National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Sripriya Kumari Murali
Student ID: X23323981

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sripriya Kumari Murali
Student ID:	23323981
Programme:	MSc Data Analytics (MSCDAD_C) **Year:** 2024-2025
Module:	MSc (Research) Practicum/Internship Part 2
Lecturer:	Christian Horn
Submission Due Date:	15-09-2025
Project Title:	A Comparative Study Of Custom CNN And MobileNetv2 for Binary Waste Classification Using Deep Learning
Word Count:	1884 **Page Count:** 7

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:	Sripriya Kumari Murali
Date:	15-09-2025

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

Sripriya Kumari Murali
23323981

1 Section 1: Data Preparation and Environment Setup

This section covers the preliminary implementation of the deep learning pipeline of binary image classification. The implementation of the project is carried out with the Python programming language and the major libraries, which include TensorFlow, NumPy, Matplotlib, and PIL. The implementation was performed on a windows 10 machine with 8GB RAM and Intel i5 processor having some hardware restrictions and it affected some decisions such as the resizing of images and the batch size to handle memory usage (Alzubaidi *et al*. 2021). The data is colleted and visualized to determine the distribution of images, dimensional consistency and class balance. In order to achieve an appropriate input format to the neural network models, the images are resized to (225, 225) and normalized in order to be labeled into the appropriate digital format.

2 Section 2: Model Development and Transfer Learning

The architecture and implementation of two classifying waste images models: a custom built deep CNN and a transfer learning model with the MobileNetV2 are described in this section. The standard CNN has a series of convolutional layers, ReLU activations, and pooling, dropout, and an output dense sigmoid layer that undertakes the binary classification. Conversely, MobileNetV2 accepts pre-trained base with frozen weights in order to exploit learned image features then apply global average pooling and dense layers in the process of classification. Both models are also produced by adam optimizer and binary cross-entropy loss, where EarlyStopping and ReduceLROnPlateau are used as callbacks to make the model training, and stop overfitting (Raj *et al*. 2023).

3 Section 3: Manual Configuration and Model Development for Image Classification using Deep Learning

```
import zipfile
import os
import matplotlib.pyplot as plt
from PIL import Image
import glob
import numpy as np
from tqdm import tqdm
from collections import Counter
```

Figure 1: Data Preparation and Visualization Setup
(Source: Self-created)

The section includes the importation of important libraries to work with image data and file functionalities. zipfile and os are used in file extrusion and path directories, whereas PIL and matplotlib.pyplot were used in displaying images. glob was used to retrieve the paths of the images, and tqdm library was used in providing a progress indicator on the loop. numpy was used to carry out numerical manipulations and the use of Counter in counting the sizes of images as part of the preprocessing and analysis procedures.

