SUPERVISED LEARNING AND UNSUPERVISED LEARNING (TEAM WORK)

SEMESTER 3, SESSION 2024/2025

COURSE CODE: SAIA 2113

COURSE NAME: MACHINE LEARNING

YEAR / PROGRAMME: 2 / BACHELOR IN ARTIFICIAL INTELLIGENCE

DURATION: 3 WEEKS

SUBMISSION DATE: 5th DECEMBER 2025

NAME	FADZLEE ADAM BIN MOHD NAZLEE MUHAMMAD ZAHIN BIN MOHD ZAMRI OSAMA ALCHIKH WARAK
MATRIC ID	A24AI0027 A24AI0065 A24AI0002
SECTION	3
LECTERUR'S NAME	DR. NOOR JANNAH BINTI ZAKARIA

Part 1: Dataset (15 marks)

Task: Create an image dataset for classification problems.

Requirements:

1. At least 2 classes (e.g., Positive or negative, normal or abnormal, yes or no, 0 or 1, Cats vs Dogs). (5 marks)

We have three classes as follows plastic_water_bottles, aluminum_soda_cans and cardboard_boxes.

2. Minimum 50 images per class. (If you have 2 classes, then the number of images should be 100). (5 marks)

Source/Category	ONLINE (Kaggle)	MANUAL
plastic_water_bottles	50	57
aluminum_soda_cans	50	52
cardboard_boxes	50	49

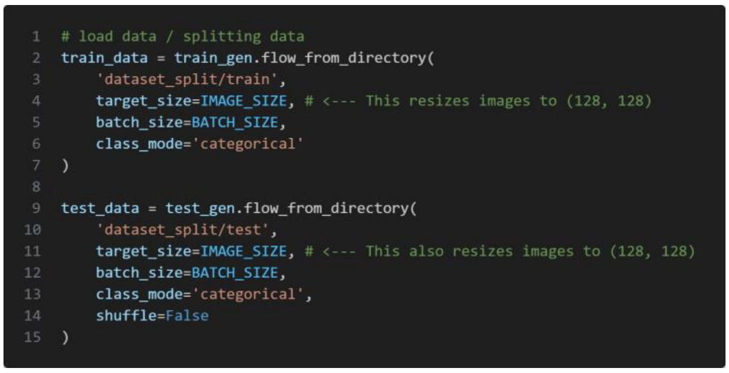
3. You can make your own image collection by taking pictures with your phone or by using free images from websites like Unsplash or Wikimedia. Feel free to mix both your own photos and ones you find online. (5 marks)

We use an online dataset from kaggle and manually capture photos to test our model on real world data.

Part 2: Supervised Learning (25 marks)

Task: Apply one supervised learning technique.

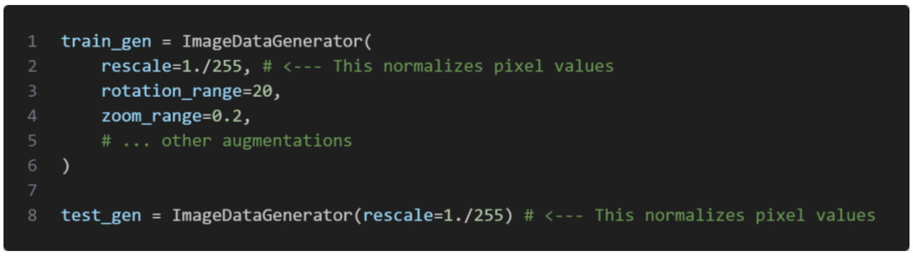
1. Preprocess by resizing and normalizing the images. Write the code and discuss the process. (5 marks)



```
# load data / splitting data
train_data = train_gen.flow_from_directory(
    'dataset_split/train',
    target_size=IMAGE_SIZE, # <--- This resizes images to (128, 128)
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

test_data = test_gen.flow_from_directory(
    'dataset_split/test',
    target_size=IMAGE_SIZE, # <--- This also resizes images to (128, 128)
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)
```

Figure 1: Loading Data Code

The `target_size` argument for `flow_from_directory` calls is responsible for resizing all images into consistent shapes which is `(128, 128)`.

