# Project for Course 5: Automating Port Operations

## By John Hamilton

Note: I had to print this Jupyter notebook in Word in order to upload it, so there are no boxes to indicate code blocks, but hopefully you can tell what is what by the differences in font size and style.

## Step 1: Build a CNN network to classify the boat

### Import the required libraries

import matplotlib.pyplot as plt  
import numpy as np  
import tensorflow as tf  
  
from tensorflow import keras  
from tensorflow.keras import layers  
from tensorflow.keras.models import Sequential

### Load classified images from disk and split the Train and Validation datasets

batch\_size = 32  
img\_height = 256  
img\_width = 256  
  
train\_ds, val\_ds = tf.keras.utils.image\_dataset\_from\_directory(  
 'Automating\_Port\_Operations\_dataset',  
 validation\_split=0.2,  
 subset="both",  
 shuffle=True,  
 seed=43,  
 image\_size=(img\_height, img\_width),  
 batch\_size=batch\_size  
 )  
  
class\_names = train\_ds.class\_names

Found 1162 files belonging to 9 classes.  
Using 930 files for training.  
Using 232 files for validation.

### Adjust datasets for use in the CNN:

1. Apply normalization to images (x)
2. Change labels to categorical (y)

normalization\_layer = layers.Rescaling(1./255)  
  
train\_ds = train\_ds.map(lambda x, y: (normalization\_layer(x), keras.utils.to\_categorical(y, num\_classes=9)))  
val\_ds = val\_ds.map(lambda x, y: (normalization\_layer(x), keras.utils.to\_categorical(y, num\_classes=9)))

### Create the Convolutional Neural Network by specifying its layers

num\_classes = 9  
model = Sequential([  
 layers.Conv2D(32, kernel\_size=(3,3), padding='same', activation='relu', input\_shape=(img\_height, img\_width, 3)),  
 layers.MaxPooling2D(),  
 layers.Conv2D(32, kernel\_size=(3,3), padding='same', activation='relu'),  
 layers.MaxPooling2D(),  
 layers.GlobalAveragePooling2D(),  
 layers.Dense(128, activation='relu'),  
 layers.Dense(128, activation='relu'),  
 layers.Dense(num\_classes, activation='softmax')  
])

c:\Users\jbham\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base\_conv.py:99: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.  
 super().\_\_init\_\_(

### Compile the CNN

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

### Train the model on the training dataset and validate using the validation dataset

epochs=20  
history = model.fit(  
 train\_ds,  
 validation\_data=val\_ds,  
 epochs=epochs  
)

