TABLE OF CONTENT

ABSTRACT

Lung diseases remain a significant global health concern, affecting millions of individuals each year. Early diagnosis and timely treatment are essential to reducing complications and mortality rates. Medical imaging techniques such as X-rays, CT scans, and MRIs play a crucial role in detecting pulmonary abnormalities. This project leverages deep learning to analyze CT scan volumes and develop automated models for classifying lung diseases such as COVID-19, pneumonia (CAP), and cancer.

Initially, 2D CNN models were employed to classify individual CT slices, followed by the implementation of 3D CNN models that utilize volumetric information for improved accuracy. Comparative analysis demonstrated that 3D CNNs outperformed 2D CNNs in classification performance. To further enhance accuracy, a specialized 3D VoxResNet architecture was introduced, which delivered superior results—especially in identifying cancer subtypes like adenocarcinoma, squamous cell carcinoma, and large cell carcinoma.

Finally, Explainable AI (XAI) techniques, specifically 3D Grad-CAM, were applied to visualize the regions of the lung that most influenced the model's decisions. This improved interpretability and provided greater clinical relevance. The study concludes that 3D VoxResNet, combined with XAI, offers a robust solution for accurate and explainable lung disease classification.

CHAPTER 1

INTRODUCTION

Lung diseases pose a significant global health challenge, affecting millions of individuals and contributing to high mortality rates worldwide. Conditions such as pneumonia, COVID-19, and lung cancer not only impact individual health but also place a substantial burden on healthcare systems. The growing prevalence of these diseases underscores the critical need for early and accurate detection, which can significantly enhance treatment outcomes and improve patient survival rates.

This project focuses on developing an advanced lung disease classification system utilizing deep learning techniques, specifically Convolutional Neural Networks (CNNs). By leveraging medical imaging data, the system aims to enhance diagnostic accuracy and efficiency in detecting various lung conditions, including Normal, Community-Acquired Pneumonia (CAP), COVID-19, and lung cancer subtypes such as Adenocarcinoma, Squamous Cell Carcinoma, and Large Cell Carcinoma.

The study employs both **2D and 3D CNN** architectures to compare their effectiveness in analyzing CT scan images for disease classification. The dataset consists of labeled lung CT scan images, categorized based on disease type, to train and evaluate the performance of different models. Through rigorous experimentation, the project aims to determine the most effective deep learning approach for lung disease diagnosis, considering factors such as accuracy, computational efficiency, and real-world applicability.

To further improve the system’s performance, the project was extended by implementing a more powerful deep learning architecture **3D VoxResNet** specifically designed to handle volumetric medical imaging data. VoxResNet's ability to capture richer spatial features in 3D CT scans led to improved classification results, particularly in complex cases such as lung cancer subtypes.

Additionally, to enhance model interpretability and transparency, **Explainable AI (XAI)** techniques were incorporated using Grad-CAM (Gradient-weighted Class Activation Mapping). **Grad-CAM** was applied to the 3D VoxResNet model to visualize the critical regions within CT scans that influenced the model's decisions, providing valuable insights for radiologists and increasing trust in AI-assisted diagnosis.

By utilizing advanced computational methods for medical imaging analysis, this research seeks to contribute to the early detection of lung diseases, ultimately assisting radiologists and healthcare professionals in making more precise diagnoses. The outcomes of this study have the potential to improve patient care by enabling timely interventions, reducing diagnostic delays, and supporting the broader goal of enhancing public health through technological advancements.

CHAPTER 2

LITERATURE SURVEY

Lung diseases, including pneumonia, COVID-19, and lung cancer, pose a major global health challenge. Recent advancements in deep learning have enabled significant progress in early detection and classification of these diseases using medical imaging techniques. Various research studies have explored different CNN architectures to improve classification accuracy.

A study [1] by Muhammed Oguz Tas and Hasan Serhan Yavuz (2022) proposed a 3D CNN-based approach for lung cancer survival prediction. The authors introduced a novel GTV1-SliceNum feature and utilized the PEN-BCE loss function, achieving superior classification performance in CT scan analysis.

Another research [2] by Sneha Balannolla (2021) focused on detecting and classifying lung carcinoma using CT scans. This study compared different CNN architectures and demonstrated the efficiency of deep feature extraction in distinguishing various lung cancer types.

A work [3] by Jalil Ahmed (2020) investigated COPD classification in CT images using 3D CNN models. The study highlighted that volumetric feature extraction provides higher accuracy than traditional 2D image analysis, reinforcing the importance of 3D deep learning techniques in medical imaging.

A notable study [4] by Hao Chen et al. (2020) introduced VoxResNet, a deep voxelwise residual network initially developed for volumetric brain segmentation. Due to its ability to capture complex spatial features, VoxResNet has been effectively adapted for lung CT analysis, showing improved accuracy in 3D medical image classification tasks.

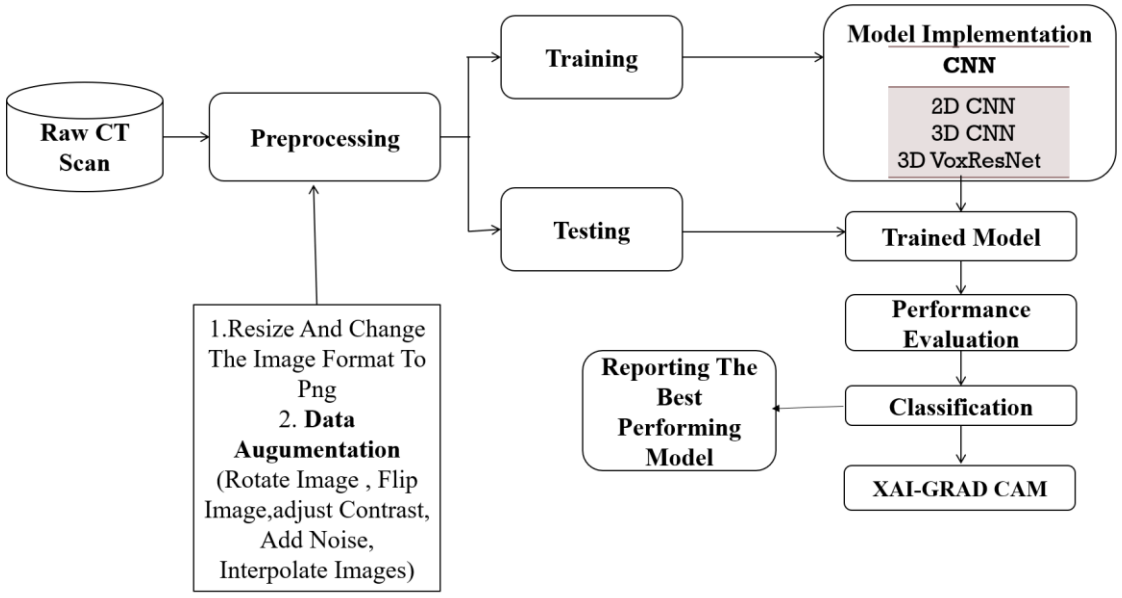
To improve model transparency, a recent study [5] by Talaat et al. (2024) introduced a 3D Inception-ResNet V2 model integrated with Grad-CAM for breast cancer classification. This approach not only achieved high diagnostic accuracy on volumetric medical data but also provided interpretable visualizations of the model's decision-making process, thereby reinforcing the potential of Explainable AI (XAI) in empowering radiologists and increasing clinical trust in AI-assisted diagnostics.

These studies illustrate the growing adoption of deep learning methods for lung disease classification. With 3D CNN architectures and residual networks like VoxResNet showing improved accuracy in analyzing volumetric CT data, and XAI techniques like Grad-CAM enhancing interpretability, future advancements may focus on hybrid models, ensemble learning strategies, and optimized loss functions for even greater diagnostic performance.

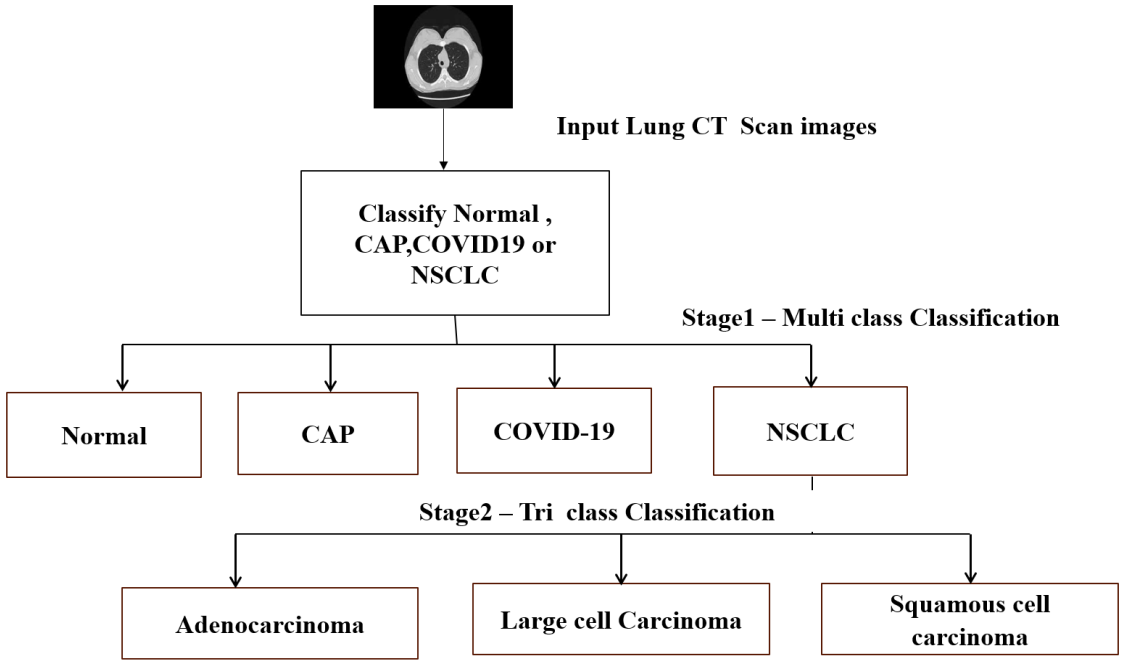
CHAPTER 3

SYSTEM DESIGN

3.1 Process Flow Diagram :



3.2 Flow Diagram of the Work



CHAPTER 4

METHODOLOGY

4.1 Overview

The proposed system aims to classify lung diseases using deep learning techniques, specifically 2D and 3D Convolutional Neural Networks (CNNs), applied to CT scan images. The goal is to develop an automated system capable of distinguishing between Normal, Community-Acquired Pneumonia (CAP), COVID-19, and Non-Small Cell Lung Cancer subtypes, including Adenocarcinoma, Squamous Cell Carcinoma, and Large Cell Carcinoma**.** The study evaluates the performance of both 2D and 3D CNN models to determine the most effective approach for lung disease classification. To improve accuracy on volumetric data, a 3D VoxResNet architecture is implemented for better spatial feature learning. Additionally, Grad-CAM based XAI techniques are used to visualize model decisions and enhance interpretability.

