The Garbage Sorter

Modeling IIT Mandi's Solid Management Section, garbage disposal bins can be categorized as:

1. The Green-coloured dustbins are meant for wet and bio-degradable wastes.
   For e.g.: kitchen wastes including vegetables and fruits skins.
2. Blue dustbins are meant for the disposal of plastic wrappers and non-bio-degradable wastes.
3. Yellow dustbins are meant for papers and glass bottles.

The Garbage Sorter is a Deep Learning model based on the principle of transfer learning which in this case has been built upon ResNet18. I got this idea while doing an online assessment. It sorts the input image into six categories:

1. Paper
2. Metal
3. Cardboard
4. Glass
5. Plastic
6. Trash

Here 'Trash' being is unspecified input, i.e. when the input cannot be classified into any of the five categories.

• Libraries Used

The code is completely done in Python3. The various libraries used are:

1. Matplotlib – For displaying images during training and testing.
2. os – For managing directories of data-set and its contents.
3. Numpy – For managing input conversions for the model.
4. shutil – For moving files.
5. fastai – For using ResNet18 and its functions.
6. pathlib – For path management of data.
7. glob – Again for managing files.

▾ 1. Importing required Libraries

```
[1]  %matplotlib inline
```

```
[2]  import numpy as np
     import os
     import shutil
```

```
[3]  from fastai.vision import *
     from fastai.metrics import error_rate, accuracy
     from pathlib import Path
     import glob
```

• Importing and Extracting Data

The data set found online is imported from Google Drive and is then shifted to the 'data-resized' directory after unzipping the file.

▾ 2. Mounting google drive so that the dataset can be brought in for training

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

```
Mounted at /content/drive
```

▾ 3. Unzipping the dataset to a folder

```
[ ]  !mkdir data-resized
     !unzip "/content/drive/MyDrive/dataset-resized.zip" -d "data-resized"
```

The data set is divided into six sub-directories each named after a class name (as formerly mentioned).

Further, the data-set is randomly divided into the standard training and validation sets using a 75-25 split respectively. All this requires a few functions.

4. A number of functions:

> a. train_val - to break the dataset into train and validation sets
>
> b. file_name - gets file names for a particular type of trash, given indices
>
> c. move_files - moves group of source files to another folder

```
def file_name(waste_type,indices):
    file_names = []
    for i in indices:
      file_names.append(waste_type+str(i)+".jpg")
    return(file_names)

def move_files(start_file,folder):
    for file in start_file:
        shutil.move(file,folder)

def train_val(folder,seed1):
    length = len(os.listdir(folder))
    full_set = list(range(1,length+1))

    # for Training data
    random.seed(seed1)
    training_set = random.sample(list(range(1,length+1)),int(.75*length))

    # Validation data
    validation_set = list(set(full_set)-set(training_set))

    return(training_set,validation_set)
```

5. Breaking the data into training and validation sets

```
subsets = ['train','valid']
waste_types = ['cardboard','glass','metal','paper','plastic','trash']

for subset in subsets:
    for i in waste_types:
        folder = os.path.join('data',subset,i)
        if not os.path.exists(folder):
            os.makedirs(folder)

if not os.path.exists(os.path.join('data','test')):
    os.makedirs(os.path.join('data','test'))

for i in waste_types:
    folder = os.path.join('data-resized/dataset-resized',i)
    train_ind, valid_ind = train_val(folder,1)

    # Training set
    train_names = file_name(i,train_ind)
    train_source_files = [os.path.join(folder,name) for name in train_names]
    train_dest = "data/train/"+i
    move_files(train_source_files,train_dest)

    # Validation set
    valid_names = file_name(i,valid_ind)
    valid_source_files = [os.path.join(folder,name) for name in valid_names]
    valid_dest = "data/valid/"+i
    move_files(valid_source_files,valid_dest)
```

