PNEUMONIA DETECTION FROM CHEST X-RAYS

USING DEEP LEARNING

META SCIFOR TECHNOLOGIES

2024

Script. Sculpt. Socialize

Guided By:

Saurav Kumar

Submitted By:

Bhushan Khushal Nimje

MST02-0020

TABLE OF CONTENT

ABSTRACT

This project focuses on developing a Convolutional Neural Network (CNN) for image classification tasks, leveraging the power of deep learning to automatically categorize images into predefined classes. Utilizing the dataset consisting of images categorized into various classes such as “Normal” and “Pneumonia” the project encompasses several critical phases: data collection, data preprocessing, model design, model training and model evaluation.

The dataset undergoes thorough preprocessing to standardize image dimensions, normalize pixel values and apply data augmentation techniques to improve model robustness. Several CNN architectures are used incorporating multiple convolutional layers, activation functions, batch normalization and pooling operations to effectively extract and learn relevant features from the images.

The model is trained on the pre-processed dataset and evaluated using performance metrics such as accuracy, precision and recall. The results demonstrate CNN’s capability to achieve high classification accuracy and robust performance across different image categories. Challenges encountered during training was data quality which is addressed.

The project showcases the efficacy of CNNs in image classification and highlights the importance of proper data handling and model architectures in achieving optional results. The insights gained contribute to the understanding of CNN applications in real-world scenarios and provide a foundation for future enhancements and research in the field of computer vision.

INTRODUCTION

Pneumonia Detection is one the evolving field in medical using artificial intelligence, which plays a significant role in enhancing interactions between humans and machines. This project centres on developing a Convolutional Neural Network (CNN) for classifying images into two distinct medical conditions: “Normal” and “Pneumonia”.

Understanding and categorizing medical conditions through visual data can revolutionize multiple sub-fields in medical. By using deep learning techniques, specifically CNN, the projects seek to build a model capable of accurately distinguishing between Normal and Pneumonia medical condition.

The project encompasses several phases: collecting a diverse set of images, preprocessing them to ensure consistency, constructing a CNN model to learn from the image data and evaluating its performance. Utilizing Tensorflow and Keras the model is trained to recognize patterns and features indicative of different medical situations, aiming to achieve high accuracy and robustness in various condition.

Ultimately this project highlights the potential of deep learning for Pneumonia Detection and its practical implications in creating more intuitive and responsive systems that can understand and interact with human emotions more effectively.

CONVOLUTIONAL NEURAL NETWORK (CNN)

Convolutional Neural Network (CNN) is a part of deep learning it is designed to automatically learn the features from input images. CNN is particularly effective for tasks involving image data due to their ability to capture local patterns.

Key Components of CNN: -

1. **Convolutional Layers:** These layers apply convolutional filters to the input images to detect patterns such as edges, textures and shapes. They help in extracting features from the image.
2. **Pooling Layers:** Pooling layers such as MaxPooling2D reduce the dimensionality of the features maps while retaining the important information. This process helps in decreasing the computational load and mitigating overfitting.
3. **Batch Normalization:** Batch Normalization is a technique used in Convolutional Neural Networks (CNNs) to improve the training speed and performance of the model.
4. **Flattening:** After the convolutional and pooling layers, the 3D feature maps are flattened into 1D vectors. This step prepares the data for the fully connected layers.
5. **Dense Layers:** These layers perform the final classification based on the features extracted by the convolutional layers. A common activation function is used in the final layer that is sigmoid function, which outputs binary classification results.

Dataset Pre-Processing:

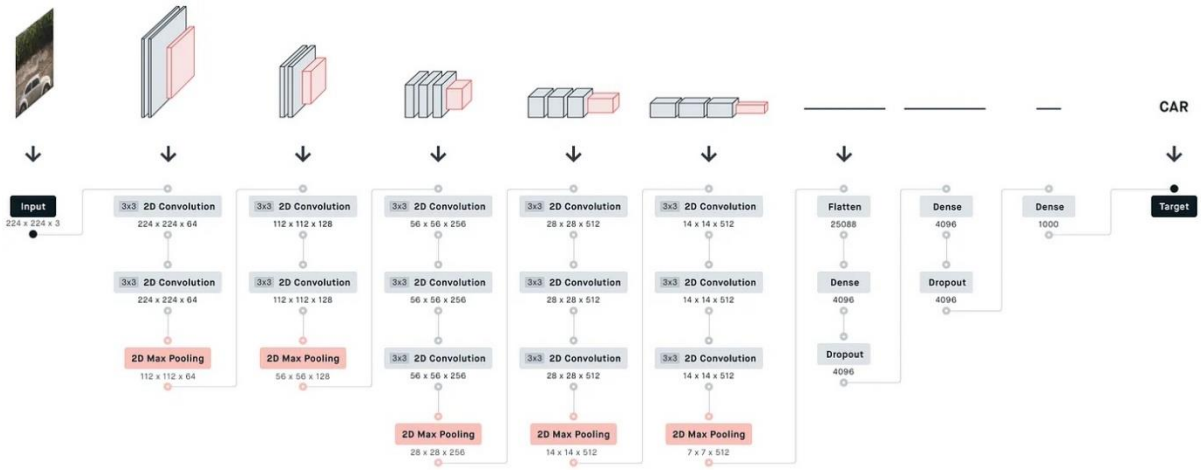
For training a CNN image classifier a well-prepared dataset is essential. In a typical image classification task, the dataset consists of image categorized into different classes. For example, in my dataset the images were labelled as “Normal” and “Pneumonia”.

- **Pre-Processing Steps:**
    - **Data Cleaning:** Removing non-images files and irrelevant data.

