Automating Port Operations

This notebook was compiled by Neelkantho Bose as part of completing the course titled "Deep Learning".

To perform the following tasks:

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.

Note:

1. There may be cells that contain rough code (mostly commented out). Pls ignore them.

1.1 : Import the necessary libraries and dataset

- Import the required libraries.
- Import the given dataset using the link provided.

In [48]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from PIL import Image
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import pathlib
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Dense, BatchNormal
from tensorflow.keras.models import Sequential
```

In [42]:
```
# Set parameters for dataset
batch_size = 32
img_height = 256
img_width = 256

directory_path='C:\\Users\\neelk\\OneDrive\\Desktop\\Simplilearn-DataScience-Course\\Co

# Create the dataset
train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
```

```
    directory_path,
    validation_split=0.2,  # Split the dataset into training and validation
    subset="training",  # Use this part as the training data
    seed=123,  # Seed for reproducibility
    image_size=(256, 256),  # Resize images to 256x256
    batch_size=32,  # Number of images to retrieve in one batch,
    shuffle=True
)

# Create the dataset
validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    directory_path,
    validation_split=0.2,  # Split the dataset into training and validation
    subset="validation",  # Use this part as the training data
    seed=123,  # Seed for reproducibility
    image_size=(256, 256),  # Resize images to 256x256
    batch_size=32,  # Number of images to retrieve in one batch
    shuffle=True
)


# Get the class names
tr_class_names = train_dataset.class_names
print("Training Class names:", tr_class_names)

val_class_names = validation_dataset.class_names
print("Validation Class names:", val_class_names)


# Visualize some images
import matplotlib.pyplot as plt

print("Training Dataset sample 10 images")
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(tr_class_names[labels[i]])
        plt.axis("off")
plt.show()
```

```
Found 1162 files belonging to 9 classes.
Using 930 files for training.
Found 1162 files belonging to 9 classes.
Using 232 files for validation.
Training Class names: ['buoy', 'cruise_ship', 'ferry_boat', 'freight_boat', 'gondola',
'inflatable_boat', 'kayak', 'paper_boat', 'sailboat']
Validation Class names: ['buoy', 'cruise_ship', 'ferry_boat', 'freight_boat', 'gondol
a', 'inflatable_boat', 'kayak', 'paper_boat', 'sailboat']
Training Dataset sample 10 images
```

In [43]:
```
# Set parameters for dataset
batch_size = 32
img_height = 256
img_width = 256

# Preprocess datasets
AUTOTUNE = tf.data.AUTOTUNE

def preprocess(image, label):
    image = tf.keras.layers.Rescaling(1./255)(image)
    return image, label

train_dataset = train_dataset.map(preprocess, num_parallel_calls=AUTOTUNE)
validation_dataset = validation_dataset.map(preprocess, num_parallel_calls=AUTOTUNE)

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)

# Define and compile the model
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(9, activation='softmax')
])

model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)

