1. Import dependencies:

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
from PIL import Image
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
import cv2
from tensorflow.keras import layers, models
```

```
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-
packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy i
s deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

2. Loading data:

A- Loading the both data:

In [ ]:
```
data_garbage = os.listdir('data\Garbage Bag Images')
data_paper = os.listdir('data\paper')
data_plastic  =  os.listdir('data\plastic')
data_biological  =  os.listdir('data\\biological')
```

B- Verify the data;

In [ ]:
```
print(data_garbage[0:5])
print(data_paper[0:5])
print(data_plastic[0:5])
print(data_biological[0:5])
```

```
['00000000.jpg', '00000001.jpg', '00000002.jpg', '00000003.jpg', '00000004.jpg']
['paper1.jpg', 'paper10.jpg', 'paper100.jpg', 'paper1000.jpg', 'paper1001.jpg']
['1 (1).jpg', '1.jpg', '1000_545_1477922352-5428-paketyi.jpg', '11.jpg', '1263540856
_7390ee75d1_b.jpg']
['biological1.jpg', 'biological10.jpg', 'biological100.jpg', 'biological101.jpg', 'b
iological102.jpg']
```

C- Number of images:

In [ ]:
```
print(len(data_garbage))
print(len(data_paper))
print(len(data_plastic))
print(len(data_biological))
```

```
5000
8338
6428
6985
```

In [ ]:
```
import random
data_garbage = random.sample(data_garbage, 5000)
data_paper = random.sample(data_paper, 5000)
data_plastic = random.sample(data_plastic, 5000)
data_biological = random.sample(data_biological, 5000)
```

In [ ]:
```
print(len(data_garbage))
print(len(data_paper))
print(len(data_plastic))
print(len(data_biological))
```

```
5000
5000
5000
5000
```

3. Label of images:

A- Creating the labels:

In [ ]:
```
garbage_label = [0]*5000
paper_label = [1]*5000
plastic_label = [2]*5000
bilogical_label = [3]*5000
```

In [ ]:
```
print(data_garbage[0:5])
print(data_paper[0:5])
print(data_plastic[0:5])
print(data_biological[0:5])
```

```
['00001450.jpg', '00002337.jpg', '00004079.jpg', '00000973.jpg', '00001122.jpg']
['00003210.jpg', '00000410.jpg', 'paper_299.jpg', 'paper461.jpg', 'paper521.jpg']
['plastic_410.jpg', '00000931.jpg', '00003206.jpg', '00001968.jpg', '00001564.jpg']
['Crispy Chicken-Train (848).jpeg', 'biological578.jpg', 'Burger-Train (804).jpeg',
'Crispy Chicken-Train (405).jpeg', 'Baked Potato-Train (69).jpeg']
```

In [ ]:
```
print(len(garbage_label))
print(len(paper_label))
print(len(plastic_label))
print(len(bilogical_label))
```

```
5000
5000
5000
5000
```

B- Creatiing the label of the both datas:

In [ ]:
```
labels = garbage_label + paper_label + plastic_label + bilogical_label
print(len(labels))
```

```
print(labels[0:5])
print(labels[-5:])
```

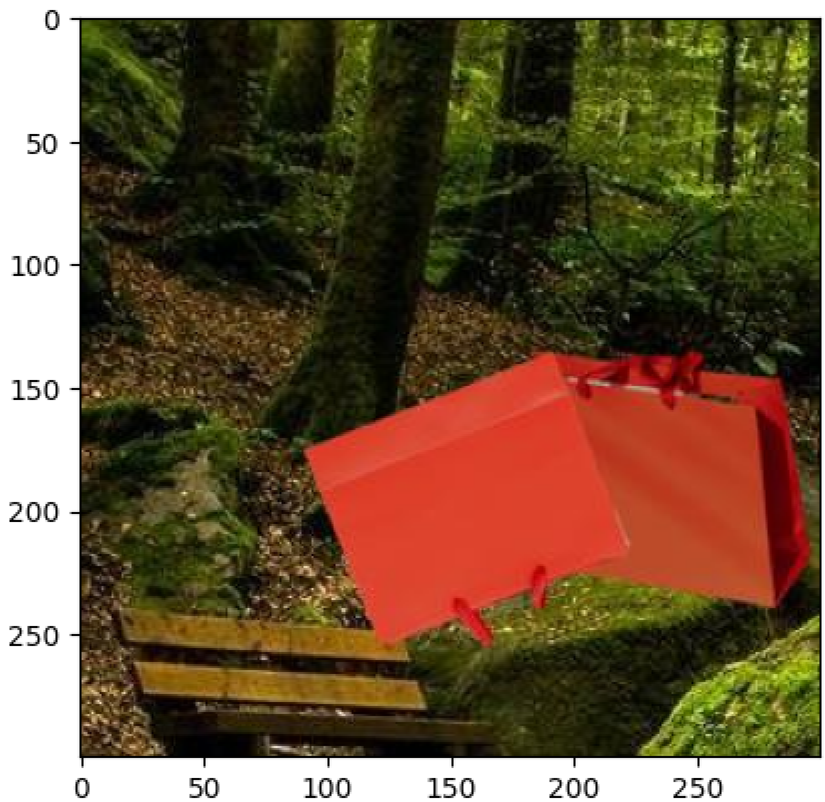
```
20000
[0, 0, 0, 0, 0]
[3, 3, 3, 3, 3]
```

