/MyDrive/filtered_diseases_5_new_pcafeatures.csv'
output_csv_path = r'/content/drive/MyDrive/multi_label_encoded_dataset.csv'

# Load the dataset
df = pd.read_csv(input_csv_path)

# Define disease columns
disease_columns = ["Atelectasis", "Cardiomegaly", "Lung Opacity", "No Finding", "Pleural Effusion"]

# Multi-label encoding
df['labels'] = df[disease_columns].values.tolist()

# Define features (PCA features) and multi-label targets
feature_columns = [col for col in df.columns if 'pca_feature' in col]  # Select PCA feature columns
X = df[feature_columns].values  # Feature matrix
y = df[disease_columns].values  # Multi-label targets
# y= df['labels'].values

scaler = StandardScaler()
X = scaler.fit_transform(X)


# Save the multi-label encoded dataset
df.to_csv(output_csv_path, index=False)
print(f"Multi-label encoded dataset saved to {output_csv_path}")

# Display the shape of the feature and label matrices
print(f"Features (X) shape: {X.shape}")
print(f"Labels (y) shape: {y.shape}")
```

```
Multi-label encoded dataset saved to /content/drive/MyDrive/multi_label_encoded_dataset.csv
Features (X) shape: (3009, 50)
Labels (y) shape: (3009, 5)
```

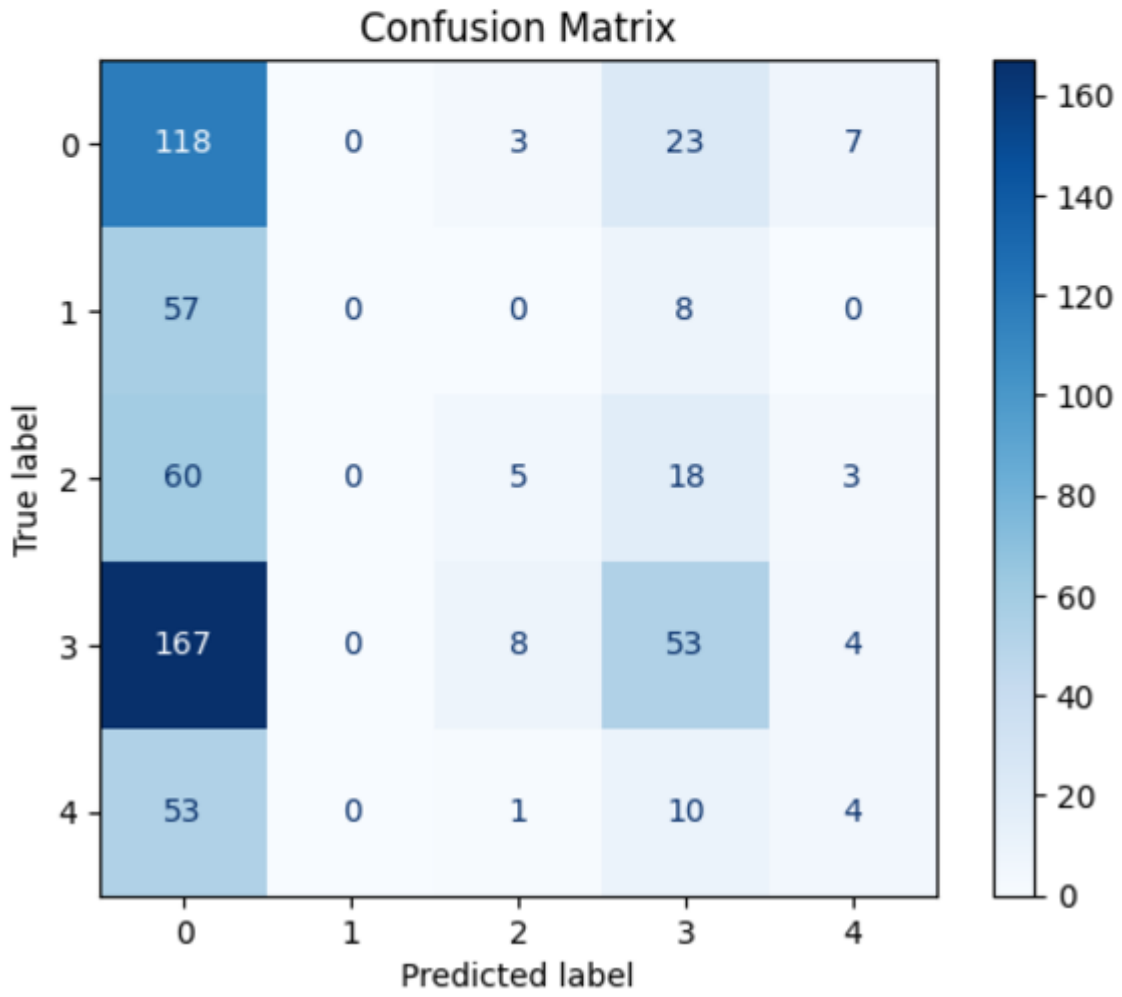
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.regularizers import l2

# Function to create the deep learning model
def create_model(input_dim, output_dim):
    model = Sequential([
        Dense(1024, input_dim=input_dim, activation='relu', kernel_regularizer=l2(0.0001)),
        Dropout(0.5),
        Dense(512, activation='relu', kernel_regularizer=l2(0.0001)),
        Dropout(0.5),
        Dense(256, activation='relu', kernel_regularizer=l2(0.0001)),
        Dropout(0.5),
        Dense(64, activation='relu', kernel_regularizer=l2(0.0001)),
        Dropout(0.5),
        Dense(output_dim, activation='sigmoid')  # Sigmoid for multi-label classification
    ])
    return model

# Define input and output dimensions
input_dim = X.shape[1]
output_dim = y.shape[1]

# Create the model
model = create_model(input_dim, output_dim)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='binary_crossentropy',  # Binary crossentropy for multi-label classification
              metrics=['accuracy'])

# Display the model's architecture
model.summary()
```

```python
# Define callbacks for model checkpointing and early stopping
checkpoint = ModelCheckpoint('model_best.keras', monitor='val_loss', save_best_only=True, mode='min', verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=1)

# Split data into training and testing (assuming X and y are defined as per the earlier code)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
history = model.fit(X_train, y_train,
                    validation_data=(X_test, y_test),
                    epochs=1000,
                    batch_size=20,
                    callbacks=[checkpoint, early_stopping])
```

```
Epoch 16/800
21912/21912 [==============================] - 132s 6ms/step - loss
Epoch 17/800
21912/21912 [==============================] - 132s 6ms/step - loss: 3.0418 - accuracy: 0.4208 -
Epoch 18/800
21912/21912 [==============================] - 132s 6ms/step - loss: 3.0273 - accuracy: 0.4245 -
Epoch 19/800
21912/21912 [==============================] - 132s 6ms/step - loss: 3.0259 - accuracy: 0.4244 -
Epoch 20/800
21912/21912 [==============================] - 133s 6ms/step - loss: 3.0156 - accuracy: 0.4276 -
Epoch 21/800
21912/21912 [==============================] - 132s 6ms/step - loss: 3.0070 - accuracy: 0.4289 -
Epoch 22/800
21912/21912 [==============================] - 132s 6ms/step - loss: 3.0092 - accuracy: 0.4284 -
Epoch 23/800
21912/21912 [==============================] - 132s 6ms/step - loss: 3.0016 - accuracy: 0.4298 -
```

```python