# Fit the model
epochs = 20
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=epochs
)
```

```
Epoch 1/20
30/30 [==============================] - 42s 614ms/step - loss: 1.9219 - accuracy: 0.32
26 - val_loss: 1.8887 - val_accuracy: 0.2759
Epoch 2/20
30/30 [==============================] - 21s 678ms/step - loss: 1.7820 - accuracy: 0.34
95 - val_loss: 1.9039 - val_accuracy: 0.2759
Epoch 3/20
30/30 [==============================] - 23s 752ms/step - loss: 1.7748 - accuracy: 0.34
95 - val_loss: 1.8853 - val_accuracy: 0.2759
Epoch 4/20
30/30 [==============================] - 23s 746ms/step - loss: 1.7716 - accuracy: 0.34
95 - val_loss: 1.8663 - val_accuracy: 0.2759
Epoch 5/20
30/30 [==============================] - 23s 739ms/step - loss: 1.7533 - accuracy: 0.35
05 - val_loss: 1.8838 - val_accuracy: 0.2888
Epoch 6/20
30/30 [==============================] - 22s 710ms/step - loss: 1.7433 - accuracy: 0.35
81 - val_loss: 1.8874 - val_accuracy: 0.2974
Epoch 7/20
30/30 [==============================] - 22s 703ms/step - loss: 1.7145 - accuracy: 0.36
99 - val_loss: 1.8218 - val_accuracy: 0.3017
Epoch 8/20
30/30 [==============================] - 24s 763ms/step - loss: 1.6891 - accuracy: 0.37
42 - val_loss: 1.7965 - val_accuracy: 0.3707
Epoch 9/20
30/30 [==============================] - 26s 817ms/step - loss: 1.7022 - accuracy: 0.37
20 - val_loss: 1.7971 - val_accuracy: 0.3448
Epoch 10/20
30/30 [==============================] - 29s 924ms/step - loss: 1.6626 - accuracy: 0.38
92 - val_loss: 1.8518 - val_accuracy: 0.3319
Epoch 11/20
30/30 [==============================] - 28s 889ms/step - loss: 1.6687 - accuracy: 0.38
71 - val_loss: 1.8219 - val_accuracy: 0.3621
Epoch 12/20
30/30 [==============================] - 31s 934ms/step - loss: 1.6570 - accuracy: 0.38
60 - val_loss: 1.7435 - val_accuracy: 0.3750
Epoch 13/20
30/30 [==============================] - 28s 910ms/step - loss: 1.6319 - accuracy: 0.39
68 - val_loss: 1.7687 - val_accuracy: 0.3836
Epoch 14/20
30/30 [==============================] - 30s 950ms/step - loss: 1.6199 - accuracy: 0.39
68 - val_loss: 1.7810 - val_accuracy: 0.3621
Epoch 15/20
30/30 [==============================] - 35s 1s/step - loss: 1.6528 - accuracy: 0.3806
- val_loss: 1.7648 - val_accuracy: 0.3319
Epoch 16/20
30/30 [==============================] - 30s 953ms/step - loss: 1.6084 - accuracy: 0.40
32 - val_loss: 1.7577 - val_accuracy: 0.3793
Epoch 17/20
30/30 [==============================] - 32s 952ms/step - loss: 1.6229 - accuracy: 0.39
46 - val_loss: 1.7052 - val_accuracy: 0.3879
Epoch 18/20
30/30 [==============================] - 29s 939ms/step - loss: 1.6068 - accuracy: 0.40
54 - val_loss: 1.7192 - val_accuracy: 0.3664
Epoch 19/20
30/30 [==============================] - 32s 1s/step - loss: 1.5773 - accuracy: 0.4172
- val_loss: 1.7241 - val_accuracy: 0.3793
Epoch 20/20
30/30 [==============================] - 31s 963ms/step - loss: 1.5899 - accuracy: 0.40
54 - val_loss: 1.6943 - val_accuracy: 0.3879
```

In [44]:
```
model.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_26 (Conv2D)          (None, 254, 254, 32)      896

 max_pooling2d_26 (MaxPooli  (None, 127, 127, 32)      0
 ng2D)

 conv2d_27 (Conv2D)          (None, 125, 125, 32)      9248

 max_pooling2d_27 (MaxPooli  (None, 62, 62, 32)        0
 ng2D)

 global_average_pooling2d_1  (None, 32)                0
 3 (GlobalAveragePooling2D)

 dense_33 (Dense)            (None, 128)               4224

 dense_34 (Dense)            (None, 128)               16512

 dense_35 (Dense)            (None, 9)                 1161

=================================================================
Total params: 32041 (125.16 KB)
Trainable params: 32041 (125.16 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