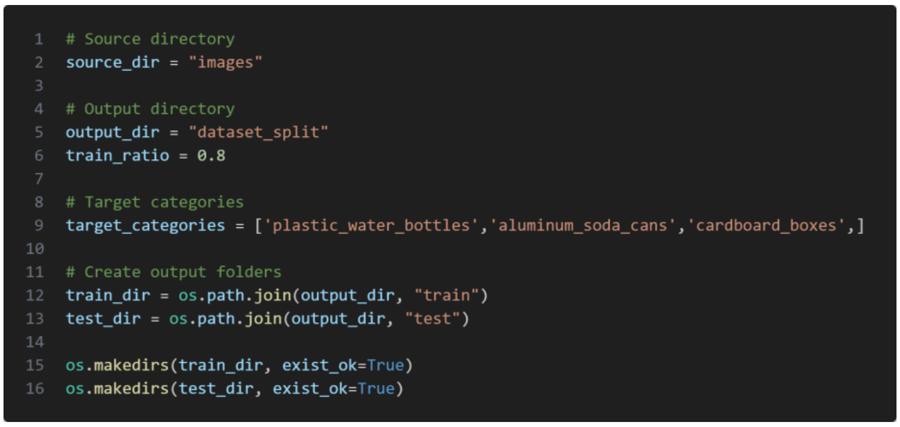


```python
train_gen = ImageDataGenerator(
    rescale=1./255, # <--- This normalizes pixel values
    rotation_range=20,
    zoom_range=0.2,
    # ... other augmentations
)

test_gen = ImageDataGenerator(rescale=1./255) # <--- This normalizes pixel values
```

Figure 2: Normalization Code

The line `rescale=1./255` as argument for ImageDataGenerator instance is responsible for the normalizing pixel values of images from the original `[0, 255]` to a `[0.0, 1.0]` range.

2. Split into training and testing sets. Write the code and discuss the process. (5 marks)



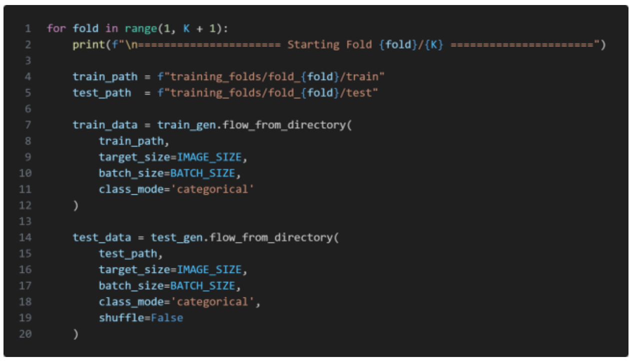
```python
# Source directory
source_dir = "images"

# Output directory
output_dir = "dataset_split"
train_ratio = 0.8

# Target categories
target_categories = ['plastic_water_bottles','aluminum_soda_cans','cardboard_boxes',]

# Create output folders
train_dir = os.path.join(output_dir, "train")
test_dir = os.path.join(output_dir, "test")

os.makedirs(train_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)
```

Figure 3: Splitting Data Code

We split the dataset into 80% for training and 20% for the testing to see the performance. We create a directory where the data will be stored, and under it we create two folders using the OS module, one for training and one for testing, to simplify the pipeline.

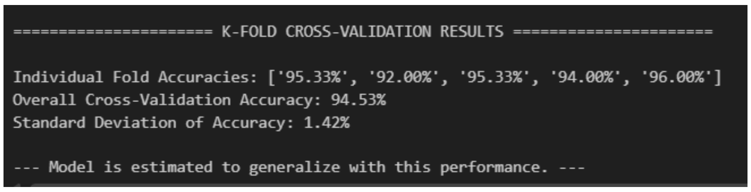
3. Use one supervised learning technique. Write the code and discuss the process. (5 marks)

We first validate the model architecture by tuning the hyperparameters to across multiple validation sessions. We use the k-fold cross validation method to find best hyperparameters.



```python
for fold in range(1, K + 1):
    print(f"\n===================== Starting Fold {fold}/{K} =====================")

    train_path = f"training_folds/fold_{fold}/train"
    test_path  = f"training_folds/fold_{fold}/test"

    train_data = train_gen.flow_from_directory(
        train_path,
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='categorical'
    )

    test_data = test_gen.flow_from_directory(
        test_path,
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        shuffle=False
    )
```

