Trash Classification Using Convolutional Neural Networks

***Abstract*—Machine Learning algorithms have become a good solution to many everyday-life problems to be faced. This project aims to exploit a computer vision approach to take images of recycling or garbage in order to classify them into six classes. For this purpose, the TrashNET[1] dataset has been used as a starting point, then it has been accurately processed and fed to a FNN and a CNN, with the aim of comparing the different results.**

I. Introduction

The impact that our actions have on the environment is always an important topic to be accounted. In particular, recycling is of fundamental importance for a sustainable society, but sorting garbage by hand is not always a good solution, because of the different problems arising from human attitude. For this reason, this project is intended to find an automatic method for doing it, so that the processing plants work with correctly sorted materials and, therefore, are more efficient. The use of machine learning techniques may help to make improvements to our society. Since a classification problem has been addressed, it has been necessary to collect some data, in particular images, to be first processed and then fed to the designed ANNs, together with the class which they belong to. After having accurately trained them, as it will be later shown, it will be possible to predict the category of garbage that an object belongs to, based on just an image. A comparison in terms of accuracy will be made, in order to explain why a neural network works better than the other one.

II. Dataset and Data Preprocessing

The first step in the Machine Learning algorithms design is the data preprocessing. Firstly, after having checked the dataset, it has been noticed that the available data of the TrashNET dataset were not enough in the training process of the neural networks; for this reason, it has been decided to enlarge it, by taking other images from another dataset belonging to a *kaggle* project and named Garbage Classification[2]. Since, it is fundamental to avoid that the training data are repeated, in order to avoid to check if some images were already existent, those belonging to the TrashNET dataset have been flipped and rotated, because the number of photos was lower with respect to the new ones. This is one of the methods used to increase the number of data, allowing the neural networks to learn in a better way. Then, the dataset has been balanced by using the under-sampling method: therefore, in order to obtain the same quantity of information for each class, some images belonging to the majority class have been removed. Finally, it has been decided to resize the images by creating two different datasets: one of 62x48, the other of 128x128. Furthermore, for the first dataset the grayscale version has been used, while for the second one it has been chosen to utilize the RGB version. The purpose of this project is not only to design and test an ANN, but also to show the variations in terms of performance when the dataset changes and also when the neural network does.

Related Code

```
import os
from PIL import Image
```

As a first thing, the two libraries of interest have been imported: ***os*** library, allowing to manipulate paths, read or write files and so on, thus it has been used in order to access to the directories for reading and saving images, and ***PIL*** library, from which the module *Image* has been utilized, in order to manipulate image file formats, for example to open, rotate and display them. In this first part, only the code related to the Garbage Classification dataset resize is shown.

```
i=0
directory = os.chdir("/Users/francescastabile/Desktop/dataset/Glass/")
directory2 = "/Users/francescastabile/Desktop/Glass32x24/"

for element in os.listdir(directory):
    if (element != '.DS_Store'):
        im = Image.open(element,"r");
        newsize = (62, 48)
        im_res = im.resize(newsize)
        i=i+1
        im_res.save(directory2 +"Glass(" +str(i) +")" +".jpg")
```

Actually, this code has been run also for resizing the TrashNET dataset, with the difference that this last has also been modified by flipping and rotating each image: in particular, a random variable has been created and, on the basis of its value, the images have been randomly modified.

```
x=random.randint(0,2)
if x==0:
  im_res = im_res.transpose(Image.FLIP_LEFT_RIGHT)
elif x==1:
  im_res = im_res.transpose(Image.FLIP_TOP_BOTTOM)
else:
  im_res = im_res.transpose(Image.ROTATE_90)
```

III. CSV

After the work on the images, it is necessary to build the csv file that is used to train the neural networks. First, it is needed to save all the data in the same folder and each of them must have as name the material that we need to classify from the relative picture. The purpose is to obtain a **csv file** with two different feature columns which are “Label” and “Pixel”. In order to do that the function “*createFilelist*” that receives as input the path and the image format was built. As an output is obtained a list of the pictures and the materials classes that is contained in the path.