4.2 Data Collection and Organization

The dataset comprises 6,720 CT scan images categorized into six classes: Normal, Community-Acquired Pneumonia (CAP), COVID-19, Adenocarcinoma, Squamous Cell Carcinoma, and Large Cell Carcinoma. Each disease category consists of 30 patient folders, with each folder containing 32 CT scan slices in PNG format. The dataset is systematically organized into two main directories to ensure proper segmentation for training and evaluation. The LungCT directory contains images for Normal, CAP, COVID-19, and an equal distribution of Non-Small Cell Lung Cancer (NSCLC) cases, ensuring a balanced representation of these conditions. The Cancer directory specifically holds CT scans of Adenocarcinoma, Squamous Cell Carcinoma, and Large Cell Carcinoma, allowing focused analysis of different lung cancer subtypes. This structured organization provides a well-balanced dataset that facilitates effective model training and performance evaluation, ensuring reliable classification of various lung diseases. To improve the performance of 3D VoxResNet, I further increased the dataset size to 13,440 images, enabling better feature learning and generalization. After training the model, I applied Explainable AI techniques using Grad-CAM to visualize the key regions that influenced the model's classification decisions, enhancing interpretability and trust in the model's predictions.

Figure 1 – Sample Dataset

4.3 Data Pre-Processing :

The preprocessing stage is a crucial step in preparing the dataset for effective deep-learning-based lung disease classification. The following steps were performed to ensure consistency, enhance model performance, and improve generalization:

4.3.1. Image Resizing and Format Conversion

All CT scan images were resized to a uniform dimension of **(256 × 256)** pixels. This standardization ensures that the deep learning model receives inputs of the same size, reducing computational complexity and improving training efficiency. Additionally, all images were converted to the **PNG format**, ensuring lossless compression and maintaining image quality for better feature extraction.

4.3.2. Data Augmentation

To increase dataset variability and improve the model's ability to generalize across different cases, various augmentation techniques were applied:

1. **Rotation:** Each image was rotated randomly by 90°, 180°, or 270°, simulating different orientations of CT scans encountered in real-world scenarios.
2. **Flipping:** The images were flipped horizontally or vertically, enhancing the model's ability to recognize features regardless of scan direction.
3. **Contrast Adjustment:** The contrast of images was modified using a random factor, ensuring that the model learns to detect lung abnormalities under varying scan qualities.
4. **Noise Addition:** Gaussian noise was introduced to the images, mimicking real-world distortions such as scanner noise, thereby improving model robustness.
5. **Interpolation:** Images were blended using weighted interpolation, creating variations in pixel intensity and making the model more adaptable to different scanning conditions.

4.3.3. Synthetic Slice Generation

To ensure a balanced dataset, a detailed **patient-wise analysis** was performed to identify cases where a particular disease had fewer than **32 CT slices** per patient. For these cases, synthetic image slices were generated using augmentation

techniques to compensate for missing data, ensuring that all patient cases were equally represented in the dataset.

4.3.4. Train-Test Split

Once preprocessing was completed, the dataset was split into **training and testing sets** to enable model evaluation:

- **Training Set:** 80% of the dataset was allocated for model training, ensuring the model learns complex patterns effectively.
- **Testing Set:** The remaining 20% was reserved for performance evaluation, allowing assessment of how well the model generalizes to unseen data.

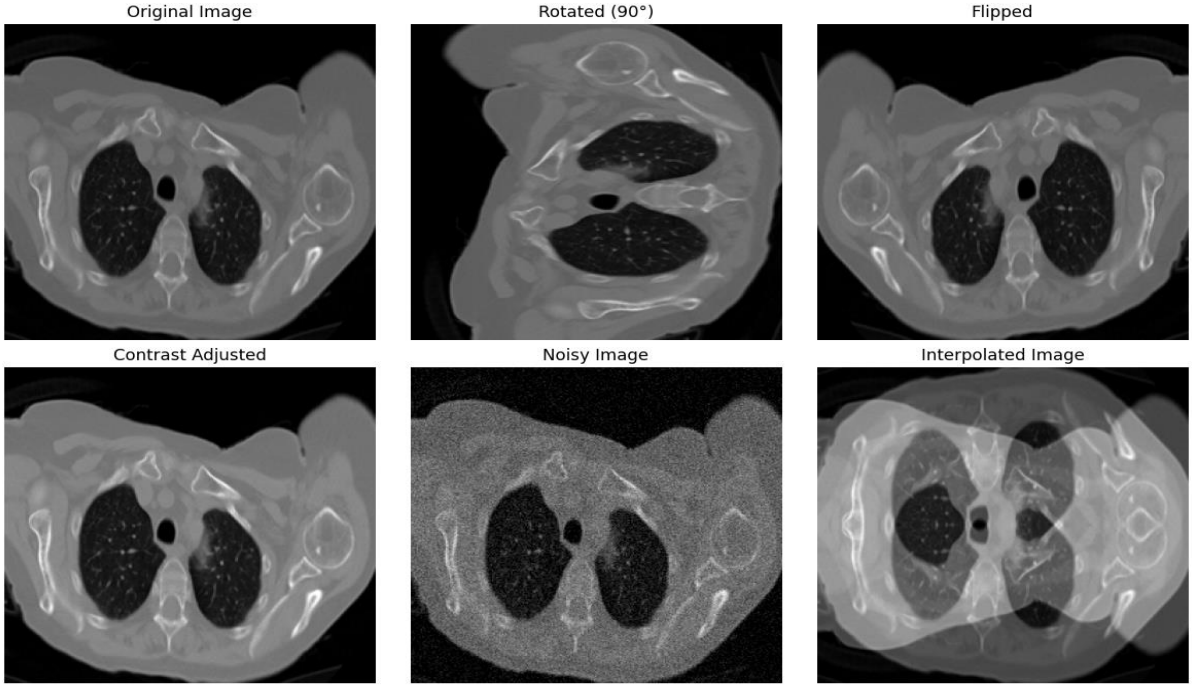


Figure 2 - Visual Output of Augmented Images

4.4. Algorithm Implementation

4.4.1 General Overview

Deep learning-based lung disease classification requires a robust and efficient algorithm to analyze CT scan images and accurately predict disease categories. Convolutional Neural Networks (CNNs) have proven to be highly effective in image classification tasks, as they automatically extract meaningful features without requiring manual feature engineering. In this project, a two-stage classification approach is implemented using 2D CNNs and 3D CNNs, where the

first stage performs broad classification, and the second stage refines the classification of lung cancer subtypes.

4.4.2 Two-Stage Classification Approach

Stage 1: Multi-Class Classification (Normal, CAP, COVID-19, Cancer)

The first-stage 2D CNN model classifies CT scans into four broad categories:

- **Normal** – Healthy lung CT scan.
- **Community-Acquired Pneumonia (CAP)** – Lungs infected with bacterial or viral pneumonia.
- **COVID-19** – Lungs affected by the SARS-CoV-2 virus.
- **Cancer** – If the scan shows signs of Non Small Cell Lung Cancer, all the NSCLC types are grouped under a single "Cancer" category in this stage.

Stage 2: Cancer Subtype Classification (Adenocarcinoma , Large Cell Carcinoma, Squamous Cell Carcinoma)

If the first-stage model predicts a scan as Cancer, it is forwarded to the second stage 2D CNN model for a more detailed classification into three Non-Small Cell Lung Cancer (NSCLC) subtypes:

- Adenocarcinoma
- Large Cell Carcinoma
- Squamous Cell Carcinoma

This two-stage approach ensures a hierarchical classification that first identifies whether a patient has lung cancer before determining the exact type.

I 2D Convolutional Neural Network (2D CNN)

A 2D CNN is a specialized deep learning model designed for analyzing two-dimensional image data. It processes pixel-based spatial hierarchies using convolutional filters to extract low-level and high-level features, making it ideal for medical image classification tasks like lung disease detection. Used in image classification, object detection, and medical imaging. A 2D Convolutional Neural Network (2D CNN) consists of multiple layers that process input images, extract meaningful patterns, and classify them into different categories.

Architecture of 2D CNN

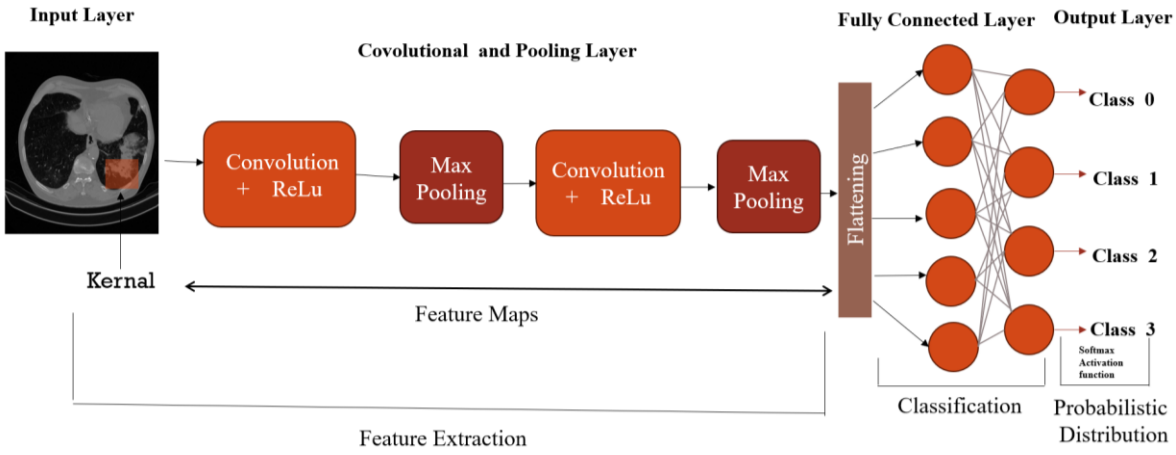


Figure 3 – 2D CNN Architecture

Layers :

1. Input Layer

The **input layer** receives raw CT scan images and prepares them for further processing. The input image is represented as a **matrix of pixel values** in the format **(Height × Width × Channels)**.

