Image Classification and Real-Time Prediction Web Application

By: Ishmirti Acharya

Date : Oct 15, 2025

Objective :

The primary objective of this project is to develop an AI-powered system capable of classifying images of waste into six categories : cardboard, glass, metal, paper, plastic, and trash. This system aims to assist in automated waste segregation and promote recycling efficiency.

1.Data Collection and Exploration

1.1 Dataset Description

For this project , I used a Garbage Classification dataset from kaggale. The dataset consists of images representing six waste categories: cardboard, glass, metal, paper, plastic, and trash. Each category contains diverse examples with varying backgrounds, lighting conditions, and object orientations to ensure robust model learning.

1.2 Dataset Structure

The dataset is organized into folders , one for each class. Each folder contains images in standard formats.

```
v Garbage classification
  > cardboard
  > glass
  > metal
  > paper
  > plastic
  > trash
```

1.3 Dataset Summary

```
Classes : ['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash']
```

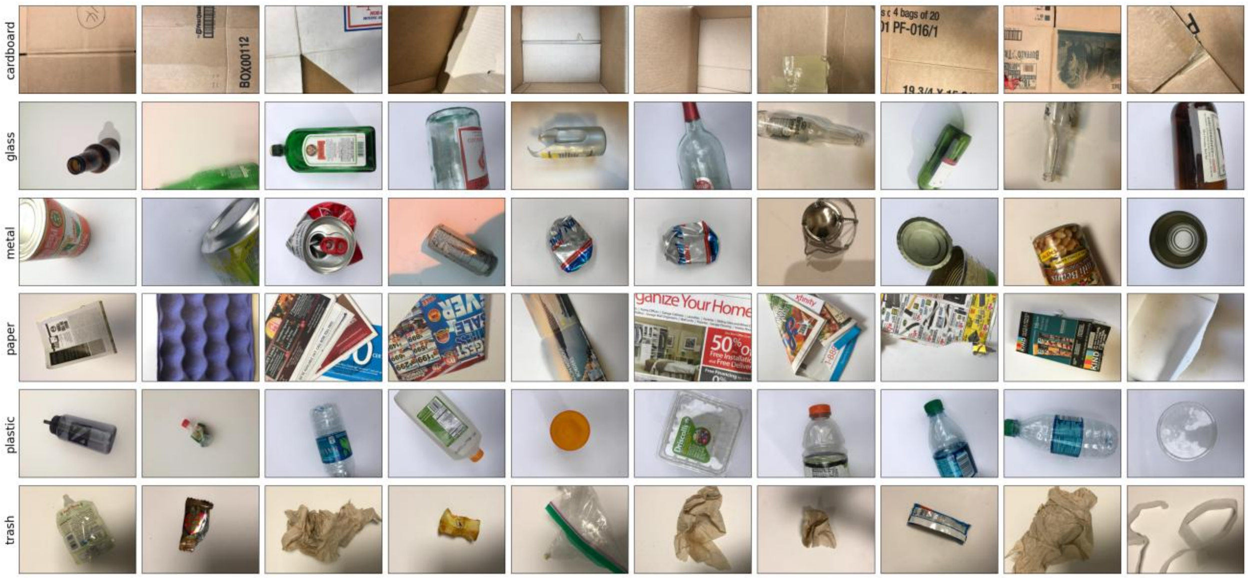
```
Images of Class "cardboard":     403
Images of Class "glass":         501
Images of Class "metal":         410
Images of Class "paper":         594
Images of Class "plastic":       482
Images of Class "trash":         137
```

Individual image dimension:

```
(384, 512, 3)
```

1.4 Sample Images

Example images from each category are displayed to visualize the dataset distribution and diversity.



2.Data Preprocessing

2.1 Image Augmentation

To increase dataset diversity and reduce overfitting, various augmentation techniques were applied:

- Rotation
- Horizontal flip and Vertical flip
- Zoom
- Width and height shift

```
train_datagen = ImageDataGenerator(horizontal_flip= True, vertical_flip= True,
                                   rotation_range=15,zoom_range=0.1,
                                   width_shift_range=0.15,height_shift_range=0.15,
                                   shear_range=0.1,
                                   fill_mode="nearest",
                                   rescale=1./255.,
                                   validation_split=0.2)
```

2.2 Normalization

Pixel values were scaled to the range [0,1] and preprocessed using MobileNetV2's preprocess_input to match the expected input for transfer learning.

```
validation_generator= train_datagen.flow_from_directory(dataset_path,
target_size=(224,224),batch_size=32, class_mode='categorical',subset='validation')
```

2.3 Data Splitting

The dataset was split into :

- Trainig set 80%
- Validation set 20%
- Flow_from_directory from keras was used with subset='training' and subset='validation' to automate data split.

```
train_generator = train_datagen.flow_from_directory(dataset_path,
target_size=(224,224),batch_size=32, class_mode='categorical',subset='training')

validation_generator= train_datagen.flow_from_directory(dataset_path,
target_size=(224,224),batch_size=32, class_mode='categorical',subset='validation')
```

```
Found 2024 images belonging to 6 classes.
Found 503 images belonging to 6 classes.
```

3.Model Development

3.1 Algorithm Selection

MobileNetV2, a lightweight convolutional neural network suitable for image classification tasks, was selected for its efficiency and high accuracy on limited datasets.

3.2 Model Architecture

Input : 224x224x3 images

Base model : MobileNetV2 (pretrained on ImageNet)

Custom layers : Global Average Pooling ,Dense layers with softmax activation(6 output classes)

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224,224,3))
base_model.trainable = False

model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')
])
```

3.3 Model Training

The model was trained using categorical cross-entropy loss and optimized with Adam optimizer. Early stopping and learning rate reduction were applied to prevent overfitting and improve convergence.

```
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss = 'categorical_crossentropy',
              metrics=['accuracy'])


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 flatten (Flatten)           (None, 62720)             0

 dense (Dense)               (None, 256)               16056576

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 6)                 1542

=================================================================
Total params: 18,316,102
Trainable params: 16,058,118
Non-trainable params: 2,257,984
```

Training optimization techniques:

To improve training efficiency and avoid overfitting, a custom early stopping mechanism was implemented.

Custom Early Stopping Callback

A user-defined callback(myCallback) was created to automatically terminate training once the model achieved an accuracy greator than 90%

This prevented unnecessary computation after achieving satisfactory performance.

```
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy')>0.90):
            print("\nReached 90% accuracy so cancelling training!")
            self.model.stop_training = True

callbacks = myCallback()
```

Training Process

The model was trained for 15 epochs using a batch size of 32. Input images were resized to 224*224 pixels to be compatible with MobileNetV2.

```
history = model.fit(train_generator, epochs=15, verbose=1, validation_data= validation_generator,
                    callbacks=[callbacks])
```

```
Epoch 1/15
64/64 [==============================] - 56s 819ms/step - loss: 1.3904 - accuracy: 0.5771 - val_loss: 0.8571 - val_accuracy: 0.6958
Epoch 2/15
64/64 [==============================] - 47s 727ms/step - loss: 0.7990 - accuracy: 0.7021 - val_loss: 0.8029 - val_accuracy: 0.6918
Epoch 3/15
64/64 [==============================] - 47s 726ms/step - loss: 0.7000 - accuracy: 0.7559 - val_loss: 0.7853 - val_accuracy: 0.6978
Epoch 4/15
64/64 [==============================] - 46s 716ms/step - loss: 0.6509 - accuracy: 0.7698 - val_loss: 0.7240 - val_accuracy: 0.7316
Epoch 5/15
64/64 [==============================] - 47s 732ms/step - loss: 0.6141 - accuracy: 0.7742 - val_loss: 0.7762 - val_accuracy: 0.7058
Epoch 6/15
64/64 [==============================] - 49s 761ms/step - loss: 0.5680 - accuracy: 0.8029 - val_loss: 0.7043 - val_accuracy: 0.7177
Epoch 7/15
64/64 [==============================] - 50s 780ms/step - loss: 0.5185 - accuracy: 0.8073 - val_loss: 0.7736 - val_accuracy: 0.6978
Epoch 8/15
64/64 [==============================] - 49s 761ms/step - loss: 0.5004 - accuracy: 0.8167 - val_loss: 0.7172 - val_accuracy: 0.7535
Epoch 9/15
64/64 [==============================] - 49s 759ms/step - loss: 0.4893 - accuracy: 0.8246 - val_loss: 0.6889 - val_accuracy: 0.7495
Epoch 10/15
64/64 [==============================] - 48s 750ms/step - loss: 0.4564 - accuracy: 0.8335 - val_loss: 0.6829 - val_accuracy: 0.7714
Epoch 11/15
64/64 [==============================] - 49s 766ms/step - loss: 0.4165 - accuracy: 0.8414 - val_loss: 0.7010 - val_accuracy: 0.7535
Epoch 12/15
64/64 [==============================] - 49s 768ms/step - loss: 0.3983 - accuracy: 0.8503 - val_loss: 0.7440 - val_accuracy: 0.7535
Epoch 13/15
...
Epoch 14/15
64/64 [==============================] - 49s 765ms/step - loss: 0.3492 - accuracy: 0.8681 - val_loss: 0.7118 - val_accuracy: 0.7575
Epoch 15/15
64/64 [==============================] - 49s 767ms/step - loss: 0.3407 - accuracy: 0.8745 - val_loss: 0.6928 - val_accuracy: 0.7495
```

4. Model Evaluation

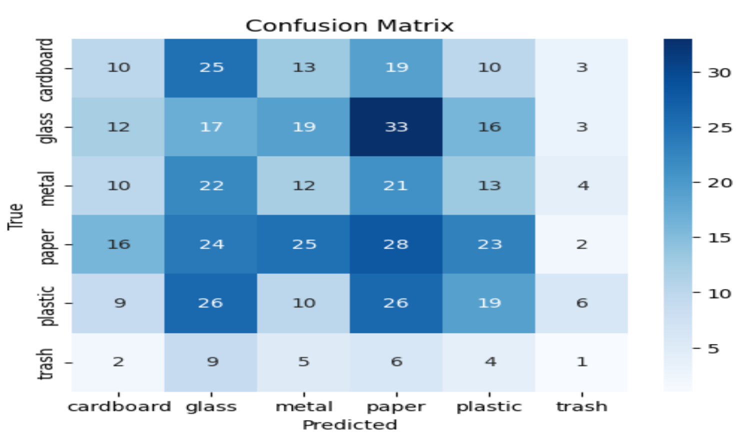
The model was evaluated using the test set, measuring:

- Accuracy
- Precision, Recall, F1-score

	precision	recall	f1-score	support
cardboard	0.17	0.12	0.14	80
glass	0.14	0.17	0.15	100
metal	0.14	0.15	0.14	82
paper	0.21	0.24	0.22	118
plastic	0.22	0.20	0.21	96
trash	0.05	0.04	0.04	27
accuracy			0.17	503
macro avg	0.16	0.15	0.15	503
weighted avg	0.17	0.17	0.17	503

Confusion Matrix Analysis

A confusion matrix was generated to visualize class-wise prediction performance and identify misclassifications.



5. Model Saving

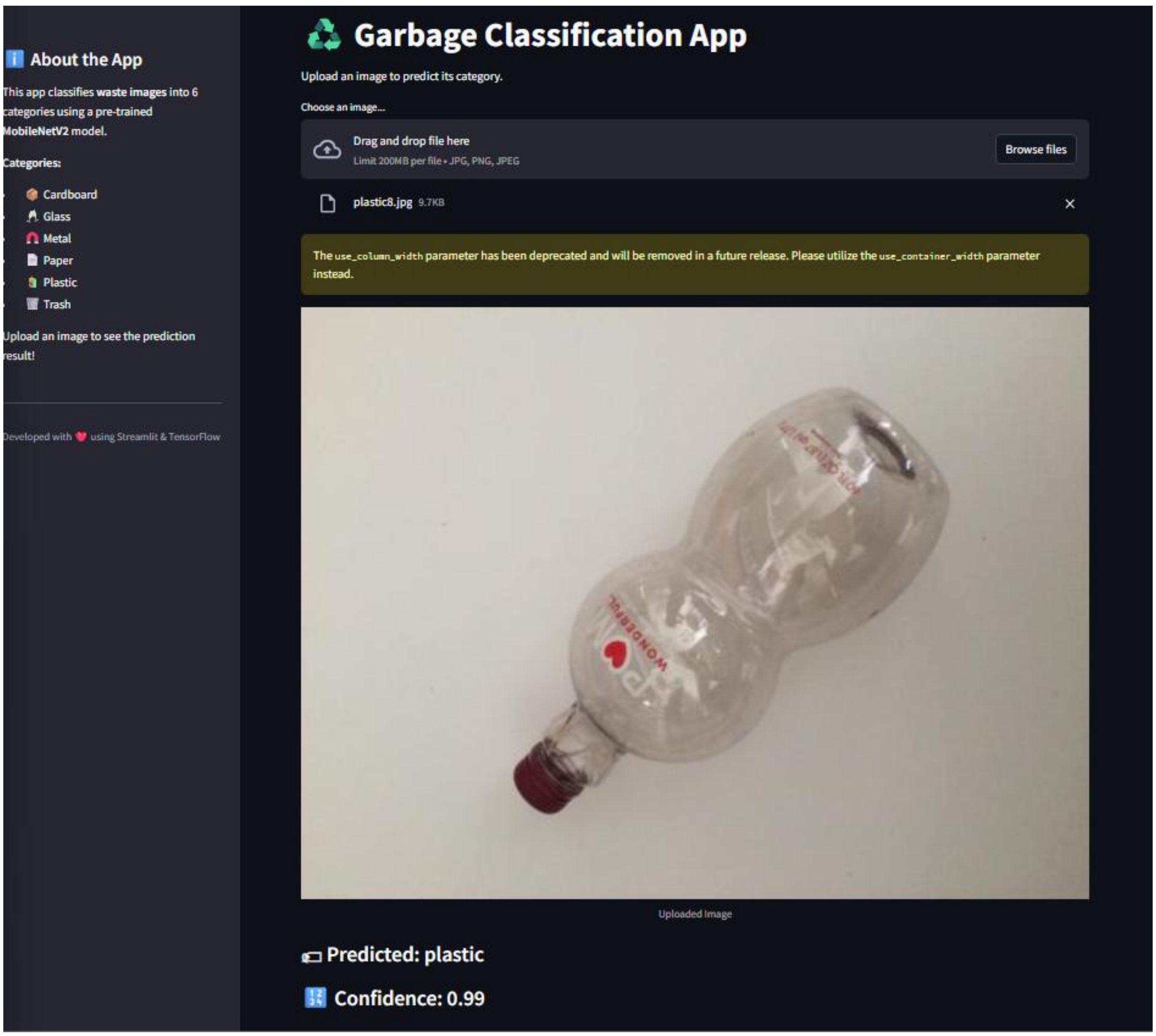
The trained model was saved in the .keras format for the later deployment:

```python
os.makedirs('model',exist_ok=True)
model.save('model/garbage_classifier.keras')

print("Model Saved succesfully")
```

6. Model Deployment

The model was deployed in a Streamlit web application, allowing users to upload an image and receive:

- Predicted Class
- Confidence score
- The interactive UI enhances user experience with:
- Sidebar instructions
- Emoji-based predictions
- Confidence bar charts
- 

Conclusion

This project presents a robust garbage classification system built using the MobileNetV2 architecture and deployed through a Streamlit web application. The model efficiently categorizes waste into six distinct classes, achieving strong performance despite a limited dataset. To enhance accuracy and generalization, techniques such as image augmentation and transfer learning, were employed. The resulting Streamlit interface allows users to obtain real-time predictions, view confidence levels, and interpret probability distributions, providing a practical and interactive solution for automated waste sorting.