[1] TrashNET dataset: https://github.com/garythung/trashnet

[2] Garbage Classification dataset: https://www.kaggle.com/datasets/sumn2u/garbage-classification-v2

```
def createFileList(myDir, format='.jpg'):

  fileList = []
  labels = []

  keywords = {"Cardboard" : "0","Glass": "1","Metal":"2","Paper":"3",
              "Plastic":"4","Trash":"5"}

  for root,dirs,files in os.walk(myDir, topdown=True):
    for name in files:
      if name.endswith(format):
        fullName = os.path.join(root, name)
        fileList.append(fullName)
        for keyword in keywords:
          if keyword in name:
            labels.append(keywords[keyword])
          else:
            continue
  return fileList, labels
```

After the initialization of the two lists where the values will be stored, it is built a dictionary that gives to every material a specific number to identify. Now, it is possible to use a cycle that allows it to see all the files in the folder. Then, it is checked if the format is equal to the '*endswith*' of the image. If the condition is satisfied, it is added in the ***fileList*** and if one of the ***keywords*** is inside of the name, it is also added the corresponding value to the ***labels list.*** Finally, it is possible to return both.

At this point, using the method ***open*** contained in the **csv** library, it is possible to build the **garbage_classification.csv** file and given to this method the argument **w** it is set it write mode. Then, with the method ***writerow*** are specified the name of the two columns.

```
with open("garbage_classification.csv", 'w') as f:
  writer = csv.writer(f)
  writer.writerow(['Label','Pixel'])

myFileList, labels = createFileList('/content/')
```

After the creation of the file it is called the already described function "*createFilelist*" to obtain the two lists needed, in particular the first input of the method is **'/content/'** and for the second input the default's format value.

All the images were uploaded inside of the folder **'/content/'** of Google colab in order to use the computational resources that provide the platform to speed up the computation of the neural network.

Finally it is possible to implement the following code:

```
for file in myFileList:

  img_file = Image.open(file)

  img_grey = img_file.convert('L')
  value = np.asarray(img_grey.getdata(), dtype=int)

  with open("garbage_classification.csv", 'a') as f:
    writer = csv.writer(f)
    writer.writerow([labels[i],value])
  i+=1
```

That allows to add the rows in the csv file.

Scrolling thorugh all the images contained in **myFileList,** it is used the method '***.convert***' with the string **'L'** in order to convert them in greyscale.

If it had chosen to use the string '**RGB**' in the method '***.convert***' would have been necessary to specify the dimension of the value adding to the method of **np** the command: **'*.reshape*((width,height,3))'**. But, the main idea is to store only the **csv** file related to greyscale and in the next part of the project using libraries work with RGB.

Now it is possible to write the value and the label in the file **garbage_classification.csv**, but differently from before it is used the character '**a**' to open it. This is due to the fact that it is wanted to **'*append*',** in every iteration of the cycle "For", the new information while save the previous and this is not possible with the **'w'** that rewrite the file. As last command of the cycle it is increased a variable '**i**' used to iterate all the list of Labels .

In this way, it is obtained a csv file where the first column identify the material with the number associated in the keyword list and the second one the Pixel in Greyscale.

In order to use the csv file in the code it is needed to write the following lines:

```
import pandas as pd

csv = pd.read_csv("/content/garbage_classification.csv")
csv.head()
```

It is used the method ***read_csv*** contained in the pandas library to read the file contained in **'/content/'.**

Finally using the method ***head*** it is possible to show the first five rows of it.

	Label	Pixel
0	2	[134 133 131 ... 145 145 144]
1	0	[130 130 130 ... 245 243 241]
2	0	[163 165 169 ... 111 129 128]
3	1	[215 217 217 ... 228 227 227]
4	4	[170 170 170 ... 221 222 221]

IV. MACHINE LEARNING MODELS

A comparison between two types of ANNs was analysed. In particular, the networks type taken under exam were ***Feed Forward*** and ***Convolutional Neural Network***.

