

Configuration Manual

MSc Research Project
Programme Name

Niranjan Pramod Desai
Student ID: 20181787

School of Computing
National College of Ireland

Supervisor: Prof. Taimur Hafeez

National College of Ireland
MSc Project Submission Sheet



School of Computing

Student Name: Niranjan Pramod Desai
Student ID: 20181787
Programme: MSc in Data Analytics **Year:** 2021-2022
Module: MSc Research Project
Lecturer: Professor Taimur Hafeez
Submission Due Date: 15-07-2022
Project Title: Classifying different Sea Species using Deep Learning techniques

Word Count: 1061

Page Count: 9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:

Date: 15-07-2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Niranjan Pramod Desai
Student ID: 20181787

1 Introduction

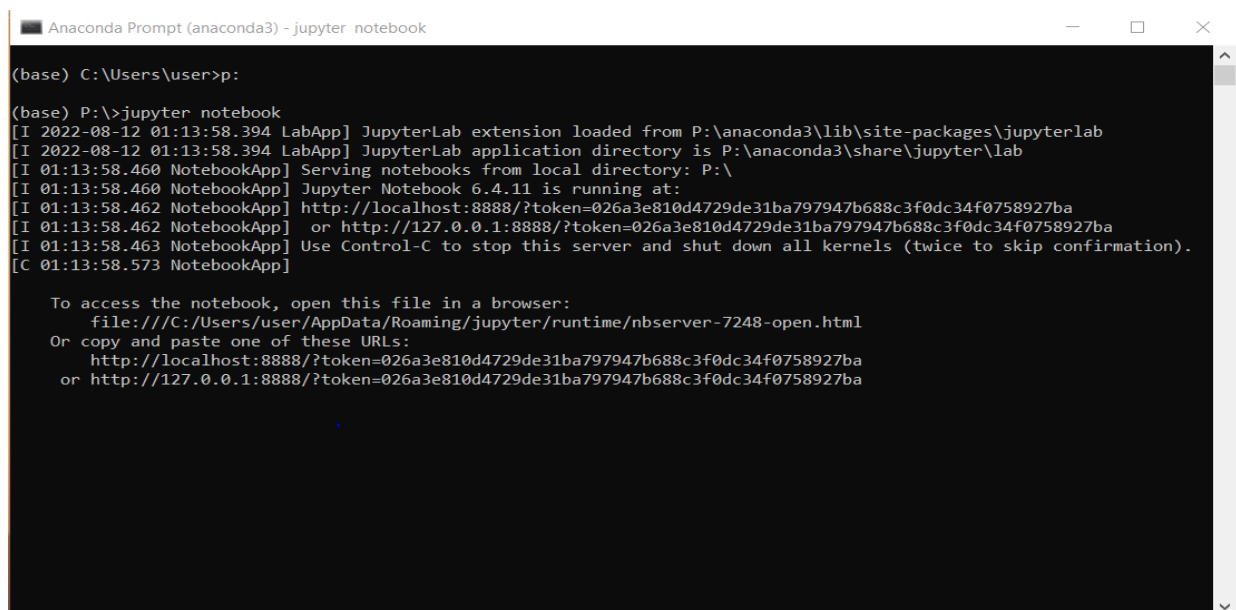
This Manual tells us a thorough explanation of all the procedures that has been followed to verify the research findings that has been done in our research project. This Manual also has an overview of all the process that has been taken for a code execution in addition to covering the various tools needed to mimic the tests.

2 Environment Setup

2.1 Jupyter Notebook

The project entails modelling two distinct networks to assess their suitability for classifying fishes using the available dataset. The model building was accomplished using the Python programming language (version 3.8.3). The Jupyter notebook has been used to run the code. .Ipynb notebooks have been used to submit the finished code. We have also a system of i5 8th gen and 8 gb ram to train our model.

1. From Anaconda.Navigator we have downloaded and install jupyter notebook.
2. From Anaconda prompt we have loaded jupyter notebook.



```
Anaconda Prompt (anaconda3) - jupyter notebook
(base) C:\Users\user>p:
(base) P:\>jupyter notebook
[I 2022-08-12 01:13:58.394 LabApp] JupyterLab extension loaded from P:\anaconda3\lib\site-packages\jupyterlab
[I 2022-08-12 01:13:58.394 LabApp] JupyterLab application directory is P:\anaconda3\share\jupyter\lab
[I 01:13:58.460 NotebookApp] Serving notebooks from local directory: P:\
[I 01:13:58.460 NotebookApp] Jupyter Notebook 6.4.11 is running at:
[I 01:13:58.462 NotebookApp] http://localhost:8888/?token=026a3e810d4729de31ba797947b688c3f0dc34f0758927ba
[I 01:13:58.462 NotebookApp] or http://127.0.0.1:8888/?token=026a3e810d4729de31ba797947b688c3f0dc34f0758927ba
[I 01:13:58.463 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 01:13:58.573 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/user/AppData/Roaming/jupyter/runtime/nbserver-7248-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=026a3e810d4729de31ba797947b688c3f0dc34f0758927ba
or http://127.0.0.1:8888/?token=026a3e810d4729de31ba797947b688c3f0dc34f0758927ba
```

Figure 1: Loading Jupyter notebook from anaconda Prompt.

3. It will load the page on your default browser. On the right side you have to click on New > Python 3 (ipykernel).
4. The code may then be written, and each cell in the notebook can be run by selecting the Run cell button located directly to the left of each cell.

3 Data Preparation

The dataset we have use here is "A LARGE-SCALE FISH DATASET" from Kaggle, which comprises 9 different sea food species gathered from a supermarket in Turkey as part of a university-industry partnership study at Izmir University of Economics. This dataset contain 2 folders one with augmented data containing 18000 images and second with original data containing 430 images in total. We are going to use the original data containing 430 images in total. The collection includes image samples from trout, gilt head bream, horse mackerel, red sea bream, sea bass, black sea sprat, red mullet, striped red mullet, and shrimp. The dataset includes nine distinct varieties of seafood.

4 Execution of Code

4.1 importing the libraries.

1. Operating system along with cv2 was imported.
2. Similarly numpy ,seaborn , from matplotlib cm and pyplot were imported .
3. From tensorflow keras models ,layers ; application such as VGG16 , MobileNetV2 were imported. The code to import the libraries; are in Figure 2.

```
import os
import cv2
import numpy as np

import seaborn as sns
import matplotlib.cm as cm
import matplotlib.pyplot as plt

from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.callbacks import EarlyStopping

from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
```

Figure 2: Importing the libraries.

4.2 Importing the dataset

As we have downloaded the dataset in p drive. We are importing the dataset from p drive by specifying the path and setting the image height along with width as 224 by 224 , the models are implemented on 10 epochs and the batch size is set to 32 so that we can have least error rate. The code to import dataset is in Figure 3.

```

from pathlib import Path
dataDir = Path ("P:\nci thesis\NA_Fish_Dataset")
img_height, img_width = 224, 224
batch_size = 32
numEpochs = 10

```

Figure 3:Importing the dataset.

4.3 Augmentation

As the dataset is less we are augmenting the images for artificially expanding dataset by applying changes like rescaling, shearing , zooming , rotation and flipping to our source photographs. Here, we have use ImageDataGenerator function for splitting the dataset in the ratio of 80:20 for training and validation. The code to be performed the augmentation techniques is shown in Figure 4.

```

datagen = ImageDataGenerator(rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    rotation_range=15,
    validation_split=0.2,) # set validation split

train_generator = datagen.flow_from_directory(
    dataDir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    shuffle=True,
    class_mode='categorical',
    subset='training') # set as training data

validation_generator = datagen.flow_from_directory(
    dataDir, # same directory as training data
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True,
    subset='validation') # set as validation data

```

Figure 4: Code for performing Data Augmenting and Splitting the dataset.

4.4 Loading and tuning the model

4.4.1 For Mobilenet V2

```

## Load model
pretrained_model = MobileNetV2(
    input_shape=(img_height, img_width, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg',)
pretrained_model.summary()
# Freeze layer
pretrained_model.trainable = False
pretrained_model.summary()

```

Figure 5: Loading Model Mobilenet_V2

We have loaded the pretrained MOBILENET_V2 by keeping pooling layer as an average as in Figure 5. And finally freezing all the trainable parameters by keeping the value as false, so that when the tuning is done , the old parameters does not get updated. For tuning we are adding 3 layers, of 128 nodes in 1st and 2nd layer by keeping activation function as relu and

adding a 3rd final sigmoid layer for classification for 9 classes by keeping activation function as SoftMax as shown in Figure 6. By using summary to check the architecture of the model.

```

## Fine Tune added layer
inputs = pretrained_model.input
x = layers.Dense(128, activation='relu')(pretrained_model.output)
x = layers.Dense(128, activation='relu')(x)
outputs = layers.Dense(train_generator.num_classes, activation='softmax')(x)
mobileNetmodel = Model(inputs=inputs, outputs=outputs)
print(mobileNetmodel.summary())

```

Figure 6: Tuning the Model Mobilenet_V2

4.4.2 For VGG16

```

## Load model
base_model = VGG16(input_shape = (img_height, img_width, 3), # Shape of our images
include_top = False, # Leave out the last fully connected layer
weights = 'imagenet',
)
print(base_model.summary())
# Freeze Layer
base_model.trainable = False

```

Figure 7: Loading the Model VGG16.

Here, we are loading the pretrained vgg16 as shown in Figure 7. And finally freezing all the trainable parameters by keeping the value as false, so that when the tuning is done, the old parameters does not get updated. For tuning we are adding a fully connected layer with 512 hidden units and activation function as relu, setting dropout as 50% so that the model does not get overfit and adding a final sigmoid layer for classification for 9 classes by keeping activation function as SoftMax as shown in Figure 8. By using summary to check the architecture of the model.

```

x = layers.Flatten()(base_model.output)

# Add a fully connected layer with 512 hidden units and ReLU activation
x = layers.Dense(512, activation='relu')(x)

# Add a dropout rate of 0.5
x = layers.Dropout(0.5)(x)

# Add a final sigmoid layer for classification
x = layers.Dense(train_generator.num_classes, activation='softmax')(x)

vggModel = Model(base_model.input, x)
print(vggModel.summary())

```

Figure 8: Tuning the Model VGG16.

4.5 Compiling the model

The loss function used for both the models is categorical cross-entropy and both the models are compiled using adam optimizer as shown in Figure 9.

```

## Mobilenet Compiling
mobileNetmodel.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

## VGG compiling
vggModel.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

```

Figure 9: Compiling the Model MOBILENET_V2 and VGG16.

4.6 Training both the models

```

print("Training Mobilenet")

## Training mobilenet
MobileNet_history = mobileNetmodel.fit(
    train_generator,
    validation_data = validation_generator,
    epochs = numEpochs,
    callbacks=[
        EarlyStopping(
            monitor='val_loss',
            patience=2,
            restore_best_weights=True
        )
    ]
)

print("\n\n")

print("Training VGG")
## Training VGG
VGG_history = vggModel.fit(
    train_generator,
    validation_data = validation_generator,
    epochs = numEpochs,
    callbacks=[
        EarlyStopping(
            monitor='val_loss',
            patience=2,
            restore_best_weights=True
        )
    ]
)

```

Figure 10: Training the Model MOBILENET_V2 and VGG16.

In this step , we are training both the models using the code shown in Figure 10 and plotting the model for accuracy and loss of MOBILENETV2 and VGG16 using the code shown in Figure 11.


```

plt.plot(MobileNet_history.history["accuracy"])
plt.plot(MobileNet_history.history["val_accuracy"])
plt.title("Mobilenet Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Apoch")
plt.legend(["train", "validation"], loc="upper left")
plt.show()

plt.plot(MobileNet_history.history["loss"])
plt.plot(MobileNet_history.history["val_loss"])
plt.title("Mobilenet Model Loss")
plt.ylabel("Loss")
plt.xlabel("epoch")
plt.legend(["train", "validation"], loc="upper left")
plt.show()

plt.plot(VGG_history.history["accuracy"])
plt.plot(VGG_history.history["val_accuracy"])
plt.title("VGG Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Apoch")
plt.legend(["train", "validation"], loc="upper left")
plt.show()

plt.plot(VGG_history.history["loss"])
plt.plot(VGG_history.history["val_loss"])
plt.title("VGG Model Loss")
plt.ylabel("Loss")
plt.xlabel("epoch")
plt.legend(["train", "validation"], loc="upper left")
plt.show()

```

Figure 10: Plotting the model MOBILENETV2 and VGG16(accuracy and loss)

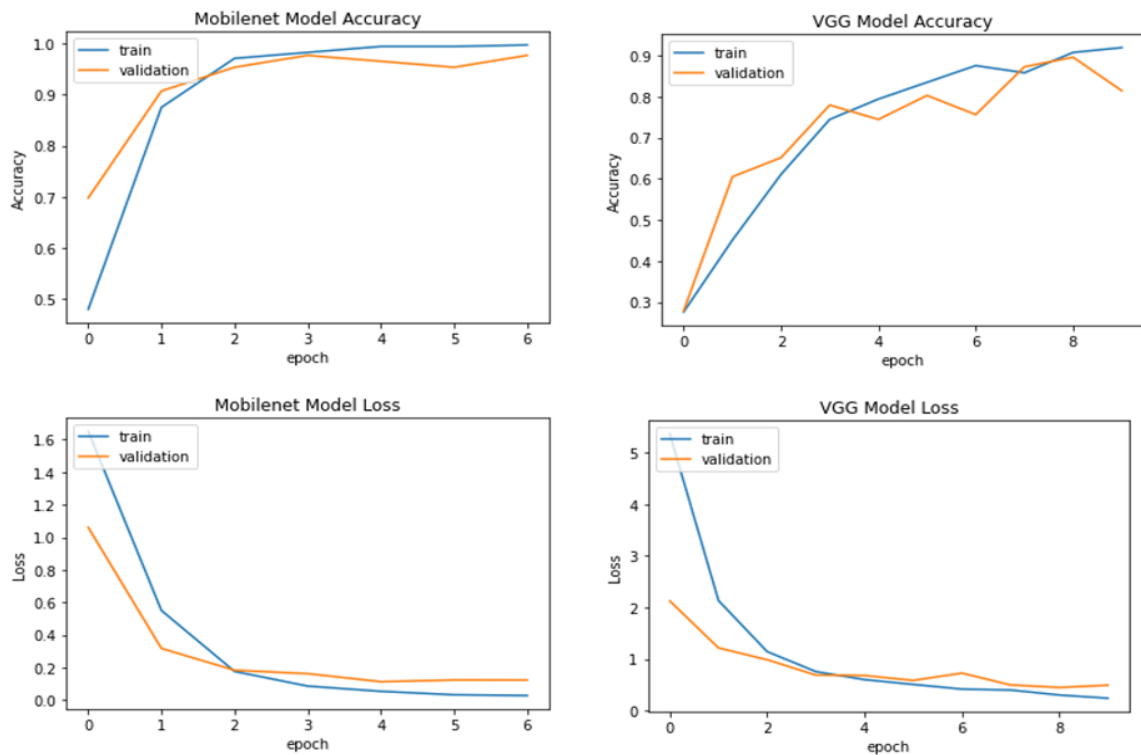


Figure 11: Output of model MOBILENETV2 and VGG16(accuracy and loss)

4.7 Comparing both the models

Here we are comparing validation accuracy of both the models by using the code shown in the Figure 12.

```
plt.figure(figsize=(15, 15))
plt.plot(MobileNet_history.history["val_accuracy"])
plt.plot(VGG_history.history["val_accuracy"])
plt.title("Model Accuracy Mobilent vs VGG")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Mobilenet", "VGG"], loc="upper left")
plt.show()
```

Figure 11: Comparing the validation accuracy for model MOBILENETV2 and VGG16

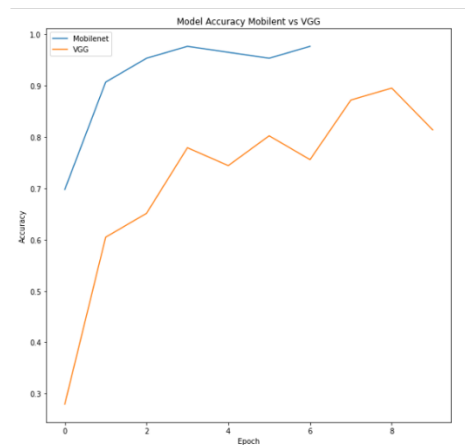


Figure 12: Output comparison for model MOBILENETV2 and VGG16(validation accuracy).

4.8 Printing the accuracy

Printing the accuracy as shown in Figure 13.

```
results = mobileNetModel.evaluate(validation_generator, verbose=0)
print("Mobiletnet Test Loss: {:.5f}".format(results[0]))
print("Mobiletnet Test Accuracy: {:.2f}%".format(results[1] * 100))

print("\n\n")
results = vggModel.evaluate(validation_generator, verbose=0)
print("VGG Test Loss: {:.5f}".format(results[0]))
print("VGG Test Accuracy: {:.2f}%".format(results[1] * 100))
```

Figure 13: Printing the Accuracy for model MOBILENETV2 and VGG16.

4.9 Testing the model/ Model selection

Model Comparison	Training		Validation		Testing	
	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
Mobilenet_v2	0.0173	0.9971	0.0742	0.9767	0.07362	98.84%
vgg16	0.3524	0.8837	0.4862	0.8605	0.41876	87.21%

4.10 Model deployment

In this stage, we'll show 25 images of the dataset's expected images together with their labels. To implement, we will generate figures and axis of the pictures that will be called by `plt.subplots` with the parameters like number of rows will be 5, number of columns will be 5, and figure size would be (25,25). We will use both model just to know how MOBILENET V2 predicts better than VGG16 model. This is the outcome of model deployment in Figure 16,

```
classes = validation_generator.classes
plt.figure(figsize=(25, 30))
for i, image_name in enumerate(validation_generator_filenames[:30]):
    # print(image_name)
    image = cv2.resize(
        cv2.cvtColor(
            cv2.imread(os.path.join(dataDir, image_name)),
            cv2.COLOR_BGR2RGB),
        (img_width, img_height))
    blob = image.reshape(1, img_height, img_width, 3) / 255.0

    preds1 = mobileNetModel.predict(blob)
    biggest_pred_index1 = np.array(preds1)[0].argmax()

    preds2 = vggModel.predict(blob)
    biggest_pred_index2 = np.array(preds2)[0].argmax()

    ax = plt.subplot(6, 5, i + 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(f"Mobilenet predicted: {biggest_pred_index1}, \nVGG Predicted: {biggest_pred_index2}, \n")
    plt.axis("off")
```

Figure 15: Code for plotting and classifying the images with the model predicted labels.

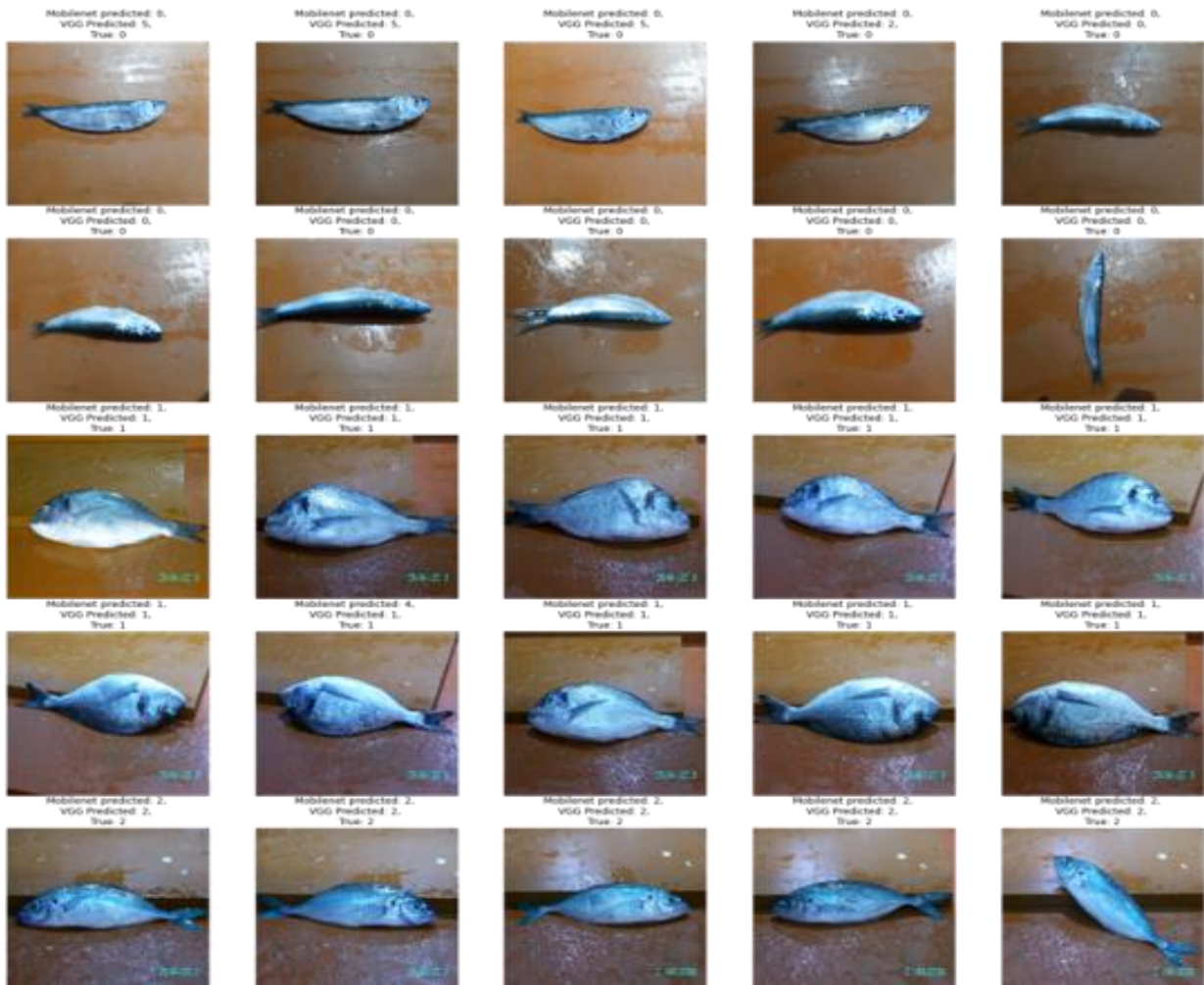


Figure 16: Output for different species classification.