Novel Application of an Object Detection Model

CSCI 4050U - Machine Learning Theory and Application - Final Project

Ajmain Khan | Mitch Nolte | Aanisha Newaz

December 12, 2023

Introduction:

Background / Problem:

Methodology:

Expected Outcomes:

Initial Setup

```
import os
from random import randint

import numpy as np
import pandas as pd
import cv2
import requests
from PIL import Image
from io import BytesIO
import matplotlib.pyplot as plt

import tensorflow as tf
import tensorflow_hub as hub

from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.model_selection import train_test_split
```

1 Data Preperation

Load Model

```
model_name = "mobilenetv2-035-128"

model_handle_map = {
  "mobilenetv2-035-128": "https://www.kaggle.com/models/google/mobilenet-v2/frameworks/TensorFlow2/variations/035-128-classification/versions/
}

model_image_size_map = {
  "mobilenetv2-035-128": 128,
}

model_handle = model_handle_map[model_name]

print(f"Selected model: {model_name} : {model_handle}")
```

```
Selected model: mobilenetv2-035-128 : https://www.kaggle.com/models/google/mobilenet-v2/frameworks/TensorFlow2/variations/035-128-classifi
```

Load Dataset

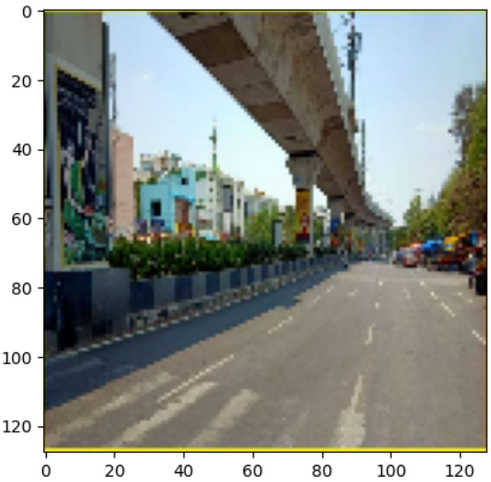
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

folder_path = '/content/drive/MyDrive/'
%cd {folder_path}
```

```
Mounted at /content/drive
/content/drive/MyDrive
```

```
images_path = 'cleandirty-road-classification/Images/'

labels_df = pd.read_csv('cleandirty-road-classification/metadata.csv')
print('\n\nlabels dataframe: \n', labels_df.head(), '\n\n')

class_names = ('clean', 'dirty')
num_classes = len(class_names)

img_size = (128, 128, 3)

print(f'{num_classes} classes: {class_names}\nimage size: {img_size}')

labels = []
images = []

for image in labels_df.itertuples(index=False):
    image_path = os.path.join(images_path, image[0])
    image_data = cv2.imread(image_path, cv2.IMREAD_COLOR)

    if image_data is not None:
        resized_image = cv2.resize(image_data, img_size[0:2])
        rgb_image = resized_image[:, :, ::-1]
        images.append(np.asarray(rgb_image))

        label = np.zeros(num_classes)
        label[int(image[1])] = 1
        labels.append(label)
    else:
        print(f"Error reading image from {image_path} - File may not exist or be inaccessible")

labels = np.asarray(labels)
images = np.asarray(images)

print(f'\nlabels shape: {labels.shape}')
print(f'images shape: {images.shape}')
```

```
labels dataframe:
         filename  label
0    dirty_2.jpg      1
1   clean_36.jpg      0
2   clean_31.jpg      0
3   dirty_69.jpg      1
4  clean_113.jpg      0


2 classes: ('clean', 'dirty')
image size: (128, 128, 3)

labels shape: (237, 2)
images shape: (237, 128, 128, 3)
```

```
plt.imshow(images[1]);
```



Utilities

```
def preprocess_image(image):
  image = np.array(image)
  # reshape into shape [batch_size, height, width, num_channels]
  img_reshaped = tf.reshape(image, [1, image.shape[0], image.shape[1], image.shape[2]])
  # Use `convert_image_dtype` to convert to floats in the [0,1] range.
  image = tf.image.convert_image_dtype(img_reshaped, tf.float32)
  return image


def show_image(image, title=''):
  image_size = image.shape[1]
  w = (image_size * 6) // 320
  plt.figure(figsize=(w, w))
  plt.imshow(image[0], aspect='equal')
  plt.axis('off')
  plt.title(title)
  plt.show()
```

2 Testing Pre-Trained Model

