In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG19
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
import os
import random
import shutil
```

In [50]:
```
# Image and batch size
image_size = 224
batch_size = 16

# Path to the main directory containing the 12 folders
main_dir = r"C:\Users\himan\Desktop\training_set\training_set"

# Target number of images per folder
target_images = 500

# Ensure each folder has only 500 images
for folder_name in os.listdir(main_dir):
    folder_path = os.path.join(main_dir, folder_name)

    # Ensure it's a directory (skip any files in the main directory)
    if os.path.isdir(folder_path):
        images = os.listdir(folder_path)

        # Check if the folder has more than the target number of images
        if len(images) > target_images:
            # Randomly select images to keep
            images_to_keep = random.sample(images, target_images)

            # Remove images not in the selection
            for image in images:
                if image not in images_to_keep:
                    os.remove(os.path.join(folder_path, image))

print("Reduction to 500 images per folder completed.")

# Now, load the dataset
train_ds = tf.keras.utils.image_dataset_from_directory(
  main_dir,
  validation_split=0.3,
  subset="training",
  seed=123,
  image_size=(image_size, image_size),
  batch_size=batch_size
)
```

```
val_ds = tf.keras.utils.image_dataset_from_directory(
  main_dir,
  validation_split=0.3,
  subset="validation",
  seed=123,
  image_size=(image_size, image_size),
  batch_size=batch_size
)
```

```
Reduction to 500 images per folder completed.
Found 5000 files belonging to 10 classes.
Using 3500 files for training.
Found 5000 files belonging to 10 classes.
Using 1500 files for validation.
```

In [51]:
```
class_names = train_ds.class_names
```

In [52]:
```
val_batches = tf.data.experimental.cardinality(val_ds)
test_ds = val_ds.take(val_batches // 2)
val_ds = val_ds.skip(val_batches // 2)
```

Configuring dataset for better performance

In [57]:
```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.prefetch(buffer_size=AUTOTUNE)
```

Data Augmentation layer

In [60]:
```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])
```

Functions to return some information about the data

In [63]:
```
def main(train_data_path):
    """
    return training and testing dataframes including 2 columns image path and its l
    Args:
      train_data_path (string): string includes your train data path
      test_data_path (string): string includes your test data path
      note: data_path argument must contains your classes dirs
    Returns:
      train_df (dataframe):  It consists of two columns (the path of the image and
      test_df (dataframe):  It consists of two columns (the path of the image and i

    """
    for dirs in [train_data_path]:

        filepaths = []
        labels = []
```

```
        classes_list = sorted(os.listdir(dirs))
        filepaths,labels = get_fileList(classes_list, dirs, filepaths, labels)
        train_df = obtain_train_df(filepaths, labels)
    return train_df
def get_fileList(classes_list, dirs, filepaths, labels):

    """
    return file paths list including all full image paths and label for each one
    Args:
      classes_list (list): list includes name of classes dirs
      dirs (string): the data path
      filepaths (list): empty list to fill with images paths later
      labels (list): empty list to fill with each image label later
    Returns:
      count (scalar):  number of images in your dataset
    """
    for s_class in classes_list:

        label = s_class
        classpath = os.path.join(dirs, s_class)
        flist = sorted(os.listdir(classpath))
        store_image_names_and_its_labels(flist, classpath, filepaths, labels, label
    return filepaths, labels
def store_image_names_and_its_labels(flist, classpath, filepaths, labels, label):

    """
    return nothing, it fills the filepaths list with all the images paths and its l
    Args:
      flist (list) : contains all the image names in certain class
      classes_list (list): list includes name of classes dirs
      filepaths (list): stores all the images paths
      labels (list): stores the label of each image in the dataset
      label (string): label for a specific image
    Returns:
      nothing
    """
    for file in flist:
        file_path = os.path.join(classpath, file)
        filepaths.append(file_path)
        labels.append(label)
def obtain_train_df(filepaths, labels):
    """
    Return Dataframe with 2 columns contains each image path with its label
    Args:
      filepaths (list): stores all the images paths
      labels (list): stores the label of each image in the dataset

    Returns:
      train_df (DataFrame):  Dataframe with 2 columns contains each image path with
    """
    train_df =pd.DataFrame({
        'image_path' : filepaths,
        'label' : labels
    })
    return train_df
```

EDA

In [66]:
```
data_df = main(main_dir)
```

In [67]:
```
data_df.head()
```

Out[67]:

	image_path	label
0	C:\Users\himan\Desktop\training_set\training_s...	Cattle Egret
1	C:\Users\himan\Desktop\training_set\training_s...	Cattle Egret
2	C:\Users\himan\Desktop\training_set\training_s...	Cattle Egret
3	C:\Users\himan\Desktop\training_set\training_s...	Cattle Egret
4	C:\Users\himan\Desktop\training_set\training_s...	Cattle Egret

In [69]:
```
num_classes = len(data_df['label'].unique())
print(f"We have {num_classes} classes")
print(f"We have {data_df.shape[0]} images")
```

```
We have 10 classes
We have 5000 images
```

In [71]:
```
data_df['label'].value_counts()
```

Out[71]:
```
label
Cattle Egret                 500
Common Kingfisher            500
Gray Wagtail                 500
Northern Lapwing             500
Red-Wattled Lapwing          500
Ruddy Shelduck               500
Sarus Crane                  500
White Wagtail                500
White-Breasted Kingfisher    500
White-Breasted Waterhen      500
Name: count, dtype: int64
```

Data is perfectly balanced!!

In [75]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
test_split_size = 0.3
train_df, test_df=train_test_split(data_df, test_size=test_split_size, shuffle=True
```

