Project Process: Lung Cancer Detection Using Convolutional Neural Network (CNN)

Data Loading:

- Loaded the dataset consisting of lung CT scan or X-ray images from a local directory or online medical dataset (e.g., Kaggle).
- Used libraries like **OpenCV** and **TensorFlow/Keras** to read and process the image files.
- Converted images to grayscale or RGB depending on the model input requirements.
- Resized all images to a fixed dimension (e.g., 64×64 or 128×128 pixels) for uniformity and efficient training.

Initial Exploration:

- Visualized a sample of the images to inspect data quality, resolution, and visual differences between cancerous and non-cancerous cases.
- Checked for any class imbalance and data distribution across categories.
- Ensured proper labeling and format of image data before feeding into the CNN model.

Data Preprocessing:

- Normalized pixel values (scaled to the range [0,1]) to facilitate faster and more stable training.
- Applied image augmentation techniques (e.g., rotation, flipping, zooming) using ImageDataGenerator to enhance model generalization and prevent overfitting.
- Split the dataset into **training**, **validation**, and **test sets** for model development and evaluation.

Model Architecture – CNN Design:

- Constructed a simple Convolutional Neural Network using **Keras Sequential API**, consisting of:

    - Multiple **Conv2D** layers with ReLU activation for feature extraction.
    - **MaxPooling2D** layers for dimensionality reduction.
    - **Dropout** layers for regularization and preventing overfitting.
    - **Flatten** and **Dense** layers for classification.
    - Final **sigmoid** activation for binary output (cancerous or not).

Model Training:

- Compiled the model using:
    - **Binary cross-entropy** as the loss function.
    - **Adam** optimizer for efficient gradient updates.
    - Accuracy as the evaluation metric.
- Trained the model on the training dataset with validation data to monitor performance.
- Visualized training and validation **accuracy and loss curves** using Matplotlib.

Evaluation & Testing:

- Evaluated the trained model on the test dataset to assess generalization.
- Calculated metrics such as:
    - **Accuracy**
    - **Precision**
    - **Recall**
    - **F1-score**
    - **Confusion Matrix**
- Identified false positives and false negatives to understand model weaknesses.

Visualization & Interpretability:

- Plotted training history to observe convergence and detect overfitting.
- Used heatmaps or Grad-CAM (optional) to highlight image regions influencing model predictions.

Modeling and Analysis:

- Assessed model sensitivity to image resolution and preprocessing variations.
- Experimented with hyperparameters like learning rate, batch size, and number of epochs.
- Compared the CNN’s performance with a basic fully connected neural network as a baseline.