- **Grayscale images** have a single channel (e.g., 256 × 256 × 1).
- **RGB images** have three channels (e.g., 256 × 256 × 3). Each pixel value represents intensity, where **0 is black, and 255 is white** for grayscale images.

2. Convolutional Layer

The **convolutional layer** is responsible for detecting patterns such as **edges, textures, and abnormalities** in lung CT scans. It uses **filters (kernels)** that slide over the input image and perform element-wise multiplications followed by summation, producing a **feature map**.The convolution operation extracts **spatial features** while maintaining the spatial relationships between pixels.

Mathematical Representation of Convolution :

$$Z(i,j) = \sum_m \sum_n X(i+m, j+n) \cdot W(m,n) + B \quad (1)$$

Where:

- $Z(i,j)$ = Output feature map
- $X(i+m, j+n)$ = Input image pixel values
- $W(m,n)$ = Filter (kernel) weights
- B = Bias term

bias ∈ {0.0}

3. Activation Function (ReLU)

After convolution, the output feature maps are passed through an activation function to introduce non-linearity, enabling the model to learn complex patterns. The most commonly used activation function is ReLU (Rectified Linear Unit)

$$f(x) = \max(0, x) \quad [2]$$

- If $x > 0$, it remains unchanged.
- If $x \leq 0$, it is replaced with 0.

Advantages of ReLU:

✓ Prevents the vanishing gradient problem

✓ Speeds up training

✓ Introduces non-linearity, making the network more expressive

4. Pooling Layer (Downsampling)

The pooling layer reduces the spatial size of the feature maps while preserving essential information. This process helps in reducing computational complexity and prevents overfitting.

Types of Pooling:

- **Max Pooling** (most commonly used)

$$P(i, j) = \max_{m,n} X(i + m, j + n) \quad [3]$$

  - Retains the **maximum value** in a selected region.
  - Helps extract **strongest features** from an image.

- **Average Pooling**

  - Computes the **average value** in a region.
  - Used when finer details are important.

Example: **A 224 × 224 image becomes 112 × 112** after applying a **2 × 2 max pooling operation.**

5. Fully Connected Layer (FC Layer or Dense Layer)

After several convolutional and pooling layers, the extracted 2D feature maps are converted into a 1D vector and passed through fully connected layers. These layers combine extracted features to make final predictions. Each neuron in a fully connected layer is connected to all neurons in the previous layer, allowing the network to learn complex representations.

Mathematical Representation of Fully Connected Layer

$$Z_{fc} = W^T X + B \qquad [4]$$

Where:

- W = Weights learned during training
- X = Flattened feature vector
- B = Bias term
- Z_{fc} = Final computed value before activation

6. Output Layer

The final layer of a CNN is responsible for **producing class probabilities** based on the extracted features. The activation function used depends on the type of classification:

1. Softmax Activation Function (For Multi-Class Classification)

Used when there are **more than two classes**, as in this lung disease classification model.

$$P(y_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \qquad [5]$$

- Outputs a **probability distribution** across multiple classes.
- The predicted class is the one with the **highest probability**.

.

$$L = -\sum_i y_i \log(\hat{y}_i) \qquad [6]$$

Where:

- y_i = True label (0 or 1)
- $\hat{y}_i$ = Predicted probability for class i

II 3D Convolutional Neural Network (3D CNN)

A 3D Convolutional Neural Network (3D CNN) is an advanced deep learning model designed to process three-dimensional volumetric data, such as CT scans, MRI scans, and video frames. Unlike 2D CNNs, which analyze single slices or frames, 3D CNNs consider spatial and depth-wise features, making them highly effective for medical imaging tasks where adjacent slices contain critical contextual information.

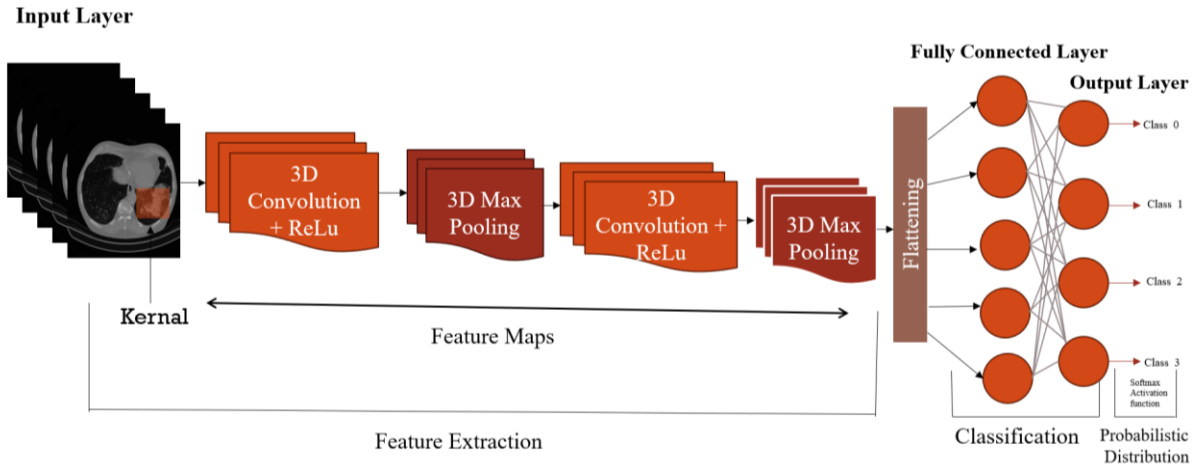


Figure 4 – 3D CNN Architecture

1. Input Layer

- The input layer receives a 3D CT scan volume, represented as **a (Depth × Height × Width × Channels) matrix.**
- For grayscale CT scans, the shape is typically (32 × 128 × 128 × 1), where:

1. **Depth (D)** = Number of slices (e.g., 32 slices per scan)
2. **Height (H) & Width (W)** = Image dimensions (e.g., 128 × 128 pixels)
3. **Channels (C)** = 1 for grayscale, 3 for RGB

Mathematically, the input is represented as:

$$X_{input} \in \mathbb{R}^{D \times H \times W \times C} \qquad [7]$$

2. 3D Convolutional Layer

The **3D convolutional layer** applies **3D filters (kernels)** to detect spatial and depth-wise features in the CT scan. This allows the model to recognize lung

structures across multiple slices. This operation extracts **3D features**, allowing the model to learn structural differences between lung disease types.

Mathematical Representation of 3D Convolution

$$Z(i,j,k) = \sum_{m}\sum_{n}\sum_{p} X(i+m, j+n, k+p) \cdot W(m,n,p) + B \qquad [8]$$

Where:

- $Z(i,j,k)$ = Output feature map
- $X(i+m, j+n, k+p)$ = Input image voxel
- $W(m,n,p)$ = 3D filter weights
- B = Bias term

3. Activation Function (ReLU)

- After convolution, a ReLU activation function is applied to introduce non-linearity. ReLU ensures that negative values are replaced with **zero**, preventing vanishing gradients and improving learning efficiency

$$f(x) = \max(0, x) \qquad [9]$$

4. 3D Pooling Layer (Downsampling)

Pooling layers reduce spatial and depth dimensions, making the model computationally efficient while preserving important information.

Types of 3D Pooling:

4.1 .3D Max Pooling

- Selects the maximum value in a defined 3D region.
- Retains the strongest feature activations.

$$P(i,j,k) = \max_{m,n,p} X(i+m, j+n, k+p) \qquad [10]$$

4.2. 3D Average Pooling:

- Computes the average of values within the pooling window.

Example: If the input volume is (32 × 128 × 128), after a (2 × 2 × 2) max pooling operation, the output reduces to (16 × 64 × 64).

5. Flatten Layer

After several convolutional and pooling layers, the 3D feature maps are converted into a 1D vector, preparing it for classification in fully connected layers.

$$X_{flattened} = Flatten(PooledFeatureMaps) \quad [11]$$

6. Fully Connected (Dense) Layer

This layer processes the flattened feature vector and applies weighted connections to classify the image.

$$Z_{fc} = W^T X + B \quad [12]$$

Where:

- W = Weights learned during training
- X = Flattened feature vector
- B = Bias term
- Z_{fc} = Final computed value before activation

7. Output Layer (Softmax Activation)

- The final layer uses Softmax Activation for multi-class classification. Assigns a probability score to each class.
- The class with the highest probability is the predicted output.

$$P(y_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad [13]$$

III 3D Vox ResNet

- 3D VoxResNet (3D Voxel Residual Network) is a deep learning architecture that extends ResNet using 3D convolutional layers to process volumetric data, enabling efficient spatial feature extraction.
- Uses 3D convolutions + residual connections to efficiently extract spatial features while preventing vanishing gradients, enabling deep network training.
- It preserves depth-wise information through residual connections, making it highly effective for applications like medical imaging and 3D object recognition.

