plant-prediction

November 1, 2024

[61]:
```python
import numpy as np
import pandas as pd

#import os
#for dirname, _, filenames in os.walk('C:/Users/KIIT0001/Desktop/archive/
 ↪medicinal/Indian Medicinal Leaves Image Datasets/medicinal_plant_dataset'):
#    for filename in filenames:
#        print(os.path.join(dirname, filename))
```

[6]:
```python
import tensorflow as tf
```

[7]:
```python
dataset = tf.keras.utils.image_dataset_from_directory("C:/Users/KIIT0001/
 ↪Desktop/archive/medicinal/Indian Medicinal Leaves Image Datasets/
 ↪medicinal_plant_dataset")
```

```
Found 5945 files belonging to 40 classes.
```

[ ]:
```python
ds_train = tf.keras.utils.image_dataset_from_directory(
    'C:/Users/KIIT0001/Desktop/archive/medicinal/Indian Medicinal Leaves Image␣
 ↪Datasets/medicinal_plant_dataset',
    image_size=(224, 224),
    batch_size=16,
    validation_split=0.4,
    subset="training",
    seed=42
)
```

```
Found 5945 files belonging to 40 classes.
Using 3567 files for training.
```

[ ]:
```python
ds_val = tf.keras.utils.image_dataset_from_directory(
    'C:/Users/KIIT0001/Desktop/archive/medicinal/Indian Medicinal Leaves Image␣
 ↪Datasets/medicinal_plant_dataset',
    image_size=(224, 224),
    batch_size=16,
    validation_split=0.4,
    subset="validation",
    seed=42
```

```
)
```

```
Found 5945 files belonging to 40 classes.
Using 2378 files for validation.
```

[10]:
```python
import tensorflow_datasets as tfds

batch_size = 64
dataset_name = dataset
class_names = dataset.class_names

print(class_names)
```

```
['Aloevera', 'Amla', 'Amruta_Balli', 'Arali', 'Ashoka', 'Ashwagandha',
'Avacado', 'Bamboo', 'Basale', 'Betel', 'Betel_Nut', 'Brahmi', 'Castor',
'Curry_Leaf', 'Doddapatre', 'Ekka', 'Ganike', 'Gauva', 'Geranium', 'Henna',
'Hibiscus', 'Honge', 'Insulin', 'Jasmine', 'Lemon', 'Lemon_grass', 'Mango',
'Mint', 'Nagadali', 'Neem', 'Nithyapushpa', 'Nooni', 'Pappaya', 'Pepper',
'Pomegranate', 'Raktachandini', 'Rose', 'Sapota', 'Tulasi', 'Wood_sorel']
```