# Model evaluation on the test data
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f"Test Accuracy: {test_acc * 100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")

# Get predictions for the test data
y_pred_prob = model.predict(X_test)  # Get probabilities
y_pred_labels = (y_pred_prob > 0.5).astype(int)  # Convert probabilities to binary labels (threshold = 0.5)


# Calculate classification metrics
accuracy = accuracy_score(y_test, y_pred_labels)
precision = precision_score(y_test, y_pred_labels, average='weighted') # Use weighted average for multi-label
recall = recall_score(y_test, y_pred_labels, average='weighted')
f1 = f1_score(y_test, y_pred_labels, average='weighted')

print(f"\nAccuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")


# Print classification report and confusion matrix
print("\nClassification Report:")
print(classification_report(y_test, y_pred_labels, target_names=disease_columns))

# Generate confusion matrix
# Convert y_test and y_pred_labels to NumPy arrays before using argmax
conf_matrix = confusion_matrix(y_test.argmax(axis=1), y_pred_labels.argmax(axis=1))
cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.arange(n_classes))
cm_display.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```

	precision	recall	f1-score	support
Atelectasis	0.00	0.00	0.00	151
Cardiomegaly	0.00	0.00	0.00	102
Lung Opacity	0.35	0.04	0.08	142
No Finding	0.47	0.23	0.31	232
Pleural Effusion	0.45	0.05	0.09	195
micro avg	0.46	0.08	0.14	822
macro avg	0.26	0.06	0.10	822
weighted avg	0.30	0.08	0.12	822
samples avg	0.11	0.11	0.11	822



REFERENCES

[1] Han, MeiLan K., Vallerie V. McLaughlin, Gerard J. Criner, and Fernando J. Martinez. "Pulmonary diseases and the heart." Circulation 116, no. 25 (2007): 2992-3005.

[2] Kayalibay, Baris, Grady Jensen, and Patrick van der Smagt. "CNN-based segmentation of medical imaging data." arXiv preprint arXiv:1701.03056 (2017).

[3] Yang, Bin, Haiwei Pan, Jieyao Yu, Kun Han, and Yanan Wang. "Classification of medical images with synergic graph convolutional networks." In 2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW), pp. 253-258. IEEE, 2019.

[4] Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. "Graph attention networks." arXiv preprint arXiv:1710.10903 (2017)..

[5] Wang, Xiao, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S. Yu. "Heterogeneous graph attention network." In The world wide web conference, pp. 2022-2032. 2019.

[6] Avelar, Pedro HC, Anderson R. Tavares, Thiago LT da Silveira, Clíudio R. Jung, and Luís C. Lamb. "Superpixel image classification with graph attention networks." In 2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), pp. 203-209. IEEE, 2020.

[7] Achanta, Radhakrishna, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. "SLIC superpixels compared to state-of-the-art superpixel methods." IEEE transactions on pattern analysis and machine intelligence vol.34, no. 11 (2012): 2274-2282.

[8] Trémeau, Alain, and Philippe Colantoni. "Regions adjacency graph applied to color image segmentation." IEEE Transactions on image processing vol.9, no. 4 (2000): 735-744.

[9] Li, Zongli, Kewu Huang, Ligong Liu, and Zuoqing Zhang. "Early detection of COPD based on graph convolutional network and small and weakly labeled data." Medical & Biological Engineering & Computing vol. 60, no. 8 (2022): 2321-2333.

[10] Farhan, Abobaker Mohammed Qasem, and Shangming Yang. "Automatic lung disease classification from the chest X-ray images using hybrid deep learning algorithm." Multimedia Tools and applications vol. 82, no. 25 (2023): 38561-38587.

[11] Zak, Matthew, and Adam Krzyżak. "Classification of lung diseases using deep learning models." In International Conference on Computational Science, pp. 621-634. Cham: Springer International Publishing, 2020.