Figure 5 – 3D VoxResNet Architecture

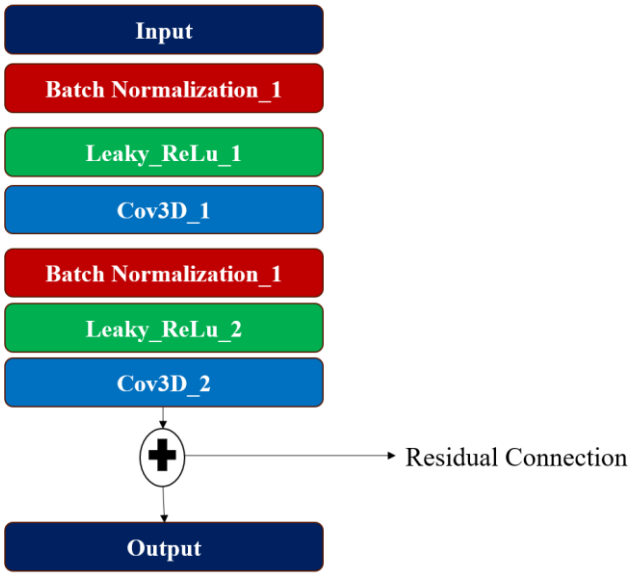


Figure 6 – Residual VoxRes Modules

1. Input Layer

The network begins with an input layer that receives the 3D CT scan as a volume of size (Depth × Height × Width × Channels). For grayscale scans, this is typically (32 × 128 × 128 × 1), representing 32 slices of 128x128-pixel images. The input layer simply passes raw voxel intensities to the next stage.

• **Mathematically, the input is:**

$$\mathbf{X} \in \mathbb{R}^{D \times H \times W \times C} \quad [14]$$

2. 3D Convolutional Layer

This layer performs a 3D convolution operation to extract early volumetric features from the CT scan. It uses 3D filters that slide across the depth, height, and width of the volume to detect local patterns like edges, corners, and textures across slices. This operation captures important structural details relevant for disease classification.

Mathematical expression:

$$Z(i,j,k) = \sum_m \sum_n \sum_p X(i+m, j+n, k+p) \cdot W(m,n,p) + B \quad [15]$$

Where:

- $Z(i,j,k)$: Output voxel
- X: Input voxel values
- W: Weight kernel
- B: Bias term

3. Batch Normalization

This layer normalizes the outputs of the convolutional layer by subtracting the batch mean and dividing by the batch standard deviation. This stabilizes training, reduces internal covariate shift, and speeds up convergence. After normalization, the data is scaled and shifted using learnable parameters γ and β

Formula

$$BN(x) = \gamma \left(\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} \right) + \beta \quad [16]$$

Where μ is the mean, σ^2 the variance, and γ, β are learnable parameters.

4.Leaky ReLU Activation

The Leaky ReLU introduces non-linearity to help the network learn complex patterns. Unlike standard ReLU, Leaky ReLU allows a small gradient for negative input values, which avoids the issue of dying neurons that stop updating.

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \quad [17]$$

Where α is a small constant, usually 0.01.

5.VoxRes Module (Residual Block)

The VoxRes module is the core unit of the VoxResNet architecture. It contains two 3D convolutional layers, each followed by batch normalization and Leaky ReLU activation. The key feature is the *residual connection*—the input is directly added to the output of the second activation. This helps mitigate the vanishing gradient problem and improves gradient flow. The module allows the network to be much deeper while still learning efficiently.

Structure:
• BN → Leaky ReLU → Conv3D → BN → Leaky ReLU → Conv3D
• Add residual connection from input to output

Formula :

$$y = \mathcal{F}(x) + x$$

Where $\mathcal{F}(x)$ is the transformation through the block. [18]

6.Global Average Pooling (GAP)

Converts the final 3D feature map into a 1D vector by averaging each channel. This reduces the number of parameters and minimizes overfitting.

Formula:

$$y^{(c)} = \frac{1}{DHW} \sum_{d,h,w} x^{(c)}_{d,h,w} \quad [19]$$

7.Fully Connected (Dense) Layer

The dense layer processes the feature vector from GAP and maps it to output class scores (logits). It acts as a classifier by learning which features are most predictive of each disease class.

$$y = Wx + b \quad [20]$$

8.Dropout Layer

Dropout randomly deactivates a percentage of neurons during training. This prevents over-reliance on specific neurons and encourages the network to learn

more general patterns. It's a regularization technique to reduce overfitting. A typical rate is 0.3 to 0.5.

9.Output Layer (Softmax Activation)

The final layer applies the softmax function to convert logits into class probabilities. The class with the highest probability becomes the model's prediction.

Formula :

$$\hat{y}_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad [21]$$

IV Explainable AI (XAI)

Explainable Artificial Intelligence (XAI) refers to a set of methods and frameworks designed to make the decisions of complex AI models particularly deep neural networks—more transparent and interpretable to human users.. While AI systems can outperform humans in many tasks, their internal mechanisms are often opaque, especially in the case of deep neural networks. These models are frequently referred to as "black boxes" because they make accurate predictions but offer little to no insight into how those predictions were derived.

In sensitive and high-stakes domains such as **healthcare**, **finance**, and **autonomous systems**, the cost of incorrect or unexplainable decisions can be extremely high. Therefore, XAI provides tools and techniques that allow human users including doctors, researchers, and regulators to understand, validate, and even challenge the decisions made by AI systems.

Key XAI Techniques :

1. Saliency Maps

These maps highlight the most influential input regions by computing the gradient of the output class with respect to input pixels or voxels. In 3D medical imaging, saliency maps extend across volume slices and reveal how much each voxel contributes to a prediction.

2. Grad-CAM (Gradient-weighted Class Activation Mapping)

Grad-CAM is one of the most widely adopted visualization techniques in XAI. It works by calculating the gradients of a class score with respect to the feature maps of the last convolutional layer in a neural network.

In 3D CNNs, Grad-CAM is extended to volumetric data allowing for **3D heatmaps** that show where the model is focusing inside a CT scan volume.

The final output is a **3D attention map**, which can be visualized as **colored overlays** on top of the original CT slices. This helps radiologists see where the model was "looking" when it made a decision.

Mathematical Representation:

For class c, Grad-CAM computes:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \sum_l \frac{\partial y^c}{\partial A_{ijk}^k}$$

$$L_{\text{Grad-CAM}}^c = \text{ReLU}\left(\sum_k \alpha_k^c A^k \right)$$

Where:

- y^c is the score for class c,
- A^k is the activation map from the k-th filter,
- α_k^c is the importance weight of feature map A^k,
- Z is the number of voxels in each feature map.

[22]

3.LIME (Local Interpretable Model-agnostic Explanations)

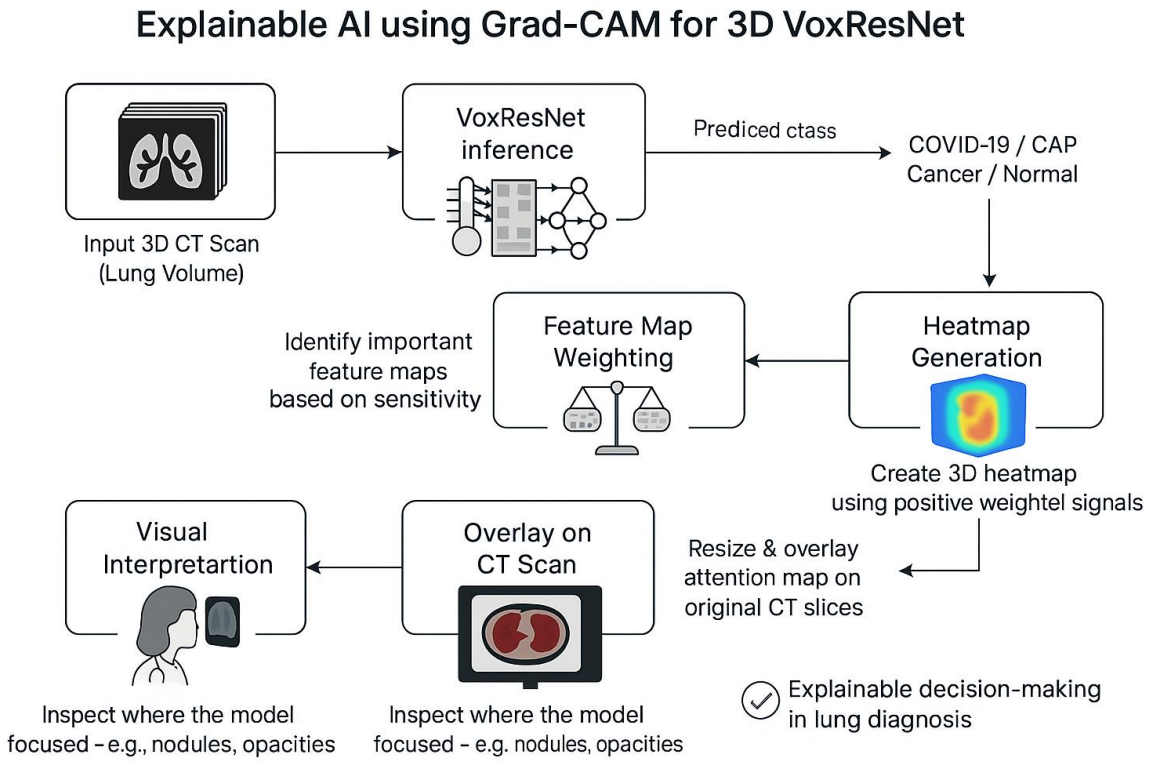
LIME generates simple interpretable models (e.g., linear regressions) around a specific prediction. It perturbs the input and observes the effect on the output, identifying which parts of the input are locally important.

While powerful, LIME's application in 3D volumes can be computationally expensive and often requires region simplification or dimensionality reduction.

4.SHAP (SHapley Additive exPlanations)

Based on game theory, SHAP assigns a contribution score to every input feature. It's theoretically grounded and guarantees consistency. In the context of medical imaging, SHAP can be used to evaluate the importance of voxel clusters, slices, or extracted features.

In this study, the interpretability of the 3D VoxResNet model is enhanced using **3D Grad-CAM**, a visualization technique that highlights the spatial regions most responsible for the model's predictions. This is especially crucial in medical imaging, where understanding *why* a model makes a decision is just as important as the decision itself.