In [76]:
```
train_df.head()
```

Out[76]:

	image_path	label
2853	C:\Users\himan\Desktop\training_set\training_s...	Ruddy Shelduck
4202	C:\Users\himan\Desktop\training_set\training_s...	White-Breasted Kingfisher
1118	C:\Users\himan\Desktop\training_set\training_s...	Gray Wagtail
2325	C:\Users\himan\Desktop\training_set\training_s...	Red-Wattled Lapwing
3689	C:\Users\himan\Desktop\training_set\training_s...	White Wagtail

In [77]:
```
train_df.shape
```

Out[77]:
```
(3500, 2)
```

In [78]:
```
test_df.head()
```

Out[78]:

	image_path	label
1818	C:\Users\himan\Desktop\training_set\training_s...	Northern Lapwing
3121	C:\Users\himan\Desktop\training_set\training_s...	Sarus Crane
2417	C:\Users\himan\Desktop\training_set\training_s...	Red-Wattled Lapwing
4590	C:\Users\himan\Desktop\training_set\training_s...	White-Breasted Waterhen
3344	C:\Users\himan\Desktop\training_set\training_s...	Sarus Crane

In [79]:
```
test_df.shape
```

Out[79]:
```
(1500, 2)
```

In [82]:
```
print(f"We have {len(train_df)} training images and {len(test_df)} testing images")
```

```
We have 3500 training images and 1500 testing images
```

In [87]:
```
train_df['label'].value_counts()
```

Out[87]:
```
label
Ruddy Shelduck               350
White-Breasted Kingfisher    350
Gray Wagtail                 350
Red-Wattled Lapwing          350
White Wagtail                350
Northern Lapwing             350
Sarus Crane                  350
Common Kingfisher            350
Cattle Egret                 350
White-Breasted Waterhen      350
Name: count, dtype: int64
```

In [89]:
```
test_df['label'].value_counts()
```

Out[89]:
```
label
Northern Lapwing             150
Sarus Crane                  150
Red-Wattled Lapwing          150
White-Breasted Waterhen      150
White Wagtail                150
Cattle Egret                 150
Ruddy Shelduck               150
Common Kingfisher            150
White-Breasted Kingfisher    150
Gray Wagtail                 150
Name: count, dtype: int64
```

Importing the base model (VGG16)

In [92]:
```
base_model = tf.keras.applications.VGG16(input_shape=(224,224,3),
                                         include_top=False,
                                         weights='imagenet')
base_model.trainable = False
```

Creating the model architecture

In [101…
```
model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./127.5, offset=-1),
    data_augmentation,
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(256,activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32,activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(len(class_names))
])
```

Compiling the model

In [104…
```
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Training the model

Early Stopping to stop training when validation accuracy has stopped improving Saving weights on all iterations Saving weights if they produce the best validaiton accuracy Saving the trained model Initial number of epochs = 60

In [109…
```
es = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=4, min_delta

# Correcting the file extension for the checkpoint
```

```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="training_weights/checkpoints/cp-{
                                        save_weights_only=True,
                                        verbose=1)

# Fixing the typo in 'monitor'
best = tf.keras.callbacks.ModelCheckpoint(filepath="training_weights/best/model_bes
                                          save_weights_only=True,
                                          monitor="val_accuracy",
                                          mode="max",
                                          save_best_only=True)

history = model.fit(train_ds,
                    epochs=60,
                    validation_data=val_ds,
                    callbacks=[es, cp, best])

model.save('training_weights/models/vgg16_modified.h5')
```

```
Epoch 1/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.1370 - loss: 2.2605
Epoch 1: saving model to training_weights/checkpoints/cp-1.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 492s 2s/step - accuracy: 0.1371 - loss: 2.2601 - val_ac
curacy: 0.3222 - val_loss: 1.8509
Epoch 2/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.2756 - loss: 1.9117
Epoch 2: saving model to training_weights/checkpoints/cp-2.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 563s 3s/step - accuracy: 0.2758 - loss: 1.9112 - val_ac
curacy: 0.4545 - val_loss: 1.3648
Epoch 3/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.4081 - loss: 1.5805
Epoch 3: saving model to training_weights/checkpoints/cp-3.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 570s 3s/step - accuracy: 0.4082 - loss: 1.5803 - val_ac
curacy: 0.5735 - val_loss: 1.1383
Epoch 4/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.4761 - loss: 1.3906
Epoch 4: saving model to training_weights/checkpoints/cp-4.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 574s 3s/step - accuracy: 0.4762 - loss: 1.3904 - val_ac
curacy: 0.5976 - val_loss: 1.0902
Epoch 5/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.5489 - loss: 1.2666
Epoch 5: saving model to training_weights/checkpoints/cp-5.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 555s 3s/step - accuracy: 0.5490 - loss: 1.2665 - val_ac
curacy: 0.5749 - val_loss: 1.1555
Epoch 6/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.5967 - loss: 1.1351
Epoch 6: saving model to training_weights/checkpoints/cp-6.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 545s 2s/step - accuracy: 0.5968 - loss: 1.1350 - val_ac
curacy: 0.7313 - val_loss: 0.7928
Epoch 7/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.6382 - loss: 1.0599
Epoch 7: saving model to training_weights/checkpoints/cp-7.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 547s 2s/step - accuracy: 0.6382 - loss: 1.0599 - val_ac
curacy: 0.7340 - val_loss: 0.7795
Epoch 8/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.6679 - loss: 0.9842
Epoch 8: saving model to training_weights/checkpoints/cp-8.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 557s 3s/step - accuracy: 0.6679 - loss: 0.9842 - val_ac
curacy: 0.7607 - val_loss: 0.7424
Epoch 9/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.6773 - loss: 0.9436
Epoch 9: saving model to training_weights/checkpoints/cp-9.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 565s 3s/step - accuracy: 0.6774 - loss: 0.9435 - val_ac
curacy: 0.7834 - val_loss: 0.6512
Epoch 10/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.6975 - loss: 0.9211
Epoch 10: saving model to training_weights/checkpoints/cp-10.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 548s 3s/step - accuracy: 0.6975 - loss: 0.9210 - val_ac
curacy: 0.7941 - val_loss: 0.6350
Epoch 11/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7299 - loss: 0.8276
Epoch 11: saving model to training_weights/checkpoints/cp-11.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 529s 2s/step - accuracy: 0.7299 - loss: 0.8276 - val_ac
curacy: 0.8021 - val_loss: 0.6243
Epoch 12/60
```

