Minor Project Report

on

Project Title: - COVID-19 Detection System Using CNN Deep Learning

Course: Data Science Toolbox

Course Code: CAP367

Submitted by

Mohit Kumar
(12415064)
Khyati Soi
(12420081)

Submitted to

Mr.Manik Mehra
UID: 27377

Lovely Faculty of Technology & Sciences

School of Computer Applications

Lovely Professional University

Punjab

1. Title of the Project

COVID-19 Detection System Using CNN Deep Learning

2. Problem Statement:

The rise in respiratory diseases like COVID-19 and Viral Pneumonia has made it clear that there is a need for better and faster diagnostic tools, especially in areas with limited healthcare resources. Traditional methods, such as physical exams or having radiologists manually interpret chest X-rays, can take a lot of time, and may lead to inconsistent results. This project aims to provide an automated solution that can quickly and accurately identify whether a chest X-ray shows Normal, Viral Pneumonia, or COVID-19. The project involves multiple operations, including:

1. **Data Collection and Preprocessing**:
    - **Collection of labeled X-ray images** for three classes: Normal, Viral Pneumonia, and COVID-19.
    - **Image preprocessing** to enhance quality and ensure uniformity across samples. This includes resizing, normalization, and image augmentation techniques to expand the dataset and improve model generalization.
2. **Model Development and Training**:
    - **Designing and implementing a Convolutional Neural Network (CNN)** to capture features specific to each condition.
    - **Training the CNN model** on preprocessed images, optimizing it to distinguish subtle differences between healthy and diseased lungs.
3. **Model Evaluation and Validation**:
    - **Evaluation using performance metrics** like accuracy, precision, recall, and F1 score to ensure the model's reliability.
    - **Validation on a separate test set** to verify model robustness and prevent overfitting.
4. **Prediction and Interpretation**:
    - **Classification of new chest X-ray images** as Normal, Viral Pneumonia, or COVID-19.
    - Generation of confidence scores and, where possible, interpretability outputs (such as heatmaps) to highlight areas of the lung contributing to the model's decision.

3. Objectives of the proposed project.

The goal of this project is to create a reliable deep learning model to classify chest X-rays into three categories: Normal, Viral Pneumonia, and COVID-19. The model will be designed to be accurate and dependable for real-life use in healthcare. We will clean and prepare the data through image resizing and other techniques and add more variety to the images to help the model learn better and avoid mistakes. The model will be fine-tuned to reduce errors, especially between COVID-19 and Viral Pneumonia. To check how well it works, well use performance measures like accuracy and precision, and test it on new images to make sure it can handle realworld cases. The end goal is to create an easy-to-use tool that helps doctors quickly identify different conditions, speeding up diagnosis. We will also add features like heatmaps to show which parts of the X-ray influenced the model's decision, helping doctors understand the results better. Finally, the model will be built in a way that makes it easy to use in hospitals for fast or batch processing of X-rays.

4. System Requirements/Front end /back end/flowchart/Algorithm:

Hardware and Software Requirements: To run this project smoothly, a GPU (like NVIDIA RTX series) is recommended for training the model due to the large dataset and complex computations. You'll need at least 16GB of RAM to handle data processing, and 20GB of storage space for the dataset, trained models, and results. The project can run on Windows, macOS, or Linux, using Python 3.8 or higher

Frontend and Backend: The front end includes Jupyter Notebook for coding and testing, displaying model results and visualizations. Optionally, a simple user interface can be created with Flask to let users upload X-ray images and see predictions. The back end could use a lightweight database like SQLite if needed, and Flask for serving the model for real-time predictions through an Camera.

Project Workflow:

1. **Start** the process.
2. **Data Collection and Preprocessing**: Load, resize, normalize, and augment X-ray images.
3. **Model Development**: Design the CNN architecture, define layers, and set parameters.
4. **Model Evaluation**: Test on a validation set and calculate metrics like accuracy and precision.
5. **Prediction and Interpretation**: Classify new images and generate confidence scores.
6. **End** the workflow.

Step-by-Step Algorithm:

1. **Data Preprocessing**: Load images, resize to 224x224, normalize values, and augment data.
2. **Model Design**: Create a CNN model with layers for image classification.
3. **Model Training**: Compile the model with a loss function and optimizer, train on the dataset in batches, and adjust weights.
4. **Model Evaluation**: Test the model, calculate metrics like accuracy and recall.
5. **Prediction**: Preprocess new images, classify as Normal, Viral Pneumonia, or COVID-19, and provide confidence scores.
6. **Deployment:** Save the model and set up a camera to receive X-rays and return predictions.

Input and Output:

- **Input**: X-ray images in PNG format, resized to 224x224 pixels.
- **Output**: Predicted category for each X-ray (Normal, Viral Pneumonia, COVID-19), model metrics (accuracy, precision, recall, F1-score), and optionally, heatmaps showing important areas in the X-ray.

5. **Constraints** applied on the system.

The system has several fixed settings to ensure consistent and accurate performance.

Image Size: All X-ray images are resized to a standard size, such as 224x224 pixels, to ensure they fit the model's input layer requirements.

Image Format: PNG formats are accepted for input.

