

Module 2 Summative Assessment

1. Import libraries

In order to train our model using a decision tree classifier, we will first install libraries in our python notebook. The program will be using such libraries in order to conduct image classification. The sklearn library will be used for applying the decision tree classifier for our model and outputting the model's evaluation metrics. The numpy and pandas libraries will also be used to manipulate our dataset.

```
from tensorflow import keras
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, classification_report, make_scorer
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.tree import DecisionTreeClassifier
import numpy as np
from pathlib import Path
import os.path
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import cv2
```

✓ 5.4s Python

Figure 1 - Imported libraries

2. Linking the dataset folder

Once the libraries have been installed, we can now proceed by linking our dataset. The code, "image_dir = Path('Fish_Dataset/Fish_Dataset')", sets the directory of the folder. This segment would also label each image based on the folder they resided in. If an image was in the folder "Shrimp," its label would be "Shrimp." By doing this process, it organizes all the images based on their labels. The result of this labeling process can be seen in Figure 2.2. Additionally, only 150 fishes per class were used due to consideration of the time and memory it takes to train the model.

```

image_dir = Path('Fish_Dataset/Fish_Dataset')
filepaths = list(image_dir.glob(r'**/*.png'))
labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], filepaths))

filepaths = pd.Series(filepaths, name='Filepath').astype(str)
labels = pd.Series(labels, name='Label')

image_df = pd.concat([filepaths, labels], axis=1)
image_df['Label'] = image_df['Label'].apply(lambda x: np.NaN if x[-2:] == 'GT' else x)
image_df = image_df.dropna(axis=0)

samples = []
for category in image_df['Label'].unique():
    category_slice = image_df.query("Label == @category")
    samples.append(category_slice.sample(150, random_state=1))

image_df = pd.concat(samples, axis=0).sample(frac=1.0, random_state=1).reset_index(drop=True)
image_df

```

✓ 0.4s

Python


Figure 2.1 - Linking dataset folder

	Filepath	Label
0	Fish_Dataset\Fish_Dataset\Red Sea Bream\Red Se...	Red Sea Bream
1	Fish_Dataset\Fish_Dataset\Red Sea Bream\Red Se...	Red Sea Bream
2	Fish_Dataset\Fish_Dataset\Red Mullet\Red Mulle...	Red Mullet
3	Fish_Dataset\Fish_Dataset\Red Mullet\Red Mulle...	Red Mullet
4	Fish_Dataset\Fish_Dataset\Sea Bass\Sea Bass\00...	Sea Bass
...
1345	Fish_Dataset\Fish_Dataset\Red Sea Bream\Red Se...	Red Sea Bream
1346	Fish_Dataset\Fish_Dataset\Shrimp\Shrimp\00930.png	Shrimp
1347	Fish_Dataset\Fish_Dataset\Striped Red Mullet\S...	Striped Red Mullet
1348	Fish_Dataset\Fish_Dataset\Gilt-Head Bream\Gilt...	Gilt-Head Bream
1349	Fish_Dataset\Fish_Dataset\Striped Red Mullet\S...	Striped Red Mullet
1350 rows × 2 columns		

Figure 2.2 - Result of labeling image files

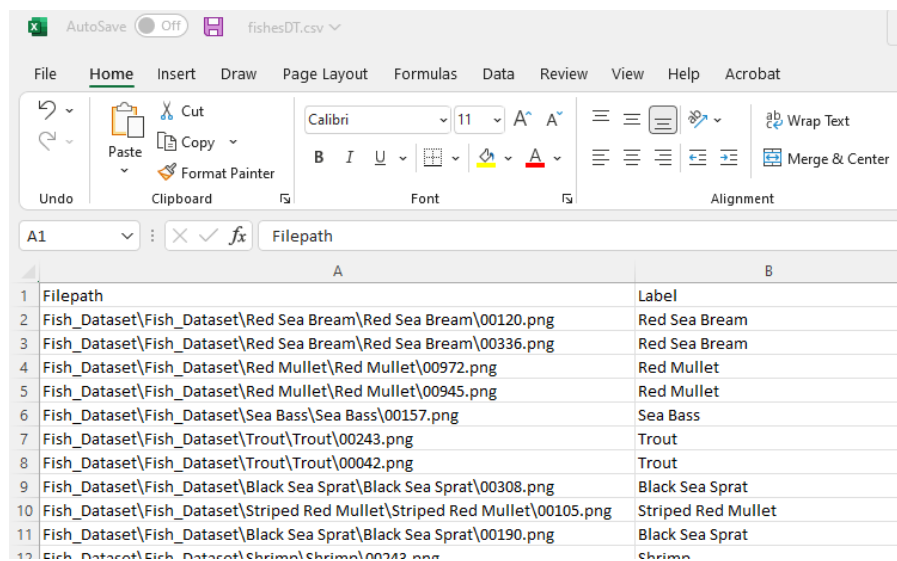
3. Saving file-path/label to .csv

After labeling each image based on its folder, the program will save the data to a .csv file. Using the code: `image_df.to_csv('fishesDT.csv', index=False)`, the file paths of each image are saved in a .csv file. The `index=False` portion of the code removes the row index. In Figure 3.2, the csv file can be opened and viewed by the user.



```
image_df.to_csv('fishesDT.csv', index=False)
```

Figure 3.1 - Result of labeling image files



	A	B
1	Filepath	Label
2	Fish_Dataset\Fish_Dataset\Red Sea Bream\Red Sea Bream\00120.png	Red Sea Bream
3	Fish_Dataset\Fish_Dataset\Red Sea Bream\Red Sea Bream\00336.png	Red Sea Bream
4	Fish_Dataset\Fish_Dataset\Red Mullet\Red Mullet\00972.png	Red Mullet
5	Fish_Dataset\Fish_Dataset\Red Mullet\Red Mullet\00945.png	Red Mullet
6	Fish_Dataset\Fish_Dataset\Sea Bass\Sea Bass\00157.png	Sea Bass
7	Fish_Dataset\Fish_Dataset\Trout\Trout\00243.png	Trout
8	Fish_Dataset\Fish_Dataset\Trout\Trout\00042.png	Trout
9	Fish_Dataset\Fish_Dataset\Black Sea Sprat\Black Sea Sprat\00308.png	Black Sea Sprat
10	Fish_Dataset\Fish_Dataset\Striped Red Mullet\Striped Red Mullet\00105.png	Striped Red Mullet
11	Fish_Dataset\Fish_Dataset\Black Sea Sprat\Black Sea Sprat\00190.png	Black Sea Sprat
12	Fish_Dataset\Fish_Dataset\Shrimp\Shrimp\00343.png	Shrimp

Figure 3.2 - Opening the .csv file

4. Converting images to numpy array

Before we can train our model, we will first need to convert the fish dataset to a numpy array. To do that, we first declare an empty numpy array with a shape of 1350, 445, 590, 3, as shown in Figure 4.1. The number 1350 indicates the rows in the numpy array, while the rest are the dimensions of the images we are dealing with. By converting our images into a numpy array, we are collecting the value of each pixel within the numpy array image.

```
a = np.zeros(shape=(1350,445,590,3))
```

✓ 0.6s Python

Figure 4.1 - Creating Empty NumPY Array

```
count = 0
for i in image_df['Filepath']:
    img_path = i
    img_arr = cv2.imread(img_path)
    a[count] = img_arr
    count += 1

a.shape
```

✓ 17.4s Python

(1350, 445, 590, 3)

Figure 4.2 - Image to NumPY Array

5. Reshape train set images

Now that our dataset images are ready, the next step is to preprocess the images by reshaping them. By reshaping the images, the program is therefore normalizing the dataset. The code in Figure 5 shows the reshaping of the training set images into a 2D array.

```
y_train = image_df['Label'].values
```

✓ 0.7s Python

```
#sklearn expects i/p to be 2d array-model=>reshape to 2d array
nsamples, nx, ny, nrgb = a.shape
X_train = a.reshape((nsamples,nx*ny*nrgb))
```

✓ 0.9s Python

Figure 5 - Reshaping images

6. Define cross-validation method

Using the previously mentioned sklearn python library, we may proceed in writing the cross-validation function. Here, we applied k-fold cross-validation in order to estimate the accuracy, precision, and recall of the model. This step is crucial since it would provide us with results on whether the model we used was effective or not. Figure 6 shows the code for implementing cross-validation.

```
def cross_validation(model, _X, _y, _cv=5):
    _scorers = {
        'accuracy_score': make_scorer(accuracy_score),
        'precision_score': make_scorer(precision_score, average='macro'),
        'recall_score': make_scorer(recall_score, average='macro')
    }
    results = cross_validate(estimator=model,
                             X=_X,
                             y=_y,
                             cv=_cv,
                             scoring=_scorers,
                             return_train_score=True)

    return {"Training Accuracy scores": results['train_accuracy_score'],
            "Mean Training Accuracy": results['train_accuracy_score'].mean()*100,
            "Training Precision scores": results['train_precision_score'],
            "Mean Training Precision": results['train_precision_score'].mean(),
            "Training Recall scores": results['train_recall_score'],
            "Mean Training Recall": results['train_recall_score'].mean(),
            "Validation Accuracy scores": results['test_accuracy_score'],
            "Mean Validation Accuracy": results['test_accuracy_score'].mean()*100,
            "Validation Precision scores": results['test_precision_score'],
            "Mean Validation Precision": results['test_precision_score'].mean(),
            "Validation Recall scores": results['test_recall_score'],
            "Mean Validation Recall": results['test_recall_score'].mean(),
            }
```

✓ 0.7s Python

Figure 6 - Cross validation

7. Decision Tree Model

Once we have prepped the cross-validation segment, we may now start and train the model by using decision tree classification. This decision tree algorithm predicts an outcome by using a tree data structure. In this case, the program is using a CART (Classification And Regression Trees) decision tree algorithm, which is a subtype of a decision tree that uses a Gini impurity index as the splitting criterion. From the first line of code from Figure 7, the criterion was set to "gini." The cross-validation was also set to 5 folds, as seen in the second line of Figure 7. By applying this algorithm, the program is training our model to predict different types of fish from our dataset. However, due to the large amount of data our dataset had, the time it took to finish training our model was 54 minutes.

```
decision_tree_model = DecisionTreeClassifier(criterion="gini", random_state=0)
decision_tree_result = cross_validation(decision_tree_model, X_train, y_train, 5)
```

✓ 54m 17.2s Python

Figure 7 - Decision Tree Model

8. Output Accuracy Results

After training the model using a CART algorithm, the program outputted the results of our model's accuracy, precision, and recall. The training scores and test scores, as well as their means, may be seen in Figure 8.

```
print(decision_tree_result)
```

✓ 0.1s Python

```
{'Training Accuracy scores': array([1., 1., 1., 1., 1.]), 'Mean Training Accuracy': 100.0,
'Training Precision scores': array([1., 1., 1., 1., 1.]), 'Mean Training Precision': 1.0,
'Training Recall scores': array([1., 1., 1., 1., 1.]), 'Mean Training Recall': 1.0,
'Validation Accuracy scores': array([0.40740741, 0.45185185, 0.38518519, 0.4037037 ,
0.43703704]), 'Mean Validation Accuracy': 41.703703703703695, 'Validation Precision
scores': array([0.40501302, 0.45817151, 0.39074708, 0.41652473, 0.43753029]), 'Mean
Validation Precision': 0.4215973238916978, 'Validation Recall scores': array([0.40740741,
0.45185185, 0.38518519, 0.4037037 , 0.43703704]), 'Mean Validation Recall':
0.417037037037037}
```

Figure 8 - Results