Configuration Manual

MSc Research Project
Data Analytics

Pravin Harish Sharma
Student ID: x22214224

School of Computing
National College of Ireland

Supervisor: Prof. Barry Haycock

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Pravin Harish Sharma		
Student ID:	x22214224		
Programme:	Data Analytics	**Year:**	2023-2024
Module:	MSc Research Project		
Lecturer:	Prof. Barry Haycock		
Submission Due Date:	12-08-2024		
Project Title:	A Deep learning approach for chicken disease detection using images of droppings		
Word Count:	1900	**Page Count: 23**	

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: PS

Date: 12-08-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
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.	□

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

Pravin Harish Sharma
Student ID: x22214224

1 Introduction

The research intents to build a deep learning pipeline for identifying sick chickens using their faecal images. The object detection model will use the latest YoloV10s and for image classification ensemble of light weight models – MobileNetV3, EfficientNetV2B2, and NasNetMobile. The output will have the bounding boxes along with the disease class predicted on the images. The technical setup and code samples from each module will be stated in this configuration manual.

2 System Configuration and Setup

Two environments were used for the development of the project. Local setup for sample scripts development/testing and data cleaning. Google Colab Pro was used training of models on different parameters.

2.1 Local Setup

We need to utilize the GPU of the laptop for training the model for lesser epochs, and for more epochs google colab will be used.

Setting Up Jupyter notebook in the windows laptop to utilize the GPU power:

1. Install Anaconda Navigator
2. Open Anaconda Prompt in Admin Mode and run following commands:
    a) Create an environment with python version 3.9

    conda create -n py39 python=3.9

    b) Activate the environment

    conda activate py39

    c) Install cudatoolkit and cudnn

    conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0

    d) Install tensorflow

    python -m pip install tensorflow==2.10

    e) Install jupyter notebook

3. Now open jupyter notebook and create a new notebook and run the below command as shown in Figure 1 and 2, to check if GPU is accessible by jupyter notebook. Figure 3 shows the local system configuration.

```
[2]: import tensorflow as tf

     print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

     Num GPUs Available:  1
```

Figure 1: Check Number of GPU available

```
[3]: !nvidia-smi

     Fri Jun 14 19:09:01 2024
     +-----------------------------------------------------------------------------------------+
     | NVIDIA-SMI 555.99                 Driver Version: 555.99         CUDA Version: 12.5     |
     |-----------------------------------------+------------------------+----------------------+
     | GPU  Name                  Driver-Model | Bus-Id          Disp.A | Volatile Uncorr. ECC |
     | Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
     |                                         |                        |               MIG M. |
     |=========================================+========================+======================|
     |   0  NVIDIA GeForce RTX 3050 ...  WDDM  |   00000000:01:00.0 Off |                  N/A |
     | N/A   51C    P0             13W /   50W |       0MiB /   4096MiB |      0%      Default |
     |                                         |                        |                  N/A |
     +-----------------------------------------+------------------------+----------------------+

     +-----------------------------------------------------------------------------------------+
     | Processes:                                                                              |
     |  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
     |        ID   ID                                                               Usage      |
     |=========================================================================================|
     |  No running processes found                                                             |
     +-----------------------------------------------------------------------------------------+
```

Figure 2: Check GPU configuration

```
System: Windows
Machine: AMD64
Processor: AMD64 Family 25 Model 80 Stepping 0, AuthenticAMD
Physical cores: 8
Total cores: 16
Total memory: 16541605888
Total disk space: 510455517184
Python Version: 3.9.19
```

Figure 3: Local System Configuration

4. The local system configuration:
   4GB GPU (RTX 3050), 16GB RAM, 8 physical cores, Windows. Python 3.9.19

2.2 Google Colab Pro

Googl colab pro was used for running models for higher epochs. The configuration of the machine is shown in the Figure 4 and 5. It consists of 16GB GPU (T4-Tesla), 51GB RAM, 4 physical cores, Linux and Python 3.10.12

```
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   63C    P8              10W /  70W |      3MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Figure 4: Google Colab GPU Configuration

```
System: Linux
Machine: x86_64
Processor: x86_64
Physical cores: 4
Total cores: 8
Total memory: 54754004992
Total disk space: 216063848448
Python Version: 3.10.12
```

Figure 5: Google Colab Overall Configuration

3 Selection of the Dataset

The datasets for this research are acquired from Zenodo opensource dataset library. The data is collected from Arusha and Kilimanjaro regions which are situated in Tanzania.
The 2-dataset used in this research are the following:

1. Manually labelled Dataset[1]
2. Lab labelled Dataset[2]

The Images in the dataset are the fecal images of chickens and are divided into following categories: Coccidiosis(Cocci) Disease, Healthy, Newcastle(NCD) Disease and Salmonella(Salmo) Disease.The manually labelled dataset have total of 6812 images which also have bounding box annotation and lab labelled dataset have 1,255 images but these images do not have bounding box annotation. The lab images were collected along with the fecal sample for running tests on them. All the images do not have a fix resolution or alignment.

4 Exploratory Data Analysis

4.1 Sample Images and Class Distribution (dataset/0_raw_data)

4.1.1 Farm Labelled Images

path: root/dataset/0_raw_data/ zenodo-Machine Learning Dataset for Poultry Diseases Diagnostics)
Each class had a separate folder. Figure 6 shows sample images from the farm labelled raw dataset from each of the classes. Figure 7 shows distribution of images across the classes in farm labelled raw dataset.

[1] https://zenodo.org/records/4628934
[2] https://zenodo.org/records/5801834

Figure 6: Sample Images from Farm labelled Raw dataset

```
Class "cocci" has 2103 images, which is 30.87% of the total.
Class "healthy" has 2057 images, which is 30.20% of the total.
Class "ncd" has 376 images, which is 5.52% of the total.
Class "salmo" has 2276 images, which is 33.41% of the total.
```

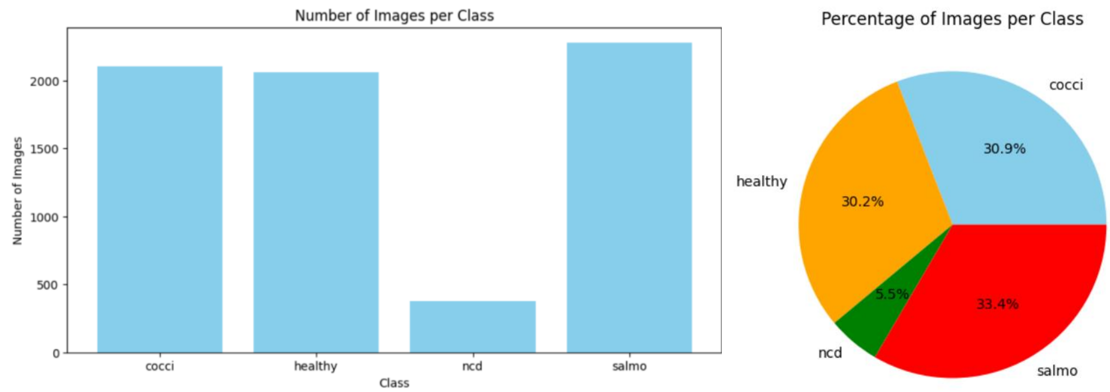


Figure 7: Farm Labelled raw dataset image distribution

4.1.2 Lab Labelled Images

(path: root/dataset/0_raw_data/ zenodo-Machine Learning Dataset for Poultry Diseases Diagnostics - PCR annotated)

Each class had a separate folder. Figure 8 shows sample images from the lab labelled raw dataset from each of the classes. Figure 9 shows distribution of images across the classes in lab labelled raw dataset.

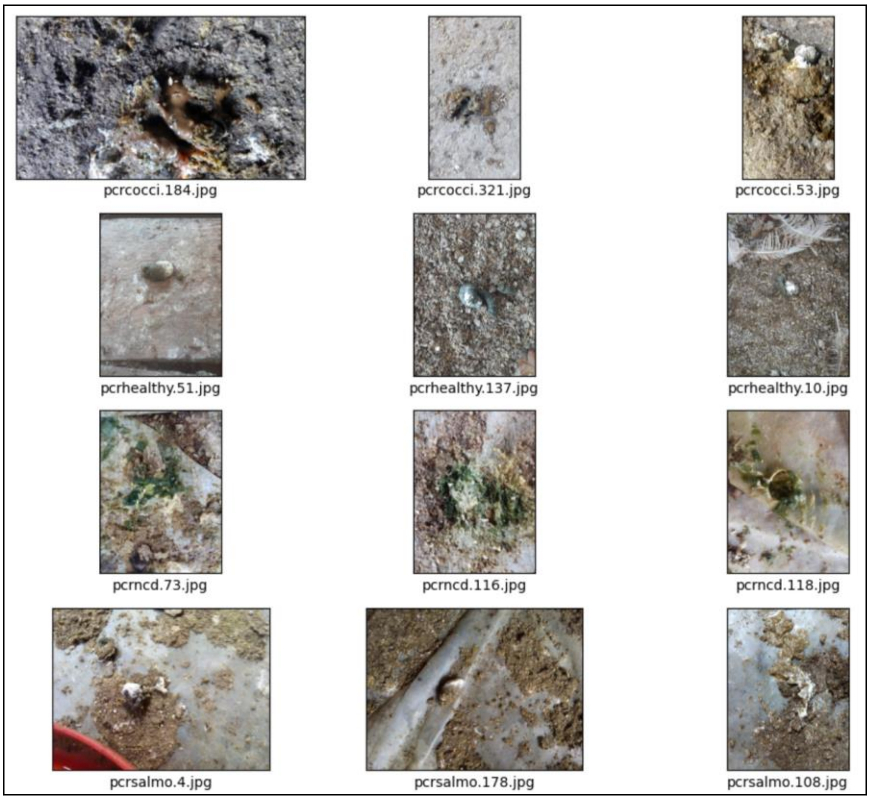


Figure 8: Sample Images from Lab labelled Raw dataset

```
Class "pcrcocci" has 373 images, which is 29.72% of the total.
Class "pcrhealthy" has 347 images, which is 27.65% of the total
Class "pcrncd" has 186 images, which is 14.82% of the total.
Class "pcrsalmo" has 349 images, which is 27.81% of the total.
```

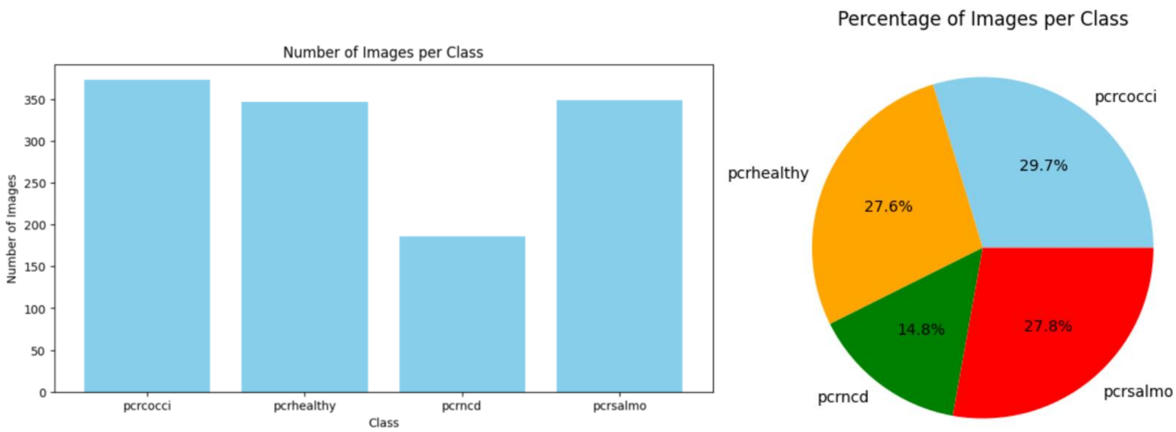


Figure 9: Lab Labelled raw dataset image distribution

4.2 Display bounding box on raw images using labelImg app

- Only farm labelled dataset have Bounding box annotation provided by Author of the dataset.
- A folder “imgObjDect_Yolo” located at “root/dataset/0_raw_data/ zenodo-Machine Learning Dataset for Poultry Diseases Diagnostics” contents all the bounding boxes annotations.

- Farm labelled dataset images and bounding boxes labels of all the classes were copied together in the same folder
  (path: root/dataset/1_Object_detection/ 3_labelimg_working_farm_labelled/images)
- Install labelImg app: pip install labelimg
- Inside command prompt type this command to open a GUI: labelimg
- Inside the APP go to the directory where images are stored along with the bounding boxes labels. Figure 10, 11, 12 and 13 show different classes of poultry fecal images with bounding boxes annotation.
  (path: root/dataset/1_Object_detection/ 3_labelimg_working_farm_labelled/images)

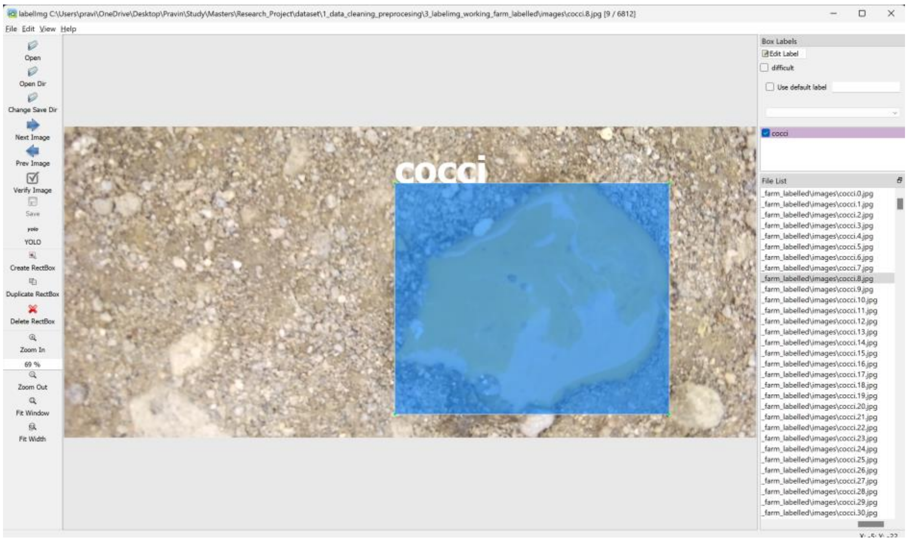


Figure 10: Cocci disease bounding box annotated

Figure 11: Healthy poultry bounding box annotated

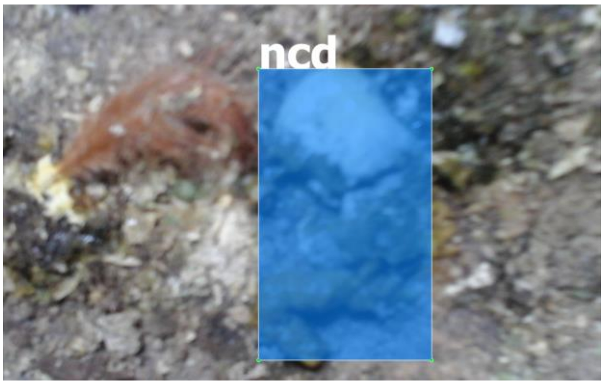


Figure 12: NCD infected poultry fecal image annotated

Figure 13: Salmo disease bounding box annotated

4.3 EDA Findings

After analysing the images from both the dataset, we can conclude that:

1. Total number of Farm labelled, and lab labelled images are **6,812** and **1,255**, respectively. Accounting to total **8,067** images.
2. Image sizes are varying.
3. **Bounding boxes** are **missing** from the Lab labelled images.
4. The folder structure required for Yolov10 is different than the existing structure.
5. **Class imbalance** is significant.

5 Data Cleaning and Preparation

5.1 Automated bounding boxes labelling

- Farm labelled Image resized to 640x640 pixel.
  (code path: code/1_Object_detection/2_resizing_bounding_box)
- Then split into train, valid and test sets.
  (dataset path: root/dataset/1_Object_detection/4_resized_farm_labelled_640)

- Feed this data to Yolov10-n for object detection and trained for 50 epochs, I got a model which predicts the bounding boxes with 79% accuracy.
(code path: root/code/1_Object_detection/3_experiment_two)

- Used this model to predict the bounding boxes on lab labelled dataset.
(model path: code/1_Object_detection/3_experiment_two/yolov10/runs/detect/train3/weights/best.pt)
- This is how we automated most of the annotation work for lab labelled images. After that I manually checked the boxes and corrected them wherever required.
(path: dataset\1_Object_detection\5_pcr_images_annotated_manually)
- Merging Farm labelled images+labels with Lab labelled image+labels and resizing to **640x640** pixels.
(**code**\1_Object_detection**6_merging_two_datasets**)
(**dataset**\1_Object_detection**8_resized_640**)
- Now we have all the 8,067 images along with their bounding boxes annotation. (**dataset** path: dataset\1_Object_detection**8_resized_640**)

5.2 Class Imbalance Mitigation

- Checking the class distribution on merged dataset (dataset path: **dataset**\1_Object_detection**8_resized_640**) as shown in figure 14.

```
Class "cocci" has 2476 images, which is 30.69% of the total.
Class "healthy" has 2404 images, which is 29.80% of the total.
Class "ncd" has 562 images, which is 6.97% of the total.
Class "salmo" has 2625 images, which is 32.54% of the total.
```

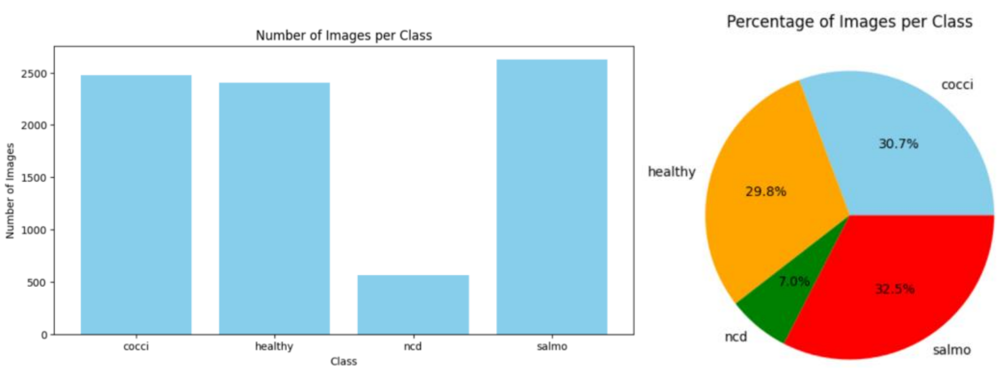


Figure 14: Merged Dataset Class Distribution

- **As NCD has low sample image (562 images**) while other classes have 2000+ images, I will oversample the images for NCD class by image augmentation techniques and make sure the augmented images have bounding boxes intact, so that I won't have to label them again.
- Each will be **augmented 5 times**, so that total number of images for **NCD class will be 2810**. As shown in the Figure 15, Albumentations library is used here for augmentation and keeping the bounding boxes intact.
(**code** path:code/1_Object_Detection/**7_class_imbalance**/2_0_script_multiple_ncd.py)

```
augmentation = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.RandomRotate90(p=0.5),
    A.ShiftScaleRotate(p=0.5, shift_limit=0.1, scale_limit=0.1, rotate_limit=45),
    A.RandomBrightnessContrast(p=0.5),
    A.RandomSizedBBoxSafeCrop(height=640, width=640, p=0.5)
], bbox_params=A.BboxParams(format='pascal_voc', label_fields=['labels']))
```

Figure 15: Augmentation for increasing sample while keep bounding boxes intact

- Now number of total images of NCD = 2810, other classes have total: 'cocci': 2476 images, 'healthy': 2404 images, 'salmo': 2625 images. Now we will increase the number of augmented images for other classes while having bounding boxes intact, so that each class we end up having 2810 images each.
  (**code** path: code/1_Object_Detection/**7_class_imbalance**/2_1_script_multiple_other_class.py)
- Class distribution after augmentation is shown in the Figure 16.

```
Class "cocci" has 2810 images, which is 25.00% of the total.
Class "healthy" has 2810 images, which is 25.00% of the total.
Class "ncd" has 2810 images, which is 25.00% of the total.
Class "salmo" has 2810 images, which is 25.00% of the total.
```

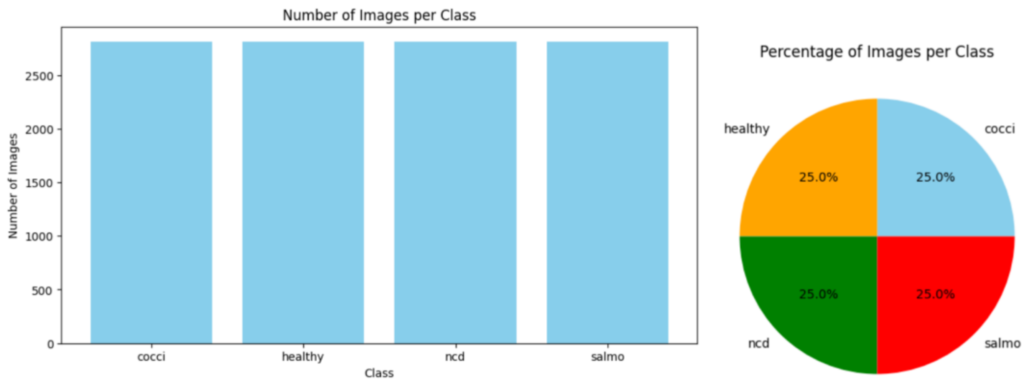


Figure 16: Class Distribution after augmentation

5.3 Folder structure and Datasplit

Data was split and arranged in specific folder structure according to different models' requirement. The Code is stored in the public github repository. https://github.com/pravin-sharma/thesis-poultry-disease-detection-and-classification-deep-learning.git

5.3.1 Object detection

For Object detection, the data should be in the format (structure is compulsory for Yolo) as shown in the figure 17. The data is splitted into train, test and valid set using code: code\1_Object_detection\8_data_split

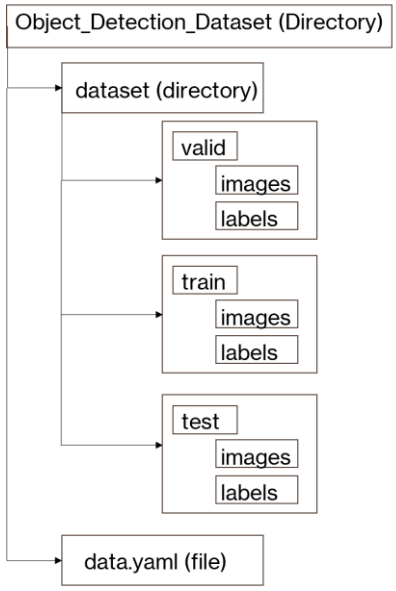


Figure 17: Dataset folder structure for Object Detection

5.3.2 Image Classification

For Image Classification, we have structured the data in the following format (opinionated). Here we don't need image labels as we require in object detection. We have manually created directory for each class and moved the images into them.

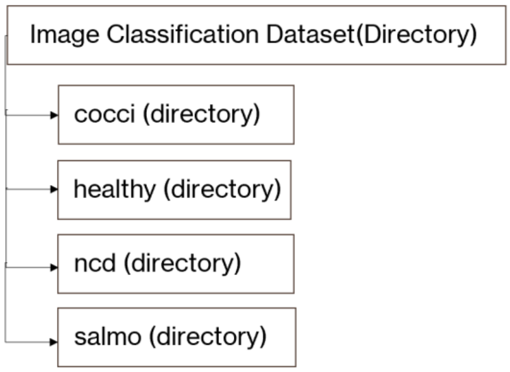


Figure 18: Dataset folder structure for Image Classification

5.4 Data Processing during Image Classification training process:

5.4.1 Data Split

The dataset was split into three sets: Train, Validation and Test split in ratio 70,20 and 10. Figure 19 shows the code for split for image classification.

```python
def split_data(path, seed, test_size=0.1, val_size=0.2):
    all_files = []
    all_labels = []

    # Iterate through directory and collect file paths and labels
    for class_dir in os.listdir(path):
        class_path = os.path.join(path, class_dir)
        if os.path.isdir(class_path):
            for img in os.listdir(class_path):
                img_path = os.path.join(class_path, img)
                all_files.append(img_path)
                all_labels.append(class_dir)

    # Convert labels to numeric format
    label_to_index = {label: index for index, label in enumerate(np.unique(all_labels))}
    all_labels = [label_to_index[label] for label in all_labels]

    # Split into train+val and test
    train_val_files, test_files, train_val_labels, test_labels = train_test_split(
        all_files, all_labels, test_size=test_size, random_state=seed, stratify=all_labels
    )

    # Split train+val into train and validation sets
    train_files, val_files, train_labels, val_labels = train_test_split(
        train_val_files, train_val_labels, test_size=val_size, random_state=seed, stratify=train_val_labels
    )

    return (train_files, train_labels), (val_files, val_labels), (test_files, test_labels)
```

Figure 19: Code for data split - Image classification for training Individual model

5.4.2 Normalization

Figure 20 shows code for normalizing the image pixels. Normalized pixel value between the range 0 to 1.

```python
def load_image(file_path, imgsz, clr):
    image = tf.io.read_file(file_path)
    image = tf.image.decode_jpeg(image, channels=3 if clr == "rgb" else 1)
    image = tf.image.resize(image, imgsz)
    image = tf.cast(image, tf.float32) / 255.0
    return image
```

Figure 20: Normalization

5.4.3 Augmentation:

Using on-the-fly augmentation of tensorflow, so each epoch will see varied images and the model will learn to generalize better. Different transfer learning model have different input size, MobileNetv3Large and NasnetMobile expect 224x224 image size, while efficientNetV2B2 expects 260x260 image size. Refer figure 21 for code used for augmentation while training individual image classification model.

```python
def augment_img(img_, lbl_):
    image = tf.image.random_flip_left_right(img_)
    image = tf.image.central_crop(image, 0.85)
    image = tf.image.resize(image, IMAGE_SIZE)
    image = tf.image.random_brightness(image, 0.2)
    image = tf.image.random_contrast(image, 0.5, 2.0)
    return tf.cast(image, tf.float32), lbl_

train_batch = train_batch.map(augment_img, tf.data.AUTOTUNE)
train_batch = train_batch.cache().prefetch(buffer_size = tf.data.AUTOTUNE)
val_batch = val_batch.cache().prefetch(buffer_size = tf.data.AUTOTUNE)
test_batch = test_batch.cache().prefetch(buffer_size = tf.data.AUTOTUNE)
```

Figure 21: Augmentation

6 Importing necessary Libraries

6.1 Object detection: Setup and All the libraries used

Following code was used for setting up Object detection.