Class Labels: The model is trained exclusively to classify images into three classes: Normal, Viral Pneumonia, and COVID-19, and cannot detect other conditions.

Normalization: Pixel values are standardized between 0 and 1 to improve the model’s efficiency.

Model Architecture: The CNN model has a fixed structure with set layers, activation functions, and dropout rates to reduce overfitting.

Hyperparameters: Values like the learning rate, batch size, and number of epochs (e.g., 0.001 learning rate, batch size of 32, 20 epochs) are predefined for stable training.

Confidence Threshold: Predictions require a confidence level of at least 0.5 to be considered reliable; otherwise, they may be flagged as uncertain.

Dataset Constraints: The model’s accuracy may be limited by the dataset's diversity; if images lack variation, the model might not generalize well to other cases.

Hardware: Performance depends on the available hardware; a GPU is recommended for faster training, as CPU-based training may be slow.

6. Benefits of the proposed project.

The model enhances diagnostic efficiency by providing quick predictions for Normal, Viral Pneumonia, and COVID-19 cases from chest X-rays, greatly reducing diagnosis time and enabling faster clinical decisions, especially during emergencies or high-demand periods like pandemics. It improves diagnostic accuracy by detecting subtle patterns that might be missed in manual reviews, lowering misdiagnosis risks and ensuring consistent results. The model is also scalable and accessible, capable of handling large volumes of X-ray images, which is valuable for hospitals and clinics, particularly in areas with limited radiologist access. By automating initial screenings, it supports overburdened healthcare workers, allowing radiologists to focus on complex cases and reducing burnout. This solution is cost-effective, leveraging existing X-ray equipment and offering a more affordable diagnostic option compared to extensive lab tests. It ensures consistent, reliable results based on learned patterns, reducing the variability of human interpretation. In pandemics, it can be quickly deployed to identify symptoms of viral pneumonia, aiding in rapid isolation and treatment of infected patients. The model's framework can also be adapted for detecting other diseases, enabling future innovations in medical imaging. Additionally, interpretability features like heatmaps make the model's decision-making process understandable for medical professionals, providing a valuable educational tool that enhances diagnostic skills and knowledge in chest radiology.

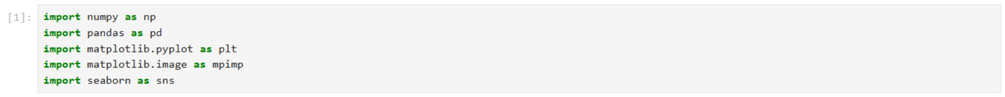
7. **Expected** outcomes.

The model aims to accurately classify chest X-ray images into three categories—Normal, Viral Pneumonia, and COVID-19—showcasing its ability to differentiate between these conditions based on subtle radiographic differences. It is expected to achieve high performance metrics, including accuracy, precision, recall, and F1-score, particularly for COVID-19 and Viral Pneumonia detection, which will validate the model's reliability and generalization. By automating the diagnosis process, the model will greatly reduce the time needed to get a preliminary result from chest X-rays, enabling faster clinical decision-making. If a user-friendly interface (such as a GUI or API) is implemented, healthcare providers can easily upload X-ray images and receive instant diagnostic predictions, making it accessible for non-technical users as well. The model will also produce interpretability outputs, like heatmaps, to highlight lung areas influencing the prediction, enhancing transparency and trust in its decisions. Designed to generalize well across different datasets, the model should perform robustly regardless of variations in image quality, patient demographics, or equipment, making it suitable for diverse healthcare settings. The end goal is to deliver a deployable model that can be integrated into clinical or mobile applications for real-time screening support. This tool can also improve pandemic preparedness by helping healthcare facilities quickly identify cases of viral pneumonia and COVID-19, allowing for timely action and effective resource allocation.

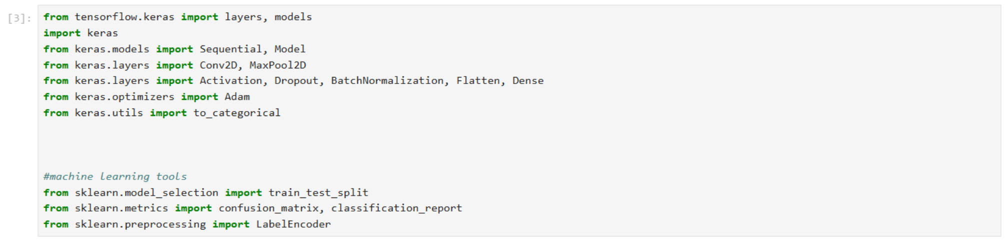
8. Future Scope

The model has potential for expansion to classify other respiratory diseases, such as tuberculosis, bacterial pneumonia, lung cancer, and COPD, increasing its value in diagnosing a broader range of conditions. Integrating it with Electronic Health Records (EHR) could add patient history and other medical data to enhance diagnostic accuracy. A mobile app could make the model accessible to healthcare providers in remote areas, allowing them to upload X-rays for instant analysis using a smartphone or tablet, which would help underserved communities. Future versions could incorporate other imaging types, like CT scans, or clinical data for a multi-modal diagnostic system, improving accuracy. The model could also be used as a real-time monitoring tool, alerting providers to worsening pulmonary conditions. Enhanced explainability features, such as saliency maps or SHAP values, would increase transparency and trust, especially in clinical use. Connecting the model with IoT-enabled X-ray machines could allow for instant predictions in busy settings, and continuous learning would enable the model to adapt to new diseases or data over time. Integration into telemedicine platforms would further extend its utility by supporting remote consultations, helping doctors diagnose and monitor patients with respiratory conditions remotely.

