All the codes used in this analysis were gotten from tensorflow website, Kaggle notebook, Medium article and tutorials and Github repositories

1. Business Understanding

- **Objective**: Detect COVID-19 using medical imaging.
- **Evaluate** and compare the performance of various transfer learning models.

2. Data Understanding

2.1. Data Loading and Visualisation

In [1]:

```
import os
import random
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.regularizers import l1_l2, l2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2, DenseNet121
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, BatchNo
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, LearningRa
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatri
import seaborn as sns
import cv2

# Set random seeds for reproducibility
seed_value = 42
tf.random.set_seed(seed_value)
np.random.seed(seed_value)
random.seed(seed_value)
```

In [2]:

```
data_dir = dataset_path = 'C:/Users/isacl/Downloads/Deep Learning project/COVID-19_
categories = ['COVID', 'Viral Pneumonia', 'Normal', 'Lung_Opacity']
IMG_SIZE = 224
SAMPLE_SIZE = 12

valid_extensions = ['.jpg', '.jpeg', '.png']

def load_images_from_category(category, num_images):
    path = os.path.join(data_dir, category)
    images = []
    labels = []
    label = categories.index(category)
    count = 0

    print(f"Loading images from: {path}")

    for root, dirs, files in os.walk(path):
        for file in files:
            if count >= num_images:
                break
            if any(file.lower().endswith(ext) for ext in valid_extensions):
                img_path = os.path.join(root, file)
                print(f"Reading image: {img_path}")
                image = cv2.imread(img_path)
                if image is not None:
                    image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RG
                    image = image / 255.0  # Normalize
                    images.append(image)
                    labels.append(label)
                    count += 1
                else:
                    print(f"Failed to read image: {img_path}")

    return images, labels

def load_sample_images(sample_size_per_category):
    all_images = []
    all_labels = []

    for category in categories:
        images, labels = load_images_from_category(category, sample_size_per_catego
        all_images.extend(images)
        all_labels.extend(labels)

    return np.array(all_images), np.array(all_labels)

# Load a sample set of images
images, labels = load_sample_images(SAMPLE_SIZE // len(categories))
print(f"Loaded {len(images)} images.")
```

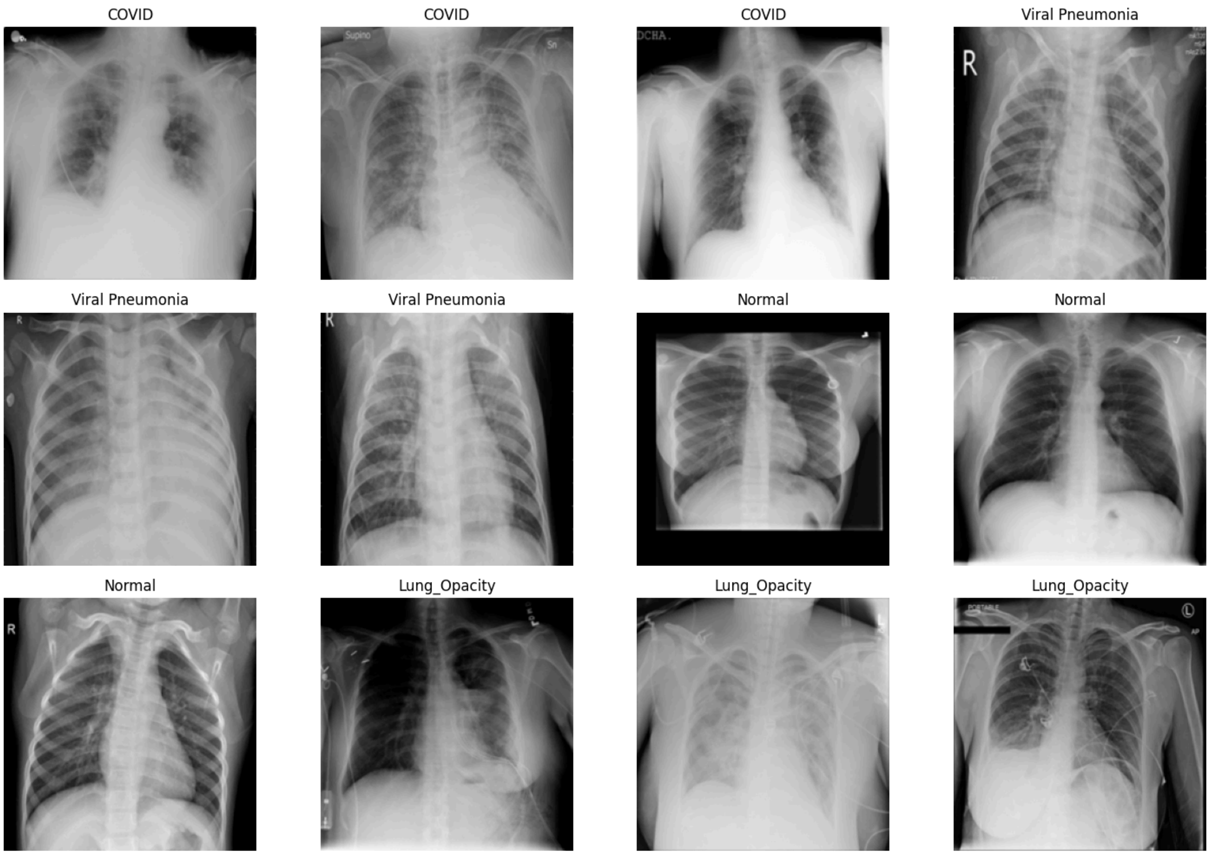
```
Loading images from: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiogr
aphy_Dataset\COVID
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\COVID\images\COVID-1.png
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\COVID\images\COVID-10.png
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\COVID\images\COVID-100.png
Loading images from: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiogr
aphy_Dataset\Viral Pneumonia
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\Viral Pneumonia\images\Viral Pneumonia-1.png
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\Viral Pneumonia\images\Viral Pneumonia-10.png
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\Viral Pneumonia\images\Viral Pneumonia-100.png
Loading images from: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiogr
aphy_Dataset\Normal
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\Normal\images\Normal-1.png
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\Normal\images\Normal-10.png
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\Normal\images\Normal-100.png
Loading images from: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiogr
aphy_Dataset\Lung_Opacity
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\Lung_Opacity\images\Lung_Opacity-1.png
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\Lung_Opacity\images\Lung_Opacity-10.png
Reading image: C:/Users/isacl/Downloads/Deep Learning project/COVID-19_Radiography_D
ataset\Lung_Opacity\images\Lung_Opacity-100.png
Loaded 12 images.
```

