MarineSpeciesMLImageClassifier

September 3, 2024

0.1 Marine Species Image Classifier

0.1.1 One-Time Set Up

[1]:
```
#install
!pip install tensorflow
!pip install kaggle
```

0.1.2 Set Up

[2]:
```
#Common Imports
import os
import numpy as np
import zipfile
import random
import shutil
import tensorflow as tf
from tensorflow import keras
from typing import Tuple, List
from shutil import copyfile
```

```
2024-08-26 23:26:46.026997: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2024-08-26 23:26:47.207985: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2024-08-26 23:26:47.556664: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-08-26 23:26:47.880168: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-08-26 23:26:48.010691: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-08-26 23:26:48.758758: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
```

```
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

[3]:
```
#Data Imports
import pandas as pd
from tqdm import tqdm
from glob import glob
from tensorflow import data as tfd
from tensorflow import image as tfi
```

[4]:
```
#Data Visualization
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import clear_output as cls
import PIL
```

[5]:
```
#Model Architecture
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Sequential
from tensorflow.keras.applications import ResNet50V2
from tensorflow.keras.applications import ResNet152V2
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications import Xception
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Dense
```

[6]:
```
# Model Training
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪classification_report
```

[7]:
```
# Constants
IMAGE_SIZE = 160 # size of the input image
BATCH_SIZE = 32 # number of samples per gradient update
EPOCHS = 50 # number of epochs to train the model
NUM_CLASSES = 9 # number of classes at the dataset
SPLIT_SIZE = .9 # Define proportion of images used for training


# Model training
LOSS = 'sparse_categorical_crossentropy'
METRICS = ['accuracy']

# Hyperparameters
```

```
LEARNING_RATE = 1e-3 # learning rate for the optimizer

# Random Seed
RANDOM_SEED = 42 # set random seed for reproducibility
tf.random.set_seed(RANDOM_SEED) # set random seed for TensorFlow
np.random.seed(RANDOM_SEED) # set random seed for NumPy
```

0.1.3 Load Data

[8]:
```
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
! kaggle datasets download "mikoajfish99/marine-animal-images"
```

```
mkdir: cannot create directory ‘/home/jupyter/.kaggle’: File exists
cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/home/jupyter/.kaggle/kaggle.json': No such file or
directory
Dataset URL: https://www.kaggle.com/datasets/mikoajfish99/marine-animal-images
License(s): CC0-1.0
marine-animal-images.zip: Skipping, found more recently modified local copy (use
--force to force download)
```

[13]:
```
! ls
```

```
MarineSpeciesMLImageClassifier.ipynb  marine-animal-images.zip  tmp
images                                notebook_template.ipynb
```

[14]:
```
# # Extract the archive

import zipfile
Original_dir = './tmp/'

try :
     #os.mkdir(Original_dir)
     local_zip = 'marine-animal-images.zip'
     zip_ref = zipfile.ZipFile(local_zip, 'r')
     zip_ref.extractall(Original_dir)
     zip_ref.close()
except:
     print("Error")

ORIGINAL_TRAIN_DIR= f"{Original_dir}images/train/"
ORIGINAL_TEST_DIR= f"{Original_dir}images/test/"
```

[17]:
```
#Load Data

ORIGINAL_TRAIN_DIR = "./tmp/images/train/"
```

```
ORIGINAL_TEST_DIR = "./tmp/images/test/"
```

[18]:
```
# Specify the directory paths for the data (train , test , validate)
# Define root directory
root_dir = './images/'
TRAINING_DIR =f"{root_dir}train/"  # path to the directory containing train␣
 ↪data
VALIDATION_DIR = f"{root_dir}valid/" # path to the directory containing␣
 ↪validation data
TEST_DIR = f"{root_dir}test/" # path to the directory containing test data

try:
    os.mkdir(root_dir)
    os.mkdir(TRAINING_DIR)
    os.mkdir(VALIDATION_DIR)
    os.mkdir(TEST_DIR)
except FileExistsError:
    print("Dir Already Exists")
```

```
Dir Already Exists
```

0.1.4 Explore Data

[19]:
```
def CheckDataInfo(TRAIN_DIR, TEST_DIR):
  # Check the class names
  class_names = sorted(os.listdir(TRAIN_DIR))
  n_classes = len(class_names)
  global NUM_CLASSES
  NUM_CLASSES = n_classes

  # Show
  print(f"Total number of classes: {n_classes}")
  print(f"Classes: {class_names}")
  print('\n')

  # Check the number of images in each class in the test dataset
  print("Total number of TRAIN imgs :", len ( glob(f"{TRAIN_DIR}/*/*")))
  No_images_per_class = []
  Class_name = []
  for i in os.listdir(TRAIN_DIR):
    Class_name.append(i)
    train_class = os.listdir(os.path.join(TRAIN_DIR,i))
    print('Number of images in {}={}'.format(i,len(train_class)))
    No_images_per_class.append(len(train_class))
  print('\n')
```

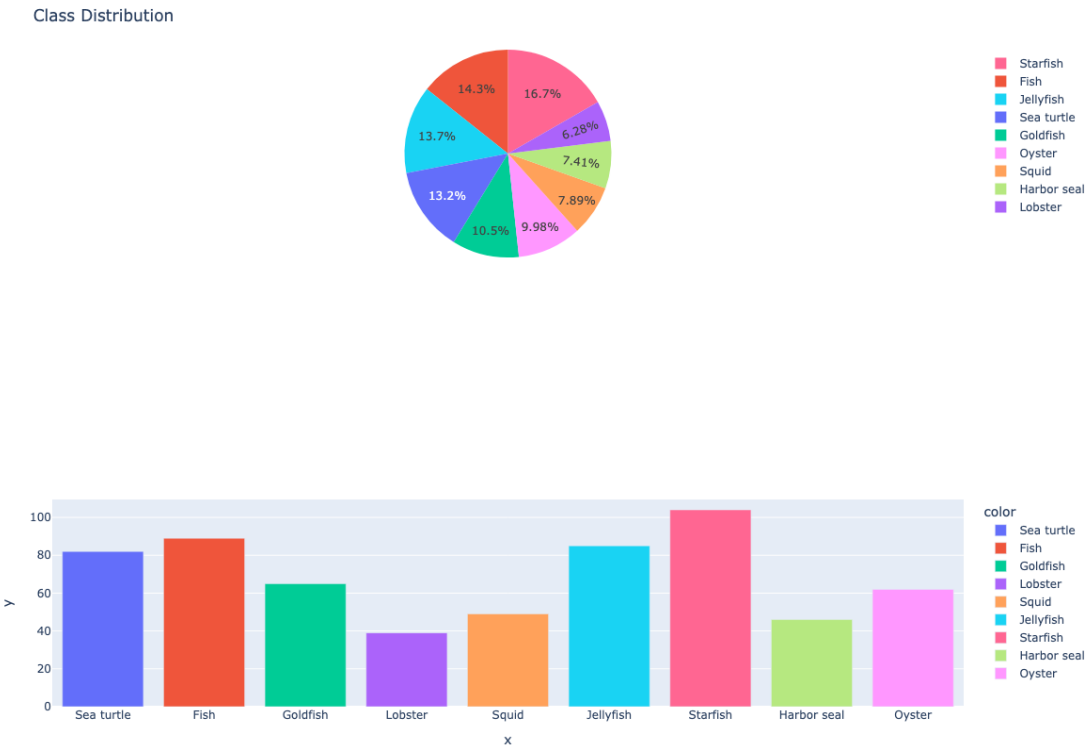
```
  # Check the number of images in each class in the test dataset
  print("Total number of TEST imgs :", len ( glob(f"{TEST_DIR}/*/*")))



  # visualize the number of images in each class in the training dataset
  VisualizeClassesDistribution (TRAIN_DIR,Class_name ,No_images_per_class )
```

[20]:
```
# visualize the number of images in each class in the training dataset
def VisualizeClassesDistribution (Original_Train_DIR ,Class_name␣
 ↪,No_images_per_class ):
  # fig = plt.figure(figsize=(8,3))
  # plt.bar(Class_name, No_images_per_class, color = sns.
 ↪color_palette("cubehelix",len(Class_name)))
  # fig.tight_layout()
  # fig.suptitle('Classes Distribution at the Train Dataset', fontsize=10)

  # Calculate class distribution
  class_dis = [len(os.listdir(Original_Train_DIR + name)) for name in␣
 ↪Class_name]

  # Visualize using interactive pie chart
  pie_chart = px.pie(values=class_dis, names=Class_name, color=Class_name)
  pie_chart.update_layout({'title':{'text':"Class Distribution"}})
  pie_chart.show()

  # Visualize using interactive bar chart
  bar_chart = px.bar(y=class_dis, x=Class_name, color=Class_name)
  bar_chart.show()
```

[21]:
```
CheckDataInfo(ORIGINAL_TRAIN_DIR , ORIGINAL_TEST_DIR )
```

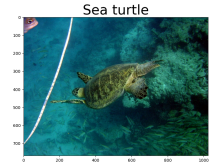
```
Total number of classes: 9
Classes: ['Fish', 'Goldfish', 'Harbor seal', 'Jellyfish', 'Lobster', 'Oyster',
'Sea turtle', 'Squid', 'Starfish']


Total number of TRAIN imgs : 621
Number of images in Sea turtle=82
Number of images in Fish=89
Number of images in Goldfish=65
Number of images in Lobster=39
Number of images in Squid=49
Number of images in Jellyfish=85
Number of images in Starfish=104
Number of images in Harbor seal=46
Number of images in Oyster=62
```

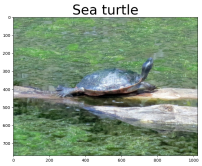
```
Total number of TEST imgs : 185
```

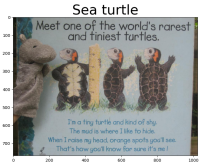


[22]:
```python
def ImgVisulalization (TRAIN_DIR , NUM_CLASSES):
  # Visualize the images in the dataset
  fig, axs = plt.subplots(NUM_CLASSES, 3, figsize = (50,50))
  count = 0
  # for every class in the dataset
  for i in os.listdir(TRAIN_DIR):
    # get the list of all images that belong to a particular class
    train_class = os.listdir(os.path.join(TRAIN_DIR, i))