Steps Involved in Applying XAI to the 3D VoxResNet Model:

Step 1: Input Volume Inference

The process begins by feeding a **3D CT scan** volume into the trained **VoxResNet** model. This volume captures the internal lung structure across multiple 2D slices stacked along the depth axis. The model processes this spatial information to generate class predictions, such as **COVID-19, CAP, Cancer, or Normal**.

The trained model maps the input volume to a set of predicted class scores:

$$\hat{\mathbf{y}} = f(V;\theta) \in \mathbb{R}^C$$

Where:

- $\hat{\mathbf{y}}$: Output vector with class probabilities
- θ: Learned model parameters
- C: Total number of disease classes

[23]

Step 2: Gradient Computation

After the model makes a prediction, we need to find out which parts of the volume were most influential in that decision. To do this, we compute the gradient of the predicted class score with respect to the final convolutional feature maps. This shows how sensitive the output is to changes in each spatial location in the feature maps.

Mathematically, we compute:

$$\frac{\partial \hat{y}_{\hat{c}}}{\partial \mathbf{A}^k}$$

Where:

- $\hat{y}_{\hat{c}}$: Predicted score for the top class $\hat{c}$
- $\mathbf{A}^k \in \mathbb{R}^{D' \times H' \times W'}$: Feature map of channel k
- $\frac{\partial \hat{y}_{\hat{c}}}{\partial \mathbf{A}^k}$: Gradient indicating how much the prediction changes with respect to feature map k

[24]

Step 3: Weighting of Feature Maps

Next, we need to quantify the importance of each feature map in the final decision. We do this by averaging the computed gradients over the entire spatial region, giving us a single scalar weight for each feature map.

Mathematically, we calculate:

$$\alpha_k = \frac{1}{D' \cdot H' \cdot W'} \sum_{d=1}^{D'} \sum_{h=1}^{H'} \sum_{w=1}^{W'} \frac{\partial \hat{y}_{\hat{c}}}{\partial \mathbf{A}^k_{d,h,w}}$$

Where:

- α_k: Importance weight of feature map k
- D', H', W': Dimensions of the feature maps
- $\frac{\partial \hat{y}_{\hat{c}}}{\partial \mathbf{A}^k_{d,h,w}}$: Gradient at location (d, h, w)

[25]

Step 4: Heatmap Generation

Now that we know which feature maps are important, we combine them to form a 3D class activation map. This is done by computing a weighted sum of all feature maps and applying a ReLU activation to keep only the positive influence regions.

Mathematically:

$$\mathbf{L}_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_{k=1}^{K} \alpha_k \cdot \mathbf{A}^k\right)$$

Where:

- $\mathbf{L}_{\text{Grad-CAM}}$: Raw attention map for the predicted class
- $\text{ReLU}(\cdot)$: Keeps only positive contributions (helps in localizing important regions)
- $\alpha_k \cdot \mathbf{A}^k$: Weighted feature map

[26]

Step 5: Overlay on CT Slices

The raw attention map is usually smaller than the original input. So we resize it back to the input volume's resolution, and overlay it slice-by-slice on the original CT scan, creating a visual explanation of what the model focused on.This final output allows us to **visually inspect which parts of the lung were important** in making the classification.

Mathematically:

$$\mathbf{L}_{\text{Resized}} = \text{Resize}(\mathbf{L}_{\text{Grad-CAM}}, (D, H, W))$$

Where:

- $\mathbf{L}_{\text{Resized}}$: Final heatmap resized to match the input scan
- Resize: Usually done using trilinear interpolation

[27]

Step 6: Visual Interpretation

The overlaid heatmaps serve as visual evidence for model reasoning. Radiologists and researchers can interpret these to understand and verify whether the model focused on medically relevant areas (e.g., nodules, lesions, opacities).

4.5 .Testing and Evaluation

The testing phase is a crucial step in evaluating the performance and generalization ability of the trained 2D and 3D CNN models. After training, the models are tested on unseen CT scan images to assess their classification accuracy, robustness, and reliability in diagnosing lung diseases. The dataset is split into 80% for training and 20% for testing, ensuring a fair evaluation of the model's ability to handle new data.

4.5.1. Testing Procedure

The testing phase involves the following steps:

- The trained **2D CNN and 3D CNN** (Stage 1 model) is used to classify CT scans into four broad categories: Normal, Pneumonia (CAP), COVID-19, and Cancer.
- If an image is classified as Cancer, it is passed to the Stage 2 model (another 2D CNN and 3D CNN) to further classify the cancer into Adenocarcinoma, Squamous Cell Carcinoma, or Large Cell Carcinoma.
- After the initial experiments with 2D and 3D CNN models, the dataset was expanded to include more patient cases for better training and generalization. With this larger dataset, a more powerful model **3D VoxResNet** was employed to better capture the complex spatial patterns in volumetric CT scans. This model delivered improved accuracy, particularly in identifying different lung cancer subtypes.
- In the final phase, **Explainable AI (XAI)** was applied using **Grad-CAM** on the **3D VoxResNet** model. This technique helped visualize the important regions in the CT scans that influenced the model's predictions, making the decision-making process more interpretable and trustworthy for clinical use.
- Model predictions were compared with the ground truth labels, and standard performance metrics such as **accuracy**, **precision**, **recall**, and **F1-score** were calculated to evaluate diagnostic effectiveness.

4.5.2. Performance Metrics

The following evaluation metrics are used to assess the model's accuracy and effectiveness:

a) Accuracy

Accuracy is one of the most important metrics for evaluating the performance of a machine learning model. It represents the proportion of correctly classified instances out of the total predictions made. Higher accuracy indicates that the model performs well in real-world conditions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where:

- TP (True Positive): Correctly predicted disease cases
- TN (True Negative): Correctly predicted normal cases
- FP (False Positive): Normal cases misclassified as diseased
- FN (False Negative): Diseased cases misclassified as normal

[28]

b) Precision

Precision determines how many of the predicted positive cases are actually positive.

$$Precision = \frac{TP}{TP + FP} \quad [29]$$

c) Recall (Sensitivity)

Recall indicates how well the model identifies actual positive cases.

$$Recall = \frac{TP}{TP + FN} \quad [30]$$

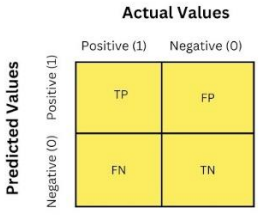
d) F1-Score

F1-score is the harmonic mean of precision and recall, balancing false positives and false negatives.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad [31]$$

e) Confusion Matrix

A confusion matrix provides a detailed breakdown of the model's predictions, displaying the number of correctly and incorrectly classified images for each class.



[32]

CHAPTER 5

EXPERIMENTAL RESULT

5.1 Accuracy Comparison of Different Models in Stage 1 and Stage 2

Model	Stage 1	Stage 2
	Accuracy	Accuracy
2D CNN	70%	60%
3D CNN	75%	61%
3D VoxResNet	79%	82%

Table 1

5.2 Prediction Result

The 2D and 3D CNN models classify lung diseases based on CT scan images. The Stage 1 (2D CNN) model predicts whether a scan belongs to Normal, CAP (Pneumonia), COVID-19, or Cancer. If classified as Cancer, the Stage 2 model (2D or 3D CNN) further identifies the cancer subtype: Adenocarcinoma, Squamous Cell Carcinoma, or Large Cell Carcinoma. The predictions are based on learned patterns from the dataset, with 2D CNN analyzing individual slices and 3D CNN utilizing volumetric data for better accuracy. Overall, the 3D CNN performed significantly better than the 2D CNN in both classification stages.

To further enhance performance, a 3D VoxResNet architecture is employed. This model, specifically designed for 3D medical imaging, delivered higher classification accuracy than the standard 3D CNN, particularly in distinguishing between cancer subtypes in Stage 2. Overall, the 3D VoxResNet model demonstrated superior performance compared to the normal 3D CNN.

Additionally, **Explainable AI (XAI)** techniques are applied using **Grad-CAM** on the **3D VoxResNet.** This allowed visualization of the most influential regions in the CT volume that contributed to the model's decisions, thereby increasing the transparency and interpretability of the predictions.

5.2.1 Final Prediction using the trained 2D CNN model :

This result illustrates the output of the Stage 1 and Stage 2 predictions made by the 2D CNN model. The model identifies the scan as 'Cancer' in Stage 1 and then classifies it further into 'Large Cell Carcinoma' in Stage 2, based on the predicted class probabilities.

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
Stage 1 Probabilities: {'CAP': 2.3898778e-05, 'COVID19': 0.0014146034, 'Cancer': 0.9975781, 'Normal': 0.0009834861}
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
Stage 2 Probabilities: {'Adenocarcinoma': 0.010909649, 'LargeCellCarcinoma': 0.989082, 'SquamousCellCarcinoma': 8.4047015e-06}
Predicted: Cancer -> LargeCellCarcinoma
```

Figure 8 – 2D CNN Prediction

5.2.2 Final Prediction using the trained 3D CNN model :

The 3D CNN model performs classification on the volumetric CT scan data. In the first example, it predicts the case as 'Normal.' In another case, it correctly identifies 'Cancer' and classifies it further into the subtype 'Squamous Cell Carcinoma.'

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Stage 1 Prediction: Normal
Final Prediction: Normal
```

Figure 9 – Stage 1 of 3D CNN

```
WARNING:tensorflow:5 out of the last 5 calls to
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 483ms/step
WARNING:tensorflow:6 out of the last 6 calls to
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 572ms/step
Predicted: Cancer -> SquamousCellCarcinoma
```

Figure 10– Stage 2 of 3D CNN

5.2.3 Final Prediction using the trained 3D VoxResNet model :

This output shows the prediction capability of the 3D VoxResNet model. It successfully classifies a case as 'COVID-19' in Stage 1 and another case as 'Cancer,' with a follow-up Stage 2 classification into 'Large Cell Carcinoma.' The model outputs include the probabilities for each class, reflecting its confidence in the predictions.