```
demoImage = preprocess_image(images[1])
classifier = hub.load(model_handle)

input_shape = demoImage.shape
warmup_input = tf.random.uniform(input_shape, 0, 1.0)
%time warmup_logits = classifier(warmup_input).numpy()
```

```
CPU times: user 726 ms, sys: 2.04 ms, total: 728 ms
Wall time: 756 ms
```

```
max_dynamic_size = 512
if model_name in model_image_size_map:
  image_size = model_image_size_map[model_name]
  dynamic_size = False
  print(f"Images will be converted to {image_size}x{image_size}")
else:
  dynamic_size = True
  print(f"Images will be capped to a max size of {max_dynamic_size}x{max_dynamic_size}")

labels_file = "https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt"

#download labels and creates a maps
downloaded_file = tf.keras.utils.get_file("labels.txt", origin=labels_file)

classes = []

with open(downloaded_file) as f:
  labels = f.readlines()
  classes = [l.strip() for l in labels]
```

```
Images will be converted to 128x128
```

```
max_dynamic_size = 512
if model_name in model_image_size_map:
  image_size = model_image_size_map[model_name]
  dynamic_size = False
  print(f"Images will be converted to {image_size}x{image_size}")
else:
  dynamic_size = True
  print(f"Images will be capped to a max size of {max_dynamic_size}x{max_dynamic_size}")

labels_file = "https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt"

#download labels and creates a maps
downloaded_file = tf.keras.utils.get_file("labels.txt", origin=labels_file)

classes = []

with open(downloaded_file) as f:
  labels = f.readlines()
  classes = [l.strip() for l in labels]
```

```
Images will be converted to 128x128
```

```
# Run model on image
%time probabilities = tf.nn.softmax(classifier(demoImage)).numpy()

top_5 = tf.argsort(probabilities, axis=-1, direction="DESCENDING")[0][:5].numpy()
np_classes = np.array(classes)

# Some models include an additional 'background' class in the predictions, so
# we must account for this when reading the class labels.
includes_background_class = probabilities.shape[1] == 1001

for i, item in enumerate(top_5):
  class_index = item if includes_background_class else item + 1
  line = f'({i+1}) {class_index:4} - {classes[class_index]}: {probabilities[0][top_5][i]}'
  print(line)

show_image(demoImage, '')
```

```
CPU times: user 10.5 ms, sys: 778 µs, total: 11.2 ms
Wall time: 10.8 ms
(1)  840 - suspension bridge: 0.08165517449378967
(2)  707 - patio: 0.07770338654518127
(3)  836 - sundial: 0.04197487607598305
(4)  443 - bell cote: 0.04000386595726013
(5)  683 - obelisk: 0.03360437974333763
```

```
demoImage = preprocess_image(images[120])
show_image(demoImage, '')
```

```
%time probabilities = tf.nn.softmax(classifier(demoImage)).numpy()

top_5 = tf.argsort(probabilities, axis=-1, direction="DESCENDING")[0][:5].numpy()
np_classes = np.array(classes)

includes_background_class = probabilities.shape[1] == 1001

for i, item in enumerate(top_5):
  class_index = item if includes_background_class else item + 1
  line = f'({i+1}) {class_index:4} - {classes[class_index]}: {probabilities[0][top_5][i]}'
  print(line)

show_image(demoImage, '')
```

```
CPU times: user 11.7 ms, sys: 0 ns, total: 11.7 ms
Wall time: 21.9 ms
(1)  666 - moped: 0.39420121908187866
(2)  492 - chain saw: 0.1085754856467247
(3)  831 - stretcher: 0.0972609743475914
(4)  671 - motor scooter: 0.051123932003974915
(5)  803 - snowmobile: 0.025528404861688614
```

3 Training

Transfer Learning

```
# model_name = "mobilenetv2-035-128"

# model_handle_map = {
#   "mobilenetv2-035-128": "https://www.kaggle.com/models/google/mobilenet-v2/frameworks/TensorFlow2/variations/035-128-feature-vector/version
# }

# model_image_size_map = {
#   "mobilenetv2-035-128": 128,
# }

model_handle = model_handle_map.get(model_name)
pixels = model_image_size_map.get(model_name, 128)

print(f"Selected model: {model_name} : {model_handle}")

IMAGE_SIZE = (pixels, pixels)
print(f"Input size {IMAGE_SIZE}")

BATCH_SIZE = 4
```

```
Selected model: mobilenetv2-035-128 : https://www.kaggle.com/models/google/mobilenet-v2/frameworks/TensorFlow2/variations/035-128-feature-
Input size (128, 128)
```