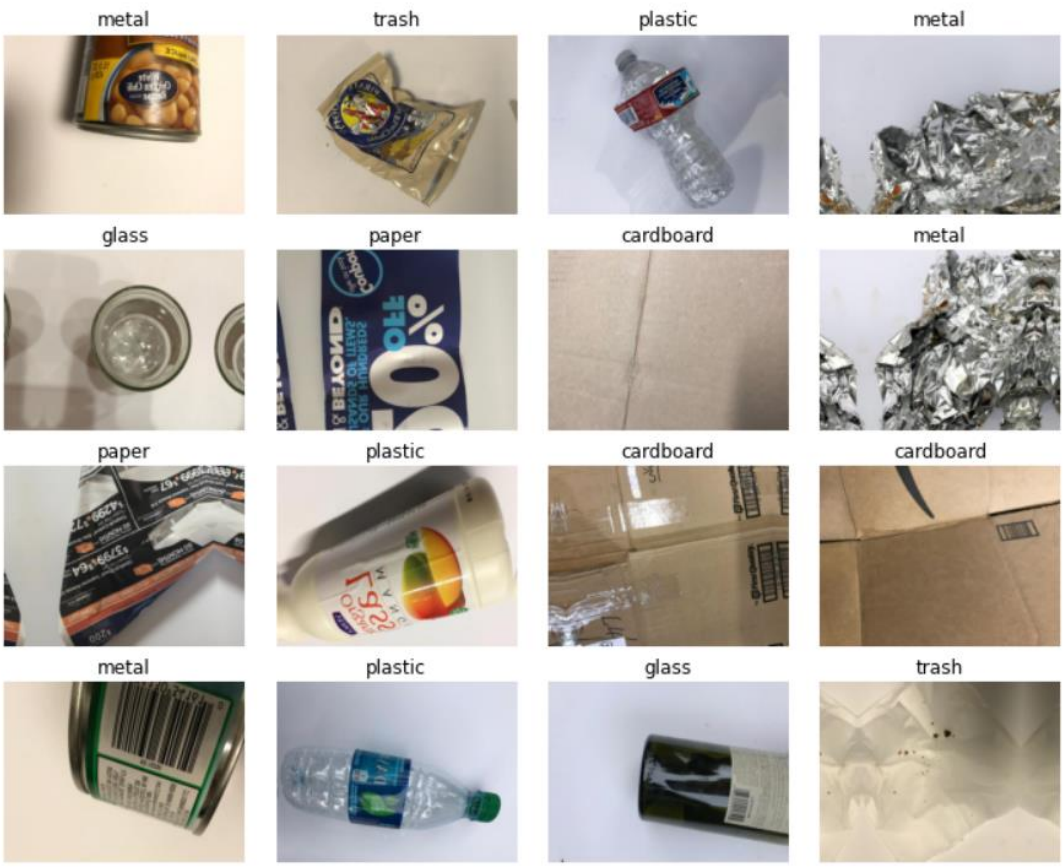
The *'get_transforms'* method is then used for data augmentation for better training whilst only using 'flip_vert = True' and a batch size of 32.

```
[30] tfms = get_transforms(do_flip=True,flip_vert=True)
     data = ImageDataBunch.from_folder(path,test="test",ds_tfms=tfms,bs=32)
```

- **Looking at the Data**

  Here is a look at a randomly chosen sample of different types of categories present in the dataset.

```
data.show_batch(rows=4,figsize=(10,8))
```



- # Creating and Training the Model

The model has been created using Transfer Learning, using the image feature extracting power of ResNet18.

The model has been trained with the hyperparameters as:

1. Starting Learning Rate (*start_lr*) –> 1.e-7
2. Ending Learning Rate (*end_lr*) –> 1.e1
   These parameters help to find the optimal learning rate in '*lr_find*' method
3. *max_lr* (Peak for Learning Rate) –> 5.5.e-03
4. epochs –> 20

The model is sensitive to these hyperparameters and may lead to various results on different initializations depending upon the optimization of the ResNet18 achieved.

▾ 7. Creating model using Resnet18

```
[15] learn = create_cnn(data,models.resnet18,metrics=[error_rate, accuracy])
```

```
[16] learn.model
```

▾ 8. Training the model

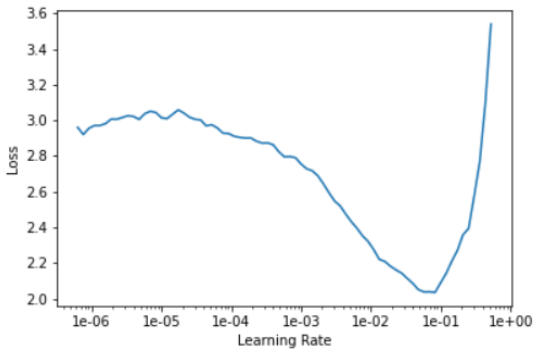
```
start_lr=1e-7
end_lr=1e1
learn.lr_find(start_lr, end_lr)
learn.recorder.plot()
```

50.00% [1/2 00:53<00:53]

epoch	train_loss	valid_loss	error_rate	accuracy	time
0	2.474701	#na#			00:53

50.85% [30/59 00:27<00:26 6.8334]

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```



```
learn.fit_one_cycle(20,max_lr=5.5e-03)
```

epoch	train_loss	valid_loss	error_rate	accuracy	time
0	1.593344	0.714116	0.275591	0.724409	01:02
1	1.028686	0.526897	0.177953	0.822047	01:00
2	0.792736	0.503335	0.177953	0.822047	00:59
3	0.650604	0.450316	0.152756	0.847244	00:59
4	0.563223	0.579375	0.181102	0.818898	00:59
5	0.584789	0.442069	0.144882	0.855118	00:59
6	0.532648	0.494021	0.163780	0.836220	00:59
7	0.463179	0.394335	0.119685	0.880315	01:00
8	0.406316	0.473151	0.154331	0.845669	00:59
9	0.394524	0.375773	0.125984	0.874016	00:59
10	0.373755	0.402112	0.141732	0.858268	00:59
11	0.308442	0.419583	0.129134	0.870866	01:00
12	0.269912	0.325656	0.107087	0.892913	01:00
13	0.220971	0.266073	0.088189	0.911811	00:59
14	0.198723	0.230974	0.077165	0.922835	00:59
15	0.176604	0.234539	0.072441	0.927559	01:00
16	0.160214	0.244954	0.086614	0.913386	00:59
17	0.126869	0.229319	0.078740	0.921260	00:59
18	0.127854	0.232304	0.078740	0.921260	01:00
19	0.116957	0.226648	0.075591	0.924409	01:00

Here, it can be seen that after 20 epochs an accuracy of about 92.5% is obtained. The accuracy can be further increased but it requires time (A lot! Each epoch takes about 1 minute) and computational power. The model being too deep fluctuates and thus needs effectively more time to converge better. Moreover, the training loss and validation loss are about 0.12 and 0.23 which can be considered quite good considering the environment created.

• Testing the Model

1. Pre-set Dataset

Here is a data set of five images that I have made from random pictures taken from the web. Although, I have previously tested the model on a test split from the data-set but prefer to show the results on images that were not a part of the data-set and thus, contain different attributes. The following code is required to get the data-set and predict the class using the model to put the category in the correct bin.

▾ 9. Testing the model

```
[19] from IPython.display import Image, display

     model = learn.model
     model = model.cuda()
     import glob
```

```
for imageName in glob.glob('/content/drive/MyDrive/0-Test/*.jpg'):
    print(imageName)
    img = open_image(imageName)
    prediction = learn.predict(img)
    print('Prediciton: '+ str(prediction[0]))
    prediction=str(prediction[0])
    if (prediction=='paper' or prediction=='glass' or prediction=='cardboard'):
      print("Throw in the Yellow Dustbin!")
    elif (prediction=='plastic' or prediction=='metal'):
      print("Throw in the Blue Dustbin!")
    else:
      print("Throw in the Green Dustbin!")

    display(Image(filename=imageName, width=300, height=200))
    print("\n")
```

```
/content/drive/MyDrive/0-Test/test1.jpg
Prediciton: paper
Throw in the Yellow Dustbin!
```

```
/content/drive/MyDrive/0-Test/test5.jpg
Prediciton: paper
Throw in the Yellow Dustbin!
```

```
/content/drive/MyDrive/0-Test/test6.jpg
Prediciton: plastic
Throw in the Blue Dustbin!
```

```
/content/drive/MyDrive/0-Test/test3.jpg
Prediciton: metal
Throw in the Blue Dustbin!
```

```
/content/drive/MyDrive/0-Test/test2.jpg
Prediciton: glass
Throw in the Yellow Dustbin!
```

It can be seen that out of five images, four have been correctly sorted. The cardboard image contains very different attributes (grass, more than one object, etc.) as compared to the data-set cardboard images and thus is very difficult to correctly categorize given the training.

2. Live Test

Here I have shown images from a live test in which I chose ‘paper’ as a category to be tested. I have used Google Colab’s built-in webcam API.


```
for imageName in glob.glob('/content/photo.jpg'):
    print(imageName)
    img = open_image(imageName)
    prediction = learn.predict(img)
    #print(prediction)
    print('Prediciton: '+ str(prediction[0]))

    if (prediction=='paper' or prediction=='glass'):
      print("Throw in the Yellow Dustbin!")
    elif (prediction=='plastic' or prediction=='metal'):
      print("Throw in the Blue Dustbin!")
    else:
      print("Throw in the Green Dustbin!")

    display(Image(filename=imageName))
    print("\n")
```

It can be seen that the paper has been correctly categorized, thus proving that the model is good enough for most of the cases, but surely it needs more training and preventive regularisation to make it better.

Thus, the Garbage Sorter seems to be a useful tool, can be converted into an app (which could not be done due to lack of time) and thus be used daily. Moreover, the model can be changed and adapt to different kinds of classifications.