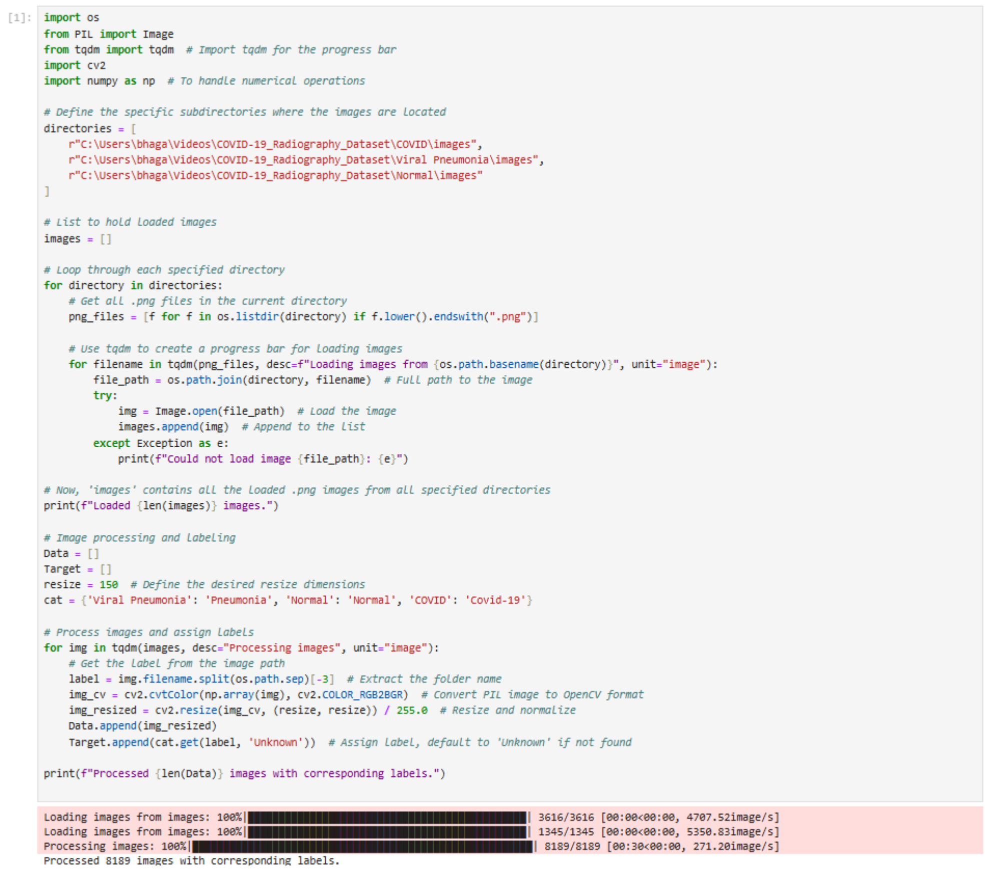
9. **Coding/layout** screenshots.



```
[1]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import matplotlib.image as mpimp
     import seaborn as sns
```



```
[3]: from tensorflow.keras import layers, models
     import keras
     from keras.models import Sequential, Model
     from keras.layers import Conv2D, MaxPool2D
     from keras.layers import Activation, Dropout, BatchNormalization, Flatten, Dense
     from keras.optimizers import Adam
     from keras.utils import to_categorical



     #machine learning tools
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import confusion_matrix, classification_report
     from sklearn.preprocessing import LabelEncoder
```



```
[1]: import os
     from PIL import Image
     from tqdm import tqdm  # Import tqdm for the progress bar
     import cv2
     import numpy as np  # To handle numerical operations

     # Define the specific subdirectories where the images are located
     directories = [
         r"C:\Users\bhaga\Videos\COVID-19_Radiography_Dataset\COVID\images",
         r"C:\Users\bhaga\Videos\COVID-19_Radiography_Dataset\Viral Pneumonia\images",
         r"C:\Users\bhaga\Videos\COVID-19_Radiography_Dataset\Normal\images"
     ]

     # List to hold loaded images
     images = []

     # Loop through each specified directory
     for directory in directories:
         # Get all .png files in the current directory
         png_files = [f for f in os.listdir(directory) if f.lower().endswith(".png")]

         # Use tqdm to create a progress bar for loading images
         for filename in tqdm(png_files, desc=f"Loading images from {os.path.basename(directory)}", unit="image"):
             file_path = os.path.join(directory, filename)  # Full path to the image
             try:
                 img = Image.open(file_path)  # Load the image
                 images.append(img)  # Append to the list
             except Exception as e:
                 print(f"Could not load image {file_path}: {e}")

     # Now, 'images' contains all the loaded .png images from all specified directories
     print(f"Loaded {len(images)} images.")

     # Image processing and labeling
     Data = []
     Target = []
     resize = 150  # Define the desired resize dimensions
     cat = {'Viral Pneumonia': 'Pneumonia', 'Normal': 'Normal', 'COVID': 'Covid-19'}

     # Process images and assign labels
     for img in tqdm(images, desc="Processing images", unit="image"):
         # Get the label from the image path
         label = img.filename.split(os.path.sep)[-3]  # Extract the folder name
         img_cv = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)  # Convert PIL image to OpenCV format
         img_resized = cv2.resize(img_cv, (resize, resize)) / 255.0  # Resize and normalize
         Data.append(img_resized)
         Target.append(cat.get(label, 'Unknown'))  # Assign label, default to 'Unknown' if not found

     print(f"Processed {len(Data)} images with corresponding labels.")
```

```
Loading images from images: 100%|██████████████████████████████████████████████| 3616/3616 [00:00<00:00, 4707.52image/s]
Loading images from images: 100%|██████████████████████████████████████████████| 1345/1345 [00:00<00:00, 5350.83image/s]