The first step is to split the dataset in order to have data for training, validation and prediction. This was made thought the function ***train_test_split*** provided by the library ***sklearn***. Both networks were trained by using the 70% of the dataset, the 24% for the validation, the rest 6% was used for the prediction. The function ***train_test_split*** was used two times and then the data was normalized in the interval [0,1]. It is shown in the code below:

```
## Split the data into training and testing sets ##
X_train, X_test, y_train, y_test = train_test_split(X2, Y, test_size=0.3, random_state=1)
X_test,x_val, y_test ,y_test_val = train_test_split(X_test,y_test,test_size=0.2, random_state=1)
# Data normalizat
X_train = X_train / 255.0
X_test = X_test / 255.0
x_val=x_val/255.0
```

ANNs

The ***FNN (Feedforward Neural Network***) can be used for image classification. The model was built by using the function ***Sequential*** provided by ***Tensorflow*** library. The model main consist of an flatten input layer which was added

with the function ***Flatten*** it receives the input image. Then, some hidden fully connected layers were added with the function ***Dense***. Finally, as an output, it was added also a fully connected layer which was sized with the number of the classification category. To create the model the following aspects must be taking as start point:

- Number of hidden neurons between the size of the input and output layer.
- Number of hidden neurons equals to 2/3 of the size of the input layer.
- Number of hidden neurons less than twice the size of input layer.

The ***CNN*** (***Convolutional Neural Network)*** is mainly used for this type of application. Also, the model of it was created using the ***Sequential*** function. The first layers of this model are for features extraction, these were added with the functions ***Conv2D*** and ***MaxPooling2D.*** After it has a ***Flatten*** layer, some ***Dense*** layers, and an output layer as the ***FNN.***

After many proofs with different layer number, layer size, activation function, and due to the behaviour of the dataset the best results were obtained with the following networks:

Feed Forward Neural Network

```
model = keras.Sequential([
    layers.Flatten(input_shape=(62, 48,1)),
    layers.Dense(1100, activation='relu'),
    layers.Dense(1100, activation='relu'),
    layers.Dense(6,activation='softmax')

])
```

Convolutional Neural Network

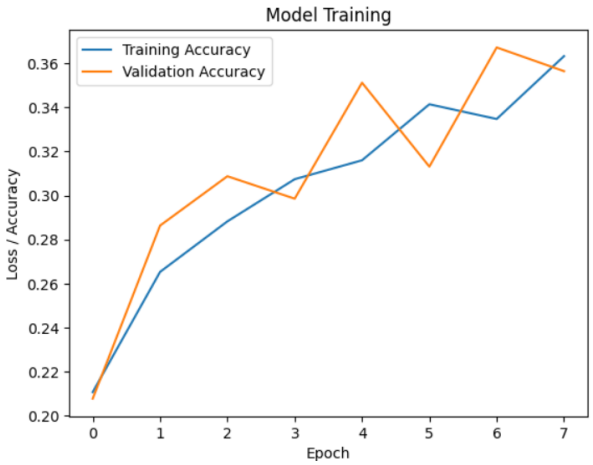
```
model1 = keras.Sequential([
layers.Conv2D(12, (3,3),1,activation='relu',input_shape=(62,48,1)),
layers.MaxPooling2D(),
layers.Conv2D(12, (3,3),1,activation='relu',kernel_regularizer=tf.keras.regularizers.l1()),
layers.MaxPooling2D(),
layers.Flatten(),
layers.Dense(512, activation='relu'),
layers.Dense(6,activation='softmax'),
])
```

The models were compiled with the function ***compile*** by using as optimizer ***Adam***. Both were trained with a ***batch size*** of 128 and a number of ***epochs*** equal to 8, for avoid the overfitting. At the end, the ***ANNs*** were evaluated in terms of prediction accuracy using the function ***evaluate*** provided by ***TensorFlow*** library. This function used the 6% of the unknown dataset by the trained networks, for make the prediction. It is shown below:

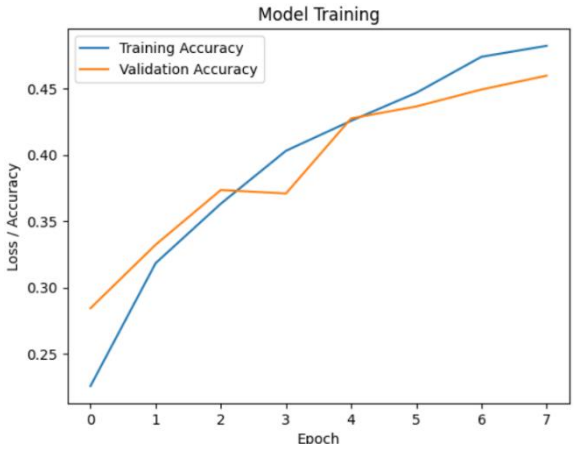
```
# Compile the model
model1.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=['accuracy'])

# Train the model
history = model1.fit(X_train, y_train, batch_size=128, epochs=12, validation_data=(X_test, y_test))

# Evaluate the model on test data
test_loss, test_acc = model1.evaluate(x_val, y_test_val, verbose=2)
print("Test accuracy:", test_acc)
```

RESULT

The training and validation accuracy are shown in the below figure. And was found out an accuracy prediction of 34.72% for the ***FNN***.



For the ***CNN***, the following figure was obtained and instead an accuracy prediction of 44%.



From this result it is possible to conclude that the ***CNN*** works better for this type of application. In fact, in the next chapter it will be tried to improve the results by using it.

V. TRASH CLASIFICATION USING ML-TOOLS

Once the garbage classification problem has been solved using the algorithms proposed in the previous section, the problem is handled using specialized machine learning tools: ***TensorFlow*** and the ***Keras*** library. This proposal arose due to the difficulty encountered in handling large dataset.

These tools were chosen due to their great benefits in developing machine learning models. Among the main ones are flexibility, scalability, compatibility, and a wide range of pre-trained models.

DEVELOPMENT MACHINE LEARNING MODEL

For the development of this section, a work environment was created in Jupyter. This allows to manage the dependencies in an isolated way, which avoids conflicts between the versions of the libraries used, this ensures the reproducibility and stability of the project. In addition, it allows to encapsulate all the necessary dependencies, which facilitates the transfer of the project to other systems or collaborators. The settings for creating the environment are shown in the following lines of code.

```
C:\Users\Ferna> python -m venv ENVIROMENT_NAME
C:\Users\Ferna> .\ENVIROMENT_NAME\Scripts\activate
```

```
C:\Users\Ferna> pip install ipykernel
C:\Users\Ferna> python -m ipykernel install --name=ENVIROMENT_NAME
```

The model creation was divided into five stages: Setup and data loading, preprocessing, deep learning model, model performance evaluation, and finally saving the model. Each of the stages is detailed below.

1. Setup and load data.

Within the work environment, the necessary dependencies for the development of the model must be installed, which are: ***TensorFlow, OpenCV-python, scikit-learn and matplotlib.***

To load the data into the environment, the ***image_dataset_from_directory*** function was used, so it is necessary to create directories that contain the images of each class. Next, the images that have a different extension than the one allowed are removed, since they can cause conflicts. In this last step the following code was used:

```
image_exts = ['jpeg','jpg', 'png']
for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir, image_class)):
        image_path = os.path.join(data_dir, image_class, image)
        try:
            img = cv2.imread(image_path)
            tip = imghdr.what(image_path)
            if tip not in image_exts:
                print('Image not in ext list {}'.format(image_path))
                os.remove(image_path)
        except Exception as e:
            print('Issue with image {}'.format(image_path))
```

2. Data Pre-processing

The versatility of the ***Keras*** library allows to do part of the pre-processing while importing the dataset from the directories. Among the most important functions are resizing images, giving a color format, creating labels, shuffle the dataset, among others. In this case, the data was modified as follows:

```
data = tf.keras.utils.image_dataset_from_directory('data',image_size=(128, 128))
```

As part of the pre-processing, the data was normalized so that instead of having values of 0-255 (pixel color intensity) there are values of 0-1. Finally, the division of the data set is carried out, assigning:

- 75% Training,
- 20% Validation and
- 5% Testing.

3. Deep learning model

3.1 CNN (Convolutional Neural Network) basic model

The model was chosen because it is highly effective in the image classification problem as shown in the previous chapter. Since it is designed to automatically extract relevant features from images through convolutional and pooling layers. This allows the model to identify important visual patterns in the garbage images, such as shapes, textures, or colors, which helps improve classification accuracy. Furthermore, they can handle the variability in the position and size of objects in an image. This means that the model could recognize a garbage object at different locations or scales within the image.

The following code shows the model created for the classification:

```
model = Sequential()
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(64, (3,3), 1, activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(6, activation='softmax'))
```

The ***CNN*** model is made up of an input convolutional layer, it has 32 filters with a 3x3 dimension window, the activation chosen for this layer is ReLU and the size of the input data is indicated. The filters are in charge of detecting specific patterns such as edges, textures, and shapes. However, an image can contain multiple instances of a feature in different locations. For this reason, MaxPooling2D is applied after the convolutional layer, to reduce the dimensionality of the extracted features and retain the most relevant information. This helps capture general features and decrease sensitivity to variations in location.

There are two hidden convolutional layers of 64 and 128 filters respectively activated by the ReLU function. The filters windows have dimensions of 3x3.

The Flatten layer is used to flatten the output of the convolutional layers into a one-dimensional tensor. Because the dense layer (fully connected) used at the end of the network needs flatten the data.

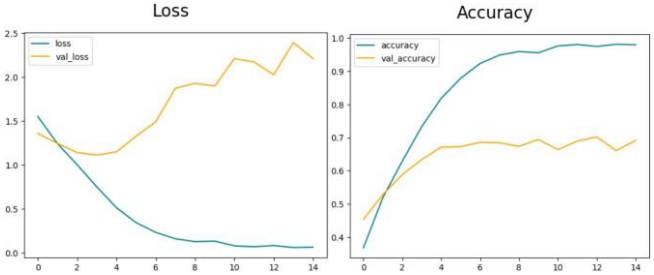
Next is a Dense layer with 64 neurons, it uses the ReLU activation function to introduce non-linearity into the model. The dense layer is responsible for performing linear calculations and combining the features extracted by the convolutional layers.

Finally, we have a dense output layer with 6 neurons, one for each class. The activation function used is SoftMax, which assigns probabilities to each class. The class with the highest probability will be the final prediction of the model.

The Adam compiler was used as the model compiler. For the training and validation of the model, 15 epochs were configured, obtaining as a result:

```
Epoch 15/15
335/335 [==============================] - 98s 291ms/step - loss: 0.0664 - accuracy: 0.9799 - val_loss: 2.2122 - val_accuracy: 0.6914
```

The graphs below show the comparison of the Loss and the Accuracy of the training and validation data.

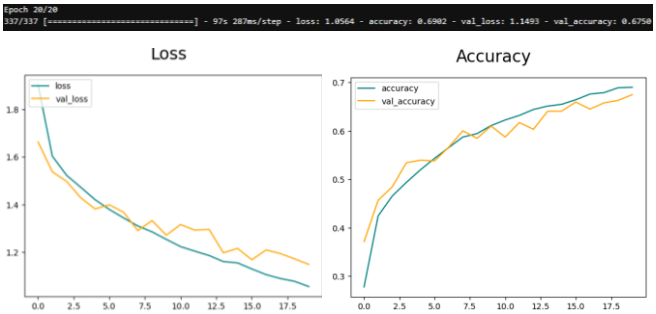


3.2 CNN with regularization

The previous results show that there is an overfitting in the model. Therefore, as a solution, it was proposed to add an L2 regularization technique to the dense layer of the model to penalize large weights.

```
model.add(Dense(64, activation='relu',kernel_regularizer = tf.keras.regularizers.l2(0.05)))
```

Using the above modification, the results improved. These are shown in the following graphs.

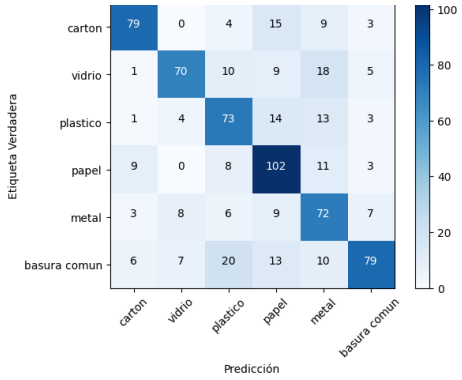


4. Model performance evaluation

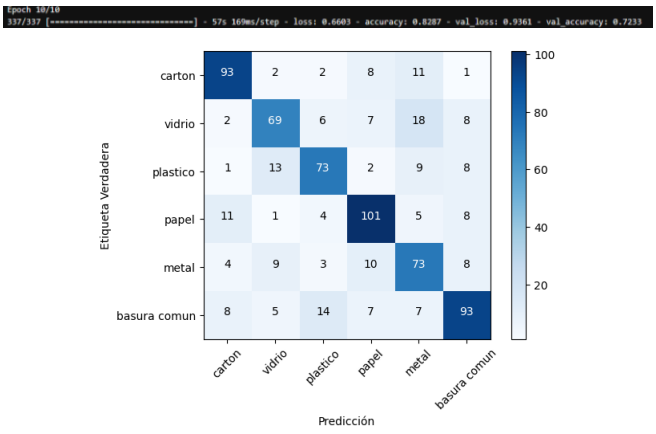
The metrics chosen to evaluate the performance of the model were: precision, recall, and accuracy. The values obtained were:

precision:	**recall:**	**accuracy:**
0.9487603306770325	0.9663299918174744	0.6747159361839294

The confusion matrix obtained is shown below:



To improve the results were performed a fine-tuning procedure. In this stage, the weights of some of the layers of the pre-trained model are adjusted. This is possible by allowing certain layers to be trainable again during another training process. The results obtained after this process are the following:



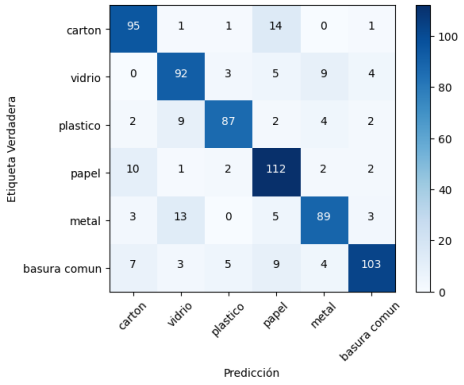
precision:	**recall:**	**accuracy:**
0.9589743614196777	0.9557070136070251	0.7130681872367859

Alternative solution

Instead of creating a new instance of the ***CNN*** model, the pre-trained ***VGG16*** model was loaded and adapted to the garbage classification problem by adding an output layer with 6 classes. All layers of the ***VGG16*** base model have been frozen, which means that the weights will not update during training. The following code was implemented to accomplish this task:

```
base_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False,
                                         input_shape=(128, 128, 3))
base_model.trainable = True
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

model = tf.keras.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(6, activation='softmax')
])
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
hist = model.fit(train, epochs=10, validation_data=val, callbacks=[tensorboard_callback])
```

The obtained results with the proposed solution were:



precision:	**recall:**	**accuracy:**
0.976460337638855	0.9515718221664429	0.8117897510528564

Finally, a test of the model's performance was made with images that were not included in the test dataset. It was observed that the model classified the provided images in the correct class 7-8/10.

5. Save the model

The last thing to do is save the model for future use. From the ***Keras*** library it is possible import the dependency called ***load_model***, and with the ***save*** function the model is saved in a directory with an .h5 (Hierarchical Data Format version 5) extension. This file format allows large volumes of data and complex structures to be stored efficiently. In this way, all the model architecture, trained weights, and other relevant information such as the optimizer used, build configuration, metrics, etc. are saved.

VI. Conclution

In this work different types of ANNs were implemented and trained in different ways. In particular, it was observed that ***CNN*** works better for this kind of application and also that the results are very sensitive to the different properties of the image like size and color and the dimension of the dataset.

It has also been observed that thanks to the use of Keras and TensorFlow tools, the preprocessing of data and the creation of the csv is much easier, but also using pre-trained models can maximize the quality of the results.

Finally, a model capable of classifying six classes of garbage with an accuracy of 80% has been saved so that it can be used in future problems giving an image belonging to any of the classes as input.

VII. References

Spezzano G. slides for machine learning course, 2022-2023

Yang M. Thung G, Classification of Trash for Recyclability Status