In [3]:

```python
def visualize_sample_images(images, labels, categories):
    if len(images) == 0:
        print("No images to display.")
        return

    plt.figure(figsize=(15, 10))
    for i in range(len(images)):
        plt.subplot(3, 4, i + 1)
        plt.imshow(images[i])
        plt.title(categories[labels[i]], fontsize=12)
        plt.axis('off')
    plt.tight_layout()
    plt.show()

visualize_sample_images(images, labels, categories)
```



3. Data Preprocessing

3.1. Creating Data Generator

In [4]:

```
# Create data generator with augmentation
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

train_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)

# Verify dataset sizes and batch sizes
print("Number of training samples:", train_generator.samples)
print("Number of validation samples:", validation_generator.samples)
print("Training batch size:", train_generator.batch_size)
print("Validation batch size:", validation_generator.batch_size)

# Calculate steps per epoch
steps_per_epoch_train = train_generator.samples // train_generator.batch_size
steps_per_epoch_val = validation_generator.samples // validation_generator.batch_si

print(f"Steps per epoch (train): {steps_per_epoch_train}")
print(f"Steps per epoch (val): {steps_per_epoch_val}")

# Use .repeat() to ensure the dataset generates enough batches
train_dataset = tf.data.Dataset.from_generator(
    lambda: (x for x in train_generator),
    output_signature=(
        tf.TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(None, 4), dtype=tf.float32)
    )
).repeat().prefetch(tf.data.AUTOTUNE)

validation_dataset = tf.data.Dataset.from_generator(
    lambda: (x for x in validation_generator),
    output_signature=(
        tf.TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(None, 4), dtype=tf.float32)
    )
).repeat().prefetch(tf.data.AUTOTUNE)
```

```
Found 33866 images belonging to 4 classes.
Found 8464 images belonging to 4 classes.
Number of training samples: 33866
Number of validation samples: 8464
Training batch size: 32
Validation batch size: 32
Steps per epoch (train): 1058
Steps per epoch (val): 264
```

