Problem Statement

Marina Pier wants to use Deep Learning techniques to build an automatic reporting system that recognizes the boat. The company is also looking to use a transfer learning approach of any lightweight pre-trained model in order to deploy in mobile devices.

As a deep learning engineer, your task is to:

1.Build a CNN network to classify the boat.

2.Build a lightweight model with the aim of deploying the solution on a mobile device using transfer learning. You can use any lightweight pre-trained model as the initial (first) layer. MobileNetV2 is a popular lightweight pre-trained model built using Keras API.

The dataset contains images of 9 types of boats. It contains a total of 1162 images. The training images are provided in the directory of the specific class itself. Classes:

- ferry_boat
- gondola
- sailboat
- cruise_ship
- kayak
- inflatable_boat
- paper_boat
- buoy
- freight_boat

Loading the Dataset

```
import PIL.Image as Image
import numpy as np
from zipfile import ZipFile
import cv2
import tensorflow as tf
#Tensorflow Hub for Transfer Learning
import tensorflow_hub as hub
from tensorflow import keras
import tf_keras
from tensorflow.keras import layers,models
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
```

```
#Upload the kaggle.json file
from google.colab import files
files.upload()
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 67 bytes, last modified: 6/16/2025 - 100% done

```
Saving kaggle.json to kaggle.json
```

```
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle datasets download -d imsparsh/dockship-boat-type-classification
```

```
Dataset URL: https://www.kaggle.com/datasets/imsparsh/dockship-boat-type-classification
License(s): CC0-1.0
Downloading dockship-boat-type-classification.zip to /content
 60% 108M/179M [00:00<00:00, 1.13GB/s]
100% 179M/179M [00:00<00:00, 876MB/s]
```

```
!unzip dockship-boat-type-classification.zip -d boat_data
```

```
  inflating: boat_data/Train/sailboat/52.jpg
```

```
   inflating: boat_data/Train/sailboat/53.jpg
   inflating: boat_data/Train/sailboat/54.jpg
   inflating: boat_data/Train/sailboat/55.jpg
   inflating: boat_data/Train/sailboat/56.jpg
   inflating: boat_data/Train/sailboat/57.jpg
   inflating: boat_data/Train/sailboat/58.jpg
   inflating: boat_data/Train/sailboat/59.jpg
   inflating: boat_data/Train/sailboat/6.jpg
   inflating: boat_data/Train/sailboat/60.jpg
   inflating: boat_data/Train/sailboat/61.jpg
   inflating: boat_data/Train/sailboat/62.jpg
   inflating: boat_data/Train/sailboat/63.jpg
   inflating: boat_data/Train/sailboat/64.jpg
   inflating: boat_data/Train/sailboat/65.jpg
   inflating: boat_data/Train/sailboat/66.jpg
   inflating: boat_data/Train/sailboat/67.jpg
   inflating: boat_data/Train/sailboat/68.jpg
   inflating: boat_data/Train/sailboat/69.jpg
   inflating: boat_data/Train/sailboat/7.jpg
   inflating: boat_data/Train/sailboat/70.jpg
   inflating: boat_data/Train/sailboat/71.jpg
   inflating: boat_data/Train/sailboat/72.jpg
   inflating: boat_data/Train/sailboat/73.jpg
   inflating: boat_data/Train/sailboat/74.jpg
   inflating: boat_data/Train/sailboat/75.jpg
   inflating: boat_data/Train/sailboat/76.jpg
   inflating: boat_data/Train/sailboat/77.jpg
   inflating: boat_data/Train/sailboat/78.jpg
   inflating: boat_data/Train/sailboat/79.jpg
   inflating: boat_data/Train/sailboat/8.jpg
   inflating: boat_data/Train/sailboat/80.jpg
   inflating: boat_data/Train/sailboat/81.jpg
   inflating: boat_data/Train/sailboat/82.jpg
   inflating: boat_data/Train/sailboat/83.jpg
   inflating: boat_data/Train/sailboat/84.jpg
   inflating: boat_data/Train/sailboat/85.jpg
   inflating: boat_data/Train/sailboat/86.jpg
   inflating: boat_data/Train/sailboat/87.jpg
   inflating: boat_data/Train/sailboat/88.jpg
   inflating: boat_data/Train/sailboat/89.jpg
   inflating: boat_data/Train/sailboat/9.jpg
   inflating: boat_data/Train/sailboat/90.jpg
   inflating: boat_data/Train/sailboat/91.jpg
   inflating: boat_data/Train/sailboat/92.jpg
   inflating: boat_data/Train/sailboat/93.jpg
   inflating: boat_data/Train/sailboat/94.jpg
   inflating: boat_data/Train/sailboat/95.jpg
   inflating: boat_data/Train/sailboat/96.jpg
   inflating: boat_data/Train/sailboat/97.jpg
   inflating: boat_data/Train/sailboat/98.jpg
   inflating: boat_data/Train/sailboat/99.jpg
   inflating: boat_data/sample_submission.csv
```

