Automatic Port Operations

- Identify different classes of boats using a Covolutional Neural Network model
- Identify different classes of boats using the MobiileNet_V2 model using transfer learning
- Compare the performnace of both and draw inferences

Tasks performed on each model

1. Evaluate Model's perfprmance by calculating accuracy and loss metrics
2. Make predictions on the test set
3. Calculate confusion matrix and display
4. Calculate classificaton report

Common Functions

In [ ]:
```
import os
import pathlib
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import seaborn as sns
import pandas as pd

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,classification

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from  tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings('ignore')

train_accuracy = {}
test_accuracy = {}
train_loss = {}
test_loss = {}


def print_scores(model, val_ds, train_ds,name='Default'):
    """
    :param model:
    :param val_ds:
    :param train_ds:
    :return:
    """
    print(f'Displaying accuracy and loss for {name}')
```

```
    score_test = model.evaluate(val_ds, verbose=0)
    score_train = model.evaluate(train_ds, verbose=0)

    print('Test loss:%.4f' % score_test[0])
    print('Test accuracy:%.4f' % score_test[1])

    print('Train loss:%.4f' % score_train[0])
    print('train accuracy:%.4f' % score_train[1])

    if not name == 'Default':
      test_loss[name] = score_test[0]
      test_accuracy[name] = score_test[1]
      train_loss[name] = score_train[0]
      train_accuracy[name] = score_train[1]



def _plot(history, param_1,param_2,ax):

    """
    :param history:
    :param param_1:
    :param param_2:
    :return:
    """

    ax.plot(history.history[param_1])
    ax.plot(history.history[param_2])
    ax.set_title(f'model {param_1}')
    ax.set_ylabel(param_1)
    ax.set_xlabel('epoch')
    ax.legend(['train', 'test'], loc='best')


def plot_accuracy_loss_graphs(history, title=None, show=True):
    """
    :param history:
    :param title:
    :param show:
    :return:
    """
    print()
    fig = plt.figure(figsize=(10, 10))

    # accuracy chart
    ax = plt.subplot(1, 2, 1)
    _plot(history,'accuracy','val_accuracy',ax)
    # loss chart
    ax = plt.subplot(1, 2, 2)
    _plot(history,'loss','val_loss',ax)
    if show:
        plt.tight_layout()
        plt.show()
    else:
        plt.savefig(f'{title}.png')
        plt.close()
```

```python
def get_predicted_labels(model,class_names):
    """
    :param model:
    :param return_proba:
    :return:
    """
    images = os.listdir(test_dir)
    image_label_map = {}


    for i, image in enumerate(images):
        path = test_dir + '/' + image
        img = tf.keras.utils.load_img(path, target_size=(height, width))
        img_array = tf.keras.utils.img_to_array(img)
        img_array = tf.expand_dims(img_array, 0)  # Create a batch
        predictions = model.predict(img_array, verbose=False)
        score = tf.nn.softmax(predictions[0])
        max_score = np.max(score)
        image_label_map[path] = class_names[np.argmax(score)]
    return image_label_map


def display_test_images(image_map, model_name):
    """

    :param image_map:
    :param model_name:
    :return:
    """
    print(f'Predictions for {model_name}\n')
    plt.figure(figsize=(25, 25))
    for i, image_path in enumerate(image_map.keys()):
        ax = plt.subplot(20, 15, i + 1)
        img = mpimg.imread(image_path)
        ax.imshow(img)
        ax.axis('off')
        label = image_map.get(image_path)
        # ax.text(0.5, .8, label, fontsize=10, fontweight='' color='green', ha='cen
        ax.set_title(label, fontsize=10)
    plt.show()


def display_cm(true_labels, predicted_labels):
    """
    :param true_labels:
    :param predicted_labels:
    :return:
    """
    cm = confusion_matrix(true_labels, predicted_labels)
    print('Confusion Matrix')
    print(cm)
    print()

    print('Confusion Matrix Displayed\n')
```

```
    fig = plt.figure(figsize=(10, 10))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
    disp.plot()
    plt.title('Confusion Matrix')
    plt.xticks(rotation=90)
    plt.show()


def get_metrics(model, validation_ds, val_split=0.2):
    num_images = int(total_train_images * val_split)
    num_batches = int(num_images / 32)
    residuals = int(num_images % 32)
    true_labels = []
    predicted_labels = []
    for images, labels in validation_ds.take(num_batches):
        true_labels.extend(labels)
        for i in range(batch_size):  # prediction for each image in the batch
            img_array = tf.keras.utils.img_to_array(images[i])
            img_array = tf.expand_dims(img_array, 0)  # Create a batch
            predictions = model.predict(img_array, verbose=False)
            score = tf.nn.softmax(predictions[0])
            predicted_labels.append(np.argmax(score))
    return true_labels, predicted_labels


def create_transfer_model(transfer_model, dropout=0.1, trainable=False, data_augmen
    transfer_model.trainable = trainable
    model_t = Sequential()
    if not data_augmentation is None:
        for aug in data_augmentation:
            print(f'added {aug} to model')
            model_t.add(aug)
    if data_augmentation is None:
        # data augmentation will add an input shape in the first layer
        model_t.add(layers.Rescaling(1. / 255, input_shape=(height, width, 3)))
    else:
        model_t.add(layers.Rescaling(1. / 255))
    model_t.add(transfer_model)
    model_t.add(layers.GlobalAveragePooling2D())
    model_t.add(layers.Dropout(dropout))


    # FCN
    model_t.add(layers.Flatten())
    model_t.add(layers.Dense(256, activation='relu'))
    model_t.add(layers.BatchNormalization())
    model_t.add(layers.Dropout(dropout))

    model_t.add(layers.Dense(128, activation='relu'))
    model_t.add(layers.BatchNormalization())
    model_t.add(layers.Dropout(dropout))

    model_t.add(layers.Dense(len(class_names), activation='softmax'))
    model_t.summary()
    return model_t
```

CNN Model

Load Datasets

In [ ]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

Extract data

In [ ]:
```
# from zipfile import ZipFile


# zip_ref = ZipFile('/content/drive/MyDrive/DL/BOATS.zip','r')
# zip_ref.extractall('/content/drive/MyDrive/DL')
# zip_ref.close()
```

Define constants , test and train directories

In [ ]:
```
# lets define some constants
train_dir_kaggle = '/kaggle/input/boats-ds/TRAIN_BOATS'
test_dir_kaggle = '/kaggle/input/boats-ds/TEST_BOATS'


train_dir_colab = '/content/drive/MyDrive/DL/TRAIN_BOATS/'
test_dir_colab = '/content/drive/MyDrive/DL/TEST_BOATS/'

train_dir = train_dir_colab
test_dir = test_dir_colab

if os.path.exists(train_dir_kaggle):
  train_dir = train_dir_kaggle
  test_dir = test_dir_kaggle
  print('Using Kaggle data')
  print(f'train_dir = {train_dir}')
  print(f'test_dir = {test_dir}')
else:
  print('Using Colab data')
  print(f'train_dir = {train_dir}')
  print(f'test_dir = {test_dir}')


height = 224
width = 224
batch_size = 32
total_train_images = 0

total_test_images = len(os.listdir(test_dir))
```

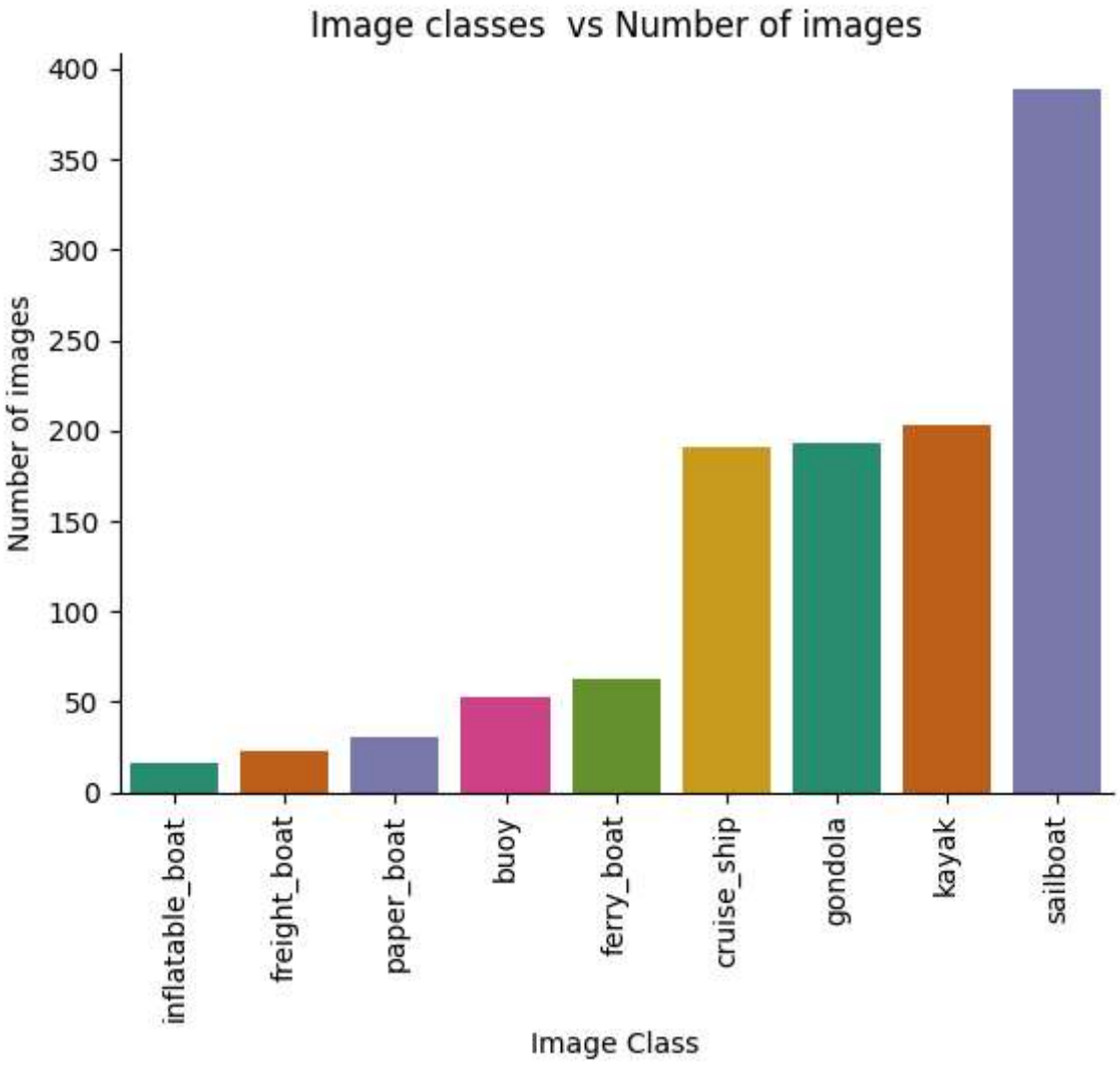
```
for sub_dir in os.listdir(train_dir):
  total_train_images += len(os.listdir(train_dir+'/'+sub_dir))

print('total train images = ', total_train_images)
print('total test images =  ', total_test_images)
```

```
Using Kaggle data
train_dir = /kaggle/input/boats-ds/TRAIN_BOATS
test_dir = /kaggle/input/boats-ds/TEST_BOATS
total train images =  1162
total test images =   300
```

Estimate number of images per class and plot

In [ ]:

```
data_dir = pathlib.Path(train_dir)

def get_subdirectories(directory):
    with os.scandir(directory) as entries:
        return [entry.name for entry in entries if entry.is_dir()]
sub_dirs = get_subdirectories(data_dir)
images = {}
image_class = []
num_images = []
class_dict = {}

for d in sub_dirs:
    image_class.append(d)
    num_images.append(len(os.listdir(train_dir+'/'+d)))

df = pd.DataFrame({'Image_Class':image_class,'Num_Images':num_images}).sort_values(

indices = sorted(df['index'].values.tolist())
class_names = df['Image_Class'].values
for i in indices:
  class_dict[i] = class_names[i]
sns.barplot(data=df,x='Image_Class',y='Num_Images',palette=sns.mpl_palette('Dark2')
plt.xlabel('Image Class')
plt.ylabel('Number of images')
plt.title(f'Image classes  vs Number of images' )
plt.xticks(rotation = 90)
plt.gca().spines[['top', 'right',]].set_visible(False)
plt.show()
df
```



Out[ ]:

	index	Image_Class	Num_Images
0	2	inflatable_boat	16
1	0	freight_boat	23
2	3	paper_boat	31
3	6	buoy	53
4	4	ferry_boat	63
5	7	cruise_ship	191
6	1	gondola	193
7	5	kayak	203
8	8	sailboat	389

There is a predominence of the sailboat class in the training data. The bottom three image classes have less then 50 images per class. Ideally we can drop them. Having them in the training/testing datasets can skew the predictions.