4. Model Training and Evaluation

4.1. MobileNetV2 with L1 and L2 Regularization and Learning Rate Scheduling

In [5]:

```
def create_mobilenetv2_model():
    base_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights=

    # Unfreeze the last 50 layers
    for layer in base_model.layers[:-50]:
        layer.trainable = False
    for layer in base_model.layers[-50:]:
        layer.trainable = True
    model = Sequential([
        base_model,
        Flatten(),
        BatchNormalization(),
        Dense(128, activation='relu', kernel_regularizer=l1_l2(l1=0.001, l2=0.001))
        Dropout(0.5),
        Dense(4, activation='softmax', kernel_regularizer=l1_l2(l1=0.001, l2=0.001)
    ])
    optimizer = Adam(learning_rate=0.0001)  # Adjusted learning rate
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['a
    return model

mobilenetv2_model = create_mobilenetv2_model()

# Learning rate schedule
def scheduler(epoch, lr):
    if epoch < 10:
        return float(lr)
    else:
        return float(lr * tf.math.exp(-0.1))

lr_scheduler = LearningRateScheduler(scheduler)

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights
mobilenetv2_checkpoint = ModelCheckpoint('best_mobilenetv2_model.keras', monitor='v

history_mobilenetv2 = mobilenetv2_model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=20,
    steps_per_epoch=steps_per_epoch_train,
    validation_steps=steps_per_epoch_val,
    callbacks=[early_stopping, mobilenetv2_checkpoint, lr_scheduler]
)
```

```
Epoch 1/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1061s 988ms/step - accuracy: 0.6091 - loss: 28.2470 -
val_accuracy: 0.6838 - val_loss: 6.3597 - learning_rate: 1.0000e-04
Epoch 2/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 770s 728ms/step - accuracy: 0.7335 - loss: 4.0839 - v
al_accuracy: 0.7939 - val_loss: 2.1837 - learning_rate: 1.0000e-04
Epoch 3/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 772s 730ms/step - accuracy: 0.7559 - loss: 2.1561 - v
al_accuracy: 0.8744 - val_loss: 1.4531 - learning_rate: 1.0000e-04
Epoch 4/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 785s 742ms/step - accuracy: 0.7703 - loss: 1.6094 - v
al_accuracy: 0.8987 - val_loss: 1.0936 - learning_rate: 1.0000e-04
Epoch 5/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 777s 735ms/step - accuracy: 0.7839 - loss: 1.3422 - v
al_accuracy: 0.8255 - val_loss: 1.1396 - learning_rate: 1.0000e-04
Epoch 6/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 737s 697ms/step - accuracy: 0.7926 - loss: 1.1764 - v
al_accuracy: 0.8021 - val_loss: 1.0337 - learning_rate: 1.0000e-04
Epoch 7/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 750s 710ms/step - accuracy: 0.8055 - loss: 1.0777 - v
al_accuracy: 0.8987 - val_loss: 0.8451 - learning_rate: 1.0000e-04
Epoch 8/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 684s 647ms/step - accuracy: 0.8101 - loss: 1.0467 - v
al_accuracy: 0.9105 - val_loss: 0.7206 - learning_rate: 1.0000e-04
Epoch 9/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 655s 619ms/step - accuracy: 0.8078 - loss: 0.9847 - v
al_accuracy: 0.8898 - val_loss: 0.7474 - learning_rate: 1.0000e-04
Epoch 10/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 671s 635ms/step - accuracy: 0.8197 - loss: 0.9168 - v
al_accuracy: 0.9097 - val_loss: 0.7131 - learning_rate: 1.0000e-04
Epoch 11/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 697s 659ms/step - accuracy: 0.8272 - loss: 0.8660 - v
al_accuracy: 0.9126 - val_loss: 0.6741 - learning_rate: 9.0484e-05
Epoch 12/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 702s 664ms/step - accuracy: 0.8323 - loss: 0.8251 - v
al_accuracy: 0.9275 - val_loss: 0.5815 - learning_rate: 8.1873e-05
Epoch 13/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 676s 640ms/step - accuracy: 0.8410 - loss: 0.7655 - v
al_accuracy: 0.9210 - val_loss: 0.5771 - learning_rate: 7.4082e-05
Epoch 14/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 669s 633ms/step - accuracy: 0.8410 - loss: 0.7345 - v
al_accuracy: 0.9280 - val_loss: 0.5173 - learning_rate: 6.7032e-05
Epoch 15/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 664s 627ms/step - accuracy: 0.8492 - loss: 0.7005 - v
al_accuracy: 0.9181 - val_loss: 0.5181 - learning_rate: 6.0653e-05
Epoch 16/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 673s 637ms/step - accuracy: 0.8560 - loss: 0.6577 - v
al_accuracy: 0.9348 - val_loss: 0.4607 - learning_rate: 5.4881e-05
Epoch 17/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 679s 642ms/step - accuracy: 0.8598 - loss: 0.6321 - v
al_accuracy: 0.9320 - val_loss: 0.4603 - learning_rate: 4.9659e-05
Epoch 18/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 676s 639ms/step - accuracy: 0.8654 - loss: 0.6150 - v
al_accuracy: 0.9286 - val_loss: 0.4699 - learning_rate: 4.4933e-05
Epoch 19/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 674s 637ms/step - accuracy: 0.8678 - loss: 0.5929 - v
```

```
al_accuracy: 0.9298 - val_loss: 0.4412 - learning_rate: 4.0657e-05
Epoch 20/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 663s 627ms/step - accuracy: 0.8725 - loss: 0.5672 - v
al_accuracy: 0.9272 - val_loss: 0.4636 - learning_rate: 3.6788e-05
```

In [6]:

