from google.colab import drive

drive.mount('/content/drive')

# Import Required Libraries

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Conv2D, Dense, Flatten, Lambda, Input, MaxPooling2D, Concatenate

from tensorflow.keras.models import Model

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_curve, roc\_auc\_score

import seaborn as sns

data\_dir = '/content/drive/MyDrive/cyclone\_DataSet'

train\_dir = data\_dir + '/train'

val\_dir = data\_dir + '/val'

test\_dir = data\_dir + '/test'

# Data Augmentation and Rescaling

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

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

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

# Rescaling and resizing to 150x150

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

    train\_dir,

    target\_size=(150, 150),  # Ensure this matches the model's input size

    batch\_size=32,

    class\_mode='binary'

)

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

    val\_dir,

    target\_size=(150, 150),  # Ensure this matches the model's input size

    batch\_size=32,

    class\_mode='binary'

)

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

    test\_dir,

    target\_size=(150, 150),  # Ensure this matches the model's input size

    batch\_size=32,

    class\_mode='binary',

    shuffle=False

)

import tensorflow as tf

from tensorflow.keras.layers import Input, Conv2D, Lambda, MaxPooling2D, Flatten, Concatenate, Dense, GlobalAveragePooling2D, GlobalMaxPooling2D, Dropout

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

# Define the sum pooling function

def sum\_pool(x):

    return tf.reduce\_sum(x, axis=(1, 2))

input\_layer = Input(shape=(150, 150, 3))

# First convolutional layer with 32 filters, kernel size of 3x3, and ReLU activation

conv\_layer1 = Conv2D(32, (3, 3), activation='relu')(input\_layer)

# Sum Pooling after the first convolutional layer

pool\_layer1 = Lambda(sum\_pool)(conv\_layer1)

# Second convolutional layer with 64 filters, kernel size of 3x3, and ReLU activation

conv\_layer2 = Conv2D(64, (3, 3), activation='relu')(input\_layer)

# MaxPooling layer after the second convolutional layer

pool\_layer2 = MaxPooling2D(pool\_size=(2, 2))(conv\_layer2)

# Flattening the pooled features from the first two layers

flat\_layer1 = Flatten()(pool\_layer1)

flat\_layer2 = Flatten()(pool\_layer2)

# Concatenating all the features: flatten layers and the pooled layers

concat\_layer = Concatenate()([flat\_layer1, flat\_layer2])

# Adding Dense and Dropout layers

x = Dense(512, activation='relu')(concat\_layer)

x = Dropout(0.4)(x)

x = Dense(256, activation='relu')(x)

x = Dropout(0.3)(x)

x = Dense(128, activation='relu')(x)

x = Dropout(0.2)(x)

# Fully connected output layer with sigmoid activation

output\_layer = Dense(1, activation='sigmoid')(x)

# Create the model

model = Model(inputs=input\_layer, outputs=output\_layer)

# Define the optimizer with a specific learning rate

learning\_rate = 0.001  # Set the learning rate

optimizer = Adam(learning\_rate=learning\_rate)

# Compile the model with the optimizer and loss function

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

# Show the model summary

model.summary()

history = model.fit(

    train\_generator,

    epochs=20,

    validation\_data=val\_generator

)

# Test accuracy

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

print(f'Test Accuracy: {test\_accuracy:.4f}')

# Train and Validation accuracy

train\_accuracy = history.history['accuracy'][-1]

val\_accuracy = history.history['val\_accuracy'][-1]

print(f'Train Accuracy: {train\_accuracy:.4f}')

print(f'Validation Accuracy: {val\_accuracy:.4f}')

# Plotting accuracy and loss

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Import necessary libraries for ROC and AUC

from sklearn.metrics import roc\_curve, roc\_auc\_score

# Get true labels and predictions

test\_generator.reset()

predictions = model.predict(test\_generator)

# Convert predictions to binary outcomes

predicted\_classes = (predictions > 0.5).astype(int)

true\_classes = test\_generator.classes

# Classification Report

report = classification\_report(true\_classes, predicted\_classes, target\_names=['Cyclone', 'No Cyclone'], output\_dict=True)

print('Classification Report:')

print(classification\_report(true\_classes, predicted\_classes, target\_names=['Cyclone', 'No Cyclone']))

# Precision, Recall, F1 Score

precision = report['Cyclone']['precision']

recall = report['Cyclone']['recall']

f1\_score = report['Cyclone']['f1-score']

print(f'Precision: {precision:.4f}')

print(f'Recall: {recall:.4f}')

print(f'F1 Score: {f1\_score:.4f}')

# Calculate ROC AUC

auc = roc\_auc\_score(true\_classes, predictions)

print(f'AUC: {auc:.4f}')

# Plot ROC curve

fpr, tpr, thresholds = roc\_curve(true\_classes, predictions)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', label=f'AUC = {auc:.4f}')

plt.plot([0, 1], [0, 1], color='red', linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend()

plt.show()

# Confusion Matrix

conf\_matrix = confusion\_matrix(true\_classes, predicted\_classes)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Cyclone', 'No Cyclone'], yticklabels=['Cyclone', 'No Cyclone'])

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()