```
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7417 - loss: 0.7933
Epoch 12: saving model to training_weights/checkpoints/cp-12.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 537s 2s/step - accuracy: 0.7417 - loss: 0.7934 - val_ac
curacy: 0.8262 - val_loss: 0.5682
Epoch 13/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7370 - loss: 0.8225
Epoch 13: saving model to training_weights/checkpoints/cp-13.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 536s 2s/step - accuracy: 0.7370 - loss: 0.8224 - val_ac
curacy: 0.8209 - val_loss: 0.6122
Epoch 14/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7648 - loss: 0.7496
Epoch 14: saving model to training_weights/checkpoints/cp-14.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 543s 2s/step - accuracy: 0.7648 - loss: 0.7497 - val_ac
curacy: 0.8302 - val_loss: 0.5777
Epoch 15/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7554 - loss: 0.7552
Epoch 15: saving model to training_weights/checkpoints/cp-15.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 547s 2s/step - accuracy: 0.7554 - loss: 0.7553 - val_ac
curacy: 0.8329 - val_loss: 0.5595
Epoch 16/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7675 - loss: 0.7129
Epoch 16: saving model to training_weights/checkpoints/cp-16.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 538s 2s/step - accuracy: 0.7674 - loss: 0.7130 - val_ac
curacy: 0.7901 - val_loss: 0.6610
Epoch 17/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7671 - loss: 0.7079
Epoch 17: saving model to training_weights/checkpoints/cp-17.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 536s 2s/step - accuracy: 0.7671 - loss: 0.7079 - val_ac
curacy: 0.8222 - val_loss: 0.5976
Epoch 18/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7845 - loss: 0.6748
Epoch 18: saving model to training_weights/checkpoints/cp-18.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 476s 2s/step - accuracy: 0.7845 - loss: 0.6748 - val_ac
curacy: 0.8088 - val_loss: 0.6001
Epoch 19/60
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.7867 - loss: 0.6608
Epoch 19: saving model to training_weights/checkpoints/cp-19.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 534s 2s/step - accuracy: 0.7867 - loss: 0.6609 - val_ac
curacy: 0.8182 - val_loss: 0.6122
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.
saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.savin
g.save_model(model, 'my_model.keras')`.
```

Fine-tuning the trained model

In [112…

```
base_model.trainable = True
for layer in base_model.layers[:12]:
    layer.trainable = False
```

Compiling the model again

In [115…

```
model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Training again for fine-tuning