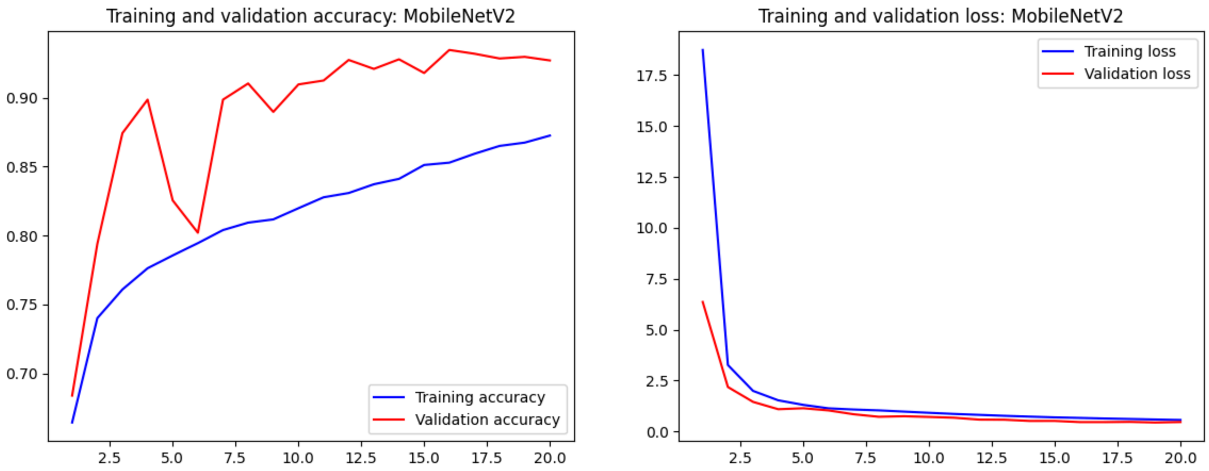
```python
# Function to plot training history
def plot_training_history(history, title):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(14, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'b', label='Training accuracy')
    plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
    plt.title(f'Training and validation accuracy: {title}')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'r', label='Validation loss')
    plt.title(f'Training and validation loss: {title}')
    plt.legend()

    plt.show()

# Plot history for MobileNetV2
plot_training_history(history_mobilenetv2, 'MobileNetV2')
```



In [7]:

```python
# Function to evaluate the model
def evaluate_model(model, validation_generator, steps):
    y_true = []
    y_pred = []

    for i in range(steps):
        x_val, y_val = next(validation_generator)
        preds = model.predict(x_val)
        y_true.extend(np.argmax(y_val, axis=1))
        y_pred.extend(np.argmax(preds, axis=1))
    # Calculate confusion matrix
    conf_matrix = confusion_matrix(y_true, y_pred)

    # Plot confusion matrix heatmap
    plt.figure(figsize=(10, 8))
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=vali
    disp.plot(cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.show()

    # Print classification report
    report = classification_report(y_true, y_pred, target_names=validation_generato
    print(report)

    # Calculate and print metrics
    accuracy = np.trace(conf_matrix) / np.sum(conf_matrix)
    precision = np.diag(conf_matrix) / np.sum(conf_matrix, axis=0)
    recall = np.diag(conf_matrix) / np.sum(conf_matrix, axis=1)
    specificity = (np.sum(conf_matrix) - np.sum(conf_matrix, axis=0) - np.sum(conf_
    f1_scores = 2 * (precision * recall) / (precision + recall)

    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision}')
    print(f'Recall: {recall}')
    print(f'Specificity: {specificity}')
    print(f'F1-Scores: {f1_scores}')

# Evaluate MobileNetV2 model
evaluate_model(mobilenetv2_model, validation_generator, steps_per_epoch_val)
```