```
!git clone https://github.com/THU-MIG/yolov10.git
```

```
# # Google Colab Specific commands
from google.colab import drive
drive.mount('/content/drive')
# # # Example: Unzip into the current working directory - since i have a zip in my drive and i want to unzip into my colab env for faster access
!unzip -q "/content/drive/MyDrive/Colab Notebooks/pravin_thesis/0_dataset/10_balanced_data_split.zip" -d '/content/yolov10/datasets/'
```

```
cd yolov10
```

```
!pip install .
```

```
import os
import urllib.request

# Create a directory for the weights in the current working directory
weights_dir = os.path.join(os.getcwd(), "weights")
os.makedirs(weights_dir, exist_ok=True)

# URLs of the weight files
# Trying Yolov10-S
urls = [
    # "https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10n.pt",
    "https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10s.pt"
]

# Download each file
for url in urls:
    file_name = os.path.join(weights_dir, os.path.basename(url))
    urllib.request.urlretrieve(url, file_name)
    print(f"Downloaded {file_name}")
```

6.2 Image classification: All the libraries used

```python
# Standard Library Imports
import os
import random
import shutil
import warnings
# Data Analysis and Manipulation
import pandas as pd
import numpy as np
# Image Processing
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import rasterio
from PIL import Image
import PIL
# Visualization
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use("fivethirtyeight")
# Machine Learning and Deep Learning
import tensorflow as tf
from tensorflow import keras
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from keras.utils import image_dataset_from_directory
from keras.optimizers import AdamW
import keras_cv
keras.mixed_precision.set_global_policy("mixed_float16")
# Progress Bars
import tqdm
from tqdm.auto import trange, tqdm
# Evaluation Metrics
import sklearn
from sklearn.metrics import precision_score, accuracy_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.utils import class_weight
```

7 Model Architecture

7.1 Object Detection

7.1.1 Yolov10-n

```
                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1      9856  ultralytics.nn.modules.block.SCDown          [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1     36096  ultralytics.nn.modules.block.SCDown          [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                  -1  1    249728  ultralytics.nn.modules.block.PSA             [256, 256]
 11                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 12             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 13                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 14                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 15             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 16                  -1  1     37248  ultralytics.nn.modules.block.C2f             [192, 64, 1]
 17                  -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3, 2]
 18            [-1, 13]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 19                  -1  1    123648  ultralytics.nn.modules.block.C2f             [192, 128, 1]
 20                  -1  1     18048  ultralytics.nn.modules.block.SCDown          [128, 128, 3, 2]
 21            [-1, 10]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 22                  -1  1    282624  ultralytics.nn.modules.block.C2fCIB          [384, 256, 1, True, True]
 23        [16, 19, 22]  1    862888  ultralytics.nn.modules.head.v10Detect        [4, [64, 128, 256]]
YOLOv10n summary: 385 layers, 2708600 parameters, 2708584 gradients, 8.4 GFLOPs
```

Total 2.7M parameters in Yolov10-n.

7.1.2 Yolov10-s

```
                   from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                  -1  1     29056  ultralytics.nn.modules.block.C2f             [64, 64, 1, True]
  3                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  4                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  5                  -1  1     36096  ultralytics.nn.modules.block.SCDown          [128, 256, 3, 2]
  6                  -1  2    788480  ultralytics.nn.modules.block.C2f             [256, 256, 2, True]
  7                  -1  1    137728  ultralytics.nn.modules.block.SCDown          [256, 512, 3, 2]
  8                  -1  1    958464  ultralytics.nn.modules.block.C2fCIB          [512, 512, 1, True, True]
  9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                  -1  1    990976  ultralytics.nn.modules.block.PSA             [512, 512]
 11                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 12             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 13                  -1  1    591360  ultralytics.nn.modules.block.C2f             [768, 256, 1]
 14                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 15             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 16                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 17                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 18            [-1, 13]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 19                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 20                  -1  1     68864  ultralytics.nn.modules.block.SCDown          [256, 256, 3, 2]
 21            [-1, 10]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 22                  -1  1   1089536  ultralytics.nn.modules.block.C2fCIB          [768, 512, 1, True, True]
 23        [16, 19, 22]  1   1641896  ultralytics.nn.modules.head.v10Detect        [4, [128, 256, 512]]
YOLOv10s summary: 402 layers, 8069448 parameters, 8069432 gradients, 24.8 GFLOPs
```

Total 8M parameters in Yolov10-s.

7.2 Image Classification

7.2.1 NasNetMobile

```python
def create_nasnetmobile_model():
    from keras.applications import NASNetMobile
    from tensorflow.keras.layers import Dense
    from tensorflow.keras.models import Model

    pre_model = NASNetMobile(input_shape=(224,224, 3),
                             include_top=False,
                             weights='imagenet',
                             pooling='avg')
    pre_model.trainable = True


    inputs = pre_model.input
    x = Dense(64, activation='relu')(pre_model.output)
    x = Dense(64, activation='relu')(x)
    outputs = Dense(NUM_CLASSES, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=outputs)
    initial_learning_rate = INITIAL_LEARNING_RATE
    optimizer = keras.optimizers.AdamW(learning_rate = initial_learning_rate)
    model.compile(optimizer = optimizer,
                  loss = keras.losses.SparseCategoricalCrossentropy(),
                  metrics = ["accuracy"])

    return model
```

```
 global_average_pooling2d (  (None, 1056)             0         ['activation_187[0][0]']
 GlobalAveragePooling2D)

 dense (Dense)               (None, 64)               67648     ['global_average_pooling2d[0][
                                                                0]']

 dense_1 (Dense)             (None, 64)               4160      ['dense[0][0]']

 dense_2 (Dense)             (None, 4)                260       ['dense_1[0][0]']

==================================================================================================
Total params: 4341784 (16.56 MB)
Trainable params: 4305046 (16.42 MB)
Non-trainable params: 36738 (143.51 KB)
__________________________________________________________________________________________________
```

7.2.2 MobileNetV3Large

```
Model: "MobileNet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 mobile_net_v3_large_backbo  (None, 7, 7, 960)         2996352
 ne (MobileNetV3Backbone)

 max_pool (GlobalMaxPooling  (None, 960)               0
 2D)

 predictions (Dense)         (None, 4)                 3844

=================================================================
Total params: 3000196 (11.44 MB)
Trainable params: 2975796 (11.35 MB)
Non-trainable params: 24400 (95.31 KB)
_________________________________________________________________
```

3M params

7.2.3 EfficientNetV2B0

```
Model: "KerasCV_efficientnet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 260, 260, 3)]     0

 efficient_net_v2b0_backbon  (None, 9, 9, 1280)        5919312
 e (EfficientNetV2Backbone)

 avg_pool (GlobalAveragePoo  (None, 1280)              0
 ling2D)

 predictions (Dense)         (None, 4)                 5124

=================================================================
Total params: 5924436 (22.60 MB)
Trainable params: 5863828 (22.37 MB)
Non-trainable params: 60608 (236.75 KB)
_________________________________________________________________
```

5.9M params

7.2.4 EfficientNetV2B2

```
Model: "KerasCV_efficientnet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 260, 260, 3)]     0         
                                                                 
 efficient_net_v2b2_backbon  (None, 9, 9, 1408)        8769374   
 e (EfficientNetV2Backbone)                                      
                                                                 
 avg_pool (GlobalAveragePoo  (None, 1408)              0         
 ling2D)                                                         
                                                                 
 predictions (Dense)         (None, 4)                 5636      
                                                                 
=================================================================
Total params: 8775010 (33.47 MB)
Trainable params: 8692722 (33.16 MB)
Non-trainable params: 82288 (321.44 KB)
_________________________________________________________________
```

8.7M Params

7.2.5 Ensemble model

```python
# Define the custom objects
# Define the custom objects for loading models
custom_objects_1 = {
    'EfficientNetV2Backbone': keras_cv.models.EfficientNetV2Backbone,
    'ImageClassifier': keras_cv.models.ImageClassifier,
    'AdamW': AdamW
}

model_1 = load_model('./models/efficientnetV2B2_best_model.h5', custom_objects=custom_objects_1)
model_1 = Model(inputs=model_1.inputs,
                outputs=model_1.outputs,
                name='efficientnetV2B2')

```

```
WARNING:tensorflow:Error in loading the saved optimizer state. As a result, your model is starting wi
```

```python
# Define the custom objects
custom_objects_2 = {
    'MobileNetV3Backbone': keras_cv.models.MobileNetV3Backbone,
    'ImageClassifier': keras_cv.models.ImageClassifier,
    'AdamW': AdamW
}
model_2 = load_model('./models/mobilenet_best_model.h5', custom_objects=custom_objects_2)
model_2 = Model(inputs=model_2.inputs,
                outputs=model_2.outputs,
                name='mobileNetV3')
```

```
WARNING:tensorflow:Error in loading the saved optimizer state. As a result, your model is starting wi
```

```python
# Define the custom objects
custom_objects_3 = {
    # 'MobileNetV3Backbone': keras_cv.models.MobileNetV3Backbone,
    # 'ImageClassifier': keras_cv.models.ImageClassifier,
    'AdamW': AdamW
}
model_3 = load_model('./models/nasnetmobile_best_model.h5', custom_objects=custom_objects_3)
model_3 = Model(inputs=model_3.inputs,
                outputs=model_3.outputs,
                name='nasnetmobile')
```

```python
# Define the input layer with a common shape
common_input_shape = (224, 224, 3)  # Chosen common input shape
model_input = Input(shape=common_input_shape)

# Resize inputs to match model_1's required input shape
resize_input_1 = Resizing(260, 260)(model_input)
output_1 = model_1(resize_input_1)

# Resize inputs to match model_2's required input shape
resize_input_2 = Resizing(224, 224)(model_input)
output_2 = model_2(resize_input_2)

# Resize inputs to match model_3's required input shape
resize_input_3 = Resizing(224, 224)(model_input)
output_3 = model_3(resize_input_3)
```

```python
# Average the outputs to create the ensemble output
ensemble_output = Average()([output_1, output_2, output_3])
```

```python
# Create the ensemble model
ensemble_model = Model(inputs=model_input, outputs=ensemble_output, name='ensemble')

# TODO: Freeze the layers
```

```python
# Compile the ensemble model
ensemble_model.compile(optimizer=AdamW(learning_rate=0.0001),
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
```

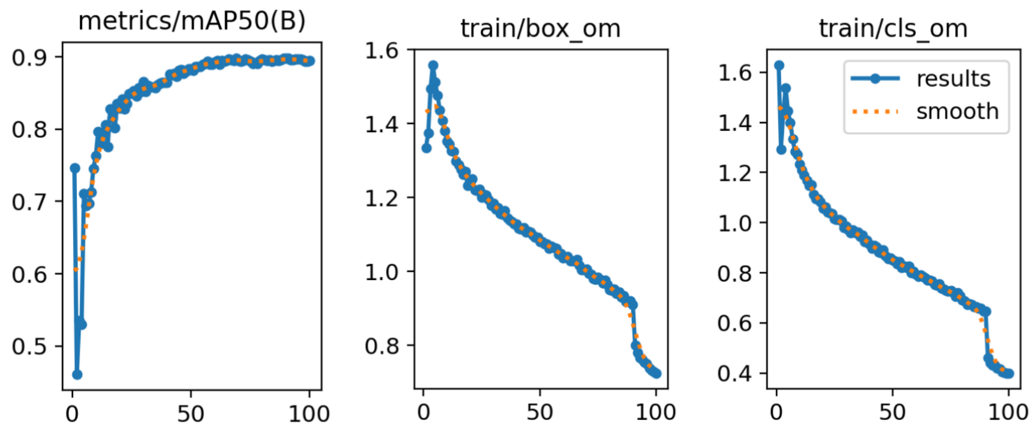
```
Model: "ensemble"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]        0         []

 resizing (Resizing)         (None, 260, 260, 3)          0         ['input_3[0][0]']

 resizing_1 (Resizing)       (None, 224, 224, 3)          0         ['input_3[0][0]']

 resizing_2 (Resizing)       (None, 224, 224, 3)          0         ['input_3[0][0]']

 efficientnetV2B2 (Function  (None, 4)                    8775010   ['resizing[0][0]']
 al)

 mobileNetV3 (Functional)    (None, 4)                    3000196   ['resizing_1[0][0]']

 nasnetmobile (Functional)   (None, 4)                    4341784   ['resizing_2[0][0]']

 average (Average)           (None, 4)                    0         ['efficientnetV2B2[0][0]',
                                                                     'mobileNetV3[0][0]',
                                                                     'nasnetmobile[0][0]']

==================================================================================================
Total params: 16116990 (61.48 MB)
Trainable params: 15973564 (60.93 MB)
Non-trainable params: 143426 (560.26 KB)
__________________________________________________________________________________________________
```

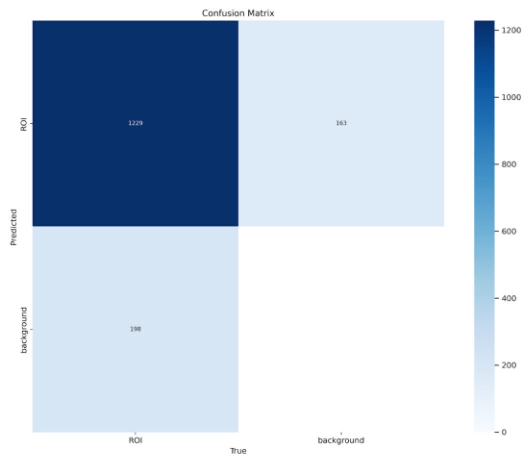
Ensemble have 16.1M parameters. A simple ResNet-50 would have 23M params

8 Evaluation Metrics

IMPORTANT: Since the checkpointing is being used, the model is stored when there is improvement in val_acc or val_loss, the score shown below is when the model was stored and not the score on final epoch.

8.1 Yolov10





8.2 NasNetMobile

Code: 3_NasNetMobile_train_val_test_split
NasNetMobile was trained for 50 epochs and Learning rate was kept at 0.0001
Pre-trained ImageNet weights were used.

Model	Data Split	Monitoring	Pateince	Loss	Acc	Val_loss	Val_Acc	Trainable layers	Verdict
NasNetMobile	Train, Val	val_acc	20 ES / 5 RLR	0.0661	0.9853	0.3232	0.9008	All Frozen	Val_loss is increasing overtime from 20th epoch - overfitting
	Train, Val	val_acc	20 ES / 5 RLR	5.6826e-07	1.0000	0.2573	0.9662	**All**	Val_loss is increasing overtime from 20th epoch - overfitting
	Train, Val	**val_loss**	**5 ES / 3 RLR**	1.0584e-05	1.0000	0.2253	0.9546	**All**	Due to early stopping

									model didn't overtrain and is stable
	Train, Val, Test	**val_loss**	**5 ES / 3 RLR**	5.9582e-06	1.0000	0.1771	0.9669	**All**	Stable

8.3 MobileNetV3Large

Code: 2_MobileNetV3_large_3_split
MobileNetV3Large was trained for 50 epochs and Learning rate was kept at 0.0001
Pre-trained ImageNet weights were used.

Model	Data Split	Monitoring	Pateince	Loss	Acc	Val_loss	Val_Acc	Trainable layers	Verdict
MobileNetV3Large	**Train, Val**	**val_acc**	**20 ES / 5 RLR**	0.0161	0.9950	0.2118	0.9542	All	Stable
	Train, Val, Test	**val loss**	10 ES/ 5 RLR	0.0154	0.9954	0.2678	0.9387	All	Stable

8.4 EfficientNetV2B0

Code: 1_efficientNetV2B0
efficientNetV2B0 was trained for 50 epochs and Learning rate was kept at 0.0001

Model	Data Split	Monitoring	Pateince	Loss	Acc	Val_loss	Val_Acc	Trainable layers	ImageNet Weights	Verdict
EfficientNetV2B0	**Train, Val**	**val_acc**	**20 ES / 5 RLR**	0.0120	0.9969	0.1416	0.9600	All	True	Stable
	Train, Val	**val_acc**	**20 ES / 5 RLR**	0.0507	0.9815	0.6046	0.8661	All	**False**	Did not coverge well. Loss is high

Imagenet weights are important even though all layers are trained again. The model converges faster. Pre-trained weights are providing a good starting point to the model. The efficientNetV2B2 model is available with more parameters and better performance, so all other experiments will be on that model.

8.5 EfficientNetV2B2

Val Loss is less and val accuracy is more as compared to B0.

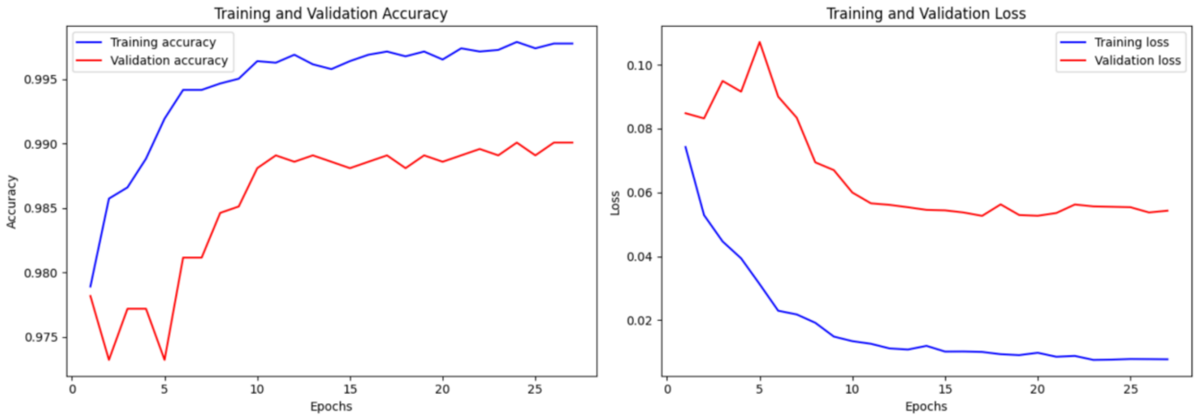
Model	Data Split	Monitoring	Pateince	Loss	Acc	Val_loss	Val_Acc	Trainable layers	Verdict
EfficientNetV2B2	**Train, Val**	**val_acc**	**20 ES / 5 RLR**	0.0085	0.9978	0.1225	0.9729	All	Stable
	Train, Val, Test	**val_loss**	**10 ES / 5 RLR**	0.0348	0.9884	0.1012	0.9674	All	Stable

8.6 Ensemble

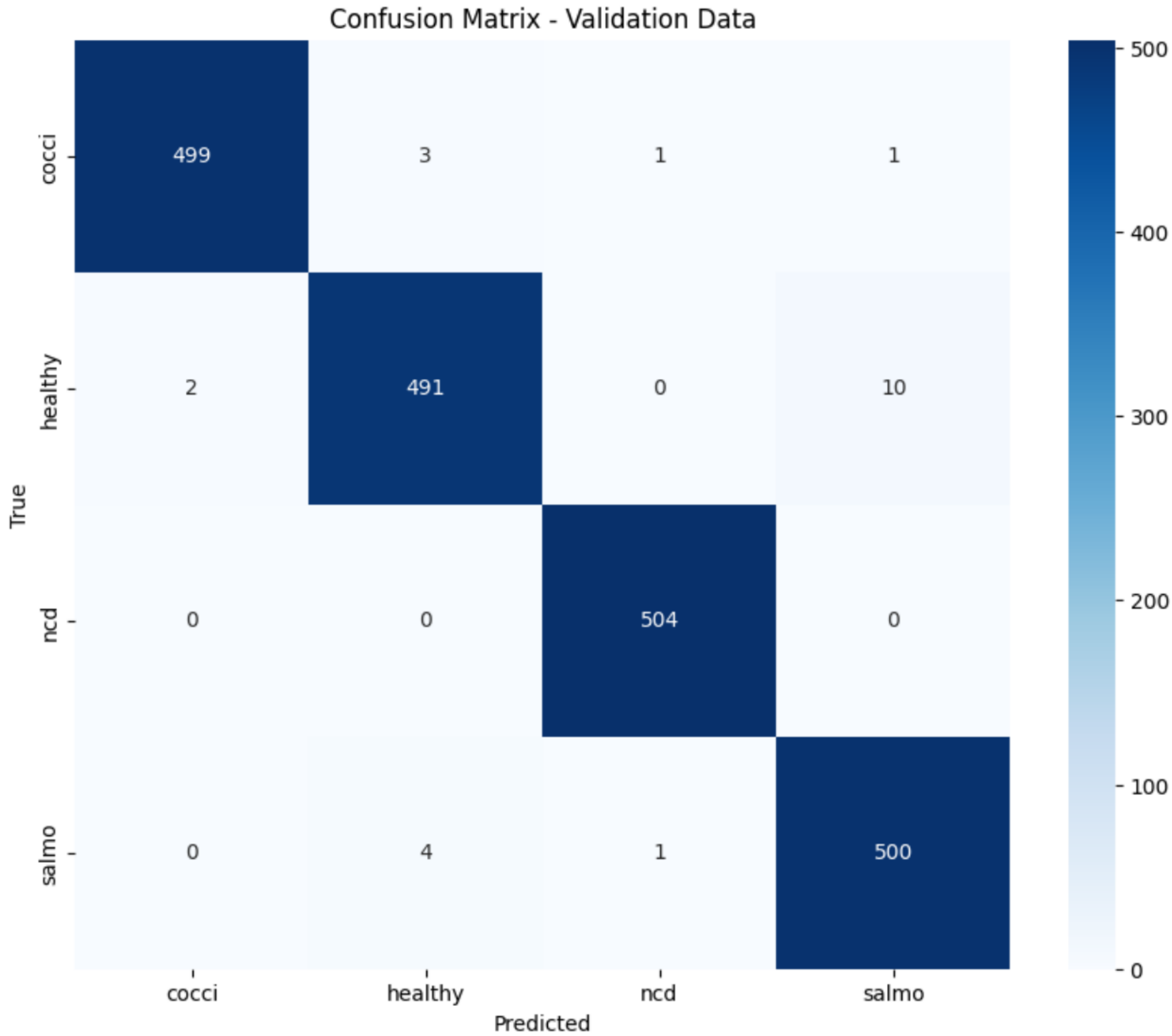
All layers are trainable for fine-tunning complete model.

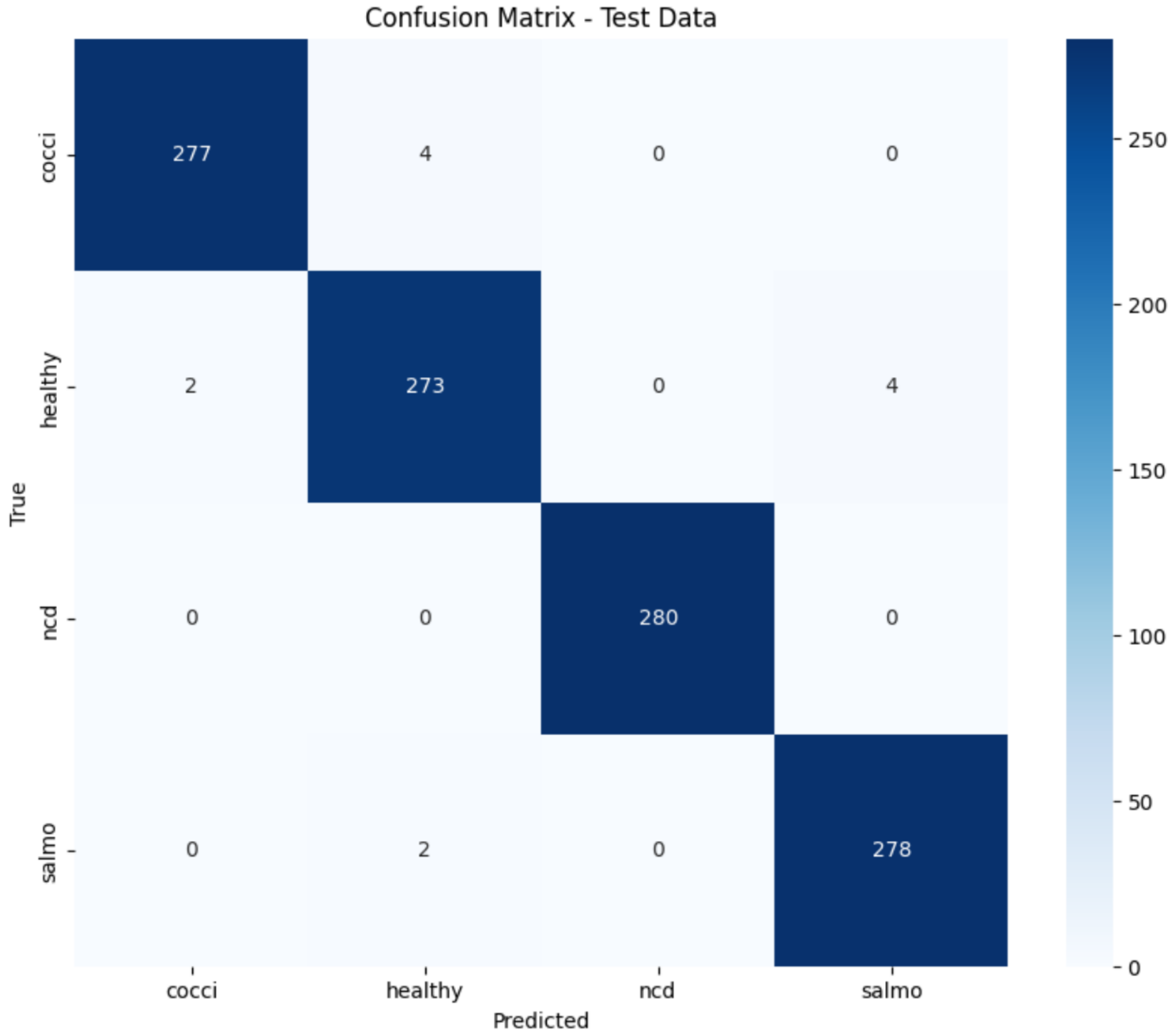
Model	Epochs	Data Split	Monitoring	Pateince	Loss	Acc	Val_loss	Val_Acc	Trainable layers	Verdict
Ensemble	**10**	**Train, val, test**	**Val_loss**	**5 ES / 3 RLR**	0.0480	0.9873	0.0749	0.9787	All	Need to run on more epoch to stablize
	30	**Train, val,**	**Val_loss**	**5 ES / 3 RLR**	0.0101	0.9971	0.0527	0.9891	All	Stable

		test								



Precision: 0.989114487340604
Recall: 0.9890873015873016
F1-score: 0.9890829980078678





Precision: 0.9892982427471103
Recall: 0.9892857142857143
F1-score: 0.9892856802418476