In [120…

```
# EarlyStopping callback
es = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy', patience=4, min_delta=0.001, restore_best_weights=True
)

# ModelCheckpoint for saving checkpoints during training
cp = tf.keras.callbacks.ModelCheckpoint(
    filepath="ft_weights/checkpoints/cp-{epoch}.weights.h5",  # Updated extension
    save_weights_only=True,
    verbose=1
)

# ModelCheckpoint for saving the best model based on validation accuracy
best = tf.keras.callbacks.ModelCheckpoint(
    filepath="ft_weights/best/model_best_val_loss.weights.h5",  # Updated extension
    save_weights_only=True,
    monitor="val_accuracy",  # Corrected typo in "monitor"
    mode="max",
    save_best_only=True
)

# Fitting the model
fhistory = model.fit(
    train_ds,
    epochs=history.epoch[-1] + 40,
    validation_data=val_ds,
    initial_epoch=history.epoch[-1],
    callbacks=[es, cp, best]
)
```

```
Epoch 19/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.7957 - loss: 0.6573
Epoch 19: saving model to ft_weights/checkpoints/cp-19.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 674s 3s/step - accuracy: 0.7958 - loss: 0.6570 - val_ac
curacy: 0.8850 - val_loss: 0.4559
Epoch 20/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.8675 - loss: 0.4171
Epoch 20: saving model to ft_weights/checkpoints/cp-20.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 779s 4s/step - accuracy: 0.8675 - loss: 0.4172 - val_ac
curacy: 0.8783 - val_loss: 0.4499
Epoch 21/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.8763 - loss: 0.3917
Epoch 21: saving model to ft_weights/checkpoints/cp-21.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 776s 4s/step - accuracy: 0.8764 - loss: 0.3917 - val_ac
curacy: 0.8824 - val_loss: 0.4582
Epoch 22/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9071 - loss: 0.3198
Epoch 22: saving model to ft_weights/checkpoints/cp-22.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 908s 4s/step - accuracy: 0.9071 - loss: 0.3198 - val_ac
curacy: 0.9011 - val_loss: 0.3748
Epoch 23/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - accuracy: 0.9159 - loss: 0.2795
Epoch 23: saving model to ft_weights/checkpoints/cp-23.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 1322s 6s/step - accuracy: 0.9159 - loss: 0.2796 - val_a
ccuracy: 0.9131 - val_loss: 0.3661
Epoch 24/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9165 - loss: 0.2488
Epoch 24: saving model to ft_weights/checkpoints/cp-24.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 794s 4s/step - accuracy: 0.9165 - loss: 0.2489 - val_ac
curacy: 0.8957 - val_loss: 0.4187
Epoch 25/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9254 - loss: 0.2406
Epoch 25: saving model to ft_weights/checkpoints/cp-25.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 819s 4s/step - accuracy: 0.9254 - loss: 0.2405 - val_ac
curacy: 0.9211 - val_loss: 0.3055
Epoch 26/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9404 - loss: 0.2098
Epoch 26: saving model to ft_weights/checkpoints/cp-26.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 815s 4s/step - accuracy: 0.9404 - loss: 0.2099 - val_ac
curacy: 0.9158 - val_loss: 0.3013
Epoch 27/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9423 - loss: 0.1946
Epoch 27: saving model to ft_weights/checkpoints/cp-27.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 810s 4s/step - accuracy: 0.9423 - loss: 0.1947 - val_ac
curacy: 0.9131 - val_loss: 0.3259
Epoch 28/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9480 - loss: 0.1850
Epoch 28: saving model to ft_weights/checkpoints/cp-28.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 815s 4s/step - accuracy: 0.9480 - loss: 0.1850 - val_ac
curacy: 0.9305 - val_loss: 0.2851
Epoch 29/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9538 - loss: 0.1623
Epoch 29: saving model to ft_weights/checkpoints/cp-29.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 800s 4s/step - accuracy: 0.9538 - loss: 0.1623 - val_ac
curacy: 0.9091 - val_loss: 0.3738
Epoch 30/58
```

```
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9429 - loss: 0.1618
Epoch 30: saving model to ft_weights/checkpoints/cp-30.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 850s 4s/step - accuracy: 0.9429 - loss: 0.1619 - val_ac
curacy: 0.9291 - val_loss: 0.2752
Epoch 31/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9555 - loss: 0.1585
Epoch 31: saving model to ft_weights/checkpoints/cp-31.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 803s 4s/step - accuracy: 0.9555 - loss: 0.1584 - val_ac
curacy: 0.9251 - val_loss: 0.3017
Epoch 32/58
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.9558 - loss: 0.1359
Epoch 32: saving model to ft_weights/checkpoints/cp-32.weights.h5
219/219 ━━━━━━━━━━━━━━━━━━━━ 787s 4s/step - accuracy: 0.9558 - loss: 0.1358 - val_ac
curacy: 0.9144 - val_loss: 0.3706
```

Classification Report

In [125…

```python
# Generate predictions on the test dataset
y_pred = []
y_true = []

# Loop through the test dataset batches and collect predictions and true labels
for images, labels in test_ds:
    predictions = model.predict(images)
    predicted_labels = tf.argmax(predictions, axis=1)
    y_pred.extend(predicted_labels.numpy())
    y_true.extend(labels.numpy())

# Print the classification report
from sklearn.metrics import classification_report

print("Classification Report:")
print(classification_report(y_true, y_pred, target_names=class_names))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
Classification Report:
                          precision    recall  f1-score   support

              Cattle Egret       0.90      0.96      0.93        81
         Common Kingfisher       0.96      0.96      0.96        70
              Gray Wagtail       0.95      0.95      0.95        77
          Northern Lapwing       0.87      0.95      0.91        87
       Red-Wattled Lapwing       0.92      0.89      0.90        79
            Ruddy Shelduck       0.90      0.93      0.91        67
```