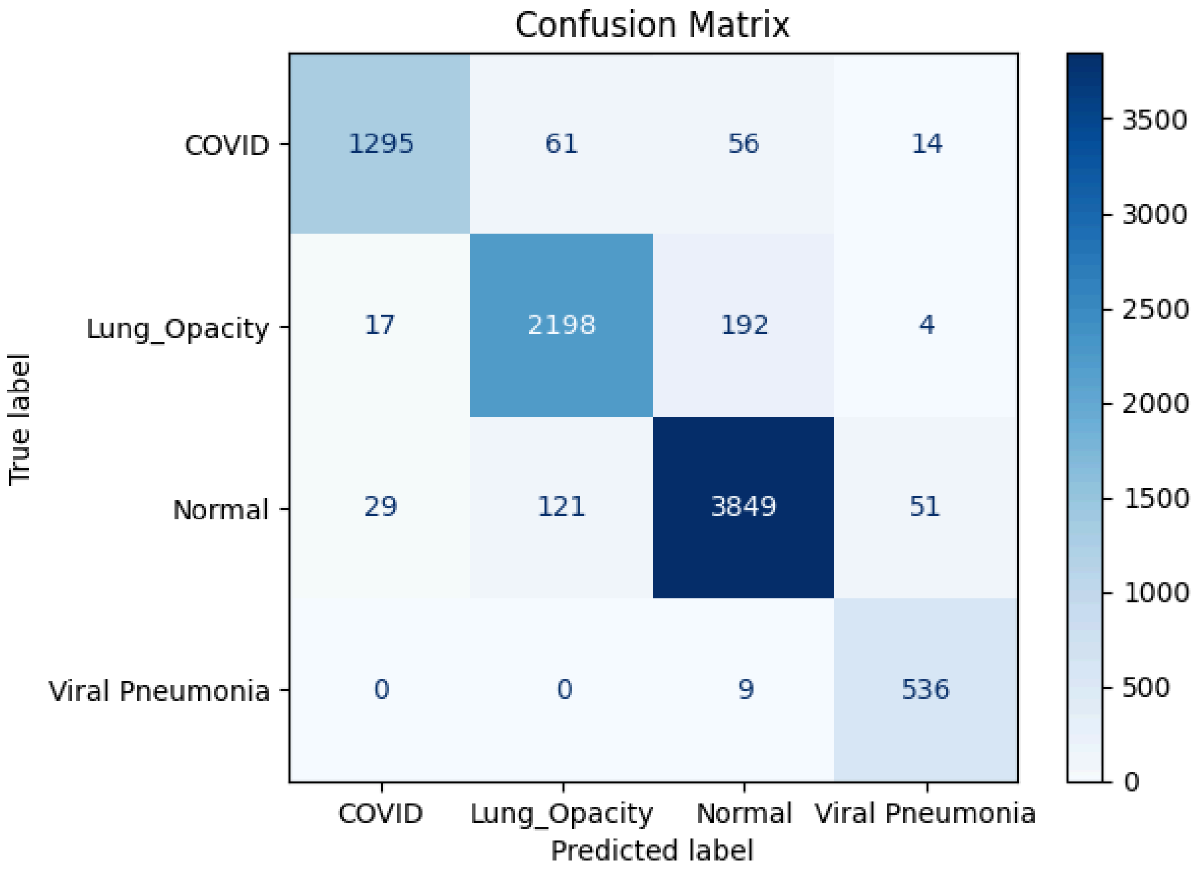
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 248ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 237ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 252ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 237ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 314ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 352ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 313ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 249ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 341ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 288ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 285ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 279ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 281ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 272ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 258ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 262ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 290ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 288ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 312ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 281ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 256ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 310ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 287ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 326ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 256ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 241ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 244ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 242ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 237ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 238ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 244ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 237ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 329ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 282ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 233ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 249ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 280ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 266ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 244ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 282ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 275ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 282ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 278ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 280ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 266ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 270ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 290ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 273ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 300ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 251ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 233ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 244ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 244ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 248ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 235ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 238ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 252ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 244ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 324ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 296ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 339ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 335ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 242ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 281ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 244ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 239ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 236ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 234ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 259ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 244ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 252ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 256ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 251ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 259ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 258ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 251ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 252ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 262ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 265ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 252ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 266ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 271ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 237ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 258ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 265ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 259ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 274ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 264ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 256ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 252ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 256ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 252ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 264ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 242ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 254ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 256ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 264ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 266ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 259ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 261ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 251ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 343ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 263ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 367ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 290ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 258ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 265ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 249ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 260ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 243ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 242ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 242ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 236ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 249ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 241ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 238ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 257ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 244ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 239ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 247ms/step
<Figure size 1000x800 with 0 Axes>
```



```
                 precision    recall  f1-score   support

          COVID       0.97      0.91      0.94      1426
   Lung_Opacity       0.92      0.91      0.92      2411
         Normal       0.94      0.95      0.94      4050
Viral Pneumonia       0.89      0.98      0.93       545

       accuracy                           0.93      8432
      macro avg       0.93      0.94      0.93      8432
   weighted avg       0.93      0.93      0.93      8432

Accuracy: 0.9343
Precision: [0.96569724 0.92352941 0.93740867 0.88595041]
Recall: [0.90813464 0.91165491 0.95037037 0.98348624]
Specificity: [0.9934342  0.96977246 0.94135098 0.99125143]
F1-Scores: [0.9360318  0.91755375 0.94384502 0.93217391]
```

4.2 DenseNet121 Model

In [26]:

```python
# Load DenseNet121 model pre-trained on ImageNet
def create_densenet_model():
    base_model = DenseNet121(input_shape=(224, 224, 3), include_top=False, weights=

    # Unfreeze the last few layers for fine-tuning
    for layer in base_model.layers[:-50]:
        layer.trainable = False
    for layer in base_model.layers[-50:]:
        layer.trainable = True
    model = Sequential([
        base_model,
        Flatten(),
        BatchNormalization(),
        Dense(128, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.5),
        Dense(4, activation='softmax', kernel_regularizer=l2(0.001))
    ])

    optimizer = Adam(learning_rate=0.0001)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['a
    return model

densenet_model = create_densenet_model()