- **Image Augmentation and Normalization:** The images are augmented (rescaled, rotated, flipped, etc.) and normalized to improve model robustness.

Training and Evaluation:

Training a CNN involves optimizing the model to minimize the loss function, commonly binary cross entropy for binary classification tasks.



TECHNOLOGIS USED

The technology used in this project consists a variety of tools essential for building a deep learning model. Below is a detailed overview of the technologies used:

1. Python Programming Language:

- The primary language used for data analysis, preprocessing, and deep learning model development.

2. Libraries and Packages:

- **NumPy:** Used for numerical operations, particularly for array and matrix operations required in data preprocessing.
- **Matplotlib:** Employed for visualizing data through plots and graphs, such as boxplots for identifying outliers.
- **Seaborn**: A statistical data visualization library built on Matplotlib, used to create more aesthetically pleasing and informative plots.
- **OS**: The os module in Python provides a way to interact with the operating system. It allows you to perform tasks like reading or writing files, manipulating paths, creating directories, and handling environment variables.
- **TensorFlow**: Tensorflow is an open-source machine learning framework developed by Google. It is used for building and deploying machine learning models, particularly deep learning models.
- **Keras**: Keras is a high-level API for building and training deep learning models. It is part of tensorflow and simplifies the process of creating complex neural networks.
- **ImageDataGenerator**: This class provides functionality for generating batches of tensor image data with real-time data augmentation. It helps to artificially expand the size of a training dataset by creating new variations of existing images.
- **Sequential**: Sequential is a linear stack of layers in keras. It allows you to create models layer by layer.
- **Dense**: Dense is a fully connected layer in a neural network. It is often used in the final layers of a network.

- **Conv2D**: Conv2D is a 2D convolutional layer used in CNNs (Convolutional Neural Networks). It applies convolution operations to the input image to detect features like edges, textures, etc.
- **Flatten**: Flatten is a layer that flattens the input, i.e., it converts a multi-dimensional tensor into a single-dimensional tensor.
- **MaxPooling2D**: MaxPooling2D is a pooling layer that down samples the input by taking the maximum value over a specified window. It helps to reduce the spatial dimensions of the input, reducing computation and helping to prevent overfitting.
- **Dropout**: Dropout is a regularization technique used to prevent overfitting. It randomly drops a fraction of the input units during training.
- **Batch Normalization**: Batch Normalization normalizes the output of a previous activation layer by re-centering and re-scaling the inputs. It helps in stabilizing and speeding up the training process.

3. Data Preprocessing Techniques:

- The data is augmented and normalized using "**ImageDataGenerator".** This helps improve the model's generalization ability.

4. Model Development:

- Four different models are developed:
    1. **VGG16:** A custom-built version of the VGG16 architecture.
    2. **VGG19:** A pre-trained VGG19 architecture with transfer learning.
    3. **ResNet50:** A pre-trained ResNet50 architecture.
    4. **InceptionV3:** A pre-trained InceptionV3 architecture.

5. Model Training:

- **Model Training:** Using the .fit() method.

6. Model Evaluation:

- Each model is evaluated on the test set, and the performance metrics include accuracy, confusion matrix and classification report.
- Additionally, validation accuracy is calculated to help select the best model for final testing.

7. Testing:

- You load an external chest X-ray image, preprocess it and use the InceptionV3 model to make the prediction.

DATASET INFORMATION

The dataset used in this project consists of images categorized into two classes: **“Normal”** and **“Pneumonia”**.

1. **Dataset:**
    - Dataset two classes in which total “**Normal”** images are 1341 and total **“Pneumonia”** images are 3875.
2. **Data Pre-Processing and Augmentation:**
    - Images were resized to uniform size of 224x224 pixels suitable for input into the CNN.
    - Applied normalization to scale pixel values to the range [0,1].

METHODOLOGY

1. **Data Preprocessing:**
    a. **Image Augmentation and Normalization:** The images are augmented (rescaled, rotated, flipped, etc.) and normalized to improve model robustness.
    b. **Data Generators**: ImageDataGenerator is used to feed the images into the model in batches for training, validation, and testing.
2. **Model Development:**
    a. **VGG16 Model:** A convolutional neural network (CNN) model built from scratch using a similar architecture to VGG16.
    b. **VGG19, ResNet50, InceptionV3 Models:** These are built using pre-trained models on ImageNet, which are fine-tuned for the binary classification task. The pre-trained layers are frozen except for the last few blocks, and additional fully connected layers are added.
    c. **Training:** Each model is trained for 25 epochs, with accuracy and loss tracked over epochs.
3. **Model Evaluation:**
    a. **Accuracy and Loss Graph:** For each model, training and validation accuracy and loss are plotted.
    b. **Confusion Matrix and Classification Reports:** These are used to evaluate model performance on the test data, displaying how well the model distinguishes between “Normal” and “Pneumonia” cases.
4. **Validation Comparison:**
    a. The validation accuracy of each model is compared to select the best-performing model for further testing.
5. **Testing:**
    a. A sample chest X-ray image is loaded, preprocessed and then classified using the InceptionV3 model because of high validation score compared to other models.