Processing images: 100%|█████████████████████████████████████████████████████████| 8189/8189 [00:30<00:00, 271.20image/s]
Processed 8189 images with corresponding labels.
```

```
[5]: print(len(Data))
     print(len(Target))
```

```
8189
8189
```

```
[7]: Target[1000:1005]
```

```
[7]: ['Covid-19', 'Covid-19', 'Covid-19', 'Covid-19', 'Covid-19']
```

```
[9]: from sklearn.preprocessing import LabelEncoder
     from tensorflow.keras.utils import to_categorical
     le=LabelEncoder()
     labels=le.fit_transform(Target)
     labels=to_categorical(labels)

     print(le.classes_)
     print(labels[0])
```

```
['Covid-19' 'Normal' 'Pneumonia']
[1. 0. 0.]
```

```
[11]: import numpy as np
      from sklearn.model_selection import train_test_split

      # First split: Split Data into train and test sets (80% train, 20% test)
      (x_train, x_test, y_train, y_test) = train_test_split(Data, labels, test_size=0.20, stratify=labels, random_state=42)

      # Second split: Split the training data into training and validation sets (80% train, 20% validation)
      (x_train, x_val, y_train, y_val) = train_test_split(x_train, y_train, test_size=0.20, stratify=y_train, random_state=42)

      # Convert lists to numpy arrays
      trainX = np.array(x_train)
      valX = np.array(x_val)
      testX = np.array(x_test)
      trainY = np.array(y_train)
      valY = np.array(y_val)
      testY = np.array(y_test)

      # Print the shapes of the datasets
      print("Training data shape:", trainX.shape)
      print("Validation data shape:", valX.shape)
      print("Testing data shape:", testX.shape)
      print("Training labels shape:", trainY.shape)
      print("Validation labels shape:", valY.shape)
      print("Testing labels shape:", testY.shape)
```

```
Training data shape: (5240, 150, 150, 3)
Validation data shape: (1311, 150, 150, 3)
Testing data shape: (1638, 150, 150, 3)
Training labels shape: (5240, 3)
Validation labels shape: (1311, 3)
Testing labels shape: (1638, 3)
```

```
[13]: from keras.models import Sequential
      from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
      import keras

      s = 150
      model = Sequential()

      # Corrected kernel_size spelling
      model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(s, s, 3), kernel_initializer="he_normal"))
      model.add(MaxPooling2D(pool_size=(2, 2)))  # Corrected MaxPool2D to MaxPooling2D
      model.add(Conv2D(filters=64, kernel_size=(3, 3), activation="relu"))
      model.add(MaxPooling2D(pool_size=(2, 2)))
      model.add(Conv2D(filters=128, kernel_size=(3, 3), activation="relu"))
      model.add(MaxPooling2D(pool_size=(2, 2), strides=(1, 1)))  # Corrected MaxPOOL2D to MaxPooling2D
      model.add(Dropout(0.25))

      model.add(Flatten())
      model.add(Dense(64, activation="relu"))
      model.add(Dropout(0.2))  # Corrected dropout to Dropout
      model.add(Dense(3, activation="softmax"))

      # Corrected model.compile spelling
      model.compile(optimizer=keras.optimizers.Adam(),
                    loss=keras.losses.categorical_crossentropy,
                    metrics=['accuracy'])

      # Training the model
      epochs = 25
      history = model.fit(trainX, trainY, epochs=epochs, batch_size=40, verbose=1, validation_data=(valX, valY))
```