    # plot 3 images per class
    for j in range(3):
      img = os.path.join(TRAIN_DIR, i, train_class[j])
      axs[count][j].set_title (i , fontsize=33)
      axs[count][j].imshow(PIL.Image.open(img))
    count += 1

  fig.tight_layout()
```

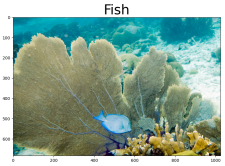
[23]:
```python
ImgVisulalization (ORIGINAL_TRAIN_DIR , NUM_CLASSES)
```


Sea turtle

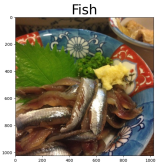

Sea turtle



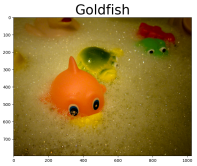
Sea turtle

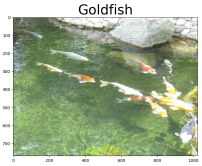

Fish

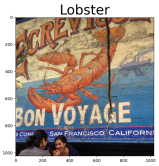

Fish


Fish

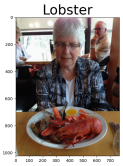
Goldfish

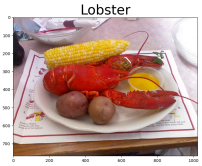

Goldfish

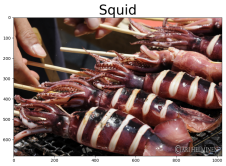

Goldfish

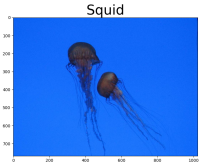


Lobster

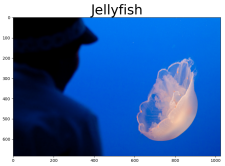

Lobster

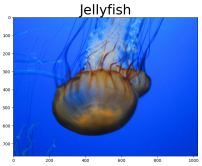

Lobster

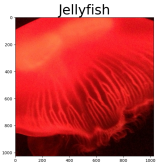

Squid

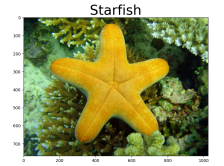

Squid


Squid


Jellyfish


Jellyfish


Jellyfish


Starfish


Starfish


Starfish


Harbor seal


Harbor seal


Harbor seal


Oyster


Oyster


Oyster

0.1.5 Split Data

[24]:
```
#  create_train_val_dirs
def create_train_val_dirs(root_path):

  os.mkdir(root_path)
  os.mkdir(root_path+'/train')
  os.mkdir(root_path+'/val')
  os.mkdir(root_path+'/train/Fish')
  os.mkdir(root_path+'/train/Goldfish')
  os.mkdir(root_path+'/train/Harbor seal')
  os.mkdir(root_path+'/train/Jellyfish')
  os.mkdir(root_path+'/train/Lobster')
  os.mkdir(root_path+'/train/Oyster')
  os.mkdir(root_path+'/train/Sea turtle')
  os.mkdir(root_path+'/train/Squid')
  os.mkdir(root_path+'/train/Starfish')

  os.mkdir(root_path+'/val/Fish')
  os.mkdir(root_path+'/val/Goldfish')
  os.mkdir(root_path+'/val/Harbor seal')
  os.mkdir(root_path+'/val/Jellyfish')
  os.mkdir(root_path+'/val/Lobster')
  os.mkdir(root_path+'/val/Oyster')
  os.mkdir(root_path+'/val/Sea turtle')
  os.mkdir(root_path+'/val/Squid')
  os.mkdir(root_path+'/val/Starfish')
```

[25]:
```
#Call create_train_val_dirs

# Empty directory to prevent FileExistsError is the function is run several␣
 ↪times
if os.path.exists(root_dir):
  shutil.rmtree(root_dir)
try:
    create_train_val_dirs(root_path=root_dir)
except FileExistsError:
    print("Error")
```

[26]:
```
# Test your create_train_val_dirs function
for rootdir, dirs, files in os.walk(root_dir):
    for subdir in dirs:
        print(os.path.join(rootdir, subdir))
```

```
./images/val
./images/train
./images/val/Sea turtle
./images/val/Fish
```

```
./images/val/Goldfish
./images/val/Lobster
./images/val/Squid
./images/val/Jellyfish
./images/val/Starfish
./images/val/Harbor seal
./images/val/Oyster
./images/train/Sea turtle
./images/train/Fish
./images/train/Goldfish
./images/train/Lobster
./images/train/Squid
./images/train/Jellyfish
./images/train/Starfish
./images/train/Harbor seal
./images/train/Oyster
```

[27]:
```python
# split_data
def split_data(SOURCE_DIR, TRAINING_DIR, VALIDATION_DIR, SPLIT_SIZE):
    """
    Splits the data into train and test sets

    Args:
        SOURCE_DIR (string): directory path containing the images
        TRAINING_DIR (string): directory path to be used for training
        VALIDATION_DIR (string): directory path to be used for validation
        SPLIT_SIZE (float): proportion of the dataset to be used for training

    Returns:
        None
    """

    files = []
    for filename in os.listdir(SOURCE_DIR):
        file = SOURCE_DIR + filename
        if os.path.getsize(file) > 0:
            files.append(filename)
        else:
            print(filename + " is zero length, so ignoring.")

    training_length = int(len(files) * SPLIT_SIZE)
    testing_length = int(len(files) - training_length)
    shuffled_set = random.sample(files, len(files))
    training_set = shuffled_set[0:training_length]
    testing_set = shuffled_set[-testing_length:]

    for filename in training_set:
```

```
        this_file = SOURCE_DIR + filename
        destination = TRAINING_DIR + filename
        copyfile(this_file, destination)

    for filename in testing_set:
        this_file = SOURCE_DIR + filename
        destination = VALIDATION_DIR + filename
        copyfile(this_file, destination)
```

[28]:
```
# Test your split_data function
# Define paths


Fish_SOURCE_DIR = f"{ORIGINAL_TRAIN_DIR}Fish/"
Goldfish_SOURCE_DIR = f"{ORIGINAL_TRAIN_DIR}Goldfish/"
Harbor_seal_SOURCE_DIR = f"{ORIGINAL_TRAIN_DIR}Harbor seal/"
Jellyfish_SOURCE_DIR = f"{ORIGINAL_TRAIN_DIR}Jellyfish/"
Lobster_SOURCE_DIR = f"{ORIGINAL_TRAIN_DIR}Lobster/"
Oyster_SOURCE_DIR = f"{ORIGINAL_TRAIN_DIR}Oyster/"
Sea_turtle_SOURCE_DIR = f"{ORIGINAL_TRAIN_DIR}Sea turtle/"
Squid_SOURCE_DIR = f"{ORIGINAL_TRAIN_DIR}Squid/"
Starfish_SOURCE_DIR = f"{ORIGINAL_TRAIN_DIR}Starfish/"


TRAINING_DIR = "images/train/"
VALIDATION_DIR = "images/val/"
TRAINING_Fish_DIR = os.path.join(TRAINING_DIR, "Fish/")
VALIDATION_Fish_DIR = os.path.join(VALIDATION_DIR, "Fish/")
TRAINING_Goldfish_DIR = os.path.join(TRAINING_DIR, "Goldfish/")
VALIDATION_Goldfish_DIR = os.path.join(VALIDATION_DIR, "Goldfish/")
TRAINING_Harbor_seal_DIR = os.path.join(TRAINING_DIR, "Harbor seal/")
VALIDATION_Harbor_seal_DIR = os.path.join(VALIDATION_DIR, "Harbor seal/")
TRAINING_Jellyfish_DIR = os.path.join(TRAINING_DIR, "Jellyfish/")
VALIDATION_Jellyfish_DIR = os.path.join(VALIDATION_DIR, "Jellyfish/")
TRAINING_Lobster_DIR = os.path.join(TRAINING_DIR, "Lobster/")
VALIDATION_Lobster_DIR = os.path.join(VALIDATION_DIR, "Lobster/")
TRAINING_Oyster_DIR = os.path.join(TRAINING_DIR, "Oyster/")
VALIDATION_Oyster_DIR = os.path.join(VALIDATION_DIR, "Oyster/")
TRAINING_Sea_turtle_DIR = os.path.join(TRAINING_DIR, "Sea turtle/")
VALIDATION_Sea_turtle_DIR = os.path.join(VALIDATION_DIR, "Sea turtle/")
TRAINING_Squid_DIR = os.path.join(TRAINING_DIR, "Squid/")
VALIDATION_Squid_DIR = os.path.join(VALIDATION_DIR, "Squid/")
TRAINING_Starfish_DIR = os.path.join(TRAINING_DIR, "Starfish/")
VALIDATION_Starfish_DIR = os.path.join(VALIDATION_DIR, "Starfish/")

# Empty directories in case you run this cell multiple times
```

```
if len(os.listdir(TRAINING_Fish_DIR)) > 0:
    for file in os.scandir(TRAINING_Fish_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_Goldfish_DIR)) > 0:
    for file in os.scandir(TRAINING_Goldfish_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_Harbor_seal_DIR)) > 0:
    for file in os.scandir(TRAINING_Harbor_seal_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_Jellyfish_DIR)) > 0:
    for file in os.scandir(TRAINING_Jellyfish_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_Lobster_DIR)) > 0:
    for file in os.scandir(TRAINING_Lobster_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_Oyster_DIR)) > 0:
    for file in os.scandir(TRAINING_Oyster_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_Sea_turtle_DIR)) > 0:
    for file in os.scandir(TRAINING_Sea_turtle_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_Squid_DIR)) > 0:
    for file in os.scandir(TRAINING_Squid_DIR):
        os.remove(file.path)
if len(os.listdir(TRAINING_Starfish_DIR)) > 0:
    for file in os.scandir(TRAINING_Starfish_DIR):
        os.remove(file.path)

