In [ ]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from tensorflow.keras import layers

import os
import shutil

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```python
DATASET_PATH = 'Automating_Port_Operations_dataset'

# Courtesy of SimpliLearn for providing os commands regarding train_test_split on d
def copy_files(img_paths, img_classes, target):
    for img_path, img_class  in zip(img_paths, img_classes):
        dest_dir = os.path.join(target, img_class)
        os.makedirs(dest_dir, exist_ok=True)
        shutil.copy(img_path, dest_dir)

def train_test_dir(test_size, rand, data_dir=DATASET_PATH, train_dir=DATASET_PATH+"
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)

    path_list = []
    classes = []

    for folder in os.listdir(data_dir) :
        folder_path = os.path.join(data_dir, folder)
        if os.path.isdir(folder_path) :
            for file in os.listdir(folder_path) :
                if file.endswith('.jpg') :
                    path_list.append(os.path.join(folder_path, file))
                    classes.append(folder)

    train_paths, test_paths, train_classes, test_classes = train_test_split(path_li

    copy_files(train_paths, train_classes, train_dir)
    copy_files(test_paths, test_classes, test_dir)
```

CNN

In [ ]:

```python
# Batch size, image size
BAT_SIZE, IMG_SIZE = 32, 180

train_test_dir(0.2, 43)

train = tf.keras.preprocessing.image_dataset_from_directory(DATASET_PATH + "/train"
```

```
                                                            label_mode = 'categoric
                                                            validation_split = 0.2,
                                                            shuffle = True,
                                                            seed = 43,
                                                            subset = 'training',
                                                            image_size = (IMG_SIZE,
                                                            batch_size = BAT_SIZE)
val = tf.keras.preprocessing.image_dataset_from_directory(DATASET_PATH + "/train",
                                                            label_mode = 'categoric
                                                            validation_split = 0.2,
                                                            shuffle = True,
                                                            seed = 43,
                                                            subset = 'validation',
                                                            image_size = (IMG_SIZE,
                                                            batch_size = BAT_SIZE)
test = train = tf.keras.preprocessing.image_dataset_from_directory(DATASET_PATH + "
                                                            label_mode = 'categoric
                                                            shuffle = True,
                                                            seed = 43,
                                                            image_size = (IMG_SIZE,
                                                            batch_size = BAT_SIZE)


class_names = train.class_names

AUTOTUNE = tf.data.AUTOTUNE

train = train.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val = val.cache().prefetch(buffer_size=AUTOTUNE)
```

```
Found 1083 files belonging to 9 classes.
Using 867 files for training.
Using 867 files for training.
Found 1083 files belonging to 9 classes.
Using 216 files for validation.
Found 503 files belonging to 9 classes.
```

In [ ]:

```
model = tf.keras.models.Sequential([
                                    layers.Rescaling(1./255, input_shape = (IMG_SIZ
                                    layers.Conv2D(32, 3, padding='same', activation
                                    layers.MaxPooling2D(),
                                    layers.Conv2D(32, 3, padding='same', activation
                                    layers.MaxPooling2D(),
                                    layers.GlobalAveragePooling2D(),
                                    layers.Flatten(),
                                    layers.Dense(128, activation='relu'),
                                    layers.Dense(128, activation='relu'),
                                    layers.Dense(9, activation='softmax')])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy', 'precision', 'recall'])

# Uncomment for model layout
# print(model.summary())

hist = model.fit(train,
```

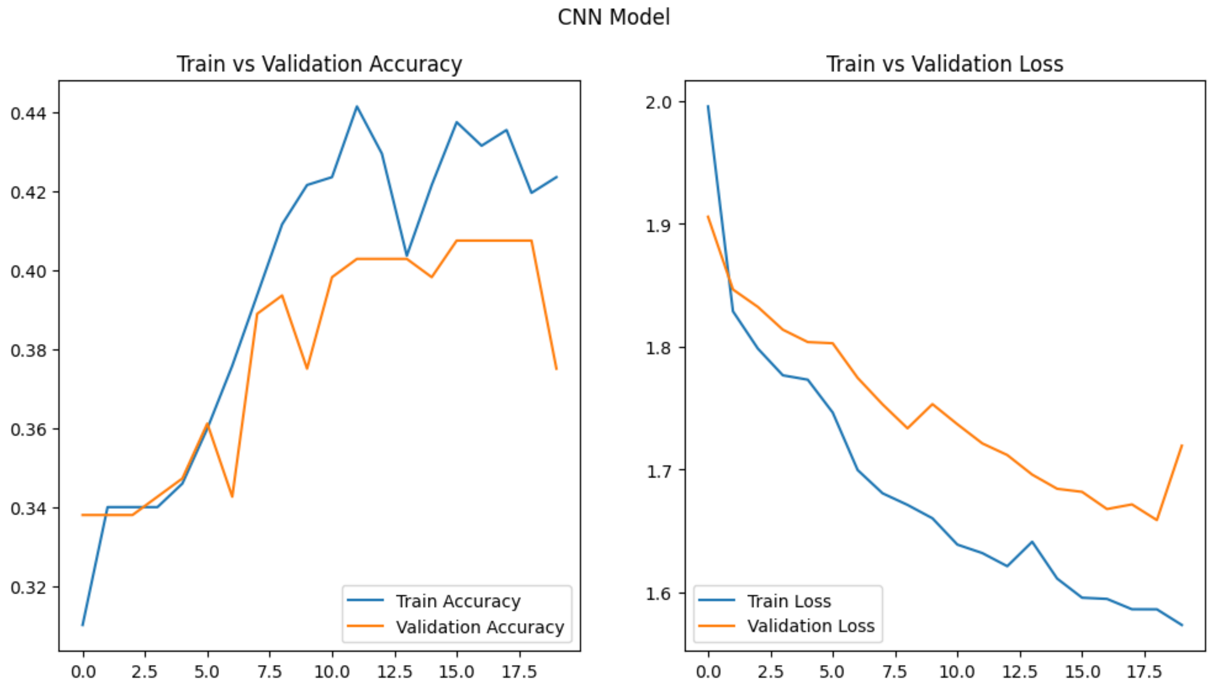
```
                 validation_data=val,
                 epochs = 20)

acc = hist.history['accuracy']
v_acc = hist.history['val_accuracy']
loss = hist.history['loss']
v_loss = hist.history['val_loss']
```

```
Epoch 1/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 5s 176ms/step - accuracy: 0.2523 - loss: 2.1046 - precisi
on: 0.1765 - recall: 0.0018 - val_accuracy: 0.3380 - val_loss: 1.9057 - val_precisio
n: 0.4478 - val_recall: 0.1389
Epoch 2/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 135ms/step - accuracy: 0.3425 - loss: 1.8286 - precisi
on: 0.4745 - recall: 0.0947 - val_accuracy: 0.3380 - val_loss: 1.8466 - val_precisio
n: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 3/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 124ms/step - accuracy: 0.3589 - loss: 1.7776 - precisi
on: 0.4941 - recall: 0.0078 - val_accuracy: 0.3380 - val_loss: 1.8323 - val_precisio
n: 0.4000 - val_recall: 0.0093
Epoch 4/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 121ms/step - accuracy: 0.3101 - loss: 1.8079 - precisi
on: 0.0882 - recall: 7.0167e-04 - val_accuracy: 0.3426 - val_loss: 1.8138 - val_prec
ision: 0.6000 - val_recall: 0.0417
Epoch 5/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 124ms/step - accuracy: 0.3774 - loss: 1.7365 - precisi
on: 0.7921 - recall: 0.0597 - val_accuracy: 0.3472 - val_loss: 1.8038 - val_precisio
n: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 6/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 124ms/step - accuracy: 0.3778 - loss: 1.7417 - precisi
on: 0.0980 - recall: 3.5878e-04 - val_accuracy: 0.3611 - val_loss: 1.8028 - val_prec
ision: 0.2222 - val_recall: 0.0093
Epoch 7/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.3820 - loss: 1.6404 - precisi
on: 0.5324 - recall: 0.0315 - val_accuracy: 0.3426 - val_loss: 1.7746 - val_precisio
n: 0.5000 - val_recall: 0.0231
Epoch 8/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 117ms/step - accuracy: 0.4286 - loss: 1.6337 - precisi
on: 0.6077 - recall: 0.0542 - val_accuracy: 0.3889 - val_loss: 1.7531 - val_precisio
n: 0.5556 - val_recall: 0.0694
Epoch 9/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 112ms/step - accuracy: 0.4278 - loss: 1.6613 - precisi
on: 0.5290 - recall: 0.0746 - val_accuracy: 0.3935 - val_loss: 1.7336 - val_precisio
n: 0.5000 - val_recall: 0.0231
Epoch 10/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 115ms/step - accuracy: 0.4136 - loss: 1.6515 - precisi
on: 0.5236 - recall: 0.0432 - val_accuracy: 0.3750 - val_loss: 1.7533 - val_precisio
n: 0.5577 - val_recall: 0.1343
Epoch 11/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 114ms/step - accuracy: 0.4126 - loss: 1.6833 - precisi
on: 0.6054 - recall: 0.1091 - val_accuracy: 0.3981 - val_loss: 1.7369 - val_precisio
n: 0.5714 - val_recall: 0.1111
Epoch 12/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 114ms/step - accuracy: 0.4137 - loss: 1.7069 - precisi
on: 0.5357 - recall: 0.0770 - val_accuracy: 0.4028 - val_loss: 1.7213 - val_precisio
n: 0.5636 - val_recall: 0.1435
Epoch 13/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 113ms/step - accuracy: 0.4460 - loss: 1.6598 - precisi
on: 0.5668 - recall: 0.1502 - val_accuracy: 0.4028 - val_loss: 1.7118 - val_precisio
n: 0.6538 - val_recall: 0.0787
Epoch 14/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 112ms/step - accuracy: 0.3859 - loss: 1.6415 - precisi
on: 0.5880 - recall: 0.1146 - val_accuracy: 0.4028 - val_loss: 1.6959 - val_precisio
n: 0.6471 - val_recall: 0.1019
```

```
Epoch 15/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 116ms/step - accuracy: 0.4365 - loss: 1.5789 - precisi
on: 0.5766 - recall: 0.1098 - val_accuracy: 0.3981 - val_loss: 1.6844 - val_precisio
n: 0.6111 - val_recall: 0.1019
Epoch 16/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 129ms/step - accuracy: 0.4395 - loss: 1.6296 - precisi
on: 0.5523 - recall: 0.0733 - val_accuracy: 0.4074 - val_loss: 1.6819 - val_precisio
n: 0.6279 - val_recall: 0.1250
Epoch 17/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 129ms/step - accuracy: 0.4368 - loss: 1.5475 - precisi
on: 0.6522 - recall: 0.1615 - val_accuracy: 0.4074 - val_loss: 1.6679 - val_precisio
n: 0.6765 - val_recall: 0.1065
Epoch 18/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.4220 - loss: 1.6160 - precisi
on: 0.7559 - recall: 0.1277 - val_accuracy: 0.4074 - val_loss: 1.6717 - val_precisio
n: 0.6094 - val_recall: 0.1806
Epoch 19/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.4012 - loss: 1.5983 - precisi
on: 0.5835 - recall: 0.1418 - val_accuracy: 0.4074 - val_loss: 1.6589 - val_precisio
n: 0.6897 - val_recall: 0.0926
Epoch 20/20
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step - accuracy: 0.4210 - loss: 1.5599 - precisi
on: 0.6939 - recall: 0.1686 - val_accuracy: 0.3750 - val_loss: 1.7194 - val_precisio
n: 0.5893 - val_recall: 0.1528
```

In [ ]:

```
plt.figure(figsize=(12,6))

plt.subplot(1, 2, 1)
plt.plot(range(20), acc, label='Train Accuracy')
plt.plot(range(20), v_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Train vs Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(20), loss, label='Train Loss')
plt.plot(range(20), v_loss, label='Validation Loss')
plt.legend(loc='lower left')
plt.title('Train vs Validation Loss')

plt.suptitle('CNN Model')
plt.show()
```

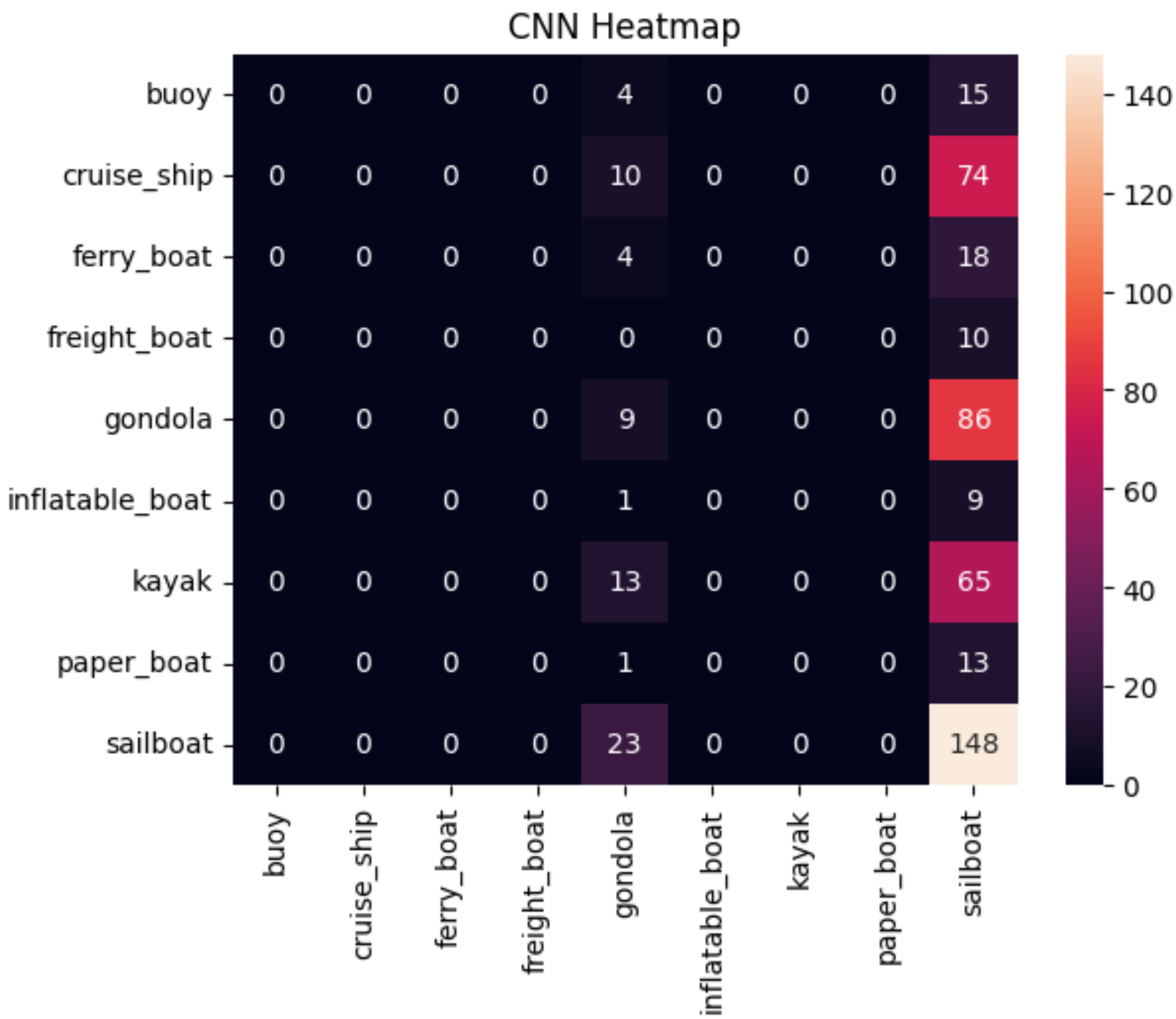


In [ ]:
```
predicts = model.predict(test)
predict = np.argmax(predicts, axis=1)
true = np.argmax(tf.concat([y for x, y in test], axis=0), axis=1)

# print(f"\nAccuracy score: {accuracy_score(true, predict)}")
# print(f"Mean Squared Error: {mean_squared_error(true, predict)}\n")

model.evaluate(test)
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - accuracy: 0.4173 - loss: 1.5982 - precisio
n: 0.5923 - recall: 0.1523
```

Out[ ]:
```
[1.6276891231536865,
 0.3956262469291687,
 0.5813953280448914,
 0.14910537004470825]
```

In [ ]:
```
import seaborn as sns

sns.heatmap(confusion_matrix(true, predict), annot=True, fmt='d',
            xticklabels=class_names, yticklabels=class_names)
plt.title('CNN Heatmap')
plt.show()

print(classification_report(true, predict))
```




```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        19
           1       0.00      0.00      0.00        84
           2       0.00      0.00      0.00        22
           3       0.00      0.00      0.00        10
           4       0.14      0.09      0.11        95
           5       0.00      0.00      0.00        10
           6       0.00      0.00      0.00        78
           7       0.00      0.00      0.00        14
           8       0.34      0.87      0.49       171

    accuracy                           0.31       503
   macro avg       0.05      0.11      0.07       503
weighted avg       0.14      0.31      0.19       503
```

MobileNet

In [ ]:

```
train_test_dir(0.3, 1)
train = tf.keras.preprocessing.image_dataset_from_directory(DATASET_PATH + "/train"
                                                            label_mode = 'categoric
                                                            validation_split = 0.3,
```

```
                                                                    shuffle = True,
                                                                    seed = 1,
                                                                    subset = 'training',
                                                                    image_size = (IMG_SIZE,
                                                                    batch_size = BAT_SIZE)
val = tf.keras.preprocessing.image_dataset_from_directory(DATASET_PATH + "/train",
                                                                    label_mode = 'categoric
                                                                    validation_split = 0.3,
                                                                    shuffle = True,
                                                                    seed = 1,
                                                                    subset = 'validation',
                                                                    image_size = (IMG_SIZE,
                                                                    batch_size = BAT_SIZE)
test = train = tf.keras.preprocessing.image_dataset_from_directory(DATASET_PATH + "
                                                                    label_mode = 'categoric
                                                                    shuffle = True,
                                                                    seed = 1,
                                                                    image_size = (IMG_SIZE,
                                                                    batch_size = BAT_SIZE)


AUTOTUNE = tf.data.AUTOTUNE

train = train.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val = val.cache().prefetch(buffer_size=AUTOTUNE)
```

```
Found 1083 files belonging to 9 classes.
Using 759 files for training.
Using 759 files for training.
Found 1083 files belonging to 9 classes.
Using 324 files for validation.
Found 503 files belonging to 9 classes.
```

In [ ]:

```
mn_model = tf.keras.applications.MobileNetV2(input_shape = (IMG_SIZE, IMG_SIZE, 3),
                                             include_top=False)
mn_model.trainable = False

# Uncomment for MobileNet model summary
# print(mn_model.summary())

# inputs
i = tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

# hidden
x = layers.Rescaling(1./255)(i)
x = mn_model(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.2)(x)
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.1)(x)
x = layers.Dense(128, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.1)(x)

#outputs
```

```python
o = layers.Dense(9, activation='softmax')(x)

model = tf.keras.Model(i, o)

# Uncomment for final model summary
# print(model.summary())

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy', 'precision', 'recall'])

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)

hist = model.fit(train,
                 validation_data = val,
                 epochs = 50,
                 callbacks=[callback])

acc = hist.history['accuracy']
v_acc = hist.history['val_accuracy']
loss = hist.history['loss']
v_loss = hist.history['val_loss']
```

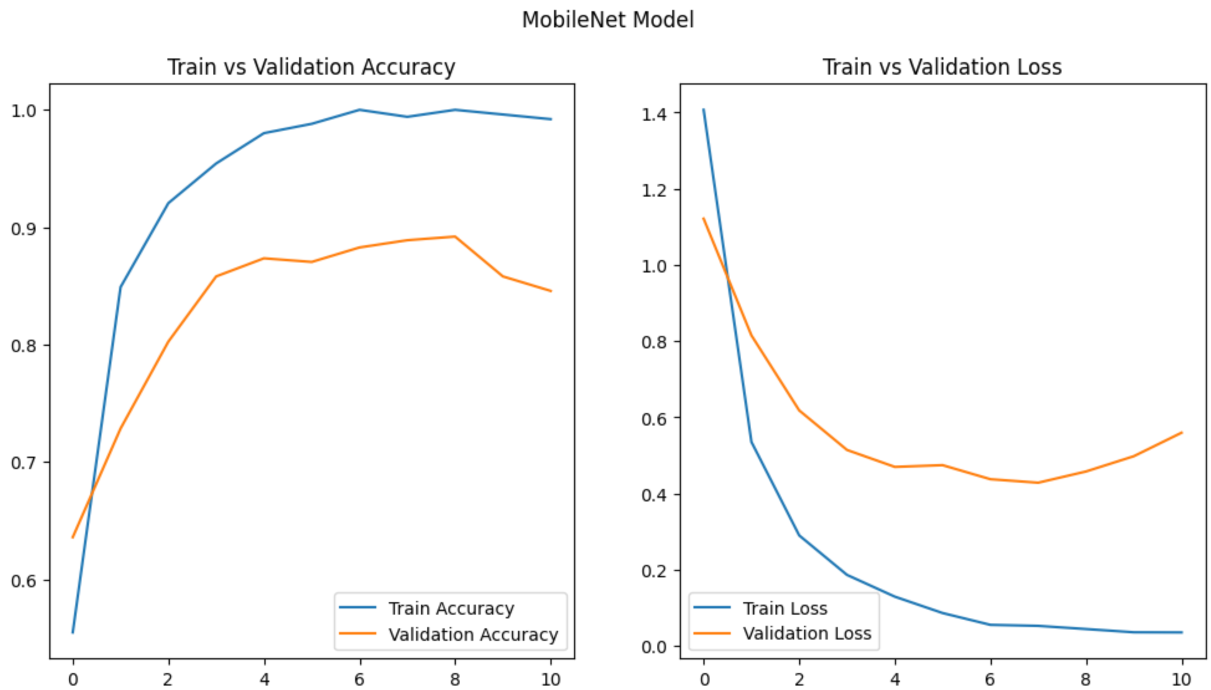
```
Epoch 1/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 11s 359ms/step - accuracy: 0.4118 - loss: 1.7864 - precis
ion: 0.5920 - recall: 0.2750 - val_accuracy: 0.6358 - val_loss: 1.1212 - val_precisi
on: 0.8962 - val_recall: 0.5062
Epoch 2/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 274ms/step - accuracy: 0.8185 - loss: 0.6302 - precisi
on: 0.9207 - recall: 0.7229 - val_accuracy: 0.7284 - val_loss: 0.8148 - val_precisio
n: 0.8734 - val_recall: 0.6389
Epoch 3/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 261ms/step - accuracy: 0.9018 - loss: 0.3420 - precisi
on: 0.9547 - recall: 0.8400 - val_accuracy: 0.8025 - val_loss: 0.6178 - val_precisio
n: 0.8931 - val_recall: 0.7222
Epoch 4/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 262ms/step - accuracy: 0.9527 - loss: 0.1844 - precisi
on: 0.9732 - recall: 0.9256 - val_accuracy: 0.8580 - val_loss: 0.5140 - val_precisio
n: 0.9184 - val_recall: 0.7994
Epoch 5/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 257ms/step - accuracy: 0.9781 - loss: 0.1265 - precisi
on: 0.9836 - recall: 0.9462 - val_accuracy: 0.8735 - val_loss: 0.4695 - val_precisio
n: 0.9215 - val_recall: 0.8333
Epoch 6/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 260ms/step - accuracy: 0.9907 - loss: 0.0873 - precisi
on: 0.9988 - recall: 0.9840 - val_accuracy: 0.8704 - val_loss: 0.4740 - val_precisio
n: 0.9130 - val_recall: 0.8426
Epoch 7/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 262ms/step - accuracy: 1.0000 - loss: 0.0592 - precisi
on: 1.0000 - recall: 0.9949 - val_accuracy: 0.8827 - val_loss: 0.4372 - val_precisio
n: 0.9200 - val_recall: 0.8519
Epoch 8/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 258ms/step - accuracy: 0.9891 - loss: 0.0556 - precisi
on: 0.9955 - recall: 0.9863 - val_accuracy: 0.8889 - val_loss: 0.4280 - val_precisio
n: 0.9123 - val_recall: 0.8673
Epoch 9/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 259ms/step - accuracy: 1.0000 - loss: 0.0423 - precisi
on: 1.0000 - recall: 0.9984 - val_accuracy: 0.8920 - val_loss: 0.4574 - val_precisio
n: 0.9150 - val_recall: 0.8642
Epoch 10/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 256ms/step - accuracy: 0.9977 - loss: 0.0326 - precisi
on: 0.9977 - recall: 0.9977 - val_accuracy: 0.8580 - val_loss: 0.4975 - val_precisio
n: 0.8980 - val_recall: 0.8426
Epoch 11/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 4s 264ms/step - accuracy: 0.9872 - loss: 0.0395 - precisi
on: 0.9919 - recall: 0.9852 - val_accuracy: 0.8457 - val_loss: 0.5593 - val_precisio
n: 0.8766 - val_recall: 0.8333
```

In [ ]:

```
model.evaluate(test)

num_iters = len(hist.history['loss'])

plt.figure(figsize=(12,6))

plt.subplot(1, 2, 1)
plt.plot(range(num_iters), acc, label='Train Accuracy')
plt.plot(range(num_iters), v_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Train vs Validation Accuracy')
```

```
plt.subplot(1, 2, 2)
plt.plot(range(num_iters), loss, label='Train Loss')
plt.plot(range(num_iters), v_loss, label='Validation Loss')
plt.legend(loc='lower left')
plt.title('Train vs Validation Loss')

plt.suptitle('MobileNet Model')
plt.show()
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 3s 170ms/step - accuracy: 0.9989 - loss: 0.0238 - precisi
on: 0.9989 - recall: 0.9964
```



Closing Statement

MobileNet's model can achieve more than twice the accuracy & a much less loss score than base CNN's model in less epochs. This is due to:

1. MobileNet having a more elaborate model - not only in having extra layers for dropout, but in the transfer model itself.

2. MobileNet is already a prefixed model that has been transferred over, meaning it is capable of being more accurate and precise than the crudely assembled CNN model.

However, base CNN's model has less of a divergence in the train-validation loss score compared to MobileNet. For example, there is a 0.1 difference in loss in the base CNN, compared to MobileNet's 0.6 difference. This is perhaps a drawback to MobileNet's more scrutinized model, with the extra layers causing a higher variation in score.