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 649ms/step
Stage 1 Probabilities: {'CAP': 0.017743241, 'COVID19': 0.96170145, 'Cancer': 0.0012545408, 'Normal': 0.01930086}
Predicted: COVID19
```

Figure 11– Stage 1 of 3D VoxResNet

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 610ms/step
Stage 1 Probabilities: {'CAP': 0.3920575, 'COVID19': 0.07242891, 'Cancer': 0.5078025, 'Normal': 0.027711077}
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 532ms/step
Stage 2 Probabilities: {'Adenocarcinoma': 0.0003420702, 'Large Cell Carcinoma': 0.63360083, 'Squamous Cell Carcinoma': 0.36605707}
Predicted: Cancer -> Large Cell Carcinoma
```

Figure 12– Stage 2 of 3D VoxResNet

5.2.4 Comparison of Model Performance for Stage 1

Model	Accuracy	Precision	Recall	F1- Score
2D CNN	70%	0.70	0.70	0.68
3D CNN	75%	0.62	0.75	0.67
3D VoxResNet	**79%**	**0.81**	**0.79**	**0.79**

Table 2

5.2.5 Comparison of Model Performance for Stage 2

Model	Accuracy	Precision	Recall	F1- Score
2D CNN	60%	0.61	0.61	0.61
3D CNN	61%	0.61	0.61	0.61
3D VoxResNet	**80%**	**0.81**	**0.81**	**0.80**

Table 3

5.2.6 Accuracy Chart for Stage 1

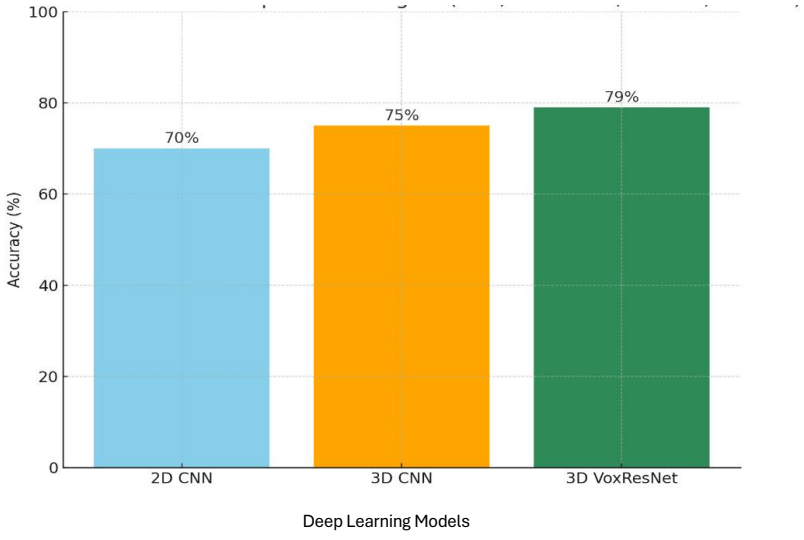


Figure 13– Accuracy Chart Stage1

5.2.7 Accuracy Chart for Stage 2

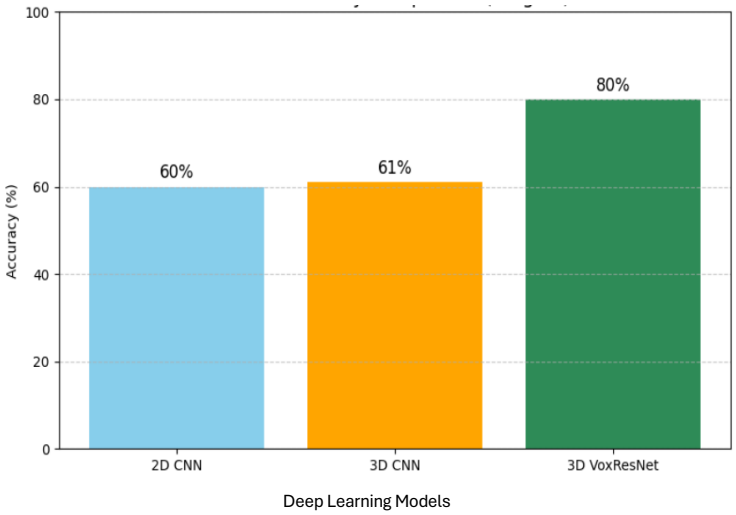


Figure 14– Accuracy Chart Stage1

5.2.8 Classification Report Chart for Stage 1

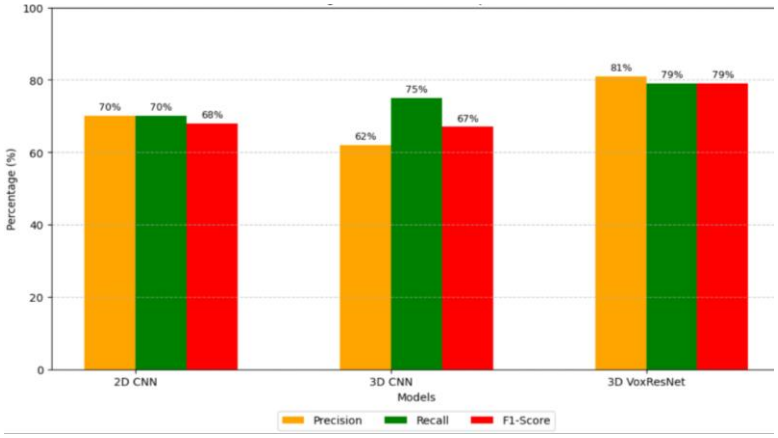


Figure 15– Classification Report Chart Stage1

5.2.9 Classification Report Chart for Stage 2

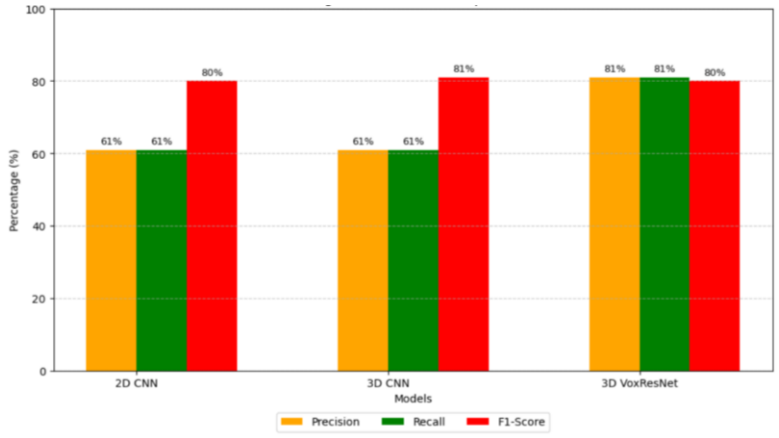


Figure 16– Classification Report Chart Stage2

5.2.10 Grad-CAM Visualization Using 3D VoxResNet for Stage 1

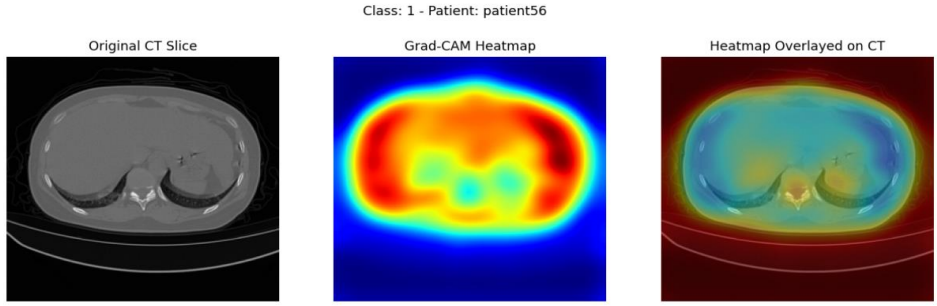


Figure 13 – Class 0 (CAP)

Figure 17 – Class 1 (COVID 19)

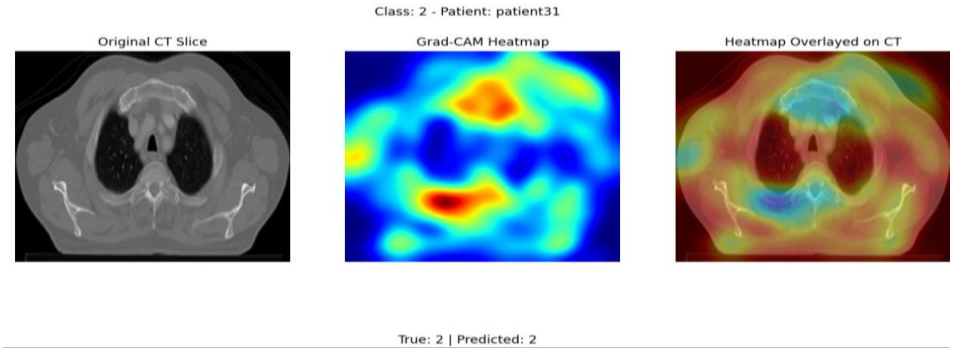


Figure 18– Class 2 (Cancer)

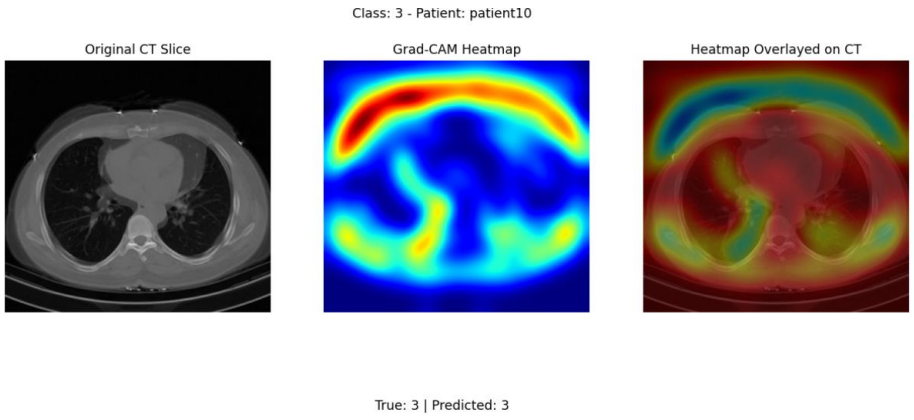


Figure 19– Class 3 (Normal)