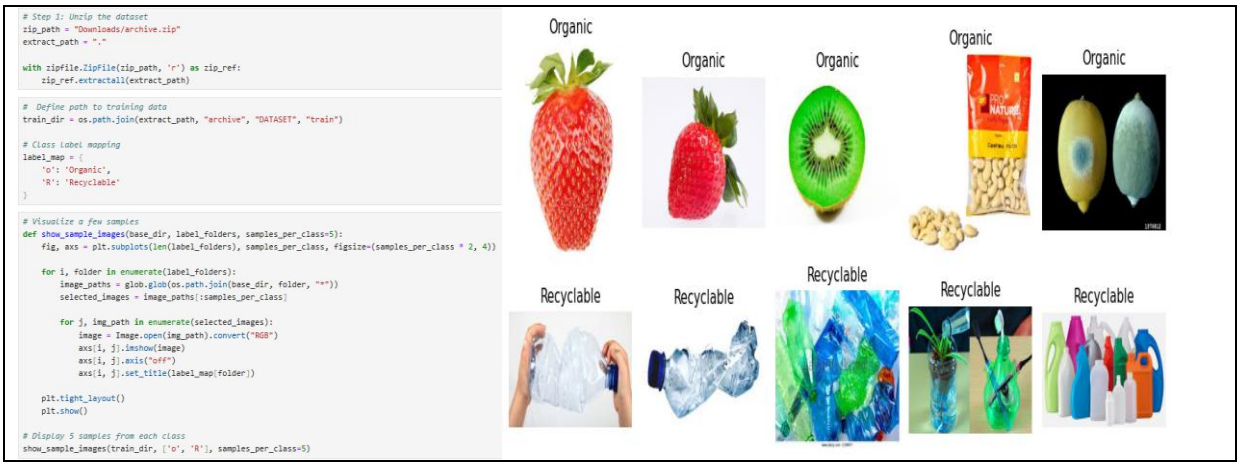


Figure 2: Dataset Extraction and Sample Visualization

(Source: Self-created)

The code starts with the unzipping of one of the datasets in the file archive.zip, in a local directory. Then, it initializes the path to the training data, and a label mapping dictionary to match the names of the folders containing the classes (.e. o - Organic, R - Recyclable) with their category. There is a method called show_sample_images, which will show some sample images of the two categories. It reads and displays five images of each class with matplotlib and PIL and gives a rough visual idea of what the dataset contains with regard to Organic and Recyclable waste.

```
# Base path for new structure
base_path = os.path.join("archive", "DATASET")

# Folder structure
subsets = ['train', 'test']
classes = ['o', 'R']
label_map = {'o': 'Organic', 'R': 'Recyclable'}

# Count dictionary
counts = {}

# Loop through subsets and classes
for subset in subsets:
    for cls in classes:
        dir_path = os.path.join(base_path, subset, cls)
        image_files = glob.glob(os.path.join(dir_path, "*"))
        class_name = f"{label_map[cls]} ({subset})"
        counts[class_name] = len(image_files)

# Print counts
for key, value in counts.items():
    print(f"{key}: {value} images")

# Bar plot
plt.figure(figsize=(8, 5))
plt.bar(counts.keys(), counts.values(), color=["green", "orange", "lightgreen", "salmon"])
plt.title("Image Count per Class (Train/Test)")
plt.ylabel("Number of Images")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Figure 3: Class Distribution Analysis Across Training and Testing Sets

(Source: Self-created)

In this part, the authors study the quantity of the images relating to each of the two classes: Organic and Recyclable, that are present in both training and testing sets. The script contains nested loops that iterates through the directories of the datasets, which keep the number of files of each type of image in each of the specific classes, and that information is saved as a dictionary. It proceeds then to print the counts of each subset and then plots the distribution on a bar chart chart. The bar plot is useful in balancing the classes that is essential in training and evaluation of effective models. The randomness of the classes avoids bias and enables good performance measurements of the two groups in binary classification.

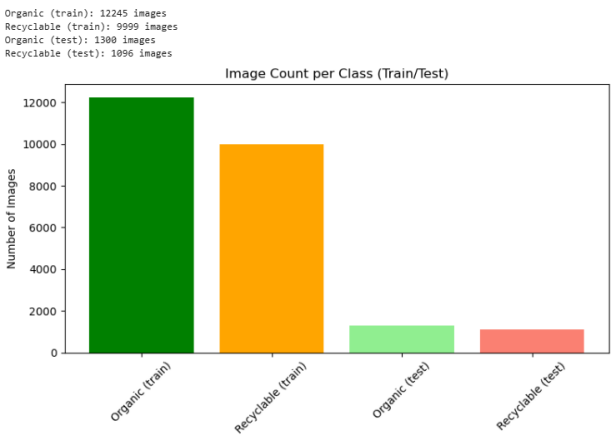


Figure 4: Image Class Distribution

(Source: Self-created)

Bar chart shows how images in training and testing data are distributed between two categories of wastes, Organic and Recyclable. As far as the analysis is concerned:

- Organic (train): 12,245 images
- Recyclable (train): 9,999 images
- Organic (test): 1,300 images
- Recyclable (test): 1,096 images

The training set is very different from the test set with the training set being significantly larger than the test set, something which is common in model development. But there is a minor imbalance when it comes to the class distribution: the images of organic waste are more numerous than recyclable images in both subsets. Unless this imbalance is corrected, (e.g., by data augmentation or weighted loss functions), it can be used to bias the model to output its predictions, from a majority class. It is further evident to the reader, with the bar chart as well, that although both categories are well represented, making further attempts to balance them would enhance overall classification and generalization in real world applications.

