REAL-TIME WASTE CLASSIFICATION
MACHINE LEARNING Assignment 8 - MINI PROJECT
SY COMP B2 (2023-24)

UCE2022532 Ananya Kale
UCE2022537 Saniya Karambelkar
UCE2022540 Radha Katdare

---

1) Problem statement

Real-time waste classification into biodegradable and non-biodegradable using the tensorflow library by making a CNN model.

---

2) Introduction

We have identified this problem and developed this solution as a proof of concept for the real-life challenge of waste segregation faced by major cities across the world. In India, approximately 50 percent of waste remains unsegregated and hence untreated.

⦿ **Solution Overview:**

Admin end application which can be utilized at the waste segregation unit,which classifies waste into two broad categories, biodegradable and non-biodegradable.It leverages TensorFlow library to achieve this functionality.
The solution is executed in the following manner:

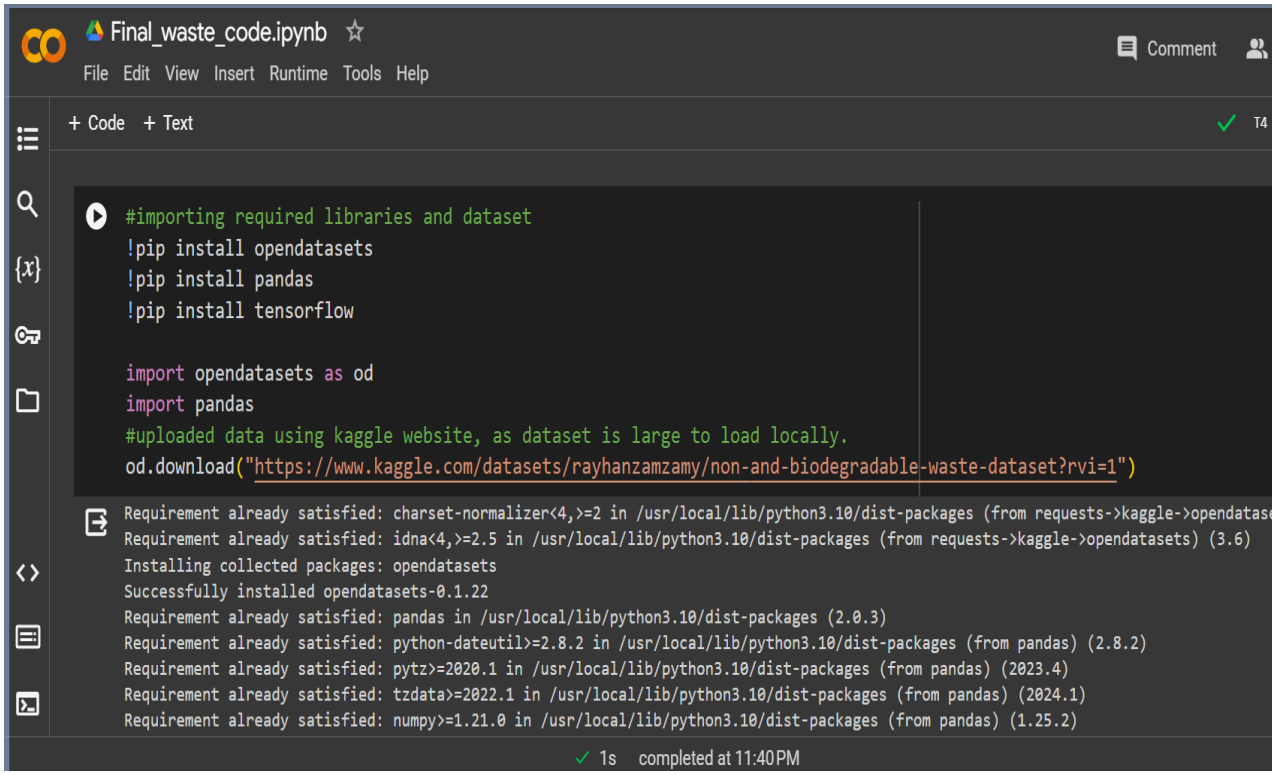
1. Data Acquisition and Preprocessing (using 2GB data containing over 15000 images)
2. Model Training and Building (CNN model)
3. Live Image Capture (use of pre trained downloaded weights and google colab image capture feature)
4. Prediction (Binary classification into non-biodegradable and biodegradable.

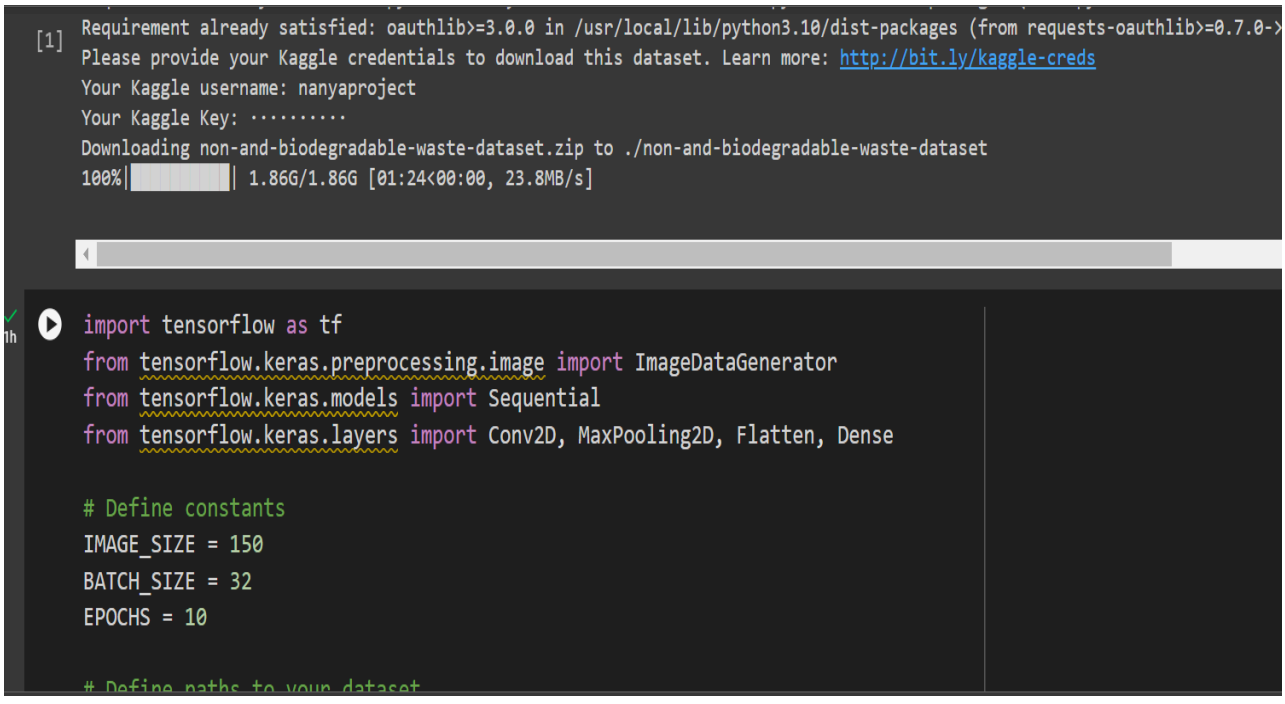
---

3)Data set information (link,few data samples etc)

Dataset link (kaggle)

4) Code and output CODE:MODEL:





```
[2]
# Define paths to your dataset
train_dir = '/content/non-and-biodegradable-waste-dataset/TRAIN.2'
validation_dir = '/content/non-and-biodegradable-waste-dataset/TEST'

# Create image data generators with data augmentation for training images
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

validation_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches using the generators
```

```
[2]
# Flow training images in batches using the generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='binary'
)

# Create a CNN model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)),
```

```
# Create a CNN model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

# Train the model
history = model.fit(
```

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

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.n // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.n // BATCH_SIZE
)


# Save the model
#model.save('/content/drive/MyDrive/wasteClassificationModel.h5')
```

```
Found 59922 images belonging to 2 classes.
Found 16726 images belonging to 2 classes.
```

+ Code + Text

T4 RAM Disk

```
[2] Found 16726 images belonging to 2 classes.
Epoch 1/10
1872/1872 [==============================] - 455s 239ms/step - loss: 0.3202 - accuracy: 0.8681 - val_loss: 0.4457 - val_accuracy: 0.7753
Epoch 2/10
1872/1872 [==============================] - 426s 228ms/step - loss: 0.2526 - accuracy: 0.8993 - val_loss: 0.5225 - val_accuracy: 0.7611
Epoch 3/10
1872/1872 [==============================] - 415s 222ms/step - loss: 0.2328 - accuracy: 0.9079 - val_loss: 0.4327 - val_accuracy: 0.8063
Epoch 4/10
1872/1872 [==============================] - 417s 223ms/step - loss: 0.2208 - accuracy: 0.9134 - val_loss: 0.4227 - val_accuracy: 0.8201
Epoch 5/10
1872/1872 [==============================] - 413s 221ms/step - loss: 0.2078 - accuracy: 0.9192 - val_loss: 0.4388 - val_accuracy: 0.8237
Epoch 6/10
1872/1872 [==============================] - 418s 223ms/step - loss: 0.2004 - accuracy: 0.9214 - val_loss: 0.8266 - val_accuracy: 0.7371
Epoch 7/10
1872/1872 [==============================] - 418s 223ms/step - loss: 0.1965 - accuracy: 0.9247 - val_loss: 0.3780 - val_accuracy: 0.8554
Epoch 8/10
1872/1872 [==============================] - 419s 224ms/step - loss: 0.1905 - accuracy: 0.9269 - val_loss: 0.8527 - val_accuracy: 0.7282
Epoch 9/10
1872/1872 [==============================] - 416s 222ms/step - loss: 0.1841 - accuracy: 0.9294 - val_loss: 0.4606 - val_accuracy: 0.8211
Epoch 10/10
1872/1872 [==============================] - 417s 223ms/step - loss: 0.1805 - accuracy: 0.9295 - val_loss: 0.5304 - val_accuracy: 0.8033
```

```
Epoch 10/10
1872/1872 [==============================] - 417s 223ms/step - loss: 0.1805 - accuracy: 0.9295 - val_loss: 0.5304 - val_accuracy: 0.8033
```

```
[4] # Evaluate the model on the validation data
validation_loss, validation_accuracy = model.evaluate(validation_generator)

# Print the validation accuracy
print(f"Validation Accuracy: {validation_accuracy * 100:.2f}%")
```

```
523/523 [==============================] - 20s 38ms/step - loss: 0.5307 - accuracy: 0.8032
Validation Accuracy: 80.32%
```

CODE: PREDICTION:

Importing downloaded weights of the model from drive

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] import tensorflow as tf

    # Load the model from Google Drive
    model_path = '/content/drive/MyDrive/wasteClassificationModel.h5'
    model = tf.keras.models.load_model(model_path)
```

Double-click (or enter) to edit

```
[ ] from IPython.display import display, Javascript
    from google.colab.output import eval_js
```

```
def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

      // Wait for Capture to be clicked.
```

```
[ ]   // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename
```

```
[ ] from IPython.display import Image
    try:
```

```
[ ]     with open(filename, 'wb') as f:
          f.write(binary)
        return filename
```

```
[ ] from IPython.display import Image
    try:
      filename = take_photo()
      print('Saved to {}'.format(filename))

      # Show the image which was just taken.
      display(Image(filename))
    except Exception as err:
      # Errors will be thrown if the user does not have a webcam or if they do not
      # grant the page permission to access it.
      print(str(err))
```

```
Saved to photo.jpg
```

1m

```
Saved to photo.jpg
```

✓ 0s completed at 12:38 AM

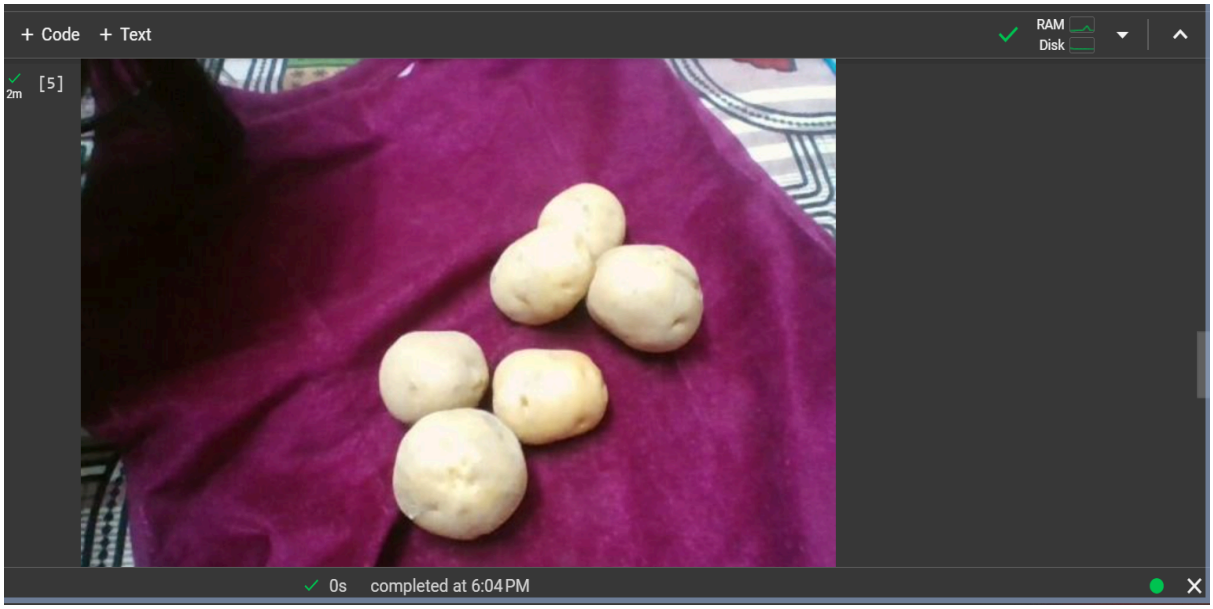
0s

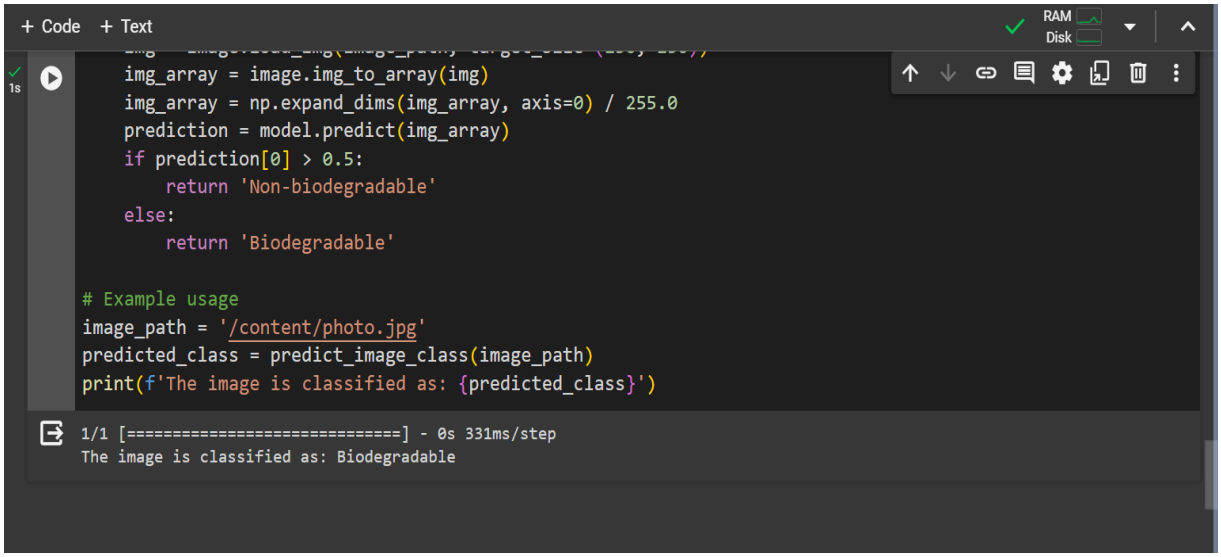
```
from tensorflow.keras.preprocessing import image
import numpy as np

# Define a function to preprocess and predict image class
def predict_image_class(image_path):
    img = image.load_img(image_path, target_size=(150, 150))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0) / 255.0
    prediction = model.predict(img_array)
    if prediction[0] > 0.5:
        return 'Non-biodegradable'
    else:
        return 'Biodegradable'

# Example usage
image_path = '/content/photo.jpg'
predicted_class = predict_image_class(image_path)
print(f'The image is classified as: {predicted_class}')
```

```
1/1 [==============================] - 0s 42ms/step
The image is classified as: Non-biodegradable
```





---

5) Conclusion

This binary waste classification model achieves 80% accuracy. The trained model captures an image real time and then predicts its class making it unsupervised. It has a CNN architecture and a total of 9 layers. These layers include convolutional, pooling, and dense layers. It uses the Sequential module of CNN having one input and output per layer.

In conclusion, this binary waste classification model demonstrates the effectiveness of CNNs for waste sorting tasks. Future work could explore completely automating the waste segregation process in dumpyards leading to better and efficient waste management.

---

6) References

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html