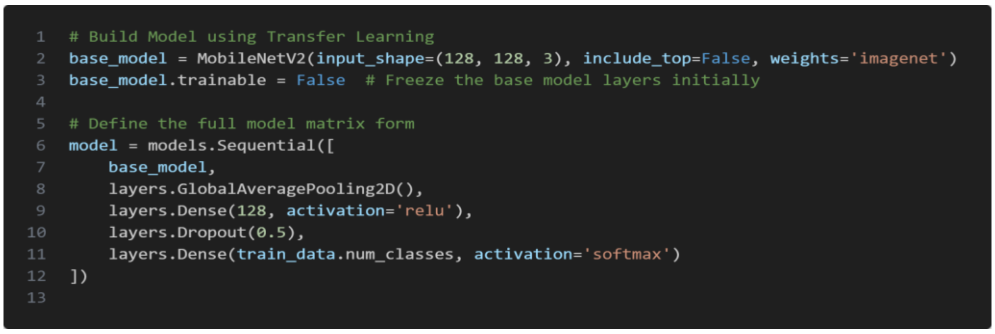
Figure 4: k-fold Cross-validation Code



```
==================== K-FOLD CROSS-VALIDATION RESULTS ====================

Individual Fold Accuracies: ['95.33%', '92.00%', '95.33%', '94.00%', '96.00%']
Overall Cross-Validation Accuracy: 94.53%
Standard Deviation of Accuracy: 1.42%

--- Model is estimated to generalize with this performance. ---
```

Figure 5: k-fold Cross-validation results

After validating our mode using the k-fold cross validation method and satisfied with the result, we proceed to use this final model architecture for our final model.

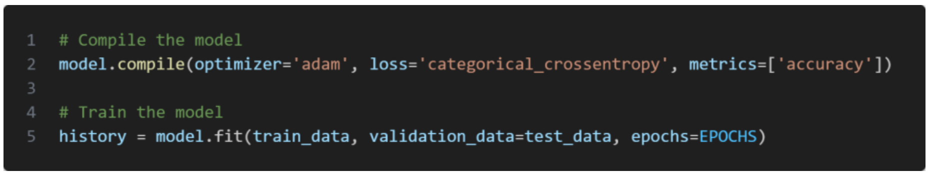


```python
# Build Model using Transfer Learning
base_model = MobileNetV2(input_shape=(128, 128, 3), include_top=False, weights='imagenet')
base_model.trainable = False  # Freeze the base model layers initially

# Define the full model matrix form
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(train_data.num_classes, activation='softmax')
])

```

Figure 6: Transfer Learning Code

MobileNetV2 model which use convolutional neural networks (CNN), best supervised learning technique for extracting features from images. We freeze its layers initially so that only new classifier layers maps into a feature vector. A `GlobalAveragePooling2D` layer converts feature maps into a feature vector. The `Dense` and `Dropout` layers added to learn classify images. And finally, a softmax layer to output class probabilities.

4. Train the model to classify your images. Write the code and discuss the process. (5 marks)



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

# Train the model
history = model.fit(train_data, validation_data=test_data, epochs=EPOCHS)
```

Figure 7: Training Model Code

After specifying the model build, we then compile it to define how it learns. `Adam` optimizer updates the model weights and reduce the loss, categorical crossentropy as our loss function and accuracy metric to monitor the performance during training.

Then we train it using `model.fit()` on the training data and validate on the test data for specified number of epochs which we set to 15.

5. Evaluate with performance metrics such as accuracy, precision, recall, and F1-score. Write the code and discuss the process. (5 marks)

```python
# Display confusion matrix
plt.figure(figsize=(8, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap=plt.cm.Blues, values_format='d')
plt.title("Confusion Matrix")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Figure 8: Displaying Code

Model performance during training

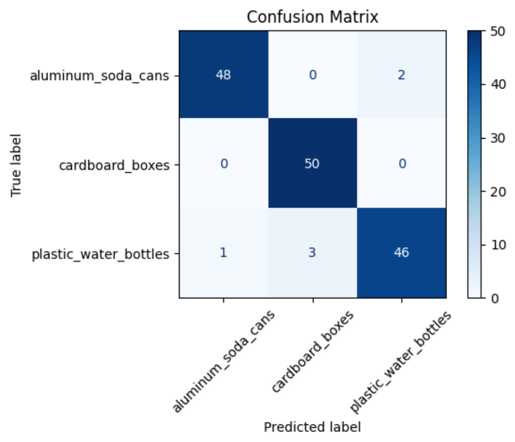


Figure 9: Confusion Matrix Result (Training)

Model performance during testing

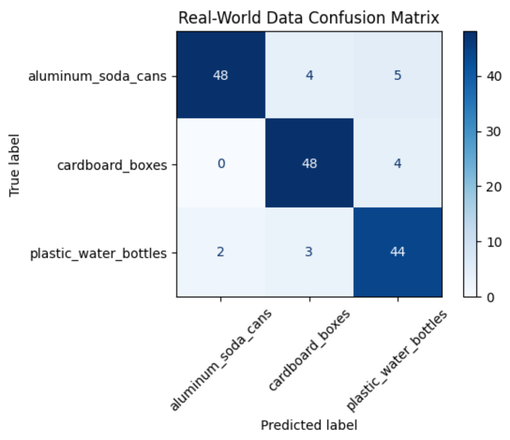


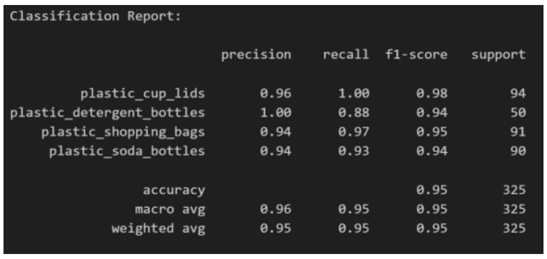
Figure 10: Confusion Matrix Result (Testing)

We evaluate our model using a confusion matrix to visualize the pattern of its predictions. The result for training is very good as false predictions across all predictions are negligible with highest false class predictions being only 3 false `plastic_detergent_bottles` as `plastic_soda_bottles`. On the other hand, the result for testing our model on real-world data shows that the model struggles a little for some photos. Simulating real-world data, we capture the photos by multiple angles with some photos introduce noise foliage around them. The results suggest that the data for training models are not as robust and failed to provide real-world challenges. Hence, some photos are misclassified. However, the model still able to classify accurately for most of the photos, suggesting that overall, it can handle real-world data.

```python
report = sklearn.metrics.classification_report(true_labels, pred_labels, target_names=class_names)
print("Classification Report:\n")
print(report)
```

Figure 11: Display Report Code

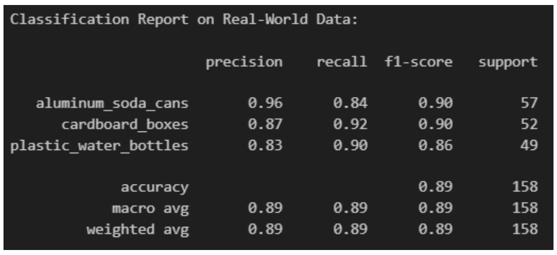
Report Result (training)



```
Classification Report:

                         precision    recall  f1-score   support

        plastic_cup_lids      0.96      1.00      0.98        94
plastic_detergent_bottles      1.00      0.88      0.94        50
    plastic_shopping_bags      0.94      0.97      0.95        91
     plastic_soda_bottles      0.94      0.93      0.94        90

                 accuracy                          0.95       325
                macro avg      0.96      0.95      0.95       325
             weighted avg      0.95      0.95      0.95       325
```

Figure 12: Evaluation Report Result (train)

Report Result (testing)



```
Classification Report on Real-World Data:

                      precision    recall  f1-score   support

     aluminum_soda_cans      0.96      0.84      0.90        57
        cardboard_boxes      0.87      0.92      0.90        52
   plastic_water_bottles      0.83      0.90      0.86        49

               accuracy                          0.89       158
              macro avg      0.89      0.89      0.89       158
           weighted avg      0.89      0.89      0.89       158
```

Figure 13: Evaluation Report Result (test)

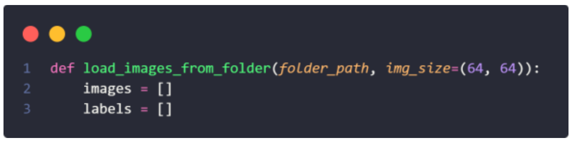
To evaluate the performance of our model, we use the module `metrics.classification_report` from scikit-learn library. Overall, the precision, recall, f1-score and support of our model is comparable good to global standard benchmarks. Comparing from the training and testing, we found that our model struggles a bit when classifying the real-world photos. This indicates that while it can learn the training distribution effectively, it struggles when generalizing to more diverse and unpredictable data. Lower precision and recall especially for certain class suggest that real-world samples contain higher variability in lighting, angle, and object appearance. Despite this, the model still maintains solid overall accuracy on the real-world dataset, showing that its generalization capability is reasonably good.

Part 3: Unsupervised Learning (25 marks)

Task: Apply k-means clustering technique.

Requirements:

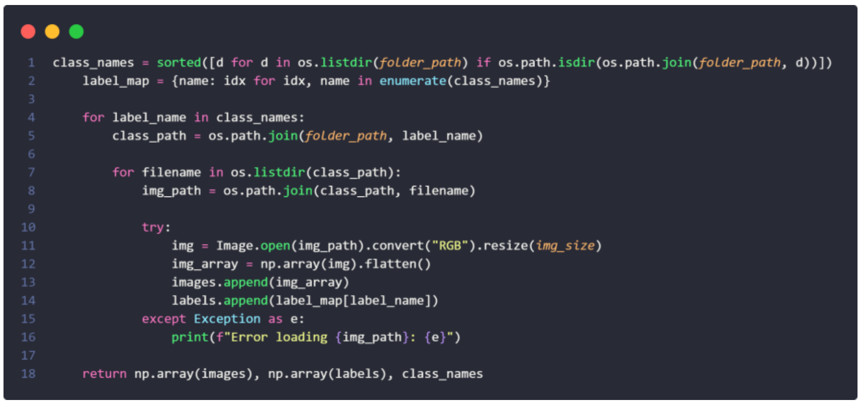
1. Apply k-means clustering technique. Write the code and discuss the process. (20 marks)



```python
def load_images_from_folder(folder_path, img_size=(64, 64)):
    images = []
    labels = []
```

Figure 14: Loading Data Code

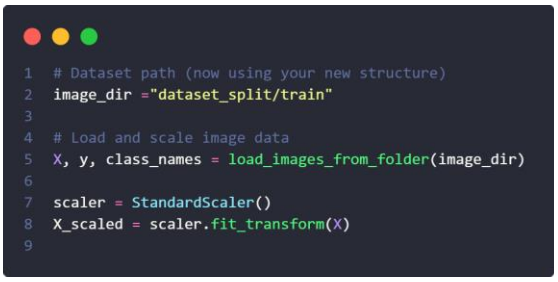
A custom function, **load_images_from_folder()** was developed to automatically load, preprocess and structure the dataset into an array before applying unsupervised learning techniques. The function automatically detects the dataset directory and identifies all the subfolders that represent classes. These folder names are stored in class_names and label mapping each of the classes are generated.



```python
class_names = sorted([d for d in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path, d))])
    label_map = {name: idx for idx, name in enumerate(class_names)}

    for label_name in class_names:
        class_path = os.path.join(folder_path, label_name)

        for filename in os.listdir(class_path):
            img_path = os.path.join(class_path, filename)

            try:
                img = Image.open(img_path).convert("RGB").resize(img_size)
                img_array = np.array(img).flatten()
                images.append(img_array)
                labels.append(label_map[label_name])
            except Exception as e:
                print(f"Error loading {img_path}: {e}")

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

Figure 15: Feature Matrix Building Code

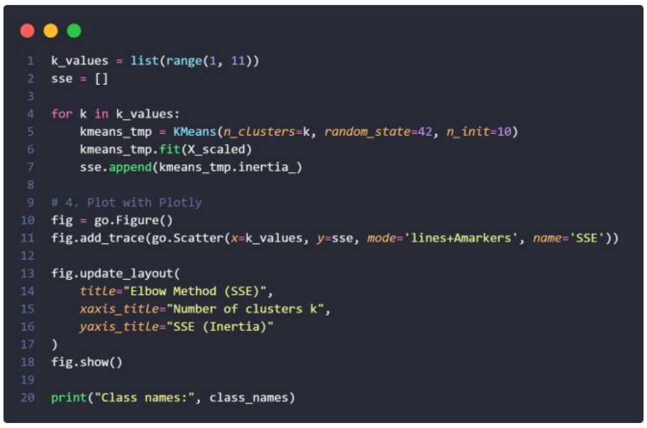
For each class folder, the function iterates through all images and constructs their absolute paths for processing. Each image undergoes RGB conversion, resizing and flatten it into 1D numerical vector that suitable for machine learning. Then, the image vectors are appended into a feature matrix images and its numerical label stored in matrix labels.



```python
# Dataset path (now using your new structure)
image_dir ="dataset_split/train"

# Load and scale image data
X, y, class_names = load_images_from_folder(image_dir)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

```

Figure 16: Dataset Directory Code

The training dataset was loaded using the previous **load_images_from_folder()** function. Each image was resized, converted into numerical pixel vectors and assigned to its correspond class. After that, the feature matrix X was standardized using StandardScaler to ensure that all pixel values had a uniform scale.



```python
k_values = list(range(1, 11))
sse = []

for k in k_values:
    kmeans_tmp = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans_tmp.fit(X_scaled)
    sse.append(kmeans_tmp.inertia_)

# 4. Plot with Plotly
fig = go.Figure()
fig.add_trace(go.Scatter(x=k_values, y=sse, mode='lines+Amarkers', name='SSE'))

fig.update_layout(
    title="Elbow Method (SSE)",
    xaxis_title="Number of clusters k",
    yaxis_title="SSE (Inertia)"
)
fig.show()

print("Class names:", class_names)
```

Figure 17: Elbow Method Code

To identify a suitable number of clusters for K-means, the Elbow Method was applied. The Sum of Squared Error was computed for k=1 to k=10. K-means model was trained for each value of k and its corresponding inertia value was recorded. The SSE values were then visualized using Plotly.

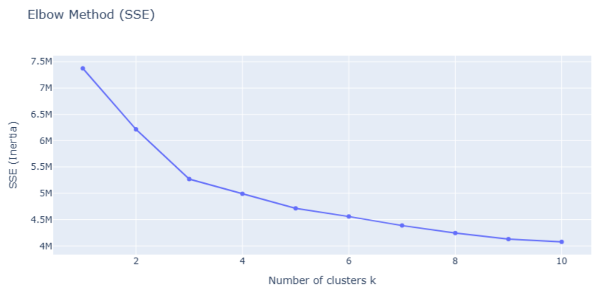
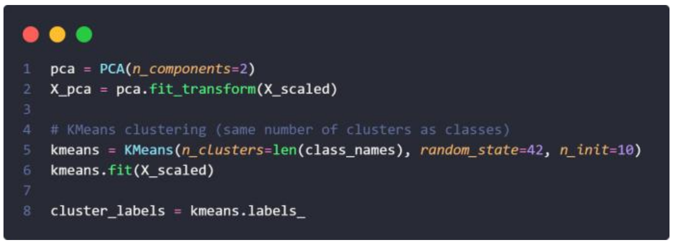


Figure 18: Elbow Method Result

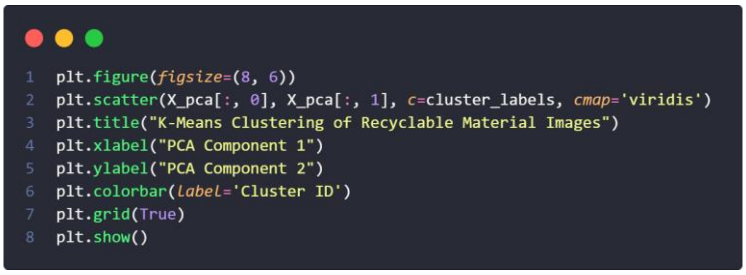
From the figure above, a noticeable "elbow" point indicates where increasing the number of clusters no longer provides significant improvement in reducing SSE. This point reflects the optimal cluster count. Printing the class names allowed verification that the dataset contained the optimal number of classes only which is 3.



```python
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# KMeans clustering (same number of clusters as classes)
kmeans = KMeans(n_clusters=len(class_names), random_state=42, n_init=10)
kmeans.fit(X_scaled)

cluster_labels = kmeans.labels_
```

Figure 19: PCA Usage Code

Since image feature vectors are high-dimensional, Principal Component Analysis (PCA) was used to reduce the feature space to 2 dimensions for visualization purposes. Without the PCA, the point is hard to be clustered in 3D space. The K-means Algorithm was applied using a number of clusters equal to the verified number of classes detected in the dataset. The model produced a label for each image which can represent the cluster it belongs. These cluster labels were then can be used for visual analysis



```python
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels, cmap='viridis')
plt.title("K-Means Clustering of Recyclable Material Images")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.colorbar(label='Cluster ID')
plt.grid(True)
plt.show()
```

Figure 20: Scatter Plot Code

A 2D scatter plot was generated using the PCA-transformed data. Each point represents an image, plotted based on its likelihood to be in the cluster. The colour of each point corresponds to the cluster assigned by K-Means algorithms. This visualization helps illustrate how well the algorithm groups similar images together based purely on pixel similarity without using any class labels during training.

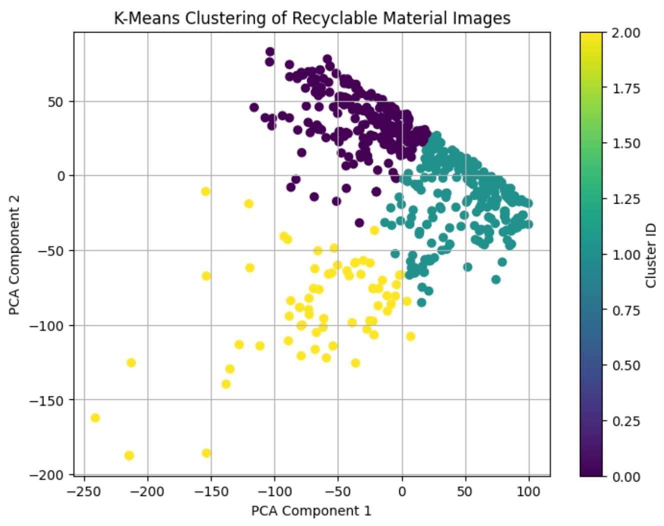
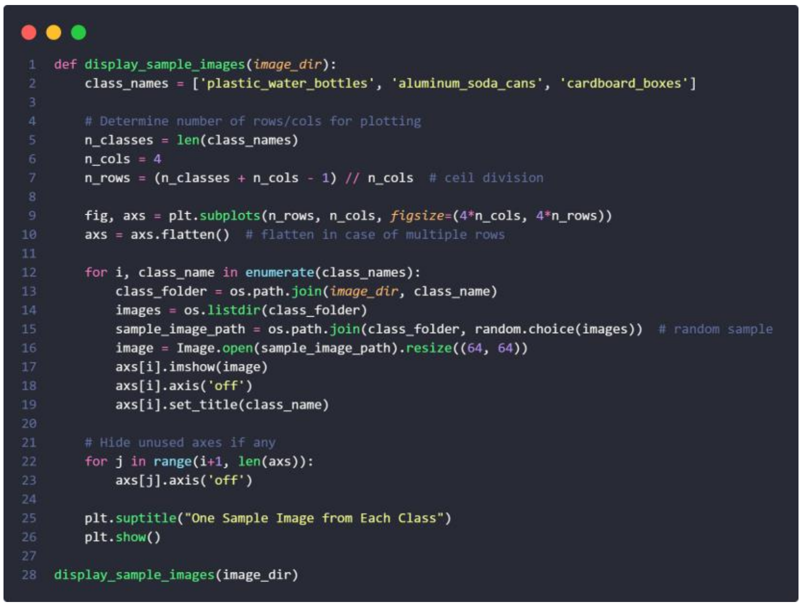


Figure 21: Scatter Plot Result

As we can see from the figure above, the plot shows K-Means grouped the Recyclable Material into three clusters based on visual similarity in the PCA-reduced space. One cluster forms a tight group on the right, another on the upper left and the last spreads toward the lower left region. Each colour basically represents images that share similar features even though no labels were given.

2. Compare the clusters generated by k-means clustering with only one actual image for each class. Discuss. (5 marks)

To compare the clustering outcome with the true visual categories, one random image was extracted from each class. This provides a real visual reference for truly analysing whether the K-Means clusters align with the actual dataset labels.



```python
def display_sample_images(image_dir):
    class_names = ['plastic_water_bottles', 'aluminum_soda_cans', 'cardboard_boxes']

    # Determine number of rows/cols for plotting
    n_classes = len(class_names)
    n_cols = 4
    n_rows = (n_classes + n_cols - 1) // n_cols  # ceil division

    fig, axs = plt.subplots(n_rows, n_cols, figsize=(4*n_cols, 4*n_rows))
    axs = axs.flatten()  # flatten in case of multiple rows

    for i, class_name in enumerate(class_names):
        class_folder = os.path.join(image_dir, class_name)
        images = os.listdir(class_folder)
        sample_image_path = os.path.join(class_folder, random.choice(images))  # random sample
        image = Image.open(sample_image_path).resize((64, 64))
        axs[i].imshow(image)
        axs[i].axis('off')
        axs[i].set_title(class_name)

    # Hide unused axes if any
    for j in range(i+1, len(axs)):
        axs[j].axis('off')

    plt.suptitle("One Sample Image from Each Class")
    plt.show()

display_sample_images(image_dir)
```

Figure 22: Displaying Sample Code

With the **display_sample_images()** function, we can display one sample image extrated from each class in the dataset.

Figure 23: Displayed Image Result

Based on the PCA scatter plot, K-Means was able to group the images into three main clusters. All three images were correctly grouped into separated clusters. This shows that the K-Means was able to capture the main visual differences between the classes like colour and texture. Even with only one image per class, the clustering reflects meaningful separation which suggests the algorithm works well for this dataset.

Part 4: Reflection & Comparative Analysis (25 marks)

Task: Analyse and discuss your findings.

Requirements:

1. Based on the task given on part 2 and 3, what are the differences between the supervised learning technique that you applied with the k-means clustering technique? (5 marks)

Supervised learning (MobileNetV2 classifier) learns from labelled data, meaning the model already knows the correct class for every training image. Because of this, it learns very specific features that distinguish the classes and produces direct class predictions.

K-means clustering is unsupervised, so it learns without labels. It only groups images based on pixel similarity and internal structure, not based on predefined classes. It cannot give class names—only cluster IDs—and it may group images differently than human-defined categories.

So the key difference is: supervised = learning with guidance, unsupervised = discovering patterns on its own.

2. What were the biggest challenges in data collection and data preprocessing? (5 marks)

The most difficult parts were ensuring consistent image quality and uniform preprocessing.

Collecting enough images for each class while keeping the background, lighting, and orientation similar was challenging. Some images had shadows, different sizes, or lower resolution.

During preprocessing, resizing images without distortion, normalizing pixel values, and ensuring correct directory structure for training/testing were the most sensitive steps, because any mistake leads to misalignment in training or clustering.

3. How well does supervised learning perform? Discuss. (5 marks)

The MobileNetV2 model performed strongly because convolutional networks excel at extracting visual patterns.

The confusion matrix showed almost no false predictions, and the classification report indicated high accuracy, precision, recall, and F1-scores across all classes.

This means the model successfully learned the key visual features of each class and generalized well to unseen test images.

4. Discuss the output of the clusters generated by k-means with the actual image? (5 marks)

The PCA scatter plot showed that K-means formed three clear clusters that roughly matched the true dataset classes.

Even though K-means did not use any labels, the algorithm still grouped similar-looking images in the same cluster based on colour, shape, or texture.

When comparing one real sample image per class, each one fell into the correct cluster, meaning the clustering captured the natural visual differences effectively.

However, since k-means relies only on pixel differences, it may still confuse images if two classes share similar colours or textures.

5. What would you improve if you had more time or data? (5 marks)

- With additional time or a larger dataset, several improvements could be made: Increase dataset size to improve model generalization and reduce overfitting.
- Use data augmentation (rotation, flip, brightness adjustment) for stronger supervised learning.
- Experiment with more advanced architectures (ResNet50, EfficientNet).
- Apply deeper dimensionality reduction (PCA with more components or t-SNE) for better clustering visualization.
- Try different unsupervised methods (DBSCAN, hierarchical clustering) to compare clustering quality.
- More data and better variety would make both supervised and unsupervised results more stable and accurate.