```python
import pandas as pd

# function to handle large number of unique image sizes
def plot_image_size_distribution(base_dir, classes):
    size_counts = Counter()

    # Count all image sizes from training data
    for cls in classes:
        image_paths = glob.glob(os.path.join(base_dir, cls, "*"))
        for img_path in image_paths:
            try:
                with Image.open(img_path) as img:
                    size = img.size  # (width, height)
                    size_counts[size] += 1
            except:
                continue

    # Convert the Counter to a DataFrame
    size_df = pd.DataFrame(size_counts.items(), columns=['Size (W x H)', 'Count'])
    size_df['Size (W x H)'] = size_df['Size (W x H)'].astype(str)
    size_df = size_df.sort_values(by='Count', ascending=False)

    # Show top 10 most common sizes in a bar chart
    top_10 = size_df.head(10)
    plt.figure(figsize=(10, 5))
    plt.bar(top_10['Size (W x H)'], top_10['Count'], color='teal')
    plt.xticks(rotation=45)
    plt.title("Top 10 Most Common Image Sizes (Train Set)")
    plt.xlabel("Image Size (Width x Height)")
    plt.ylabel("Number of Images")
    plt.tight_layout()
    plt.show()

    # ✅ Print full variation in a table (sorted by count)
    print("\n Full Image Size Variation Table:")
    pd.set_option('display.max_rows', None)  # <--- this line forces full row display
    display(size_df.reset_index(drop=True))  # For Jupyter notebook environments

    # Return the most common size
    most_common_size = eval(top_10.iloc[0]['Size (W x H)'])  # Convert string back to tuple
    print(f"\n Most common image size: {most_common_size} with {top_10.iloc[0]['Count']} images")
    return most_common_size
```

Figure 5: Image Size Distribution in the Dataset

(Source: Self-created)

The code plots the occurrence of dimension of the training images. It reads every picture in the given class folders, gets it width and length values and counts the frequency of each distinct length and width combination in a Counter. The data is translated into a DataFrame to sort and depict in a bar chart the most frequent image sizes values by 10. This visualization can be used to find out which image dimensions are predominant which is paramount during the preprocessing stage, since deep-learning models usually need image sizes that are the same in size. The most common image size is also printed, which helps in making a decision regarding resizing as a component of model training. Furthermore, all the distinct image size measurements are posted in a full table, organized by their frequency, to evaluate consistency in datasets. This will provide an improved ability to force standardization of the input shapes, minimising the size-induced computational errors and in the way, enhancing the model performance by removing the inconsistencies in sizes that may be observed throughout the dataset.

Figure 6: Top 10 Most Common Image Sizes (Train Set)
(Source: Self-created)

The bar graph shows 10 of the most frequently occurring size of images used in training data. The most popular is (225, 225) containing more than 4,000 pictures, then it is (275, 183) and (259, 194). It indicates that there is a significant spread in the dimension of images in the dataset, even though, few sizes predominate. This inconsistency in image dimension requires standardization (e.g., resizing) prior to the deep learning training. The task of determining the most frequent dimensions assists in the process of choosing the convenient target size that would ensure less distortion and preserve image quality combined with the maximization of performance and the complexity of computing models during training.