# Callbacks for early stopping and saving the best model
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights
model_checkpoint = ModelCheckpoint('best_densenet_model.keras', monitor='val_loss',

# Learning rate scheduler
def scheduler(epoch, lr):
    return float(lr * tf.math.exp(-0.1))

lr_scheduler = LearningRateScheduler(scheduler)

# Training the DenseNet model
history_densenet = densenet_model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=20,
    steps_per_epoch=steps_per_epoch_train,
    validation_steps=steps_per_epoch_val,
    callbacks=[early_stopping, model_checkpoint, lr_scheduler]
)
```

```
Epoch 1/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1337s 1s/step - accuracy: 0.6301 - loss: 1.4840 - val
_accuracy: 0.8647 - val_loss: 0.6600 - learning_rate: 9.0484e-05
Epoch 2/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1266s 1s/step - accuracy: 0.7337 - loss: 0.9238 - val
_accuracy: 0.8678 - val_loss: 0.6001 - learning_rate: 8.1873e-05
Epoch 3/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1291s 1s/step - accuracy: 0.7520 - loss: 0.8599 - val
_accuracy: 0.8907 - val_loss: 0.5240 - learning_rate: 7.4082e-05
Epoch 4/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1474s 1s/step - accuracy: 0.7706 - loss: 0.7938 - val
_accuracy: 0.8987 - val_loss: 0.4983 - learning_rate: 6.7032e-05
Epoch 5/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1262s 1s/step - accuracy: 0.7794 - loss: 0.7612 - val
_accuracy: 0.8982 - val_loss: 0.4833 - learning_rate: 6.0653e-05
Epoch 6/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1248s 1s/step - accuracy: 0.7844 - loss: 0.7365 - val
_accuracy: 0.8991 - val_loss: 0.4709 - learning_rate: 5.4881e-05
Epoch 7/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1248s 1s/step - accuracy: 0.7926 - loss: 0.7162 - val
_accuracy: 0.9012 - val_loss: 0.4644 - learning_rate: 4.9659e-05
Epoch 8/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1237s 1s/step - accuracy: 0.8028 - loss: 0.6799 - val
_accuracy: 0.9025 - val_loss: 0.4668 - learning_rate: 4.4933e-05
Epoch 9/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1238s 1s/step - accuracy: 0.8059 - loss: 0.6671 - val
_accuracy: 0.9099 - val_loss: 0.4238 - learning_rate: 4.0657e-05
Epoch 10/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1249s 1s/step - accuracy: 0.8093 - loss: 0.6414 - val
_accuracy: 0.9166 - val_loss: 0.4201 - learning_rate: 3.6788e-05
Epoch 11/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1238s 1s/step - accuracy: 0.8116 - loss: 0.6271 - val
_accuracy: 0.9071 - val_loss: 0.4183 - learning_rate: 3.3287e-05
Epoch 12/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1249s 1s/step - accuracy: 0.8178 - loss: 0.6069 - val
_accuracy: 0.9125 - val_loss: 0.4290 - learning_rate: 3.0119e-05
Epoch 13/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1285s 1s/step - accuracy: 0.8198 - loss: 0.5981 - val
_accuracy: 0.9171 - val_loss: 0.3965 - learning_rate: 2.7253e-05
Epoch 14/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1283s 1s/step - accuracy: 0.8234 - loss: 0.5832 - val
_accuracy: 0.9160 - val_loss: 0.4155 - learning_rate: 2.4660e-05
Epoch 15/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1289s 1s/step - accuracy: 0.8286 - loss: 0.5707 - val
_accuracy: 0.9177 - val_loss: 0.3959 - learning_rate: 2.2313e-05
Epoch 16/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1301s 1s/step - accuracy: 0.8291 - loss: 0.5645 - val
_accuracy: 0.9176 - val_loss: 0.3835 - learning_rate: 2.0190e-05
Epoch 17/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1258s 1s/step - accuracy: 0.8296 - loss: 0.5567 - val
_accuracy: 0.9249 - val_loss: 0.3941 - learning_rate: 1.8268e-05
Epoch 18/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1243s 1s/step - accuracy: 0.8342 - loss: 0.5390 - val
_accuracy: 0.9233 - val_loss: 0.3622 - learning_rate: 1.6530e-05
Epoch 19/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1250s 1s/step - accuracy: 0.8348 - loss: 0.5374 - val
```