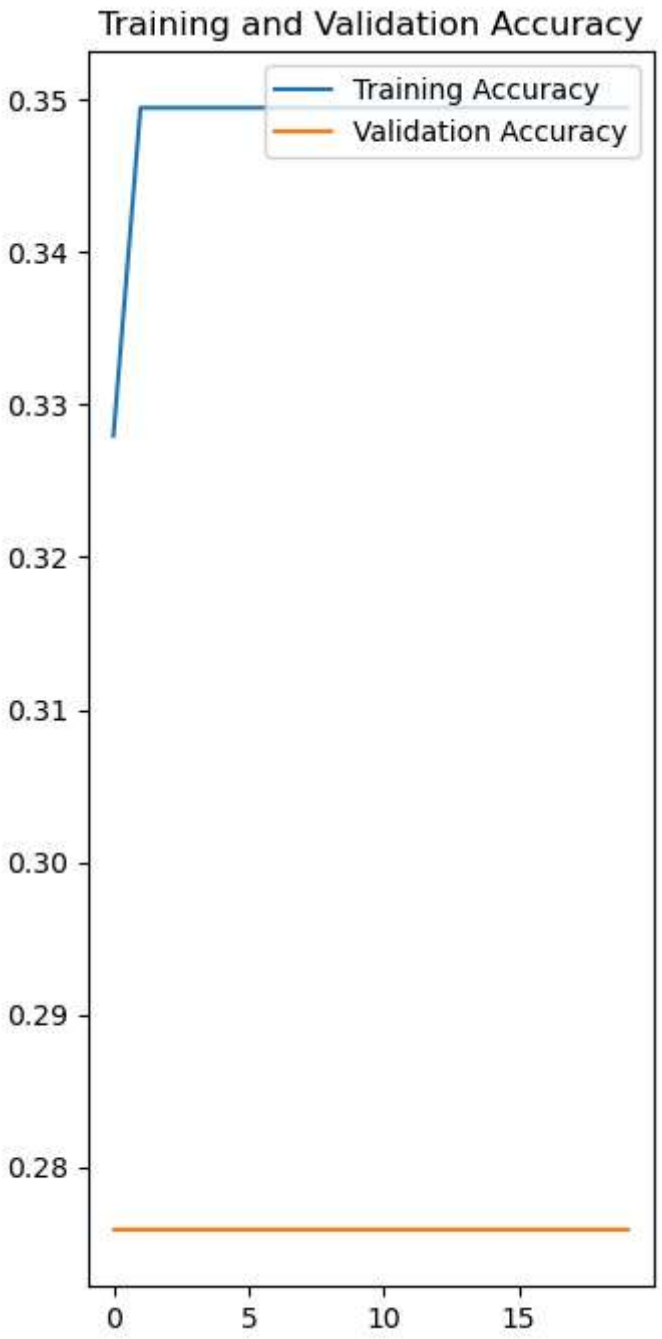
In [45]:
```
# Print the training and validation accuracies
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
print(f"Train Accuracy: {acc[-1]:.2f}")
print(f"Validation Accuracy: {val_acc[-1]:.2f}")
```

```
Train Accuracy: 0.41
Validation Accuracy: 0.39
```

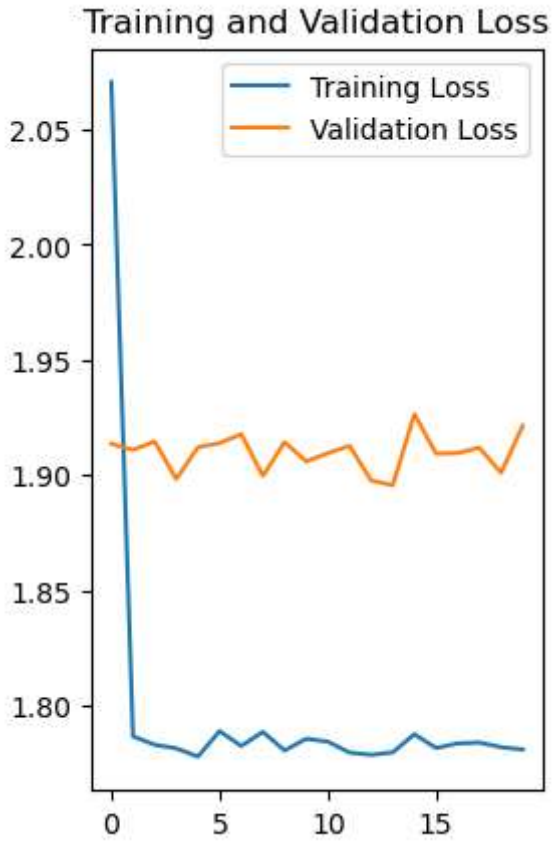
In [25]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='upper right')
plt.title('Training and Validation Accuracy')
```

Out[25]:
```
Text(0.5, 1.0, 'Training and Validation Accuracy')
```



In [46]:
```
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Use a Lightweight model (pre-trained) - MobileNetV2

In [47]:
```
# Set parameters for dataset
batch_size = 32
img_height = 256
img_width = 256

directory_path='C:\\Users\\neelk\\OneDrive\\Desktop\\Simplilearn-DataScience-Course\\Co

# Create the dataset
prm_train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    directory_path,
    validation_split=0.3,  # Split the dataset into training and validation
    subset="training",  # Use this part as the training data
    seed=123,  # Seed for reproducibility
    image_size=(256, 256),  # Resize images to 256x256
    batch_size=32,  # Number of images to retrieve in one batch,
    shuffle=True
)

# Create the dataset
prm_validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    directory_path,
    validation_split=0.3,  # Split the dataset into training and validation
    subset="validation",  # Use this part as the training data
    seed=123,  # Seed for reproducibility
    image_size=(256, 256),  # Resize images to 256x256
    batch_size=32,  # Number of images to retrieve in one batch
    shuffle=True
)
```

```
# Get the class names
prm_tr_class_names = prm_train_dataset.class_names
print("Training Class names:", prm_tr_class_names)

prm_val_class_names = prm_validation_dataset.class_names
print("Validation Class names:", prm_val_class_names)


# Visualize some images
import matplotlib.pyplot as plt

print("Training Dataset sample 10 images")
plt.figure(figsize=(10, 10))
for images, labels in prm_train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(tr_class_names[labels[i]])
        plt.axis("off")
plt.show()
```

```
Found 1162 files belonging to 9 classes.
Using 814 files for training.
Found 1162 files belonging to 9 classes.
Using 348 files for validation.
Training Class names: ['buoy', 'cruise_ship', 'ferry_boat', 'freight_boat', 'gondola',
'inflatable_boat', 'kayak', 'paper_boat', 'sailboat']
Validation Class names: ['buoy', 'cruise_ship', 'ferry_boat', 'freight_boat', 'gondol
a', 'inflatable_boat', 'kayak', 'paper_boat', 'sailboat']
Training Dataset sample 10 images
```

In [50]:
```
# Set parameters for dataset
batch_size = 32
img_height = 256
img_width = 256

# Preprocess datasets
AUTOTUNE = tf.data.AUTOTUNE

def preprocess(image, label):
    image = tf.keras.layers.Rescaling(1./255)(image)
    return image, label

prm_train_dataset = prm_train_dataset.map(preprocess, num_parallel_calls=AUTOTUNE)
prm_validation_dataset = prm_validation_dataset.map(preprocess, num_parallel_calls=AUTO

prm_train_dataset = prm_train_dataset.prefetch(buffer_size=AUTOTUNE)
prm_validation_dataset = prm_validation_dataset.prefetch(buffer_size=AUTOTUNE)

# Load MobileNetV2 model, exclude the top layers, and use pre-trained weights from 'ima
```

```
base_model = MobileNetV2(input_shape=(256, 256, 3), include_top=False, weights='imagene

# Freezing the base model
base_model.trainable = False

# Define and compile the model
# Build the model
model = Sequential([ base_model,
                     GlobalAveragePooling2D(),
                     Dropout(0.2),
                     Dense(256, activation='relu'),
                     BatchNormalization(),
                     Dropout(0.1),
                     Dense(128, activation='relu'),
                     BatchNormalization(),
                     Dropout(0.1),
                     Dense(9, activation='softmax')])


model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)

# Fit the model
epochs = 50
history = model.fit(
    prm_train_dataset,
    validation_data=prm_validation_dataset,
    epochs=epochs
)
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 1
28, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobi
lenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9406464/9406464 [==============================] - 1s 0us/step
Epoch 1/50
26/26 [==============================] - 17s 580ms/step - loss: 1.3202 - accuracy: 0.60
32 - val_loss: 0.8828 - val_accuracy: 0.7356
Epoch 2/50
26/26 [==============================] - 18s 678ms/step - loss: 0.3997 - accuracy: 0.87
59 - val_loss: 0.5882 - val_accuracy: 0.8190
Epoch 3/50
26/26 [==============================] - 19s 700ms/step - loss: 0.2757 - accuracy: 0.91
77 - val_loss: 0.5341 - val_accuracy: 0.8477
Epoch 4/50
26/26 [==============================] - 19s 702ms/step - loss: 0.1845 - accuracy: 0.94
84 - val_loss: 0.5317 - val_accuracy: 0.8247
Epoch 5/50
26/26 [==============================] - 19s 702ms/step - loss: 0.1523 - accuracy: 0.95
82 - val_loss: 0.5510 - val_accuracy: 0.8190
Epoch 6/50
26/26 [==============================] - 19s 696ms/step - loss: 0.1083 - accuracy: 0.97
67 - val_loss: 0.4775 - val_accuracy: 0.8506
Epoch 7/50
26/26 [==============================] - 19s 720ms/step - loss: 0.0684 - accuracy: 0.99
02 - val_loss: 0.4656 - val_accuracy: 0.8534
Epoch 8/50
26/26 [==============================] - 19s 713ms/step - loss: 0.0712 - accuracy: 0.98
53 - val_loss: 0.4468 - val_accuracy: 0.8621
Epoch 9/50
26/26 [==============================] - 19s 710ms/step - loss: 0.0550 - accuracy: 0.99
02 - val_loss: 0.4697 - val_accuracy: 0.8563
Epoch 10/50
26/26 [==============================] - 20s 736ms/step - loss: 0.0427 - accuracy: 0.99
39 - val_loss: 0.4601 - val_accuracy: 0.8678
Epoch 11/50
26/26 [==============================] - 20s 736ms/step - loss: 0.0474 - accuracy: 0.98
77 - val_loss: 0.4563 - val_accuracy: 0.8678
Epoch 12/50
26/26 [==============================] - 19s 728ms/step - loss: 0.0340 - accuracy: 0.99
26 - val_loss: 0.5181 - val_accuracy: 0.8563
Epoch 13/50
26/26 [==============================] - 21s 783ms/step - loss: 0.0356 - accuracy: 0.99
26 - val_loss: 0.5259 - val_accuracy: 0.8649
Epoch 14/50
26/26 [==============================] - 23s 849ms/step - loss: 0.0364 - accuracy: 0.99
51 - val_loss: 0.5066 - val_accuracy: 0.8563
Epoch 15/50
26/26 [==============================] - 22s 833ms/step - loss: 0.0326 - accuracy: 0.99
14 - val_loss: 0.5605 - val_accuracy: 0.8621
Epoch 16/50
26/26 [==============================] - 24s 913ms/step - loss: 0.0259 - accuracy: 0.99
63 - val_loss: 0.5293 - val_accuracy: 0.8534
Epoch 17/50
26/26 [==============================] - 22s 809ms/step - loss: 0.0249 - accuracy: 0.99
26 - val_loss: 0.5393 - val_accuracy: 0.8707
Epoch 18/50
26/26 [==============================] - 22s 813ms/step - loss: 0.0232 - accuracy: 0.99
51 - val_loss: 0.5899 - val_accuracy: 0.8621
Epoch 19/50
```

```
26/26 [==============================] - 22s 807ms/step - loss: 0.0277 - accuracy: 0.99
39 - val_loss: 0.5903 - val_accuracy: 0.8506
Epoch 20/50