[11]:
```python
len(class_names)
```

[11]:
```
40
```

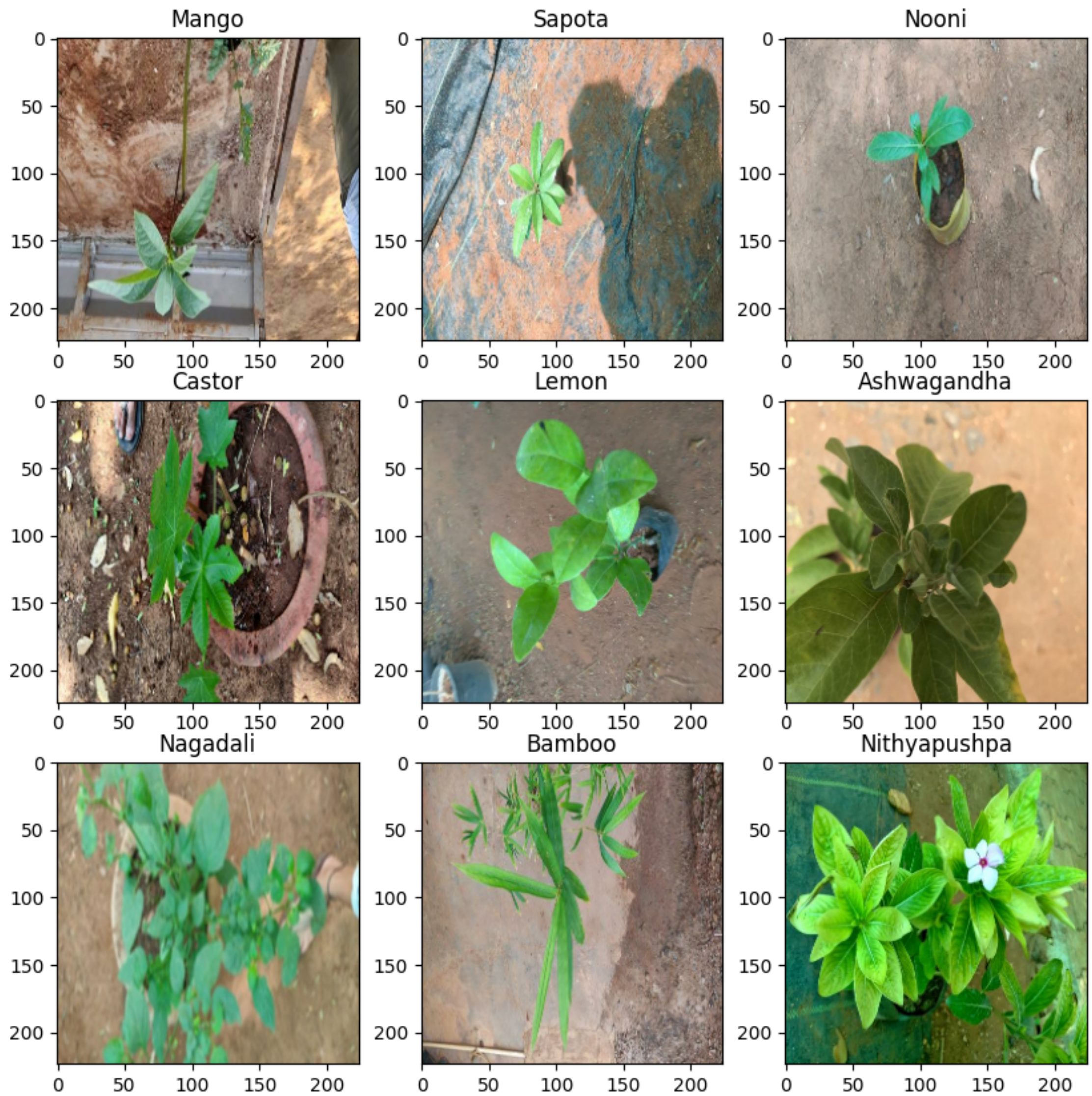
[12]:
```python
type(class_names)
```

[12]:
```
list
```

[13]:
```python
size = (224,224)
ds_train = ds_train.map(lambda image, label: (tf.image.resize(image,size) ,␣
↪label))
ds_val = ds_train.map(lambda image, label: (tf.image.resize(image,size) ,␣
↪label))
```

[14]:
```python
import matplotlib.pyplot as plt
```

[15]:
```python
plt.figure(figsize = (10,10))
for images,labels in ds_train.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("on")
```



[16]:

```
# Initialize an empty list to store images
images_list = []

# Initialize an empty NumPy array for labels
labels_array = np.array([])

# Iterate through the dataset
for image_batch, label_batch in ds_train:
    # Append the image batch to the images list
    images_list.append(image_batch.numpy())

    # Concatenate the label batch to the existing labels_array
    labels_array = np.concatenate([labels_array, label_batch.numpy()], axis=0)
```

```python
# Concatenate the list of image batches into a NumPy array
images_array = np.concatenate(images_list, axis=0)

# Now, 'images_array' contains all the images in a 4D NumPy array, and␣
↪'labels_array' contains all the labels as a NumPy array
```

[17]:
```python
images_array.shape
```

[17]:
```
(3567, 224, 224, 3)
```

[18]:
```python
labels_array
```

[18]:
```
array([18., 25., 25., …, 20.,  1.,  2.])
```

[19]:
```python
images_array[0]
```

[19]:
```
array([[[184.7496 , 180.7496 , 177.7496 ],
        [195.97075, 191.97075, 188.97075],
        [199.6789 , 194.6789 , 190.6789 ],
        …,
        [166.30316, 156.30316, 146.30316],
        [172.80484, 162.80484, 152.80484],
        [169.85579, 159.85579, 149.85579]],

       [[190.51427, 186.20624, 182.89821],
        [196.19077, 191.88274, 188.57469],
        [189.06744, 184.06744, 180.06744],
        …,
        [167.57985, 157.57985, 147.8879 ],
        [170.82776, 160.82776, 151.13579],
        [170.25853, 160.25853, 150.25853]],

       [[194.62302, 189.62302, 185.62302],
        [194.40865, 189.40865, 185.40865],
        [181.74083, 176.74083, 172.74083],
        …,
        [168.41255, 158.41255, 149.41255],
        [169.27028, 159.27028, 150.27028],
        [172.21446, 162.21446, 152.21446]],

       …,

       [[208.2597 , 203.2597 , 199.2597 ],
        [219.00333, 211.00333, 208.00333],
        [214.03223, 206.03223, 203.03223],
        …,
```

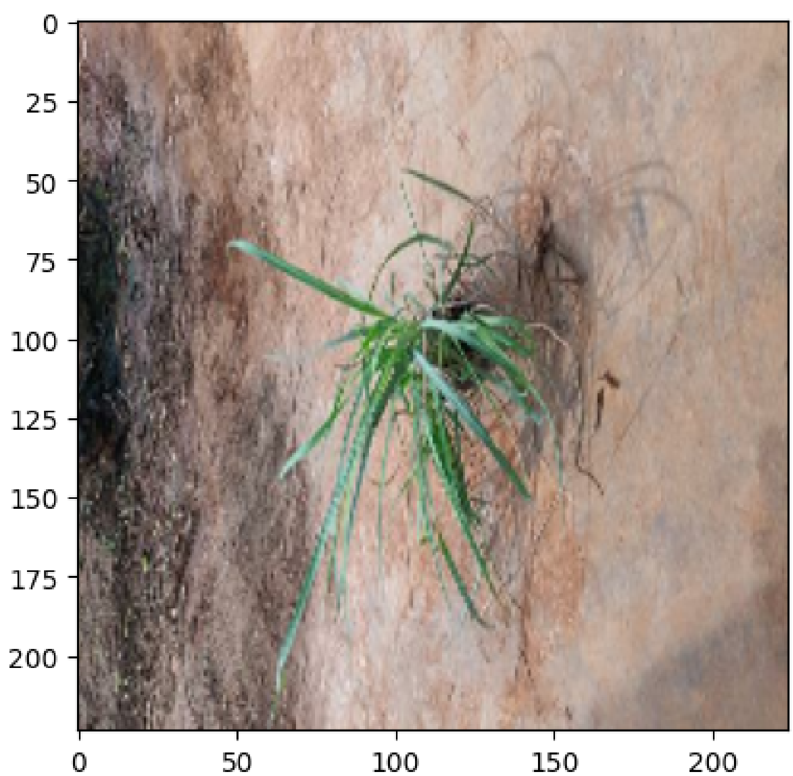
```
 [176.33624, 177.51117, 175.89697],
 [169.93007, 167.97415, 166.96475],
 [164.251  , 162.02573, 159.46089]],

[[216.25323, 209.17728, 205.86926],
 [215.01393, 207.01393, 204.01393],
 [216.77718, 208.08519, 203.70122],
 …,
 [168.1254 , 170.1254 , 168.85358],
 [180.52542, 178.22838, 179.648  ],
 [197.1066 , 193.94708, 189.51361]],

[[225.57068, 217.57068, 214.57068],
 [215.28612, 207.28612, 204.28612],
 [190.13632, 181.13632, 176.13632],
 …,
 [173.1664 , 175.1664 , 173.77357],
 [172.84341, 170.84341, 172.87906],
 [184.34416, 179.54948, 175.15842]]], dtype=float32)
```