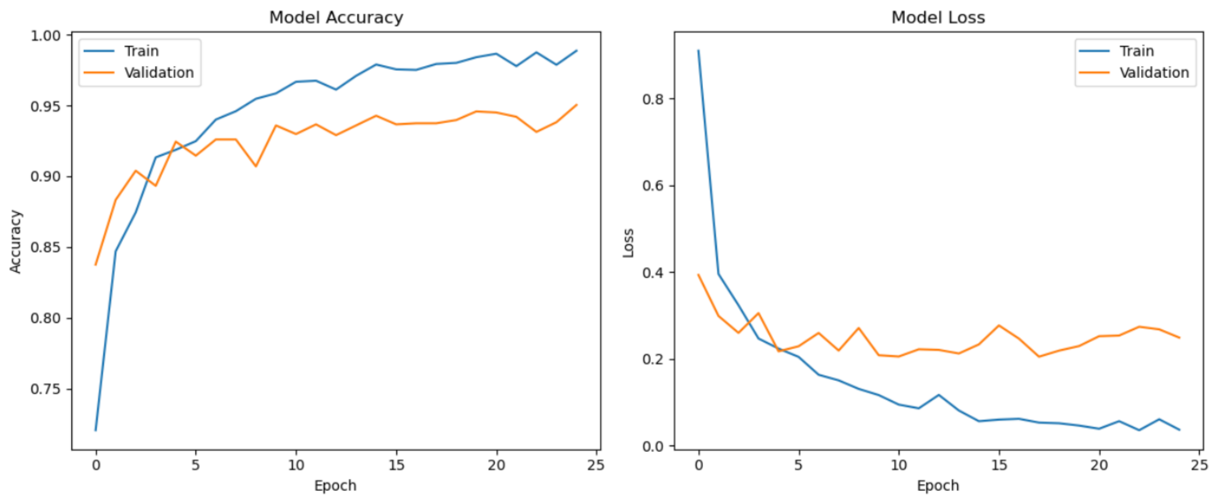
```
Epoch 1/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 45s 321ms/step - accuracy: 0.6056 - loss: 2.1149 - val_accuracy: 0.8375 - val_loss: 0.3936
Epoch 2/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 40s 301ms/step - accuracy: 0.8397 - loss: 0.4102 - val_accuracy: 0.8833 - val_loss: 0.2991
Epoch 3/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 299ms/step - accuracy: 0.8770 - loss: 0.3248 - val_accuracy: 0.9039 - val_loss: 0.2600
Epoch 4/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 296ms/step - accuracy: 0.9077 - loss: 0.2501 - val_accuracy: 0.8932 - val_loss: 0.3056
Epoch 5/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 301ms/step - accuracy: 0.9136 - loss: 0.2411 - val_accuracy: 0.9245 - val_loss: 0.2174
Epoch 6/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 299ms/step - accuracy: 0.9355 - loss: 0.1825 - val_accuracy: 0.9146 - val_loss: 0.2291
Epoch 7/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 298ms/step - accuracy: 0.9369 - loss: 0.1734 - val_accuracy: 0.9260 - val_loss: 0.2598
Epoch 8/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 300ms/step - accuracy: 0.9512 - loss: 0.1367 - val_accuracy: 0.9260 - val_loss: 0.2192
Epoch 9/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 295ms/step - accuracy: 0.9549 - loss: 0.1239 - val_accuracy: 0.9069 - val_loss: 0.2713
Epoch 10/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 295ms/step - accuracy: 0.9619 - loss: 0.1154 - val_accuracy: 0.9359 - val_loss: 0.2084
Epoch 11/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 296ms/step - accuracy: 0.9689 - loss: 0.0864 - val_accuracy: 0.9298 - val_loss: 0.2055
Epoch 12/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 294ms/step - accuracy: 0.9671 - loss: 0.0906 - val_accuracy: 0.9367 - val_loss: 0.2222
Epoch 13/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 295ms/step - accuracy: 0.9653 - loss: 0.1061 - val_accuracy: 0.9291 - val_loss: 0.2208
Epoch 14/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 295ms/step - accuracy: 0.9705 - loss: 0.0795 - val_accuracy: 0.9359 - val_loss: 0.2125
Epoch 15/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 299ms/step - accuracy: 0.9798 - loss: 0.0586 - val_accuracy: 0.9428 - val_loss: 0.2335
Epoch 16/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 40s 302ms/step - accuracy: 0.9744 - loss: 0.0651 - val_accuracy: 0.9367 - val_loss: 0.2772
Epoch 17/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 299ms/step - accuracy: 0.9723 - loss: 0.0725 - val_accuracy: 0.9375 - val_loss: 0.2467
Epoch 18/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 299ms/step - accuracy: 0.9806 - loss: 0.0518 - val_accuracy: 0.9375 - val_loss: 0.2051
```

```
Epoch 19/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 300ms/step - accuracy: 0.9799 - loss: 0.0536 - val_accuracy: 0.9397 - val_loss: 0.2191
Epoch 20/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 299ms/step - accuracy: 0.9817 - loss: 0.0480 - val_accuracy: 0.9458 - val_loss: 0.2296
Epoch 21/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 40s 302ms/step - accuracy: 0.9880 - loss: 0.0363 - val_accuracy: 0.9451 - val_loss: 0.2523
Epoch 22/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 297ms/step - accuracy: 0.9803 - loss: 0.0485 - val_accuracy: 0.9420 - val_loss: 0.2538
Epoch 23/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 300ms/step - accuracy: 0.9858 - loss: 0.0339 - val_accuracy: 0.9314 - val_loss: 0.2741
Epoch 24/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 39s 300ms/step - accuracy: 0.9777 - loss: 0.0576 - val_accuracy: 0.9382 - val_loss: 0.2681
Epoch 25/25
131/131 ━━━━━━━━━━━━━━━━━━━━ 40s 303ms/step - accuracy: 0.9875 - loss: 0.0424 - val_accuracy: 0.9504 - val_loss: 0.2491
```

```
[15]: history.history
```

```
[15]: {'accuracy': [0.7204198241233826,
        0.8469465374946594,
        0.8744274973869324,
        0.9133588075637817,
        0.9187023043632507,
        0.9248091578483582,
        0.9400763511657715,
        0.9459923505783081,
        0.9547709822654724,
        0.9585877656936646,
        0.9667938947677612,
        0.9675572514533997,
        0.9612595438957214,
        0.970992386341095,
        0.9790076613426208,
        0.9755725264549255,
        0.9751908183097839,
        0.9793893098831177,
        0.9801526665687561,
        0.984160304069519,
        0.9866412281990051,
        0.9778625965118408,
        0.9875954389572144,
        0.97881680727005,
        0.9887404441833496],
       'loss': [0.910325288772583,
        0.3959138095378876,
        0.32372719049453735,
        0.24694658815860748,
        0.22377540171146393,
        0.20465749502182007,
        0.16343843936920166,
        0.15049909055233002,
        0.13088104128837585,
        0.11658362299203873,
```

