# Install necessary libraries

!pip install tensorflow keras numpy matplotlib

# Import necessary libraries

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

import os

import zipfile

!pip install tensorflow keras kaggle

# Import necessary libraries

import os

import zipfile

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Set up Kaggle environment

!mkdir -p ~/.kaggle

!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json

# Download dataset from Kaggle

!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia

# Unzip the dataset

with zipfile.ZipFile("chest-xray-pneumonia.zip", 'r') as zip\_ref:

    zip\_ref.extractall("chest\_xray")

# Path to dataset

base\_dir = 'chest\_xray/chest\_xray'

train\_dir = os.path.join(base\_dir, 'train')

test\_dir = os.path.join(base\_dir, 'test')

# Set image size and batch size

IMAGE\_SIZE = (150, 150)

BATCH\_SIZE = 32

# Data Augmentation and Preprocessing for Training and Validation Sets

train\_datagen = ImageDataGenerator(

    rescale=1./255,             # Normalize pixel values

    rotation\_range=20,          # Random rotations

    width\_shift\_range=0.2,      # Horizontal shift

    height\_shift\_range=0.2,     # Vertical shift

    shear\_range=0.2,            # Shear transformation

    zoom\_range=0.2,             # Random zoom

    horizontal\_flip=True,       # Flip images horizontally

    fill\_mode='nearest',        # Fill in new pixels after transformations

    validation\_split=0.2        # Split the training data into train/validation sets

)

# Train Generator

train\_generator = train\_datagen.flow\_from\_directory(

    train\_dir,

    target\_size=IMAGE\_SIZE,

    batch\_size=BATCH\_SIZE,

    class\_mode='binary',

    subset='training'  # Use 80% for training

)

# Validation Generator

validation\_generator = train\_datagen.flow\_from\_directory(

    train\_dir,

    target\_size=IMAGE\_SIZE,

    batch\_size=BATCH\_SIZE,

    class\_mode='binary',

    subset='validation'  # Use 20% for validation

)

# Test Data Generator (No augmentation for test set, only rescaling)

test\_datagen = ImageDataGenerator(rescale=1./255)

test\_generator = test\_datagen.flow\_from\_directory(

    test\_dir,

    target\_size=IMAGE\_SIZE,

    batch\_size=BATCH\_SIZE,

    class\_mode='binary'

)

!pip install tensorflow keras

# Initialize the CNN model

model = Sequential()

# First Convolutional Layer

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(150, 150, 3)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

# Second Convolutional Layer

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

# Third Convolutional Layer

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

# Flattening the layers to connect to Dense layers

model.add(Flatten())

# Fully Connected Layer

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))  # Dropout to prevent overfitting

model.add(Dense(1, activation='sigmoid'))  # Output layer for binary classification

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Summary of the model

model.summary()

# Set image size and batch size

IMAGE\_SIZE = (150, 150)

BATCH\_SIZE = 32

# Data Augmentation and Preprocessing for Training and Validation Sets

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,

    fill\_mode='nearest',

    validation\_split=0.2

)

# Train Generator

train\_generator = train\_datagen.flow\_from\_directory(

    train\_dir,

    target\_size=IMAGE\_SIZE,

    batch\_size=BATCH\_SIZE,

    class\_mode='binary',

    subset='training'

)

# Validation Generator

validation\_generator = train\_datagen.flow\_from\_directory(

    train\_dir,

    target\_size=IMAGE\_SIZE,

    batch\_size=BATCH\_SIZE,

    class\_mode='binary',

    subset='validation'

)

# Test Data Generator (No augmentation for test set, only rescaling)

test\_datagen = ImageDataGenerator(rescale=1./255)

test\_generator = test\_datagen.flow\_from\_directory(

    test\_dir,

    target\_size=IMAGE\_SIZE,

    batch\_size=BATCH\_SIZE,

    class\_mode='binary'

)

import tensorflow as tf

from tensorflow.keras import layers, models

# Initialize the model

model = models.Sequential()

# First Convolutional Layer

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(150, 150, 3)))

model.add(layers.MaxPooling2D(pool\_size=(2, 2)))

# Second Convolutional Layer

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D(pool\_size=(2, 2)))

# Third Convolutional Layer

model.add(layers.Conv2D(128, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D(pool\_size=(2, 2)))

# Flatten the layers to connect to Dense layers

model.add(layers.Flatten())

# Fully Connected Layer

model.add(layers.Dense(128, activation='relu'))

model.add(layers.Dropout(0.5))  # Dropout to prevent overfitting

model.add(layers.Dense(1, activation='sigmoid'))  # Output layer for binary classification

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Set up data augmentation for training and validation sets

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,

    fill\_mode='nearest',

    validation\_split=0.2

)

# Train Generator

train\_generator = train\_datagen.flow\_from\_directory(

    train\_dir,

    target\_size=(150, 150),

    batch\_size=32,

    class\_mode='binary',

    subset='training'

)

# Validation Generator

validation\_generator = train\_datagen.flow\_from\_directory(

    train\_dir,

    target\_size=(150, 150),

    batch\_size=32,

    class\_mode='binary',

    subset='validation'

)

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

# Set up early stopping

early\_stopping = EarlyStopping(

    monitor='val\_loss',  # Monitor validation loss

    patience=5,          # Wait for 5 epochs before stopping

    restore\_best\_weights=True  # Restore the best weights after stopping

)

# Train the model

history = model.fit(

    train\_generator,

    steps\_per\_epoch=train\_generator.samples // 32,

    validation\_data=validation\_generator,

    validation\_steps=validation\_generator.samples // 32,

    epochs=50,  # Set a high number of epochs to allow early stopping

    callbacks=[early\_stopping]

)

# Evaluate the model on the test dataset

test\_loss, test\_accuracy = model.evaluate(validation\_generator)

# Predictions on the test set

import numpy as np

from sklearn.metrics import classification\_report, confusion\_matrix

# Generate predictions

predictions = model.predict(validation\_generator)

predicted\_classes = np.where(predictions > 0.5, 1, 0)  # Convert probabilities to binary predictions

# True labels

true\_classes = validation\_generator.classes

# Classification report for precision, recall, and F1 score

report = classification\_report(true\_classes, predicted\_classes, target\_names=['Normal', 'Pneumonia'])

print(report)