[20]:
```
plt.imshow(images_array[2].astype("uint8"))
plt.axis("on")
```

[20]: (-0.5, 223.5, 223.5, -0.5)



[21]:
```
class_names[int(labels_array[2])]
```

[21]: 'Lemon_grass'

Training the model

[22]:
```
from tensorflow.keras.layers import Input,Dense,Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
```

[23]:
```
IMAGE_SIZE = [224,224]
```

[24]:
```
vgg = VGG16(input_shape=IMAGE_SIZE +[3] , weights = 'imagenet' , include_top =␣
↪False)
```

[25]:
```
for layer in vgg.layers:
    layer.trainable = False
```

[26]:
```
x = Flatten()(vgg.output)
```

[27]:
```
prediction = Dense(len(class_names) , activation = "softmax")(x)
```

[28]:
```
model = Model(inputs = vgg.input,outputs = prediction)
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584

```
block2_pool (MaxPooling2D)        (None, 56, 56, 128)                  0

block3_conv1 (Conv2D)             (None, 56, 56, 256)            295,168

block3_conv2 (Conv2D)             (None, 56, 56, 256)            590,080

block3_conv3 (Conv2D)             (None, 56, 56, 256)            590,080

block3_pool (MaxPooling2D)        (None, 28, 28, 256)                  0

block4_conv1 (Conv2D)             (None, 28, 28, 512)          1,180,160

block4_conv2 (Conv2D)             (None, 28, 28, 512)          2,359,808

block4_conv3 (Conv2D)             (None, 28, 28, 512)          2,359,808

block4_pool (MaxPooling2D)        (None, 14, 14, 512)                  0

block5_conv1 (Conv2D)             (None, 14, 14, 512)          2,359,808

block5_conv2 (Conv2D)             (None, 14, 14, 512)          2,359,808

block5_conv3 (Conv2D)             (None, 14, 14, 512)          2,359,808

block5_pool (MaxPooling2D)        (None, 7, 7, 512)                    0

flatten (Flatten)                 (None, 25088)                        0

dense (Dense)                     (None, 40)                   1,003,560
```

Total params: 15,718,248 (59.96 MB)

Trainable params: 1,003,560 (3.83 MB)

Non-trainable params: 14,714,688 (56.13 MB)

[29]:

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

[30]:

```
labels_array
```

```
[30]: array([18., 25., 25., …, 20.,  1.,  2.])
```

```
[31]: # One-hot encode the labels
labels_one_hot = tf.keras.utils.to_categorical(labels_array, num_classes=40)
```

```
[32]: r = model.fit(
    images_array,
    labels_one_hot,
    batch_size=16,
    epochs = 5
)
```

```
Epoch 1/5
223/223              973s 4s/step -
accuracy: 0.4400 - loss: 19.5057
Epoch 2/5
223/223              1035s 5s/step -
accuracy: 0.9143 - loss: 2.8841
Epoch 3/5
223/223              593s 3s/step -
accuracy: 0.9565 - loss: 1.3523
Epoch 4/5
223/223              455s 2s/step -
accuracy: 0.9685 - loss: 1.1977
Epoch 5/5
223/223              453s 2s/step -
accuracy: 0.9800 - loss: 0.9391
```

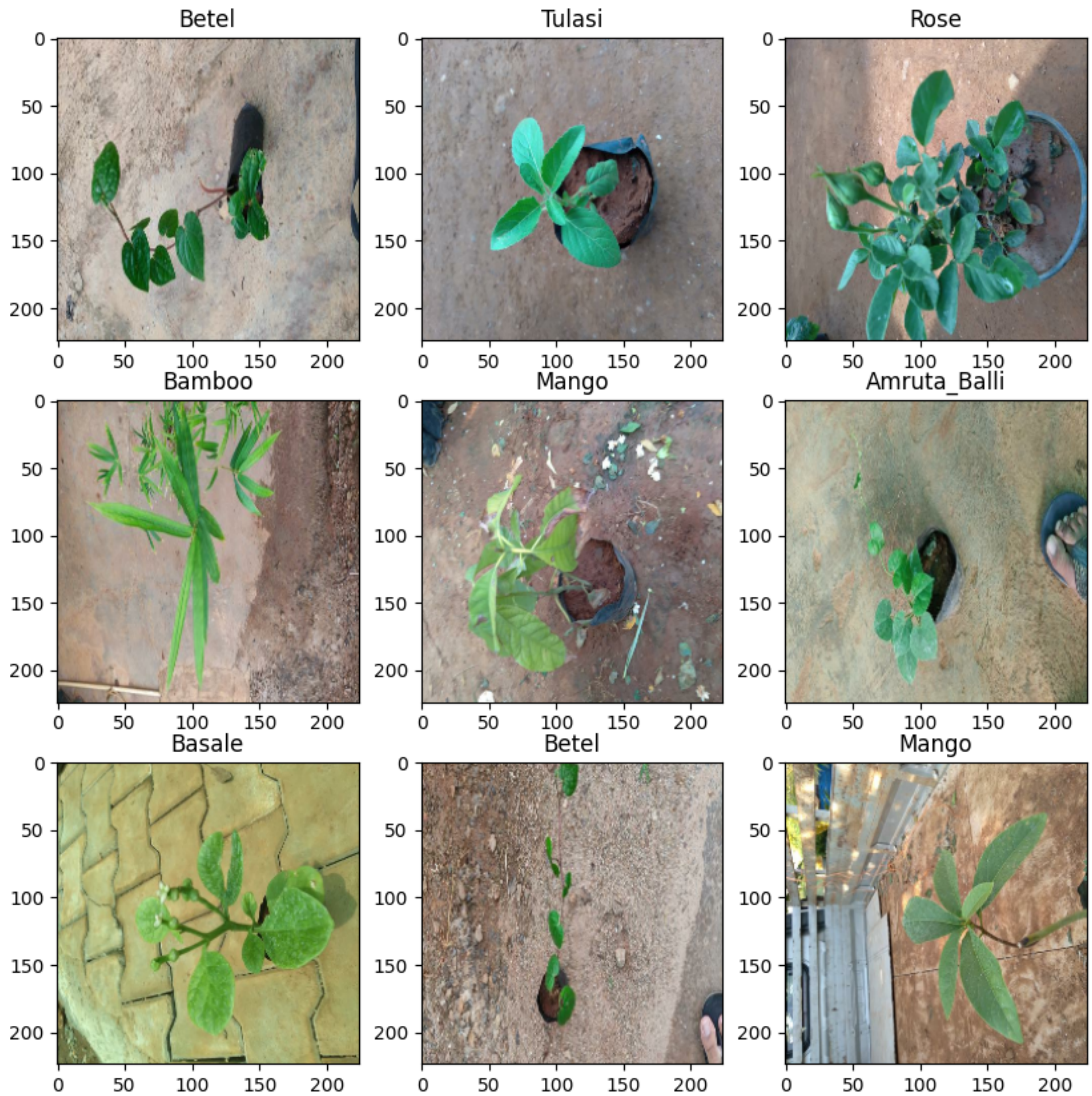
```
[33]: # Save the entire model, including architecture, weights, and optimizer state
model.save("my_model.h5")  # Provide a filename with a '.h5' extension
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

Testing model;

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

```
[35]: plt.figure(figsize = (10,10))
for images,labels in ds_val.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("on")
```



[36]:
```python
import numpy as np
```

[37]:
```python
# Initialize empty lists to store images and labels
images_list_v = []
labels_list_v = []

# Iterate through the dataset
for image_batch, label_batch in ds_val:
    # Append the image batch to the images list
    images_list_v.append(image_batch.numpy())

    # Append the label batch to the labels list
    labels_list_v.append(label_batch.numpy())
```

```
# Concatenate the list of image batches into a NumPy array
images_array_v = np.concatenate(images_list_v, axis=0)

# Concatenate the list of label batches into a NumPy array
labels_array_v = np.concatenate(labels_list_v, axis=0)

# Now, 'images_array' contains all the images in a 4D NumPy array, and
↪'labels_array' contains all the corresponding labels as a NumPy array
```

[38]:
```
images_array_v.shape
```

[38]: (3567, 224, 224, 3)

[39]:
```
labels_array_v.shape
```

[39]: (3567,)

[40]:
```
# One-hot encode the labels
labels_one_hot_v = tf.keras.utils.to_categorical(labels_array_v, num_classes=40)
```

[41]:
```
model_new = tf.keras.models.load_model("my_model.h5")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.
```

[42]:
```
model_new.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0

block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 40)	1,003,560

Total params: 15,718,250 (59.96 MB)

Trainable params: 1,003,560 (3.83 MB)

Non-trainable params: 14,714,688 (56.13 MB)

Optimizer params: 2 (12.00 B)

[43]:
```
model_new.evaluate(images_array_v,labels_one_hot_v)
```

```
112/112              449s 4s/step -
accuracy: 0.9888 - loss: 0.7650
```

[43]: [0.45777466893196106, 0.9873843789100647]

[44]:
```
y_pred = model_new.predict(images_array_v)
```

```
112/112              454s 4s/step
```

[49]:
```
y_pred
```

[49]:
```
array([[0., 0., 0., …, 0., 0., 0.],
       [0., 1., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 0., 0., 0.],
       [0., 0., 0., …, 0., 0., 0.],
       [0., 1., 0., …, 0., 0., 0.]], dtype=float32)
```

[46]:
```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(labels_array_v,␣
 ↪y_pred_classes))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        99
           1       1.00      1.00      1.00        97
           2       0.94      1.00      0.97        94
           3       1.00      0.99      0.99        91
           4       1.00      1.00      1.00        89
           5       0.98      1.00      0.99        86
           6       0.94      1.00      0.97        81
           7       1.00      1.00      1.00        96
           8       1.00      0.98      0.99        87
           9       0.97      0.99      0.98        91
          10       1.00      0.99      0.99        96
          11       1.00      0.99      0.99        83
          12       0.96      1.00      0.98        98
          13       1.00      1.00      1.00        70
          14       0.99      1.00      0.99        90
          15       0.98      1.00      0.99       103
          16       0.98      0.91      0.95        69
          17       0.99      1.00      0.99        88
          18       0.99      1.00      0.99        82
          19       1.00      0.97      0.98        94
          20       1.00      1.00      1.00        88
          21       1.00      0.99      0.99        80
          22       1.00      0.89      0.94        87
          23       0.99      0.99      0.99       111
```

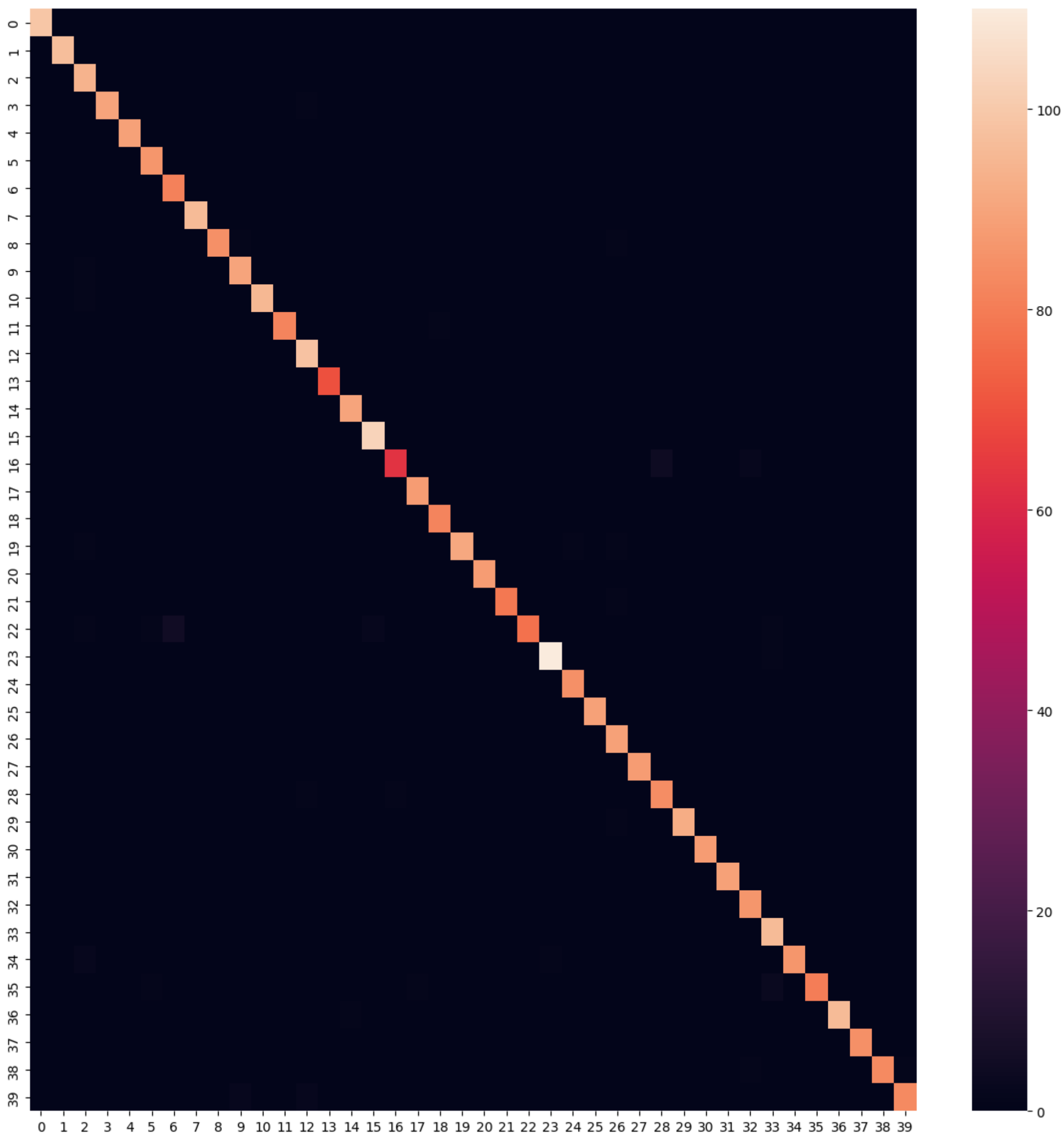
```
          24       0.99      1.00      0.99        85
          25       1.00      1.00      1.00        89
          26       0.96      1.00      0.98        89
          27       1.00      1.00      1.00        88
          28       0.95      0.98      0.97        86
          29       1.00      0.99      0.99        93
          30       1.00      1.00      1.00        88
          31       1.00      1.00      1.00        89
          32       0.97      1.00      0.98        86
          33       0.95      1.00      0.97        96
          34       1.00      0.97      0.98        89
          35       1.00      0.94      0.97        85
          36       1.00      0.99      0.99        97
          37       1.00      1.00      1.00        85
          38       1.00      0.98      0.99        85
          39       0.99      0.95      0.97        87

    accuracy                           0.99      3567
   macro avg       0.99      0.99      0.99      3567
weighted avg       0.99      0.99      0.99      3567
```

[56]:
```python
import seaborn as sns

plt.figure(figsize=(15,15))
sns.heatmap(confusion_matrix(labels_array_v, y_pred_classes))
```

[56]:
```
<Axes: >
```


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
100
80
60
40
20
0

[ ]: