جامعة ٦ أكتوبر
October 6 University

Image Processing

Garbage Classification

Prof.Ass. Hossam Albehery

Team Member

- 202201998 احمد اسماعيل السيد السيد عزام
- 202202448 احمد محمد حسين طه
- 202202718 احمد سامي عبداللطيف
- 202201949 احمد عصام صابر احمد
- 202205969 احمد رافت محمود غريب
- 202203366 احمد رضا احمد علي

Garbage Classification Using Deep Learning

Information Systems and Computer Science Faculty
October 6 University
May 12, 2025

Brief Introduction

This project focuses on image classification using deep learning techniques implemented in TensorFlow and Keras. It involves loading and preprocessing image data, building and training a convolutional neural network (CNN), and evaluating its performance. The model is designed to classify images into different categories accurately. Visualization tools are used to analyze training progress and model predictions.

Problem Definition

The objective of this project is to build a deep learning model that can classify images into specific categories. By training a convolutional neural network (CNN) on labeled image data, the model learns to recognize visual patterns and features. The main challenge is to ensure high accuracy and reliable performance on unseen test images.

Proposed Methodology: System Data Flow

1.The process follows a standard image classification pipeline with deep learning:
Data Loading: Import image dataset organized by class folders.

2.**Data Preprocessing**: Resize images, normalize pixel values, and split into training and validation sets.

3.**Model Design**: Build a CNN architecture using Keras with layers like Conv2D, MaxPooling, Flatten, and Dense.

4.**Compilation**: Configure the model with loss function, optimizer (e.g., Adam), and evaluation metrics.

5.**Training**: Train the model on the training dataset while validating on the validation set.

6. **Evaluation & Prediction**: Evaluate model accuracy and loss, visualize performance, and make predictions on new data.

Core Functionality Examples

Function Block 1

```python
# Standard library imports
import os
import random
import shutil  # For file operations and directory management
import warnings

# Data manipulation and visualization
import pandas as pd  # For data frame operations
import numpy as np  # Numerical computing
import seaborn as sns  # Statistical visualization
import matplotlib.pyplot as plt  # Plotting and image display

# Image processing and computer vision
import cv2  # OpenCV for image manipulation
from tqdm import tqdm  # Progress bars for loops

# Machine learning and deep learning
from sklearn.model_selection import train_test_split  # Data splitting
from sklearn.metrics import classification_report, confusion_matrix  # Model evaluation
import tensorflow as tf  # Deep learning framework
from keras.layers import (  # Neural network components
    Conv2D, MaxPooling2D, BatchNormalization, Dropout,
    Flatten, Dense, Input, Rescaling, Resizing
)
from keras.models import Sequential  # Linear stack of layers
from tensorflow.keras.preprocessing.image import (  # Image data handling
    ImageDataGenerator, array_to_img, img_to_array, load_img
)
from tensorflow.keras.applications import MobileNetV2  # Pre-trained model

# Suppress warnings to keep output clean
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

# Set ggplot style for matplotlib visualizations
plt.style.use('ggplot')
```

Function Block 2

```python
def DataFrameSpliting(df, ratio, classesList):

    # Initialize empty DataFrames for training and testing data
    trainDf = pd.DataFrame(columns=['imgPath', 'label'])
    testDf = pd.DataFrame(columns=['imgPath', 'label'])

    # Process each class independently to maintain ratio
    for clas in classesList:
        # Filter DataFrame for current class
        tempDf = df[df['label'] == clas]

        # Calculate split point - NOTE: Using int() truncates decimal values
        lastIndex = int(len(tempDf) * ratio)  # Training cutoff index

        # Split into training and testing portions
        trainClassDf = tempDf[:lastIndex]    # First 'ratio' portion -> training
        testClassDf = tempDf[lastIndex:]     # Remaining portion -> testing

        # Aggregate class-specific DataFrames
        trainDf = pd.concat([trainDf, trainClassDf], axis=0)
        testDf = pd.concat([testDf, testClassDf], axis=0)

    # Final shuffle and index reset for both sets
    return (
        trainDf.sample(frac=1).reset_index(drop=True),  # Shuffle training data
        testDf.sample(frac=1).reset_index(drop=True)    # Shuffle testing data
    )
```

Function Block 3

```python
"""
Create an ImageDataGenerator for training data with real-time data augmentation.
Includes rescaling, zooming, flipping, and rotation to enhance model generalization.
"""

datagenTrain = ImageDataGenerator(
    rescale=1./255,              # Normalize pixel values to [0, 1] range
    zoom_range=(1.0, 1.2),       # Randomly zoom in on images up to 20%
    horizontal_flip=True,        # Randomly flip images horizontally
    vertical_flip=True,          # Randomly flip images vertically
    rotation_range=45,          # Randomly rotate images up to 45 degrees
)
```

Function Block 4

```
"""
Create training and testing data generators from DataFrames using Keras' ImageDataGenerator.
Applies data augmentation to the training data and only rescaling to the test data.
"""

IMG_SIZE = (224, 224)  # Define target size for input images

# Create training data generator with augmentation
trainGenerator = datagenTrain.flow_from_dataframe(
    trainDf,                    # DataFrame containing training image paths and labels
    x_col='imgPath',            # Column name for image file paths
    y_col='label',              # Column name for image labels
    target_size=IMG_SIZE,       # Resize images to (224, 224)
    batch_size=64,              # Generate 64 augmented images per batch
    class_mode='categorical'    # Use categorical mode for multi-class classification
)

# Create test data generator (only rescaling, no augmentation)
datagenTest = ImageDataGenerator(rescale=1./255)

testGenerator = datagenTest.flow_from_dataframe(
    testDf,                     # DataFrame containing test image paths and labels
    x_col='imgPath',            # Column name for image file paths
    y_col='label',              # Column name for image labels
    target_size=IMG_SIZE,       # Resize images to (224, 224)
    batch_size=8,               # Generate 8 images per batch
    class_mode='categorical',   # Use categorical mode for multi-class classification
    shuffle=False               # Do not shuffle test data (important for evaluation)
)

# Display the number of samples in the training and testing sets
print(f"Training set size: {trainGenerator.samples}")
print(f"Testing set size: {testGenerator.samples}")
```

Function Block 5

```
"""
Build a transfer learning image classification model using MobileNetV2 as the base.
The model is constructed on GPU with additional custom dense layers for classification.
Image preprocessing (resizing/rescaling) is assumed to be handled by the data generators.
"""

with tf.device('/GPU:0'):  # Use GPU for model construction and training
    Model = Sequential([

        # Resizing(IMG_SIZE),      # Skipped: resizing is done by the data generator
        # Rescaling(1./255),       # Skipped: rescaling is also done by the data generator

        MobileNetV2(
            weights='imagenet',        # Load pre-trained weights from ImageNet
            include_top=False,         # Exclude the top classification layers
            input_shape=(224, 224, 3)  # Input image shape
        ),

        Flatten(),                     # Flatten the output from convolutional base

        Dense(64, activation='relu'),  # Add a fully connected layer with ReLU activation
    ])
```

Function Block 6

```
"""
Train the model using the training and testing data generators, with early stopping
to prevent overfitting. The model will stop training if validation accuracy doesn't improve
for 4 consecutive epochs.
"""

history = Model.fit(
    trainGenerator,                                 # Training data generator
    validation_data=testGenerator,                  # Validation data generator
    epochs=50,                                      # Train for up to 50 epochs
    # batch_size=64,                               # Batch size is already defined in the generator
    verbose=1,                                      # Print progress during training
    callbacks=[tf.keras.callbacks.EarlyStopping(    # Early stopping callback
        patience=4,                                 # Stop training after 4 epochs with no improvement
        monitor='val_accuracy',                     # Monitor validation accuracy
        restore_best_weights=True                   # Restore model weights from the best epoch
    )]
)
```

Code & Libraries Used

- **os, pandas, numpy** for data handling.
- **seaborn, matplotlib** for visualization.
- **cv2** (OpenCV) for image processing.
- **tensorflow and keras** for building and training convolutional neural networks (CNNs).
- **sklearn** for data splitting and evaluation metrics.
- **ImageDataGenerator** from Keras for image augmentation