[17]:
```
import matplotlib.pyplot as plt  # Corrected import for matplotlib.pyplot

# Plot training and validation accuracy values
plt.figure(figsize=(12, 5))

# Subplot for accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])  # Corrected 'hisotry' to 'history'
plt.title("Model Accuracy")  # Corrected 'tittle' to 'title'
plt.xlabel("Epoch")  # Changed to 'Epoch' for clarity
plt.ylabel("Accuracy")  # Added ylabel for clarity
plt.legend(['Train', 'Validation'])

# Subplot for loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])  # Corrected 'hisotry' to 'history'
plt.title("Model Loss")  # Corrected 'tittle' to 'title'
plt.xlabel("Epoch")  # Changed to 'Epoch' for clarity
plt.ylabel("Loss")  # Added ylabel for clarity
plt.legend(['Train', 'Validation'])

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



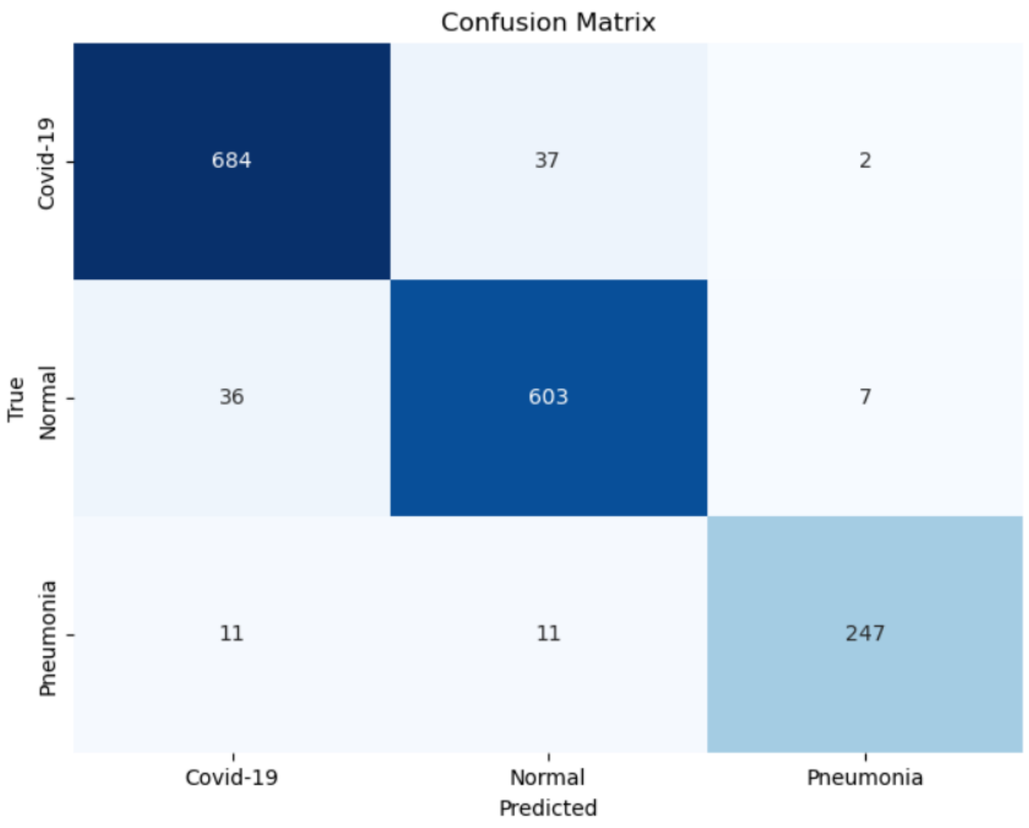
[19]:
```
#plotting Confusion Matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

predictions=model.predict(testX)
y_pred=np.argmax(predictions,axis=1)

y_true=np.argmax(testY,axis=1)

cm=confusion_matrix(y_true,y_pred)
print(cm)
```

```
52/52 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step
[[684  37   2]
 [ 36 603   7]
 [ 11  11 247]]
```

[21]:
```
class_names=le.classes_
class_names
```

[21]:
```
array(['Covid-19', 'Normal', 'Pneumonia'], dtype='<U9')
```

[23]:
```python
import pandas as pd
import seaborn as sns
from sklearn.metrics import confusion_matrix
# Create a DataFrame for the confusion matrix
confusion_Matrix = pd.DataFrame(cm, index=class_names, columns=class_names)

# Plot the confusion matrix using seaborn's heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_Matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```



[25]:
```python
from sklearn.metrics import classification_report
#classification report on test
print(classification_report(y_true,y_pred,target_names=le.classes_,digits=5))
```

```
              precision    recall  f1-score   support

    Covid-19    0.93570   0.94606   0.94085       723
      Normal    0.92627   0.93344   0.92984       646
   Pneumonia    0.96484   0.91822   0.94095       269

    accuracy                        0.93651      1638
   macro avg    0.94227   0.93257   0.93721      1638
weighted avg    0.93677   0.93651   0.93653      1638
```

[27]:
```python
#save model and label encoder

model.save("CNN_Covid19_Xray_Version.h5")
import pickle
pickle.dump(le,open("Label_encoder.pkl","wb"))
print("saved")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
saved
```

[29]:
```python
# Save model and label encoder
model.save("CNN_Covid19_Xray_Version.keras")  # Using the native Keras format
import pickle

# Save the label encoder using pickle
with open("Label_encoder.pkl", "wb") as f:
    pickle.dump(le, f)

print("Model and label encoder saved.")
```

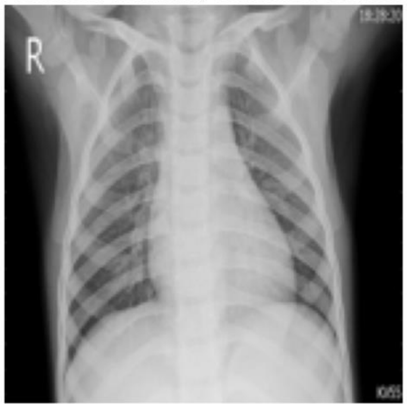
```
Model and label encoder saved.
```

[51]:
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

def detection_system(image_path, model, le, image_size=150):
    # Load the image
    image = cv2.imread(image_path)
    if image is None:
        raise ValueError(f"Image not found at this path: {image_path}")

    # Preprocessing
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Corrected capitalization
    image_resized = cv2.resize(image_rgb, (image_size, image_size))
    image_normalized = image_resized / 255.0  # Normalize the pixel values
    image_input = np.expand_dims(image_normalized, axis=0)  # Corrected function name

    # Make prediction
    predictions = model.predict(image_input)  # Ensure the correct variable name is used
    predicted_index = np.argmax(predictions)
    confidence_score = predictions[0][predicted_index]
    label = le.inverse_transform([predicted_index])[0]  # Corrected method name

    plt.imshow(image_resized)
    plt.title(f"Predicted: {label}, Confidence: {confidence_score * 100:.2f}%")  # Use label here
    plt.axis("off")
    plt.show()

    return label, confidence_score
```

[124]:
```python
# Viral Pneumonia
image_path = r"C:\Users\bhaga\Videos\COVID-19_Radiography_Dataset\Viral Pneumonia\images\Viral Pneumonia-2.png"
predicted_label, confidence_score = detection_system(image_path, model, le)
print(f'Predicted Label: {predicted_label}, Confidence Score: {confidence_score * 100:.2f}%')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
```

Predicted: Pneumonia, Confidence: 100.00%

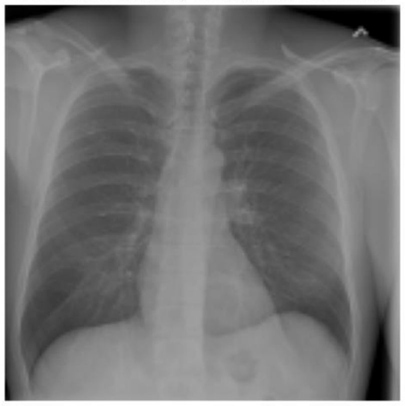


```
Predicted Label: Pneumonia, Confidence Score: 100.00%
```

```
[122]: #normal
image_path = r"C:\Users\bhaga\Videos\COVID-19_Radiography_Dataset\Normal\images\Normal-6.png"
predicted_label, confidence_score = detection_system(image_path, model, le)
print(f'Predicted Label: {predicted_label}, Confidence Score: {confidence_score * 100:.2f}%')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
```

Predicted: Normal, Confidence: 100.00%

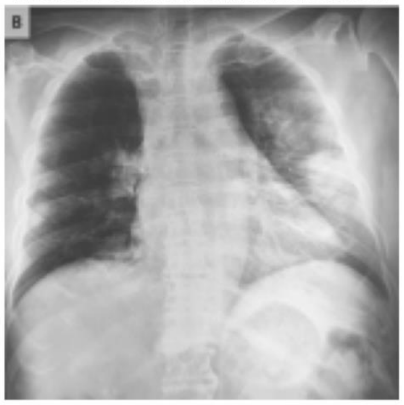


```
Predicted Label: Normal, Confidence Score: 100.00%
```

```
[120]: #covid
image_path = r"C:\Users\bhaga\Videos\COVID-19_Radiography_Dataset\COVID\images\COVID-5.png"
predicted_label, confidence_score = detection_system(image_path, model, le)
print(f'Predicted Label: {predicted_label}, Confidence Score: {confidence_score * 100:.2f}%')
```

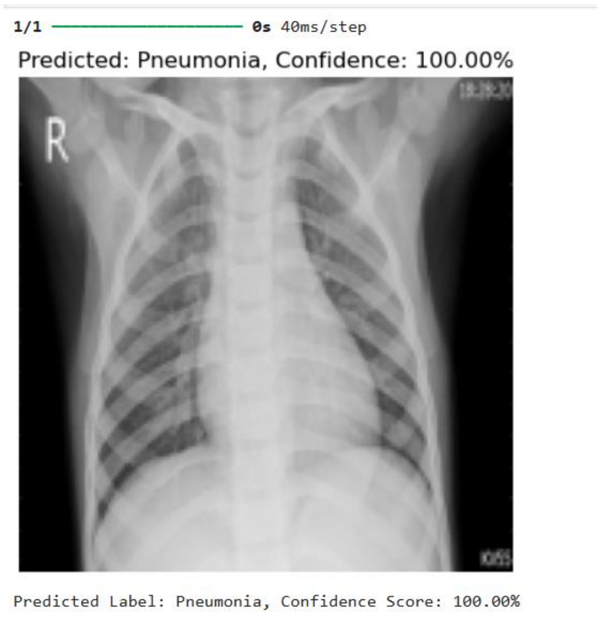
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```

Predicted: Covid-19, Confidence: 100.00%



```
Predicted Label: Covid-19, Confidence Score: 100.00%
```

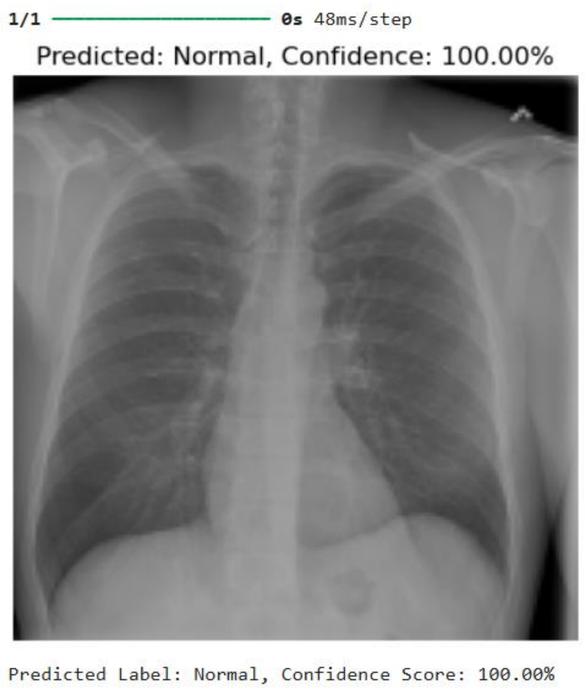
9. Results screenshots.



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
```

Predicted: Pneumonia, Confidence: 100.00%

```
Predicted Label: Pneumonia, Confidence Score: 100.00%
```



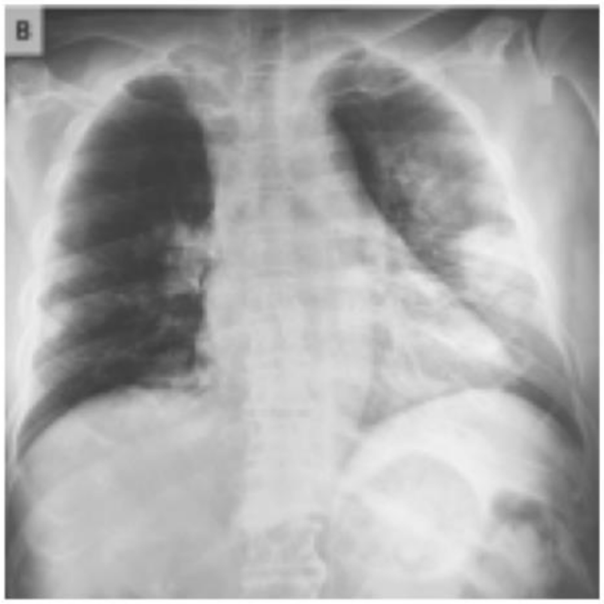
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
```

Predicted: Normal, Confidence: 100.00%

```
Predicted Label: Normal, Confidence Score: 100.00%
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```

Predicted: Covid-19, Confidence: 100.00%



```
Predicted Label: Covid-19, Confidence Score: 100.00%
```

```
              precision    recall  f1-score   support

    Covid-19    0.93570   0.94606   0.94085       723
      Normal    0.92627   0.93344   0.92984       646
   Pneumonia    0.96484   0.91822   0.94095       269

    accuracy                        0.93651      1638
   macro avg    0.94227   0.93257   0.93721      1638
weighted avg    0.93677   0.93651   0.93653      1638
```