4. Image preprocessing:

A- Displaying an highway images:

In [ ]:
```
img = mpimg.imread('data\paper\\00000001.jpg')
plt.imshow(img)
```

Out[ ]: <matplotlib.image.AxesImage at 0x262cdd14fd0>



C- Converting & resising image to numpy array:

In [ ]:
```
garbage = 'data/Garbage Bag Images/'
paper = 'data/paper/'
plastic = 'data/plastic/'
bilogical = 'data/biological/'
data = []
labels = []

# Charger les images et les étiquettes
for img in data_garbage:
    image = Image.open(os.path.join(garbage, img))
    image = image.resize((128, 128))
```

```python
    image = image.convert('RGB')
    image = np.array(image)
    data.append(image)
    labels.append(0)

for img in data_paper:
    image = Image.open(os.path.join(paper, img))
    image = image.resize((128, 128))
    image = image.convert('RGB')
    image = np.array(image)
    data.append(image)
    labels.append(1)

for img in data_plastic:
    image = Image.open(os.path.join(plastic, img))
    image = image.resize((128, 128))
    image = image.convert('RGB')
    image = np.array(image)
    data.append(image)
    labels.append(2)

for img in data_biological:
    image = Image.open(os.path.join(bilogical, img))
    image = image.resize((128, 128))
    image = image.convert('RGB')
    image = np.array(image)
    data.append(image)
    labels.append(3)
```

D- Type of the data:

In [ ]:
```python
type(data)
```

Out[ ]: list

In [ ]:
```python
type(labels)
```

Out[ ]: list

In [ ]:
```python
data = np.array(data)
labels = np.array(labels)
```

E- Number of images in data:

In [ ]:
```python
len(data)
```

Out[ ]: 20000

G- Type & dimmenssion of the images:

In [ ]:
```python
print(data.shape)
print(type(data[0]))
```

```
print(labels.shape)
print(type(labels))
```

```
(20000, 256, 256, 3)
<class 'numpy.ndarray'>
(20000,)
<class 'numpy.ndarray'>
```

5. To numpy array:

A- Separating the variables:

In [ ]:
```
labels.shape
```

Out[ ]:
```
(20000,)
```

In [ ]:
```
# Diviser les données en ensembles d'entraînement et de validation
X_train, X_val, y_train, y_val = train_test_split(data, labels, test_size=0.2, rand

# Normaliser les données
X_train = X_train / 255.0
X_val = X_val / 255.0

# Convertir les étiquettes en one-hot encoding
y_train = keras.utils.to_categorical(y_train, num_classes=4)
y_val = keras.utils.to_categorical(y_val, num_classes=4)
```

B- Type of X & Y:

In [ ]:
```
print(type(X_train))
print(type(y_train))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

C- Shape of X & Y:

In [ ]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(16000, 128, 128, 3)
(16000, 4)
```

6. Train test split:

A- Creating the variables of testing & training:

In [ ]:
```
data = np.array(data)
labels = np.array(labels)
```

In [ ]:
```
# Initialiser le modèle (couche par couche)
model = models.Sequential()
```

```
# 1ère couche de convolution avec:
# 32: 32 noyaux ou nombre de filtres
# (3,3) : taille de chaque filtre (carré 3*3)
# La fonction d'activation : ajouter la non-linéarité au modèle
# Input shape: définit la forme des données d'entrée (image 256x256 avec format (RG
model.add(layers.Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(128,

# 1ère couche de pooling
# Pool size (2,2): taille de fenêtre de pooling qui réduit chaque dim de la carte p
model.add(layers.MaxPooling2D(pool_size=(2,2)))

# 2ème couche de convolution:
# 64: 64 noyaux ou nombre de filtres
# (3,3) : taille de chaque filtre (carré 3*3)
# La fonction d'activation : ajouter la non-linéarité au modèle
model.add(layers.Conv2D(64, kernel_size=(3,3), activation='relu'))

# 2ème couche de pooling
# Pool size (2,2): taille de fenêtre de pooling qui réduit chaque dim de la carte p
model.add(layers.MaxPooling2D(pool_size=(2,2)))

# Applatissement des données:
# Convertir les caractéristiques 2D en un vecteur 1D
model.add(layers.Flatten())

# 1ère couche dense:
# Ajout d'une couche dense avec 128 unités
# Utilisation de la fonction d'activation ReLU
model.add(layers.Dense(128, activation='relu'))
# 1ère couche de dropout:
# Couche qui abandonne 50% des caractéristiques (diversité des caractéristiques)
model.add(layers.Dropout(0.5))

# 2ème couche dense:
# Couche avec 64 unités
model.add(layers.Dense(64, activation='relu'))

# 2ème couche de dropout:
model.add(layers.Dropout(0.5))

# Couche de sortie:
# Identifier le nombre de classes à prédire + la fonction d'activation
model.add(layers.Dense(4, activation='softmax'))

# Compiler le modèle
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
```

```
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

B- Training the model:

In [ ]:

```
model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=10,
    batch_size=8
)
```

```
Epoch 1/10
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

2000/2000 [==============================] - 163s 80ms/step - loss: 0.8335 - accuracy: 0.6491 - val_loss: 0.6226 - val_accuracy: 0.7788
Epoch 2/10
2000/2000 [==============================] - 151s 76ms/step - loss: 0.5669 - accuracy: 0.7867 - val_loss: 0.4538 - val_accuracy: 0.8205
Epoch 3/10
2000/2000 [==============================] - 152s 76ms/step - loss: 0.4647 - accuracy: 0.8339 - val_loss: 0.4107 - val_accuracy: 0.8482
Epoch 4/10
2000/2000 [==============================] - 153s 76ms/step - loss: 0.3850 - accuracy: 0.8652 - val_loss: 0.4145 - val_accuracy: 0.8428
Epoch 5/10
2000/2000 [==============================] - 153s 76ms/step - loss: 0.3311 - accuracy: 0.8814 - val_loss: 0.4447 - val_accuracy: 0.8403
Epoch 6/10
2000/2000 [==============================] - 152s 76ms/step - loss: 0.2757 - accuracy: 0.9039 - val_loss: 0.4432 - val_accuracy: 0.8497
Epoch 7/10
2000/2000 [==============================] - 152s 76ms/step - loss: 0.2331 - accuracy: 0.9230 - val_loss: 0.4905 - val_accuracy: 0.8475
Epoch 8/10
2000/2000 [==============================] - 157s 78ms/step - loss: 0.1991 - accuracy: 0.9336 - val_loss: 0.6139 - val_accuracy: 0.8478
Epoch 9/10
2000/2000 [==============================] - 153s 77ms/step - loss: 0.1813 - accuracy: 0.9430 - val_loss: 0.5855 - val_accuracy: 0.8528
Epoch 10/10
2000/2000 [==============================] - 155s 78ms/step - loss: 0.1497 - accuracy: 0.9530 - val_loss: 0.5126 - val_accuracy: 0.8535
```

Out[ ]:
```
<keras.src.callbacks.History at 0x262fdc46fa0>
```

E- Accuracy & score:

In [ ]:
```python
loss , accuracy = model.evaluate(X_val , y_val)
print("Accuracy score:", accuracy)
print("Loss score: ", loss)
```

```
125/125 [==============================] - 4s 29ms/step - loss: 0.5126 - accuracy: 0.8535
Accuracy score: 0.8535000085830688
Loss score:  0.5126371383666992
```

G- Save the model:

In [ ]:
```
# After training the model, save it to a file:
model.save('path_to_my_model.h5')
```

```
c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\engin
e\training.py:3103: UserWarning: You are saving your model as an HDF5 file via `mode
l.save()`. This file format is considered legacy. We recommend using instead the nat
ive Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

8. Example:

A- Loading the model:

In [ ]:
```
model = keras.models.load_model('path_to_my_model.h5')
```

In [ ]:
```
def trashPrediction(path, model):
    """
    Prédire la classe d'une image de déchet en utilisant le modèle entraîné.

    Arguments:
    path -- chemin vers l'image à classer
    model -- modèle entraîné de Keras
    """
    # Charger l'image
    input_image = Image.open(path)

    # Redimensionner l'image à 256x256 pixels
    image_resized = input_image.resize((128, 128))

    # Convertir l'image PIL en un array numpy
    image_np = np.array(image_resized)

    # Normaliser les pixels
    image_np_normalized = image_np / 255.0

    # Reshaper l'image pour correspondre à l'entrée du modèle
    image_reshaped = np.reshape(image_np_normalized, [1, 128, 128, 3])

    # Prédire la classe
    predict = model.predict(image_reshaped)
    print(predict)
    predict_trash = np.argmax(predict)

    # Afficher l'image originale
    plt.imshow(input_image)
    plt.axis('off')
    plt.show()

    # Afficher la classe prédite
    class_labels = ['garbage', 'paper', 'plastic', 'biological']
    predicted_class = class_labels[predict_trash]
    print(f'The predicted number is: {predict_trash}')
    print(f'The image is: {predicted_class}')
```

```python
# Exemple d'utilisation
path = "WhatsApp Image 2024-06-23 at 15.19.03.jpeg"
trashPrediction(path, model)
```

```
1/1 [==============================] - 0s 35ms/step
[[1.2295777e-03 1.7635196e-02 9.8104787e-01 8.7265020e-05]]
```

```
The predicted number is: 2
The image is: plastic
```