```
# Display 16 pictures from the dataset
fig, axs = plt.subplots(4, 4, figsize=(10, 10))

for x in range(4):
    for y in range(4):
        i = randint(0, len(images))

        axs[x][y].imshow(images[i])

        # delete x and y ticks and set x label as picture label
        axs[x][y].set_xticks([])
        axs[x][y].set_yticks([])
        axs[x][y].set_xlabel(class_names[np.argmax(labels[i])])

plt.show()
```


```
images_path = 'cleandirty-road-classification/Images/'

labels_df = pd.read_csv('cleandirty-road-classification/metadata.csv')
print('\n\nlabels dataframe: \n', labels_df.head(), '\n\n')

class_names = ('clean', 'dirty')
num_classes = len(class_names)

img_size = (128, 128, 3)

print(f'{num_classes} classes: {class_names}\nimage size: {img_size}')


labels = []
images = []
for image in labels_df.iloc:
    images.append(np.asarray(cv2.resize(cv2.imread(images_path + image[0], cv2.IMREAD_COLOR), img_size[0:2])[:, :, ::-1]))

    # labels will be in the form of a vector: [0, 1] or [1, 0]
    label = np.zeros(num_classes)
    label[image[1]] = 1
    labels.append(label)

labels = np.asarray(labels)
images = np.asarray(images)

print(f'\nlabels shape: {labels.shape}')
print(f'images shape: {images.shape}')
```

```
labels dataframe:
         filename  label
0    dirty_2.jpg      1
1   clean_36.jpg      0
2   clean_31.jpg      0
3   dirty_69.jpg      1
4  clean_113.jpg      0


2 classes: ('clean', 'dirty')
image size: (128, 128, 3)

labels shape: (237, 2)
images shape: (237, 128, 128, 3)
```

```
# DEFINE TRAIN/TEST SPLIT
X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.1, random_state=42)

print(f'train images shape: {X_train.shape}\ntrain labels shape: {y_train.shape}\n\nvalidation images shape: {X_val.shape}\nvalidation labels
```

```
train images shape: (213, 128, 128, 3)
train labels shape: (213, 2)

validation images shape: (24, 128, 128, 3)
validation labels shape: (24, 2)
```

```
# ImageDataGenerator for train images
train_images_generator = tf.keras.preprocessing.image.ImageDataGenerator(shear_range=0.3,
                                                                          rotation_range=15,
                                                                          zoom_range=0.3,
                                                                          vertical_flip=True,
                                                                          horizontal_flip=True)
train_images_generator = train_images_generator.flow(X_train, y=y_train)

# ImageDataGenerator for validation images
validation_images_generator = tf.keras.preprocessing.image.ImageDataGenerator(vertical_flip=True,
                                                                               horizontal_flip=True)
validation_images_generator = validation_images_generator.flow(X_val, y=y_val)
```

```
do_fine_tuning = False

print("Building model with", model_handle)
model = tf.keras.Sequential([
    # Explicitly define the input shape so the model can be properly
    # loaded by the TFLiteConverter
    tf.keras.layers.InputLayer(input_shape=IMAGE_SIZE + (3,)),
    hub.KerasLayer(model_handle, trainable=do_fine_tuning),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(len(class_names),
                          kernel_regularizer=tf.keras.regularizers.l2(0.0001))
])
model.build((None,)+IMAGE_SIZE+(3,))
model.summary()
```

```
Building model with https://www.kaggle.com/models/google/mobilenet-v2/frameworks/TensorFlow2/variations/035-128-feature-vector/versions/2
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              410208

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 2)                 2562

=================================================================
Total params: 412770 (1.57 MB)
Trainable params: 2562 (10.01 KB)
Non-trainable params: 410208 (1.56 MB)
_________________________________________________________________
```