Epoch 1/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 103ms/step - accuracy: 0.2100 - loss: 2.0828 - precision: 0.2449 - recall: 0.0122 - val\_accuracy: 0.3017 - val\_loss: 1.8207 - val\_precision: 0.0000e+00 - val\_recall: 0.0000e+00  
Epoch 2/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 100ms/step - accuracy: 0.3543 - loss: 1.8005 - precision: 0.3847 - recall: 0.0144 - val\_accuracy: 0.3017 - val\_loss: 1.8178 - val\_precision: 0.0000e+00 - val\_recall: 0.0000e+00  
Epoch 3/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 101ms/step - accuracy: 0.3399 - loss: 1.7827 - precision: 0.4343 - recall: 0.0095 - val\_accuracy: 0.3017 - val\_loss: 1.7951 - val\_precision: 0.0000e+00 - val\_recall: 0.0000e+00  
Epoch 4/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 101ms/step - accuracy: 0.3500 - loss: 1.7524 - precision: 0.3633 - recall: 0.0066 - val\_accuracy: 0.2974 - val\_loss: 1.7748 - val\_precision: 1.0000 - val\_recall: 0.0043  
Epoch 5/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 100ms/step - accuracy: 0.3554 - loss: 1.7522 - precision: 0.5328 - recall: 0.0396 - val\_accuracy: 0.3190 - val\_loss: 1.7587 - val\_precision: 0.8462 - val\_recall: 0.0474  
Epoch 6/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 100ms/step - accuracy: 0.3858 - loss: 1.6745 - precision: 0.6917 - recall: 0.0595 - val\_accuracy: 0.3103 - val\_loss: 1.7449 - val\_precision: 0.6667 - val\_recall: 0.0086  
Epoch 7/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 0.4046 - loss: 1.6579 - precision: 0.6290 - recall: 0.0473 - val\_accuracy: 0.3190 - val\_loss: 1.7255 - val\_precision: 0.4000 - val\_recall: 0.0086  
Epoch 8/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 97ms/step - accuracy: 0.4180 - loss: 1.6597 - precision: 0.6252 - recall: 0.0625 - val\_accuracy: 0.3147 - val\_loss: 1.7399 - val\_precision: 0.4355 - val\_recall: 0.1164  
Epoch 9/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 0.4085 - loss: 1.6489 - precision: 0.6134 - recall: 0.2086 - val\_accuracy: 0.3276 - val\_loss: 1.7019 - val\_precision: 0.6429 - val\_recall: 0.0388  
Epoch 10/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 0.4250 - loss: 1.6375 - precision: 0.5973 - recall: 0.0978 - val\_accuracy: 0.3233 - val\_loss: 1.7813 - val\_precision: 0.4828 - val\_recall: 0.1207  
Epoch 11/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 0.4123 - loss: 1.6663 - precision: 0.5516 - recall: 0.1761 - val\_accuracy: 0.3276 - val\_loss: 1.7555 - val\_precision: 0.4390 - val\_recall: 0.1552  
Epoch 12/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 0.4180 - loss: 1.6328 - precision: 0.6046 - recall: 0.1783 - val\_accuracy: 0.3448 - val\_loss: 1.6769 - val\_precision: 0.6667 - val\_recall: 0.0603  
Epoch 13/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - accuracy: 0.4395 - loss: 1.5956 - precision: 0.6257 - recall: 0.1168 - val\_accuracy: 0.3233 - val\_loss: 1.6749 - val\_precision: 0.5758 - val\_recall: 0.0819  
Epoch 14/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 101ms/step - accuracy: 0.4479 - loss: 1.5698 - precision: 0.6903 - recall: 0.1494 - val\_accuracy: 0.3405 - val\_loss: 1.6811 - val\_precision: 0.6667 - val\_recall: 0.0948  
Epoch 15/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - accuracy: 0.4611 - loss: 1.5492 - precision: 0.6582 - recall: 0.1662 - val\_accuracy: 0.3534 - val\_loss: 1.6337 - val\_precision: 0.6000 - val\_recall: 0.0647  
Epoch 16/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - accuracy: 0.4316 - loss: 1.5951 - precision: 0.6420 - recall: 0.1221 - val\_accuracy: 0.3276 - val\_loss: 1.6589 - val\_precision: 0.6098 - val\_recall: 0.1078  
Epoch 17/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 0.4466 - loss: 1.5239 - precision: 0.6864 - recall: 0.2235 - val\_accuracy: 0.3405 - val\_loss: 1.6408 - val\_precision: 0.5172 - val\_recall: 0.1293  
Epoch 18/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 97ms/step - accuracy: 0.4540 - loss: 1.5350 - precision: 0.6742 - recall: 0.2059 - val\_accuracy: 0.3578 - val\_loss: 1.6670 - val\_precision: 0.5957 - val\_recall: 0.1207  
Epoch 19/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - accuracy: 0.4597 - loss: 1.5466 - precision: 0.6896 - recall: 0.1681 - val\_accuracy: 0.3621 - val\_loss: 1.6248 - val\_precision: 0.4756 - val\_recall: 0.1681  
Epoch 20/20  
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - accuracy: 0.4685 - loss: 1.5000 - precision: 0.6504 - recall: 0.2329 - val\_accuracy: 0.3707 - val\_loss: 1.6735 - val\_precision: 0.7826 - val\_recall: 0.0776