5.2.11 Grad-CAM Visualization Using 3D VoxResNet for Stage 2

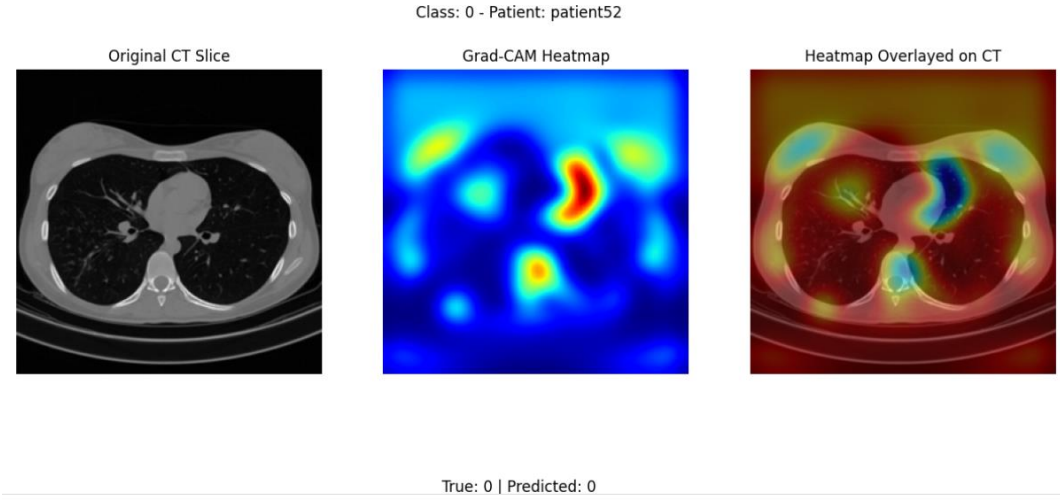


Figure 20– Class 0(Adenocarcinoma)

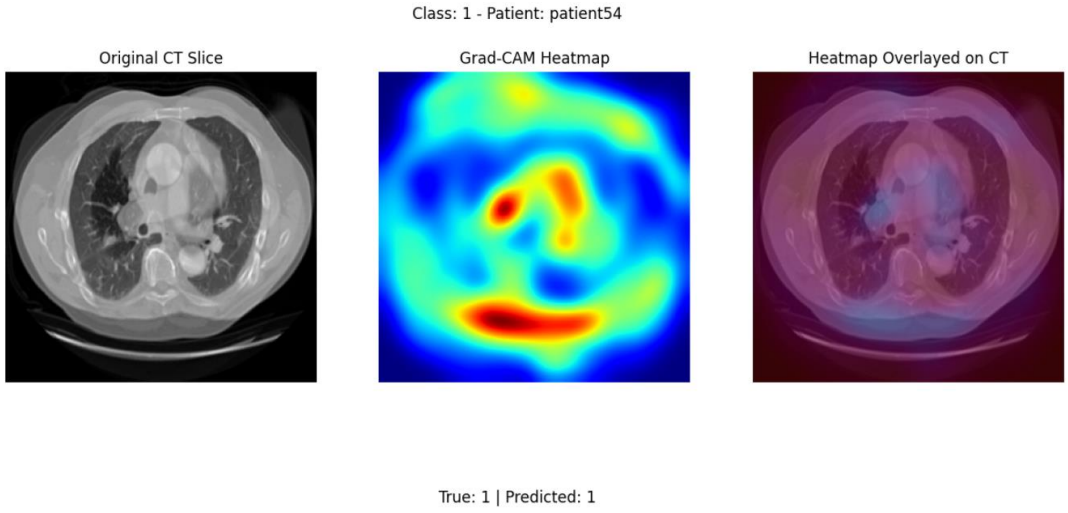


Figure 21– Class 1 (Large Cell Carcinoma)

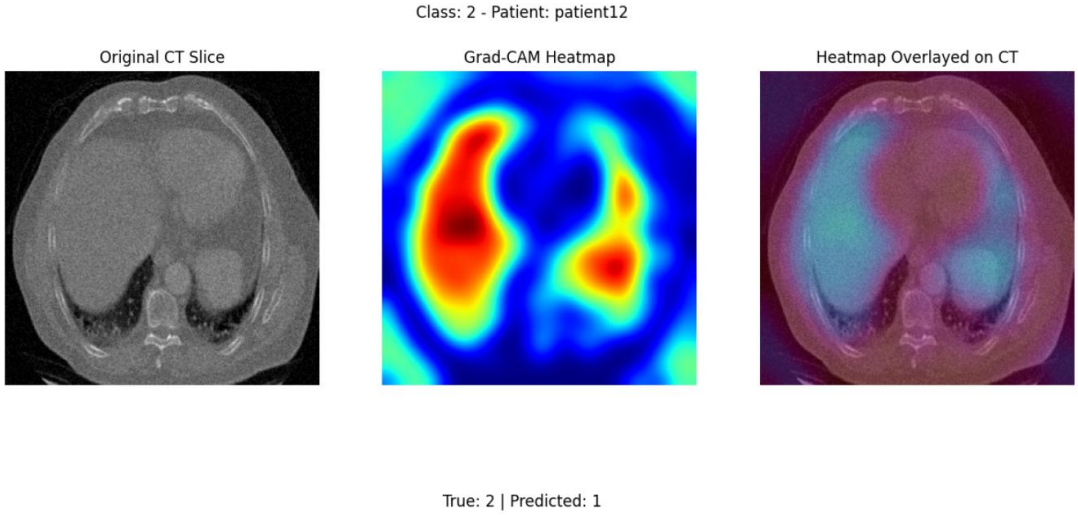


Figure 22– Class 2 (Squamous Cell Carcinoma)

CHAPTER 6

CONCLUSION

This study successfully implements a two-stage lung disease classification system using 2D and 3D CNN models for analyzing CT scan images. The Stage 1 model efficiently classifies scans into Normal, CAP (Pneumonia), COVID-19, and Cancer, while the Stage 2 model further distinguishes cancer cases into Adenocarcinoma, Squamous Cell Carcinoma, and Large Cell Carcinoma. The integration of 2D CNN allows for effective single-slice image analysis, whereas the 3D CNN enhances classification accuracy by capturing volumetric information across multiple slices.

To further boost diagnostic performance, the 3D VoxResNet architecture was employed, offering superior results due to its ability to model complex spatial features in volumetric medical data. Additionally, Explainable AI (XAI) techniques were implemented using Grad-CAM to highlight the key regions within CT scans that influenced the model's predictions, thereby enhancing model transparency and clinical trust.

Extensive training, testing, and evaluation demonstrate that deep learning can effectively assist in early lung disease detection, reducing diagnostic errors and aiding medical professionals in decision-making. Performance metrics, including accuracy, precision, recall, and F1-score, validate the robustness of the models, with 3D VoxResNet outperforming baseline CNNs in cancer subtype classification.

This research highlights the potential of deep learning in medical imaging and its role in improving automated diagnosis. Future work could involve larger datasets, hybrid models, transfer learning, and real-time clinical validation to further enhance accuracy and reliability. The findings of this project contribute to the advancement of diagnostic tools, making lung disease detection more efficient, interpretable, and accessible in healthcare applications.

CHAPTER 7

REFERENCES

1. Tas, M. O., & Yavuz, H**. S.** (2024). Enhancing Lung Cancer Survival Prediction: 3D CNN Analysis of CT Images Using Novel GTV1-SliceNum Feature and PEN-BCE Loss Function. *Diagnostics*, 14(12), 1309 Sneha Balannolla (2021) – Detecting and Classification of Lung Carcinoma Using CT Scans.
2. Sneha Balanagolla, **S.** (2022). Detection and Classification of Lung Carcinoma Using CT Scans. *Journal of Physics: Conference Series*, 2286(1), 012011.
3. Ahmed, J., Vesal, S., Durlak, F., Kaergel, R., Ravikumar, N., Remy-Jardin, M., & Maier, A. (2020). COPD Classification in CT Images Using a 3D Convolutional Neural Network. In *Bildverarbeitung für die Medizin 2020* (pp. 45–50). Springer Vieweg, Wiesbaden.
4. Chen, H., Dou, Q., Yu, L., Qin, J., & Heng, P. A. (2018). VoxResNet: Deep voxelwise residual networks for brain segmentation from 3D MR images. *NeuroImage*, 170, 446–455
5. Talaat, F. M., Gamel, S. A., El-Balka, R. M., Shehata, M., & ZainEldin, H. (2024). Grad-CAM Enabled Breast Cancer Classification with a 3D Inception-ResNet V2: Empowering Radiologists with Explainable Insights. *Cancers*, 16(21), 3668. Classification in CT Scans. IEEE Transactions on Medical Imaging.
6. Polat, H., & Danaei Mehr, H. (2019). Classification of Pulmonary CT Images by Using Hybrid 3D-Deep Convolutional Neural Network Architecture. *Applied Sciences*, 9(5), 940. Liu, X., Hou, F., Qin, H., & Hao, A. (2018) – Lung Cancer Detection Using Convolutional Neural Networks. Springer Lecture Notes in Computer Science.
7. Chen, H., Dou, Q., Yu, L., Qin, J., & Heng, P. A. (2018). VoxResNet: Deep voxelwise residual networks for brain segmentation from 3D MR images. *NeuroImage*, 170, 446–455.

APPENDIX

I – Source Code :

Google Colab- lungdiseaseclassification.ipynb

```
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.utils.class_weight import compute_class_weight
import matplotlib.pyplot as plt

# CLEAR SESSION TO AVOID GRAPH EXECUTION ERRORS
tf.keras.backend.clear_session()

# Directories for both stages
stage1_train_dir = "/content/drive/MyDrive/splited_Dataset-stage1/train"
stage1_test_dir = "/content/drive/MyDrive/splited_Dataset-stage1/test"

stage2_train_dir = "/content/drive/MyDrive/splited_Dataset-stage2/train"
stage2_test_dir = "/content/drive/MyDrive/splited_Dataset-stage2/test"

# Image size and batch size
IMG_SIZE = (128, 128)
BATCH_SIZE = 32

# ✅ Data Augmentation for Training Set
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=25,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

# ✅ Load Stage 1 Data
train_generator1 = train_datagen.flow_from_directory(
    stage1_train_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=True
)

test_generator1 = test_datagen.flow_from_directory(
    stage1_test_dir,
```

```
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)

# ✅ Build Stage 1 Model
def build_stage1_model():
    model = keras.Sequential([
        layers.Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(128, 128, 3)),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),

        layers.Conv2D(64, (3,3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),

        layers.Conv2D(128, (3,3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),

        layers.Conv2D(256, (3,3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),

        layers.Flatten(),
        layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
        layers.Dropout(0.3),
        layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
        layers.Dropout(0.2),
        layers.Dense(4, activation='softmax')  # 4 classes: Normal, CAP, COVID-19, Cancer
    ])

    model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

stage1_model = build_stage1_model()
stage1_model.summary()

# ✅ Callbacks
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
lr_schedule = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', patience=3, factor=0.5,
verbose=1)

# ✅ Train Stage 1 Model
history1 = stage1_model.fit(
    train_generator1,
    validation_data=test_generator1,
    epochs=30,
    class_weight=class_weight_dict1,
    callbacks=[early_stopping, lr_schedule]
```

```
)

# ✅ Save Stage 1 Model
stage1_model.save("/content/drive/MyDrive/1model.h5")
print("✅ Stage 1 Model Saved!")
# ✅ Load Stage 2 Data
train_generator2 = train_datagen.flow_from_directory(
    stage2_train_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=True
)

test_generator2 = test_datagen.flow_from_directory(
    stage2_test_dir,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)

# ✅ Compute Class Weights for Stage 2
class_labels2 = list(train_generator2.class_indices.keys())
class_weights2 = compute_class_weight('balanced', classes=np.unique(train_generator2.classes),
y=train_generator2.classes)
class_weight_dict2 = {i: class_weights2[i] for i in range(len(class_labels2))}

print("✅ Stage 2 Class Weights:", class_weight_dict2)

# ✅ Build Stage 2 Model
def build_stage2_model():
    model = keras.Sequential([
        layers.Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(128, 128, 3)),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),
        layers.Conv2D(64, (3,3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),
        layers.Conv2D(128, (3,3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),
        layers.Conv2D(256, (3,3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D(2,2),
        layers.Flatten(),
        layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
        layers.Dropout(0.3),
        layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
        layers.Dropout(0.2),
        layers.Dense(3, activation='softmax')  # 3 classes: Adenocarcinoma, Large Cell, Squamous Cell
    ])
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
```

```
            metrics=['accuracy'])
    return model
stage2_model = build_stage2_model()
stage2_model.summary()
# ✅ Train Stage 2 Model
history2 = stage2_model.fit(
    train_generator2,
    validation_data=test_generator2,
    epochs=30,
    class_weight=class_weight_dict2,
    callbacks=[early_stopping, lr_schedule]
)
# ✅ Save Stage 2 Model
stage2_model.save("/content/drive/MyDrive/two_model.h5")
print("✅ Stage 2 Model Saved!")
```

Explainable AI – Grad Cam implementation for Stage1

```
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import cv2
import random
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
from scipy.ndimage import gaussian_filter
#  Register Custom Objects (Fix Unknown Layer 'Cast')
class Cast(tf.keras.layers.Layer):
    def call(self, inputs):
        return tf.cast(inputs, tf.float32)
# Constants
NUM_SLICES = 32
IMG_SIZE = (256, 256)
TEST_DIR_STAGE2 = "/kaggle/input/lung-stage2/test"
NUM_CLASSES_STAGE2 = 3
MODEL_PATH_STAGE2 = "/kaggle/input/stage2h5/voxresnet_stage2.h5"
#  Load Model with Custom Objects
try:
    with tf.keras.utils.custom_object_scope({'Cast': Cast}):
        loaded_model = load_model(MODEL_PATH_STAGE2, compile=False)
except ValueError:
    print("⚠️ Warning: Removing unknown layers and reloading model")
    with tf.keras.utils.custom_object_scope({'Cast': Cast}):
        loaded_model = load_model(MODEL_PATH_STAGE2, compile=False)
# ✅ Get Last Conv Layer Name
conv_layers = [layer.name for layer in loaded_model.layers if isinstance(layer,
tf.keras.layers.Conv3D)]
LAST_CONV_LAYER = conv_layers[-1]
print(f"✅ Last Conv Layer Found: {LAST_CONV_LAYER}")
# ✅ Load 3D Data Function
def load_3d_data(root_dir, num_classes):
```

```
    classes = sorted(os.listdir(root_dir))
    class_indices = {cls: i for i, cls in enumerate(classes)}
    X, Y, patient_names = [], [], []
    for disease in classes:
        disease_path = os.path.join(root_dir, disease)
        for patient in os.listdir(disease_path):
            patient_path = os.path.join(disease_path, patient)
            if os.path.isdir(patient_path):
                slices = sorted(os.listdir(patient_path))
                patient_volume = []
                for slice_file in slices[:NUM_SLICES]:
                    slice_path = os.path.join(patient_path, slice_file)
                    img = load_img(slice_path, color_mode='grayscale', target_size=IMG_SIZE)
                    img_array = img_to_array(img).astype(np.float32) / 255.0  # ✅ Normalize to float32
                    patient_volume.append(img_array)
                if len(patient_volume) == NUM_SLICES:
                    X.append(np.stack(patient_volume, axis=0).astype(np.float32))
                    Y.append(class_indices[disease])
                    patient_names.append(patient)
    return np.array(X, dtype=np.float32), to_categorical(np.array(Y), num_classes=num_classes),
patient_names
# ✅ Load test dataset
X_test, Y_test, patient_names = load_3d_data(TEST_DIR_STAGE2,
num_classes=NUM_CLASSES_STAGE2)
# ✅ Grad-CAM for 3D images
def compute_gradcam_3d(model, img_array, class_index, last_conv_layer_name):
    grad_model = tf.keras.models.Model(
        inputs=model.input,
        outputs=[model.get_layer(last_conv_layer_name).output, model.output]
    )
    img_tensor = tf.convert_to_tensor(np.expand_dims(img_array, axis=0), dtype=tf.float32)
    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_tensor)
        loss = predictions[:, class_index]
    grads = tape.gradient(loss, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2, 3))
    conv_outputs = conv_outputs.numpy()[0]
    pooled_grads = pooled_grads.numpy()
    # Apply gradients to feature maps
    for i in range(conv_outputs.shape[-1]):
        conv_outputs[:, :, :, i] *= pooled_grads[i]
    heatmap = np.mean(conv_outputs, axis=-1)
    heatmap = np.maximum(heatmap, 0)  # ReLU activation
    # Convert to float32 before applying Gaussian filter
    heatmap = heatmap.astype(np.float32)
    heatmap = gaussian_filter(heatmap, sigma=2)
    # Normalize to 0-1 scale (avoid division by zero)
    if np.max(heatmap) != 0:
        heatmap /= np.max(heatmap)
    return heatmap
# ✅ Overlay Heatmap with Mask
def overlay_heatmap_3d_with_mask(heatmap, original_volume, mask, alpha=0.4):
    num_slices = min(heatmap.shape[0], original_volume.shape[0])
    middle_slice = num_slices // 2  # Middle slice of the available range
```

```
    original_slice = original_volume[middle_slice, :, :, 0]
    # Resize heatmap and apply color map
    heatmap_resized = cv2.resize(heatmap[middle_slice], (256, 256))
    heatmap_color = cv2.applyColorMap(np.uint8(255 * heatmap_resized),
cv2.COLORMAP_PLASMA)
    # Resize mask
    mask_resized = cv2.resize(mask[middle_slice], (256, 256))
    mask_resized = np.uint8(mask_resized)
    # Convert original slice to RGB
    original_slice_rgb = cv2.cvtColor(cv2.convertScaleAbs(original_slice, alpha=255),
cv2.COLOR_GRAY2BGR)
    # Convert mask to RGB
    mask_resized_rgb = np.repeat(mask_resized[..., np.newaxis], 3, axis=-1)
    # Blend original image and heatmap
    blended_heatmap = cv2.addWeighted(original_slice_rgb, 1 - alpha, heatmap_color, alpha, 0)
    # Apply the mask
    blended_image = cv2.addWeighted(blended_heatmap, 1, mask_resized_rgb, 0.5, 0)
    return original_slice, heatmap_resized, blended_image
# ✅ Visualization
patients_per_class = 2  # Patients to visualize per class
selected_patients = []
for class_index in range(NUM_CLASSES_STAGE2):  # ✅ Fixed: Use Stage 2 classes
    class_patients = [i for i, sample_img in enumerate(X_test) if np.argmax(Y_test[i]) == class_index]
    random.shuffle(class_patients)
    selected_patients.extend(class_patients[:patients_per_class])
# ✅ Generate Grad-CAM Visualizations
for i in selected_patients:
    sample_img = X_test[i]
    true_class = np.argmax(Y_test[i])
    # Placeholder mask (replace with real lung mask)
    mask = np.ones_like(sample_img)
    # Compute Grad-CAM
    heatmap = compute_gradcam_3d(loaded_model, sample_img, true_class, LAST_CONV_LAYER)
    # Make prediction
    prediction = np.argmax(loaded_model.predict(np.expand_dims(sample_img, axis=0)))
    # Apply overlay
    original, heatmap_vis, overlayed = overlay_heatmap_3d_with_mask(heatmap, sample_img, mask)
    # Plot images
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))
    axes[0].imshow(original, cmap='gray')
    axes[0].set_title("Original CT Slice")
    axes[0].axis("off")
    axes[1].imshow(heatmap_vis, cmap='jet')
    axes[1].set_title("Grad-CAM Heatmap")
    axes[1].axis("off")
    axes[2].imshow(overlayed)
    axes[2].set_title("Heatmap Overlayed on CT")
    axes[2].axis("off")
    fig.text(0.5, -0.1, f"True: {true_class} | Predicted: {prediction}",
             ha="center", va="center", fontsize=12, color="black")
    plt.suptitle(f"Class: {true_class} - Patient: {patient_names[i]}")
    plt.show()
```