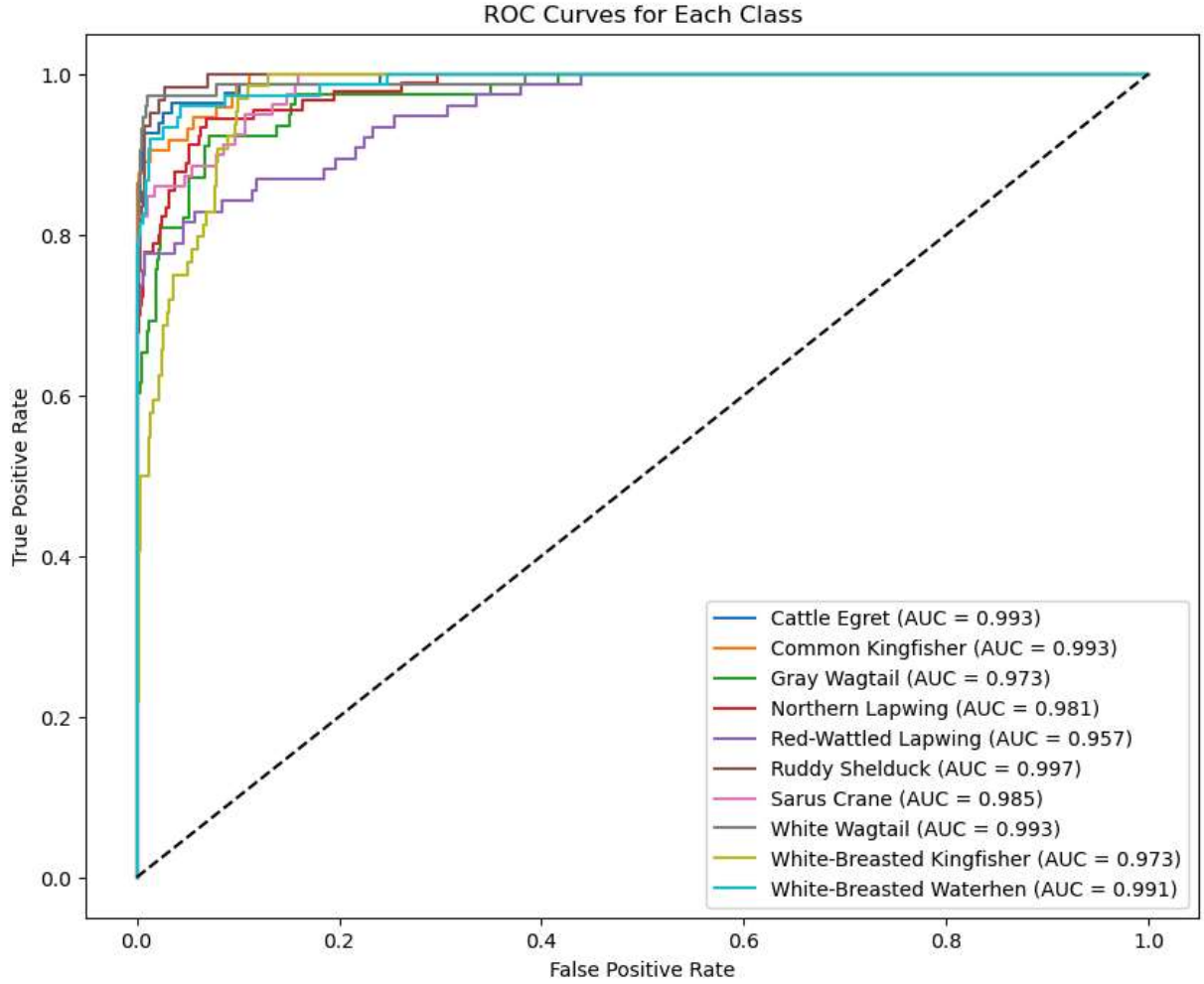
```
              Sarus Crane       0.97      0.85      0.91        81
            White Wagtail       0.97      0.97      0.97        71
White-Breasted Kingfisher       0.98      0.94      0.96        64
  White-Breasted Waterhen       0.93      0.93      0.93        75

                 accuracy                           0.93       752
                macro avg       0.94      0.93      0.93       752
             weighted avg       0.93      0.93      0.93       752
```

In [127…

```
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
import numpy as np

# Get the true labels and predictions for ROC AUC
y_true = []
y_pred_proba = []

for images, labels in test_ds:
    y_true.extend(labels.numpy())
    predictions = model.predict(images)
    y_pred_proba.extend(predictions)

# Binarize the true labels for multi-class ROC AUC
y_true_bin = label_binarize(y_true, classes=range(len(class_names)))

# Calculate the AUC for each class and the macro-average
auc_values = {}
for i in range(len(class_names)):
    auc_values[class_names[i]] = roc_auc_score(y_true_bin[:, i], np.array(y_pred_pr

# Print AUC values for each class
print("AUC for each class:")
for class_name, auc_value in auc_values.items():
    print(f"{class_name}: {auc_value:.3f}")

# Plot ROC curves for each class
plt.figure(figsize=(10, 8))
for i in range(len(class_names)):
    fpr, tpr, _ = roc_curve(y_true_bin[:, i], np.array(y_pred_proba)[:, i])
    plt.plot(fpr, tpr, label=f"{class_names[i]} (AUC = {auc_values[class_names[i]]:

# Plot diagonal line for random chance
plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for Each Class")
plt.legend(loc="lower right")
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
AUC for each class:
Cattle Egret: 0.993
Common Kingfisher: 0.993
Gray Wagtail: 0.973
Northern Lapwing: 0.981
Red-Wattled Lapwing: 0.957
Ruddy Shelduck: 0.997
Sarus Crane: 0.985
White Wagtail: 0.993
```

```
White-Breasted Kingfisher: 0.973
White-Breasted Waterhen: 0.991
```



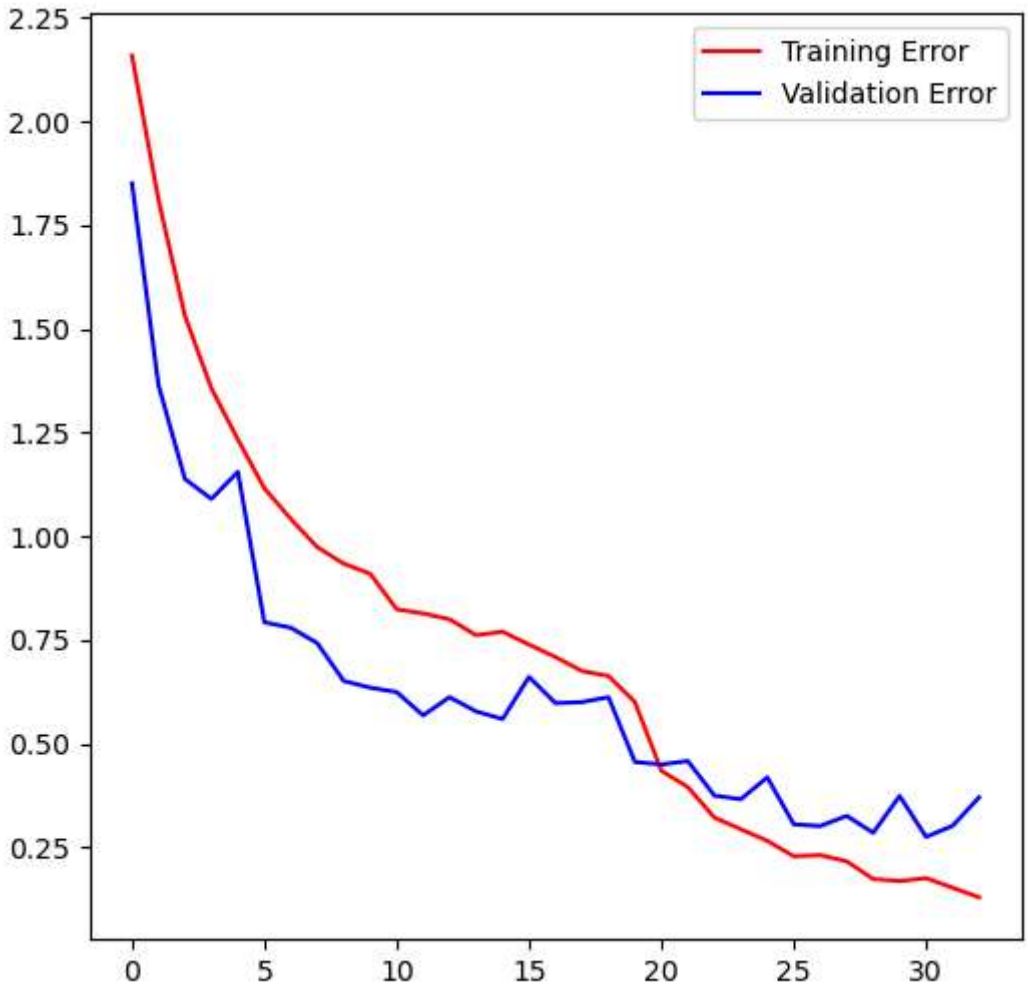
Plotting the learning curves

In [130…

```
loss = history.history['loss'] + fhistory.history['loss']
val_loss = history.history['val_loss'] + fhistory.history['val_loss']
accuracy = history.history['accuracy'] + fhistory.history['accuracy']
val_accuracy = history.history['val_accuracy'] + fhistory.history['val_accuracy']
```

In [132…

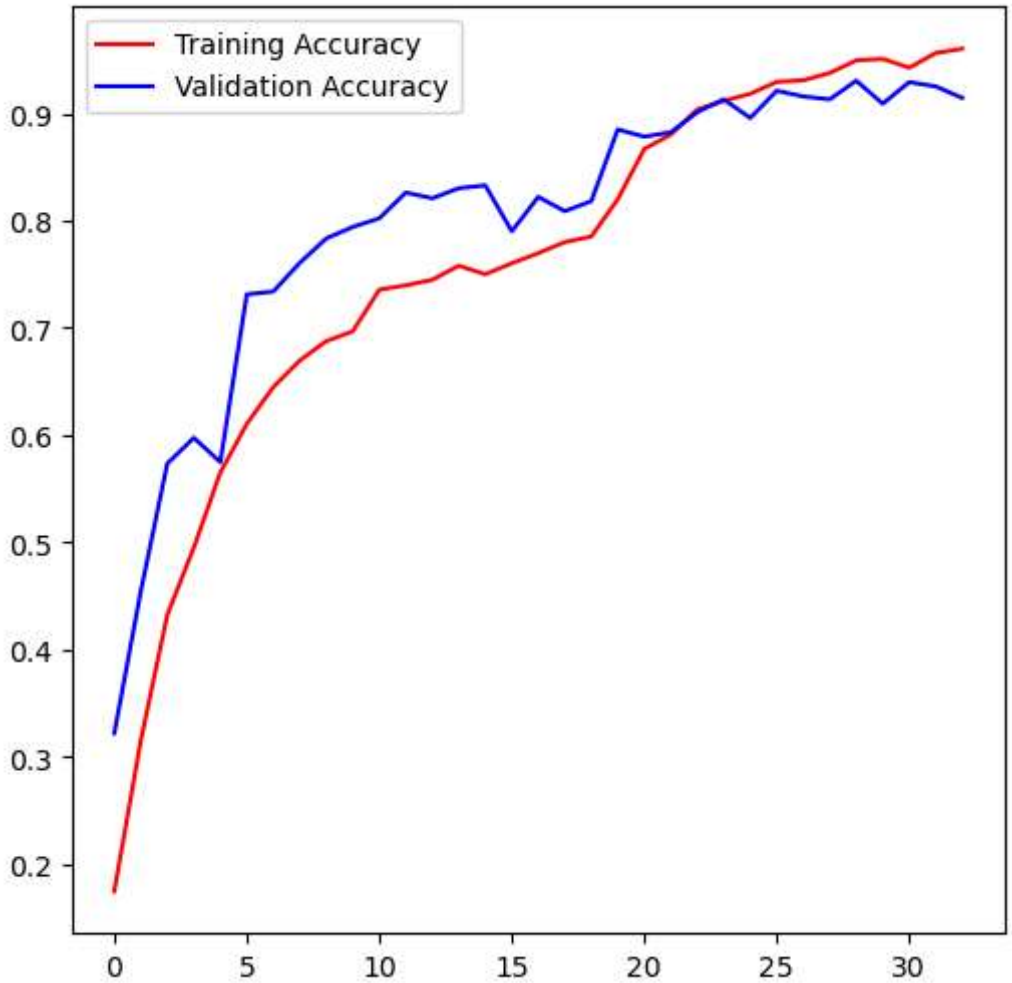
```
plt.figure(figsize=(6,6))
plt.plot(loss,color='r',label='Training Error')
plt.plot(val_loss, color='b',label='Validation Error')
plt.legend()
plt.show()
```



Accuracy Graph

In [135…

```
plt.figure(figsize=(6,6))
plt.plot(accuracy,color='r',label='Training Accuracy')
plt.plot(val_accuracy, color='b',label='Validation Accuracy')
plt.legend()
plt.show()
```



Evaluating the model on the Test Set

In [140…

```
test_loss, test_accuracy = model.evaluate(test_ds, verbose=0)
print("Loss on test set: {}".format(test_loss))
print("Accuracy on test set: {:.4f}%".format(test_accuracy*100))
```

```
Loss on test set: 0.2734771966934204
Accuracy on test set: 92.6862%
```

Plotting the Confusion Matrix

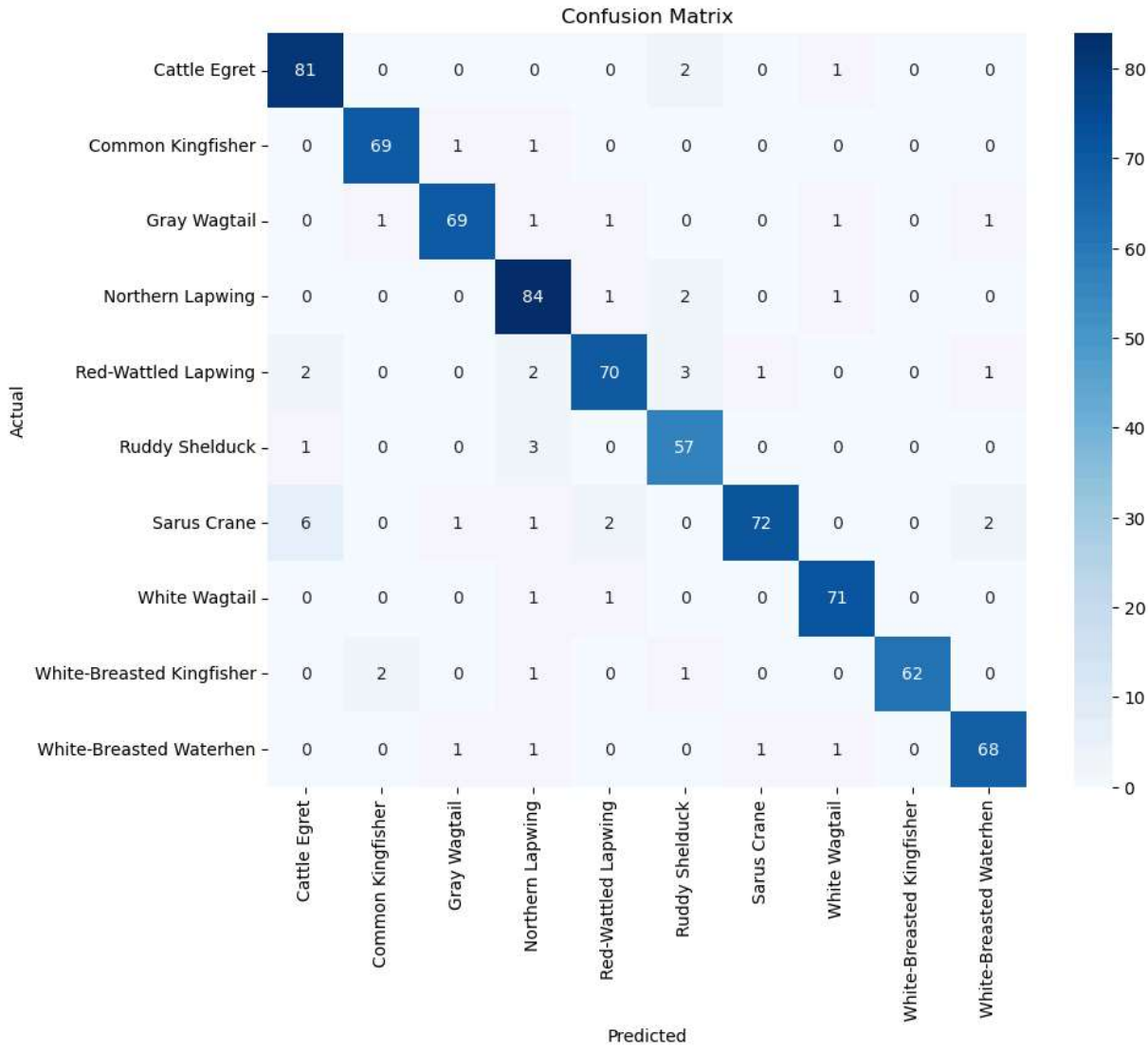
In [146…

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Evaluate the model on the test dataset
test_loss, test_accuracy = model.evaluate(test_ds, verbose=0)

print("Loss on test set: {:.4f}".format(test_loss))
print("Accuracy on test set: {:.4f}%".format(test_accuracy * 100))

# Generate predictions for the test dataset
y_true = []
y_pred = []

for images, labels in test_ds:
```

```
    y_true.extend(labels.numpy())
    predictions = model.predict(images)
    predicted_labels = tf.argmax(predictions, axis=1)
    y_pred.extend(predicted_labels.numpy())

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=class_names))

# Compute and plot the confusion matrix
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, ytickla
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```

```
Loss on test set: 0.2537
Accuracy on test set: 92.6862%
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

Classification Report:
                          precision    recall  f1-score   support

              Cattle Egret       0.90      0.96      0.93        84
         Common Kingfisher       0.96      0.97      0.97        71
              Gray Wagtail       0.96      0.93      0.95        74
```

```
              Northern Lapwing       0.88      0.95      0.92        88
           Red-Wattled Lapwing       0.93      0.89      0.91        79
                Ruddy Shelduck       0.88      0.93      0.90        61
                   Sarus Crane       0.97      0.86      0.91        84
                  White Wagtail       0.95      0.97      0.96        73
  White-Breasted Kingfisher       1.00      0.94      0.97        66
    White-Breasted Waterhen       0.94      0.94      0.94        72

                      accuracy                           0.93       752
                     macro avg       0.94      0.94      0.94       752
                  weighted avg       0.94      0.93      0.93       752
```



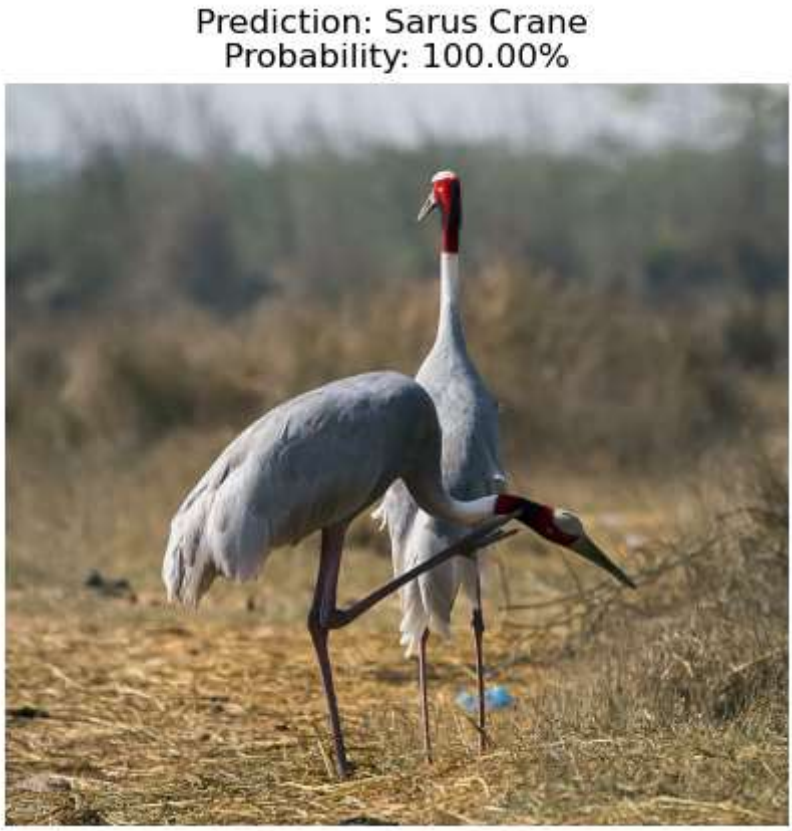
Saving the final model

In [148…

```
model.save('model/model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.
saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.savin
g.save_model(model, 'my_model.keras')`.
```

Making test predictions

In [153…

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf

# Load and preprocess the image
image = tf.keras.utils.load_img(r"C:\Users\himan\Desktop\training_set\training_set\
input_arr = tf.keras.utils.img_to_array(image)
input_arr = np.array([input_arr])
input_arr = tf.image.resize(input_arr, (224, 224))

# Display the image with prediction
plt.imshow(image)
plt.axis('off')
plt.title(f'Prediction: {predicted_class} \nProbability: {confidence:.2f}%')
plt.show()

# Make predictions
predictions = model.predict(input_arr, verbose=0)
score = tf.nn.softmax(predictions[0])
predicted_class = class_names[np.argmax(predictions)]
confidence = 100 * np.max(score)
```