**Step 1: Install Required Libraries**

!pip install tensorflow keras opencv-python numpy matplotlib scikit-learn

**Step 2: Import Libraries**

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import VGG19

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Dense, Flatten, Dropout

from tensorflow.keras.optimizers import Adam

import numpy as np

import cv2

import os

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

**Step 3: Load and Preprocess Dataset**

def load\_images(data\_dir, img\_size=(224, 224)):

images, labels = [], []

classes = os.listdir(data\_dir)

for label, fish\_type in enumerate(classes):

fish\_path = os.path.join(data\_dir, fish\_type)

for img\_file in os.listdir(fish\_path):

img\_path = os.path.join(fish\_path, img\_file)

img = cv2.imread(img\_path)

img = cv2.resize(img, img\_size)

images.append(img)

labels.append(label)

images = np.array(images) / 255.0 # Normalize the images

labels = np.array(labels)

return images, labels, classes

data\_dir = 'path\_to\_fish\_dataset' # Update with actual path

images, labels, class\_names = load\_images(data\_dir)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(images, labels, test\_size=0.2, random\_state=42)

**Step 4: Build the CNN Model**

def build\_cnn\_model(input\_shape, num\_classes):

base\_model = VGG19(weights='imagenet', include\_top=False, input\_shape=input\_shape)

for layer in base\_model.layers:

layer.trainable = False

x = Flatten()(base\_model.output)

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

x = Dropout(0.5)(x)

output = Dense(num\_classes, activation='softmax')(x)

model = Model(inputs=base\_model.input, outputs=output)

model.compile(optimizer=Adam(learning\_rate=0.0001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

return model

model = build\_cnn\_model((224, 224, 3), len(class\_names))

model.summary()

**Step 5: Train the Model**

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, y\_test))

**Step 6: Evaluate the Model**

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print(f"Test Accuracy: {test\_acc:.2f}")

**Step 7: Optimize with Chaotic Oppositional Based Whale Optimization Algorithm (CO-WOA)**

(Implementing CO-WOA for hyperparameter tuning)

# Placeholder function for CO-WOA optimization

def chaotic\_oppositional\_woa(model, param\_grid, X\_train, y\_train):

best\_params = {'learning\_rate': 0.0001, 'batch\_size': 32} # Example best params

print("Best hyperparameters found:", best\_params)

return best\_params

param\_grid = {'learning\_rate': [0.001, 0.0001], 'batch\_size': [16, 32]}

best\_params = chaotic\_oppositional\_woa(model, param\_grid, X\_train, y\_train)

**Step 8: Retrain Model with Optimized Parameters**

model.compile(optimizer=Adam(learning\_rate=best\_params['learning\_rate']),

loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=15, batch\_size=best\_params['batch\_size'], validation\_data=(X\_test, y\_test))

**Step 9: Visualize Model Performance**

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

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

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Model Accuracy')

plt.show()

**Step 10: Save the Model**

model.save('fish\_species\_classifier.h5')

print("Model saved successfully.")