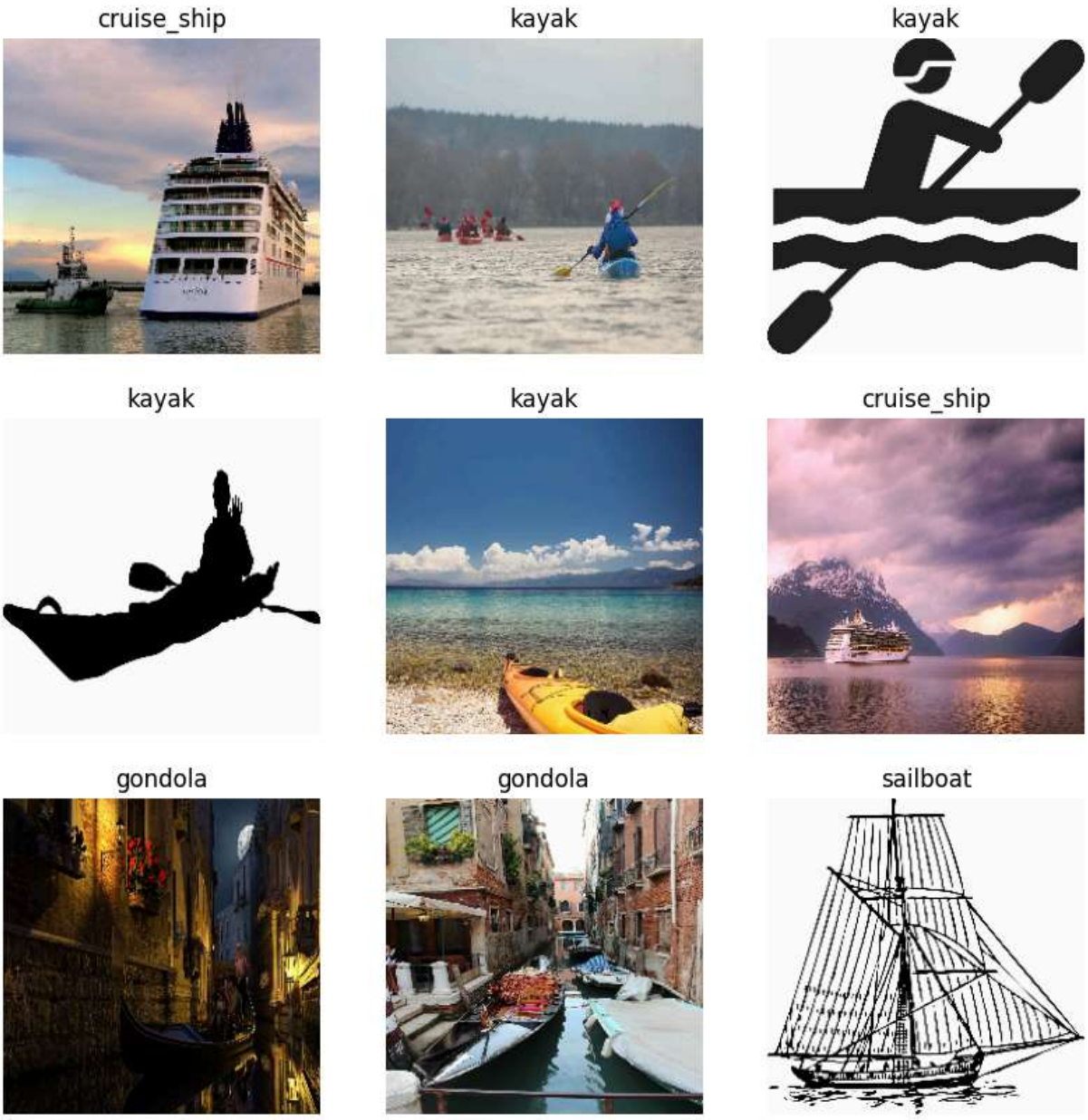
Define datasets for training and validatating the model

In [ ]:
```
#Not using ImageDataGenerator asit is deprecated and it doesnt allow for
#caching and prefetching

# https://www.tensorflow.org/guide/data
import tensorflow as tf

def get_ds(data_dir,validation_split=0.2,subset='training',seed = 43,image_size=(25
    train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=validation_split,
    subset=subset,
    seed=seed,
    image_size=image_size,
    batch_size=batch_size)
    train_ds.class_names
    return train_ds

train_ds = get_ds(data_dir,image_size=(height,width),batch_size=batch_size)
val_ds = get_ds(data_dir,validation_split=0.2,subset='validation',seed = 43,image_s
class_names = train_ds.class_names
print(class_names)
```

```
Found 1162 files belonging to 9 classes.
Using 930 files for training.
Found 1162 files belonging to 9 classes.
Using 232 files for validation.
['buoy', 'cruise_ship', 'ferry_boat', 'freight_boat', 'gondola', 'inflatable_boat',
'kayak', 'paper_boat', 'sailboat']
```

Plot some images from the training dataset

In [ ]:
```
import matplotlib.pyplot as plt

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

for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



Set up the datasets for caching and prefetching

In [ ]:
```
# Using `cache`, `prfetch` and `AUTOTUNE` for efficient handling of input data.
# https://www.tensorflow.org/guide/data_performance
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

CNN model exactly as specfied in the problem

2 Conv2d layers with Max 2d pooling

1 2D Global Average Pooling

2 FFN with 2 dense layers of 128 neurons each

1 outpt layer with 9 neurons and softmax activation

In [ ]:
```
model = Sequential([

  # Image Scaling layer
  layers.Rescaling(1./255, input_shape=(height, width, 3)),

  # First convolution layer.convolution with 32 filters with a 3x3 kernel matrix,no
  layers.Conv2D(32, (3,3),  activation='relu'),
  layers.MaxPooling2D(),
  #layers.Dropout(0.1),

  # Second Convolution layer
  layers.Conv2D(32, (3,3),  activation='relu'),
  layers.MaxPooling2D(),

  #Global Average Pooling
  layers.GlobalAveragePooling2D(),

  # FFN
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(128, activation='relu'),
  layers.Dense(len(class_names),activation='softmax')

])
```

Compile the model

In [ ]:
```
model.compile(optimizer='adam',
              #https://www.tensorflow.org/api_docs/python/tf/keras/losses/Categoric
              #It states:
              # "We expect labels to be provided in a one_hot representation.
              #  If you want to provide labels as integers, please use SparseCatego
              loss = 'sparse_categorical_crossentropy',
              # keras has removed support for batch level precision and recall metr
              metrics=['accuracy']
             )
model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	
rescaling_5 (Rescaling)	(None, 224, 224, 3)	
conv2d_6 (Conv2D)	(None, 222, 222, 32)	
max_pooling2d_5 (MaxPooling2D)	(None, 111, 111, 32)	
conv2d_7 (Conv2D)	(None, 109, 109, 32)	
max_pooling2d_6 (MaxPooling2D)	(None, 54, 54, 32)	
global_average_pooling2d_5 (GlobalAveragePooling2D)	(None, 32)	
flatten_5 (Flatten)	(None, 32)	
dense_15 (Dense)	(None, 128)	
dense_16 (Dense)	(None, 128)	
dense_17 (Dense)	(None, 9)	

Total params: 32,041 (125.16 KB)

Trainable params: 32,041 (125.16 KB)

Non-trainable params: 0 (0.00 B)

Train the model with Early Stopping and Checkpoint

In [ ]:
```
from tensorflow.keras.callbacks import EarlyStopping,ModelCheckpoint

#Optimizing for maximum accuracy instead of minimum loss.( min loss does not always
early_stop=EarlyStopping(monitor='val_accuracy',mode='max',verbose=1,patience=10,mi
checkpoint=ModelCheckpoint('best_model.keras',monitor='val_accuracy',mode='max',ver

epochs=100
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  batch_size = batch_size,
  # not specified. But i thought this was a good idea
  callbacks = [early_stop,checkpoint]
)
```

```
Epoch 1/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.1599 - loss: 2.0934
Epoch 1: val_accuracy improved from -inf to 0.30172, saving model to best_model.kera
s
30/30 ━━━━━━━━━━━━━━━━━━━━ 8s 104ms/step - accuracy: 0.1632 - loss: 2.0889 - val_acc
uracy: 0.3017 - val_loss: 1.8214
Epoch 2/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3541 - loss: 1.8238
Epoch 2: val_accuracy did not improve from 0.30172
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.3537 - loss: 1.8233 - val_accu
racy: 0.3017 - val_loss: 1.8013
Epoch 3/100
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3513 - loss: 1.8119
Epoch 3: val_accuracy did not improve from 0.30172
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.3508 - loss: 1.8114 - val_accu
racy: 0.3017 - val_loss: 1.7909
Epoch 4/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3459 - loss: 1.7848
Epoch 4: val_accuracy did not improve from 0.30172
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.3458 - loss: 1.7853 - val_accu
racy: 0.3017 - val_loss: 1.7798
Epoch 5/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3528 - loss: 1.8014
Epoch 5: val_accuracy did not improve from 0.30172
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.3527 - loss: 1.8003 - val_accu
racy: 0.2888 - val_loss: 1.7624
Epoch 6/100
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3320 - loss: 1.7539
Epoch 6: val_accuracy did not improve from 0.30172
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.3329 - loss: 1.7529 - val_accu
racy: 0.2931 - val_loss: 1.7621
Epoch 7/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3678 - loss: 1.7387
Epoch 7: val_accuracy improved from 0.30172 to 0.31897, saving model to best_model.k
eras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.3685 - loss: 1.7378 - val_accu
racy: 0.3190 - val_loss: 1.7659
Epoch 8/100
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3983 - loss: 1.6843
Epoch 8: val_accuracy did not improve from 0.31897
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.3982 - loss: 1.6857 - val_accu
racy: 0.3147 - val_loss: 1.7223
Epoch 9/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4083 - loss: 1.6965
Epoch 9: val_accuracy improved from 0.31897 to 0.33190, saving model to best_model.k
eras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.4085 - loss: 1.6957 - val_accu
racy: 0.3319 - val_loss: 1.7124
Epoch 10/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4022 - loss: 1.6326
Epoch 10: val_accuracy did not improve from 0.33190
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4020 - loss: 1.6336 - val_accu
racy: 0.3190 - val_loss: 1.7338
Epoch 11/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4030 - loss: 1.6648
Epoch 11: val_accuracy did not improve from 0.33190
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4031 - loss: 1.6649 - val_accu
racy: 0.3147 - val_loss: 1.7488
Epoch 12/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3887 - loss: 1.6601
Epoch 12: val_accuracy did not improve from 0.33190
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.3892 - loss: 1.6601 - val_accu
racy: 0.3103 - val_loss: 1.7197
Epoch 13/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4193 - loss: 1.6074
Epoch 13: val_accuracy improved from 0.33190 to 0.33621, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.4190 - loss: 1.6084 - val_accu
racy: 0.3362 - val_loss: 1.7222
Epoch 14/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4171 - loss: 1.6394
Epoch 14: val_accuracy did not improve from 0.33621
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4170 - loss: 1.6393 - val_accu
racy: 0.2931 - val_loss: 1.7300
Epoch 15/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.3950 - loss: 1.6497
Epoch 15: val_accuracy did not improve from 0.33621
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.3956 - loss: 1.6496 - val_accu
racy: 0.3190 - val_loss: 1.6948
Epoch 16/100
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.4550 - loss: 1.5851
Epoch 16: val_accuracy did not improve from 0.33621
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4534 - loss: 1.5872 - val_accu
racy: 0.3103 - val_loss: 1.7035
Epoch 17/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.4223 - loss: 1.6248
Epoch 17: val_accuracy did not improve from 0.33621
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.4224 - loss: 1.6248 - val_accu
racy: 0.3233 - val_loss: 1.7079
Epoch 18/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4485 - loss: 1.5898
Epoch 18: val_accuracy improved from 0.33621 to 0.35345, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.4478 - loss: 1.5901 - val_accu
racy: 0.3534 - val_loss: 1.6711
Epoch 19/100
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4384 - loss: 1.5599
Epoch 19: val_accuracy did not improve from 0.35345
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4374 - loss: 1.5619 - val_accu
racy: 0.3233 - val_loss: 1.6635
Epoch 20/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4442 - loss: 1.5684
Epoch 20: val_accuracy did not improve from 0.35345
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4434 - loss: 1.5694 - val_accu
racy: 0.3233 - val_loss: 1.7167
Epoch 21/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4419 - loss: 1.6075
Epoch 21: val_accuracy did not improve from 0.35345
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4417 - loss: 1.6072 - val_accu
racy: 0.3319 - val_loss: 1.6669
Epoch 22/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4393 - loss: 1.5923
```

```
Epoch 22: val_accuracy improved from 0.35345 to 0.36638, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.4396 - loss: 1.5915 - val_accu
racy: 0.3664 - val_loss: 1.6401
Epoch 23/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4486 - loss: 1.5733
Epoch 23: val_accuracy did not improve from 0.36638
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4486 - loss: 1.5729 - val_accu
racy: 0.3362 - val_loss: 1.6786
Epoch 24/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.4342 - loss: 1.5815
Epoch 24: val_accuracy did not improve from 0.36638
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.4349 - loss: 1.5805 - val_accu
racy: 0.3103 - val_loss: 1.7410
Epoch 25/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4226 - loss: 1.6282
Epoch 25: val_accuracy did not improve from 0.36638
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4233 - loss: 1.6265 - val_accu
racy: 0.3491 - val_loss: 1.6155
Epoch 26/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4329 - loss: 1.5421
Epoch 26: val_accuracy did not improve from 0.36638
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4334 - loss: 1.5424 - val_accu
racy: 0.3621 - val_loss: 1.6843
Epoch 27/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4332 - loss: 1.5223
Epoch 27: val_accuracy improved from 0.36638 to 0.38793, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.4340 - loss: 1.5223 - val_accu
racy: 0.3879 - val_loss: 1.6404
Epoch 28/100
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4486 - loss: 1.5298
Epoch 28: val_accuracy improved from 0.38793 to 0.43966, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.4490 - loss: 1.5297 - val_accu
racy: 0.4397 - val_loss: 1.5738
Epoch 29/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4716 - loss: 1.5447
Epoch 29: val_accuracy did not improve from 0.43966
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4719 - loss: 1.5435 - val_accu
racy: 0.3922 - val_loss: 1.6152
Epoch 30/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4536 - loss: 1.5128
Epoch 30: val_accuracy did not improve from 0.43966
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4541 - loss: 1.5128 - val_accu
racy: 0.4009 - val_loss: 1.5887
Epoch 31/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.5038 - loss: 1.4957
Epoch 31: val_accuracy did not improve from 0.43966
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.5033 - loss: 1.4952 - val_accu
racy: 0.3836 - val_loss: 1.5913
Epoch 32/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4610 - loss: 1.5313
Epoch 32: val_accuracy did not improve from 0.43966
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.4618 - loss: 1.5298 - val_accu
racy: 0.4095 - val_loss: 1.5636
```

```
Epoch 33/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4689 - loss: 1.5401
Epoch 33: val_accuracy did not improve from 0.43966
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4695 - loss: 1.5383 - val_accu
racy: 0.4267 - val_loss: 1.5878
Epoch 34/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.5177 - loss: 1.4321
Epoch 34: val_accuracy did not improve from 0.43966
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.5169 - loss: 1.4334 - val_accu
racy: 0.4095 - val_loss: 1.5417
Epoch 35/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4668 - loss: 1.5206
Epoch 35: val_accuracy did not improve from 0.43966
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4670 - loss: 1.5197 - val_accu
racy: 0.3793 - val_loss: 1.6306
Epoch 36/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4859 - loss: 1.4642
Epoch 36: val_accuracy did not improve from 0.43966
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.4860 - loss: 1.4646 - val_accu
racy: 0.4095 - val_loss: 1.5504
Epoch 37/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.4988 - loss: 1.4550
Epoch 37: val_accuracy did not improve from 0.43966
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.4985 - loss: 1.4549 - val_accu
racy: 0.4267 - val_loss: 1.5399
Epoch 38/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.5219 - loss: 1.3347
Epoch 38: val_accuracy improved from 0.43966 to 0.44828, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.5211 - loss: 1.3380 - val_accu
racy: 0.4483 - val_loss: 1.5192
Epoch 38: early stopping
```

Print the metrics

In [ ]:
```
from tensorflow.keras.models import load_model
model = load_model('best_model.keras')
print_scores(model,val_ds,train_ds)
```

```
Displaying accuracy and loss for Default
Test loss:1.5192
Test accuracy:0.4483
Train loss:1.4294
train accuracy:0.5000
```

Modified CNN Model

In [ ]:
```
model = Sequential([
  # Rescaling  images
  layers.Rescaling(1./255, input_shape=(height, width, 3)),

  # First convolution layer.convolution with 32 filters with a 3x3 kernel matrix
  layers.Conv2D(32, (3,3), padding='same', activation='relu'),
  layers.MaxPooling2D(), # Use max pooling
```

```
  layers.Dropout(0.1),

  # Second Convolution layer
  layers.Conv2D(32, (3,3), padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.1),

  # 3rd Convolution layer
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.1),

  # 4th Convolution layer
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.GlobalAveragePooling2D(),
  layers.Dropout(0.1),

  #FCN
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(128, activation='relu'),
  layers.Dropout(0.2),
  layers.Dense(len(class_names),activation='softmax')

])
```

Compile the model

In [ ]:
```
model.compile(optimizer='adam',
              #https://www.tensorflow.org/api_docs/python/tf/keras/losses/Categoric
              #It states:
              # "We expect labels to be provided in a one_hot representation.
              #  If you want to provide labels as integers, please use SparseCatego
              loss = 'sparse_categorical_crossentropy',
              # keras has removed support for batch level precision and recall metr
              metrics=['accuracy']
             )
model.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	
rescaling_6 (Rescaling)	(None, 224, 224, 3)	
conv2d_8 (Conv2D)	(None, 224, 224, 32)	
max_pooling2d_7 (MaxPooling2D)	(None, 112, 112, 32)	
dropout_14 (Dropout)	(None, 112, 112, 32)	
conv2d_9 (Conv2D)	(None, 112, 112, 32)	
max_pooling2d_8 (MaxPooling2D)	(None, 56, 56, 32)	
dropout_15 (Dropout)	(None, 56, 56, 32)	
conv2d_10 (Conv2D)	(None, 56, 56, 64)	
max_pooling2d_9 (MaxPooling2D)	(None, 28, 28, 64)	
dropout_16 (Dropout)	(None, 28, 28, 64)	
conv2d_11 (Conv2D)	(None, 28, 28, 128)	
global_average_pooling2d_6 (GlobalAveragePooling2D)	(None, 128)	
dropout_17 (Dropout)	(None, 128)	
flatten_6 (Flatten)	(None, 128)	
dense_18 (Dense)	(None, 128)	
dense_19 (Dense)	(None, 128)	
dropout_18 (Dropout)	(None, 128)	
dense_20 (Dense)	(None, 9)	

Total params: 136,681 (533.91 KB)

Trainable params: 136,681 (533.91 KB)

Non-trainable params: 0 (0.00 B)

Train the model

In [ ]:
```
histories = []
```

In [ ]:
```
from tensorflow.keras.callbacks import EarlyStopping,ModelCheckpoint

early_stop=EarlyStopping(monitor='val_accuracy',mode='max',verbose=1,patience=10,mi
checkpoint=ModelCheckpoint('best_model.keras',monitor='val_accuracy',mode='max',ver
```

```
epochs=100
history_c = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  batch_size = batch_size,
  callbacks = [early_stop,checkpoint]
)

histories.append(history_c)
```

```
Epoch 1/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step - accuracy: 0.2754 - loss: 2.0274
Epoch 1: val_accuracy improved from -inf to 0.30172, saving model to best_model.kera
s
30/30 ━━━━━━━━━━━━━━━━━━━━ 10s 181ms/step - accuracy: 0.2771 - loss: 2.0240 - val_ac
curacy: 0.3017 - val_loss: 1.8864
Epoch 2/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.3608 - loss: 1.8412
Epoch 2: val_accuracy did not improve from 0.30172
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.3601 - loss: 1.8411 - val_accu
racy: 0.3017 - val_loss: 1.8320
Epoch 3/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.3178 - loss: 1.8541
Epoch 3: val_accuracy did not improve from 0.30172
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.3185 - loss: 1.8533 - val_accu
racy: 0.3017 - val_loss: 1.8310
Epoch 4/100
29/30 ━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.3232 - loss: 1.7825
Epoch 4: val_accuracy did not improve from 0.30172
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.3239 - loss: 1.7845 - val_accu
racy: 0.3017 - val_loss: 1.8549
Epoch 5/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.3408 - loss: 1.8114
Epoch 5: val_accuracy improved from 0.30172 to 0.31034, saving model to best_model.k
eras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.3412 - loss: 1.8105 - val_accu
racy: 0.3103 - val_loss: 1.7483
Epoch 6/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.3815 - loss: 1.7528
Epoch 6: val_accuracy did not improve from 0.31034
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.3814 - loss: 1.7524 - val_accu
racy: 0.2888 - val_loss: 1.7845
Epoch 7/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.3681 - loss: 1.7552
Epoch 7: val_accuracy improved from 0.31034 to 0.32759, saving model to best_model.k
eras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.3686 - loss: 1.7540 - val_accu
racy: 0.3276 - val_loss: 1.7307
Epoch 8/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.3784 - loss: 1.7068
Epoch 8: val_accuracy improved from 0.32759 to 0.34052, saving model to best_model.k
eras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.3791 - loss: 1.7061 - val_accu
racy: 0.3405 - val_loss: 1.7074
Epoch 9/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.3926 - loss: 1.6597
Epoch 9: val_accuracy improved from 0.34052 to 0.36207, saving model to best_model.k
eras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.3931 - loss: 1.6595 - val_accu
racy: 0.3621 - val_loss: 1.7020
Epoch 10/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4222 - loss: 1.6332
Epoch 10: val_accuracy did not improve from 0.36207
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.4222 - loss: 1.6338 - val_accu
racy: 0.3534 - val_loss: 1.6633
Epoch 11/100
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4166 - loss: 1.6169
Epoch 11: val_accuracy did not improve from 0.36207
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.4170 - loss: 1.6170 - val_accu
racy: 0.3448 - val_loss: 1.6754
Epoch 12/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4442 - loss: 1.6281
Epoch 12: val_accuracy did not improve from 0.36207
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.4439 - loss: 1.6276 - val_accu
racy: 0.3362 - val_loss: 1.7065
Epoch 13/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4412 - loss: 1.6160
Epoch 13: val_accuracy improved from 0.36207 to 0.41379, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.4409 - loss: 1.6160 - val_accu
racy: 0.4138 - val_loss: 1.5993
Epoch 14/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4534 - loss: 1.5165
Epoch 14: val_accuracy improved from 0.41379 to 0.41810, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.4538 - loss: 1.5179 - val_accu
racy: 0.4181 - val_loss: 1.6002
Epoch 15/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4760 - loss: 1.5215
Epoch 15: val_accuracy did not improve from 0.41810
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.4750 - loss: 1.5226 - val_accu
racy: 0.3621 - val_loss: 1.6471
Epoch 16/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4237 - loss: 1.6118
Epoch 16: val_accuracy improved from 0.41810 to 0.42672, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.4246 - loss: 1.6103 - val_accu
racy: 0.4267 - val_loss: 1.5459
Epoch 17/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4937 - loss: 1.4927
Epoch 17: val_accuracy improved from 0.42672 to 0.45690, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.4937 - loss: 1.4924 - val_accu
racy: 0.4569 - val_loss: 1.5236
Epoch 18/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4701 - loss: 1.4927
Epoch 18: val_accuracy did not improve from 0.45690
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.4699 - loss: 1.4938 - val_accu
racy: 0.4181 - val_loss: 1.5472
Epoch 19/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5091 - loss: 1.4368
Epoch 19: val_accuracy did not improve from 0.45690
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.5084 - loss: 1.4384 - val_accu
racy: 0.4267 - val_loss: 1.5439
Epoch 20/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4972 - loss: 1.4598
Epoch 20: val_accuracy did not improve from 0.45690
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.4971 - loss: 1.4606 - val_accu
racy: 0.4397 - val_loss: 1.5420
Epoch 21/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4993 - loss: 1.4833
Epoch 21: val_accuracy did not improve from 0.45690
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.4991 - loss: 1.4830 - val_accu
racy: 0.4095 - val_loss: 1.6414
Epoch 22/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4759 - loss: 1.4987
Epoch 22: val_accuracy did not improve from 0.45690
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.4764 - loss: 1.4985 - val_accu
racy: 0.4483 - val_loss: 1.5204
Epoch 23/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4975 - loss: 1.4531
Epoch 23: val_accuracy did not improve from 0.45690
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.4977 - loss: 1.4533 - val_accu
racy: 0.4483 - val_loss: 1.5195
Epoch 24/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5120 - loss: 1.4290
Epoch 24: val_accuracy did not improve from 0.45690
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.5116 - loss: 1.4292 - val_accu
racy: 0.4181 - val_loss: 1.5476
Epoch 25/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5013 - loss: 1.4618
Epoch 25: val_accuracy improved from 0.45690 to 0.46121, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.5015 - loss: 1.4617 - val_accu
racy: 0.4612 - val_loss: 1.5595
Epoch 26/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.4985 - loss: 1.4689
Epoch 26: val_accuracy improved from 0.46121 to 0.48707, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.4993 - loss: 1.4675 - val_accu
racy: 0.4871 - val_loss: 1.4750
Epoch 27/100
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.5643 - loss: 1.3430
Epoch 27: val_accuracy improved from 0.48707 to 0.49138, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.5630 - loss: 1.3460 - val_accu
racy: 0.4914 - val_loss: 1.4607
Epoch 28/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5517 - loss: 1.3956
Epoch 28: val_accuracy did not improve from 0.49138
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.5512 - loss: 1.3958 - val_accu
racy: 0.4138 - val_loss: 1.5355
Epoch 29/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5216 - loss: 1.4137
Epoch 29: val_accuracy did not improve from 0.49138
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.5214 - loss: 1.4141 - val_accu
racy: 0.4828 - val_loss: 1.4935
Epoch 30/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5246 - loss: 1.4157
Epoch 30: val_accuracy improved from 0.49138 to 0.50431, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.5246 - loss: 1.4157 - val_accu
racy: 0.5043 - val_loss: 1.4560
Epoch 31/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5597 - loss: 1.3598
Epoch 31: val_accuracy improved from 0.50431 to 0.56466, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.5600 - loss: 1.3590 - val_accu
```

```
racy: 0.5647 - val_loss: 1.4156
Epoch 32/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5635 - loss: 1.3315
Epoch 32: val_accuracy did not improve from 0.56466
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.5636 - loss: 1.3313 - val_accu
racy: 0.4871 - val_loss: 1.4844
Epoch 33/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6046 - loss: 1.2624
Epoch 33: val_accuracy did not improve from 0.56466
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6041 - loss: 1.2633 - val_accu
racy: 0.5086 - val_loss: 1.5689
Epoch 34/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5677 - loss: 1.3315
Epoch 34: val_accuracy did not improve from 0.56466
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.5670 - loss: 1.3325 - val_accu
racy: 0.4871 - val_loss: 1.5114
Epoch 35/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5418 - loss: 1.3916
Epoch 35: val_accuracy did not improve from 0.56466
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.5417 - loss: 1.3905 - val_accu
racy: 0.4310 - val_loss: 1.5115
Epoch 36/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5630 - loss: 1.3022
Epoch 36: val_accuracy improved from 0.56466 to 0.58621, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.5630 - loss: 1.3027 - val_accu
racy: 0.5862 - val_loss: 1.3985
Epoch 37/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5943 - loss: 1.2611
Epoch 37: val_accuracy improved from 0.58621 to 0.59483, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.5937 - loss: 1.2614 - val_accu
racy: 0.5948 - val_loss: 1.3884
Epoch 38/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6036 - loss: 1.1550
Epoch 38: val_accuracy did not improve from 0.59483
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6029 - loss: 1.1584 - val_accu
racy: 0.5474 - val_loss: 1.4566
Epoch 39/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.5618 - loss: 1.2791
Epoch 39: val_accuracy did not improve from 0.59483
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.5630 - loss: 1.2775 - val_accu
racy: 0.5905 - val_loss: 1.4025
Epoch 40/100
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.6124 - loss: 1.1888
Epoch 40: val_accuracy improved from 0.59483 to 0.60776, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.6125 - loss: 1.1893 - val_accu
racy: 0.6078 - val_loss: 1.3550
Epoch 41/100
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.6414 - loss: 1.1299
Epoch 41: val_accuracy improved from 0.60776 to 0.62500, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.6392 - loss: 1.1346 - val_accu
racy: 0.6250 - val_loss: 1.3407
Epoch 42/100
```

```
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.6685 - loss: 1.0741
Epoch 42: val_accuracy did not improve from 0.62500
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.6674 - loss: 1.0778 - val_accu
racy: 0.5819 - val_loss: 1.3568
Epoch 43/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6295 - loss: 1.1373
Epoch 43: val_accuracy did not improve from 0.62500
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6298 - loss: 1.1376 - val_accu
racy: 0.5948 - val_loss: 1.3499
Epoch 44/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6640 - loss: 1.1272
Epoch 44: val_accuracy did not improve from 0.62500
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6631 - loss: 1.1287 - val_accu
racy: 0.5776 - val_loss: 1.3912
Epoch 45/100
29/30 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.6260 - loss: 1.1668
Epoch 45: val_accuracy did not improve from 0.62500
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6269 - loss: 1.1619 - val_accu
racy: 0.5905 - val_loss: 1.4323
Epoch 46/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6272 - loss: 1.2053
Epoch 46: val_accuracy did not improve from 0.62500
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6275 - loss: 1.2039 - val_accu
racy: 0.5819 - val_loss: 1.3794
Epoch 47/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6475 - loss: 1.0929
Epoch 47: val_accuracy did not improve from 0.62500
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6477 - loss: 1.0931 - val_accu
racy: 0.5647 - val_loss: 1.3837
Epoch 48/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6200 - loss: 1.1485
Epoch 48: val_accuracy improved from 0.62500 to 0.63362, saving model to best_model.
keras
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.6203 - loss: 1.1478 - val_accu
racy: 0.6336 - val_loss: 1.2437
Epoch 49/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6466 - loss: 1.0783
Epoch 49: val_accuracy did not improve from 0.63362
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6467 - loss: 1.0790 - val_accu
racy: 0.6121 - val_loss: 1.2960
Epoch 50/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6902 - loss: 1.0119
Epoch 50: val_accuracy did not improve from 0.63362
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6895 - loss: 1.0135 - val_accu
racy: 0.6034 - val_loss: 1.2998
Epoch 51/100
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.6802 - loss: 0.9967
Epoch 51: val_accuracy did not improve from 0.63362
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6801 - loss: 0.9983 - val_accu
racy: 0.6336 - val_loss: 1.2546
Epoch 51: early stopping
```

Print Metrics

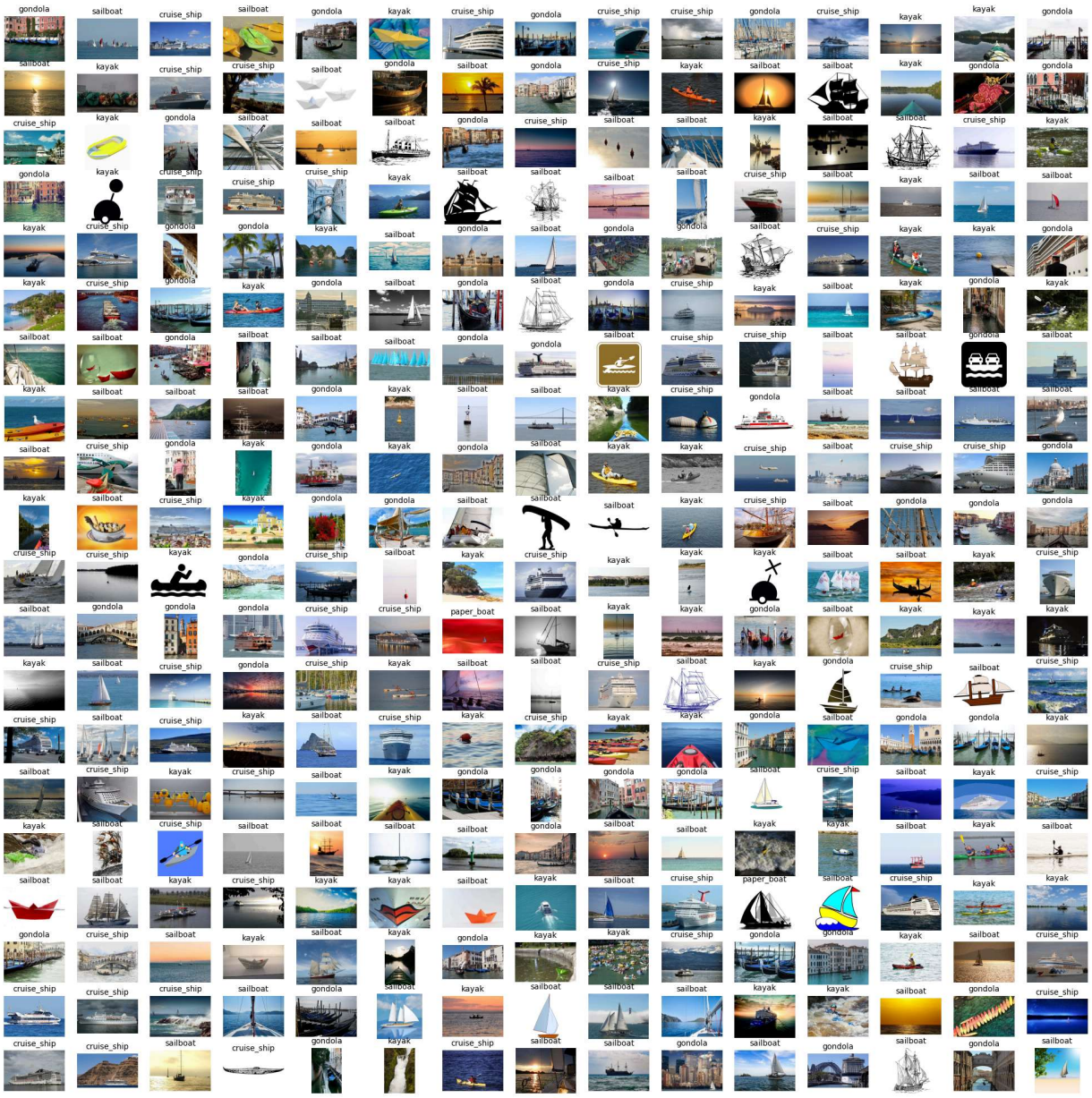
In [ ]:
```
from tensorflow.keras.models import load_model
model = load_model('best_model.keras')
print_scores(model,val_ds,train_ds,'CNN Model')
```

```
Displaying accuracy and loss for CNN Model
Test loss:1.2437
Test accuracy:0.6336
Train loss:1.0040
train accuracy:0.6925
```

Display the predictions

In [ ]:
```
image_map = get_predicted_labels(model,class_names)
display_test_images(image_map,'CNN Model. Title is the predicted label')
```

```
Predictions for CNN Model. Title is the predicted label
```



Notice how classes with very few images in training set tend get misclassified. sailboat , which has the largest number of images is the best predicted class.This can be seen in the confusion matrix as well

Confusion Matrix

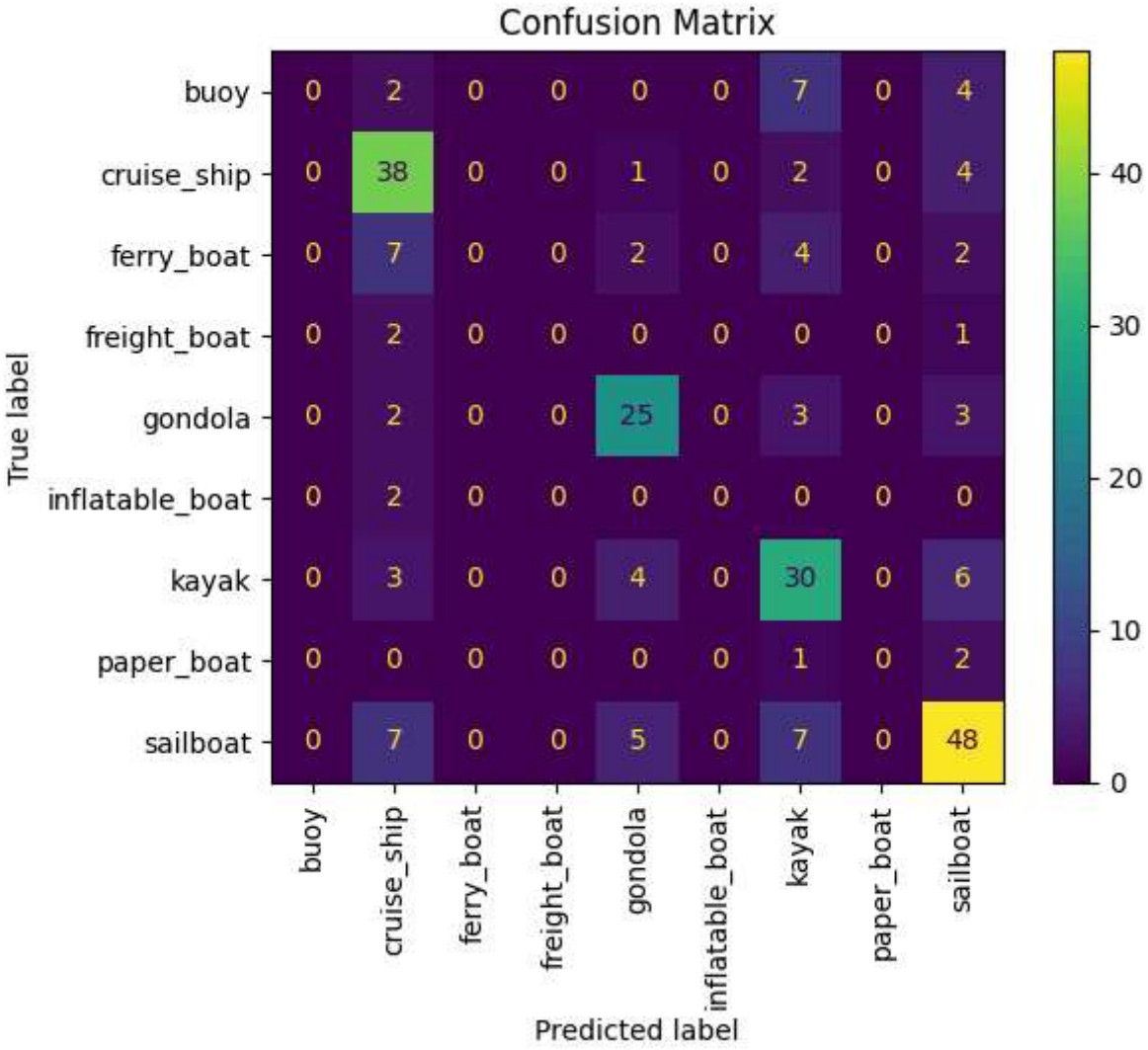
Get the predicted and True labels

In [ ]:
```
true_labels, predicted_labels = get_metrics(model,val_ds)
```

Display Confusion Matrix

In [ ]:
```
display_cm(true_labels,predicted_labels)
```

```
Confusion Matrix
[[ 0  2  0  0  0  0  7  0  4]
 [ 0 38  0  0  1  0  2  0  4]
 [ 0  7  0  0  2  0  4  0  2]
 [ 0  2  0  0  0  0  0  0  1]
 [ 0  2  0  0 25  0  3  0  3]
 [ 0  2  0  0  0  0  0  0  0]
 [ 0  3  0  0  4  0 30  0  6]
 [ 0  0  0  0  0  0  1  0  2]
 [ 0  7  0  0  5  0  7  0 48]]

Confusion Matrix Displayed

<Figure size 1000x1000 with 0 Axes>
```



As expected Sailboat,Kayak,Gondola and Cruise ship has high true positive values because of large number of images in the training data. Most mis-classifications are seen in the other image classes

Classification Report

In [ ]:
```
cl_report = classification_report(true_labels,predicted_labels)
print(cl_report)

print('Index to name mapping:\n')

for k,v in class_dict.items():
    print(f'{k} - > {v}')
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        13
           1       0.60      0.84      0.70        45
           2       0.00      0.00      0.00        15
           3       0.00      0.00      0.00         3
           4       0.68      0.76      0.71        33
           5       0.00      0.00      0.00         2
           6       0.56      0.70      0.62        43
           7       0.00      0.00      0.00         3
           8       0.69      0.72      0.70        67

    accuracy                           0.63       224
   macro avg       0.28      0.34      0.30       224
weighted avg       0.53      0.63      0.57       224

Index to name mapping:

0 - > inflatable_boat
1 - > freight_boat
2 - > paper_boat
3 - > buoy
4 - > ferry_boat
5 - > cruise_ship
6 - > gondola
7 - > kayak
8 - > sailboat
```

As expected the classes with higher support have the better overall scores

Transfer Learning with MobileNet application

Import mobilenet_v2 and freeze the weights

In [ ]:
```
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2
mob_model = tf.keras.applications.MobileNet(
    input_shape=(height,width,3),
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
    name='MobileNet',

)
mob_model.trainable=False
mob_model.summary()
```

```
Model: "MobileNet"
```

Layer (type)	Output Shape	
input_layer_8 (InputLayer)	(None, 224, 224, 3)	
conv1 (Conv2D)	(None, 112, 112, 32)	
conv1_bn (BatchNormalization)	(None, 112, 112, 32)	
conv1_relu (ReLU)	(None, 112, 112, 32)	
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	
conv_dw_1_bn (BatchNormalization)	(None, 112, 112, 32)	
conv_dw_1_relu (ReLU)	(None, 112, 112, 32)	
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	
conv_pw_1_bn (BatchNormalization)	(None, 112, 112, 64)	
conv_pw_1_relu (ReLU)	(None, 112, 112, 64)	
conv_pad_2 (ZeroPadding2D)	(None, 113, 113, 64)	
conv_dw_2 (DepthwiseConv2D)	(None, 56, 56, 64)	
conv_dw_2_bn (BatchNormalization)	(None, 56, 56, 64)	
conv_dw_2_relu (ReLU)	(None, 56, 56, 64)	
conv_pw_2 (Conv2D)	(None, 56, 56, 128)	
conv_pw_2_bn (BatchNormalization)	(None, 56, 56, 128)	
conv_pw_2_relu (ReLU)	(None, 56, 56, 128)	
conv_dw_3 (DepthwiseConv2D)	(None, 56, 56, 128)	
conv_dw_3_bn (BatchNormalization)	(None, 56, 56, 128)	
conv_dw_3_relu (ReLU)	(None, 56, 56, 128)	
conv_pw_3 (Conv2D)	(None, 56, 56, 128)	
conv_pw_3_bn (BatchNormalization)	(None, 56, 56, 128)	
conv_pw_3_relu (ReLU)	(None, 56, 56, 128)	
conv_pad_4 (ZeroPadding2D)	(None, 57, 57, 128)	
conv_dw_4 (DepthwiseConv2D)	(None, 28, 28, 128)	
conv_dw_4_bn (BatchNormalization)	(None, 28, 28, 128)	
conv_dw_4_relu (ReLU)	(None, 28, 28, 128)	

conv_pw_4 (Conv2D)	(None, 28, 28, 256)	
conv_pw_4_bn (BatchNormalization)	(None, 28, 28, 256)	
conv_pw_4_relu (ReLU)	(None, 28, 28, 256)	
conv_dw_5 (DepthwiseConv2D)	(None, 28, 28, 256)	
conv_dw_5_bn (BatchNormalization)	(None, 28, 28, 256)	
conv_dw_5_relu (ReLU)	(None, 28, 28, 256)	
conv_pw_5 (Conv2D)	(None, 28, 28, 256)	
conv_pw_5_bn (BatchNormalization)	(None, 28, 28, 256)	
conv_pw_5_relu (ReLU)	(None, 28, 28, 256)	
conv_pad_6 (ZeroPadding2D)	(None, 29, 29, 256)	
conv_dw_6 (DepthwiseConv2D)	(None, 14, 14, 256)	
conv_dw_6_bn (BatchNormalization)	(None, 14, 14, 256)	
conv_dw_6_relu (ReLU)	(None, 14, 14, 256)	
conv_pw_6 (Conv2D)	(None, 14, 14, 512)	
conv_pw_6_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_pw_6_relu (ReLU)	(None, 14, 14, 512)	
conv_dw_7 (DepthwiseConv2D)	(None, 14, 14, 512)	
conv_dw_7_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_dw_7_relu (ReLU)	(None, 14, 14, 512)	
conv_pw_7 (Conv2D)	(None, 14, 14, 512)	
conv_pw_7_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_pw_7_relu (ReLU)	(None, 14, 14, 512)	
conv_dw_8 (DepthwiseConv2D)	(None, 14, 14, 512)	
conv_dw_8_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_dw_8_relu (ReLU)	(None, 14, 14, 512)	
conv_pw_8 (Conv2D)	(None, 14, 14, 512)	
conv_pw_8_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_pw_8_relu (ReLU)	(None, 14, 14, 512)	

conv_dw_9 (DepthwiseConv2D)	(None, 14, 14, 512)	
conv_dw_9_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_dw_9_relu (ReLU)	(None, 14, 14, 512)	
conv_pw_9 (Conv2D)	(None, 14, 14, 512)	
conv_pw_9_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_pw_9_relu (ReLU)	(None, 14, 14, 512)	
conv_dw_10 (DepthwiseConv2D)	(None, 14, 14, 512)	
conv_dw_10_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_dw_10_relu (ReLU)	(None, 14, 14, 512)	
conv_pw_10 (Conv2D)	(None, 14, 14, 512)	
conv_pw_10_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_pw_10_relu (ReLU)	(None, 14, 14, 512)	
conv_dw_11 (DepthwiseConv2D)	(None, 14, 14, 512)	
conv_dw_11_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_dw_11_relu (ReLU)	(None, 14, 14, 512)	
conv_pw_11 (Conv2D)	(None, 14, 14, 512)	
conv_pw_11_bn (BatchNormalization)	(None, 14, 14, 512)	
conv_pw_11_relu (ReLU)	(None, 14, 14, 512)	
conv_pad_12 (ZeroPadding2D)	(None, 15, 15, 512)	
conv_dw_12 (DepthwiseConv2D)	(None, 7, 7, 512)	
conv_dw_12_bn (BatchNormalization)	(None, 7, 7, 512)	
conv_dw_12_relu (ReLU)	(None, 7, 7, 512)	
conv_pw_12 (Conv2D)	(None, 7, 7, 1024)	
conv_pw_12_bn (BatchNormalization)	(None, 7, 7, 1024)	
conv_pw_12_relu (ReLU)	(None, 7, 7, 1024)	
conv_dw_13 (DepthwiseConv2D)	(None, 7, 7, 1024)	
conv_dw_13_bn (BatchNormalization)	(None, 7, 7, 1024)	
conv_dw_13_relu (ReLU)	(None, 7, 7, 1024)	

conv_pw_13 (Conv2D)	(None, 7, 7, 1024)	
conv_pw_13_bn (BatchNormalization)	(None, 7, 7, 1024)	
conv_pw_13_relu (ReLU)	(None, 7, 7, 1024)	

Total params: 3,228,864 (12.32 MB)

Trainable params: 0 (0.00 B)

Non-trainable params: 3,228,864 (12.32 MB)

Build an FCN

Get training and validation set at 70:30 ratio and seed 1

In [ ]:
```
train_ds = get_ds(data_dir,validation_split=0.3,subset='training',seed = 1,image_si
val_ds = get_ds(data_dir,validation_split=0.3,subset='validation',seed = 1,image_si
```

```
Found 1162 files belonging to 9 classes.
Using 814 files for training.
Found 1162 files belonging to 9 classes.
Using 348 files for validation.
```

Set up dataset with caching and prefetch

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

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Add a FCN to the mobile net model(no data augmentation)

In [ ]:
```
model_t = create_transfer_model(mob_model)
model_t.name = 'Transfer_Model'
```

Model: "sequential_7"

Layer (type)	Output Shape	
rescaling_7 (Rescaling)	(None, 224, 224, 3)	
MobileNet (Functional)	(None, 7, 7, 1024)	
global_average_pooling2d_7 (GlobalAveragePooling2D)	(None, 1024)	
dropout_19 (Dropout)	(None, 1024)	
flatten_7 (Flatten)	(None, 1024)	
dense_21 (Dense)	(None, 256)	
batch_normalization_6 (BatchNormalization)	(None, 256)	
dropout_20 (Dropout)	(None, 256)	
dense_22 (Dense)	(None, 128)	
batch_normalization_7 (BatchNormalization)	(None, 128)	
dropout_21 (Dropout)	(None, 128)	
dense_23 (Dense)	(None, 9)	

Total params: 3,526,857 (13.45 MB)

Trainable params: 297,225 (1.13 MB)

Non-trainable params: 3,229,632 (12.32 MB)

Complile the model

In [ ]:
```
model_t.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=[
print('model_t compiled')
```

```
model_t compiled
```

Configure for early stopping and Checkpoint and train the model

In [ ]:
```
early_stop=EarlyStopping(monitor='val_accuracy',mode='max',verbose=1,patience=10,mi
checkpoint=ModelCheckpoint('best_model_t.keras',monitor='val_accuracy',mode='max',v

epochs=100
history_t = model_t.fit(
  train_ds,
  validation_data=val_ds,
```

```
  epochs=epochs,
  batch_size = batch_size,
  callbacks = [early_stop,checkpoint]
)

histories.append(history_t)
```

```
Epoch 1/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step - accuracy: 0.4443 - loss: 1.7191
Epoch 1: val_accuracy improved from -inf to 0.63506, saving model to best_model_t.ke
ras
26/26 ━━━━━━━━━━━━━━━━━━━━ 14s 263ms/step - accuracy: 0.4504 - loss: 1.7020 - val_ac
curacy: 0.6351 - val_loss: 1.1627
Epoch 2/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8644 - loss: 0.4747
Epoch 2: val_accuracy improved from 0.63506 to 0.75000, saving model to best_model_
t.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.8654 - loss: 0.4709 - val_accu
racy: 0.7500 - val_loss: 0.7554
Epoch 3/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9494 - loss: 0.2022
Epoch 3: val_accuracy improved from 0.75000 to 0.81897, saving model to best_model_
t.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9488 - loss: 0.2033 - val_accu
racy: 0.8190 - val_loss: 0.5880
Epoch 4/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9734 - loss: 0.1345
Epoch 4: val_accuracy improved from 0.81897 to 0.83046, saving model to best_model_
t.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9738 - loss: 0.1337 - val_accu
racy: 0.8305 - val_loss: 0.5583
Epoch 5/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9918 - loss: 0.0761
Epoch 5: val_accuracy improved from 0.83046 to 0.83621, saving model to best_model_
t.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9916 - loss: 0.0765 - val_accu
racy: 0.8362 - val_loss: 0.5430
Epoch 6/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9971 - loss: 0.0521
Epoch 6: val_accuracy improved from 0.83621 to 0.85057, saving model to best_model_
t.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9970 - loss: 0.0527 - val_accu
racy: 0.8506 - val_loss: 0.5130
Epoch 7/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9991 - loss: 0.0378
Epoch 7: val_accuracy did not improve from 0.85057
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9991 - loss: 0.0381 - val_accu
racy: 0.8362 - val_loss: 0.5311
Epoch 8/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9929 - loss: 0.0428
Epoch 8: val_accuracy did not improve from 0.85057
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9931 - loss: 0.0423 - val_accu
racy: 0.8448 - val_loss: 0.5245
Epoch 9/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9946 - loss: 0.0288
Epoch 9: val_accuracy did not improve from 0.85057
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9946 - loss: 0.0291 - val_accu
racy: 0.8506 - val_loss: 0.5401
Epoch 10/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 1.0000 - loss: 0.0213
Epoch 10: val_accuracy did not improve from 0.85057
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 1.0000 - loss: 0.0215 - val_accu
racy: 0.8448 - val_loss: 0.5676
```

```
Epoch 11/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9983 - loss: 0.0242
Epoch 11: val_accuracy did not improve from 0.85057
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9983 - loss: 0.0241 - val_accu
racy: 0.8362 - val_loss: 0.5963
Epoch 12/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9984 - loss: 0.0224
Epoch 12: val_accuracy did not improve from 0.85057
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9984 - loss: 0.0223 - val_accu
racy: 0.8448 - val_loss: 0.5942
Epoch 13/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9994 - loss: 0.0180
Epoch 13: val_accuracy did not improve from 0.85057
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9993 - loss: 0.0178 - val_accu
racy: 0.8506 - val_loss: 0.5981
Epoch 14/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 1.0000 - loss: 0.0142
Epoch 14: val_accuracy did not improve from 0.85057
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 1.0000 - loss: 0.0141 - val_accu
racy: 0.8391 - val_loss: 0.6027
Epoch 15/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9989 - loss: 0.0161
Epoch 15: val_accuracy did not improve from 0.85057
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9989 - loss: 0.0161 - val_accu
racy: 0.8420 - val_loss: 0.5770
Epoch 16/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9997 - loss: 0.0098
Epoch 16: val_accuracy improved from 0.85057 to 0.85345, saving model to best_model_
t.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9997 - loss: 0.0099 - val_accu
racy: 0.8534 - val_loss: 0.5668
Epoch 16: early stopping
```

Print metrics

In [ ]:
```
from tensorflow.keras.models import load_model
model_t = load_model('best_model_t.keras')
print_scores(model_t,val_ds,train_ds,'Transfer Model')
```

```
Displaying accuracy and loss for Transfer Model
Test loss:0.5668
Test accuracy:0.8534
Train loss:0.0064
train accuracy:0.9988
```

Severe overfitting can be observed. We will modify this model by increasing dropouts. The below model uses a droput that i have arrived at after trial and error which produced lesser overfitting

Dropout adjusted Transfer Model(no augmentation)

In [ ]:
```
model_t1 = create_transfer_model(mob_model,dropout=0.4)
```

Model: "sequential_8"

Layer (type)	Output Shape	
rescaling_8 (Rescaling)	(None, 224, 224, 3)	
MobileNet (Functional)	(None, 7, 7, 1024)	
global_average_pooling2d_8 (GlobalAveragePooling2D)	(None, 1024)	
dropout_22 (Dropout)	(None, 1024)	
flatten_8 (Flatten)	(None, 1024)	
dense_24 (Dense)	(None, 256)	
batch_normalization_8 (BatchNormalization)	(None, 256)	
dropout_23 (Dropout)	(None, 256)	
dense_25 (Dense)	(None, 128)	
batch_normalization_9 (BatchNormalization)	(None, 128)	
dropout_24 (Dropout)	(None, 128)	
dense_26 (Dense)	(None, 9)	

Total params: 3,526,857 (13.45 MB)

Trainable params: 297,225 (1.13 MB)

Non-trainable params: 3,229,632 (12.32 MB)

Compile the model

In [ ]:
```
model_t1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=
print('model_t1 compiled')
```

```
model_t1 compiled
```

Train the model

In [ ]:
```
early_stop=EarlyStopping(monitor='val_accuracy',mode='max',verbose=1,patience=10,mi
checkpoint=ModelCheckpoint('best_model_t1.keras',monitor='val_accuracy',mode='max',

epochs=100
history_t1 = model_t1.fit(
  train_ds,
  validation_data=val_ds,
```

```
  epochs=epochs,
  batch_size = batch_size,
  callbacks = [early_stop,checkpoint]
)

histories.append(history_t1)
```

```
Epoch 1/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step - accuracy: 0.1931 - loss: 2.7958
Epoch 1: val_accuracy improved from -inf to 0.66954, saving model to best_model_t1.k
eras
26/26 ━━━━━━━━━━━━━━━━━━━━ 12s 249ms/step - accuracy: 0.2018 - loss: 2.7578 - val_ac
curacy: 0.6695 - val_loss: 1.0688
Epoch 2/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.5864 - loss: 1.2809
Epoch 2: val_accuracy improved from 0.66954 to 0.74138, saving model to best_model_t
1.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.5901 - loss: 1.2710 - val_accu
racy: 0.7414 - val_loss: 0.9228
Epoch 3/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.7302 - loss: 0.9100
Epoch 3: val_accuracy improved from 0.74138 to 0.78448, saving model to best_model_t
1.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.7307 - loss: 0.9077 - val_accu
racy: 0.7845 - val_loss: 0.7035
Epoch 4/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.7770 - loss: 0.7447
Epoch 4: val_accuracy improved from 0.78448 to 0.79310, saving model to best_model_t
1.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.7769 - loss: 0.7444 - val_accu
racy: 0.7931 - val_loss: 0.6559
Epoch 5/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8037 - loss: 0.5972
Epoch 5: val_accuracy improved from 0.79310 to 0.81034, saving model to best_model_t
1.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.8031 - loss: 0.5992 - val_accu
racy: 0.8103 - val_loss: 0.6076
Epoch 6/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.8101 - loss: 0.5960
Epoch 6: val_accuracy did not improve from 0.81034
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.8098 - loss: 0.5953 - val_accu
racy: 0.8075 - val_loss: 0.5836
Epoch 7/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8543 - loss: 0.4423
Epoch 7: val_accuracy did not improve from 0.81034
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.8540 - loss: 0.4457 - val_accu
racy: 0.8103 - val_loss: 0.5712
Epoch 8/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.8315 - loss: 0.4871
Epoch 8: val_accuracy improved from 0.81034 to 0.82184, saving model to best_model_t
1.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.8312 - loss: 0.4868 - val_accu
racy: 0.8218 - val_loss: 0.5730
Epoch 9/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.8515 - loss: 0.4685
Epoch 9: val_accuracy did not improve from 0.82184
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.8521 - loss: 0.4666 - val_accu
racy: 0.8218 - val_loss: 0.5616
Epoch 10/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.8949 - loss: 0.3810
Epoch 10: val_accuracy improved from 0.82184 to 0.83908, saving model to best_model_
t1.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.8938 - loss: 0.3825 - val_accu
```

racy: 0.8391 - val_loss: 0.5261
Epoch 11/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.8752 - loss: 0.3914
Epoch 11: val_accuracy improved from 0.83908 to 0.84483, saving model to best_model_
t1.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ **1s** 43ms/step - accuracy: 0.8753 - loss: 0.3905 - val_accu
racy: 0.8448 - val_loss: 0.5041
Epoch 12/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.8810 - loss: 0.3801
Epoch 12: val_accuracy did not improve from 0.84483
26/26 ━━━━━━━━━━━━━━━━━━━━ **1s** 33ms/step - accuracy: 0.8811 - loss: 0.3777 - val_accu
racy: 0.8420 - val_loss: 0.5128
Epoch 13/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.9208 - loss: 0.2581
Epoch 13: val_accuracy improved from 0.84483 to 0.84770, saving model to best_model_
t1.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ **1s** 43ms/step - accuracy: 0.9194 - loss: 0.2609 - val_accu
racy: 0.8477 - val_loss: 0.5019
Epoch 14/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.8953 - loss: 0.2992
Epoch 14: val_accuracy did not improve from 0.84770
26/26 ━━━━━━━━━━━━━━━━━━━━ **1s** 33ms/step - accuracy: 0.8950 - loss: 0.2996 - val_accu
racy: 0.8420 - val_loss: 0.4931
Epoch 15/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.9088 - loss: 0.2837
Epoch 15: val_accuracy did not improve from 0.84770
26/26 ━━━━━━━━━━━━━━━━━━━━ **1s** 33ms/step - accuracy: 0.9099 - loss: 0.2807 - val_accu
racy: 0.8391 - val_loss: 0.5045
Epoch 16/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.9237 - loss: 0.2536
Epoch 16: val_accuracy improved from 0.84770 to 0.85345, saving model to best_model_
t1.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ **1s** 43ms/step - accuracy: 0.9232 - loss: 0.2542 - val_accu
racy: 0.8534 - val_loss: 0.4847
Epoch 17/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.9011 - loss: 0.2985
Epoch 17: val_accuracy did not improve from 0.85345
26/26 ━━━━━━━━━━━━━━━━━━━━ **1s** 33ms/step - accuracy: 0.9023 - loss: 0.2950 - val_accu
racy: 0.8506 - val_loss: 0.4887
Epoch 18/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.9077 - loss: 0.2545
Epoch 18: val_accuracy did not improve from 0.85345
26/26 ━━━━━━━━━━━━━━━━━━━━ **1s** 33ms/step - accuracy: 0.9087 - loss: 0.2531 - val_accu
racy: 0.8448 - val_loss: 0.5326
Epoch 19/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.9525 - loss: 0.1505
Epoch 19: val_accuracy did not improve from 0.85345
26/26 ━━━━━━━━━━━━━━━━━━━━ **1s** 33ms/step - accuracy: 0.9515 - loss: 0.1542 - val_accu
racy: 0.8448 - val_loss: 0.5286
Epoch 20/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.8927 - loss: 0.2991
Epoch 20: val_accuracy did not improve from 0.85345
26/26 ━━━━━━━━━━━━━━━━━━━━ **1s** 33ms/step - accuracy: 0.8949 - loss: 0.2936 - val_accu
racy: 0.8534 - val_loss: 0.5113
Epoch 21/100
25/26 ━━━━━━━━━━━━━━━━━━━━ **0s** 23ms/step - accuracy: 0.9301 - loss: 0.1972

```
Epoch 21: val_accuracy improved from 0.85345 to 0.86207, saving model to best_model_
t1.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9308 - loss: 0.1971 - val_accu
racy: 0.8621 - val_loss: 0.4871
Epoch 22/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9414 - loss: 0.1773
Epoch 22: val_accuracy did not improve from 0.86207
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9413 - loss: 0.1782 - val_accu
racy: 0.8592 - val_loss: 0.5078
Epoch 23/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9329 - loss: 0.1977
Epoch 23: val_accuracy did not improve from 0.86207
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9330 - loss: 0.1972 - val_accu
racy: 0.8563 - val_loss: 0.5152
Epoch 24/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9255 - loss: 0.1991
Epoch 24: val_accuracy did not improve from 0.86207
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9262 - loss: 0.1980 - val_accu
racy: 0.8477 - val_loss: 0.5228
Epoch 25/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9411 - loss: 0.1728
Epoch 25: val_accuracy did not improve from 0.86207
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9417 - loss: 0.1718 - val_accu
racy: 0.8592 - val_loss: 0.5448
Epoch 26/100
24/26 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9517 - loss: 0.1329
Epoch 26: val_accuracy did not improve from 0.86207
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9505 - loss: 0.1369 - val_accu
racy: 0.8563 - val_loss: 0.5329
Epoch 26: early stopping
```

Print metrics

In [ ]:
```
from tensorflow.keras.models import load_model
model_t1 = load_model('best_model_t1.keras')
print_scores(model_t1,val_ds,train_ds,'Dropout Adjusted Transfer Model')
# plot_accuracy_loss_graphs(history_t1,'Dropout adjusted transfer model')
```

```
Displaying accuracy and loss for Dropout Adjusted Transfer Model
Test loss:0.4871
Test accuracy:0.8621
Train loss:0.0267
train accuracy:0.9975
```

Increaing dropout to 0.4 has reduced overfitting a little

Augmented Transfer Model

In [ ]:
```
data_augmentation = [layers.RandomFlip("horizontal",input_shape=(height,width,3)),l
model_augmented = create_transfer_model(mob_model,dropout=0.4,data_augmentation=dat
```

```
added <RandomFlip name=random_flip_1, built=False> to model
added <RandomRotation name=random_rotation_1, built=False> to model
added <RandomZoom name=random_zoom_1, built=False> to model
```

Model: "sequential_9"

Layer (type)	Output Shape	
random_flip_1 (RandomFlip)	(None, 224, 224, 3)	
random_rotation_1 (RandomRotation)	(None, 224, 224, 3)	
random_zoom_1 (RandomZoom)	(None, 224, 224, 3)	
rescaling_9 (Rescaling)	(None, 224, 224, 3)	
MobileNet (Functional)	(None, 7, 7, 1024)	
global_average_pooling2d_9 (GlobalAveragePooling2D)	(None, 1024)	
dropout_25 (Dropout)	(None, 1024)	
flatten_9 (Flatten)	(None, 1024)	
dense_27 (Dense)	(None, 256)	
batch_normalization_10 (BatchNormalization)	(None, 256)	
dropout_26 (Dropout)	(None, 256)	
dense_28 (Dense)	(None, 128)	
batch_normalization_11 (BatchNormalization)	(None, 128)	
dropout_27 (Dropout)	(None, 128)	
dense_29 (Dense)	(None, 9)	

Total params: 3,526,857 (13.45 MB)

Trainable params: 297,225 (1.13 MB)

Non-trainable params: 3,229,632 (12.32 MB)

Compile the model

In [ ]:
```
model_augmented.compile(optimizer='adam', loss='sparse_categorical_crossentropy', m
print('model_augmented compiled')
```

```
model_augmented compiled
```

Train the model

In [ ]:

```
early_stop=EarlyStopping(monitor='val_accuracy',mode='max',verbose=1,patience=10,mi
checkpoint=ModelCheckpoint('best_model_a.keras',monitor='val_accuracy',mode='max',v

epochs=100
history_a = model_augmented.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  batch_size = batch_size,
  callbacks = [early_stop,checkpoint]
)

histories.append(history_a)
```

```
Epoch 1/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.1496 - loss: 3.0357
Epoch 1: val_accuracy improved from -inf to 0.57471, saving model to best_model_a.ke
ras
26/26 ━━━━━━━━━━━━━━━━━━━━ 6s 101ms/step - accuracy: 0.1573 - loss: 3.0023 - val_acc
uracy: 0.5747 - val_loss: 1.4767
Epoch 2/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.5069 - loss: 1.6670
Epoch 2: val_accuracy improved from 0.57471 to 0.69540, saving model to best_model_
a.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.5099 - loss: 1.6567 - val_accu
racy: 0.6954 - val_loss: 1.0913
Epoch 3/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.6366 - loss: 1.1682
Epoch 3: val_accuracy improved from 0.69540 to 0.75000, saving model to best_model_
a.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.6382 - loss: 1.1690 - val_accu
racy: 0.7500 - val_loss: 0.8929
Epoch 4/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.6630 - loss: 1.0743
Epoch 4: val_accuracy improved from 0.75000 to 0.75862, saving model to best_model_
a.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.6630 - loss: 1.0738 - val_accu
racy: 0.7586 - val_loss: 0.8364
Epoch 5/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.7214 - loss: 0.9195
Epoch 5: val_accuracy improved from 0.75862 to 0.79598, saving model to best_model_
a.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.7227 - loss: 0.9156 - val_accu
racy: 0.7960 - val_loss: 0.7289
Epoch 6/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.7442 - loss: 0.8462
Epoch 6: val_accuracy did not improve from 0.79598
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.7440 - loss: 0.8461 - val_accu
racy: 0.7902 - val_loss: 0.6744
Epoch 7/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.7408 - loss: 0.8022
Epoch 7: val_accuracy improved from 0.79598 to 0.81322, saving model to best_model_
a.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - accuracy: 0.7422 - loss: 0.7980 - val_accu
racy: 0.8132 - val_loss: 0.6340
Epoch 8/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.7761 - loss: 0.7034
Epoch 8: val_accuracy improved from 0.81322 to 0.83046, saving model to best_model_
a.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - accuracy: 0.7748 - loss: 0.7089 - val_accu
racy: 0.8305 - val_loss: 0.6193
Epoch 9/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.7796 - loss: 0.6652
Epoch 9: val_accuracy did not improve from 0.83046
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.7798 - loss: 0.6653 - val_accu
racy: 0.8276 - val_loss: 0.5868
Epoch 10/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8015 - loss: 0.6470
Epoch 10: val_accuracy improved from 0.83046 to 0.83621, saving model to best_model_
a.keras
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 73ms/step - accuracy: 0.8008 - loss: 0.6476 - val_accu
racy: 0.8362 - val_loss: 0.5887
Epoch 11/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.7641 - loss: 0.6470
Epoch 11: val_accuracy did not improve from 0.83621
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.7656 - loss: 0.6454 - val_accu
racy: 0.8247 - val_loss: 0.5875
Epoch 12/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8203 - loss: 0.5679
Epoch 12: val_accuracy did not improve from 0.83621
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.8197 - loss: 0.5696 - val_accu
racy: 0.8247 - val_loss: 0.6251
Epoch 13/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8355 - loss: 0.5282
Epoch 13: val_accuracy did not improve from 0.83621
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.8338 - loss: 0.5311 - val_accu
racy: 0.8333 - val_loss: 0.5653
Epoch 14/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8180 - loss: 0.5351
Epoch 14: val_accuracy improved from 0.83621 to 0.83908, saving model to best_model_
a.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.8179 - loss: 0.5359 - val_accu
racy: 0.8391 - val_loss: 0.5499
Epoch 15/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8382 - loss: 0.5065
Epoch 15: val_accuracy improved from 0.83908 to 0.84195, saving model to best_model_
a.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - accuracy: 0.8395 - loss: 0.5016 - val_accu
racy: 0.8420 - val_loss: 0.5385
Epoch 16/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8522 - loss: 0.4531
Epoch 16: val_accuracy improved from 0.84195 to 0.85632, saving model to best_model_
a.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.8519 - loss: 0.4539 - val_accu
racy: 0.8563 - val_loss: 0.5208
Epoch 17/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8400 - loss: 0.5243
Epoch 17: val_accuracy did not improve from 0.85632
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8400 - loss: 0.5238 - val_accu
racy: 0.8420 - val_loss: 0.5117
Epoch 18/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.8335 - loss: 0.4532
Epoch 18: val_accuracy did not improve from 0.85632
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8335 - loss: 0.4533 - val_accu
racy: 0.8506 - val_loss: 0.5156
Epoch 19/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8402 - loss: 0.4745
Epoch 19: val_accuracy improved from 0.85632 to 0.86494, saving model to best_model_
a.keras
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.8396 - loss: 0.4747 - val_accu
racy: 0.8649 - val_loss: 0.5149
Epoch 20/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8213 - loss: 0.5143
Epoch 20: val_accuracy did not improve from 0.86494
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8225 - loss: 0.5128 - val_accu
racy: 0.8420 - val_loss: 0.5512
```

```
Epoch 21/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8384 - loss: 0.4986
Epoch 21: val_accuracy did not improve from 0.86494
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8387 - loss: 0.4962 - val_accu
racy: 0.8448 - val_loss: 0.5497
Epoch 22/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8576 - loss: 0.4206
Epoch 22: val_accuracy did not improve from 0.86494
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8575 - loss: 0.4212 - val_accu
racy: 0.8391 - val_loss: 0.5469
Epoch 23/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8533 - loss: 0.4191
Epoch 23: val_accuracy did not improve from 0.86494
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8530 - loss: 0.4190 - val_accu
racy: 0.8534 - val_loss: 0.5381
Epoch 24/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8595 - loss: 0.4622
Epoch 24: val_accuracy did not improve from 0.86494
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8599 - loss: 0.4589 - val_accu
racy: 0.8477 - val_loss: 0.5954
Epoch 25/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8685 - loss: 0.4034
Epoch 25: val_accuracy did not improve from 0.86494
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8691 - loss: 0.4048 - val_accu
racy: 0.8420 - val_loss: 0.5749
Epoch 26/100
25/26 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8640 - loss: 0.4186
Epoch 26: val_accuracy did not improve from 0.86494
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8641 - loss: 0.4179 - val_accu
racy: 0.8448 - val_loss: 0.5551
Epoch 26: early stopping
```

Print accuracy and loss curves

In [ ]:
```python
from tensorflow.keras.models import load_model
model_a = load_model('best_model_a.keras')
print_scores(model_t1,val_ds,train_ds,'Augmented Transfer Model')
# plot_accuracy_loss_graphs(history_a,'Augmented transfer model')
```

```
Displaying accuracy and loss for Augmented Transfer Model
Test loss:0.4871
Test accuracy:0.8621
Train loss:0.0267
train accuracy:0.9975
```

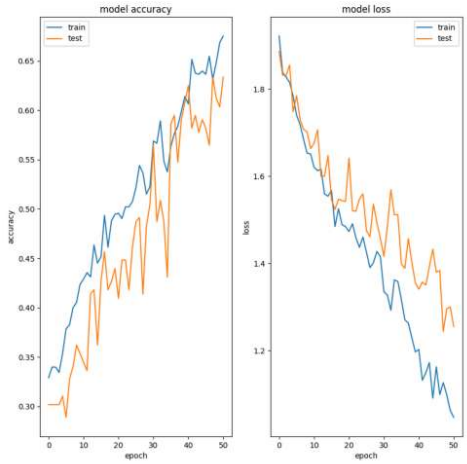
Plot loss/accuracy curves for all models

The final accuracy and loss are the same for augmented ad non augmented models. The augmented model has a slightly lesser ovetfitting

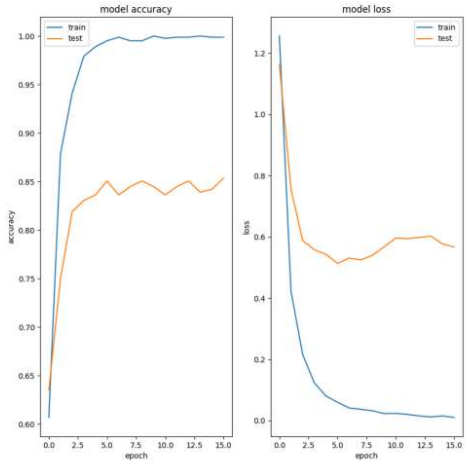
In [ ]:
```python
import matplotlib.image as mpimg
histories = [history_c,history_t,history_t1,history_a]
#titles = ['CNN Model','Transfer Learning model using MobileNet','Dropout adjusted
```

```
#print(test_accuracy.keys())

titles = list(test_accuracy.keys())

for i,history in enumerate(histories):
  plot_accuracy_loss_graphs(history,titles[i],show=False)
plt.figure(figsize=(16,16))
for i,title in enumerate(titles):
  title = titles[i]
  plt.subplot(2,2,i+1)
  figure = title+'.png'
  img = mpimg.imread(figure)
  plt.imshow(img)
  plt.axis('off')
  accuracy =  test_accuracy[title]
  loss = test_loss[title]
  title = title + f'\n(validation accuracy = {accuracy:.4f}, loss = {loss:.4f})'
  plt.title(title)
plt.show()
```

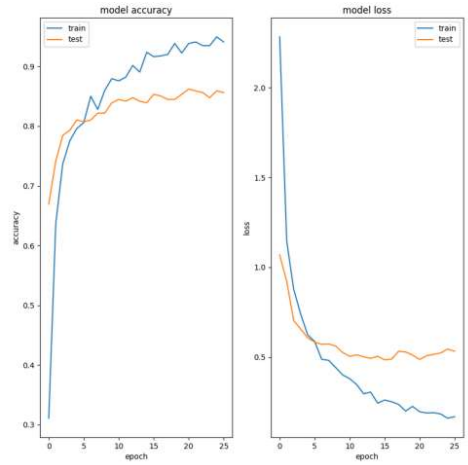
CNN Model
(validation accuracy = 0.6336, loss = 1.2437)



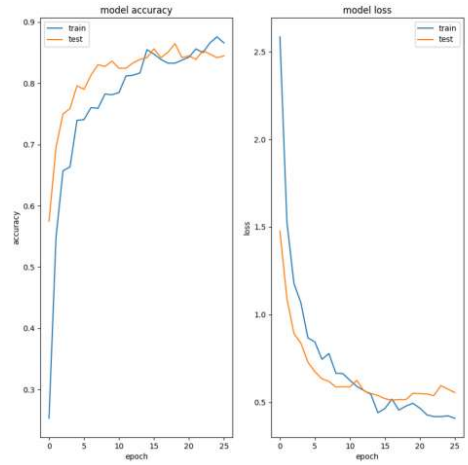
Transfer Model
(validation accuracy = 0.8534, loss = 0.5668)



Dropout Adjusted Transfer Model
(validation accuracy = 0.8621, loss = 0.4871)



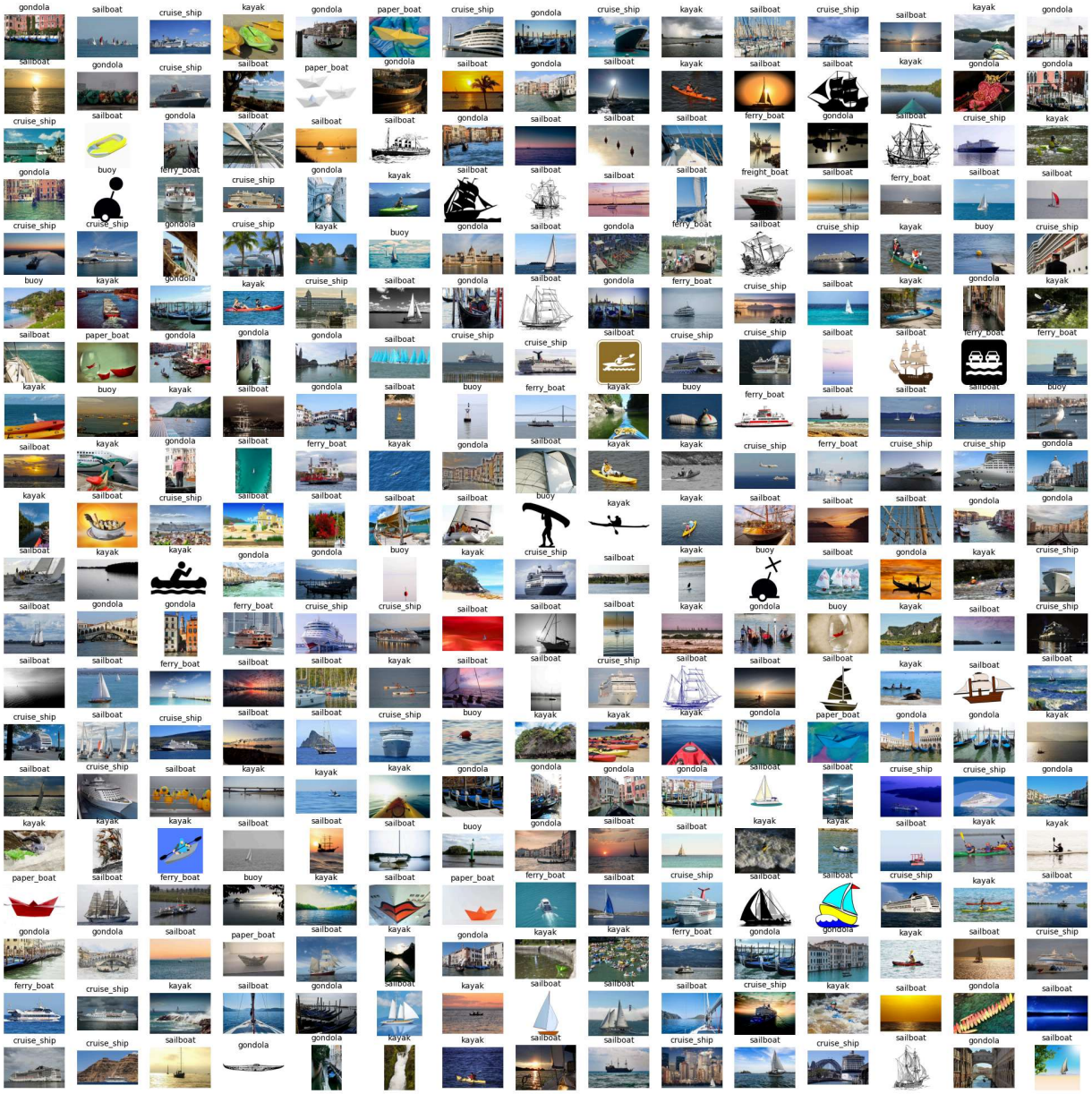
Augmented Transfer Model
(validation accuracy = 0.8621, loss = 0.4871)



Visualize the predictions

In [ ]:
```
image_map = get_predicted_labels(model_a,class_names)
print('size of image map = ',len(image_map))
display_test_images(image_map,'Transfer Learning model using MobileNet. Title is th
```

```
size of image map =  300
Predictions for Transfer Learning model using MobileNet. Title is the prediction
```



The predictions with the transfer model is much more accurate (expected as accuracy is 86%) when commpared to the CNN model. We can see that even classes with very few train images have been correctly identified(paper boat for example)

Metrics

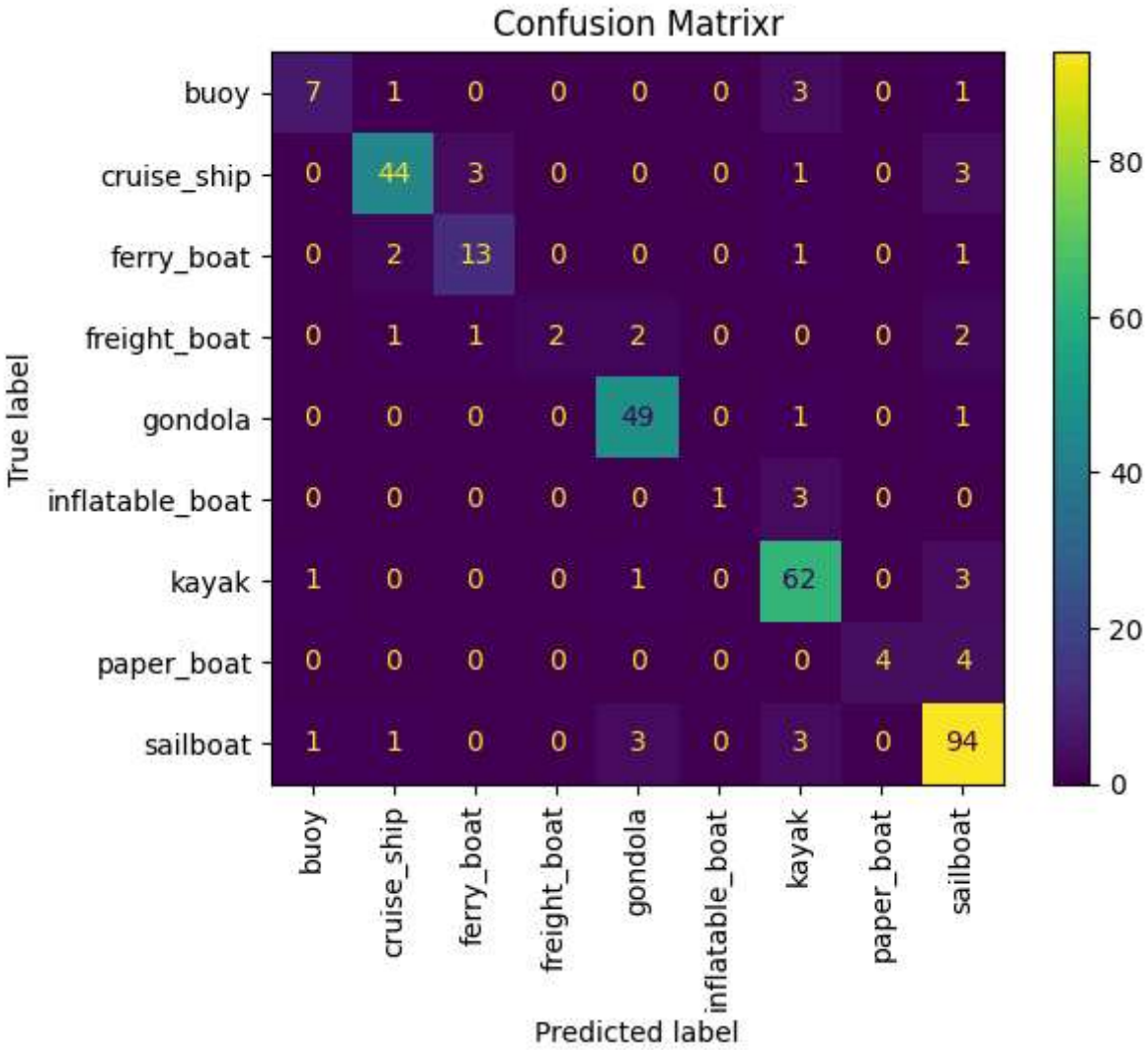
In [ ]:
```
true_labels, predicted_labels = get_metrics(model_a,val_ds,val_split=0.3)
```

Confusion Matrix

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(true_labels,predicted_labels)
print('Confusion Matrix')
print(cm)
print()
```

```
print('Confusion Matrix Displayed\n')
plt.figure(figsize = (10,10))
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=class_names)
disp.plot()
plt.title('Confusion Matrixr')
plt.xticks(rotation=90)
plt.show()
```

```
Confusion Matrix
[[ 7  1  0  0  0  0  3  0  1]
 [ 0 44  3  0  0  0  1  0  3]
 [ 0  2 13  0  0  0  1  0  1]
 [ 0  1  1  2  2  0  0  0  2]
 [ 0  0  0  0 49  0  1  0  1]
 [ 0  0  0  0  0  1  3  0  0]
 [ 1  0  0  0  1  0 62  0  3]
 [ 0  0  0  0  0  0  0  4  4]
 [ 1  1  0  0  3  0  3  0 94]]

Confusion Matrix Displayed

<Figure size 1000x1000 with 0 Axes>
```



Classification Report

In [ ]:
```
from sklearn.metrics import classification_report

class_name_dict = {
    0 : ''
}

cl_report = classification_report(true_labels,predicted_labels)
print(cl_report)

print('Index to name mapping:\n')

for k,v in class_dict.items():
    print(f'{k} - > {v}')
```

```
              precision    recall  f1-score   support

           0       0.78      0.58      0.67        12
           1       0.90      0.86      0.88        51
           2       0.76      0.76      0.76        17
           3       1.00      0.25      0.40         8
           4       0.89      0.96      0.92        51
           5       1.00      0.25      0.40         4
           6       0.84      0.93      0.88        67
           7       1.00      0.50      0.67         8
           8       0.86      0.92      0.89       102

    accuracy                           0.86       320
   macro avg       0.89      0.67      0.72       320
weighted avg       0.87      0.86      0.85       320

Index to name mapping:

0 - > inflatable_boat
1 - > freight_boat
2 - > paper_boat
3 - > buoy
4 - > ferry_boat
5 - > cruise_ship
6 - > gondola
7 - > kayak
8 - > sailboat
```

As expected the classes that have higher support(more train images) have better overall scores

Inferences

CNN Model

The CNN model built in part 1 of this project gave an accuracy of around 65%. When tested on the 300 test images, we could see many mis-classificatons. This could be because the training set contained just 1162 images and the code used only 80% of that for training. This is a very low number for the model to learn well. Given this limitation, an accuracy of 55-65% is good. The model has a bit of overfitting, though :

Transfer Learning

Accuracy significantly increases even with plain vanilla transfer model (Mobilenet_V2). However there is significant over fitting with low dropout values.

With higher dropout values, overfitting gets reduced without sacrificing accuracy

Higher dropout value with additional image augmentation reduces overfitting further without compromising accuracy

The higher accuracy is evident in the number of correct predictions that can be seen in the confusion matrix and classification report. This has been verified visually too on the unlabelled test data set

Conclusion

For custom daasets that do not have sufficiently large number of images, transfer learning is the preferred method of choice. By freezing the model's weights and then fine tuning with a fully connected layer gives us a very accurrate model that otherwise would not have been possible.