# Validation Dirs

if len(os.listdir(VALIDATION_Fish_DIR)) > 0:
    for file in os.scandir(VALIDATION_Fish_DIR):
        os.remove(file.path)
if len(os.listdir(VALIDATION_Goldfish_DIR)) > 0:
    for file in os.scandir(VALIDATION_Goldfish_DIR):
        os.remove(file.path)
if len(os.listdir(VALIDATION_Harbor_seal_DIR)) > 0:
    for file in os.scandir(VALIDATION_Harbor_seal_DIR):
        os.remove(file.path)
if len(os.listdir(VALIDATION_Jellyfish_DIR)) > 0:
    for file in os.scandir(VALIDATION_Jellyfish_DIR):
        os.remove(file.path)
if len(os.listdir(VALIDATION_Lobster_DIR)) > 0:
    for file in os.scandir(VALIDATION_Lobster_DIR):
        os.remove(file.path)
if len(os.listdir(VALIDATION_Oyster_DIR)) > 0:
    for file in os.scandir(VALIDATION_Oyster_DIR):
```

```
        os.remove(file.path)
if len(os.listdir(VALIDATION_Sea_turtle_DIR)) > 0:
    for file in os.scandir(VALIDATION_Sea_turtle_DIR):
        os.remove(file.path)
if len(os.listdir(VALIDATION_Squid_DIR)) > 0:
    for file in os.scandir(VALIDATION_Squid_DIR):
        os.remove(file.path)
if len(os.listdir(VALIDATION_Starfish_DIR)) > 0:
    for file in os.scandir(VALIDATION_Starfish_DIR):
        os.remove(file.path)



# Run the function
split_data(Fish_SOURCE_DIR, TRAINING_Fish_DIR, VALIDATION_Fish_DIR, SPLIT_SIZE)
split_data(Goldfish_SOURCE_DIR , TRAINING_Goldfish_DIR,␣
 ↪VALIDATION_Goldfish_DIR, SPLIT_SIZE)
split_data(Harbor_seal_SOURCE_DIR , TRAINING_Harbor_seal_DIR ,␣
 ↪VALIDATION_Harbor_seal_DIR, SPLIT_SIZE)
split_data(Jellyfish_SOURCE_DIR, TRAINING_Jellyfish_DIR,␣
 ↪VALIDATION_Jellyfish_DIR, SPLIT_SIZE)
split_data(Lobster_SOURCE_DIR, TRAINING_Lobster_DIR, VALIDATION_Lobster_DIR,␣
 ↪SPLIT_SIZE)
split_data(Oyster_SOURCE_DIR, TRAINING_Oyster_DIR, VALIDATION_Oyster_DIR,␣
 ↪SPLIT_SIZE)
split_data(Sea_turtle_SOURCE_DIR, TRAINING_Sea_turtle_DIR,␣
 ↪VALIDATION_Sea_turtle_DIR, SPLIT_SIZE)
split_data(Squid_SOURCE_DIR, TRAINING_Squid_DIR, VALIDATION_Squid_DIR,␣
 ↪SPLIT_SIZE)
split_data(Starfish_SOURCE_DIR, TRAINING_Starfish_DIR, VALIDATION_Starfish_DIR,␣
 ↪SPLIT_SIZE)



# Check that the number of images matches the expected output
# function should perform copies rather than moving images so original␣
 ↪directories should contain unchanged images

# Training and validation splits
print(f"Original Fish's directory has {len(os.listdir(Fish_SOURCE_DIR))}␣
 ↪images")
print(f"There are {len(os.listdir(TRAINING_Fish_DIR))} images of Fish for␣
 ↪training")
print(f"There are {len(os.listdir(VALIDATION_Fish_DIR))} images of Fish for␣
 ↪validation")
print('\n\n')
```

```
print(f"Original Goldfish's directory has {len(os.
 ↪listdir(Goldfish_SOURCE_DIR))} images")
print(f"There are {len(os.listdir(TRAINING_Goldfish_DIR))} images of Goldfish␣
 ↪for training")
print(f"There are {len(os.listdir(VALIDATION_Goldfish_DIR))} images of Goldfish␣
 ↪for validation")
print('\n\n')

print(f"Original Harbor seal's directory has {len(os.
 ↪listdir(Harbor_seal_SOURCE_DIR))} images")
print(f"There are {len(os.listdir(TRAINING_Harbor_seal_DIR))} images of Harbor␣
 ↪seal for training")
print(f"There are {len(os.listdir(VALIDATION_Harbor_seal_DIR))} images of␣
 ↪Harbor seal for validation")
print('\n\n')

print(f"Original Jellyfish's directory has {len(os.
 ↪listdir(Jellyfish_SOURCE_DIR))} images")
print(f"There are {len(os.listdir(TRAINING_Jellyfish_DIR))} images of Jellyfish␣
 ↪for training")
print(f"There are {len(os.listdir(VALIDATION_Jellyfish_DIR))} images of␣
 ↪Jellyfish for validation")
print('\n\n')

print(f"Original Lobster's directory has {len(os.listdir(Lobster_SOURCE_DIR))}␣
 ↪images")
print(f"There are {len(os.listdir(TRAINING_Lobster_DIR))} images of Lobster for␣
 ↪training")
print(f"There are {len(os.listdir(VALIDATION_Lobster_DIR))} images of Lobster␣
 ↪for validation")
print('\n\n')

print(f"Original Oyster's directory has {len(os.listdir(Oyster_SOURCE_DIR))}␣
 ↪images")
print(f"There are {len(os.listdir(TRAINING_Oyster_DIR))} images of Oyster for␣
 ↪training")
print(f"There are {len(os.listdir(VALIDATION_Oyster_DIR))} images of Oyster for␣
 ↪validation")
print('\n\n')

print(f"Original Sea turtle's directory has {len(os.
 ↪listdir(Sea_turtle_SOURCE_DIR))} images")
print(f"There are {len(os.listdir(TRAINING_Sea_turtle_DIR))} images of Sea␣
 ↪turtle for training")
```

```
print(f"There are {len(os.listdir(VALIDATION_Sea_turtle_DIR))} images of Sea␣
 ↪turtle for validation")
print('\n\n')

print(f"Original Squid's directory has {len(os.listdir(Squid_SOURCE_DIR))}␣
 ↪images")
print(f"There are {len(os.listdir(TRAINING_Squid_DIR))} images of Squid for␣
 ↪training")
print(f"There are {len(os.listdir(VALIDATION_Squid_DIR))} images of Squid for␣
 ↪validation")
print('\n\n')

print(f"Original Starfish's directory has {len(os.
 ↪listdir(Starfish_SOURCE_DIR))} images")
print(f"There are {len(os.listdir(TRAINING_Starfish_DIR))} images of Starfish␣
 ↪for training")
print(f"There are {len(os.listdir(VALIDATION_Starfish_DIR))} images of Starfish␣
 ↪for validation")
print('\n\n')
```

```
Original Fish's directory has 89 images
There are 80 images of Fish for training
There are 9 images of Fish for validation



Original Goldfish's directory has 65 images
There are 58 images of Goldfish for training
There are 7 images of Goldfish for validation



Original Harbor seal's directory has 46 images
There are 41 images of Harbor seal for training
There are 5 images of Harbor seal for validation



Original Jellyfish's directory has 85 images
There are 76 images of Jellyfish for training
There are 9 images of Jellyfish for validation



Original Lobster's directory has 39 images
There are 35 images of Lobster for training
There are 4 images of Lobster for validation
```

```
Original Oyster's directory has 62 images
There are 55 images of Oyster for training
There are 7 images of Oyster for validation
```

```
Original Sea turtle's directory has 82 images
There are 73 images of Sea turtle for training
There are 9 images of Sea turtle for validation
```

```
Original Squid's directory has 49 images
There are 44 images of Squid for training
There are 5 images of Squid for validation
```

```
Original Starfish's directory has 104 images
There are 93 images of Starfish for training
There are 11 images of Starfish for validation
```

0.1.6 Create Data Generators

[29]:
```
#  train_val_generators
def train_val_generators(TRAINING_DIR, VALIDATION_DIR , TEST_DIR):
    """
    Creates the training and validation data generators

    Args:
        TRAINING_DIR (string): directory path containing the training images
        VALIDATION_DIR (string): directory path containing the testing/
 ↪validation images

    Returns:
        train_generator, validation_generator - tuple containing the generators
    """

    # Instantiate the ImageDataGenerator class
    train_datagen = ImageDataGenerator(rescale=1.0/255.) #set the rescale␣
 ↪argument
```

```
    # Pass in the appropriate arguments to the flow_from_directory method
    train_generator = train_datagen.flow_from_directory(directory=TRAINING_DIR,
                                                        batch_size=BATCH_SIZE,
                                                        class_mode='sparse',
                                                        target_size=(IMAGE_SIZE,␣
 ↪IMAGE_SIZE),
                                                        shuffle = True)

    # Instantiate the ImageDataGenerator class
    validation_datagen = ImageDataGenerator(rescale=1.0/255.) #set the rescale␣
 ↪argument

    # Pass in the appropriate arguments to the flow_from_directory method
    validation_generator = validation_datagen.
 ↪flow_from_directory(VALIDATION_DIR,
                                                                 ␣
 ↪batch_size=BATCH_SIZE,
                                                                 ␣
 ↪class_mode='sparse',
                                                                 ␣
 ↪target_size=(IMAGE_SIZE, IMAGE_SIZE),
                                                                  shuffle = True)

    # Instantiate the ImageDataGenerator class  , For test datagenerator, we␣
 ↪only normalize the data.
    test_datagen = ImageDataGenerator(rescale=1./255.)

    # Pass in the appropriate arguments to the flow_from_directory method
    test_generator = test_datagen.flow_from_directory(
        TEST_DIR,
        target_size=(IMAGE_SIZE, IMAGE_SIZE),
        batch_size=BATCH_SIZE,
        class_mode='sparse')

    return train_generator, validation_generator , test_generator
```

[30]:
```
# Test your generators
train_generator, validation_generator , test_generator =␣
 ↪train_val_generators(TRAINING_DIR, VALIDATION_DIR, ORIGINAL_TEST_DIR)
```

```
Found 555 images belonging to 9 classes.
Found 66 images belonging to 9 classes.
Found 185 images belonging to 9 classes.
```

0.2 Backbone Analysis

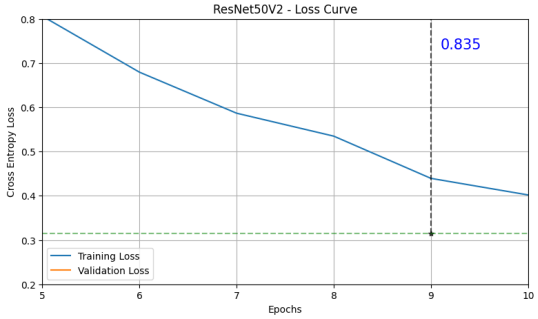
[31]:
```
# Name all the backbones
backbone_names = [
    'ResNet50V2',
    'ResNet152V2',
    'InceptionV3',
    'Xception',
    'MobileNetV2'
]

# Load all the backbones
backbones = [
    ResNet50V2(include_top=False, weights='imagenet', input_shape=(IMAGE_SIZE,␣
 ↪IMAGE_SIZE, 3)),
    ResNet152V2(include_top=False, weights='imagenet', input_shape=(IMAGE_SIZE,␣
 ↪IMAGE_SIZE, 3)),
    InceptionV3(include_top=False, weights='imagenet', input_shape=(IMAGE_SIZE,␣
 ↪IMAGE_SIZE, 3)),
    Xception(include_top=False, weights='imagenet', input_shape=(IMAGE_SIZE,␣
 ↪IMAGE_SIZE, 3)),
    MobileNetV2(include_top=False, weights='imagenet', input_shape=(IMAGE_SIZE,␣
 ↪IMAGE_SIZE, 3)),
]

# Freeze the weights of all the backbones
for backbone in backbones:
    backbone.trainable = False
```

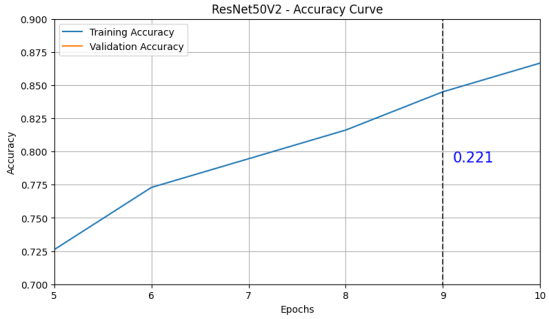
[32]:
```
# Record the learing curve for all the backbones
HISTORIES = {}

# Loop over all the backbones
for name, backbone in zip(backbone_names, backbones):

    # Show
    print(f"Training: {name}\n")

    # Make a small model
    model = Sequential([
        backbone,
        GlobalAveragePooling2D(),
        Dropout(0.5),
        Dense(NUM_CLASSES, activation='softmax')
    ])

    # Compile the model
```

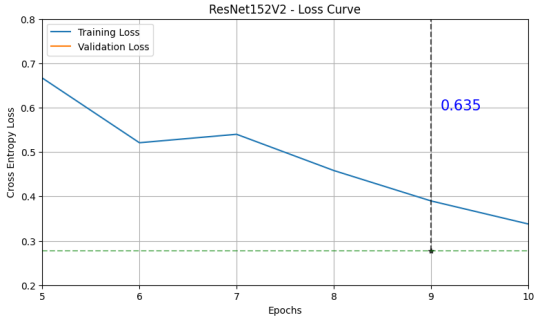
```
    model.compile(
        loss=LOSS,
        optimizer=Adam(LEARNING_RATE),
        metrics=METRICS
    )

    # Train the model
    model_history = model.fit(
        train_generator,
        validation_data= validation_generator,
        epochs=15,
        batch_size=BATCH_SIZE,
        verbose = 2,
    )

    # Store the history
    HISTORIES[name] = model_history.history
    cls()
```

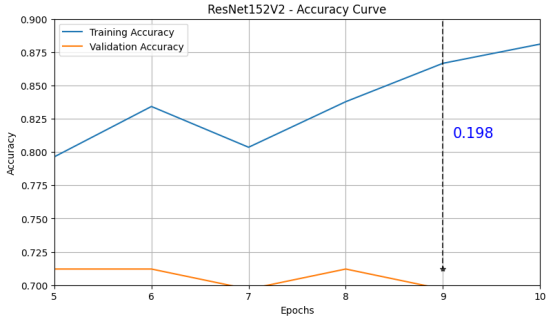
[35]:
```
# Loop over the model training curves
for name, history in HISTORIES.items():

    # Create a new figure for each backbone
    plt.figure(figsize=(20,5))

    # Plot the loss curve in the first subplot
    plt.subplot(1, 2, 1)
    plt.title(f"{name} - Loss Curve")
    plt.plot(history['loss'], label="Training Loss")
    plt.plot(history['val_loss'], label="Validation Loss")

    # Plot a vertical line at epoch 9 and annotate the difference between the␣
 ↪validation loss and the training loss
    plt.plot([9, 9],[min(history['loss']), min(history['val_loss'])],␣
 ↪linestyle='--', marker="*", color='k', alpha=0.7)
    plt.text(x=9.1, y=np.mean([min(history['loss']),␣
 ↪min(history['val_loss'])]), s=str(np.round(min(history['val_loss']) -␣
 ↪min(history['loss']),3)), fontsize=15, color='b')

    # Plot a horizontal line at epoch 9 and annotate the values for the␣
 ↪validation loss and training loss.
    plt.axhline(min(history['loss']), color='g', linestyle="--", alpha=0.5)

    # Set the x- and y-labels, and the x- and y-limits
    plt.xlabel("Epochs")
    plt.ylabel("Cross Entropy Loss")
    plt.ylim([0.2, 0.8])
```

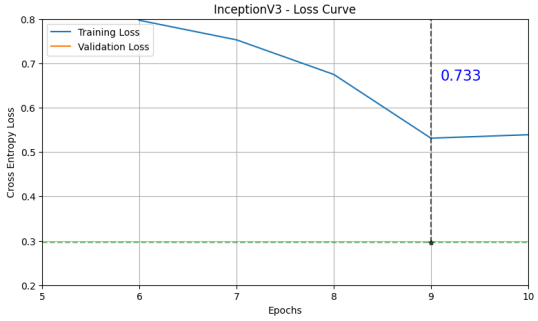
```
    plt.xlim([5, 10])

    # Show the legend and grid
    plt.legend()
    plt.grid()

    # Plot the accuracy curve in the second subplot
    plt.subplot(1, 2, 2)
    plt.title(f"{name} - Accuracy Curve")
    plt.plot(history['accuracy'], label="Training Accuracy")
    plt.plot(history['val_accuracy'], label="Validation Accuracy")

    # Plot a vertical line at epoch 9 and annotate the difference between the␣
↪validation accuracy and the training accuracy
    plt.plot([9, 9],[max(history['accuracy']), max(history['val_accuracy'])],␣
↪linestyle='--', marker="*", color='k', alpha=0.7)
    plt.text(x=9.1, y=np.mean([max(history['accuracy']),␣
↪max(history['val_accuracy'])]), s=str(np.round(max(history['accuracy']) -␣
↪max(history['val_accuracy']),3)), fontsize=15, color='b')

    # Set the x- and y-labels, and the x- and y-limits
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.ylim([0.7,0.9])
    plt.xlim([5, 10])