```
!ls -lh /content/
```

```
total 180M
drwxr-xr-x 4 root root 4.0K Jun 17 07:59 boat_data
-rw-r--r-- 1 root root 180M Feb  6  2021 dockship-boat-type-classification.zip
drwxr-xr-x 1 root root 4.0K Jun 13 13:36 sample_data
```

```
!ls boat_data
```

```
sample_submission.csv  TEST  Train
```

1. Build a CNN network to classify the boat.

1.1.Split the dataset into train and test in the ratio 80:20, with shuffle and random state=43.

```
# Make a new subfolder called "Unknown" inside TEST
!mkdir -p /content/boat_data/TEST/Test
```

```
# Move all image files into that folder (you can add more formats if needed)
!mv /content/boat_data/TEST/*.jpg /content/boat_data/TEST/Test/
```

1.2.Use tf.keras.preprocessing.image_dataset_from_directory to load the train and test datasets. This function also supports data normalization.

1.3.Load train, validation and test dataset in batches of 32 using the function initialized in the above step.

```
import pathlib
# Set paths
train_path = pathlib.Path("/content/boat_data/Train")
test_path = pathlib.Path("/content/boat_data/TEST")

# Load training dataset
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_path,
    image_size=(180, 180),
    batch_size=32,
    label_mode='categorical'  # Use 'categorical' if more than 2 classes
)

# Load testing dataset
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    test_path,
    image_size=(180, 180),
    batch_size=32,
    label_mode='categorical'
)
```

```
Found 1162 files belonging to 9 classes.
Found 300 files belonging to 1 classes.
```

Data Normalization of Train and Test Images

```
!ls /content/boat_data/Train
```

```
buoy         ferry_boat    gondola          kayak       sailboat
cruise_ship  freight_boat  inflatable_boat  paper_boat
```

```
from tensorflow.keras.layers import Rescaling

# Example: Add rescaling to training dataset
normalization_layer = Rescaling(1./255)

train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds = test_ds.map(lambda x, y: (normalization_layer(x), y))
```

train_ds and test_ds are tensors now

```
image_count = 0
for batch in train_ds:
    image_count += batch[0].shape[0]  # batch[0] is the image batch
print(f"Total images in train_ds: {image_count}")

image_count = 0
for batch in test_ds:
    image_count += batch[0].shape[0]
print(f"Total images in test_ds: {image_count}")
```

```
Total images in train_ds: 1162
Total images in test_ds: 300
```

See the images

```
!ls /content/boat_data/Train/
```

```
buoy         ferry_boat    gondola          kayak       sailboat
cruise_ship  freight_boat  inflatable_boat  paper_boat
```

```
img_path = '/content/boat_data/Train/ferry_boat/1.jpg'
img = Image.open(img_path)

plt.imshow(img)
plt.title("ferry_boat Image")
plt.axis("off")
plt.show()
```

ferry_boat Image

```
img_path = '/content/boat_data/Train/buoy/1.jpg'
img = Image.open(img_path)

plt.imshow(img)
plt.title("Buoy Image")
plt.axis("off")
plt.show()
```

Buoy Image

if your goal is to manually split and manipulate the data into X_train, y_train, X_test, and y_test (e.g., for passing into scikit-learn models or doing custom training/testing), then you'll need to convert the TensorFlow train_ds and test_ds datasets into NumPy arrays.

```
import numpy as np

# Function to convert tf.data.Dataset to NumPy arrays
def dataset_to_numpy(ds):
    X = []
    y = []
    for images, labels in ds:
        X.append(images.numpy())
        y.append(labels.numpy())
    return np.concatenate(X), np.concatenate(y)

# Convert training dataset
X_train, y_train = dataset_to_numpy(train_ds)

# Convert test dataset
X_test, y_test = dataset_to_numpy(test_ds)

# Check shapes
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (1162, 180, 180, 3)
y_train shape: (1162, 9)
X_test shape: (300, 180, 180, 3)
y_test shape: (300, 1)
```

Inferences

- We can infer that train data is 1162 images and test data is 300 images. All images are of size 180*180 and they are color images(RGB).
- Train data have 9 classes and Test data has only 1 class

```
y_train
```

```
array([[0., 0., 1., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

1.4.Build a CNN network using Keras with the following layers

1. Cov2D with 32 filters, kernel size 3,3, and activation relu, followed by MaxPool2D
2. Cov2D with 32 filters, kernel size 3,3, and activation relu, followed by MaxPool2D
3. GLobalAveragePooling2D layer
4. Dense layer with 128 neurons and activation relu
5. Dense layer with 128 neurons and activation relu
6. Dense layer with 9 neurons and activation softmax.

```
boat_cnn=models.Sequential([
    #Cov2D with 32 filters, kernel size 3,3, and activation relu, followed by MaxPool2D
    layers.Conv2D(input_shape=(180,180,3),filters=32,kernel_size=(3,3),activation='relu'),
    layers.MaxPool2D((2,2)),

    #Cov2D with 32 filters, kernel size 3,3, and activation relu, followed by MaxPool2D
    layers.Conv2D(filters=32,kernel_size=(3,3),activation='relu'),
    layers.MaxPool2D((2,2)),

    #GLobalAveragePooling2D layer
    layers.AveragePooling2D((2,2)),

    layers.Flatten(),

    #Dense layer with 128 neurons and activation relu
    layers.Dense(128,activation='relu'),

    #Dense layer with 128 neurons and activation relu
    layers.Dense(128,activation='relu'),

    #Dense layer with 9 neurons and activation softmax.
    layers.Dense(9,activation='softmax'),

])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
boat_cnn.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_1 (Conv2D)	(None, 87, 87, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 43, 43, 32)	0
average_pooling2d (AveragePooling2D)	(None, 21, 21, 32)	0
flatten (Flatten)	(None, 14112)	0
dense (Dense)	(None, 128)	1,806,464
dense_1 (Dense)	(None, 128)	16,512
dense_2 (Dense)	(None, 9)	1,161

Total params: 1,834,281 (7.00 MB)

1.5 Compile the model with Adam optimizer, categorical_crossentropy loss, and with metrics accuracy, precision, and recall.

```
boat_cnn.compile(optimizer='adam',
              loss='categorical_crossentropy',  # or 'sparse_categorical_crossentropy'
              metrics=['accuracy','precision','recall'])
```

1.6 Train the model for 20 epochs and plot training loss and accuracy against epochs.

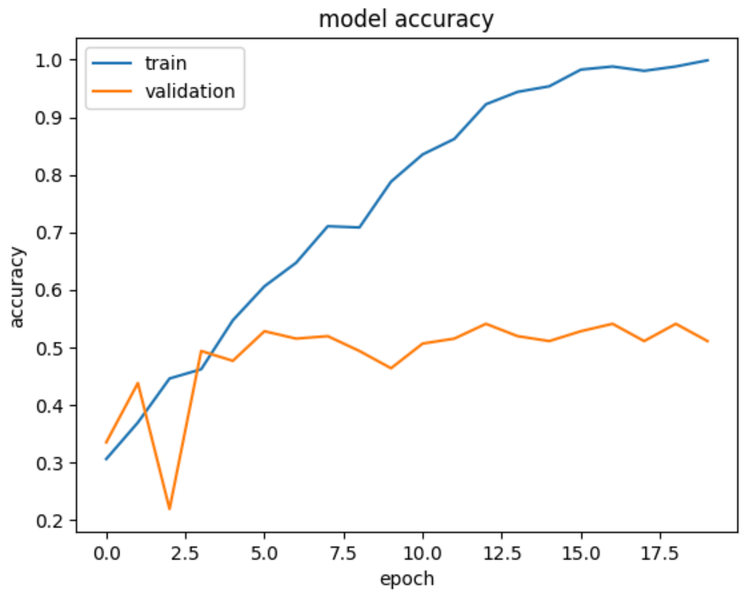
```
boat_cnn_training=boat_cnn.fit(X_train,y_train,epochs=20,validation_split=0.2)
```

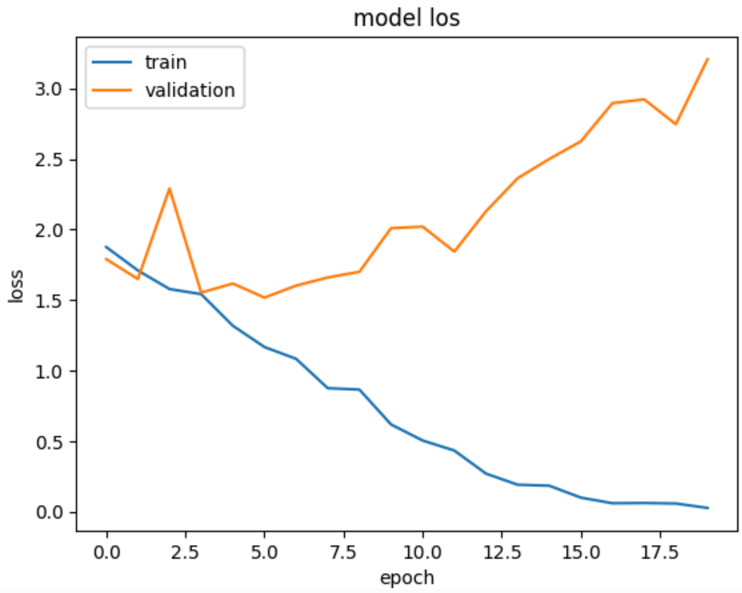
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - accuracy: 0.2841 - loss: 1.9596 - precision: 0.5400 - recall: 0.0226 - val_accuracy: 0.334
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - accuracy: 0.3345 - loss: 1.7244 - precision: 0.5395 - recall: 0.0375 - val_accuracy: 0.437
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.4333 - loss: 1.6084 - precision: 0.6482 - recall: 0.1818 - val_accuracy: 0.218
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.3955 - loss: 1.6897 - precision: 0.4978 - recall: 0.1500 - val_accuracy: 0.493
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - accuracy: 0.5490 - loss: 1.3267 - precision: 0.7172 - recall: 0.3574 - val_accuracy: 0.476
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.5807 - loss: 1.2093 - precision: 0.7482 - recall: 0.4272 - val_accuracy: 0.527
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.6369 - loss: 1.1296 - precision: 0.8134 - recall: 0.4248 - val_accuracy: 0.515
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.6798 - loss: 0.9511 - precision: 0.8057 - recall: 0.5600 - val_accuracy: 0.519
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - accuracy: 0.7030 - loss: 0.8782 - precision: 0.8183 - recall: 0.5729 - val_accuracy: 0.493
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.7688 - loss: 0.6544 - precision: 0.8946 - recall: 0.6957 - val_accuracy: 0.463
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 45s 1s/step - accuracy: 0.8173 - loss: 0.5402 - precision: 0.8906 - recall: 0.7367 - val_accuracy: 0.506
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - accuracy: 0.8583 - loss: 0.4448 - precision: 0.9137 - recall: 0.8085 - val_accuracy: 0.515
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 35s 1s/step - accuracy: 0.9199 - loss: 0.2900 - precision: 0.9610 - recall: 0.8747 - val_accuracy: 0.540
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 36s 1s/step - accuracy: 0.9519 - loss: 0.1787 - precision: 0.9778 - recall: 0.9274 - val_accuracy: 0.519
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - accuracy: 0.9588 - loss: 0.1915 - precision: 0.9800 - recall: 0.9257 - val_accuracy: 0.510
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - accuracy: 0.9796 - loss: 0.1231 - precision: 0.9888 - recall: 0.9595 - val_accuracy: 0.527
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - accuracy: 0.9882 - loss: 0.0638 - precision: 0.9925 - recall: 0.9849 - val_accuracy: 0.540
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - accuracy: 0.9888 - loss: 0.0471 - precision: 0.9945 - recall: 0.9868 - val_accuracy: 0.510
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 35s 1s/step - accuracy: 0.9850 - loss: 0.0590 - precision: 0.9897 - recall: 0.9802 - val_accuracy: 0.540
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - accuracy: 0.9998 - loss: 0.0276 - precision: 0.9998 - recall: 0.9978 - val_accuracy: 0.510
```

```
import matplotlib.pyplot as plt
%matplotlib inline
# list all data in training , training has a dictionary called history and its printing all the keys in history dictionary
```

```
print(boat_cnn_training.history.keys())

# summarize training for accuracy
plt.plot(boat_cnn_training.history['accuracy']) # training accuracy values
plt.plot(boat_cnn_training.history['val_accuracy']) #validation accuracy values
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# summarize traning for loss
plt.plot(boat_cnn_training.history['loss']) # training loss values
plt.plot(boat_cnn_training.history['val_loss']) #validation loss values
plt.title('model los')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
dict_keys(['accuracy', 'loss', 'precision', 'recall', 'val_accuracy', 'val_loss', 'val_precision', 'val_recall'])
```





1.7.Evaluate the model on test images and print the test loss and accuracy.

```
for images, labels in test_ds.take(1):
    print("Image batch shape:", images.shape)
    print("Label batch shape:", labels.shape)
```

```
Image batch shape: (32, 180, 180, 3)
Label batch shape: (32, 1)
```

```
import numpy as np

# Extract images and labels from the dataset
X_test = []
y_test = []

for images, labels in test_ds:
    X_test.append(images.numpy())
    y_test.append(labels.numpy())

# Convert to single NumPy arrays
X_test = np.concatenate(X_test)
y_test = np.concatenate(y_test)

print(X_test.shape)  # (N, 180, 180, 3)
print(y_test.shape)  # (N,) or (N, 9) depending on label_mode
```

```
(300, 180, 180, 3)
(300, 1)
```

```
y_test = y_test.squeeze()
y_test
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.], dtype=float32)
```

```
boat_cnn.evaluate(X_test,y_test)
```

```
----> 1 boat_cnn.evaluate(X_test,y_test)

1 frames
/usr/local/lib/python3.11/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs,
ctx, name)
     57       e.message += " name: " + name
     58     raise core._status_to_exception(e) from None
---> 59   except TypeError as e:
     60     keras_symbolic_tensors = [x for x in inputs if _is_keras_symbolic_tensor(x)]
     61     if keras_symbolic_tensors:

InvalidArgumentError: Graph execution error:

Detected at node LogicalAnd_1 defined at (most recent call last):
  File "<frozen runpy>", line 198, in _run_module_as_main

  File "<frozen runpy>", line 88, in _run_code

  File "/usr/local/lib/python3.11/dist-packages/colab_kernel_launcher.py", line 37, in <module>

  File "/usr/local/lib/python3.11/dist-packages/traitlets/config/application.py", line 992, in launch_instance

  File "/usr/local/lib/python3.11/dist-packages/ipykernel/kernelapp.py", line 712, in start

  File "/usr/local/lib/python3.11/dist-packages/tornado/platform/asyncio.py", line 205, in start

  File "/usr/lib/python3.11/asyncio/base_events.py", line 608, in run_forever

  File "/usr/lib/python3.11/asyncio/base_events.py", line 1936, in _run_once

  File "/usr/lib/python3.11/asyncio/events.py", line 84, in _run

  File "/usr/local/lib/python3.11/dist-packages/ipykernel/kernelbase.py", line 510, in dispatch_queue

  File "/usr/local/lib/python3.11/dist-packages/ipykernel/kernelbase.py", line 499, in process_one

  File "/usr/local/lib/python3.11/dist-packages/ipykernel/kernelbase.py", line 406, in dispatch_shell

  File "/usr/local/lib/python3.11/dist-packages/ipykernel/kernelbase.py", line 730, in execute_request

  File "/usr/local/lib/python3.11/dist-packages/ipykernel/ipkernel.py", line 383, in do_execute

  File "/usr/local/lib/python3.11/dist-packages/ipykernel/zmqshell.py", line 528, in run_cell

  File "/usr/local/lib/python3.11/dist-packages/IPython/core/interactiveshell.py", line 2975, in run_cell

  File "/usr/local/lib/python3.11/dist-packages/IPython/core/interactiveshell.py", line 3030, in _run_cell

  File "/usr/local/lib/python3.11/dist-packages/IPython/core/async_helpers.py", line 78, in _pseudo_sync_runner

  File "/usr/local/lib/python3.11/dist-packages/IPython/core/interactiveshell.py", line 3257, in run_cell_async

  File "/usr/local/lib/python3.11/dist-packages/IPython/core/interactiveshell.py", line 3473, in run_ast_nodes

  File "/usr/local/lib/python3.11/dist-packages/IPython/core/interactiveshell.py", line 3553, in run_code

  File "<ipython-input-24-4182366883>", line 1, in <cell line: 0>

  File "/usr/local/lib/python3.11/dist-packages/keras/src/utils/traceback_utils.py", line 117, in error_handler

  File "/usr/local/lib/python3.11/dist-packages/keras/src/backend/tensorflow/trainer.py", line 484, in evaluate

  File "/usr/local/lib/python3.11/dist-packages/keras/src/backend/tensorflow/trainer.py", line 219, in function

  File "/usr/local/lib/python3.11/dist-packages/keras/src/backend/tensorflow/trainer.py", line 132, in multi_step_on_iterator

  File "/usr/local/lib/python3.11/dist-packages/keras/src/backend/tensorflow/trainer.py", line 113, in one_step_on_data

  File "/usr/local/lib/python3.11/dist-packages/keras/src/backend/tensorflow/trainer.py", line 99, in test_step

  File "/usr/local/lib/python3.11/dist-packages/keras/src/trainers/trainer.py", line 490, in compute_metrics

  File "/usr/local/lib/python3.11/dist-packages/keras/src/trainers/compile_utils.py", line 334, in update_state

  File "/usr/local/lib/python3.11/dist-packages/keras/src/trainers/compile_utils.py", line 21, in update_state

  File "/usr/local/lib/python3.11/dist-packages/keras/src/metrics/confusion_metrics.py", line 378, in update_state

  File "/usr/local/lib/python3.11/dist-packages/keras/src/metrics/metrics_utils.py", line 592, in
update_confusion_matrix_variables

  File "/usr/local/lib/python3.11/dist-packages/keras/src/metrics/metrics_utils.py", line 565, in weighted_assign_add

  File "/usr/local/lib/python3.11/dist-packages/keras/src/ops/numpy.py", line 3617, in logical_and

  File "/usr/local/lib/python3.11/dist-packages/keras/src/backend/tensorflow/numpy.py", line 1520, in logical_and
```

```
Incompatible shapes: [1,288] vs. [1,32]
     [[{{node LogicalAnd_1}}]] [Op:__inference_multi_step_on_iterator_25447]
```

Next steps: Explain error

```
# Evaluate the model on the test dataset
test_loss, test_accuracy = boat_cnn.evaluate(X_test,y_test)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-35-2643051298> in <cell line: 0>()
      1 # Evaluate the model on the test dataset
----> 2 test_loss, test_accuracy = boat_cnn.evaluate(X_test,y_test)

                              1 frames
/usr/local/lib/python3.11/dist-packages/optree/ops.py in tree_map(func, tree, is_leaf, none_is_leaf, namespace, *rests)
    764     leaves, treespec = _C.flatten(tree, is_leaf, none_is_leaf, namespace)
    765     flat_args = [leaves] + [treespec.flatten_up_to(r) for r in rests]
--> 766     return treespec.unflatten(map(func, *flat_args))
    767
    768

ValueError: Cannot take the length of shape with unknown rank.
```

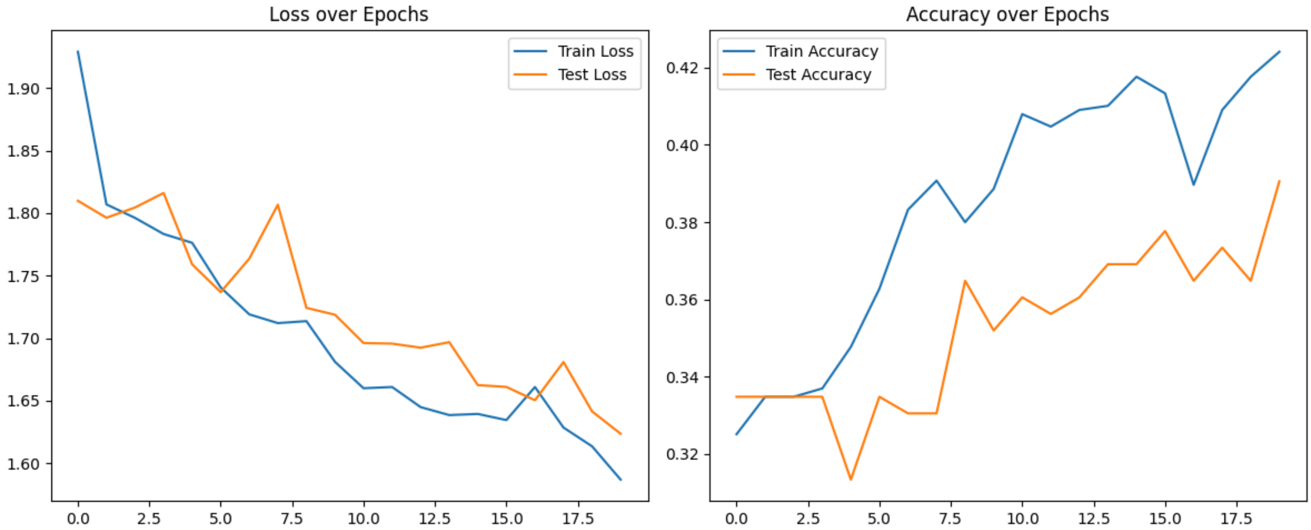
Next steps: Explain error

1.8.Plot heatmap of the confusion matrix and print classification report.

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import os
import pathlib
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras import layers, models
from tensorflow.keras.metrics import Precision, Recall
from sklearn.model_selection import train_test_split
import shutil

# Step 0: Setup (Assuming dataset is already downloaded and extracted to /content/boat_data)
base_dir = "/content/boat_data"
train_dir = os.path.join(base_dir, "Train")

# Step 1.1: Get class names and prepare image paths
class_names = sorted(entry.name for entry in os.scandir(train_dir) if entry.is_dir())
image_paths = []
labels = []

for idx, class_name in enumerate(class_names):
    class_folder = os.path.join(train_dir, class_name)
    for fname in os.listdir(class_folder):
        if fname.endswith(('.jpg', '.jpeg', '.png')):
            image_paths.append(os.path.join(class_folder, fname))
            labels.append(class_name)

# Split into train and test set
train_paths, test_paths, train_labels, test_labels = train_test_split(
    image_paths, labels, test_size=0.2, stratify=labels, random_state=43
)

# Helper function to create directory structure for train/test
def organize_dataset(image_paths, labels, output_dir):
    for path, label in zip(image_paths, labels):
        label_dir = os.path.join(output_dir, label)
        os.makedirs(label_dir, exist_ok=True)
        shutil.copy(path, os.path.join(label_dir, os.path.basename(path)))

# Create structured folders
structured_train = os.path.join(base_dir, "Structured_Train")
structured_test = os.path.join(base_dir, "Structured_Test")

if os.path.exists(structured_train):
    shutil.rmtree(structured_train)
if os.path.exists(structured_test):
    shutil.rmtree(structured_test)

organize_dataset(train_paths, train_labels, structured_train)
organize_dataset(test_paths, test_labels, structured_test)
```

```
# Step 1.2 & 1.3: Load datasets
batch_size = 32
img_size = (180, 180)

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    structured_train,
    image_size=img_size,
    batch_size=batch_size,
    label_mode='categorical',
    shuffle=True
)

test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    structured_test,
    image_size=img_size,
    batch_size=batch_size,
    label_mode='categorical',
    shuffle=False
)

# Normalize data
normalization_layer = layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds = test_ds.map(lambda x, y: (normalization_layer(x), y))

# Step 1.4: Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(180, 180, 3)),
    layers.MaxPooling2D(),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D(),
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(9, activation='softmax')
])

# Step 1.5: Compile model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy', Precision(name='precision'), Recall(name='recall')]
)

# Step 1.6: Train the model
history = model.fit(train_ds, epochs=20, validation_data=test_ds)

# Plot loss and accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.legend()
plt.title('Loss over Epochs')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.legend()
plt.title('Accuracy over Epochs')
plt.tight_layout()
plt.show()

# Step 1.7: Evaluate model
test_loss, test_accuracy, test_precision, test_recall = model.evaluate(test_ds)
test_loss, test_accuracy, test_precision, test_recall
```

```
Found 929 files belonging to 9 classes.
Found 233 files belonging to 9 classes.
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
30/30 ━━━━━━━━━━━━━━━━━━━━ 115s 1s/step - accuracy: 0.2969 - loss: 2.0334 - precision: 0.3264 - recall: 0.0067 - val_accuracy: 0.
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.3323 - loss: 1.8383 - precision: 0.7261 - recall: 0.0233 - val_accuracy: 0.3
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - accuracy: 0.3301 - loss: 1.7972 - precision: 0.9352 - recall: 0.0052 - val_accuracy: 0.3
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 83s 1s/step - accuracy: 0.3314 - loss: 1.8333 - precision: 0.9919 - recall: 0.0054 - val_accuracy: 0.3
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.3273 - loss: 1.8351 - precision: 0.3860 - recall: 0.0815 - val_accuracy: 0.3
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.3495 - loss: 1.7593 - precision: 0.4318 - recall: 0.0091 - val_accuracy: 0.3
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.3677 - loss: 1.7530 - precision: 0.5693 - recall: 0.0195 - val_accuracy: 0.3
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.3722 - loss: 1.7287 - precision: 0.6941 - recall: 0.0443 - val_accuracy: 0.3
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.3546 - loss: 1.7759 - precision: 0.4692 - recall: 0.1321 - val_accuracy: 0.3
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - accuracy: 0.3767 - loss: 1.7125 - precision: 0.6384 - recall: 0.0862 - val_accuracy: 0.3
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - accuracy: 0.4148 - loss: 1.6677 - precision: 0.5930 - recall: 0.0874 - val_accuracy: 0.3
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.3838 - loss: 1.6907 - precision: 0.5084 - recall: 0.0497 - val_accuracy: 0.3
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.3794 - loss: 1.7001 - precision: 0.4893 - recall: 0.0728 - val_accuracy: 0.3
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - accuracy: 0.3860 - loss: 1.7023 - precision: 0.5687 - recall: 0.1126 - val_accuracy: 0.3
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - accuracy: 0.3997 - loss: 1.6647 - precision: 0.5971 - recall: 0.1070 - val_accuracy: 0.3
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - accuracy: 0.4030 - loss: 1.6355 - precision: 0.5650 - recall: 0.1263 - val_accuracy: 0.3
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.3678 - loss: 1.6937 - precision: 0.4943 - recall: 0.1070 - val_accuracy: 0.3
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - accuracy: 0.3867 - loss: 1.6835 - precision: 0.5665 - recall: 0.0696 - val_accuracy: 0.3
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 80s 1s/step - accuracy: 0.3933 - loss: 1.6417 - precision: 0.6247 - recall: 0.1104 - val_accuracy: 0.3
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - accuracy: 0.4201 - loss: 1.6077 - precision: 0.5868 - recall: 0.1146 - val_accuracy: 0.3
```




```
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 539ms/step - accuracy: 0.2361 - loss: 1.8481 - precision: 0.4266 - recall: 0.1047
(1.6235002279281616,
 0.3905579447746277,
 0.5593220591545105,
 0.14163090288639069)
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import pathlib

# Define the dataset path
dataset_path = "/content/boat_data/Train"  # Adjust if needed
```

```
dataset_path = pathlib.Path(dataset_path)

# Load train and validation datasets (70:30 split)
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_path,
    validation_split=0.3,
    subset="training",
    seed=1,
    image_size=(180, 180),
    batch_size=32,
    label_mode='categorical'
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_path,
    validation_split=0.3,
    subset="validation",
    seed=1,
    image_size=(180, 180),
    batch_size=32
)

test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    structured_test,
    image_size=img_size,
    batch_size=batch_size,
    label_mode='categorical',
    shuffle=False
)


# Normalize images
normalization_layer = layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds = test_ds.map(lambda x, y: (normalization_layer(x), y))

# Cache and prefetch
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# Load pre-trained MobileNetV2
base_model = MobileNetV2(input_shape=(180, 180, 3), include_top=False, weights='imagenet')
base_model.trainable = False

# Build the transfer learning model
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.2),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.1),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.1),
    layers.Dense(9, activation='softmax')
])

# Compile model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()]
)

# Train model with early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    train_ds,
    validation_data=test_ds,
    epochs=50,
    callbacks=[early_stopping]
)

# Plotting
def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
```