```
_accuracy: 0.9227 - val_loss: 0.3721 - learning_rate: 1.4957e-05
Epoch 20/20
1058/1058 ━━━━━━━━━━━━━━━━━━━━ 1292s 1s/step - accuracy: 0.8387 - loss: 0.5306 - val
_accuracy: 0.9224 - val_loss: 0.3686 - learning_rate: 1.3534e-05
```

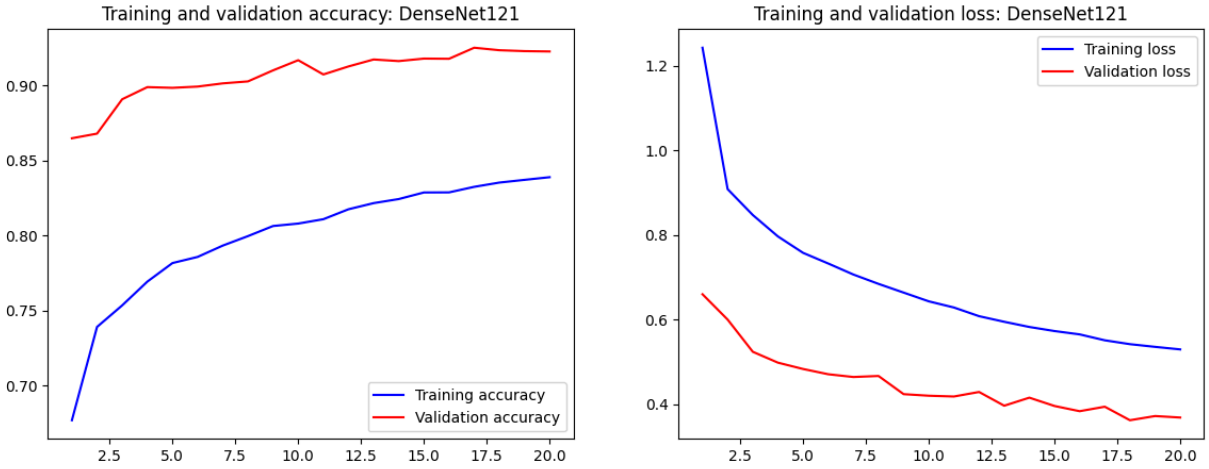
In [29]:
```
# Function to plot training history
def plot_training_history(history, title):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(14, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'b', label='Training accuracy')
    plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
    plt.title(f'Training and validation accuracy: {title}')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'r', label='Validation loss')
    plt.title(f'Training and validation loss: {title}')
    plt.legend()

    plt.show()

# Plot history for DenseNet121
plot_training_history(history_densenet, 'DenseNet121')
```



In [31]:
```
# Function to evaluate the model
def evaluate_model(model, validation_generator, steps):
    y_true = []
    y_pred = []

    for i in range(steps):
        x_val, y_val = next(validation_generator)
        preds = model.predict(x_val)
        y_true.extend(np.argmax(y_val, axis=1))
        y_pred.extend(np.argmax(preds, axis=1))
    # Calculate confusion matrix
    conf_matrix = confusion_matrix(y_true, y_pred)

    # Plot confusion matrix heatmap
    plt.figure(figsize=(10, 8))
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=vali
    disp.plot(cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.show()

    # Print classification report
    report = classification_report(y_true, y_pred, target_names=validation_generato
    print(report)

    # Calculate and print metrics
    accuracy = np.trace(conf_matrix) / np.sum(conf_matrix)
    precision = np.diag(conf_matrix) / np.sum(conf_matrix, axis=0)
    recall = np.diag(conf_matrix) / np.sum(conf_matrix, axis=1)
    specificity = (np.sum(conf_matrix) - np.sum(conf_matrix, axis=0) - np.sum(conf_
    f1_scores = 2 * (precision * recall) / (precision + recall)

    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision}')
    print(f'Recall: {recall}')
    print(f'Specificity: {specificity}')
    print(f'F1-Scores: {f1_scores}')

# Evaluate DenseNet121 model
evaluate_model(densenet_model, validation_generator, steps_per_epoch_val)
```

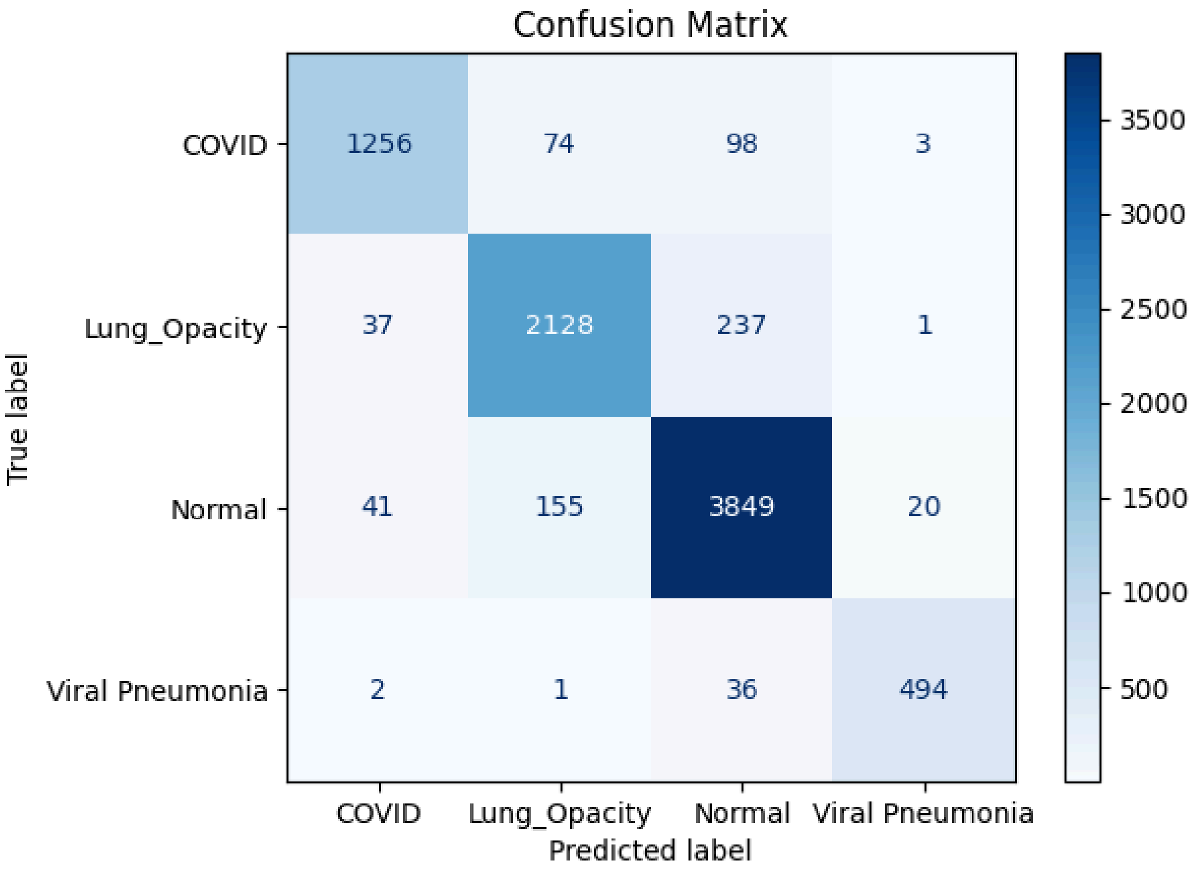
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 885ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 883ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 872ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 887ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 979ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 963ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 934ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 930ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 910ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 918ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 911ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 930ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 932ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 937ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 939ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 980ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 928ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 930ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 940ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 923ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 926ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 950ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 948ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 904ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 883ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 890ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 866ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 874ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 866ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 864ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 858ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 871ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 881ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 873ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 858ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 863ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 864ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 861ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 861ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 882ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 866ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 896ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 870ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 876ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 885ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 859ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 936ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 864ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 886ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 883ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 876ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 870ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 856ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 847ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 864ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 868ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 857ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 867ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 866ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 869ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 875ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 868ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 870ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 869ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 853ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 852ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 863ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 868ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 883ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 867ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 863ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 880ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 857ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 849ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 863ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 856ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 854ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 847ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 856ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 843ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 898ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 878ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 890ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 862ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 869ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 861ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 848ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 857ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 852ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 841ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 853ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 866ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 859ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 866ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 837ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 851ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 842ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 862ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 844ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 868ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 883ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 869ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 904ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 875ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 886ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 851ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 846ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 844ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 838ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 857ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 875ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 868ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 859ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 848ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 853ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 856ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 847ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 849ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 884ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 842ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 830ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 853ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 880ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 839ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 851ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 849ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 841ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 833ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 838ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 862ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 864ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 847ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 859ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 870ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 977ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 917ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 895ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 906ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 920ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 916ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 911ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 925ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 965ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 904ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 853ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 843ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 843ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 843ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 856ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 852ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 870ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 864ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 858ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 858ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 863ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 874ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 870ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 891ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 863ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 913ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 892ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 841ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 858ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 857ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 846ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 847ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 855ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 853ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 844ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 842ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 865ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 846ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 832ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 863ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 863ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 861ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 850ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 864ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 864ms/step
<Figure size 1000x800 with 0 Axes>
```



Confusion Matrix

True label \ Predicted label	COVID	Lung_Opacity	Normal	Viral Pneumonia
COVID	1256	74	98	3
Lung_Opacity	37	2128	237	1
Normal	41	155	3849	20
Viral Pneumonia	2	1	36	494

```
                 precision    recall  f1-score   support

          COVID       0.94      0.88      0.91      1431
   Lung_Opacity       0.90      0.89      0.89      2403
         Normal       0.91      0.95      0.93      4065
Viral Pneumonia       0.95      0.93      0.94       533

       accuracy                           0.92      8432
      macro avg       0.93      0.91      0.92      8432
   weighted avg       0.92      0.92      0.92      8432

Accuracy: 0.9164
Precision: [0.94011976 0.90245971 0.91208531 0.95366795]
Recall: [0.8777079  0.88555972 0.94686347 0.92682927]
Specificity: [0.98857306 0.96185105 0.91504465 0.99696164]
F1-Scores: [0.90784243 0.89392985 0.92914906 0.94005709]
```

```
In [ ]:
```