    # Show the legend and grid
    plt.legend()
    plt.grid()

    # Show the plot
    plt.show()
```

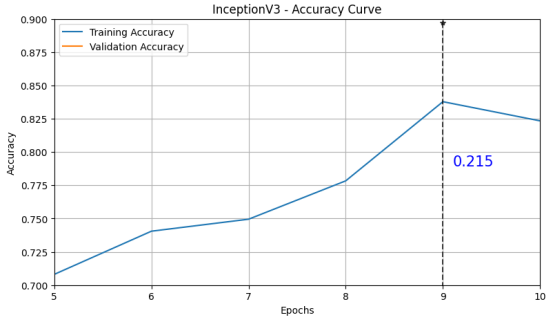


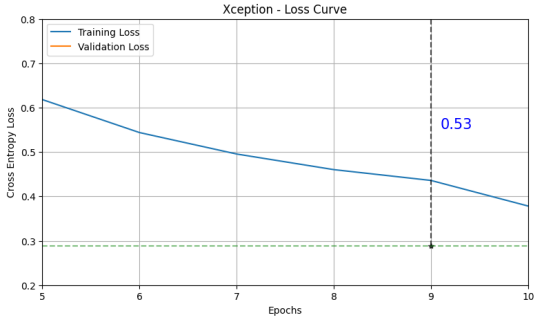


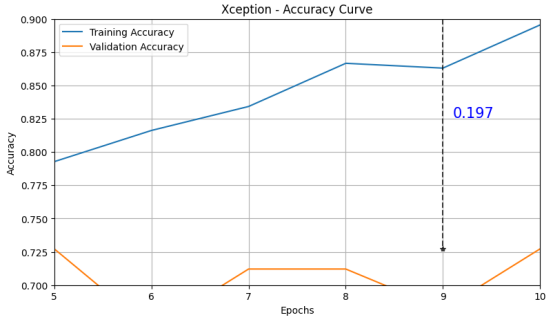

ResNet152V2 - Loss Curve
Training Loss
Validation Loss
0.635
Cross Entropy Loss
Epochs

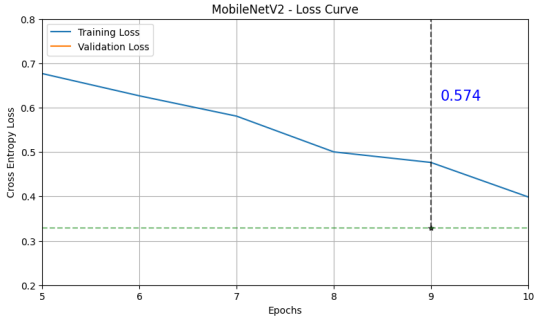

ResNet152V2 - Accuracy Curve
Training Accuracy
Validation Accuracy
0.198
Accuracy
Epochs

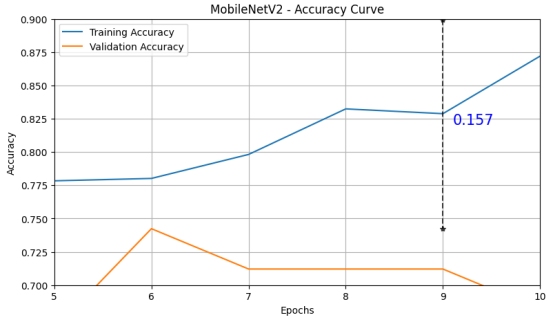

InceptionV3 - Loss Curve
Training Loss
Validation Loss
0.733
Cross Entropy Loss
Epochs


InceptionV3 - Accuracy Curve
Training Accuracy
Validation Accuracy
0.215
Accuracy
Epochs


Xception - Loss Curve
Training Loss
Validation Loss
0.53
Cross Entropy Loss
Epochs


Xception - Accuracy Curve
Training Accuracy
Validation Accuracy
0.197
Accuracy
Epochs


MobileNetV2 - Loss Curve
Training Loss
Validation Loss
0.574
Cross Entropy Loss
Epochs


MobileNetV2 - Accuracy Curve
Training Accuracy
Validation Accuracy
0.157
Accuracy
Epochs

[40]:
```
# Xception Backbone
print("Loading Xception Backbone: ")
xception = Xception(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3),␣
 ↪weights='imagenet', include_top=False)

# Freeze the model weights
xception.trainable = False

# The Mobilenet Model baseline
xception = Sequential([
    xception,
    GlobalAveragePooling2D(),
    Dropout(0.5),
    Dense(NUM_CLASSES, activation='softmax')
])

# Compile the Baseline
xception.compile(
    loss=LOSS,
    optimizer=Adam(learning_rate=LEARNING_RATE),
    metrics=METRICS
)

# Train the Xception Baseline Model
print("\nTraining Baseline Model: ")
xception.fit(
   train_generator,
   validation_data= validation_generator,
   epochs=EPOCHS ,
   callbacks=[
#         EarlyStopping(patience=3, restore_best_weights=True),
        ModelCheckpoint("XceptionBaseline.keras", save_best_only=True)
    ],
    batch_size=BATCH_SIZE
)
```

```
Loading Xception Backbone:

Training Baseline Model:
Epoch 1/50
18/18             40s 2s/step -
accuracy: 0.2383 - loss: 2.3225 - val_accuracy: 0.6212 - val_loss: 1.2353
Epoch 2/50
18/18             33s 2s/step -
accuracy: 0.5248 - loss: 1.3310 - val_accuracy: 0.6970 - val_loss: 0.9593
Epoch 3/50
18/18             34s 2s/step -
accuracy: 0.6863 - loss: 0.9417 - val_accuracy: 0.7273 - val_loss: 0.8529
```

```
Epoch 4/50
18/18             32s 2s/step -
accuracy: 0.7406 - loss: 0.7475 - val_accuracy: 0.6970 - val_loss: 0.8174
Epoch 5/50
18/18             34s 2s/step -
accuracy: 0.7588 - loss: 0.7139 - val_accuracy: 0.6818 - val_loss: 0.8196
Epoch 6/50
18/18             39s 2s/step -
accuracy: 0.8010 - loss: 0.6298 - val_accuracy: 0.7121 - val_loss: 0.8001
Epoch 7/50
18/18             35s 2s/step -
accuracy: 0.8044 - loss: 0.5658 - val_accuracy: 0.7121 - val_loss: 0.7882
Epoch 8/50
18/18             33s 2s/step -
accuracy: 0.8540 - loss: 0.4802 - val_accuracy: 0.7424 - val_loss: 0.8225
Epoch 9/50
18/18             33s 2s/step -
accuracy: 0.8694 - loss: 0.4367 - val_accuracy: 0.6970 - val_loss: 0.8242
Epoch 10/50
18/18             33s 2s/step -
accuracy: 0.8816 - loss: 0.3833 - val_accuracy: 0.7121 - val_loss: 0.8296
Epoch 11/50
18/18             32s 2s/step -
accuracy: 0.8856 - loss: 0.3771 - val_accuracy: 0.7576 - val_loss: 0.8223
Epoch 12/50
18/18             43s 2s/step -
accuracy: 0.8586 - loss: 0.3770 - val_accuracy: 0.6970 - val_loss: 0.8268
Epoch 13/50
18/18             32s 2s/step -
accuracy: 0.8901 - loss: 0.3375 - val_accuracy: 0.7424 - val_loss: 0.8459
Epoch 14/50
18/18             34s 2s/step -
accuracy: 0.9173 - loss: 0.2861 - val_accuracy: 0.7273 - val_loss: 0.8246
Epoch 15/50
18/18             33s 2s/step -
accuracy: 0.9243 - loss: 0.2848 - val_accuracy: 0.7273 - val_loss: 0.8387
Epoch 16/50
18/18             33s 2s/step -
accuracy: 0.9169 - loss: 0.2603 - val_accuracy: 0.7424 - val_loss: 0.8170
Epoch 17/50
18/18             42s 2s/step -
accuracy: 0.9247 - loss: 0.2583 - val_accuracy: 0.7424 - val_loss: 0.8458
Epoch 18/50
18/18             32s 2s/step -
accuracy: 0.9494 - loss: 0.2482 - val_accuracy: 0.7273 - val_loss: 0.8319
Epoch 19/50
18/18             34s 2s/step -
accuracy: 0.9384 - loss: 0.2405 - val_accuracy: 0.7273 - val_loss: 0.8470
```

```
Epoch 20/50
18/18             33s 2s/step -
accuracy: 0.9329 - loss: 0.2501 - val_accuracy: 0.7273 - val_loss: 0.8553
Epoch 21/50
18/18             33s 2s/step -
accuracy: 0.9472 - loss: 0.2077 - val_accuracy: 0.7273 - val_loss: 0.8655
Epoch 22/50
18/18             33s 2s/step -
accuracy: 0.9638 - loss: 0.1754 - val_accuracy: 0.7121 - val_loss: 0.8688
Epoch 23/50
18/18             32s 2s/step -
accuracy: 0.9592 - loss: 0.1744 - val_accuracy: 0.6970 - val_loss: 0.8748
Epoch 24/50
18/18             34s 2s/step -
accuracy: 0.9639 - loss: 0.1661 - val_accuracy: 0.6970 - val_loss: 0.8861
Epoch 25/50
18/18             32s 2s/step -
accuracy: 0.9624 - loss: 0.1684 - val_accuracy: 0.6970 - val_loss: 0.8793
Epoch 26/50
18/18             35s 2s/step -
accuracy: 0.9533 - loss: 0.1727 - val_accuracy: 0.7121 - val_loss: 0.8895
Epoch 27/50
18/18             32s 2s/step -
accuracy: 0.9512 - loss: 0.1604 - val_accuracy: 0.7121 - val_loss: 0.8786
Epoch 28/50
18/18             34s 2s/step -
accuracy: 0.9782 - loss: 0.1384 - val_accuracy: 0.6970 - val_loss: 0.9275
Epoch 29/50
18/18             32s 2s/step -
accuracy: 0.9684 - loss: 0.1503 - val_accuracy: 0.6818 - val_loss: 0.8914
Epoch 30/50
18/18             34s 2s/step -
accuracy: 0.9759 - loss: 0.1614 - val_accuracy: 0.6818 - val_loss: 0.9105
Epoch 31/50
18/18             32s 2s/step -
accuracy: 0.9604 - loss: 0.1487 - val_accuracy: 0.6818 - val_loss: 0.9225
Epoch 32/50
18/18             41s 2s/step -
accuracy: 0.9674 - loss: 0.1495 - val_accuracy: 0.6818 - val_loss: 0.9368
Epoch 33/50
18/18             34s 2s/step -
accuracy: 0.9747 - loss: 0.1322 - val_accuracy: 0.6818 - val_loss: 0.9198
Epoch 34/50
18/18             32s 2s/step -
accuracy: 0.9821 - loss: 0.1187 - val_accuracy: 0.6818 - val_loss: 0.9396
Epoch 35/50
18/18             34s 2s/step -
accuracy: 0.9714 - loss: 0.1399 - val_accuracy: 0.6818 - val_loss: 0.9474
```

