DETECTION OF MICROPLASTICS USING MACHINE LEARNING

ZENON CHACZKO[1], PETER WAJS-CHACZKO[2], DAVID TIEN[3], YOUSEF HAIDAR[1]

[1]University of Technology Sydney, Ultimo, 2007, Australia
[2]Macquarie University, Macquarie Park, 2109, Australia
[3]School of Computing & Mathematics, Charles Sturt University, NSW, Australia
E-MAIL: Zenon.Chaczko@uts.edu.au, Peter.Wajs-Chaczko@students.mq.edu.au, dtien@csu.edu.au, Yousef.Haidar@uts.edu.au

Abstract:

Monitoring the presence of micro-plastics in human and animal habitats is fast becoming an important research theme due to a need to preserve healthy ecosystems. Microplastics pollute the environment and can represent a serious threat for biological organisms including the human body, as they can be inadvertently consumed through the food chain. To perceive and understand the level of microplastics pollution threats in the environment there is a need to design and develop reliable methodologies and tools that can detect and classify the different types of the microplastics. This paper presents results of our work related to exploration of methods and techniques useful for detecting suspicious objects in their respective ecosystem captured in hyperspectral images and then classifying these objects with the use of Neural Networks technique.

Keywords:

Microplastics; Machine learning; Neural network; Identification; Classification; Hyperspectral image

1. Introduction

Microplastics are fast becoming persistent contaminants in both aquatic and terrestrial environments [1-4]. There is an increasing scholarly work about the presence and negative effects of microplastics on marine environments [2, 3]. So far, a relatively small number of reports can be found about the need for studying contamination sources, pollution characteristics and ecological impact of microplastics. There is mounting evidence that macroplastics can influence soil biota at different trophic levels [4], and thus pose a serious threat to animal and human health in the food web. Further in-depth investigation is required to uncover the full impact and define ecological threats related to microplastics at various levels and in all kinds of biological ecosystems. To study the impact of microplastics on various environments and humans, researchers require better analytical methodologies and tools which would allow to determine the characteristics of pollution and the impact of macroplastics on the aquatic, soils and aerial ecology. However, there is an absence of standardised protocols, methodologies, best practices and tools for the quantification [4]. And there is an urgent need for the uniformity of methods, procedures and protocols related to microplastics presence, extraction and their classification (identification). This research aims to address this need. In order to better understand the level and impact of microplastics pollution, methods, computational solutions and sensing devices that can detect and classify the different types of the microplastics urgently required [4, 5]. In the field of plastic recycling there are some useful and practical tools present; hyperspectral imaging is on of most promising technologies, for it successes [5-9].

The main objectives and outcomes of this research work project is to explore a reliable approach for finding an effective computing solution for locating objects of interest in an image (canvas), as well as, to perform classification of these objects for correct identification.

A. Scope, Aims & Objectives

At this stage, this research is only concerned about exploring methods of detecting the number of objects along with their respective location in a hyperspectral image, whilst the classification of objects applies the Neural Networks. Thus, the main objectives and outcomes of this project is to find a reliable method for:

- locating objects in an image/canvas
- classifying objects

The proposed Neural Network-based solution is to employ the supervised learning approach which needs data to be used as training samples. Even though there are other methods of training Neural Network models without data, such as the unsupervised learning approach, they still require

some information about the application or scenario; in the unsupervised learning approach a way of observing a model or a system is required to adjust and learn, to reach the optimal solution. In this project's scenario it is unrealistic to have a model that simulate the way plastics react to light synthetically and accurately to the real world. Thus, the supervised learning approach was chosen.

2. Methodology

As previously stated, a dataset is required for training the Neural Network model, but none were to be found in the public domain. Some creative thinking was needed to keep the research alive and one main idea stood out; that idea was to use a dataset of hyperspectral images of objects, and discarding the requirement of having them be images of microplastics. The motivation was, that at this stage this research project is concerned about seeing if hyperspectral images can be used effectively in Neural Networks, to apply that in the field of microplastics pollution research.

The "*SpecTex* " (University of Eastern Finland n.d.) dataset was used for this project [9]. This dataset includes spectral images of sixty textile samples with different texture patterns [9]. In this research case, those samples can be considered as (suspicious contaminant) objects. The images have a size of 640 x 640 x 39, as in image high times image width times number of channels. The number of channels refer to different snapshot of the object in different wavelengths, ranging from 400-*nm* to 780-*nm* with a 10-*nm* increment. To prepare the data for training, a limited number of files (10), that contain the samples, were selected for the experiment (see TABLE 1).

TABLE 1. The list of files selected for the experiment

T01.tif	T02.tif
T03.tif	T04.tif
T05.tif	T06.tif
T10.tif	T11.tif
T14.tif	T60.tif

A random point was taken from each file, and using that point *28x28x39* sized images were cropped at the center of that point to simulate plastics that were broken down to small pieces by the sun and ocean waves. Then the cropped images were slightly distorted by using a circular mask to simulate the shape of a pellet. This process was repeated 100 times each file, and another 10 times for each files. The first time to generate the training samples and the second time to generate the evaluation/testing samples. However, there were two regions that were chosen for each type of data samples to be generated from (see Fig. 1). The smaller region is used for the testing as it fewer samples to be generated are needed than the training samples. The reason behind this separation is because to further ensure that the evaluation samples are new data that are not seen in the training phase; this will give an indication of how the system will perform on unseen data.

A. Tools And Libraries

The following selection of software development environment and analytical tools were explored, integrated and used in the experimental solution:

- *Python* – main language used
- *tensorflow* – GPU enabled library for building Neural Networks
- *numpy* – data manipulation module (library)
- *matplotlib* – graphing and plotting
- *tensorboard* – graphing and plotting
- *scikit-learn* – used for the confusion matrix and for shuffling data
- *scikit-image* – used for finding the contours for object detection
- *panda* – usually used for data manipulation, but in this case, it was opportunistically used for displaying tables
- *Jupyter* – a playground for quickly and conveniently experimenting with python

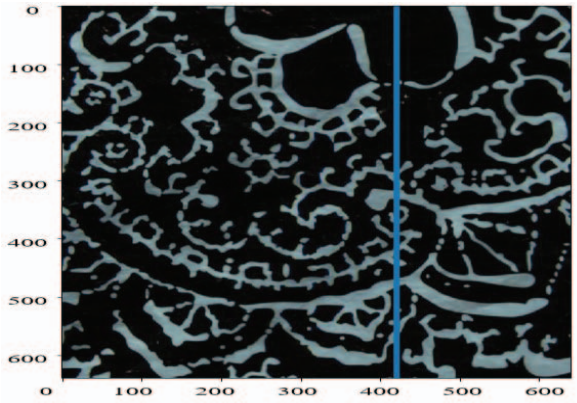

FIGURE 1. A single layer of 39 layers in the T14.tif file

Key Concepts

In 2013, Zhang et al. in their research work [8] on tensor discriminative locality alignment and spectral–spatial feature extraction in hyperspectral image discussed dimensionality reduction of hyperspectral data using the Principal Component Analysis (PCA) approach. The purpose was to remove the redundancy of information and thus reduce the size of the data. The redundancy comes from the inter-band correlation which is can be very high, and it can be reduced, with a rather low chance of losing significant information. However, giving the fact that the

size of the data set used in this initial phase was rather small to begin with, PCA was not considered but not employed in the project. Also, Zhang et al [8] introduced a very useful concept of tensor representation of hyperspectral images (HSIs) that we used to put forward the HSI feature extraction, which is called tensor discriminative locality alignment (TDLA) method. In our experimentation, the tensor representation of HSIs concept was employed. The HSIs are composed of several images/layers, that represent a spectral-channel/spectral- wavelength. Images/layers could be then represented by a grayscale image, where the values of pixel represent the intensity of the light wave in that pixel corresponding to the spectral-channel.

3. The design and implementation of Machine Learning Solution

The Machine Learning system need to execute in the deployment state. It is assumed that when the system is deployed, its computing model is already trained. The deployed system (see Fig. 2) executes the following steps:

1. The objects/microplastics are placed in a canvas
2. A hyperspectral image is taken of the whole canvas
3. One layer of the hyperspectral image is supplied to the Object Detector
4. The locations of the objects are supplied to the "*crop&Resize*" module
5. All the layers are supplied to the "*crop&Resize*" module
6. Cropped and correctly resized hyperspectral images is supplied to the Neural Network model
7. The Neural Network provides the prediction

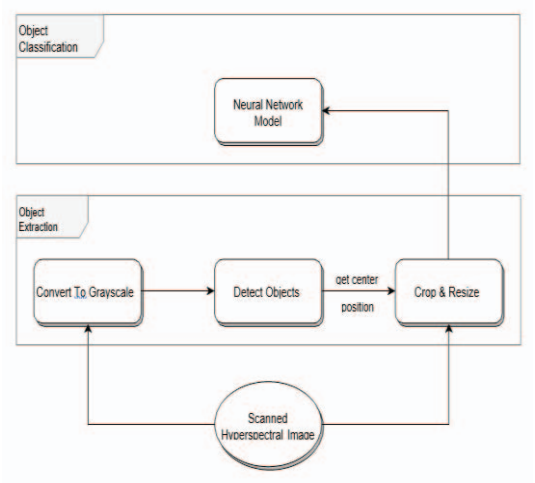

FIGURE 2. System Design – Already Trained NN Model

Note, Step 5 can also be placed before Step 3, since the cropping and resizing can only happen after all the first 5 steps are completed. However, the rest of the steps should be in their respective order.

A. Object Detection

Assume that the figure 3a below as an example of a grayscale image of an object, applying the find contours function from scikit-image, will yield what is shown in figure 3b. The con- tour's maximum and minimum values in both the Y and X axis, are calculated to construct the bounding-box of the object, and then using that bounding-box the image is then cropped as shown in 3c. That cropped image is then resized to fit the input specification of the Neural Network, but that is not shown here. It is important to note that the cropped image will not be the grayscale image but rather all the 39 layers of different wavelength data, but this is shown just for clarity of what would happen.

The goal of this process is to automate the extraction of multiple objects within a single snapshot, to properly feed them to the Neural Network model. For example, the Figure 4a below shows a *500 x 500* canvas where objects are randomly placed in it, and the noise was introduced to see the reliability of the method; the Figure 4b show the result of the find contours function, which detected 6 objects and colored them differently. Similarly, the max and min is used to extract the objects (Fig. 4c). Again, the actual extraction/cropping will be applied to all 39 layers for each object (Fig. 5); In the Figure 5 it is hard to distinguish the difference between each layer; remember that each layer represents an image of a wavelength, ranging from *400-nm* to *780-nm* with a *10-nm* increment. However, a computer algorithm or for this case a trained Neural Network model, could possibly recognise the subtle differences and use them to correctly classify the objects.

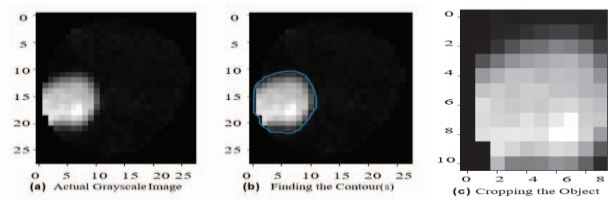

FIGURE 3. Object detection example

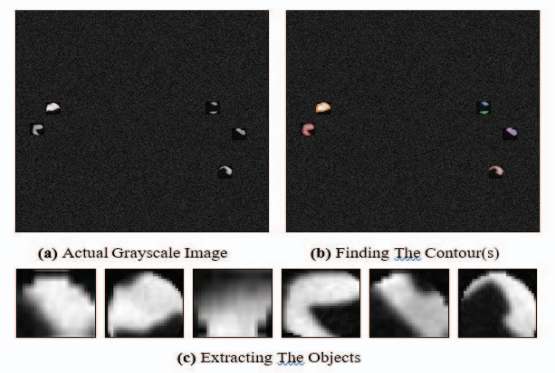

FIGURE 4. Example of objects extraction

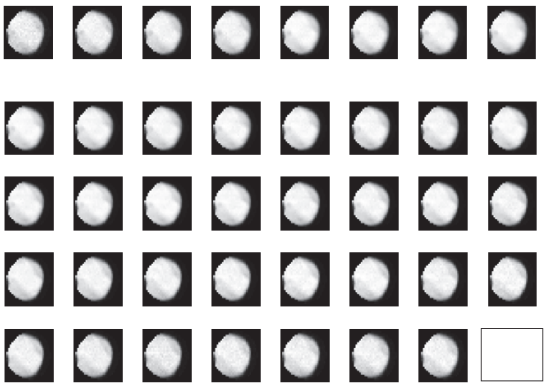

FIGURE 5. Example of all 39 Layers of an object

The contours of every object are being calculated and used to crop the objects as shown in LISTING 1. The loop in line 5, is a secondary filter that ensure that the found contour must have more than 30 points representing it, because otherwise, the found contour is likely representing a anomaly or noise in the image rather than an actual object. Line 13 and 14 show how the bounding box are calculated from the points in the contour.

LISTING 1. Object Detection and Extraction

```
1  import numpy as np
2  import tensorflow as tf
3
4  img_shape = img_width , img_ height , img_wavelength = 28 , 28 , 39
5  img_size_flat = img_width * img_height * img_wavelength
6
7  num_classes = 10
8  with tf. variable_scope ('input'):
9      x = tf. placeholder (tf . float32, shape =[None, img_size_flat] , name='x')
10     pass
11 with tf. variable_scope ('target'):
12     y_true = tf. placeholder (
13         tf. float32, shape =[ None , num_classes ], name='y_true'
14     )
15     pass
16 with tf. variable_scope ('dense_layer 1'):
```

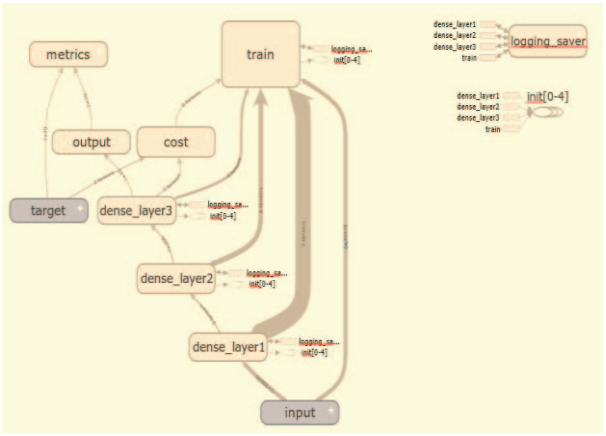

FIGURE 6. Neural Network Implementation Design

B. Object Classification

This part of the system was implemented using *tensorflow*; this section will describe the implementation details, along with *tensorflow*'s layers and train application programing interface(API). The structure of the system is as follow, the input layer, three fully-connected dense layers, and lastly the output layer; each layer, along with the training component shall be discussed in the sections below. Also, there will be an extra section that describes the other, less important, components. The layers API is a python module that contains a set of functions to construct Neural Network related layers, such as convolution layer, denser layer, and more. The way it is designed allow for stacking of layers, as in the output of the previous layer can be passed to the next layer. Neural Network programing especial the backpropagation stage is highly error prone; the layers API take care of the low-level implementation details, and leaves high-level design up to the user.

Moreover, *tensorflow*'s train API will take care of the backpropagation process. By using both modules, this will hence eliminate most of the chances of introducing errors in the model. The Figure 6 below shows the graph generated by the *tensorboard* tool; while the tool only allows exporting the graph in "*png*" format, one can merely copy the *svg* code from the Internet browser and convert it to a pdf file for a better quality, which is what was done and shown in the figure.

All layers, except the 3rd dense layer, have a rectified linear unit or a RLU as their activation function. The RLU can be defined with the equation below.

$$f(x) = x+ = max(0, x) \quad (1)$$

Also the weights and biases were initialized with the *xavier* initializer provided by *tensorflow* ; it is consider to be the state of the art technique for initializing the weights. Rather than initializing the weights randomly using this function is assumed to have less chance for the Neural Network to get stuck on the local minimum of the cost function in the training stage, and hence, it get a better chance of finding the global minimum.

LISTING 2. Neural Network Model's Source Code

```
import numpy as np
import tensorflow as tf

img_shape = img_width , img_ height , img_wavelength = 28 , 28 , 39
img_size_flat = img_width * img_height * img_wavelength

num_classes = 10
withtf.variable_scope ('input'):
    x = tf.placeholder (tf.float32, shape =[None, img_size_flat], name='x')
    pass
withtf.variable_scope ('target'):
    y_true =tf.placeholder (
      tf.float32, shape =[ None , num_classes ], name='y_true'
    )
    pass
withtf.variable scope ('dense layer 1');
    net = x
    net = tf.layers.dense(
        inputs=net, name='layer_dense1',
        units=256, activation=tf.nn.relu,
    kernel_initializer=tf.contrib.layers.xavier_initializer () 22
    )
    pass
with tf.variable_scope ('dense_layer2'):
    net = tf.layers.flatten (net)
    net = tf.layers.dense(
        inputs=net, name='layer_dense2',
        units=256, activation=tf.nn.relu,
    kernel_initializer=tf.contrib.layers.xavier_initializer () 30
    )
    pass
with tf.variable_scope ('dense_layer3'):
    net = tf.layers.dense(
        inputs=net, name='layer_dense3',
        units=num_classes, activation=None,
    kernel_initializer=tf.contrib.layers.xavier_initializer () 37
    )
    pass

with tf.variable_scope ('output'):
    logits = net
    y_pred = tf.nn.softmax (logits=logits)
    pass
with tf.variable_scope ('metrics'):
    y_pred_cls = tf.argmax (y_pred , axis=1)
    y_true_cls = tf.argmax (y_true , axis=1)
    correct_prediction = tf.equal (y_pred_cls , y_true_cls)
    accuracy = tf.reduce_mean (tf.cast (correct_prediction , tf.float32))
    pass
with tf.variable_scope ('cost'):
    cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(
    labels=y_true, logits=logits 54
    )
    loss = tf.reduce_mean (cross_entropy)
    pass
with tf.variable_scope ('train'):
    global_step = tf.Variable (0, name='global_step', trainable=False)
    optimizer = tf.train.AdamOptimizer (learning_rate=1e-4).minimize (loss,
    global_step=global_step)
    pass
with tf.variable_scope ('logging_saver'):
    tf.summary.scalar ('loss', loss)
    tf.summary.scalar ('accuracy', accuracy)
    summary =tf.summary.merge_ all ()
    saver =tf.train.Saver ()
    pass
```

C. Input Layer

The "target" component, shown in Figure 6 is an input, that contain the true label; it is only used in training, and not in prediction mode.

TABLE 2. Input Data Shape

	Data Shape Description	Data Shape[1]
x_image	batch size[2]	20
	flattened image size: 28 × 28 × 39 image width × image height × No. channels	30576
y_true	batch size	20
	No. Classes	10

[1] for example x_image shape is then 20 by 30576.
[2] The size 20 was chosen arbitrarily.

D. Densely Connected Layers

TABLE 3. Densely Connected Layers Data Shape

	Data Shape Description	Data Shape
dense_layer1	batch size	20
	No. dense_layer1's neurons	128
dense_layer2	batch size	20
	No. dense_layer2's neurons	256
dense_layer3	batch size	20
	No. dense_layer3's neurons which has to be No. classes	10

E. Output Layer

TABLE 4. Output Layer Data Shape

	Data Shape Description	Data Shape
prediction_softmax	batch size	20
	No. Classes	10

F. Training Components

The *"softmax_cross_entropy_with_logits"* function supplied by *tensorflow*'s API is used to represent the cost functions. The cross entropy function require the actual true labels in order to calculate the cost. The calculated cost is then reduced by averaging the container tensor on the batch axis, in other words the shape of the tensor would be [1 10] rather than the original [20 10] shape. The averaging help in smoothing the learning curve. This is what is called batch mode; no conclusive research on literature was done by this report, but batch mode was used in most examples on the Internet, so batch mode was automatically chosen over stochastic mode. After that the cost is supplied to the training component, which is just an *"AdamOptimizer "*, that is implemented by *tensorflow*. The *tensorflow* API provide

different optimizers, and all of them managed the backpropagation process; it incorporates the weights regulation process, which requires the gradient that is calculated from the loss; the gradient calculation is also managed by the optimizer. The optimizer requires a scalar value to define the learning rate, so the value 1 10−4 was picked. The *"AdamOptimizer* " was picked as it seemed to be the state of the art algorithm that every article/tutorial used.

G. Related Components

The "metrics" and the "*logging_saver*" are the only components left unmentioned. Both components are related and they are unnecessary for the application, but are used for gathering metrics about the training phase progress. Logs are captured on every 10th iteration of training steps, and they contain the accuracy-and-loss, resulted within each training step/batch. The data is shown in Figure 7.

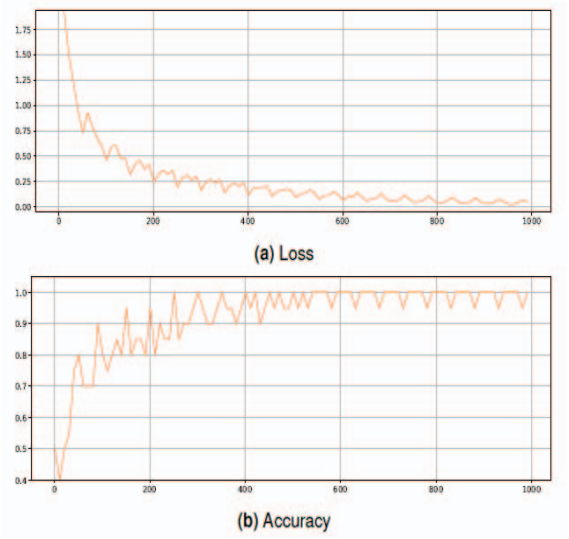

FIGURE 7. Metrics Over Training

4. Experimental Results

This section will only focus on the object classification part, or the Neural Network model; that is because the object detection is a well-known area and the results taken from the experiments were to be expected, and are already shown in previous sections. There were two stages in the building of the Neural Network model, to be discussed; the optimization stage and the evaluation stage. Both sections will include a brief description of how these results were realized and gathered.

A. Optimization

The sample data used for training had a size of 1000. The data were shuffled in a random order using function designed for this purpose from the *scikit-learn* tool; the function merely shuffles the data and there is nothing special about it, but it was used, nonetheless, merely for convenience. As discussed earlier, the results are generated (Fig. 7), using the data collected from the "*logging_saver* " component, which logs scalar values every 10 iteration of training steps; the data were conveniently accessible from the *tensorboard* tool. The loss results (Fig. 7) show that these values continue to decrease until they stabilise. The loss value refers to how much of a change is needed to minimize the error and reach the optimal solution. Just because the loss has stabilized, it does not necessarily mean that it had reached the optimal solution; in fact, there is no global method, across all scenarios and situations, that proves that the found solution is optimal. To do that one would need specific knowledge on the field of interest. On the other hand, the accuracy value (Fig. 7b) refers to the accuracy over the training batch, and it does not necessarily evaluate or prove that the model is perfect, like how it is shown to be reaching a 100% accuracy. However, it does show that, at least, within the training dataset, the model is figuring out the relationship between different features in the data and classifying them correctly.

LISTING 3. Optimization Process – Source Code

```
train_batch_size = 20
im_data_shuffled, im_label_shuffled = sklearn.utils.shuffle(im_data, im_label,
    random_state=0)

def optimize_model(num_iterations):
    session = nn_model.session
    global_step = nn_model.global_step
    summary = nn_model.summary
    accuracy = nn_model.accuracy
    x = nn_model.x
    y_true = nn_model.y_true
    optimizer = nn_model.optimizer

    steps = session.run(global_step)
    for i in range(steps, steps + num_iterations):
        FROM = i * train_batch_size % len(im_masked_shuffled)
        TO = FROM + train_batch_size
        x_batch = im_masked_shuffled[FROM:TO].reshape([20, 28*28*39])
        y_true_batch = im_label_shuffled[FROM:TO]
        feed_dict_train = {x: x_batch, y_true: y_true_batch}
        session.run(optimizer, feed_dict=feed_dict_train)

        if i % 10 == 0:
            testing_summary , cgsteps = session.run([summary, global_step],
    feed_dict=feed_dict_train)
            testing_writer.add_summary(testing_summary, cgsteps)
            testing_writer.flush()
            acc = session.run(accuracy, feed_dict=feed_dict_train)
            msg = "Optimization Iteration: {0:>6}, Training Accuracy: {1:>6.1%}"
            print(msg.format(i + 1, acc))
```

LISTING 4. The Optimization Process Output

```
...
Optimization Iteration:    871, Training Accuracy: 100.0%
Optimization Iteration:    881, Training Accuracy:  95.0%
Optimization Iteration:    891, Training Accuracy: 100.0%
Optimization Iteration:    901, Training Accuracy: 100.0%
Optimization Iteration:    911, Training Accuracy: 100.0%
Optimization Iteration:    921, Training Accuracy: 100.0%
Optimization Iteration:    931, Training Accuracy:  95.0%
Optimization Iteration:    941, Training Accuracy: 100.0%
Optimization Iteration:    951, Training Accuracy: 100.0%
Optimization Iteration:    961, Training Accuracy: 100.0%
Optimization Iteration:    971, Training Accuracy: 100.0%
Optimization Iteration:    981, Training Accuracy:  95.0%
Optimization Iteration:    991, Training Accuracy: 100.0%
CPU times: user 5.48 s, sys: 683 ms, total: 6.16 s
Wall time: 5.93 s
```

FIGURE 8. Falsely Classified Images – After Optimization

B. Evaluation of Results

After a thousand iteration of training, the evaluation resulted in an accuracy of 95%, with a 95 correct classifications out of 100 samples, that were never seen by the Neural Network model. The figure 8 below show all 5 misclassified objects. While this may seem impressive, it is not an indication of performing as good in the real-world scenario. That is because it is unclear what did the Neural Network model learn from the training phase; for example, the features that it learned about and used, does not exist in hyperspectral images of microplastics. The question is then, can the Neural Network, given a proper dataset, find features that distinguish the types of different microplastics, during the training phase. As previously state, hyperspectral imaging of plastics is used in the classification of plastics in the recycling industry, so it is highly likely that the answer to that question is yes. However, given the fact that no data about hyperspectral imaging of microplastics were available in public, at the time of this report was made, it is impossible to have a comprehensive conclusion, a one that is free of assumptions. In the end, these results do indicate the possibility of Neural Networks model, trained with hyperspectral images of microplastics, being applicable.

TABLE 5. Confusion Matrix – After Optimization

		Predicted as									
		0	1	2	3	4	5	6	7	8	9
True Label	0	10	0	0	0	0	0	0	0	0	0
	1	0	10	0	0	0	0	0	0	0	0
	2	0	0	10	0	0	0	0	0	0	0
	3	0	0	0	10	0	0	0	0	0	0
	4	0	0	1	1	8	0	0	0	0	0
	5	0	0	0	0	0	10	0	0	0	0
	6	0	0	0	0	0	0	8	0	2	0
	7	0	0	0	0	0	0	0	10	0	0
	8	0	0	0	0	0	0	1	0	9	0
	9	0	0	0	0	0	0	0	0	0	10

LISTING 5. Evaluation Process – Source Code

```
def evaluate_model(show_example_errors=False, show_confusion_matrix=False):
    session = nn_model.session
    global_step = nn_model.global_step
    x = nn_model.x
    y_true = nn_model.y_true
    y_pred_cls = nn_model.y_pred_cls


    num_test = len(im_masked_test)
    # Allocate an array for the predicted classes which
    # will be calculated in batches and filled into this array.
    cls_pred = np.zeros(shape=num_test, dtype=np.int)

    steps = session.run(global_step)
    print('Steps: {}'.format(steps))

    feed_dict = {
        x: im_masked_test.reshape([-1, 28*28*39]),
        y_true: im_label_test
    }

    cls_pred = session.run(y_pred_cls, feed_dict=feed_dict)
    cls_true = im_label_test.argmax(axis=1)

```

5. Conclusions

This paper has demonstrated how the Machine Learning-based solutions can be designed and built for the effective identification and classification of microplastics objects.
A viable, practical and possibly standardised solution for these tasks can be archived by incorporating and integrating various public domain advanced tools and libraries. However, it was found, if we cannot solely rely on such a solution alone. Each time, when somehow a newer set of data is introduced, it is likely some further changes and adaptations will be still required. This is because open-source tools in most cases address very generic solutions to the algorithm and rarely these solutions cater for the adaption and their continued improvement to satisfy the required accuracy.

References

[1] Browne, M.A., Galloway, T. and Thompson, R., 2007. Microplastic—an emerging contaminant of potential concern? Integrated environmental assessment and Management, 3(4), pp.559-561.

[2] Lusher, A., 2015. Microplastics in the marine environment: distribution, interactions and effects. In Marine anthropogenic litter (pp. 245-307). Springer, Cham.

[3] Hidalgo-Ruz, V., Gutow, L., Thompson, R.C. and Thiel, M., 2012. Microplastics in the marine environment: a review of the methods used for identification and quantification. Environmental science & technology, 46(6), pp.3060-3075.

[4] Defu He, Yongming Luo, Shibo Lua, Mengting Liu, Yang Song Lili Leia, 2018. Microplastics In Soils:

Analytical Methods, Pollution Characteristics And Ecological Risks TrAC Trends in Analytical Chemistry, Volume 109, Dec 2018, pp. 163-172.

[5] Chaczko Z., Kale A., Santana-Rodríguez José Juan, Suárez-Araujo Carmen Paz, 2018. Towards an IOT Based System for Detection and Monitoring of Microplastics in Aquatic Environments, IEEE 22nd International Conference on Intelligent Engineering Systems (INES 2018), June 21 2018, Las Palmas, Spain, pp. 000057-000062

[6] Kale A., Chaczko, Z., Evolutionary Feature Optimization and Classification for Monitoring Floating Objects, Computational Intelligence and Efficiency in Engineering Systems, Volume 595 of the series Studies in Computational Intelligence, Springer 2015, pp.3-16.

[7] Karaca, A.C., Ertürk, A., Güllü, M.K., Elmas, M. & Ertürk, S. 2013, 'Plastic waste sorting using infrared hyperspectral imaging system', pp. 1-4.

[8] Zhang, L., Zhang, L., Tao, D. & Huang, X. 2013, 'Tensor Discriminative Locality Alignment for Hyperspectral Image Spectral–Spatial Feature Extraction', IEEE Transactions on Geo-science and Remote Sen,

[9] University of Eastern Finland n.d., Spectex | UEF, University Website, Finland, viewed 28 May 2018, <https://www.uef.fi/web/spectral/spectex>., 2015) (pp. 107-112).