CODE SNIPPET

Installing Dependencies

[1]:
```
# Importing libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout, BatchNormalization
from sklearn.metrics import classification_report, confusion_matrix

import warnings
warnings.filterwarnings('ignore')
```

Data Loading

[2]:
```
# Set the directory paths for the dataset
train_dir = "chest_xray/train/"
test_dir = "chest_xray/test/"
val_dir = "chest_xray/val/"
```

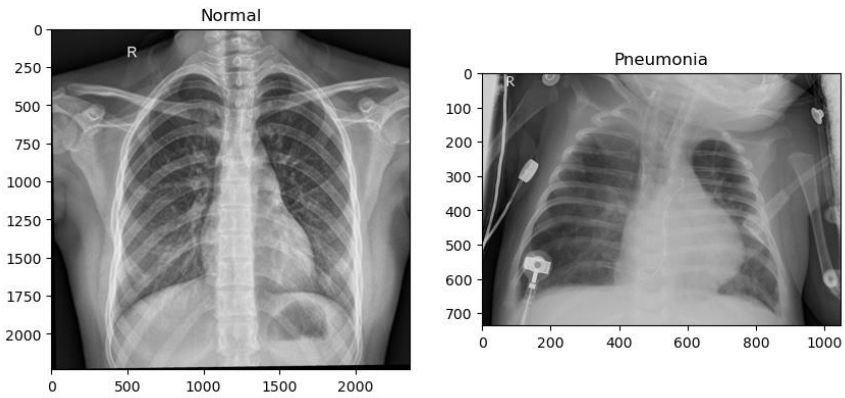
[3]:
```
# Explore distribution of the data
def explore_data(directory):
    categories = ["Normal", "Pneumonia"]
    for category in categories:
        path = os.path.join(directory, category)
        print(f"{category}: {len(os.listdir(path))} images")


explore_data(train_dir)
```

```
Normal: 1341 images
Pneumonia: 3875 images
```

[4]:
```
# Visualize sample images
def plot_sample_images(directory):
    categories = ["Normal", "Pneumonia"]
    fig, axes = plt.subplots(1,2, figsize=(10,5))
    for i, category in enumerate(categories):
        path = os.path.join(directory, category)
        image_path = os.listdir(path)[0]
        img = plt.imread(os.path.join(path, image_path))
        axes[i].imshow(img, cmap='grey')
        axes[i].set_title(category)
    plt.show()

plot_sample_images(train_dir)
```



Data Preprocessing

[5]:
```
# ImageDataGenerator for Augmentation and Normalization
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    rotation_range=20,
    horizontal_flip=True,
    fill_mode='nearest'
)


val_datagen = ImageDataGenerator(rescale = 1.0/255.0)
test_datagen = ImageDataGenerator(rescale = 1.0/255.0)
```

[6]:
```
# Flow images from directories
train_generator = train_datagen.flow_from_directory(
    train_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
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary'
)
```

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

Model Preparation

3.1 Model Development with VGG16 Architecture

[7]:
```
# Create a sequential model
model_vgg16 = Sequential()
```

[8]:
```
# 1st Convolutional Block
# 1st Convolutional Layer
model_vgg16.add(Conv2D(filters=64, input_shape=(224,224,3), kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# 2nd Convolutional Layer
model_vgg16.add(Conv2D(filters=64, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# Max Pooling
model_vgg16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

# 2nd Convolutional Block
# 3rd Convolutional Layer
model_vgg16.add(Conv2D(filters=128, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# 4th Convolutional Layer
model_vgg16.add(Conv2D(filters=128, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# Max Pooling
model_vgg16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

# 3rd Convolutional Block
# 5th Convolutional Layer
model_vgg16.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# 6th Convolutional Layer
model_vgg16.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# 7th Convolutional Layer
model_vgg16.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# Max Pooling
model_vgg16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

# 4th Convolutional Block
# 8th Convolutional Layer
model_vgg16.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# 9th Convolutional Layer
model_vgg16.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# 10th Convolutional Layer
model_vgg16.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# Max Pooling
model_vgg16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
```

```
# 5th Convolutional Block
# 11th Convolutional Layer
model_vgg16.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# 12th Convolutional Layer
model_vgg16.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# 13th Convolutional Layer
model_vgg16.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model_vgg16.add(BatchNormalization())
# Max Pooling
model_vgg16.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

# Fully Connected Layers
# Flatten
model_vgg16.add(Flatten())
# FC1
model_vgg16.add(Dense(4096, activation='relu'))
model_vgg16.add(BatchNormalization())
# Adding DropOut to prevent Overfitting
model_vgg16.add(Dropout(0.5))
#FC2
model_vgg16.add(Dense(4096, activation='relu'))
model_vgg16.add(BatchNormalization())
# Adding DropOut to prevent Overfitting
model_vgg16.add(Dropout(0.5))
# Output Layer
model_vgg16.add(Dense(1, activation='sigmoid'))
```

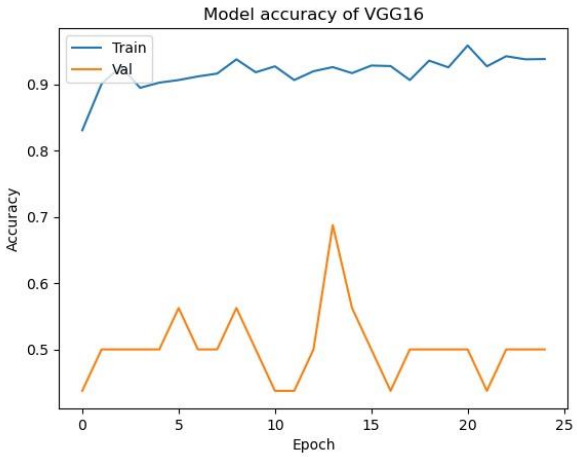
[10]:
```
# Compile the Model
from tensorflow.keras.optimizers import Adam
model_vgg16.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Training VGG16 Architecture

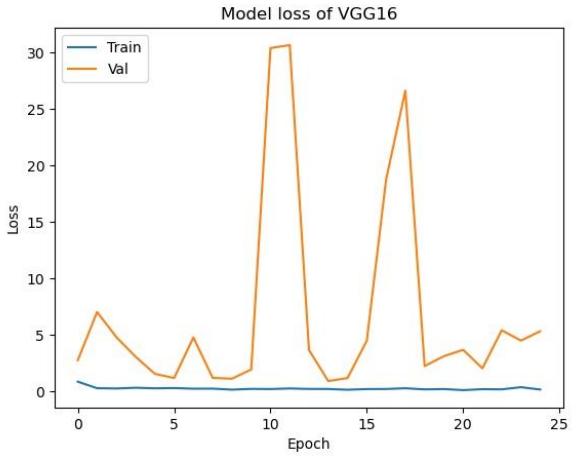
[11]:
```
history_vgg16 = model_vgg16.fit(
    train_generator,
    steps_per_epoch = 80,
    validation_data=val_generator,
    epochs=25
)
```

Performance Graph of VGG16

[13]:
```
plt.plot(history_vgg16.history['accuracy'])  # Plot training accuracy
plt.plot(history_vgg16.history['val_accuracy'])  # Plot validation accuracy
plt.title('Model accuracy of VGG16')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