```python
# Mapping classes to numerical labels for training
label_map = {'o': 0, 'R': 1}

# Preprocessing function
def preprocess_images(subset='train'):
    images = []
    labels = []

    for cls_folder in ['o', 'R']:
        dir_path = os.path.join(base_path, subset, cls_folder)
        image_paths = glob.glob(os.path.join(dir_path, "*"))

        for img_path in tqdm(image_paths, desc=f"Processing {label_map[cls_folder]} - {subset}"):
            try:
                img = Image.open(img_path).convert('RGB')
                img = img.resize(target_size)
                img_array = np.asarray(img, dtype=np.float32) / 255.0  #  Reduce memory usage

                images.append(img_array)
                labels.append(label_map[cls_folder])
            except Exception as e:
                print(f"Error processing {img_path}: {e}")

    return np.array(images), np.array(labels)
```

Figure 7: Image Preprocessing and Label Encoding
(Source: Self-created)

The code determines a preprocessing pipeline of images in order to train the model. All of the images are divided into two categories, which are, respectively, o or R according to the label_map. The preprocess_images() is used to read images in the mentioned folders, transform them into RGB and resize their size to a standard target_size and normalize pixel values to a range between 0 and 1 in order to use less memory and converge faster during training. The program retrieves the trained images and the labels and exposes them as the NumPy arrays to feed into a neural network.

```
# Preprocess both training and test datasets
X_train, y_train = preprocess_images('train')


# Display shape of processed data
print("Train data shape:", X_train.shape, y_train.shape)
```

```
Processing 0 - train: 100%|████████████████████████████████████████| 12245/12245 [00:33<00:00, 368.27it/s]
Processing 1 - train: 100%|████████████████████████████████████████| 9999/9999 [00:29<00:00, 337.57it/s]
Train data shape: (22244, 225, 225, 3) (22244,)
```

```
# Preprocess both training and test datasets
X_test, y_test = preprocess_images('test')

