fd2lj5lfk

June 24, 2024

0.1 1. Loading Libraries

This section of the code is dedicated to importing all the necessary libraries required for building and training a Convolutional Neural Network (CNN) to classify medical images, specifically X-rays. TensorFlow and Keras are imported to construct and train the neural network. The Sequential model is used for creating a linear stack of layers, and various types of layers (Conv2D, MaxPooling2D, Flatten, Dense, Dropout, and GlobalAveragePooling2D) are included for building the CNN architecture. ImageDataGenerator is used for data augmentation, and callbacks such as EarlyStopping, ModelCheckpoint, and ReduceLROnPlateau are used to optimize the training process. VGG16 is a pre-trained model that can be used for transfer learning. Matplotlib is used for plotting images and results, numpy for numerical operations, sklearn for performance metrics, pathlib for file path operations, and os for interacting with the operating system. Additionally, random is used for random sampling.

[41]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,␣
 ↪ReduceLROnPlateau
from tensorflow.keras.applications import VGG16
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from pathlib import Path
import os
import random
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,␣
 ↪classification_report, accuracy_score, precision_score, recall_score,␣
 ↪f1_score
import matplotlib.image as mpimg
```

0.2 2. Preprocessing I

This code segment initializes three ImageDataGenerator objects to perform data augmentation and preprocessing on the images. The train_datagen is set up to rescale pixel values to the range [0, 1] and apply various augmentations (rotation, shifting, shearing, zooming, and flipping) to increase the diversity of the training data. The validation_datagen and test_datagen are simpler, only rescaling pixel values, as data augmentation is generally not applied to validation and test datasets.

[42]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)

validation_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

0.3 3. Load Data

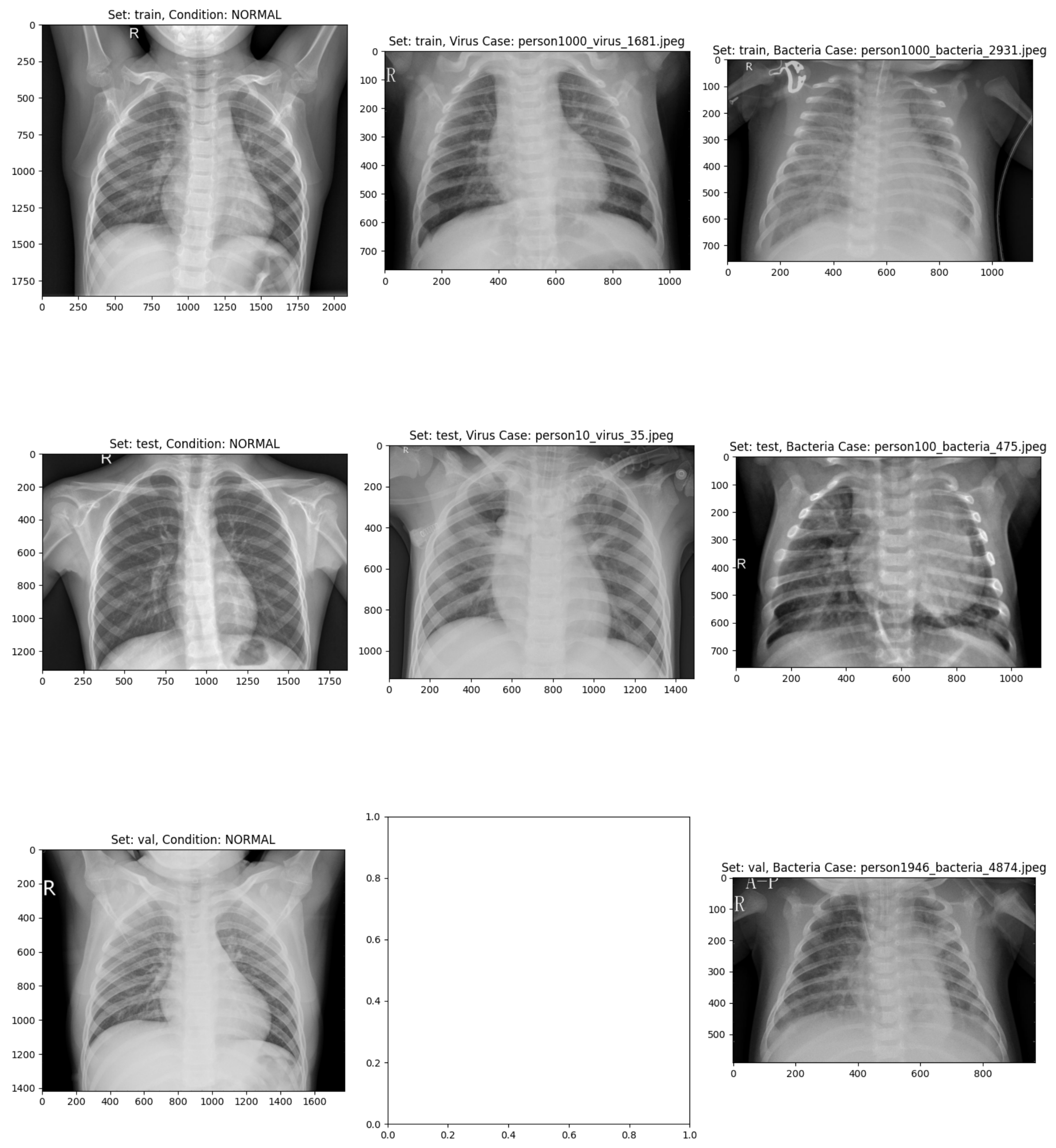
This code specifies the directory paths where the training, validation, and test datasets are stored. These paths are later used by the data generators to load the images for training and evaluation.

[43]:
```
train_dir = 'dataset/train'
test_dir =  'dataset/test'
validation_dir = 'dataset/val'
```

0.4 4. Sort Cases

This function, `sort_pneumonia_images`, iterates through a specified folder, identifying and categorizing images into those depicting pneumonia caused by viruses and bacteria based on the filename. It returns two lists: one for virus images and another for bacteria images. The `plot_images` function visualizes samples from the dataset, displaying a normal case, a virus pneumonia case, and a bacteria pneumonia case. The subsequent code defines paths to dataset directories and ensures that if the pneumonia directory exists and is not empty, the images are sorted and displayed using the defined functions. This process helps in visual verification of the dataset before training the model.

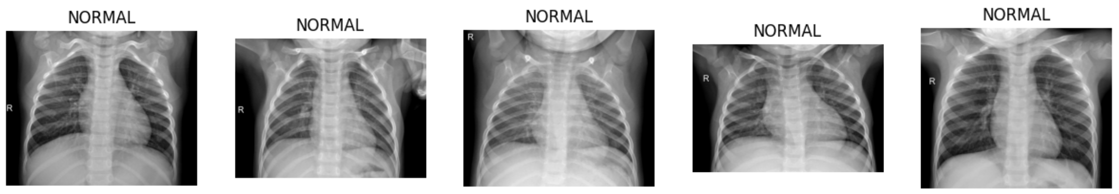
[44]:
```
def sort_pneumonia_images(folder_path):
    pneumonia_virus_images = []
    pneumonia_bacteria_images = []

    for filename in os.listdir(folder_path):
        file_path = os.path.join(folder_path, filename)
        if os.path.isfile(file_path) and filename.lower().endswith('.jpeg'):
            file_stem = Path(filename).stem.lower()
            if 'virus' in file_stem:
                pneumonia_virus_images.append(file_path)
            elif 'bacteria' in file_stem:
                pneumonia_bacteria_images.append(file_path)
    return pneumonia_virus_images, pneumonia_bacteria_images

def plot_images(dataset_name, image_dir, pneumonia_virus_images=None,␣
 ↪pneumonia_bacteria_images=None):
    images = [img for img in Path(image_dir).iterdir() if img.suffix.lower() ==␣
 ↪'.jpeg']

    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    ax = ax.ravel()

    if images:
        image_path = images[0]
        ax[0].imshow(plt.imread(str(image_path)), cmap='gray')
        ax[0].set_title(f'Set: {dataset_name}, Condition: NORMAL')

        if pneumonia_virus_images:
            ax[1].imshow(plt.imread(pneumonia_virus_images[0]), cmap='gray')
            ax[1].set_title(f'Set: {dataset_name}, Virus Case:␣
 ↪{Path(pneumonia_virus_images[0]).name}')

        if pneumonia_bacteria_images:
            ax[2].imshow(plt.imread(pneumonia_bacteria_images[0]), cmap='gray')
            ax[2].set_title(f'Set: {dataset_name}, Bacteria Case:␣
 ↪{Path(pneumonia_bacteria_images[0]).name}')

    plt.tight_layout()
    plt.show()


dataset_paths = {
    'train': train_dir,
    'test': test_dir,
    'val': validation_dir
}

for dataset_name, dataset_path in dataset_paths.items():
    normal_dir = os.path.join(dataset_path, 'NORMAL')
    pneumonia_dir = os.path.join(dataset_path, 'PNEUMONIA')

    if os.path.exists(pneumonia_dir) and os.listdir(pneumonia_dir):
        pneumonia_virus_images, pneumonia_bacteria_images =␣
 ↪sort_pneumonia_images(pneumonia_dir)
        plot_images(dataset_name, normal_dir, pneumonia_virus_images,␣
 ↪pneumonia_bacteria_images)
    else:
        print(f"Warning: Directory {pneumonia_dir} does not exist or is empty")
```

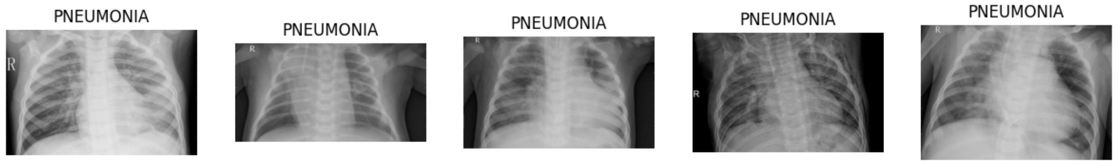

Set: train, Condition: NORMAL
Set: train, Virus Case: person1000_virus_1681.jpeg
Set: train, Bacteria Case: person1000_bacteria_2931.jpeg
Set: test, Condition: NORMAL
Set: test, Virus Case: person10_virus_35.jpeg
Set: test, Bacteria Case: person100_bacteria_475.jpeg
Set: val, Condition: NORMAL
Set: val, Bacteria Case: person1946_bacteria_4874.jpeg

0.5 5. Display Sample Data

This function, `display_sample_images`, selects and displays five random sample images from a specified class within a directory. It ensures the selected files are valid image formats and raises an error if there are not enough images. The images are then plotted using matplotlib, which helps visually inspect the data and ensure the dataset's quality and variety before training the model.

[45]:
```python
def display_sample_images(directory, class_name):
    class_dir = os.path.join(directory, class_name)
    # Filter out .DS_Store and non-image files
    valid_extensions = ('.png', '.jpg', '.jpeg')
    image_files = [f for f in os.listdir(class_dir) if f.lower().
↪endswith(valid_extensions)]

    if len(image_files) < 5:
        raise ValueError(f"Not enough images in {class_name} directory to␣
↪display.")

    sample_images = random.sample(image_files, 5)

    plt.figure(figsize=(15, 5))
    for i, img_name in enumerate(sample_images):
        img_path = os.path.join(class_dir, img_name)
        img = mpimg.imread(img_path)
        plt.subplot(1, 5, i+1)
        plt.imshow(img, cmap='gray')
        plt.title(class_name)
        plt.axis('off')
    plt.show()

display_sample_images(train_dir, 'NORMAL')
display_sample_images(train_dir, 'PNEUMONIA')
```





0.6 6. Preprocessing II

This section initializes data generators for training, validation, and testing datasets using the previously defined ImageDataGenerator instances. It specifies the target image size (224x224 pixels), batch size, and class mode (binary classification). The train_generator includes data augmentation, while the validation_generator and test_generator simply rescale the images. The test_generator does not shuffle the data to maintain the order for evaluation consistency.

[46]:
```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
```

```
Found 5216 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

0.7 7. Lets Deploy A CNN Model

This section defines the architecture of the Convolutional Neural Network (CNN) using Keras' Sequential API. It starts with a convolutional layer with 32 filters, followed by max-pooling. This pattern is repeated with increasing filter sizes (64, 128). After the convolutional layers, the output is flattened and passed through a dense layer with 512 neurons and a dropout layer to prevent overfitting. Finally, a dense layer with a sigmoid activation function outputs the binary classification result. The model is then compiled using the Adam optimizer and binary cross-entropy loss function, with accuracy as the evaluation metric.

[47]:
```
base_model = VGG16(input_shape=(224, 224, 3), include_top=False,␣
 ↪weights='imagenet')
base_model.trainable = False

model = Sequential([
    base_model,
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])
```

0.8 8. Lets Start Training

This section sets up the training process for the model. It defines three callbacks: EarlyStopping to halt training if the validation loss doesn't improve for 5 epochs, ModelCheckpoint to save the best model during training, and ReduceLROnPlateau to reduce the learning rate if the validation loss plateaus. The model is then trained using the fit method, specifying the training and validation data generators, number of epochs, and the callbacks to be used. The training process adjusts the model's parameters to minimize the loss and improve accuracy on the validation data.

[48]:
```
early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
↪restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True,␣
↪monitor='val_loss')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2,␣
↪min_lr=0.001)

# Train
history = model.fit(
    train_generator,
    epochs=20,
    validation_data=validation_generator,
    callbacks=[early_stopping, model_checkpoint, reduce_lr]
)
```

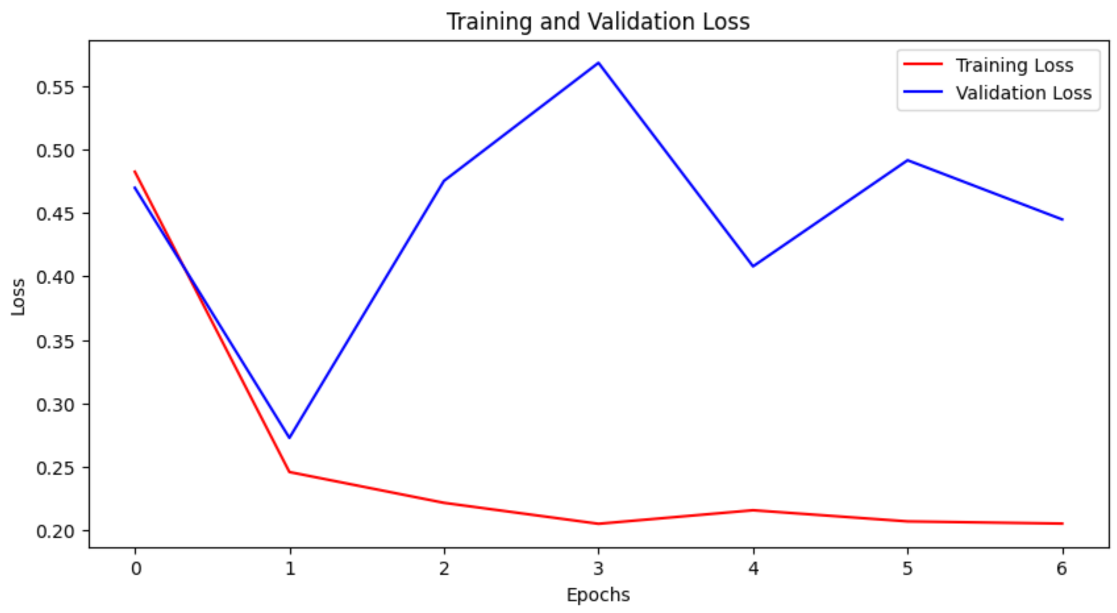
```
Epoch 1/20

c:\Users\m.el-hachmi-
mahti\Desktop\workspace\2024\epitech\IA\T-DEV-810-TLS_5\env\Lib\site-
packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:120:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()

163/163               949s 6s/step -
accuracy: 0.8072 - loss: 0.9789 - val_accuracy: 0.7500 - val_loss: 0.4697 -
learning_rate: 0.0010
Epoch 2/20
163/163               15128s 93s/step -
accuracy: 0.8918 - loss: 0.2517 - val_accuracy: 0.9375 - val_loss: 0.2727 -
learning_rate: 0.0010
Epoch 3/20
163/163               696s 4s/step -
accuracy: 0.9135 - loss: 0.2224 - val_accuracy: 0.8125 - val_loss: 0.4752 -
```

```
learning_rate: 0.0010
Epoch 4/20
163/163               757s 5s/step -
accuracy: 0.9057 - loss: 0.2134 - val_accuracy: 0.6875 - val_loss: 0.5682 -
learning_rate: 0.0010
Epoch 5/20
163/163               775s 5s/step -
accuracy: 0.9134 - loss: 0.2120 - val_accuracy: 0.8750 - val_loss: 0.4078 -
learning_rate: 0.0010
Epoch 6/20
163/163               801s 5s/step -
accuracy: 0.9145 - loss: 0.2092 - val_accuracy: 0.7500 - val_loss: 0.4914 -
learning_rate: 0.0010
Epoch 7/20
163/163               806s 5s/step -
accuracy: 0.9235 - loss: 0.1978 - val_accuracy: 0.8125 - val_loss: 0.4448 -
learning_rate: 0.0010
```

0.9 9. Model Evaluation

0.9.1 A. Accessing Loss Values for Training and Validation Sets

[49]:
```python
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Generate a range of epochs for the x-axis
epochs = range(len(train_loss))
# Visualize the loss function
plt.figure(figsize=(10, 5))
plt.plot(epochs, train_loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



0.9.2 B. model summary

This command prints a detailed summary of the model architecture. It includes the names and types of each layer, their output shapes, the number of parameters in each layer, and the total number of trainable and non-trainable parameters. This summary helps in understanding the structure and complexity of the model, ensuring all layers are correctly configured and providing insight into the model's capacity and depth.

[50]:
```
model.summary()
```

```
Model: "sequential_2"


 Layer (type)                       Output Shape               Param #

 vgg16 (Functional)                 (None, 7, 7, 512)       14,714,688

 flatten_2 (Flatten)                (None, 25088)                    0

 dense_4 (Dense)                    (None, 512)             12,845,568

 dropout_2 (Dropout)                (None, 512)                      0

 dense_5 (Dense)                    (None, 1)                      513
```

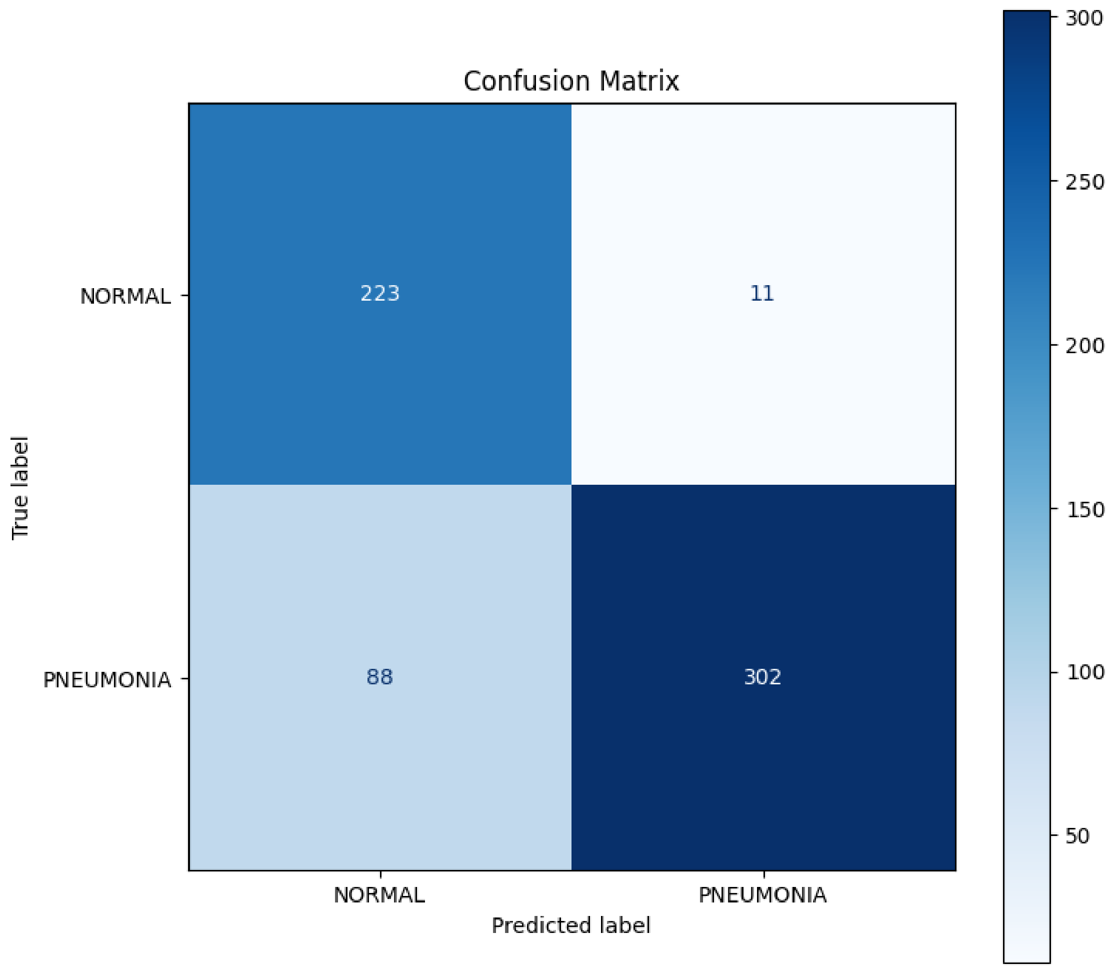
```
Total params: 53,252,933 (203.14 MB)

Trainable params: 12,846,081 (49.00 MB)

Non-trainable params: 14,714,688 (56.13 MB)

Optimizer params: 25,692,164 (98.01 MB)
```

0.9.3 C. Confusion Matrix

This section evaluates the trained model on the test dataset and generates a confusion matrix to visualize the performance. The test generator is reset to ensure a fresh evaluation. True labels are retrieved, and predictions are made by the model. Predicted probabilities are converted to binary class labels. A confusion matrix is then created to compare true labels with predicted labels, and it's displayed using matplotlib. This matrix provides a clear view of the model's classification performance, showing how many instances were correctly and incorrectly classified for each class.

[51]:
```python
# Ensure the test generator is reset
test_generator.reset()

# Get the true labels and predictions
test_labels = test_generator.classes
test_pred_prob = model.predict(test_generator)
test_pred_classes = (test_pred_prob > 0.5).astype("int32").flatten()

# Generate confusion matrix
cm = confusion_matrix(test_labels, test_pred_classes)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['NORMAL',␣
 ↪'PNEUMONIA'])

# Plot confusion matrix and save as PNG
plt.figure(figsize=(8, 8))
disp.plot(cmap=plt.cm.Blues, ax=plt.gca(), values_format='d')
plt.title('Confusion Matrix')
plt.savefig('confusion_matrix.png')
plt.show()
```

```
20/20                 96s 5s/step
```



0.9.4 D. Classification Report

This part generates and prints a classification report, which provides detailed metrics of the model's performance on the test set. It includes precision, recall, F1-score, and support for each class. These metrics give a comprehensive understanding of the model's strengths and weaknesses in classifying each class, beyond what is evident from the confusion matrix alone. The classification report helps in identifying any bias towards a particular class and in assessing the overall effectiveness of the model.

[52]:
```
report = classification_report(test_labels, test_pred_classes,␣
 ↪target_names=['NORMAL', 'PNEUMONIA'])
print("Classification Report:\n", report)
```

```
Classification Report:
               precision    recall  f1-score   support

      NORMAL       0.72      0.95      0.82       234
```

```
   PNEUMONIA       0.96      0.77      0.86       390

    accuracy                           0.84       624
   macro avg       0.84      0.86      0.84       624
weighted avg       0.87      0.84      0.84       624
```

0.9.5 E. Save Model

This command saves the trained model to a file named 'model.keras'. Saving the model allows it to be loaded later for making predictions without retraining, which is useful for deployment and further evaluation. The saved model includes the architecture, weights, and training configuration, ensuring that it can be fully restored and used in future applications.

[58]:
```
# Save the model in HDF5 format
model.save('model.keras')
```