[14]:
```
plt.plot(history_vgg16.history['loss'])  # Plot training loss
plt.plot(history_vgg16.history['val_loss'])  # Plot validation loss
plt.title('Model loss of VGG16')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



Model Evaluation of VGG16

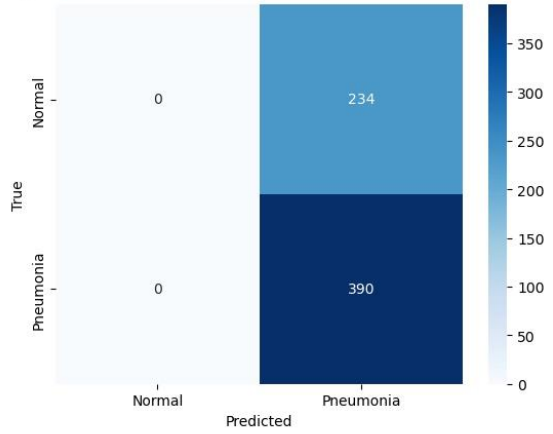
[15]:
```
# Evaluate the model on the test set
test_loss, test_acc = model_vgg16.evaluate(test_generator)
print(f"Test Accuracy: {test_acc}")
print(f"Test Loss: {test_loss}")
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 6s 286ms/step - accuracy: 0.2984 - loss: 6.5207
Test Accuracy: 0.625
Test Loss: 3.4199531078338623
```

[16]:
```
# Confusion Matrix
test_predictions = model_vgg16.predict(test_generator)
test_predictions = (test_predictions > 0.5).astype(int)

cm = confusion_matrix(test_generator.classes, test_predictions)
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=["Normal", "Pneumonia"], yticklabels=["Normal", "Pneumonia"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 301ms/step
```



[17]:
```
# Classification Report
print(classification_report(test_generator.classes, test_predictions, target_names=['NORMAL', 'PNEUMONIA']))
```

```
              precision    recall  f1-score   support

      NORMAL       0.00      0.00      0.00       234
   PNEUMONIA       0.62      1.00      0.77       390

    accuracy                           0.62       624
   macro avg       0.31      0.50      0.38       624
weighted avg       0.39      0.62      0.48       624
```

3.2 Model Development with VGG19 Pre-Trained Architecture

[18]:
```
from tensorflow.keras.applications import VGG19
```

[19]:
```
# Load the VGG19 model
vgg19_base = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

[20]:
```
# Freeze the layers except the last block
for layer in vgg19_base.layers:
    layer.trainable = False
```

[21]:
```
# Create a sequential model
model_vgg19 = Sequential()
```

[22]:
```
# Add the VGG19 base model
model_vgg19.add(vgg19_base)
model_vgg19.add(Flatten())

# Fully connected layers
model_vgg19.add(Dense(4096, activation='relu'))
model_vgg19.add(BatchNormalization())
model_vgg19.add(Dropout(0.5))
model_vgg19.add(Dense(4096, activation='relu'))
model_vgg19.add(BatchNormalization())
model_vgg19.add(Dropout(0.5))
# Output layer
model_vgg19.add(Dense(1, activation='sigmoid'))
```

[24]:
```
# Display a summary of the model vgg19 architecture
model_vgg19.summary()
```

Model: "sequential_1"

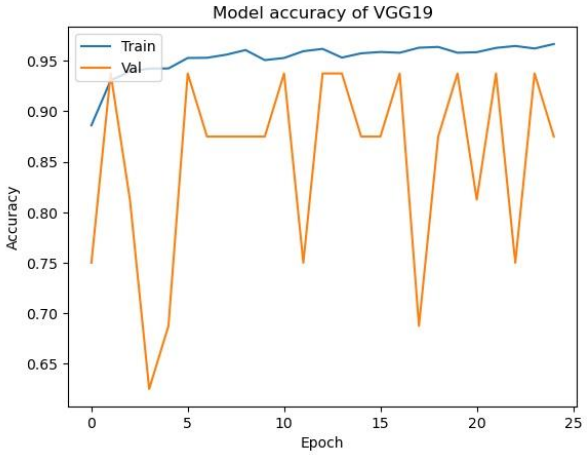
Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 7, 7, 512)	20,024,384
flatten_1 (Flatten)	(None, 25088)	0
dense_3 (Dense)	(None, 4096)	102,764,544
batch_normalization_15 (BatchNormalization)	(None, 4096)	16,384
dropout_2 (Dropout)	(None, 4096)	0
dense_4 (Dense)	(None, 4096)	16,781,312
batch_normalization_16 (BatchNormalization)	(None, 4096)	16,384
dropout_3 (Dropout)	(None, 4096)	0
dense_5 (Dense)	(None, 1)	4,097

Training VGG19 Model

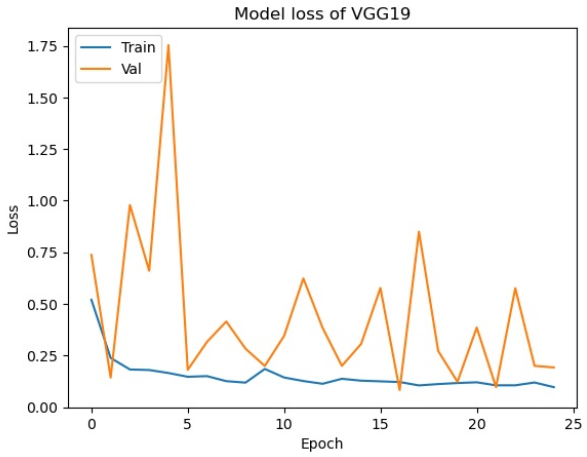
[26]:
```
# Train the model
history_vgg19 = model_vgg19.fit(
    train_generator,
    epochs=25,
    validation_data=val_generator
)
```

Performance Graph of VGG19

[27]:
```
plt.plot(history_vgg19.history['accuracy'])  # Plot training accuracy
plt.plot(history_vgg19.history['val_accuracy'])  # Plot validation accuracy
plt.title('Model accuracy of VGG19')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



[28]:
```
plt.plot(history_vgg19.history['loss'])  # Plot training loss
plt.plot(history_vgg19.history['val_loss'])  # Plot validation loss
plt.title('Model loss of VGG19')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



Model Evaluation of VGG19

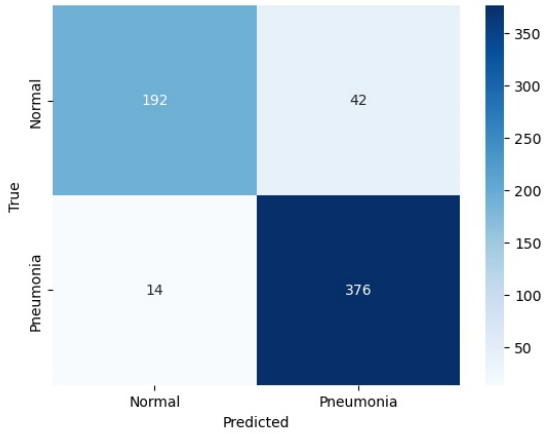
[29]:
```
# Evaluate the model on the test data
test_loss, test_acc = model_vgg19.evaluate(test_generator)
print(f"Test Accuracy: {test_acc}")
print(f"Test Loss: {test_loss}")
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 6s 298ms/step - accuracy: 0.8814 - loss: 0.4192
Test Accuracy: 0.9102563858032227
Test Loss: 0.31004223227500916
```

[30]:
```
# Confusion Matrix
test_predictions = model_vgg19.predict(test_generator)
test_predictions = (test_predictions > 0.5).astype(int)

cm = confusion_matrix(test_generator.classes, test_predictions)
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=["Normal", "Pneumonia"], yticklabels=["Normal", "Pneumonia"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 302ms/step
```



[31]:
```
# Classification Report
print(classification_report(test_generator.classes, test_predictions, target_names=['NORMAL', 'PNEUMONIA']))
```

```
              precision    recall  f1-score   support

      NORMAL       0.93      0.82      0.87       234
   PNEUMONIA       0.90      0.96      0.93       390

    accuracy                           0.91       624
   macro avg       0.92      0.89      0.90       624
weighted avg       0.91      0.91      0.91       624
```

3.3 Model Development with ResNet50 Pre-Trained Architecture

[33]:
```
from tensorflow.keras.applications import ResNet50
```

[34]:
```
# Loading the ResNet50 model
resnet_base = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

[35]:
```
# Freeze the ResNet50 base model layers
for layer in resnet_base.layers:
    layer.trainable = False
```

[36]:
```
# Create a sequential model
model_resnet50 = Sequential()
```

[37]:
```
# Add the ResNet50 base
model_resnet50.add(resnet_base)
model_resnet50.add(Flatten())
model_resnet50.add(Dense(1024, activation='relu'))
model_resnet50.add(BatchNormalization())
model_resnet50.add(Dropout(0.5))
model_resnet50.add(Dense(512, activation='relu'))
model_resnet50.add(BatchNormalization())
model_resnet50.add(Dropout(0.5))

# Output layer
model_resnet50.add(Dense(1, activation='sigmoid'))
```

[38]:
```
# Display a summary of the model resnet50 architecture
model_resnet50.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
flatten_2 (Flatten)	(None, 100352)	0
dense_6 (Dense)	(None, 1024)	102,761,472
batch_normalization_17 (BatchNormalization)	(None, 1024)	4,096
dropout_4 (Dropout)	(None, 1024)	0
dense_7 (Dense)	(None, 512)	524,800
batch_normalization_18 (BatchNormalization)	(None, 512)	2,048
dropout_5 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 1)	513