```
Epoch 36/50
18/18             39s 2s/step -
accuracy: 0.9673 - loss: 0.1255 - val_accuracy: 0.6970 - val_loss: 0.9657
Epoch 37/50
18/18             34s 2s/step -
accuracy: 0.9785 - loss: 0.1126 - val_accuracy: 0.6970 - val_loss: 0.9719
Epoch 38/50
18/18             40s 2s/step -
accuracy: 0.9790 - loss: 0.1068 - val_accuracy: 0.6818 - val_loss: 0.9505
Epoch 39/50
18/18             41s 2s/step -
accuracy: 0.9747 - loss: 0.1179 - val_accuracy: 0.6970 - val_loss: 0.9616
Epoch 40/50
18/18             34s 2s/step -
accuracy: 0.9873 - loss: 0.0924 - val_accuracy: 0.6970 - val_loss: 0.9743
Epoch 41/50
18/18             41s 2s/step -
accuracy: 0.9796 - loss: 0.0986 - val_accuracy: 0.6818 - val_loss: 0.9722
Epoch 42/50
18/18             32s 2s/step -
accuracy: 0.9865 - loss: 0.0917 - val_accuracy: 0.6970 - val_loss: 0.9839
Epoch 43/50
18/18             34s 2s/step -
accuracy: 0.9819 - loss: 0.1207 - val_accuracy: 0.6667 - val_loss: 0.9935
Epoch 44/50
18/18             32s 2s/step -
accuracy: 0.9733 - loss: 0.1034 - val_accuracy: 0.6970 - val_loss: 1.0070
Epoch 45/50
18/18             34s 2s/step -
accuracy: 0.9750 - loss: 0.1114 - val_accuracy: 0.6667 - val_loss: 0.9662
Epoch 46/50
18/18             32s 2s/step -
accuracy: 0.9747 - loss: 0.1037 - val_accuracy: 0.6970 - val_loss: 0.9844
Epoch 47/50
18/18             34s 2s/step -
accuracy: 0.9863 - loss: 0.0856 - val_accuracy: 0.6818 - val_loss: 0.9727
Epoch 48/50
18/18             32s 2s/step -
accuracy: 0.9775 - loss: 0.0881 - val_accuracy: 0.6667 - val_loss: 0.9878
Epoch 49/50
18/18             34s 2s/step -
accuracy: 0.9961 - loss: 0.0744 - val_accuracy: 0.6818 - val_loss: 0.9937
Epoch 50/50
18/18             32s 2s/step -
accuracy: 0.9809 - loss: 0.0915 - val_accuracy: 0.6818 - val_loss: 0.9774
```

[40]: <keras.src.callbacks.history.History at 0x7f46d3f82230>

[42]:
```
# # Testing Evaluation
xception_test_loss, xception_test_acc = xception.evaluate(test_generator,␣
↪verbose =1)


print(f"\nXception Baseline Testing Loss     : {xception_test_loss}.")
print(f"Xception Baseline Testing Accuracy : {xception_test_acc}.")
```

```
6/6             9s 1s/step -
accuracy: 0.7470 - loss: 1.0776

Xception Baseline Testing Loss     : 0.7922331094741821.
Xception Baseline Testing Accuracy : 0.7837837934494019.
```

[43]:
```
def build_model(hp):

    # Define all hyperparms
    xception = Xception(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3),␣
↪weights='imagenet', include_top=False)
    xception.trainable = False
    n_layers = hp.Choice('n_layers', [2, 4])
    dropout_rate = hp.Choice('rate', [0.4, 0.7])
    n_units = hp.Choice('units', [256, 512])
    learning_rate = hp.Choice('lr', [LEARNING_RATE, LEARNING_RATE * 0.1,␣
↪LEARNING_RATE*0.01])

    # Mode architecture
    model = Sequential([
        xception,
        GlobalAveragePooling2D(),
    ])

    # Add hidden/top layers
    for _ in range(n_layers):
        model.add(Dense(n_units, activation='relu',␣
↪kernel_initializer='he_normal'))

    # Add Dropout Layer
    model.add(Dropout(dropout_rate))

    # Output Layer
    model.add(Dense(NUM_CLASSES, activation='softmax'))

    # Compile the model
    model.compile(
        loss=LOSS,
        optimizer = Adam(learning_rate),
```

```
        metrics = METRICS
    )
    # Return model
    return model
```

[44]:
```
# Model Hypertunning
!pip install -q keras_tuner
cls()
import keras_tuner as kt
```

[45]:
```
# Initialize Random Searcher
random_searcher = kt.RandomSearch(
    hypermodel=build_model,
    objective='val_loss',
    max_trials=10,
    seed=42,
    project_name="XceptionSearch",
    loss=LOSS)

# Start Searching
search = random_searcher.search(
    train_generator,
    validation_data= validation_generator,
    epochs = 10,
    batch_size = BATCH_SIZE
)
```

```
Trial 5 Complete [00h 05m 54s]
val_loss: 1.540732741355896

Best val_loss So Far: 0.8141117691993713
Total elapsed time: 00h 29m 33s

Search: Running Trial #6

Value             |Best Value So Far |Hyperparameter
4                 |4                 |n_layers
0.4               |0.4               |rate
256               |256               |units
1e-05             |0.001             |lr

Epoch 1/10
18/18             40s 2s/step -
accuracy: 0.0957 - loss: 2.4916 - val_accuracy: 0.1212 - val_loss: 2.2665
Epoch 2/10
18/18             35s 2s/step -
accuracy: 0.1178 - loss: 2.4398 - val_accuracy: 0.1515 - val_loss: 2.1766
Epoch 3/10
```

```
18/18              33s 2s/step -
accuracy: 0.1466 - loss: 2.3500 - val_accuracy: 0.1970 - val_loss: 2.1050
Epoch 4/10
18/18              35s 2s/step -
accuracy: 0.1470 - loss: 2.2356 - val_accuracy: 0.3182 - val_loss: 2.0450
Epoch 5/10
 1/18              2:44 10s/step -
accuracy: 0.1250 - loss: 2.1707
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[45], line 11
      2 random_searcher = kt.RandomSearch(
      3     hypermodel=build_model,
      4     objective='val_loss',
   (…)
      7     project_name="XceptionSearch",
      8     loss=LOSS)
     10 # Start Searching
---> 11 search = random_searcher.search(
     12     train_generator,
     13     validation_data= validation_generator,
     14     epochs = 10,
     15     batch_size = BATCH_SIZE
     16 )

File /opt/conda/lib/python3.10/site-packages/keras_tuner/src/engine/base_tuner.
 ↪py:234, in BaseTuner.search(self, *fit_args, **fit_kwargs)
    231         continue
    233     self.on_trial_begin(trial)
--> 234     self._try_run_and_update_trial(trial, *fit_args, **fit_kwargs)
    235     self.on_trial_end(trial)
    236 self.on_search_end()

File /opt/conda/lib/python3.10/site-packages/keras_tuner/src/engine/base_tuner.
 ↪py:274, in BaseTuner._try_run_and_update_trial(self, trial, *fit_args,␣
 ↪**fit_kwargs)
    272 def _try_run_and_update_trial(self, trial, *fit_args, **fit_kwargs):
    273     try:
--> 274         self._run_and_update_trial(trial, *fit_args, **fit_kwargs)
    275         trial.status = trial_module.TrialStatus.COMPLETED
    276         return

File /opt/conda/lib/python3.10/site-packages/keras_tuner/src/engine/base_tuner.
 ↪py:239, in BaseTuner._run_and_update_trial(self, trial, *fit_args,␣
 ↪**fit_kwargs)
    238 def _run_and_update_trial(self, trial, *fit_args, **fit_kwargs):
--> 239     results = self.run_trial(trial, *fit_args, **fit_kwargs)
```

```
    240     if self.oracle.get_trial(trial.trial_id).metrics.exists(
    241         self.oracle.objective.name
    242     ):
    243         # The oracle is updated by calling `self.oracle.update_trial()`
 ↪in
    244         # `Tuner.run_trial()`. For backward compatibility, we support␣
 ↪this
    245         # use case. No further action needed in this case.
    246         warnings.warn(
    247             "The use case of calling "
    248             "`self.oracle.update_trial(trial_id, metrics)` "
   (…)
    254             stacklevel=2,
    255         )

File /opt/conda/lib/python3.10/site-packages/keras_tuner/src/engine/tuner.py:
 ↪314, in Tuner.run_trial(self, trial, *args, **kwargs)
    312     callbacks.append(model_checkpoint)
    313     copied_kwargs["callbacks"] = callbacks
--> 314     obj_value = self._build_and_fit_model(trial, *args, **copied_kwargs
    316     histories.append(obj_value)
    317 return histories

File /opt/conda/lib/python3.10/site-packages/keras_tuner/src/engine/tuner.py:
 ↪233, in Tuner._build_and_fit_model(self, trial, *args, **kwargs)
    231 hp = trial.hyperparameters
    232 model = self._try_build(hp)
--> 233 results = self.hypermodel.fit(hp, model, *args, **kwargs)
    235 # Save the build config for model loading later.
    236 if backend.config.multi_backend():

File /opt/conda/lib/python3.10/site-packages/keras_tuner/src/engine/hypermodel.
 ↪py:149, in HyperModel.fit(self, hp, model, *args, **kwargs)
    125 def fit(self, hp, model, *args, **kwargs):
    126     """Train the model.
    127
    128     Args:
   (…)
    147         If return a float, it should be the `objective` value.
    148     """
--> 149     return model.fit(*args, **kwargs)

File /opt/conda/lib/python3.10/site-packages/keras/src/utils/traceback_utils.py
 ↪117, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    115 filtered_tb = None
    116 try:
--> 117     return fn(*args, **kwargs)
    118 except Exception as e:
```

```
    119     filtered_tb = _process_traceback_frames(e.__traceback__)

File /opt/conda/lib/python3.10/site-packages/keras/src/backend/tensorflow/
 ↪trainer.py:320, in TensorFlowTrainer.fit(self, x, y, batch_size, epochs,␣
 ↪verbose, callbacks, validation_split, validation_data, shuffle, class_weight,␣
 ↪sample_weight, initial_epoch, steps_per_epoch, validation_steps,␣
 ↪validation_batch_size, validation_freq)
    318 for step, iterator in epoch_iterator.enumerate_epoch():
    319     callbacks.on_train_batch_begin(step)
--> 320     logs = self.train_function(iterator)
    321     logs = self._pythonify_logs(logs)
    322     callbacks.on_train_batch_end(step, logs)

File /opt/conda/lib/python3.10/site-packages/tensorflow/python/util/
 ↪traceback_utils.py:150, in filter_traceback.<locals>.error_handler(*args,␣
 ↪**kwargs)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File /opt/conda/lib/python3.10/site-packages/tensorflow/python/eager/
 ↪polymorphic_function/polymorphic_function.py:833, in Function.__call__(self,␣
 ↪*args, **kwds)
    830 compiler = "xla" if self._jit_compile else "nonXla"
    832 with OptionalXlaContext(self._jit_compile):
--> 833   result = self._call(*args, **kwds)
    835 new_tracing_count = self.experimental_get_tracing_count()
    836 without_tracing = (tracing_count == new_tracing_count)

File /opt/conda/lib/python3.10/site-packages/tensorflow/python/eager/
 ↪polymorphic_function/polymorphic_function.py:878, in Function._call(self,␣
 ↪*args, **kwds)
    875 self._lock.release()
    876 # In this case we have not created variables on the first call. So we ca
    877 # run the first trace but we should fail if variables are created.
--> 878 results = tracing_compilation.call_function(
    879     args, kwds, self._variable_creation_config
    880 )
    881 if self._created_variables:
    882   raise ValueError("Creating variables on a non-first call to a functio"
    883                    " decorated with tf.function.")

File /opt/conda/lib/python3.10/site-packages/tensorflow/python/eager/
 ↪polymorphic_function/tracing_compilation.py:139, in call_function(args,␣
 ↪kwargs, tracing_options)
    137 bound_args = function.function_type.bind(*args, **kwargs)
    138 flat_inputs = function.function_type.unpack_inputs(bound_args)
```

```
--> 139 return function._call_flat(  # pylint: disable=protected-access
    140     flat_inputs, captured_inputs=function.captured_inputs
    141 )

File /opt/conda/lib/python3.10/site-packages/tensorflow/python/eager/
 ↪polymorphic_function/concrete_function.py:1322, in ConcreteFunction.
 ↪_call_flat(self, tensor_inputs, captured_inputs)
   1318 possible_gradient_type = gradients_util.PossibleTapeGradientTypes(args)
   1319 if (possible_gradient_type == gradients_util.POSSIBLE_GRADIENT_TYPES_NO E
   1320     and executing_eagerly):
   1321   # No tape is watching; skip to running the function.
-> 1322   return self._inference_function.call_preflattened(args)
   1323 forward_backward = self._select_forward_and_backward_functions(
   1324     args,
   1325     possible_gradient_type,
   1326     executing_eagerly)
   1327 forward_function, args_with_tangents = forward_backward.forward()

File /opt/conda/lib/python3.10/site-packages/tensorflow/python/eager/
 ↪polymorphic_function/atomic_function.py:216, in AtomicFunction.
 ↪call_preflattened(self, args)
    214 def call_preflattened(self, args: Sequence[core.Tensor]) -> Any:
    215   """Calls with flattened tensor inputs and returns the structured␣
 ↪output."""
--> 216   flat_outputs = self.call_flat(*args)
    217   return self.function_type.pack_output(flat_outputs)

File /opt/conda/lib/python3.10/site-packages/tensorflow/python/eager/
 ↪polymorphic_function/atomic_function.py:251, in AtomicFunction.call_flat(self ␣
 ↪*args)
    249 with record.stop_recording():
    250   if self._bound_context.executing_eagerly():
--> 251     outputs = self._bound_context.call_function(
    252         self.name,
    253         list(args),
    254         len(self.function_type.flat_outputs),
    255     )
    256   else:
    257     outputs = make_call_op_in_graph(
    258         self,
    259         list(args),
    260         self._bound_context.function_call_options.as_attrs(),
    261     )

File /opt/conda/lib/python3.10/site-packages/tensorflow/python/eager/context.py
 ↪1552, in Context.call_function(self, name, tensor_inputs, num_outputs)
   1550 cancellation_context = cancellation.context()
   1551 if cancellation_context is None:
```

```
-> 1552   outputs = execute.execute(
   1553       name.decode("utf-8"),
   1554       num_outputs=num_outputs,
   1555       inputs=tensor_inputs,
   1556       attrs=attrs,
   1557       ctx=self,
   1558   )
   1559 else:
   1560   outputs = execute.execute_with_cancellation(
   1561       name.decode("utf-8"),
   1562       num_outputs=num_outputs,
   (…)
   1566       cancellation_manager=cancellation_context,
   1567   )

File /opt/conda/lib/python3.10/site-packages/tensorflow/python/eager/execute.py
 ↪53, in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51 try:
     52   ctx.ensure_initialized()
---> 53   tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name
     54                                       inputs, attrs, num_outputs)
     55 except core._NotOkStatusException as e:
     56   if name is not None:

KeyboardInterrupt:
```

[47]:
```python
# Best Hyper params
best_hps = random_searcher.get_best_hyperparameters()[0]
print(f"Best Hyper Params founded: {best_hps.values}\n")

# Build the best model
xception_model = build_model(best_hps)
xception_model.summary()

# Compile the model
print("\nTraining Best Model Architecture : ")
xception_model_history = xception_model.fit(
    train_generator,
    validation_data= validation_generator,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=[
        EarlyStopping(patience=3, restore_best_weights=True),
        ModelCheckpoint('BestXception.keras', save_best_only=True)
    ]
)
```

```
Best Hyper Params founded: {'n_layers': 4, 'rate': 0.4, 'units': 256, 'lr':
0.001}
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
xception (Functional)	(None, 5, 5, 2048)	20,861,480
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 2048)	0
dense_10 (Dense)	(None, 256)	524,544
dense_11 (Dense)	(None, 256)	65,792
dense_12 (Dense)	(None, 256)	65,792
dense_13 (Dense)	(None, 256)	65,792
dropout_2 (Dropout)	(None, 256)	0
dense_14 (Dense)	(None, 9)	2,313

Total params: 21,585,713 (82.34 MB)

Trainable params: 724,233 (2.76 MB)

Non-trainable params: 20,861,480 (79.58 MB)

```
Training Best Model Architecture :
Epoch 1/50
18/18             39s 2s/step -
accuracy: 0.2693 - loss: 2.0857 - val_accuracy: 0.5909 - val_loss: 1.1886
Epoch 2/50
18/18             35s 2s/step -
accuracy: 0.6495 - loss: 0.9705 - val_accuracy: 0.6970 - val_loss: 0.9742
Epoch 3/50
18/18             35s 2s/step -
accuracy: 0.7999 - loss: 0.5952 - val_accuracy: 0.6970 - val_loss: 0.9012
Epoch 4/50
18/18             32s 2s/step -
```

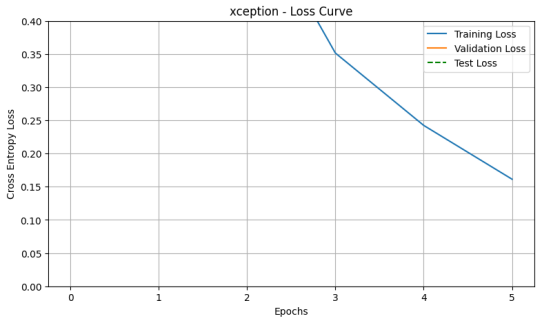
```
accuracy: 0.8968 - loss: 0.3326 - val_accuracy: 0.7121 - val_loss: 1.1334
Epoch 5/50
18/18             34s 2s/step -
accuracy: 0.9377 - loss: 0.2159 - val_accuracy: 0.5909 - val_loss: 1.3910
Epoch 6/50
18/18             32s 2s/step -
accuracy: 0.9534 - loss: 0.1507 - val_accuracy: 0.6061 - val_loss: 1.2700
```

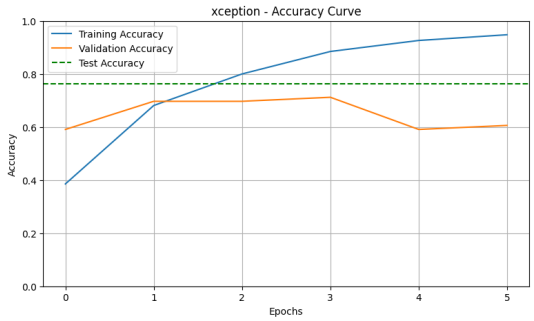
[49]:
```
test_loss, test_acc = xception_model.evaluate(test_generator)
print("Test Loss     : {:.4} | Baseline : {:.4}".format(test_loss,␣
 ↪xception_test_loss))
print("Test Accuracy: {:.4}% | Baseline : {:.4}%".format(test_acc*100,␣
 ↪xception_test_acc*100))
```

```
6/6             11s 2s/step -
accuracy: 0.7533 - loss: 0.7471
Test Loss     : 0.7798 | Baseline : 0.7922
Test Accuracy: 76.22% | Baseline : 78.38%
```

[50]:
```
# Collect the history of the training run
xception_model_history = pd.DataFrame(xception_model_history.history)

# Create a figure to display the model's performance
plt.figure(figsize=(20, 5))

# Plot the loss curve in the first subplot
plt.subplot(1, 2, 1)
plt.title("xception - Loss Curve")
plt.plot(xception_model_history['loss'], label="Training Loss")
plt.plot(xception_model_history['val_loss'], label="Validation Loss")

# Horizontal line to show the testing performance
plt.axhline(y=test_loss, label="Test Loss", linestyle='--', color='green')

# Set the x- and y-labels, and the x- and y-limits
plt.xlabel("Epochs")
plt.ylabel("Cross Entropy Loss")
plt.ylim([0, 0.4])

# Show the legend and grid
plt.legend()
plt.grid()

# Plot the accuracy curve in the second subplot
plt.subplot(1, 2, 2)
plt.title("xception - Accuracy Curve")
plt.plot(xception_model_history['accuracy'], label="Training Accuracy")
plt.plot(xception_model_history['val_accuracy'], label="Validation Accuracy")
```

```
# Horizontal line to show the testing performance
plt.axhline(y=test_acc, label="Test Accuracy", linestyle='--', color='green')

# Set the x- and y-labels, and the x- and y-limits
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
# plt.ylim([0.85, 1])
plt.ylim([0, 1])


# Show the legend and grid
plt.legend()
plt.grid()

# Display the plot
plt.show()
```





[51]:
```
# assigning label names to the corresponding indexes
labels = {0: 'Fish', 1: 'Goldfish', 2: 'Harbor seal', 3:'Jellyfish', 4:␣
 ↪'Lobster', 5:'Oyster',6:'Sea turtle',7:'Squid', 8:'Starfish'}
```

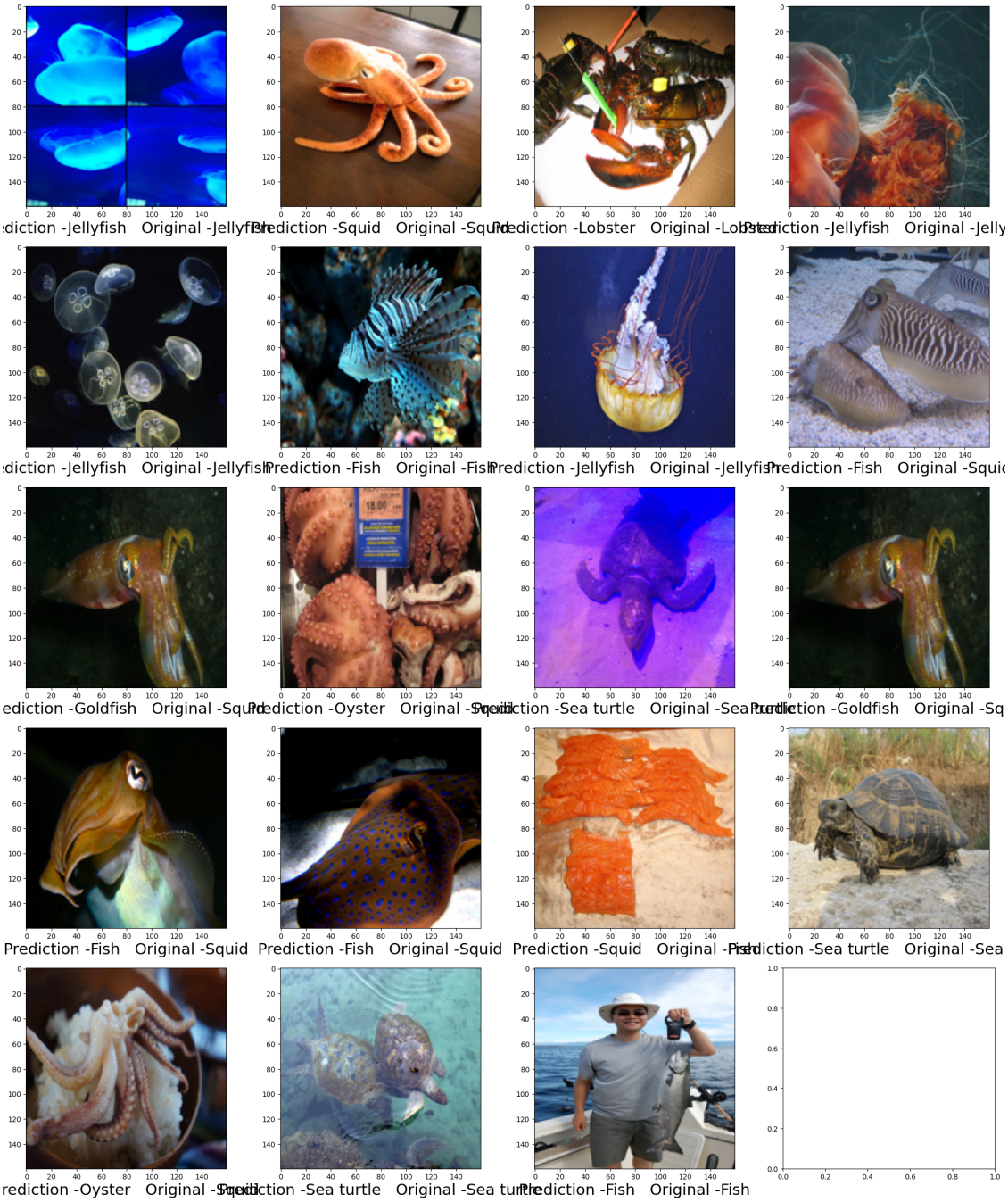
[1]:
```
# loading images and their predictions

from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score
# import cv2

prediction = []
original = []
image = []
count = 0
for i in os.listdir(ORIGINAL_TEST_DIR):
    for item in os.listdir(os.path.join(ORIGINAL_TEST_DIR,i)):
        #code to open the image
```

```
        img= PIL.Image.open(os.path.join(ORIGINAL_TEST_DIR,i,item))
        #resizing the image to (256,256)
        img = img.resize((IMAGE_SIZE,IMAGE_SIZE))
        #appending image to the image list
        image.append(img)
        #converting image to array
        img = np.asarray(img, dtype= np.float32)
        #normalizing the image
        img = img / 255
        #reshaping the image in to a 4D array
        img = img.reshape(-1,IMAGE_SIZE,IMAGE_SIZE,3)
        #making prediction of the model
        predict = model.predict(img)
        #getting the index corresponding to the highest value in the prediction
        predict = np.argmax(predict)
        #appending the predicted class to the list
        prediction.append(labels[predict])
        #appending original class to the list
        original.append(i)

        cls()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 10
      8 image = []
      9 count = 0
---> 10 for i in os.listdir(ORIGINAL_TEST_DIR):
     11     for item in os.listdir(os.path.join(ORIGINAL_TEST_DIR,i)):
     12         #code to open the image
     13         img= PIL.Image.open(os.path.join(ORIGINAL_TEST_DIR,i,item))

NameError: name 'os' is not defined
```

[66]:
```
# visualizing the results
import random
fig=plt.figure(figsize = (25,30))
for i in range(20):
    j = random.randint(0,len(image))
    fig.add_subplot(5,4,i+1)
    fig.suptitle('Test Predictions', fontsize=40)
    plt.xlabel("Prediction -" + prediction[j] +"   Original -" + original[j] ,␣
 ↪fontsize=22)
    plt.imshow(image[j])
fig.tight_layout()
```

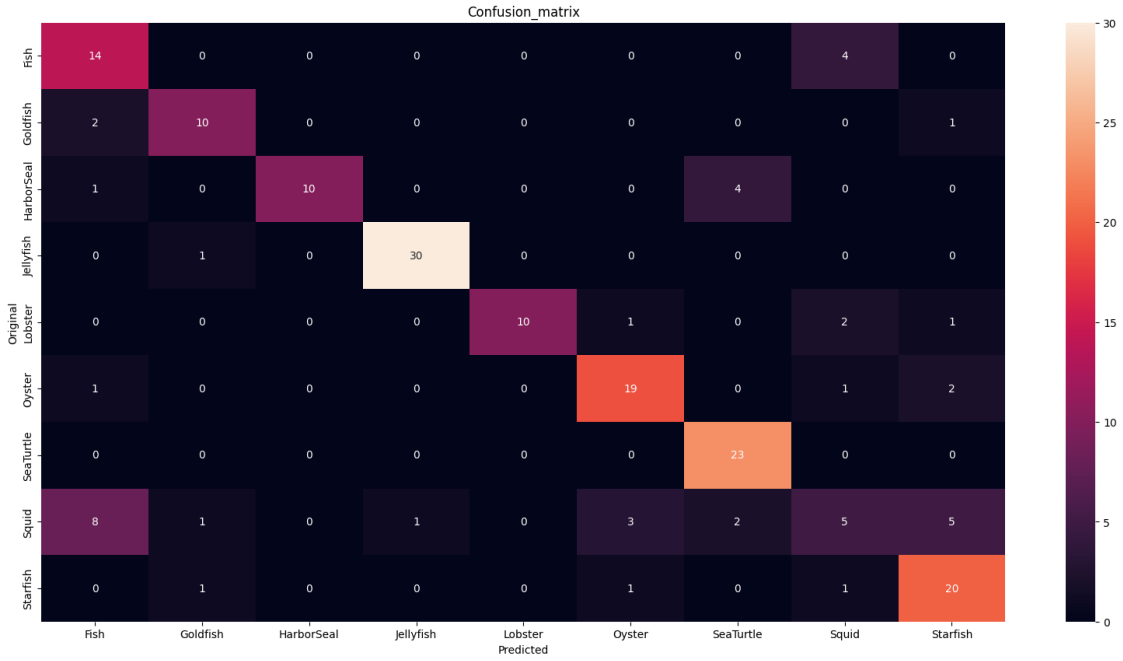
```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[66], line 8
      6     fig.add_subplot(5,4,i+1)
      7     fig.suptitle('Test Predictions', fontsize=40)
----> 8     plt.xlabel("Prediction -" + prediction[j] +"   Original -" +␣
 ↪original[j] , fontsize=22)
      9     plt.imshow(image[j])
     10 fig.tight_layout()

IndexError: list index out of range
```



[56]:
```
# plot confusion matrix
```

```
plt.figure(figsize=(20,10))

display_labels=['Fish', 'Goldfish', 'HarborSeal', 'Jellyfish', 'Lobster',␣
 ↪'Oyster', 'SeaTurtle', 'Squid', 'Starfish']
cm = confusion_matrix(np.asarray(original), np.asarray(prediction))
ax = plt.subplot()
sns.heatmap(cm, annot = True, ax = ax , xticklabels=display_labels,␣
 ↪yticklabels=display_labels)

ax.set_xlabel('Predicted')
ax.set_ylabel('Original')
ax.set_title('Confusion_matrix')
```

[56]: Text(0.5, 1.0, 'Confusion_matrix')



[ ]: