Ministry of Education
Umm Al-Qura University
College of Computing

Computer Science & Artificial Intelligence Department
Machine Learning Course
1st Term 2024/2025

Assignment:

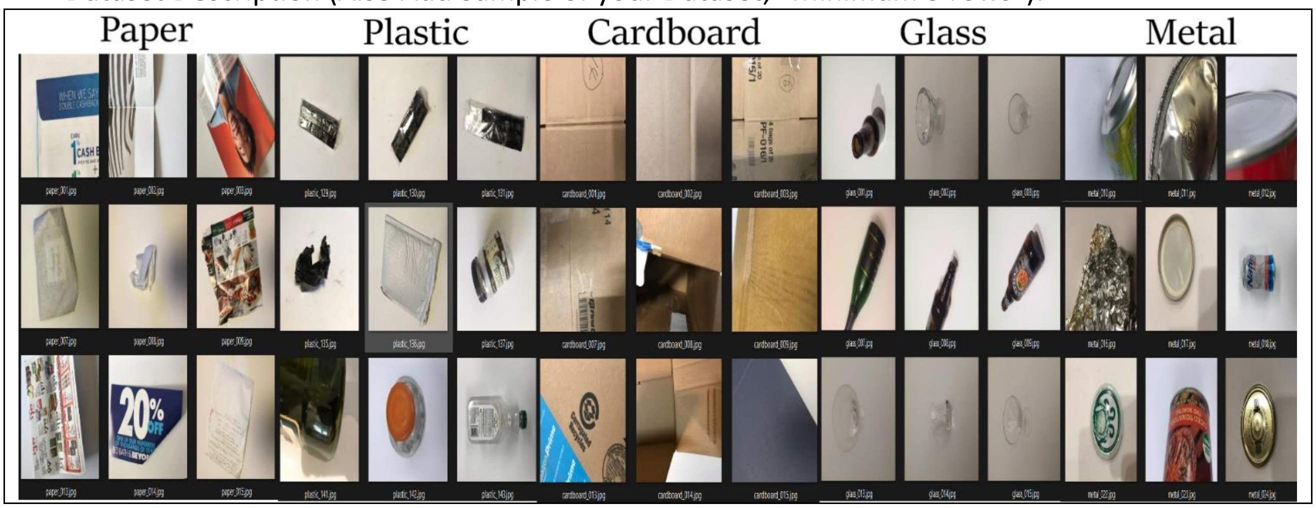
Student Details:

Student ID	Student Name
443009752	Abdulaziz Hussain Abutaleb

Project Title (Maximum 10 words):

Using CNN Image Classification to tell whether or not The shown Garbage is Recyclable.
https://colab.research.google.com/drive/1W2ppf_s5wvIUyxv3GSGb2CqTFEGB7l-D?usp=sharing

Dataset Description (Also Add Sample of your Dataset, "Minimum 5 rows"):



The used Code with explanation (i.e. code with comments):

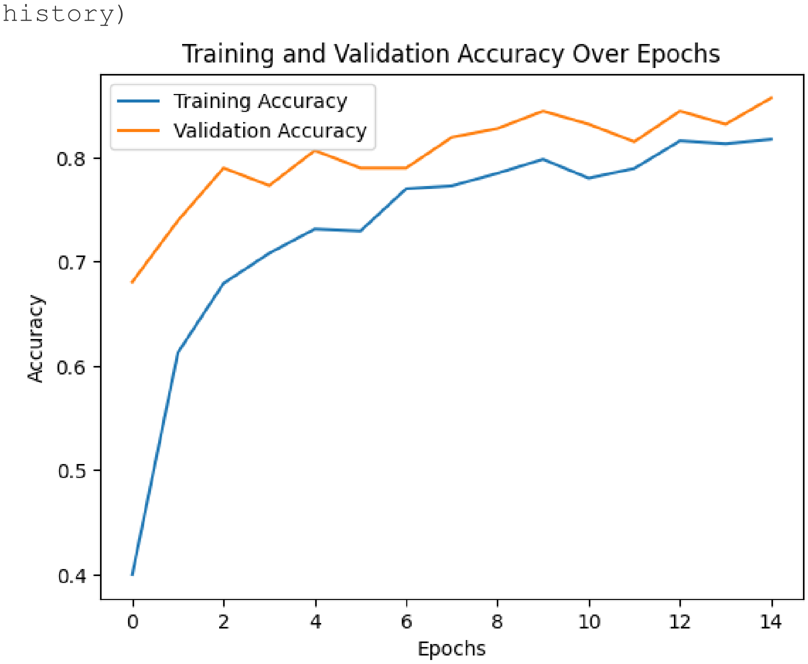
```
# Importing libraries AND Mounting drive
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.applications import MobileNetV2
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive, files
from PIL import Image
drive.mount('/content/drive')

# Define the model using MobileNetV2 (no fine-tuning here, keeping it simple!)
def create_model(num_classes=5):
    # Load MobileNetV2 with pre-trained weights, Without the top classification
layer
    base_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False,
weights='imagenet')
    base_model.trainable = False  # Freeze MobileNetV2 layers so we only train
our custom layers

    # Add extra custom layers on top to adapt MobileNetV2 to classification
task
```

```
    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),  # Flatten the output
        layers.Dense(512, activation='relu'),  # A fully connected layer with
512 units
        layers.Dropout(0.5),  # Dropout to help prevent overfitting
        layers.Dense(num_classes, activation='softmax')  # Output layer for
multi-class classification
    ])

    # Compile the model: using a small learning rate because we're working with
pre-trained weights
    model.compile(optimizer=optimizers.Adam(learning_rate=0.0001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Create the model with the five classes
num_classes = 5
model = create_model(num_classes=num_classes)

# Setting up data augmentation and generators for training and validation
dataset_path = '/content/drive/MyDrive/Recycle Now!'  # Path to images
batch_size = 64  # Setting Batch number
img_height, img_width = 224, 224  # Image dimensions for MobileNetV2

# Setting the Augmantaitions for extra accuracy
train_datagen = ImageDataGenerator(
    rescale=1.0/255,            # Normalize pixel values to the range [0, 1]
for better model performance, as the original range [0, 255] is too large!
    rotation_range=30,          # Randomly rotate images by up to 30 degrees to
make the model robust to slight rotations in the input images
    width_shift_range=0.3,      # Randomly shift images horizontally by up to
30% of the width, adding variability and making the model resilient to
positional changes
    height_shift_range=0.3,     # Randomly shift images vertically by up to 30%
of the height, allowing the model to handle images where objects are slightly
higher or lower
    shear_range=0.3,            # Apply random shearing transformations
(skewing the image), helping the model generalize better with various
perspectives
    zoom_range=0.3,             # Randomly zoom in or out by up to 30%, so the
model can adapt to images that might be closer or farther away
    horizontal_flip=True,       # Randomly flip images horizontally to simulate
mirror-image variations, improving the model's robustness to left-right
orientation changes
    fill_mode='nearest',        # Fill in any missing pixels that result from
transformations by using the nearest pixel values to avoid black gaps or
artifacts
    validation_split=0.1        # Set aside 10% of the data as a validation set
to evaluate model performance without manual data splitting
)

# Load training and validation datasets with the specified augmentations
train_generator = train_datagen.flow_from_directory(
    dataset_path,
```

```
    target_size=(img_height, img_width),  # Resize all images to match
MobileNetV2 input
    batch_size=batch_size,
    class_mode='sparse',  # Use sparse labels (integers) for our multi-class
task
    subset='training'     # This subset is for training
)
validation_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='sparse',
    subset='validation'  # This subset is for validation
)

# Print out the class names to see what we're working with :)
class_names = list(train_generator.class_indices.keys())
print("Class names:", class_names)

# Set up early stopping to avoid overfitting
early_stopping = EarlyStopping(
    monitor='val_accuracy',  # Watch validation accuracy to decide when to stop
    patience=5,              # Stop if no improvement after 5 epochs
    restore_best_weights=True  # Go back to the best weights once training
stops
)

# Training function that shows accuracy and loss for each epoch
def train_model(model, train_data, val_data, epochs=20):
    # Train the model with early stopping and print epoch-wise metrics
    history = model.fit(
        train_data,
        validation_data=val_data,
        epochs=epochs,
        verbose=1,
        callbacks=[early_stopping]  # Pass in early stopping callback to stop
if needed
    )
    return history

# Train the model x epochs as a max, but will stop early if accuracy Stops
Improving
history = train_model(model, train_generator, validation_generator, epochs=15)
```

```
Class names: ['cardboard', 'glass', 'metal', 'paper', 'plastic']
Epoch 1/15
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your
  self._warn_if_super_not_called()
34/34 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - accuracy: 0.3185 - loss: 1.6928 - val_accuracy: 0.6807 - val_loss: 0.8774
Epoch 2/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 38s 913ms/step - accuracy: 0.5927 - loss: 1.0397 - val_accuracy: 0.7395 - val_loss: 0.6863
Epoch 3/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 41s 915ms/step - accuracy: 0.6637 - loss: 0.8510 - val_accuracy: 0.7899 - val_loss: 0.6143
Epoch 4/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 41s 924ms/step - accuracy: 0.7035 - loss: 0.7458 - val_accuracy: 0.7731 - val_loss: 0.5924
Epoch 5/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 37s 911ms/step - accuracy: 0.7182 - loss: 0.6959 - val_accuracy: 0.8067 - val_loss: 0.5461
Epoch 6/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 38s 900ms/step - accuracy: 0.7132 - loss: 0.6835 - val_accuracy: 0.7899 - val_loss: 0.5342
Epoch 7/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 41s 889ms/step - accuracy: 0.7559 - loss: 0.6207 - val_accuracy: 0.7899 - val_loss: 0.5247
Epoch 8/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.7705 - loss: 0.5894 - val_accuracy: 0.8193 - val_loss: 0.4632
Epoch 9/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 46s 1s/step - accuracy: 0.7930 - loss: 0.5544 - val_accuracy: 0.8277 - val_loss: 0.4725
Epoch 10/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 38s 917ms/step - accuracy: 0.7900 - loss: 0.5585 - val_accuracy: 0.8445 - val_loss: 0.4684
Epoch 11/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.7819 - loss: 0.5496 - val_accuracy: 0.8319 - val_loss: 0.4264
Epoch 12/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 78s 922ms/step - accuracy: 0.7979 - loss: 0.5097 - val_accuracy: 0.8151 - val_loss: 0.4521
Epoch 13/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - accuracy: 0.8053 - loss: 0.5186 - val_accuracy: 0.8445 - val_loss: 0.3912
Epoch 14/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 37s 917ms/step - accuracy: 0.8264 - loss: 0.4678 - val_accuracy: 0.8319 - val_loss: 0.4204
Epoch 15/15
34/34 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - accuracy: 0.8088 - loss: 0.4826 - val_accuracy: 0.8571 - val_loss: 0.4045
```

```python
# Plotting the accuracy over each epoch
def plot_accuracy(history):
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy Over Epochs')
    plt.legend()
    plt.show()
# Call the function to plot accuracy
plot_accuracy(history)
```

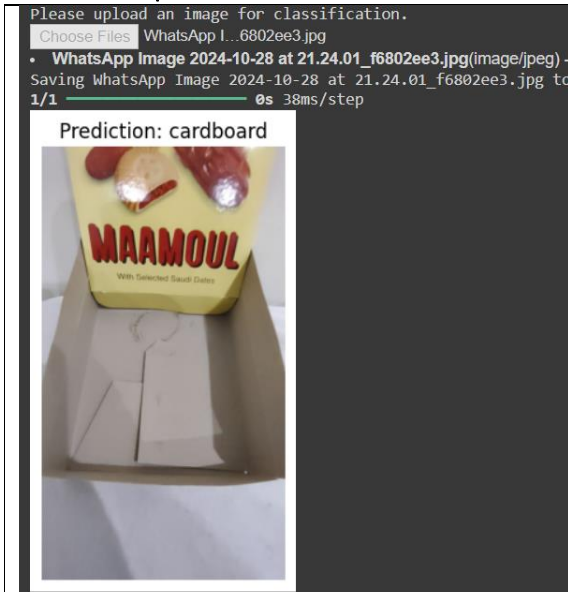


Evaluation Metrics (with explanation):

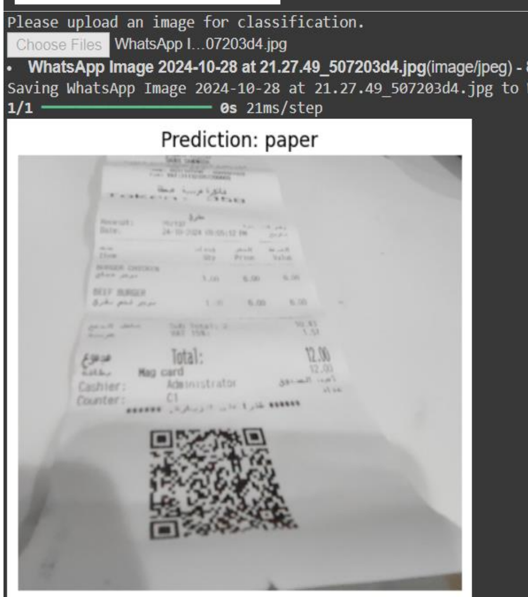
```
# Function to evaluate the model on the validation set and print classification
report
def evaluate_model_with_report(model, val_data, class_names):
    # Get true labels and predictions
    true_labels = []
    predictions = []

    # Limit the loop to the exact number of validation batches
    steps = len(val_data)
    for i, (images, labels) in enumerate(val_data):
        if i >= steps:  # Stop after going through all batches in validation
set
            break
        preds = model.predict(images)
        preds = np.argmax(preds, axis=1)
        predictions.extend(preds)
        true_labels.extend(labels)

    # Print the classification report
    print("\nClassification Report:\n")
    print(classification_report(true_labels, predictions,
target_names=class_names))

    # Calculate and print the accuracy as a percentage
    correct_predictions = sum(np.array(true_labels) == np.array(predictions))
    accuracy = (correct_predictions / len(true_labels)) * 100
    print(f"\nOverall Accuracy: {accuracy:.2f}%")

# Evaluate the model on the validation set after training
evaluate_model_with_report(model, validation_generator, class_names)
```

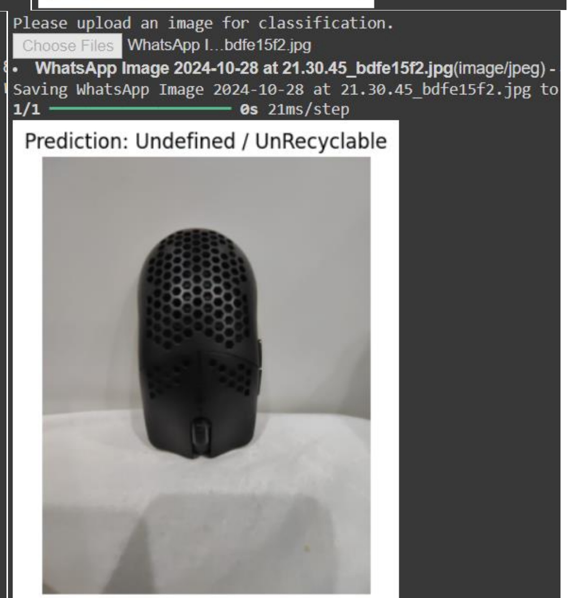
```
Classification Report:

              precision    recall  f1-score   support

   cardboard       0.93      0.97      0.95        40
       glass       0.80      0.82      0.81        50
       metal       0.88      0.85      0.86        41
       paper       0.93      0.93      0.93        59
     plastic       0.80      0.77      0.79        48

    accuracy                           0.87       238
   macro avg       0.87      0.87      0.87       238
weighted avg       0.87      0.87      0.87       238


Overall Accuracy: 86.97%
```

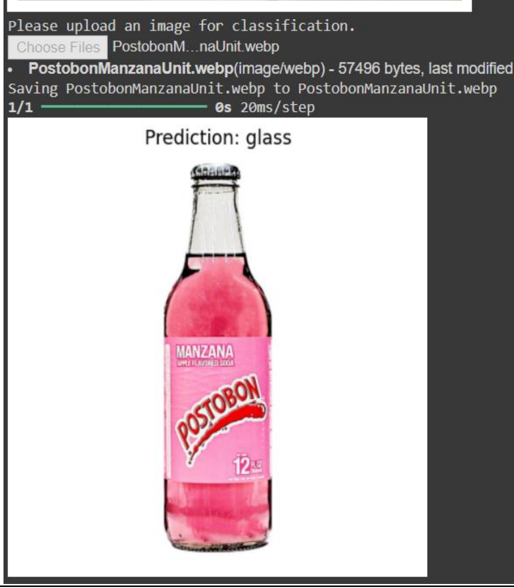
Examples:

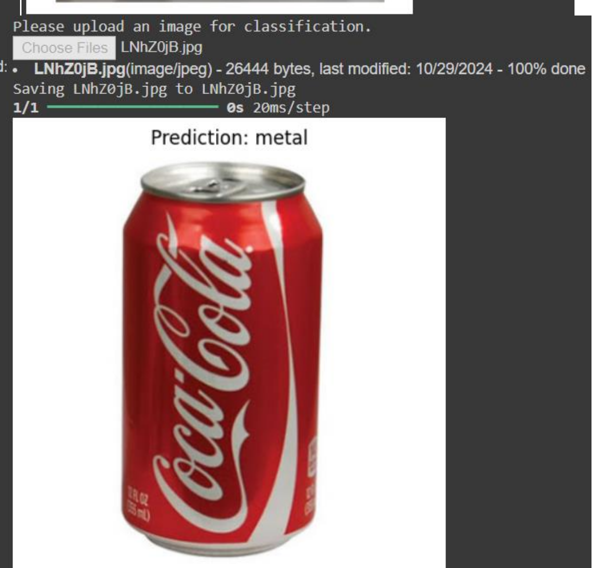

Please upload an image for classification.
Choose Files WhatsApp I...6802ee3.jpg
WhatsApp Image 2024-10-28 at 21.24.01_f6802ee3.jpg(image/jpeg) -
Saving WhatsApp Image 2024-10-28 at 21.24.01_f6802ee3.jpg to
1/1 0s 38ms/step
Prediction: cardboard
MAAMOUL


Please upload an image for classification.
Choose Files WhatsApp I...79eb916.jpg
WhatsApp Image 2024-10-28 at 21.28.27_c79eb916.jpg(image/jpeg) -
Saving WhatsApp Image 2024-10-28 at 21.28.27_c79eb916.jpg to
1/1 0s 21ms/step
Prediction: plastic


Please upload an image for classification.
Choose Files WhatsApp I...07203d4.jpg
WhatsApp Image 2024-10-28 at 21.27.49_507203d4.jpg(image/jpeg) -
Saving WhatsApp Image 2024-10-28 at 21.27.49_507203d4.jpg to
1/1 0s 21ms/step
Prediction: paper


Please upload an image for classification.
Choose Files WhatsApp I...bdfe15f2.jpg
WhatsApp Image 2024-10-28 at 21.30.45_bdfe15f2.jpg(image/jpeg) -
Saving WhatsApp Image 2024-10-28 at 21.30.45_bdfe15f2.jpg to
1/1 0s 21ms/step
Prediction: Undefined / UnRecyclable


Please upload an image for classification.
Choose Files PostobonM...naUnit.webp
PostobonManzanaUnit.webp(image/webp) - 57496 bytes, last modified:
Saving PostobonManzanaUnit.webp to PostobonManzanaUnit.webp
1/1 0s 20ms/step
Prediction: glass
MANZANA
POSTOBON


Please upload an image for classification.
Choose Files LNhZ0jB.jpg
LNhZ0jB.jpg(image/jpeg) - 26444 bytes, last modified: 10/29/2024 - 100% done
Saving LNhZ0jB.jpg to LNhZ0jB.jpg
1/1 0s 20ms/step
Prediction: metal
Coca-Cola