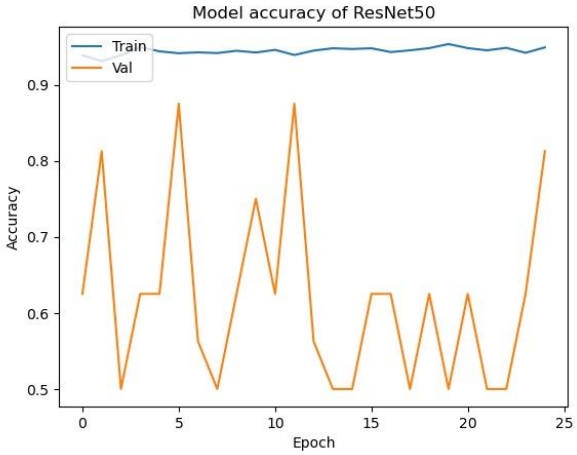
[39]:
```
# Compile the model
model_resnet50.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Training ResNet50 Model

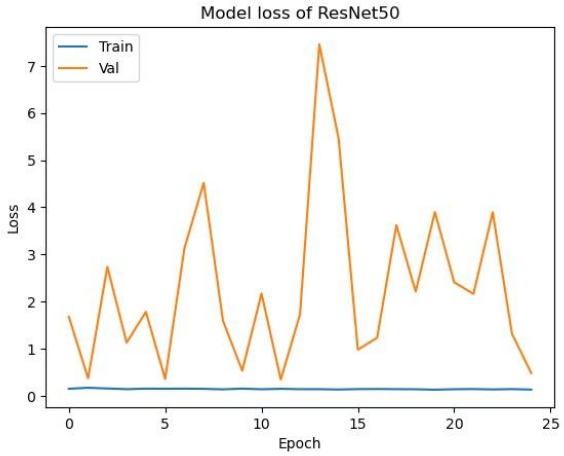
[40]:
```
# Train the model
history_resnet50 = model_resnet50.fit(
    train_generator,
    epochs=25,
    validation_data=val_generator
)
```

Performance Graph of ResNet50

[55]:
```
plt.plot(history_resnet50.history['accuracy'])  # Plot training accuracy
plt.plot(history_resnet50.history['val_accuracy'])  # Plot validation accuracy
plt.title('Model accuracy of ResNet50')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



[56]:
```
plt.plot(history_resnet50.history['loss'])  # Plot training loss
plt.plot(history_resnet50.history['val_loss'])  # Plot validation loss
plt.title('Model loss of ResNet50')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



Model Evaluation of ResNet50

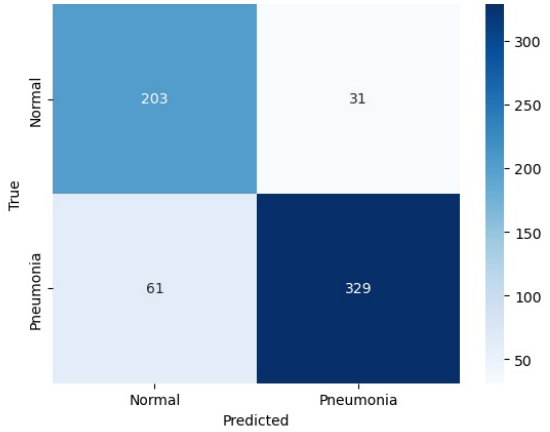
[57]:
```
# Evaluate the model on the test data
test_loss, test_acc = model_resnet50.evaluate(test_generator)
print(f"Test Accuracy: {test_acc}")
print(f"Test Loss: {test_loss}")
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 4s 196ms/step - accuracy: 0.8669 - loss: 0.4568
Test Accuracy: 0.8525640964508057
Test Loss: 0.5053150057792664
```

[58]:
```
# Confusion Matrix
test_predictions = model_resnet50.predict(test_generator)
test_predictions = (test_predictions > 0.5).astype(int)

cm = confusion_matrix(test_generator.classes, test_predictions)
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=["Normal", "Pneumonia"], yticklabels=["Normal", "Pneumonia"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 4s 188ms/step
```



[59]:
```
# Classification Report
print(classification_report(test_generator.classes, test_predictions, target_names=['NORMAL', 'PNEUMONIA']))
```

```
              precision    recall  f1-score   support

      NORMAL       0.77      0.87      0.82       234
   PNEUMONIA       0.91      0.84      0.88       390

    accuracy                           0.85       624
   macro avg       0.84      0.86      0.85       624
weighted avg       0.86      0.85      0.85       624
```

3.4 Model Development with InceptionV3 Pre-Trained Architecture

[48]:
```
from tensorflow.keras.applications import InceptionV3

# Loading the InceptionV3 model
inception_base = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

[49]:
```
# Freeze the base layers of InceptionV3
for layer in inception_base.layers:
    layer.trainable = False
```

[50]:
```
# Create a sequential model
model_inception = Sequential()
```

[51]:
```
# Add the InceptionV3 base
model_inception.add(inception_base)

model_inception.add(Flatten())
model_inception.add(Dense(1024, activation='relu'))
model_inception.add(BatchNormalization())
model_inception.add(Dropout(0.5))
model_inception.add(Dense(512, activation='relu'))
model_inception.add(BatchNormalization())
model_inception.add(Dropout(0.5))

# Output layer
model_inception.add(Dense(1, activation='sigmoid'))
```

[52]:
```
# Display a summary of the model InceptionV3 architecture
model_inception.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 5, 5, 2048)	21,802,784
flatten_3 (Flatten)	(None, 51200)	0
dense_9 (Dense)	(None, 1024)	52,429,824
batch_normalization_113 (BatchNormalization)	(None, 1024)	4,096
dropout_6 (Dropout)	(None, 1024)	0
dense_10 (Dense)	(None, 512)	524,800
batch_normalization_114 (BatchNormalization)	(None, 512)	2,048
dropout_7 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 1)	513

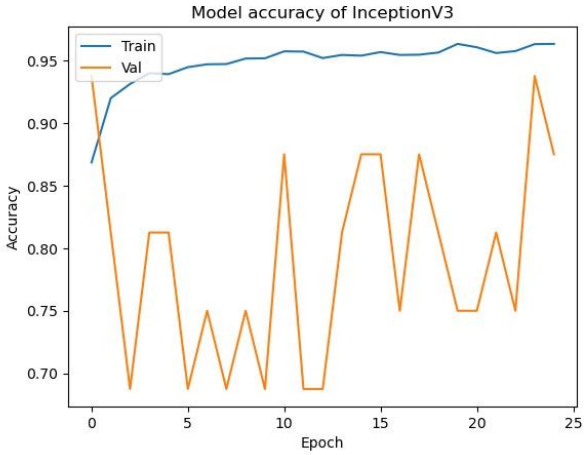
[53]:
```
# Compile the model
model_inception.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Training InceptionV3 Model

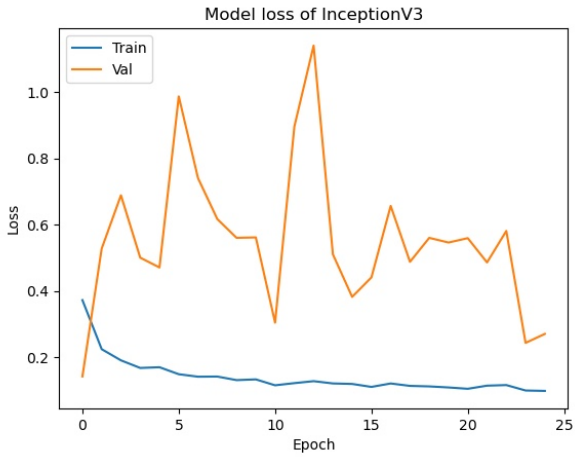
[54]:
```
# Train the model
history_inception = model_inception.fit(
    train_generator,
    epochs=25,
    validation_data=val_generator
)
```

Performance Graph of InceptionV3

[68]:
```
plt.plot(history_inception.history['accuracy'])  # Plot training accuracy
plt.plot(history_inception.history['val_accuracy'])  # Plot validation accuracy
plt.title('Model accuracy of InceptionV3')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



[69]:
```
plt.plot(history_inception.history['loss'])  # Plot training loss
plt.plot(history_inception.history['val_loss'])  # Plot validation loss
plt.title('Model loss of InceptionV3')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



Model Evaluation of InceptionV3 ¶

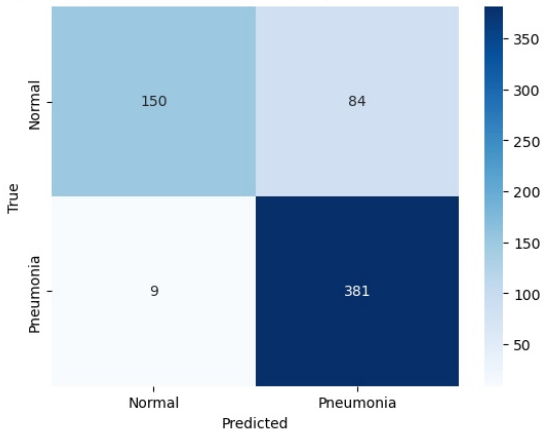
[70]:
```
# Evaluate the model on the test data
test_loss, test_acc = model_inception.evaluate(test_generator)
print(f"Test Accuracy: {test_acc}")
print(f"Test Loss: {test_loss}")
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 150ms/step - accuracy: 0.7450 - loss: 0.7237
Test Accuracy: 0.8509615659713745
Test Loss: 0.4236687421798706
```

[71]:
```
# Confusion Matrix
test_predictions = model_inception.predict(test_generator)
test_predictions = (test_predictions > 0.5).astype(int)

cm = confusion_matrix(test_generator.classes, test_predictions)
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=["Normal", "Pneumonia"], yticklabels=["Normal", "Pneumonia"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 17s 464ms/step
```



[72]:
```
# Classification Report
print(classification_report(test_generator.classes, test_predictions, target_names=['NORMAL', 'PNEUMONIA']))
```

```
              precision    recall  f1-score   support

      NORMAL       0.94      0.64      0.76       234
   PNEUMONIA       0.82      0.98      0.89       390

    accuracy                           0.85       624
   macro avg       0.88      0.81      0.83       624
weighted avg       0.87      0.85      0.84       624
```

Model Evaluation

4. Model Evaluation On Validation Data

[73]:
```
# Evaluating all the model on the validation data to select the ideal model for testing.

vgg16_acc = model_vgg16.evaluate(val_generator)[1]
vgg19_acc = model_vgg19.evaluate(val_generator)[1]
resnet50_acc = model_resnet50.evaluate(val_generator)[1]
inceptionv3_acc = model_inception.evaluate(val_generator)[1]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 460ms/step - accuracy: 0.5000 - loss: 5.3324
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 292ms/step - accuracy: 0.8750 - loss: 0.1920
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 352ms/step - accuracy: 0.8125 - loss: 0.4830
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 196ms/step - accuracy: 0.8750 - loss: 0.2699
```

[74]:
```
print(f"VGG16 Validation Accuracy: {vgg16_acc:.4f}")
print(f"VGG19 Validation Accuracy: {vgg19_acc:.4f}")
print(f"ResNet50 Validation Accuracy: {resnet50_acc:.4f}")
print(f"InceptionV3 Validation Accuracy: {inceptionv3_acc:.4f}")
```

```
VGG16 Validation Accuracy: 0.5000
VGG19 Validation Accuracy: 0.8750
ResNet50 Validation Accuracy: 0.8125
InceptionV3 Validation Accuracy: 0.8750
```

InceptionV3 shows the highest validation accuracy (0.8750), making it the most promising candidate for testing.

Testing

5. Testing

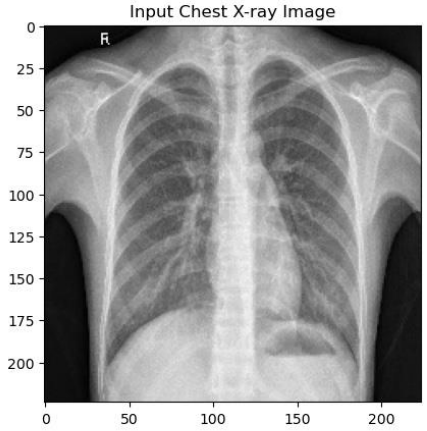
[63]:
```
# Load and preprocess the image
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import load_model
```

Normal Image

[64]:
```
image_path = r"/Users/bhushannimje/Documents/Company/Scifor Technologies/Major Project/Pneumonia Detection from Chest X-Rays Using Deep Learni
img = load_img(image_path, target_size=(224, 224))
```

[65]:
```
# Convert to array and normalize pixel values
img = img_to_array(img)
img = img / 255.0
img = np.expand_dims(img, axis=0)
```

[66]:
```
# Show the preprocessed image
plt.imshow(np.uint8(img[0] * 255))  # Convert back to [0, 255] range for display
plt.title('Input Chest X-ray Image')
plt.show()
```



[67]:
```
# Make a prediction
rslt = model_inception.predict(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 6s 6s/step
```

[68]:
```
# Interpreting the result
# Assuming the model outputs 0 for NORMAL and 1 for PNEUMONIA
if rslt[0][0] > 0.5:
    prediction = "PNEUMONIA"
else:
    prediction = "NORMAL"

# Print the prediction
print(f"Prediction: {prediction}")

# Probability score
print(f"Model Output (Probability): {rslt[0][0]}")
```

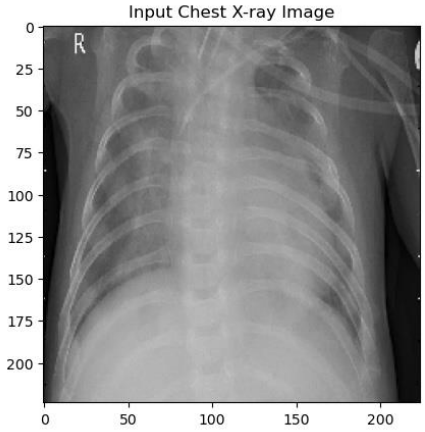
```
Prediction: NORMAL
Model Output (Probability): 0.09562698751688004
```

Pneumonia Image

[69]:
```
image_path = r"/Users/bhushannimje/Documents/Company/Scifor Technologies/Major Project/Pneumonia Detection from Chest X-Rays Using Deep Learni
img = load_img(image_path, target_size=(224, 224))
```

[70]:
```
# Convert to array and normalize pixel values
img = img_to_array(img)
img = img / 255.0
img = np.expand_dims(img, axis=0)
```

[71]:
```
# Show the preprocessed image
plt.imshow(np.uint8(img[0] * 255))  # Convert back to [0, 255] range for display
plt.title('Input Chest X-ray Image')
plt.show()
```



[72]:
```
# Make a prediction
rslt = model_inception.predict(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 409ms/step
```

[73]:
```
# Interpreting the result
# Assuming the model outputs 0 for NORMAL and 1 for PNEUMONIA
if rslt[0][0] > 0.5:
    prediction = "PNEUMONIA"
else:
    prediction = "NORMAL"

# Print the prediction
print(f"Prediction: {prediction}")

# Probability score
print(f"Model Output (Probability): {rslt[0][0]}")
```

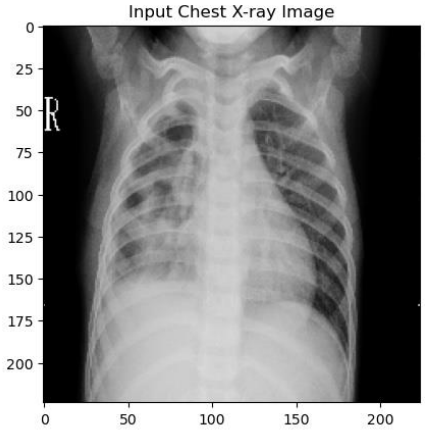
```
Prediction: PNEUMONIA
Model Output (Probability): 0.9999631643295288
```

RESULT

The InceptionV3 model demonstrate impressive performance with an overall validation accuracy of 87.50 % reflecting its strong ability to correctly predict the images. Precision and recall metrics indicate that the model excels in distinguishing between the two classes: “Normal” images with a precision of 0.94 and recall of 0.64 and “Pneumonia” images with the precision of 0.82 and recall of 0.98. The high F1-Score of 0.89 for “Pneumonia” and 0.76 for “Normal”.

The model effectively classifies images into “Normal” and “Pneumonia” categories demonstrating strong performance in distinguishing between these two medical conditions.

It correctly identified “Pneumonia” and “Normal” images, validating it robustness in handling Pneumonia Detection task.



[79]:
```
# Make a prediction
rslt = model_inception.predict(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 195ms/step
```

[80]:
```
# Interpreting the result
# Assuming the model outputs 0 for NORMAL and 1 for PNEUMONIA
if rslt[0][0] > 0.5:
    prediction = "PNEUMONIA"
else:
    prediction = "NORMAL"

# Print the prediction
print(f"Prediction: {prediction}")

# Probability score
print(f"Model Output (Probability): {rslt[0][0]}")
```

```
Prediction: PNEUMONIA
Model Output (Probability): 0.9221228957176208
```

CONCLUSION

In this project we successfully implemented a CNN to classify images into two categories: “Normal” and “Pneumonia”. The model was trained and evaluated on the dataset that was augmented and normalized to enhance performance.

The result demonstrated that the model effectively learned to distinguished between the two classes, achieving satisfactory accuracy.