### Plot the following:

1. Training vs Validation accuracy
2. Training vs Validation loss

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 left')  
plt.title('Training and Validation Accuracy')  
  
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()



### Test the model on a picture of a boat (random picture downloaded on the internet)

import matplotlib.image as mpimg  
  
path = 'view-small-boat.jpg'  
  
image = mpimg.imread(path)  
plt.imshow(image)  
plt.axis('off')   
plt.show()  
  
img = tf.keras.utils.load\_img(  
 path, target\_size=(img\_height, img\_width)  
)  
img\_array = tf.keras.utils.img\_to\_array(img)  
img\_array = tf.expand\_dims(img\_array, 0)  
  
predictions = model.predict(img\_array)  
score = tf.nn.softmax(predictions[0])  
  
print(  
 "This image most likely belongs to {} with a {:.2f} percent confidence."  
 .format(class\_names[np.argmax(score)], 100 \* np.max(score))  
)



1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step  
This image most likely belongs to sailboat with a 25.36 percent confidence.

### Evaluate the model and display Validation loss and Validation accuracy

images, labels = next(iter(val\_ds))  
test\_loss, test\_accuracy, test\_precision, test\_recall = model.evaluate(images, labels, batch\_size=batch\_size)  
  
print(f"Validation Loss: {test\_loss}")  
print(f"Validation Accuracy: {test\_accuracy}")

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.3750 - loss: 1.6395 - precision: 1.0000 - recall: 0.0938  
Validation Loss: 1.6394543647766113  
Validation Accuracy: 0.375

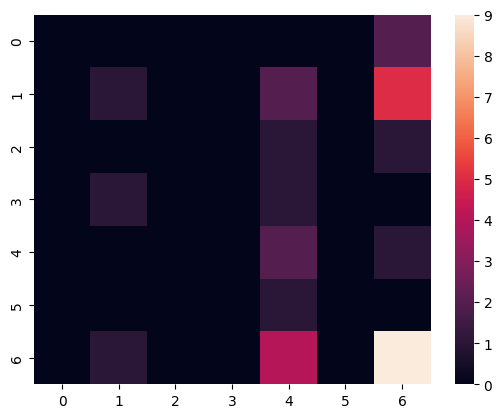
### Print confusion matrix, classification report, and heatmap of confusion matrix

import pandas as pd  
import seaborn as sns  
from sklearn.metrics import confusion\_matrix, classification\_report  
  
y\_actual = np.argmax(labels, axis=1)  
y\_pred = model.predict(images)  
y\_pred\_classes = np.argmax(y\_pred, axis=1)  
  
cm = confusion\_matrix(y\_actual, y\_pred\_classes)  
cr = classification\_report(y\_actual, y\_pred\_classes)  
print(cm)  
print(cr)  
sns.heatmap(cm)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step  
[[0 0 0 0 0 0 2]  
 [0 1 0 0 2 0 5]  
 [0 0 0 0 1 0 1]  
 [0 1 0 0 1 0 0]  
 [0 0 0 0 2 0 1]  
 [0 0 0 0 1 0 0]  
 [0 1 0 0 4 0 9]]  
 precision recall f1-score support  
  
 0 0.00 0.00 0.00 2  
 1 0.33 0.12 0.18 8  
 2 0.00 0.00 0.00 2  
 4 0.00 0.00 0.00 2  
 6 0.18 0.67 0.29 3  
 7 0.00 0.00 0.00 1  
 8 0.50 0.64 0.56 14  
  
 accuracy 0.38 32  
 macro avg 0.15 0.20 0.15 32  
weighted avg 0.32 0.38 0.32 32

c:\Users\jbham\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\metrics\\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, f"{metric.capitalize()} is", len(result))  
c:\Users\jbham\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\metrics\\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, f"{metric.capitalize()} is", len(result))  
c:\Users\jbham\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\metrics\\_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
 \_warn\_prf(average, modifier, f"{metric.capitalize()} is", len(result))

<Axes: >



## Step 2: Build a lightweight model meant to be deployed to a mobile device

### Load classified images from disk and split the Train and Validation datasets

batch\_size = 32  
img\_height = 256  
img\_width = 256  
  
train\_ds, val\_ds = tf.keras.utils.image\_dataset\_from\_directory(  
 'Automating\_Port\_Operations\_dataset',  
 validation\_split=0.3,  
 subset="both",  
 shuffle=True,  
 seed=1,  
 image\_size=(img\_height, img\_width),  
 batch\_size=batch\_size  
 )

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

### Adjust datasets for use in the CNN:

1. Apply normalization to images (x)
2. Change labels to categorical (y)

normalization\_layer = layers.Rescaling(1./255)  
  
train\_ds = train\_ds.map(lambda x, y: (normalization\_layer(x), keras.utils.to\_categorical(y, num\_classes=9)))  
val\_ds = val\_ds.map(lambda x, y: (normalization\_layer(x), keras.utils.to\_categorical(y, num\_classes=9)))

### Create the Convolutional Neural Network by specifying its layers

from keras.applications.mobilenet\_v2 import MobileNetV2  
  
num\_classes = 9  
model = Sequential([  
 MobileNetV2(include\_top=False, input\_shape=(img\_height, img\_width, 3)),  
 layers.GlobalAveragePooling2D(),  
 layers.Dropout(0.1),  
 layers.Dense(256, activation='relu'),  
 layers.BatchNormalization(),  
 layers.Dropout(0.1),  
 layers.Dense(128, activation='relu'),  
 layers.BatchNormalization(),  
 layers.Dropout(0.1),  
 layers.Dense(num\_classes, activation='softmax')  
])

C:\Users\jbham\AppData\Local\Temp\ipykernel\_22284\2526748378.py:5: UserWarning: `input\_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.  
 MobileNetV2(include\_top=False, input\_shape=(img\_height, img\_width, 3)),

### Compile the CNN

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

### Create Early Stopping callback to be passed into model.fit()

from keras.callbacks import EarlyStopping  
  
early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, verbose=1)

### Train the model on the training dataset and validate using the validation dataset

epochs=50  
history = model.fit(  
 train\_ds,  
 validation\_data=val\_ds,  
 epochs=epochs,  
 callbacks=[early\_stopping]  
)

Epoch 1/50  
26/26 ━━━━━━━━━━━━━━━━━━━━ 34s 864ms/step - accuracy: 0.4626 - loss: 1.8722 - precision: 0.5982 - recall: 0.3597 - val\_accuracy: 0.3420 - val\_loss: 3.4822 - val\_precision: 0.3758 - val\_recall: 0.3391  
Epoch 2/50  
26/26 ━━━━━━━━━━━━━━━━━━━━ 22s 829ms/step - accuracy: 0.8063 - loss: 0.6458 - precision: 0.8701 - recall: 0.7503 - val\_accuracy: 0.4684 - val\_loss: 2.8589 - val\_precision: 0.5211 - val\_recall: 0.4253  
Epoch 3/50  
26/26 ━━━━━━━━━━━━━━━━━━━━ 22s 824ms/step - accuracy: 0.8829 - loss: 0.3683 - precision: 0.9166 - recall: 0.8510 - val\_accuracy: 0.2586 - val\_loss: 2.5692 - val\_precision: 0.3043 - val\_recall: 0.2213  
Epoch 4/50  
26/26 ━━━━━━━━━━━━━━━━━━━━ 21s 820ms/step - accuracy: 0.9254 - loss: 0.2255 - precision: 0.9563 - recall: 0.9045 - val\_accuracy: 0.4368 - val\_loss: 2.4492 - val\_precision: 0.5125 - val\_recall: 0.4138  
Epoch 5/50  
26/26 ━━━━━━━━━━━━━━━━━━━━ 22s 829ms/step - accuracy: 0.9504 - loss: 0.1594 - precision: 0.9662 - recall: 0.9410 - val\_accuracy: 0.4626 - val\_loss: 2.0970 - val\_precision: 0.5271 - val\_recall: 0.4195  
Epoch 6/50  
26/26 ━━━━━━━━━━━━━━━━━━━━ 22s 827ms/step - accuracy: 0.9628 - loss: 0.1490 - precision: 0.9698 - recall: 0.9530 - val\_accuracy: 0.4483 - val\_loss: 2.9853 - val\_precision: 0.4983 - val\_recall: 0.4224  
Epoch 7/50  
26/26 ━━━━━━━━━━━━━━━━━━━━ 23s 861ms/step - accuracy: 0.9514 - loss: 0.1405 - precision: 0.9529 - recall: 0.9414 - val\_accuracy: 0.3764 - val\_loss: 4.1954 - val\_precision: 0.4106 - val\_recall: 0.3563  
Epoch 8/50  
26/26 ━━━━━━━━━━━━━━━━━━━━ 23s 893ms/step - accuracy: 0.9417 - loss: 0.1913 - precision: 0.9535 - recall: 0.9322 - val\_accuracy: 0.3276 - val\_loss: 8.4822 - val\_precision: 0.3404 - val\_recall: 0.3247  
Epoch 8: early stopping

### Plot the following:

1. Training vs Validation accuracy
2. Training vs Validation loss

epochs = early\_stopping.stopped\_epoch+1  
  
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='right')  
plt.title('Training and Validation Accuracy')  
  
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 left')  
plt.title('Training and Validation Loss')  
plt.show()



### Test the model on a picture of a boat (random picture downloaded on the internet)

import matplotlib.image as mpimg  
  
path = 'view-small-boat.jpg'  
  
image = mpimg.imread(path)  
plt.imshow(image)  
plt.axis('off')   
plt.show()  
  
img = tf.keras.utils.load\_img(  
 path, target\_size=(img\_height, img\_width)  
)  
img\_array = tf.keras.utils.img\_to\_array(img)  
img\_array = tf.expand\_dims(img\_array, 0)  
  
predictions = model.predict(img\_array)  
score = tf.nn.softmax(predictions[0])  
  
print(  
 "This image most likely belongs to {} with a {:.2f} percent confidence."  
 .format(class\_names[np.argmax(score)], 100 \* np.max(score))  
)



1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 516ms/step  
This image most likely belongs to sailboat with a 22.16 percent confidence.

### Evaluate the model and display Validation loss and Validation accuracy

images, labels = next(iter(val\_ds))  
test\_loss, test\_accuracy, test\_precision, test\_recall = model.evaluate(images, labels, batch\_size=batch\_size)  
  
print(f"Validation Loss: {test\_loss}")  
print(f"Validation Accuracy: {test\_accuracy}")

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 225ms/step - accuracy: 0.2812 - loss: 10.8304 - precision: 0.2903 - recall: 0.2812  
Validation Loss: 10.830405235290527  
Validation Accuracy: 0.28125

## Step 3: Compare Models

* Observations:
  1. Both Training and Validation performed decently in the Step 1 model, though Training did much better than Validation in terms of accuracy and even the Training dataset did not get over 50%.
  2. In the Step 2 model, Training did exceptionally well, although the Validation set did not. Validation almost reached 50% but for some reason the Validation accuracy kept bouncing up and down and never settled.
* Conclusions:
  1. The superior performance of the Training and Validation datasets in the Step 2 model makes it the clear winner, which is no surprise, since it starts off using the well-trained MobileNetV2.
  2. However, it is worth noting that the Step 1 model was able to guess the random test image with more confidence, and when the models were evaluated using model.evaluate(), the Step 1 model did slightly better.
  3. Still, the Step 2 model shows superior accuracy by far.