```
# creating ModelChecpoint callback
checkpoint_callback = ModelCheckpoint('cnn_model/model{epoch:02d}')
```

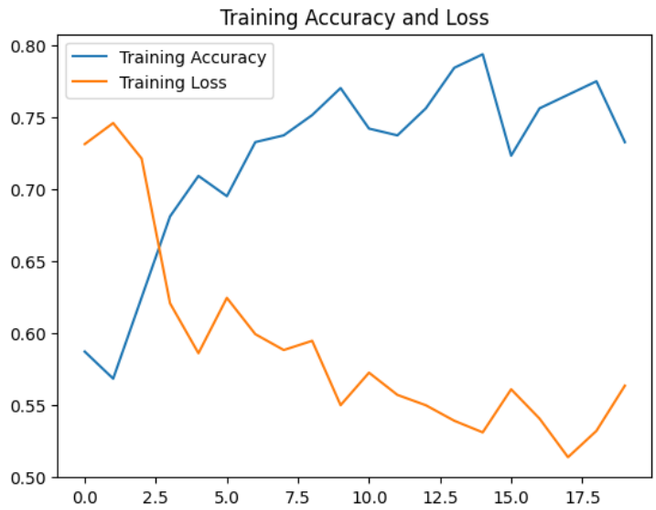
Training Model on Dataset

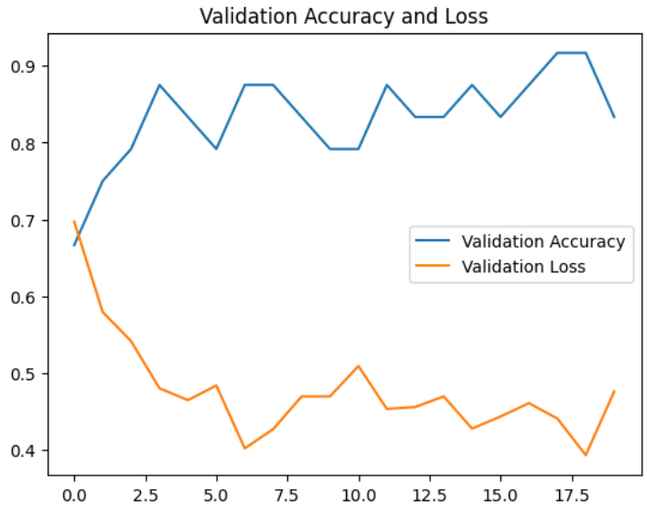
```
model.compile(
  optimizer='adam',
  loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True, label_smoothing=0.1),
  metrics=['accuracy'])

train_size = X_train.shape[0]
valid_size = X_val.shape[0]
steps_per_epoch = train_size // BATCH_SIZE
validation_steps = valid_size // BATCH_SIZE
hist = model.fit(
    train_images_generator,
    epochs=20,
    verbose=1,
    validation_data=validation_images_generator, callbacks=[checkpoint_callback])
```

```
Epoch 1/20
7/7 [==============================] - 22s 3s/step - loss: 0.7310 - accuracy: 0.5869 - val_loss: 0.6970 - val_accuracy: 0.6667
Epoch 2/20
7/7 [==============================] - 10s 2s/step - loss: 0.7456 - accuracy: 0.5681 - val_loss: 0.5796 - val_accuracy: 0.7500
Epoch 3/20
7/7 [==============================] - 9s 2s/step - loss: 0.7210 - accuracy: 0.6244 - val_loss: 0.5417 - val_accuracy: 0.7917
Epoch 4/20
7/7 [==============================] - 8s 1s/step - loss: 0.6204 - accuracy: 0.6808 - val_loss: 0.4803 - val_accuracy: 0.8750
Epoch 5/20
7/7 [==============================] - 9s 1s/step - loss: 0.5857 - accuracy: 0.7089 - val_loss: 0.4649 - val_accuracy: 0.8333
Epoch 6/20
7/7 [==============================] - 9s 2s/step - loss: 0.6242 - accuracy: 0.6948 - val_loss: 0.4840 - val_accuracy: 0.7917
Epoch 7/20
7/7 [==============================] - 8s 1s/step - loss: 0.5990 - accuracy: 0.7324 - val_loss: 0.4022 - val_accuracy: 0.8750
Epoch 8/20
7/7 [==============================] - 10s 2s/step - loss: 0.5879 - accuracy: 0.7371 - val_loss: 0.4272 - val_accuracy: 0.8750
Epoch 9/20
7/7 [==============================] - 7s 1s/step - loss: 0.5944 - accuracy: 0.7512 - val_loss: 0.4697 - val_accuracy: 0.8333
Epoch 10/20
7/7 [==============================] - 10s 2s/step - loss: 0.5496 - accuracy: 0.7700 - val_loss: 0.4698 - val_accuracy: 0.7917
Epoch 11/20
7/7 [==============================] - 8s 1s/step - loss: 0.5722 - accuracy: 0.7418 - val_loss: 0.5093 - val_accuracy: 0.7917
Epoch 12/20
7/7 [==============================] - 11s 2s/step - loss: 0.5568 - accuracy: 0.7371 - val_loss: 0.4534 - val_accuracy: 0.8750
Epoch 13/20
7/7 [==============================] - 9s 2s/step - loss: 0.5496 - accuracy: 0.7559 - val_loss: 0.4560 - val_accuracy: 0.8333
Epoch 14/20
7/7 [==============================] - 8s 1s/step - loss: 0.5388 - accuracy: 0.7840 - val_loss: 0.4698 - val_accuracy: 0.8333
Epoch 15/20
7/7 [==============================] - 9s 1s/step - loss: 0.5307 - accuracy: 0.7934 - val_loss: 0.4279 - val_accuracy: 0.8750
Epoch 16/20
7/7 [==============================] - 9s 1s/step - loss: 0.5607 - accuracy: 0.7230 - val_loss: 0.4436 - val_accuracy: 0.8333
Epoch 17/20
7/7 [==============================] - 8s 1s/step - loss: 0.5403 - accuracy: 0.7559 - val_loss: 0.4610 - val_accuracy: 0.8750
Epoch 18/20
7/7 [==============================] - 10s 2s/step - loss: 0.5135 - accuracy: 0.7653 - val_loss: 0.4410 - val_accuracy: 0.9167
Epoch 19/20
7/7 [==============================] - 8s 1s/step - loss: 0.5318 - accuracy: 0.7746 - val_loss: 0.3934 - val_accuracy: 0.9167
Epoch 20/20
7/7 [==============================] - 10s 2s/step - loss: 0.5632 - accuracy: 0.7324 - val_loss: 0.4762 - val_accuracy: 0.8333
```

4 Validation

```
accuracy = hist.history['accuracy']
val_accuracy = hist.history['val_accuracy']

loss = hist.history['loss']
val_loss = hist.history['val_loss']

epochs = range(len(accuracy))


plt.figure()
plt.plot(epochs, accuracy, label='Training Accuracy')
plt.plot(epochs, loss, label='Training Loss')
plt.legend()
plt.title('Training Accuracy and Loss')

plt.figure()
plt.plot(epochs, val_accuracy, label='Validation Accuracy')
plt.plot(epochs, val_loss, label='Validation Loss')
plt.legend()
plt.title('Validation Accuracy and Loss')

plt.show()
```






```
fig, axs = plt.subplots(5, 4, figsize=(13, 16))

i = 0
for x in range(5):
    for y in range(4):
        prediction = model.predict(X_val[i][None, ...], verbose=0)[0]

        axs[x][y].set_xticks([])
        axs[x][y].set_yticks([])

        if np.argmax(prediction) != np.argmax(y_val[i]):
            axs[x][y].set_xlabel(f'prediction: {class_names[np.argmax(prediction)]} | label: {class_names[np.argmax(y_val[i])]}', color='red')
        else:
            axs[x][y].set_xlabel(f'prediction: {class_names[np.argmax(prediction)]} | label: {class_names[np.argmax(y_val[i])]}')

        axs[x][y].imshow(X_val[i])

        i += 1
plt.show()
```


prediction: dirty | label: dirty


prediction: dirty | label: dirty


prediction: dirty | label: dirty


prediction: clean | label: clean


prediction: clean | label: clean


prediction: clean | label: clean

prediction: clean | label: clean


prediction: dirty | label: dirty


prediction: dirty | label: dirty


prediction: dirty | label: dirty


prediction: dirty | label: dirty

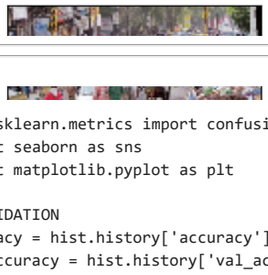

prediction: clean | label: clean

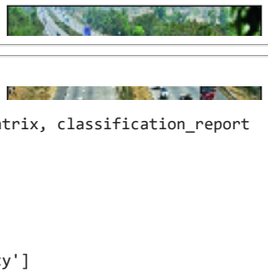

prediction: dirty | label: dirty

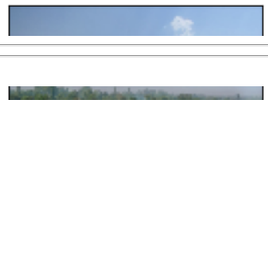

prediction: dirty | label: dirty

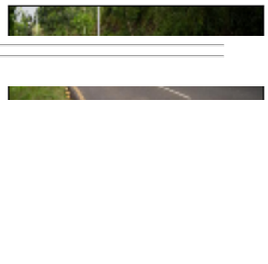
prediction: dirty | label: clean


prediction: clean | label: dirty

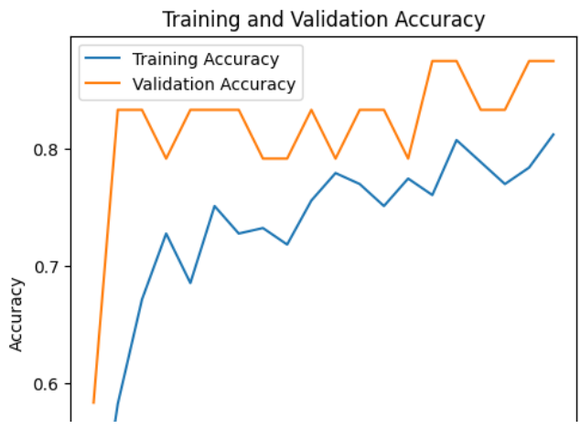


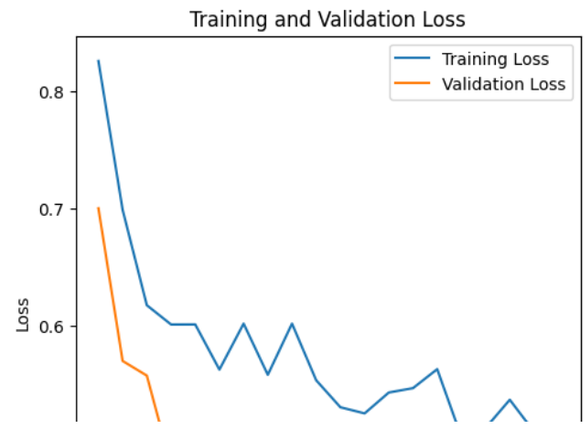






```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# VALIDATION
accuracy = hist.history['accuracy']
val_accuracy = hist.history['val_accuracy']

loss = hist.history['loss']
val_loss = hist.history['val_loss']

epochs = range(len(accuracy))

# Plot training accuracy and loss
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs, accuracy, label='Training Accuracy')
plt.plot(epochs, val_accuracy, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, loss, label='Training Loss')
plt.plot(epochs, val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

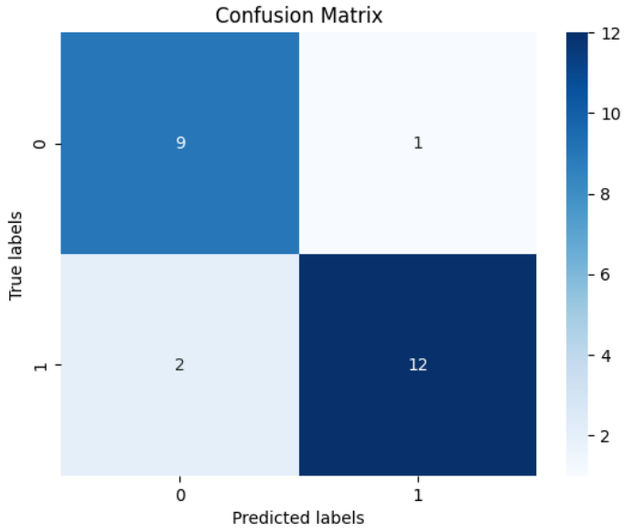




Aanisha's Mediocre gpt4 $30 prompt

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Assuming 'X_val' is your validation set and 'y_val' is the corresponding true labels
# First, get the predictions from the model
y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_val, axis=1)

cm = confusion_matrix(y_true, y_pred_classes)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

```
1/1 [==============================] - 1s 526ms/step
```




```
from sklearn.metrics import classification_report

print(classification_report(y_true, y_pred_classes, target_names=class_names))
```

```
              precision    recall  f1-score   support

       clean       0.82      0.90      0.86        10
       dirty       0.92      0.86      0.89        14

    accuracy                           0.88        24
   macro avg       0.87      0.88      0.87        24
weighted avg       0.88      0.88      0.88        24
```

```
error_indices = np.where(y_pred_classes != y_true)[0]
for i in error_indices[:10]:  # Display first 10 errors
    print(f"Index: {i}, Predicted: {class_names[y_pred_classes[i]]}, True: {class_names[y_true[i]]}")
    # Optionally, display the image: plt.imshow(X_val[i])
```

```
Index: 8, Predicted: clean, True: dirty
```