# Display shape of processed data
print("Test data shape:", X_test.shape, y_test.shape)
```

```
Processing 0 - test: 100%|████████████████████████████████████████| 1300/1300 [00:11<00:00, 113.05it/s]
Processing 1 - test: 100%|████████████████████████████████████████| 1096/1096 [00:10<00:00, 108.16it/s]
Test data shape: (2396, 225, 225, 3) (2396,)
```

Figure 8: Training and Testing Data Preprocessing

(Source: Self-created)

The code snippet reveals how the code snippet preprocesses both the training and testing image datasets with the help of the preprocess_images() function. In the case of the training set, we were able to process and adjust the shape of 22,244 images to a similar size of (225, 225, 3) and divide them into two classes (that is, 'o' and 'R'). On the same note, the test set contains 2,396 images that are treated similarly. The shape results attest to the uniformity in the dimensions of the images and their assigned labels. The tqdm progress bars tell the speed that the processing runs, and also the number of seconds in total usable time, so that the image is loaded fully and prepared to run further machine learning or deep learning activities.

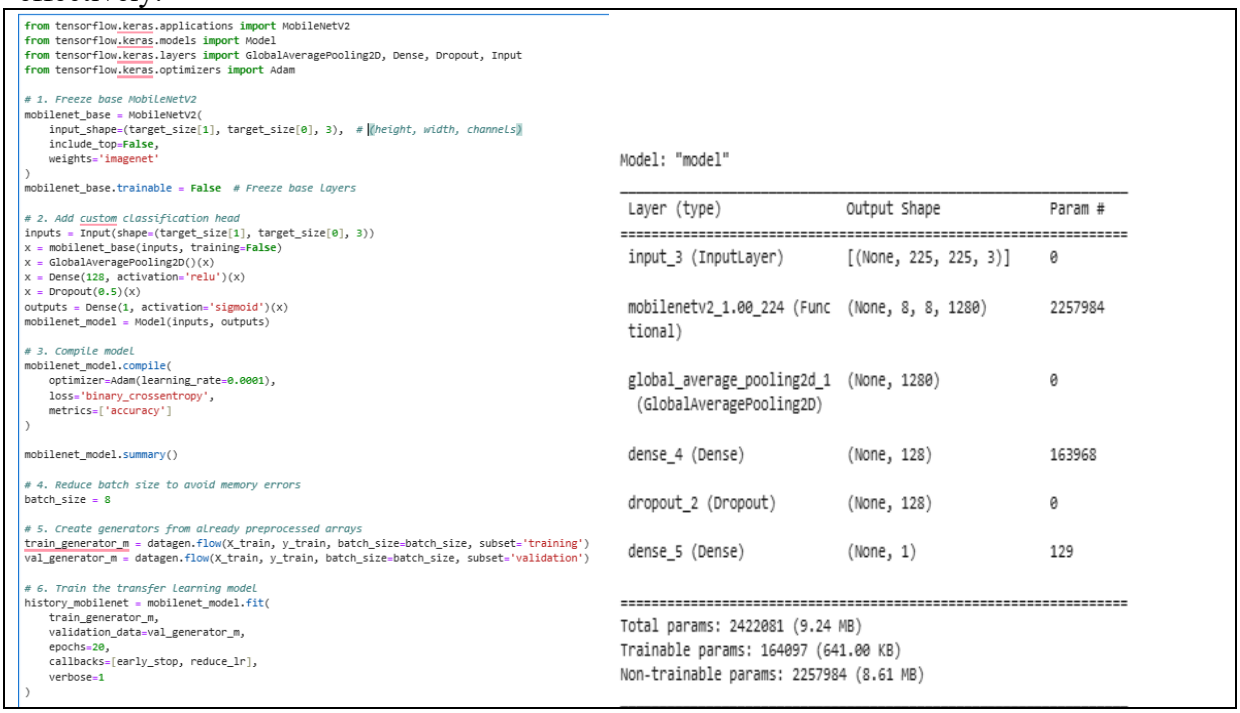
```
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# 1. Define a deeper CNN model with one more Conv Layer
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(target_size[1], target_size[0], 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # NEW Layer
    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')  # Binary classification
])

# 2. Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 223, 223, 32)      896

max_pooling2d (MaxPooling2   (None, 111, 111, 32)      0
D)

conv2d_1 (Conv2D)            (None, 109, 109, 64)      18496

max_pooling2d_1 (MaxPoolin   (None, 54, 54, 64)        0
g2D)

conv2d_2 (Conv2D)            (None, 52, 52, 128)       73856

max_pooling2d_2 (MaxPoolin   (None, 26, 26, 128)       0
g2D)

conv2d_3 (Conv2D)            (None, 24, 24, 256)       295168

max_pooling2d_3 (MaxPoolin   (None, 12, 12, 256)       0
g2D)

flatten (Flatten)            (None, 36864)             0

dense_4 (Dense)              (None, 256)               9437440

dropout_2 (Dropout)          (None, 256)               0

dense_5 (Dense)              (None, 1)                 257

=================================================================
Total params: 9826113 (37.48 MB)
Trainable params: 9826113 (37.48 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 9: Deep CNN Model Architecture

(Source: Self-created)

The represented model architecture depicts deep Convolutional Neural Network (CNN) architecture to be used in the classification of binary images. It has an input layer that accepts images of 225x225 as RGB. The four blocks of the network are convolutional, each of which uses a Conv2D layer and subsequently a MaxPooling2D layer. The filters grow exponentially (32, 64, 128, 256). This way, the model is able to reproduce more complex spatial behaviors. All convolutions have ReLU activation that is utilized to induce non-linearity. Unique features of pooling layers are the minimization of the space dimensions, managing overfitting and computation expenses (Chollet, 2021). The model then flattens its feature maps after the convolutional layers, then applying a fully connected (Dense) layer of 256 neurons with the ReLU activation followed by Dropout layer (rate = 0.5) to avoid overfitting. Lastly, the binary classification is done using a single neuron whose activation is sigmoid. The model possesses close to 9.8 million parameters hence is well suited in learning fine image patterns effectively.



```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam

# 1. Freeze base MobileNetV2
mobilenet_base = MobileNetV2(
    input_shape=(target_size[1], target_size[0], 3),  # (height, width, channels)
    include_top=False,
    weights='imagenet'
)
mobilenet_base.trainable = False  # Freeze base layers

# 2. Add custom classification head
inputs = Input(shape=(target_size[1], target_size[0], 3))
x = mobilenet_base(inputs, training=False)
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
outputs = Dense(1, activation='sigmoid')(x)
mobilenet_model = Model(inputs, outputs)

# 3. Compile model
mobilenet_model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

mobilenet_model.summary()

# 4. Reduce batch size to avoid memory errors
batch_size = 8

# 5. Create generators from already preprocessed arrays
train_generator_m = datagen.flow(X_train, y_train, batch_size=batch_size, subset='training')
val_generator_m = datagen.flow(X_train, y_train, batch_size=batch_size, subset='validation')

# 6. Train the transfer learning model
history_mobilenet = mobilenet_model.fit(
    train_generator_m,
    validation_data=val_generator_m,
    epochs=20,
    callbacks=[early_stop, reduce_lr],
    verbose=1
)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 225, 225, 3)]     0

 mobilenetv2_1.00_224 (Func  (None, 8, 8, 1280)        2257984
 tional)

 global_average_pooling2d_1  (None, 1280)              0
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 128)               163968

 dropout_2 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 2422081 (9.24 MB)
Trainable params: 164097 (641.00 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

Figure 10: MobileNetV2 Transfer Learning Model

(Source: Self-created)

The code contains an example of classification classification of binary images using MobileNetV2 based on transfer learning. It is used to extract features and is frozen to keep the weights that were learned (via pre-training on ImageNet pre-trained with include_top=False). The input shape (225, 225, 3) differs a little bit with the default one, but TensorFlow loads weights that match as closely as possible and performs well. There is a GlobalAveragePooling2D layer that narrows down space and then a Dense layer (128 units, ReLU) and a Dropout layer (0.5 rate) are used to peddle overfitting.(Pandey and Silakari, 2023). A sigmoid activation is applied to the last output layer because the problem is understandably a binary one. Adam optimizer (learning rate = 0.0001), binary cross-entropy loss, and accuracy are the metric used to compile the model. It is equipped with a small batch training of 8 with the use of data augmentation through ImageDataGenerator. ReduceLROnPlateau and EarlyStopping are the types of callbacks that improve the stability of training. This model is highly efficient and has both parameters and good performance in binary classification, as it contains approximately 2.42 million parameters, of which only approximately 164K are trainable

References

Alzubaidi, L., Duan, Y., Al-Dujaili, A., Ibraheem, I.K., Alkenani, A.H., Santamaría, J., Fadhel, M.A., Al-Shamma, O. and Zhang, J., 2021. Deepening into the suitability of using pre-trained models of ImageNet against a lightweight convolutional neural network in medical imaging: An experimental study. *PeerJ Computer Science*, *7*, p.e715.

Chollet, F., 2021. *Deep learning with Python*. simon and schuster.

Hasan, R.T. and Sallow, A.B., 2021. Face detection and recognition using opencv. *Journal of Soft Computing and Data Mining*, *2*(2), pp.86-97.

Pandey, R. and Silakari, S., 2023. Investigations on optimizing performance of the distributed computing in heterogeneous environment using machine learning technique for large scale data set. *Materials Today: Proceedings*, *80*, pp.2976-2982.

Raj, R., Tiwari, P., Gourisaria, M.K. and Das, H., 2023, December. Efficient object detection and labeling in retail environments using MobileNetV2 with inverted residuals. In 2023 OITS International Conference on Information Technology (OCIT) (pp. 634-639). IEEE.