26/26 [==============================] - 22s 809ms/step - loss: 0.0251 - accuracy: 0.99
39 - val_loss: 0.6099 - val_accuracy: 0.8563
Epoch 21/50
26/26 [==============================] - 22s 804ms/step - loss: 0.0150 - accuracy: 0.99
88 - val_loss: 0.5369 - val_accuracy: 0.8534
Epoch 22/50
26/26 [==============================] - 22s 808ms/step - loss: 0.0215 - accuracy: 0.99
51 - val_loss: 0.5099 - val_accuracy: 0.8764
Epoch 23/50
26/26 [==============================] - 22s 812ms/step - loss: 0.0222 - accuracy: 0.99
39 - val_loss: 0.5549 - val_accuracy: 0.8621
Epoch 24/50
26/26 [==============================] - 25s 952ms/step - loss: 0.0340 - accuracy: 0.99
14 - val_loss: 0.5897 - val_accuracy: 0.8678
Epoch 25/50
26/26 [==============================] - 168s 7s/step - loss: 0.0504 - accuracy: 0.9865
- val_loss: 0.6505 - val_accuracy: 0.8506
Epoch 26/50
26/26 [==============================] - 23s 857ms/step - loss: 0.0243 - accuracy: 0.99
02 - val_loss: 0.6265 - val_accuracy: 0.8477
Epoch 27/50
26/26 [==============================] - 22s 816ms/step - loss: 0.0382 - accuracy: 0.98
89 - val_loss: 0.6069 - val_accuracy: 0.8420
Epoch 28/50
26/26 [==============================] - 22s 817ms/step - loss: 0.0244 - accuracy: 0.99
39 - val_loss: 0.6041 - val_accuracy: 0.8707
Epoch 29/50
26/26 [==============================] - 22s 819ms/step - loss: 0.0258 - accuracy: 0.99
51 - val_loss: 0.6462 - val_accuracy: 0.8678
Epoch 30/50
26/26 [==============================] - 22s 815ms/step - loss: 0.0199 - accuracy: 0.99
26 - val_loss: 0.5582 - val_accuracy: 0.8707
Epoch 31/50
26/26 [==============================] - 23s 843ms/step - loss: 0.0267 - accuracy: 0.99
39 - val_loss: 0.6645 - val_accuracy: 0.8592
Epoch 32/50
26/26 [==============================] - 22s 841ms/step - loss: 0.0288 - accuracy: 0.99
39 - val_loss: 0.8359 - val_accuracy: 0.8333
Epoch 33/50
26/26 [==============================] - 23s 851ms/step - loss: 0.0190 - accuracy: 0.99
39 - val_loss: 0.6472 - val_accuracy: 0.8506
Epoch 34/50
26/26 [==============================] - 22s 842ms/step - loss: 0.0307 - accuracy: 0.99
26 - val_loss: 0.5944 - val_accuracy: 0.8477
Epoch 35/50
26/26 [==============================] - 22s 834ms/step - loss: 0.0187 - accuracy: 0.99
39 - val_loss: 0.6480 - val_accuracy: 0.8420
Epoch 36/50
26/26 [==============================] - 22s 837ms/step - loss: 0.0173 - accuracy: 0.99
51 - val_loss: 0.6463 - val_accuracy: 0.8477
Epoch 37/50
26/26 [==============================] - 22s 819ms/step - loss: 0.0377 - accuracy: 0.98
40 - val_loss: 0.6280 - val_accuracy: 0.8391
Epoch 38/50
26/26 [==============================] - 22s 822ms/step - loss: 0.0282 - accuracy: 0.99
14 - val_loss: 0.7116 - val_accuracy: 0.8477
Epoch 39/50
```

```
26/26 [==============================] - 23s 850ms/step - loss: 0.0219 - accuracy: 0.99
51 - val_loss: 0.6634 - val_accuracy: 0.8563
Epoch 40/50
26/26 [==============================] - 24s 912ms/step - loss: 0.0165 - accuracy: 0.99
63 - val_loss: 0.6039 - val_accuracy: 0.8563
Epoch 41/50
26/26 [==============================] - 24s 905ms/step - loss: 0.0140 - accuracy: 0.99
39 - val_loss: 0.6269 - val_accuracy: 0.8534
Epoch 42/50
26/26 [==============================] - 23s 854ms/step - loss: 0.0165 - accuracy: 0.99
26 - val_loss: 0.9638 - val_accuracy: 0.8276
Epoch 43/50
26/26 [==============================] - 23s 867ms/step - loss: 0.0232 - accuracy: 0.99
26 - val_loss: 0.7064 - val_accuracy: 0.8506
Epoch 44/50
26/26 [==============================] - 24s 887ms/step - loss: 0.0183 - accuracy: 0.99
39 - val_loss: 0.7363 - val_accuracy: 0.8534
Epoch 45/50
26/26 [==============================] - 24s 892ms/step - loss: 0.0089 - accuracy: 1.00
00 - val_loss: 0.7130 - val_accuracy: 0.8592
Epoch 46/50
26/26 [==============================] - 23s 869ms/step - loss: 0.0183 - accuracy: 0.99
39 - val_loss: 0.8437 - val_accuracy: 0.8448
Epoch 47/50
26/26 [==============================] - 22s 822ms/step - loss: 0.0282 - accuracy: 0.98
89 - val_loss: 0.6555 - val_accuracy: 0.8621
Epoch 48/50
26/26 [==============================] - 22s 829ms/step - loss: 0.0128 - accuracy: 0.99
88 - val_loss: 0.6086 - val_accuracy: 0.8736
Epoch 49/50
26/26 [==============================] - 22s 821ms/step - loss: 0.0182 - accuracy: 0.99
75 - val_loss: 0.6408 - val_accuracy: 0.8649
Epoch 50/50
26/26 [==============================] - 22s 839ms/step - loss: 0.0118 - accuracy: 0.99
75 - val_loss: 0.6846 - val_accuracy: 0.8707
```

In [51]:
```
model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Func  (None, 8, 8, 1280)        2257984
 tional)

 global_average_pooling2d_1  (None, 1280)              0
 4 (GlobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense_36 (Dense)            (None, 256)               327936

 batch_normalization (Batch  (None, 256)               1024
 Normalization)

 dropout_1 (Dropout)         (None, 256)               0

 dense_37 (Dense)            (None, 128)               32896

 batch_normalization_1 (Bat  (None, 128)               512
 chNormalization)

 dropout_2 (Dropout)         (None, 128)               0

 dense_38 (Dense)            (None, 9)                 1161

=================================================================
Total params: 2621513 (10.00 MB)
Trainable params: 362761 (1.38 MB)
Non-trainable params: 2258752 (8.62 MB)
_________________________________________________________________
```

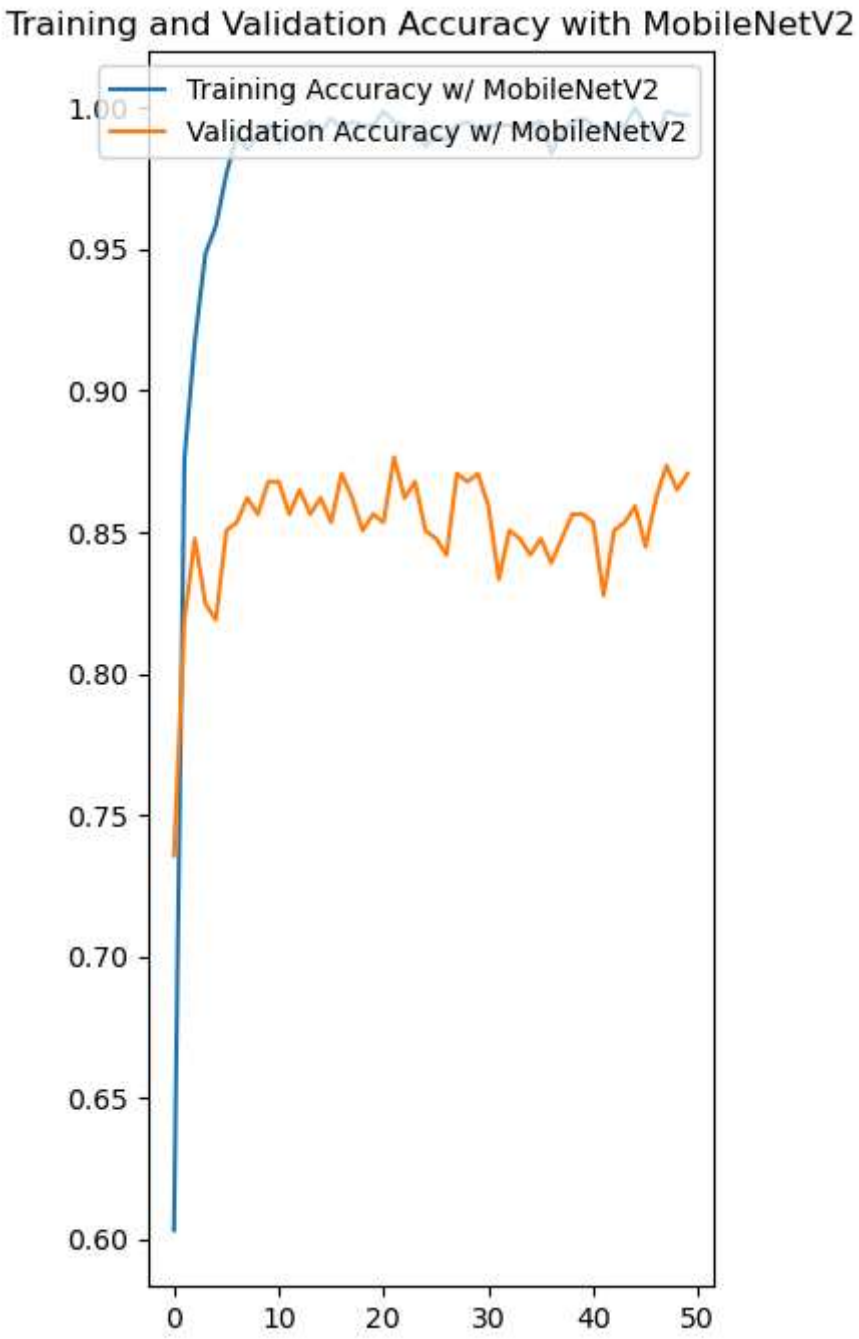
In [53]:
```
# Print the training and validation accuracies For MobileNetV2 model
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
print(f"Train Accuracy with MobileNetV2: {acc[-1]:.2f}")
print(f"Validation Accuracy with MobileNetV2: {val_acc[-1]:.2f}")
```

```
Train Accuracy with MobileNetV2: 1.00
Validation Accuracy with MobileNetV2: 0.87
```

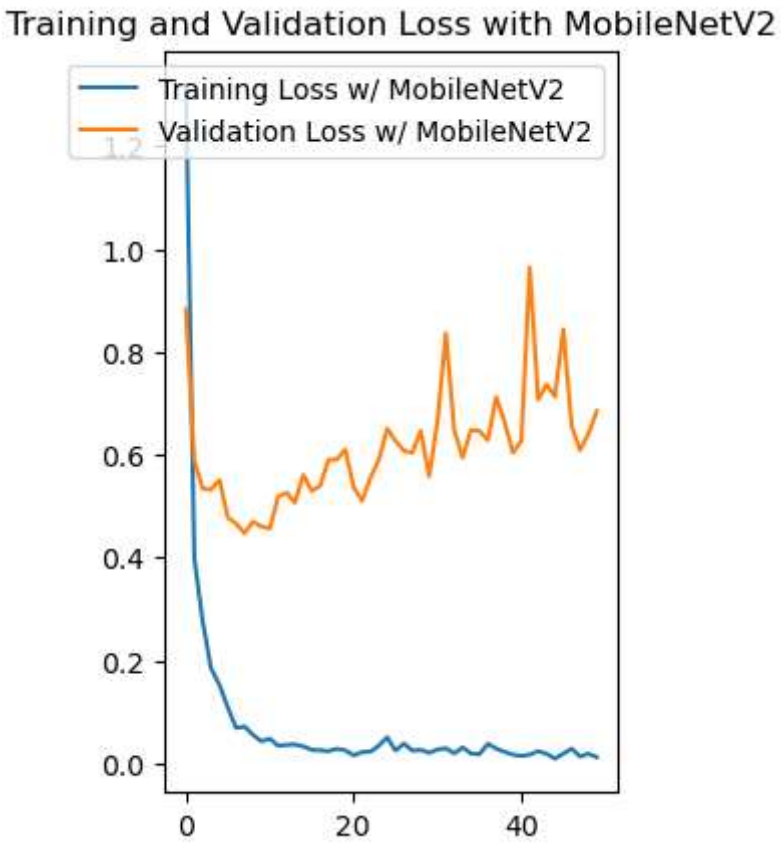
In [54]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy w/ MobileNetV2')
plt.plot(epochs_range, val_acc, label='Validation Accuracy w/ MobileNetV2')
plt.legend(loc='upper right')
plt.title('Training and Validation Accuracy with MobileNetV2')
```

Out[54]:
```
Text(0.5, 1.0, 'Training and Validation Accuracy with MobileNetV2')
```



In [55]:
```
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss w/ MobileNetV2')
plt.plot(epochs_range, val_loss, label='Validation Loss w/ MobileNetV2')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss with MobileNetV2')
plt.show()
```



Observation:

The output indicates the following:

- The evaluation was performed on both custom CNN model and a Pre-trained model (MobileNetV2)
- Custom CNN Model
  - Results: **Train Accuracy was 0.41 and Validation Accuracy was 0.39**
- Pre-Trained Model-MobileNetV2
  - Results: **Train Accuracy was 1.00 and Validation Accuracy was 0.87**

Therefore, it can be seen that a **Pre-trained Lightweight Model like MobileNetV2 performs much better than a custom CNN model.**

In [ ]:

In [ ]: