Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Oliver Istad Funch and Robert Marhaug

# Detecting Improperly Sorted Materials In Trash Bags

## The Development of a System for Analyzing Household Trash Bags with Sound and Metal Detection using Artificial Neural Networks

Master's thesis in Mechanical Engineering
Supervisor: Martin Steinert
June 2020

NTNU
Norwegian University of Science and Technology

Oliver Istad Funch and Robert Marhaug

# Detecting Improperly Sorted Materials In Trash Bags

## The Development of a System for Analyzing Household Trash Bags with Sound and Metal Detection using Artificial Neural Networks

Master's thesis in Mechanical Engineering
Supervisor: Martin Steinert
June 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Kunnskap for en bedre verden

**Abstract**

In this thesis, the application of machine learning to identify glass and metal in municipal waste is investigated. The system utilizes sound and metal detector data from an experimental setup as inputs for the machine learning model. The experiment was made to simulate the emptying of waste performed by a waste collection truck, as this location was the intended implementation location. The experiment involved the emptying of one bag at a time, as it was seen as a necessary first step. The machine learning model used was a CNN model, developed iteratively through an ablation study. Both multiclass classification and multilabel classification was tested. Multilabeling was the favored approach where an accuracy of 96.25% was reached on independent test data. These were obtained using condenser microphones, a Beat-frequency oscillation metal detector, and both Mel spectrograms and MFCC spectrograms to represent the sound recordings. The results shows that the system works well for identifying glass and metal in singular bags, and thereby shows promise in the proposed location, as well as other implementation areas. The system will however require some additional testing after installation on a collection truck, as it was only tested in the experimental setting.

## Acknowledgement

We would like to extend our gratitude to Renovasjonsetaten for enabling this project. Special thanks goes to Jørgen Simensen Almankaas and Jan Haakon Ellefsen-Killerud for your extensive knowledge and valuable reflections in many of our discussion on waste management.

We would also like to extend our greatest gratitude to our supervisors for motivating feedback and valuable advice throughout this process. Special thanks to Sampsa Kohtala for his constructive advice and many insightful suggestions and who has instrumental in the writing of our paper.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Formal Problem Description

In municipal waste management (MWM), information about sorting quality and location variances in sorting is important for being able to make correct and effective improvement measures. The waste collection system in Oslo makes information gathering difficult, so a way of effectively gathering the desired info needs to be devised.

## 1.2 Introduction to the Master's Thesis

Renovasjonsetaten (REN) is the department responsible for municipal waste management in Oslo. REN is interested in gathering more information about the collected waste, especially concerning the quality of household sorting performed by the consumer. The goal of this initiative is to better understand today's waste management system so effective improvement measures can be implemented. The sorting and recycling process is also considered relevant for this thesis even though it is performed by a separate department, as the departments are soon to merge.

Information about household sorting, specifically concerning materials being wrongly sorted, is of high value to help develop measures that can improve the status quo. Also, the ability to track the time and location of bad sorting acts as an incentive to the consumer to follow the guidelines given. This is made clear by Rada et al. [30], who found that controlled household containers had better sorting quality than uncontrolled road containers as they could not be traced to a specific household. This was true even for areas where the population was highly respectful of the environment. This control refers to the implementation of Radio-Frequency Identification (RFID) tracking in trash bins. This system does not provide automatic collected information about the sorting quality of the collected trash, only the means of connecting a container to a household. It can therefore be suspected that a combination of an automatic classification system for sorting quality along with an RFID system might further incite the consumer to follow the sorting guidelines. Locating households or areas responsible for poor sorting can also lead to more effective measures taken.

According to the field association for recycling in Norway, Avfall Norge [28], the target goals for recycling are 50%, 55%, 60%, and 65%, within the years 2020, 2025, 2030, and 2035, respectively. These numbers are set by the EU, and relates to the amount of materials/waste to be properly handled. According to REN, the current level is at about 35% [4], where most of the loss stems from incineration of recyclable materials.

This Master's thesis is based on a previous project work (Appendix A.3) where the goal was to determine the best way to approach the given problem de-

scription from REN. The assignment for the project was to determine where in the waste collection cycle to collect information and find the best method for performing the task. The conclusion of the project was to develop a machine learning based identification system that can be mounted to the back of a garbage collection truck. This allows the detection of sorting quality at an early stage in the trash collection cycle. This conclusion was largely based on the success of using sound recordings with machine learning to identify the presence of glass, and the ease of applying a metal detector to detect metal. The initial motivation for using sound in conjunction with machine learning, was that during a ride along on two of the trash collection routes done by the authors, they could effortlessly recognize the sound of glass breaking upon hitting the back of the truck. If one could log this occurrence and where it happened, an overview of which areas are bad at sorting could be obtained. Combining both sound and metal detection data could potentially yield a high accuracy for classifying the glass and metal presence of individual trash bins. Also, identifying poor sorting at this stage has the advantage of knowing where the trash originated, as the bins are equipped with RFID tags that identify their location in combination with GPS on the collection trucks.

The goal of this thesis is to develop this classification system and improve it to the point that it can confidently determine the presence of glass and/or metal in singular trash bags. Even though trash bags are emptied in batches, this project focuses on the classification of one bag at a time to prove the ease of applying deep learning in conjunction with sound and metal detection data to detect glass and/or metal. The system must still be designed so it can be installed at the back of a garbage truck, be easy to operate, and preferably be autonomous. As current circumstances (COVID 19) have prevented any prototyping on a waste collection truck, the aim has instead been to develop such a system to function on a locally simulated version of a garbage truck. This Master's Thesis is comprised of two main components:

*The design of a trash collection simulation involving sensors and a controlling algorithm*

and

*The development of a deep learning model able to predict the classes of the data collected from the trash collection simulation.*

As a result of the work performed in this project, the paper "Detecting improperly sorted content of trash bags during waste collection using convolutional neural networks" has been written and submitted for review on June 30, 2020, for the Waste Management journal. The article can be seen as an overview of this thesis, however, presents some additional details. Additionally, the article utilizes a multi-labeling approach when performing the ablation study as opposed to multi-classification. As such, the paper is added as its own section

before the conclusion.

Section 2 will give the necessary background and theory to give better understanding of the methods used in this thesis. Section 3 presents the method, development and intermediate results of the two main components of the system. Section 4 presents results and a discussion on the simulation as well as results and a discussion on the most promising models emerged from previous sections. Section 5 describes the limitations and application of the developed system and possible future improvements/additions to the system. Section 6 will give a thorough walk-through on how REN may fit the system to their garbage trucks. Lastly, Section 8 contains the conclusion of this thesis.

# 2 Background and Theory

## 2.1 Waste Collection

### 2.1.1 Collection Cycle

The main stages of the collection cycle are shown in figure 1.

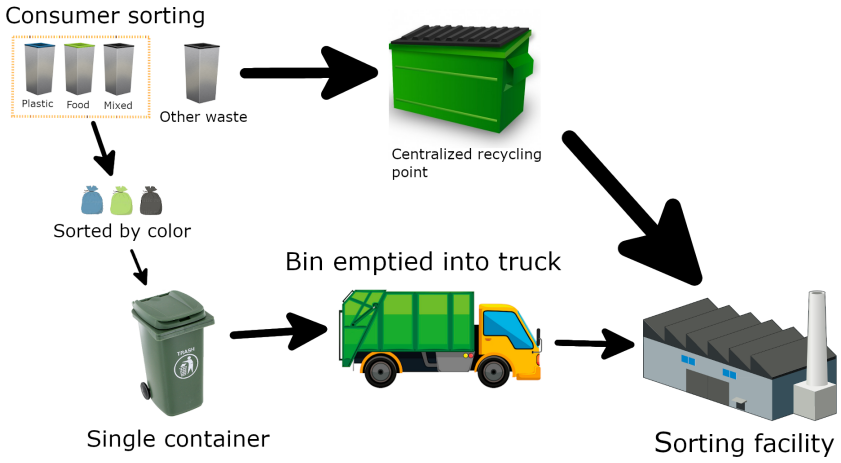



Figure 1: Collection cycle

The consumer sorts their waste according to categories. Clean plastic goes into blue bags, food waste into green bags, and mixed waste into grocery bags which are largely white in color (shown as black in the figure). Waste that does not belong in either category, i.e. glass, metal, hazardous waste and electrical waste, goes into a separate category labelled "other waste" in the figure. "Other waste" has to be disposed of at designated locations. The blue, green and white bags are placed in the consumers personal bin, or a shared bin for apartment complexes. The same bin is used for all the separate colored bags. The bins are

then emptied by a waste collection truck, who brings the collected waste from multiple locations to the sorting facility for further processing. Any information about the origins of the bags are lost after the truck has emptied bins from different locations, as the bags are mixed together inside the truck.

In the sorting facility the bags are separated by color. The blue bags are sent to a different facility for plastic recycling, the green bags are sent to a biogas production facility, and the mixed waste bags are sent to incineration.

### 2.1.2 Waste Statistics

Every year REN performs an analysis of a subset of the collected waste in Oslo. In the waste analysis from 2019 [33], 4259kg of waste in total from 10 selected areas was sorted manually and the contents recorded in detail. The distribution between green, blue and other bags are shown in figure 2.

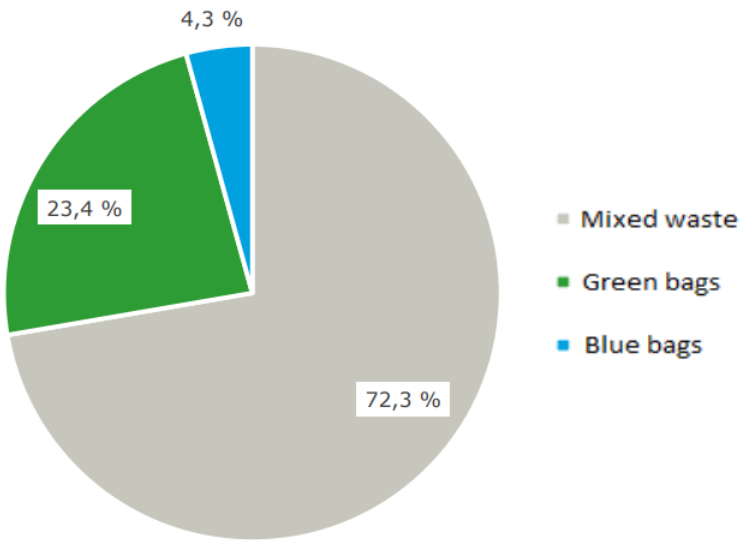



Figure 2: Distribution of bags from waste analysis, weight percent (Text in figure has been translated from Norwegian to English) [33]

According to the analysis, the amount of glass and metal waste found in the bags constitutes 4.6 weight percent, displayed in figure 3. This corresponds to a ratio of approximately $\frac{1}{20}$ between bags containing metal and/or glass to bags that are correctly sorted.

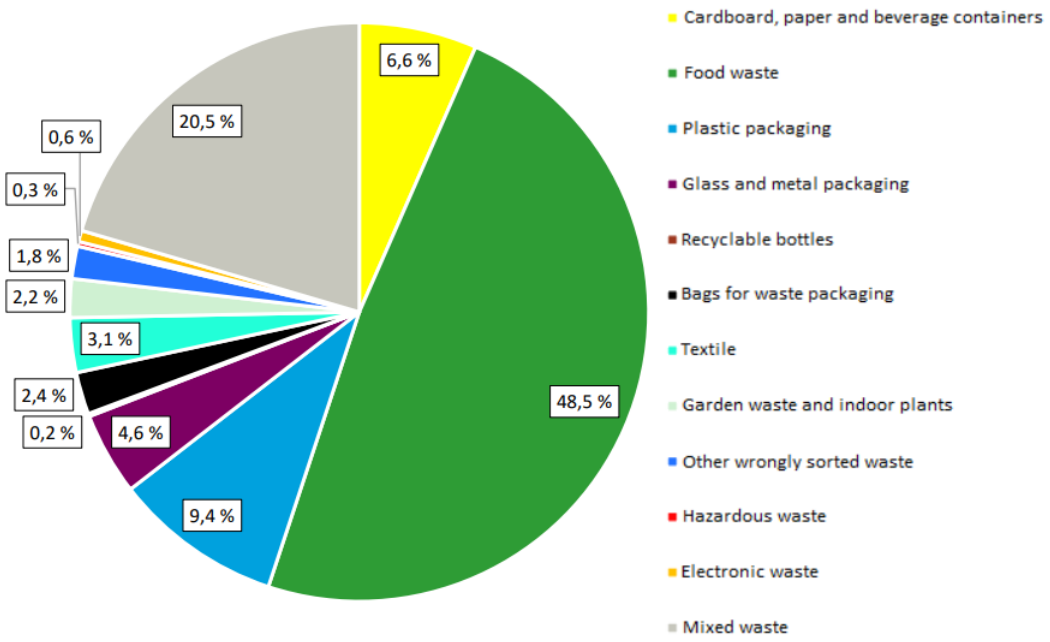



Figure 3: Distribution of materials, in weight percent (Text in figure has been translated from Norwegian to English) [33]

For the mixed waste bags alone, the amount of glass and metal is 6.1 weight percent, so these materials are most likely found in these bags. Table 4 shows the distribution of correctly sorted bags versus wrongly sorted bags for each of the 10 collection areas. This clearly shows that some areas take the sorting more seriously than others as the wrongly sorted share varies from 9.8% to 63.4%.

| Area | Correctly sorted | Wrongly sorted |
|---|---|---|
| 1 | 68,1 % | 31,9 % |
| 2 | 77,1 % | 22,9 % |
| 3 | 86,8 % | 13,2 % |
| 4 | 81,5 % | 18,5 % |
| 5 | 84,0 % | 16,0 % |
| 6 | 83,8 % | 16,2 % |
| 7 | 72,3 % | 27,7 % |
| 8 | 36,6 % | 63,4 % |
| 9 | 75,8 % | 24,2 % |
| 10 | 90,2 % | 9,8 % |
| Average | 75,6 % | 24,4 % |

Figure 4: Sorting degree for separate areas (Text in figure has been translated from Norwegian to English) [33]

## 2.2 Machine Learning

### 2.2.1 General Overview of Machine Learning

Machine learning has become a prominent field of study within computer science in later years. As advancements within the field continue, scientists and engineers are finding ever more use for it. Although its existence dates back to at least 1958 when Arthur Samuel is said to have coined the term [34], it isn't until recently that the availability of computational power and the accessibility of large amounts of data has unlocked its true potential [21].

As mentioned in the project thesis (A.3), a paraphrasing of A. Samuel often goes as follows: "Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed." In other words, machine learning is the study of algorithms that automatically adapts to complex data.

Machine learning has applications in many fields. Speech recognition, speech synthesis, and audio and music processing in sound recognition areas; document and text processing in natural language processing; search algorithms and filtration techniques in information retrieval or data mining; and object detection in image classification and computer vision [9]. Machine learning methods are great for solving complex problems which can seem trivial for humans but often difficult using conventional computer algorithms (hand engineering solutions).

Machine learning is commonly divided into two subcategories: Unsupervised learning, where no guidance is given where the algorithm learns to detect patterns and group data points without knowledge of their meanings, and supervised learning, where already recorded data along with its implications are given to the algorithm so that it may learn to predict the meaning of unseen data.

Supervised learning is further divided into regression, with a continuous numerical output, and classification, where the output is a set of probabilities. With classification, certain discrete possible outcomes (classes) are pre-defined, such that the output is the probabilities of the instance (the analyzed data) belonging to each individual class. Classification happens when, for instance, the largest probability is chosen. For example, determining the amount of time passed since an apple has been picked based on its color is a regression problem, whereas determining the type of apple based on color is a classification problem. The output of the former would be a time-unit, while the output of the latter would be the probability of the apple belonging to each of the specified types (for instance, 0.32 Aroma, 0.68 Cripps Pink).

In the example given above, the classes are mutually exclusive. That is, an apple cannot be both an Aroma and a Cripps Pink simultaneously. There are, however, many cases where an instance might belong to several classes. A trash bag may contain both glass and metal. In that case, one may specify additional possible classes describing the occurrence of both classes. For instance, the classes may be "Only Metal", "Only Glass", "Metal and Glass" and "Neither Metal Nor Glass". These would then be mutually exclusive. Another means for applying multiple labels is by use of a *multi-label classifier*. That is, the sum of the output probabilities is not necessarily one, rather, it describes individual probabilities of the instance having a certain label independent of the other labels. By defining a threshold value, which denotes the minimum confidence required for an input to be assigned a label, it may be predicted which labels are present, which might be several.

There are several machine learning models, Including Decision Trees, Support Vector Machines, Regression Analysis, Hidden Markov Model, Bayesian Networks and more. Perhaps the most popular are Artificial Neural Networks (ANN), that are loosely aimed at modelling the human brain through its artificial neurons connected to each other. There are many variations within ANNs, though it is the convolutional neural network (CNN) and its content that will be explained and explored in later sections.

ANNs must be given enough labeled data to perform adequately ([21]). It is common to split this labeled data into a bigger training set, and a smaller validation set. As such, the performance of the network can be measured on the unseen validation set. When training, a *batch size* amount of the samples is given to the network to train on. When it has trained on the whole batch, a *step* is completed. When the number of batches (steps) that constitutes the whole data set has been trained on, an *epoch* has been finished. Several epochs of training is common to gradually optimize the model while logging the performance on the validation set between a certain number of epochs to monitor and evaluate.

### 2.2.2 Convolutional Neural Networks

The CNN is a very common neural network that have shown high effectiveness in image recognition [21]. A CNN mainly consists of convolutional blocks (convolutional layers, pooling layers and dropout layers) and fully connected layers, all necessary for a complete CNN model [11]. To get a complete understanding of a convolutional neural network, an introduction to the fundamentals of a neural network will be given before each component of a CNN will be explained.

*Neural Network*
A neural network consist of multiple connected neurons each holding a number between 0 and 1. These neurons can be activated such that some of their values are transmitted to other connected neurons. The amount transmitted to another neuron is determined by the connection strength (weight) between them. Whether the combined neuron value and weight is high enough for transmission, is governed by an activation function. Such a function often accentuates high values and weakens low values, and thereby "selects" which signals are transmitted. When using *Sigmoid* as activation function all signals are transmitted but lower values will fade quickly as they progress through neurons. A neural network commonly consists of layers of neurons connected to neighbouring layers of neurons. The input layer is given the initial data, while the output layer gives the final prediction. It is the value of the neurons in the output layer that represents the probabilities of the classes being present. With additional layers in between (hidden layers), it is called a deep neural network.

Figure 5 shows the mapping of one input node, $a_1^1$ to an output layer, $\mathbf{a^3} = [a_1^3 a_2^3]$ , through one hidden layer, $\mathbf{a^2}$. The value of $a_1^2$, for instance, is determined by: the value of the neuron connected to it, $a_1^1$ and the weight, $w_{11}^1$ between them are multiplied

$$z_1^2 = a_1^1 w_{11}^1$$

which yields the output value of the input node. This value is further processed in an activation function

$$a_1^2 = \sigma(z_1^2)$$

which ultimately results in a value for $a_1^2$. Note that for a neuron getting input from several connected neurons, its value is determined by summing all the inputs, such as for $a_1^3$:

$$a_1^3 = \sum_{j=1}^{j} \sigma(z_j^2)$$

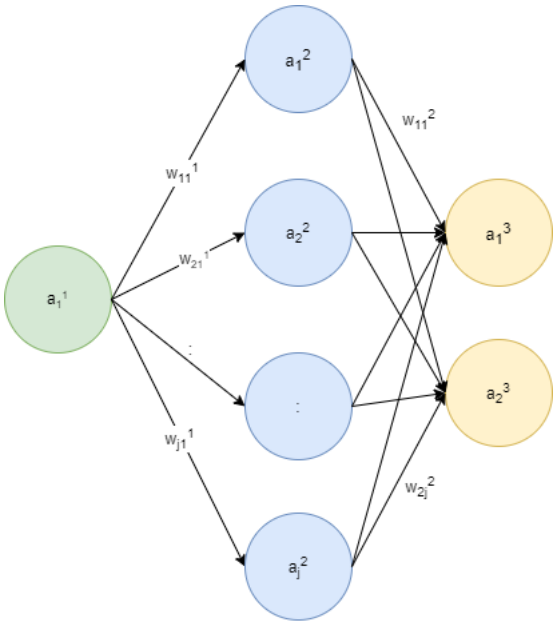



Figure 5: Neural Network

During training, the weights are altered until the input causes the activation of the neurons that ultimately results in the correct answer (decided by the developer). In a *feed forward neural network*, where all signals propagate strictly in one direction (as in Figure 5) this alteration of weights is usually done by a backpropagation algorithm [10], which will not be explained further.

*Convolutional Layer*
The convolutional layer is the cornerstone of the CNN model. Through *kernel convolutions*, these layers are able to extract distinguishing features of an image or a spectrogram. In image processing, when for instance performing edge detection, a filter is applied to an image. In such a filter, for instance the kernel

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

which is a matrix, is convolved with the image and thereby accentuate areas where pixel values drastically change. A CNN may learn these filters on its own. A convolution is a matrix operation involving two matrices, where they are multiplied elementwise and summed up, resulting in one value.

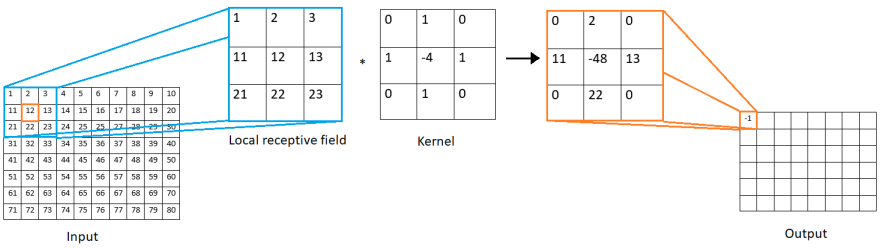



Figure 6: Kernel Convolution

In other words, the resulting value is the weighted sum of a pixel and its neighbors with weights defined in the filter. The kernel in Fig 6 with a filter as demonstrated above will stride along the input image doing the same operation on every jump until the output matrix is filled. Notice that the output will have two less rows and columns compared to the input. This is because the kernel cannot include non-existing values outside the matrix. To include the outer rows and columns in the convolution, one may for instance zero pad the matrix (add extra rows and columns of zeros) such that the kernel is able to stride the outer values as well. However, it is also common to let the spatial dimension reduce through the application of convolutions. If there are multiple matrices representing one image (*multichannel*), as for instance three (RGB), the convolution is performed on all three matrices on every stride. As such, the kernel has the same depth as the input. All values generated from each matrix are then summed up.

In a convolutional layer these filters are learned through backpropagation. The values of the filter might also differ in the depth resulting in different feature extractions for different channels as shown in Figure 7. The values in the kernel are analogous to the weights described above in that they accentuate constellations of matrix elements worth examining to reach the correct answer. Note that there is a "bias" added to the resulting sum of the kernel convolutions. A bias can be added to any neuron as the developer see fit.

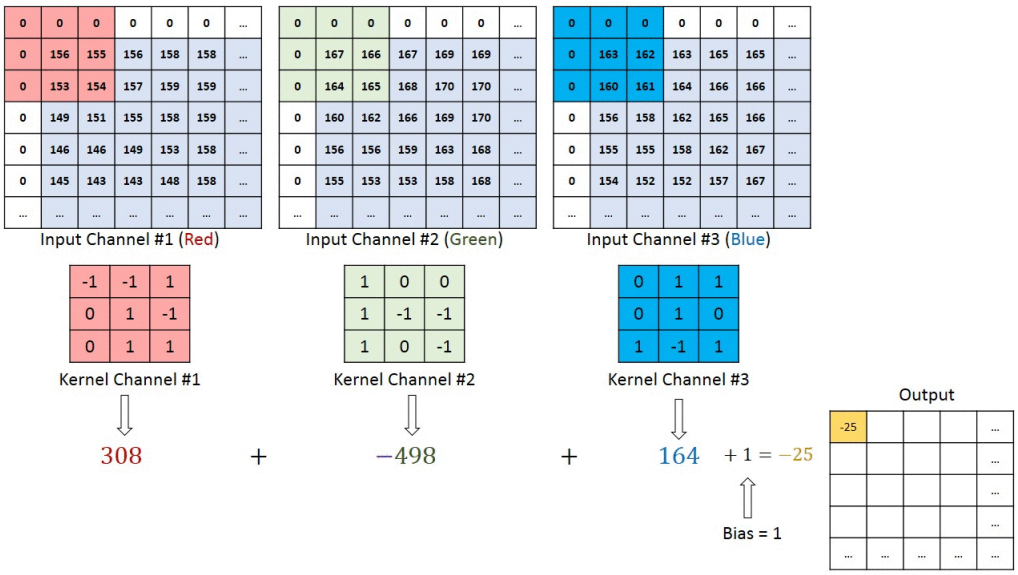



Figure 7: Multichannel Kernel Convolution

When adding a convolutional layer to a neural network, one must specify the number of filters that will be learned and the sizes of them (kernel size). If a multichannel image of size 256x256 with three channels is given, the input would be a *tensor* with shape (256,256,3). Now, if the specified number of filters is 16, there will be learned 16 filters each with a depth of 3, that when concatenated results in an output shape of (256,256,16). As such, each channel is given some separate attention. In sound recognition, these channels might be spectrograms (Figure 16) of the same sound event but from different microphones. Forward feeding this to a new convolutional layer, would results in kernels in having a depth of 16, as illustrated in Figure 8.

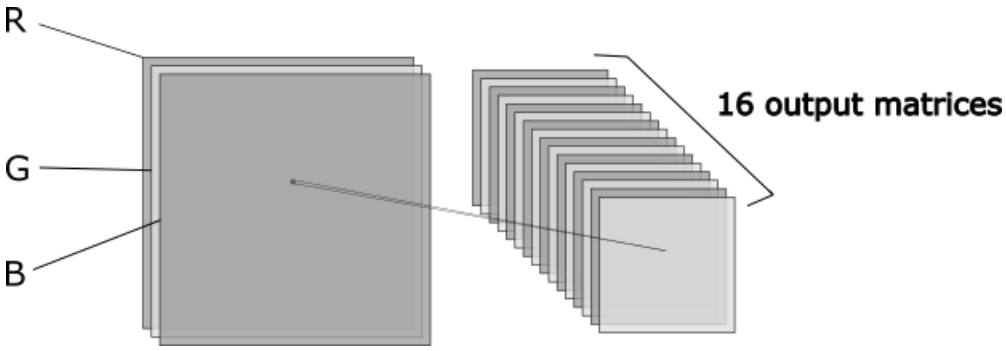



Figure 8: Input dimensionality and resulting output dimensionality of a convolutional layer with 16 filters

*Activation Function*
After convolutions are performed and an output matrix is produced, the values of the output matrix are processed through an activation function in the same manner as described above. This results in a matrix of the same size, but with altered values according to the activation function. Such an output is called an activation/feature map and contains the features the convolutional layer has extracted. The Rectified Linear Unit (ReLU) function is very common in conjunction with convolutional layers due to its computational efficiency [21].

*Pooling Layer*
When for instance applying 3x3 kernel convolutions in the convolutional layer, relatively small and precise features of a matrix are extracted (features the size of 3x3). In for instance image recognition, a small rotation of the object might result in completely different filters being generated even though the object is the same. To battle this, pooling is applied after convolutional layers. This reduces the resolution of the image by merging patches of matrix elements (commonly with the size of 2x2). This is often done by calculating the average value of the four matrix elements (*Average Pooling*), or by applying the maximum value found in the patch (*Max Pooling*). The first method will present the most average features, while the second accentuates the most prominent features. As such, the convolutional layer will increase a tensor's depth while the pooling layer will decrease its height and width.

*Dropout Layer*
When training a neural network to reach a specific output, it may find one path through is network to the output and stick only to that path. This may cause problems if the network is introduced to new data that represents the same class but in a slightly different manner. The network is in other words highly proficient at classifying the data it has trained on but will perform poorly when encountering new data that does not exactly match any of the training data.

This phenomenon is termed *overfit*. A dropout layer [39] randomly chooses neurons to be ignored at each epoch, such that the network is forced to find new paths to the same goal, and thereby making the model more robust and able to generalize.

*Fully Connected Layer*
A CNN is not able to land on a specific class with only convolutional blocks. The output of these are also matrices (feature maps) such that a layer that outputs single values containing class probabilities, is needed. In CNN, such a layer is often a fully connected layer. Fig 5 shows examples of fully connected layers. All neurons in layer 1 are connected to all neurons in layer two, which are all connected to all neurons in layer 3. Hence "fully connected". Since this layer requires input in the form of a 1-dimensional vector, the feature maps coming from the last convolutional block can be sequentially fed to the fully connected layer (flattened). Another common approach is to apply a global average pooling layer, which calculates the average of a complete feature map and adds it as one element to a vector. As such, a vector with the same length as the number of feature maps in the last convolutional block is produced and fed to the fully connected layer.

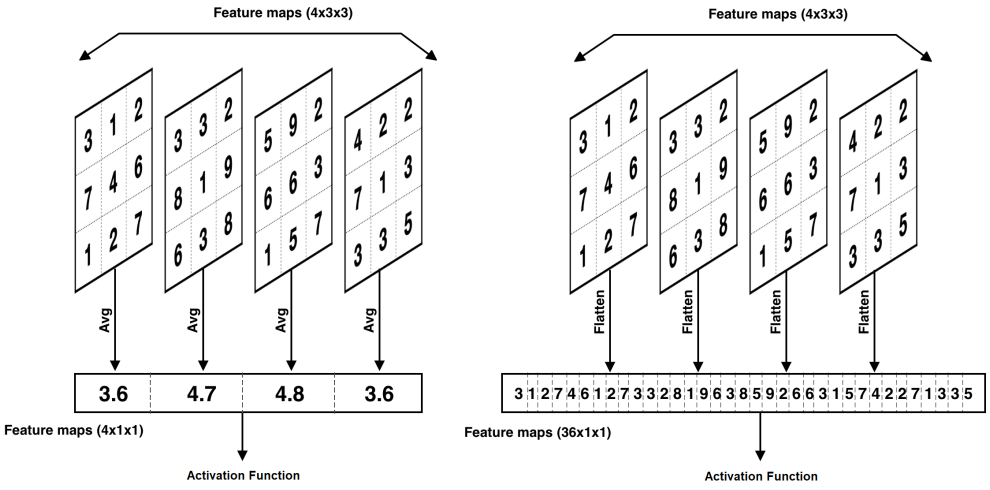



Figure 9: Global averaging vs Flatten

An example of a complete CNN architecture is shown in Figure 10. The features are extracted in the convolutional blocks followed by a classification based on the features in the fully connected layers.

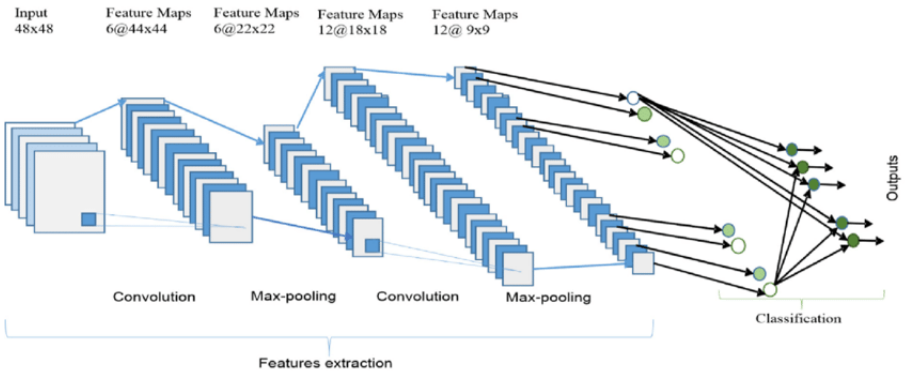



Figure 10: Complete CNN architecture [5]

### 2.2.3 Keras and TensorFlow

Every layer and operation described in the previous section must be coded in a programming language somehow. Since many people have done this before, it is unnecessary to do this every time. Therefore, machine learning *frameworks* exist. These are libraries where operations and layers described above are already coded into easily accessible objects and functions (API), such that one may for instance create a convolutional layer using one line of code.

There are many different frameworks able to perform the same tasks. These often differ in the way they are structured, how they initialize the weights (static and dynamic graph (Graph Neural Networks: A Review of Methods and Applications). Some also enable the GPU to help perform calculations such that both training and inference are performed quicker. Only TensorFlow and Keras (based on Tensorflow) will be explained further.

*TensorFlow*
TensorFlow is a machine learning framework developed by a research team at Google for internal use, that has since been released to the public. The framework is now free and open source [14]. The library offers source code for many different machine learning methods, including neural networks. It supports both extensive control with low-level API as well as easy-to-use high-level API (tf.keras). The framework offers code able to run on CPU, GPU and TPU. TPU (tensor processing unit) is a chip specialized at performing tensor operations.

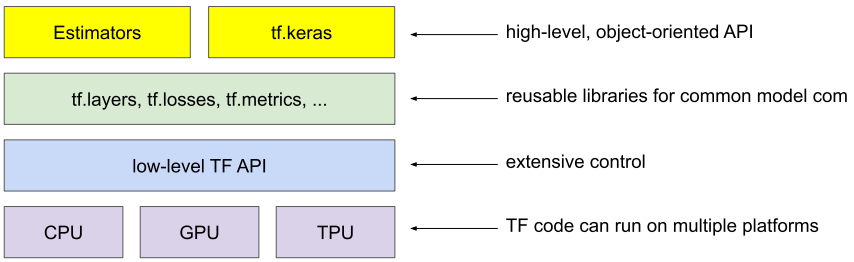



Figure 11: TensorFlow Hierarchy [10]

*Keras*
Keras is an API developed specifically for neural networks [36]. Originally Keras was able to utilize several different low-level API's such as TensorFlow, Microsoft Cognitive Toolkit, R, Theano, and PlaidML. In other words, it was an API that presented other frameworks in a more high-level manner. Google has since supported the library and added it to TensorFlow as their high-level API.

Through its easy-to-use high-level representation of neural networks, it allows for rapid prototyping and fast experimentation. The implementation of each component described in Section 2.2.2 will be briefly presented and explained.

*The model*
The very first operation performed, is the creation of the model object:

```
model = Sequential()
```

`Sequential()` is the name of the class that represents sequential models. A model is sequential when all layers are added sequentially after each other (no parallel layers).

*Convolutional layer*

```
model.add(Conv2D(filters=32, kernel_size=(3,3), strides = (1,1),
    input_shape=(100,256,256,3), activation='relu'))
```

This convolutional layer will learn 32 filters for each channel all with a 3x3 size as defined by the kernel size. The `strides` = (1,1) describes the amount of elements to jump between every convolution. As described in the previous section, the kernel strides along the matrix until the whole matrix is convoluted. With a stride of (2,2), the kernel will perform convolutions on every other matrix element and its neighbors along one row and skip the entire next row. `input_shape` = (100,256,256,3) specifies the shape of the input tensor. In this case, there are 100 samples, having 256x256 matrix elements, with a depth of 3 (for instance rgb). `activation='relu'` defines which activation function to use, in this case, relu is chosen.

*Pooling layers*

```
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.add(AveragePooling2D(pool_size=(2,2)))
```

A max pooling or an average pooling layer can be added. pool_size = (2,2) describes how many matrix elements will be concatenated. A pool size (2,2) will cause the resolution of the image to be halved in both height and width.

*Dropout layer*

```
model.add(Dropout(0.2))
```

This dropout layer will randomly set 20% of its input units to 0 on every step during training. As such, the selected units are effectively ignored during that specific step.

*Fully connected layer*

```
model.add(Dense(units=3))
```

A fully connected layer in Keras is termed Dense. units=3 defines the number of neurons, in this case 3. Note that for the output coming from the convolutional block to fit as input to this layer, either a *global average pooling layer*

```
model.add(GlobalAveragePooling2D())
```

or a *flatten layer*

```
model.add(Flatten())
```

must be added before the fully connected layer.

*A complete model*

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3,3), strides = (1,1),
    input_shape=(100,256,256,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=(3,3), strides = (1,1),
    activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=(3,3), strides = (1,1),
    activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(GlobalAveragePooling2D())

model.add(Dense(units=3,activation='softmax'))
```

The code above shows an example of a CNN model architecture created using Keras. Notice that only the first convolutional layer needs to know the input shape. Input shapes in-between layers are calculated and defined automatically.

To train the model, the function

```
model.fit(x_train, y_train, batch_size=32, epochs=100,
    validation_data=(x_val, y_val))
```

is called. Here the batch size and amount of epochs (2.2) are specified.

### 2.2.4 Evaluation Metrics

When evaluating an artificial neural network on a certain set of data, it is common to utilize standardized metrics to measure its performance. There are many metrics to consider, all representing different nuances of the performance of the algorithm.

Most of the metrics used are defined by the contents of a confusion matrix. A confusion matrix in relation to machine learning shows the algorithm's predictions on a data set. How many positive predictions were right or wrong, and how many negative predictions were right or wrong. In other words, true and false positives, and true and false negatives. Such a confusion matrix is shown in Table 1.
*Multi Class*
If the problem is not binary and there are several classes, like in a multi-class classification, there should be more rows and columns to the confusion matrix. An example is given in Table 2.

Table 1: Confusion Matrix

| | Predictions | |
|---|---|---|
| Truth | True positives | False Negatives |
| | False positives | True Negatives |

Table 2: Multi Class confusion matrix example

| | | Predictions | | | |
|---|---|---|---|---|---|
| | | Class 1 | Class 2 | Class 3 | Class 4 |
| Truth | Class 1 | 8 | 2 | 2 | 1 |
| | Class 3 | 2 | 7 | 2 | 1 |
| | Class 3 | 0 | 0 | 11 | 2 |
| | Class 4 | 2 | 0 | 1 | 10 |

Here all values on the diagonal are True Positives(TP) while values anywhere else constitutes False Positives(FP).

Table 3: Multi class confusion matrix with TPs and FPs

| | | Predictions | | | |
|---|---|---|---|---|---|
| | | Class 1 | Class 2 | Class 3 | Class 4 |
| Truth | Class 1 | TP | FP | FP | FP |
| | Class 2 | FP | TP | FP | FP |
| | Class 3 | FP | FP | TP | FP |
| | Class 4 | FP | FP | FP | TP |

Notice that Table 3 has no True Negatives(TN) or False Negatives(FN). TN and FN are specific to classes. For instance, if a sample has class 1 as true label, but is predicted as class 2, it is an FP from an overall point of view. Class 1 sees this as an FN, while class 3 and 4 sees it as TN. Figure 12 shows how predictions are separated into TP, FP, FN and TN for each class.

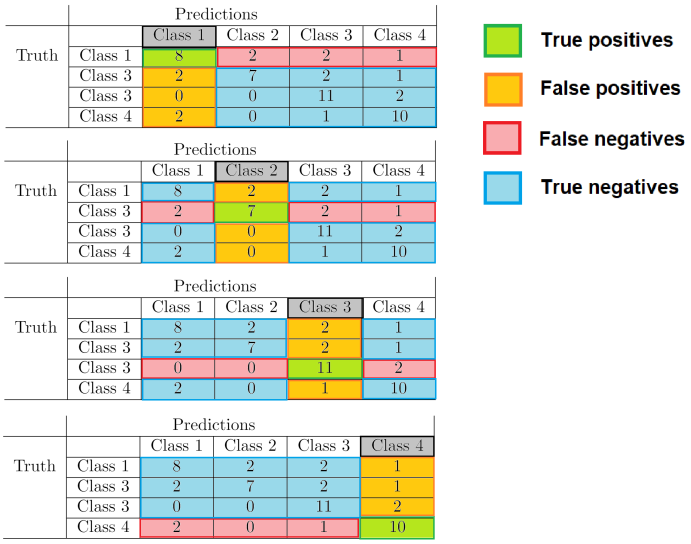



Figure 12: TP, FP, TN and FN in multiclass

*Multi Label*
In the case of multi label, it is common to have one confusion matrix as in Table 1 for each class, as for instance with two classes as shown in Table 4. Here the presence of class 1 and 2 are termed Pos C1 and Pos C2, while the absence of class 1 and 2 are termed Neg C1 and Neg C2.

Table 4: Confusion matrices for two classes

| | | Predictions | |
|---|---|---|---|
| | | Pos C 1 | Neg C 1 |
| Truth | Pos C 1 | TP | FN |
| | Neg C 1 | FP | TN |

| | | Predictions | |
|---|---|---|---|
| | | Pos C 2 | Neg C 2 |
| Truth | Pos C 2 | TP | FN |
| | Neg C 2 | FP | TN |

#### 2.2.4.1 Accuracy

Perhaps the most common metric is the accuracy and it is widely used when bench marking models. It can be described as the portion of correct predictions among all predictions made. It is usually the accuracy that is monitored while training a neural network.

$$Accuracy = \frac{Correct\ Predictions}{All\ Predictions} = \frac{TP + TN}{TP + TN + FP + FN}$$

#### 2.2.4.2 Loss

Loss is a metric of how good the model is at fitting the examples provided. There are several different functions used to calculate the loss, so there is no single correct way of doing it. A decreasing loss indicates that the model is improving, to a certain degree (see overfit below). A widely used function is "Mean square error", shown in equation 1, which calculates the average loss for all examples.

$$MSE = \frac{1}{N} \sum_{(x,y)\epsilon D} (y - prediction(x))^2 \tag{1}$$

#### 2.2.4.3 Precision

Precision measures the percentage of positive identifications that were actually true. This is calculated using equation 2.

$$Precision = \frac{True\,positives}{True\,positives + False\,positives} = \frac{TP}{TP + FP} \tag{2}$$

#### 2.2.4.4 Recall

Recall measures the percentage of positive samples that were identified as positives. This is calculated using equation 3.

$$Recall = \frac{True\,positives}{True\,positives + False\,negatives} = \frac{TP}{TP + FN} \tag{3}$$

#### 2.2.4.5 Micro and Macro Averaged

The overall accuracy, precision and recall of a multi-labelling model is commonly averaged in two different ways. A *macro averaged* accuracy, precision or recall assumes equal weight to all classes, such that the overall score is simply the average of the individual class scores. With *micro averaged* scores each individual class score is weighted according to how many samples of the class are in the data set.

#### 2.2.4.6 Overfit

An overfit model means a model that has achieved a very low loss but does a poor job of predicting new data. This happens because the model has become too specialized in predicting the provided examples. There are several ways to prevent overfitting. The first is to include a large sample size, more samples decrease the probability of the model becoming too specialized on the insignificant features found in the samples. It is also important to ensure that the samples have a wide variation, representing most of the possible examples that can be encountered. Another factor is to train the model for the right amount of epochs. Training too long can result in overfitting as the model starts to go

too deep into insignificant features.

### 2.2.5 Artificial Intelligence in Waste Management

Several different approaches to trash classification has been investigated. A project thesis from Stanford by Yang and Thung [42] uses image classification to categorize different classes of waste. Their data set contains about 2,400 images sorted into six classes: glass, paper, cardboard, plastic, metal and trash. The best accuracy achieved was 63%. This project and its dataset have been used by others, including Adedeji and Wang [3] for the same purpose, achieving an accuracy of 87%. While this approach seems to work well, it is not very applicable to the collection cycle at REN, where the trash is contained within plastic bags. Since the contents of the bags cannot be seen with regular imaging methods, image classification will not be applicable. In addition, the dataset used in these projects contain images on a uniform background, which is probably quite different from real world scenarios.

A study by Korucu et al. [19], uses sound recordings of different materials taken during free fall impact, impact from a pneumatic cylinder and hydraulic crushing to train a neural network to separate between material types. The paper shows good results, although similarly to the image recognition approach, it is not directly applicable to the current problem as the objects in this study were individual instead of mixed together inside a plastic bag. The main concept is however quite similar, which is sorting between materials based on sound recordings. The study is also different from the approach proposed in this thesis in that they only utilized one type of input data, i.e. images or sound, and not a combination of different sources.

Utilizing multiple sources of data to classify waste has been attempted by Chu et. al. [8] where a hybrid deep-learning system was developed to sort waste as recyclable or not. In this study a camera, a weight scale and a metal detector were used to gather the input data for their deep-learning model. For feature extraction and classification, several CNN layers were used to extract features from images taken by the camera. 22 outputs from the last CNN layer were then combined with the data from the weight scale and metal detector in a fully connected layer (see Section 2.2.2) that in turn resulted in a classification. Achieving over 90% accuracy, which they claim was significantly better than reference models, they showed that including several sources of data could be beneficiary when classifying waste. This study also collected data from single objects placed in front of a uniform background, again making it less applicable to REN's collection cycle.

The system developed in this project is aimed at detecting unwanted materials without the need for removing the waste from the plastic bags. It also provides the option of recording information about where the unwanted materials were

collected, presenting possibilities for mapping behavior patterns in certain areas and more. While the system is meant to work with the trash contained in a bag, it does not exclude its use from situations where waste is loosely contained, for instance if transported along a conveyor belt.

### 2.2.6 Machine Learning for Sound Recognition

The use of sound data in conjunction with machine learning is a well studied field with applications such as speech recognition, audio surveillance, environmental sound recognition and sound event recognition [35]. In speech recognition, Hidden Markov Models (HMM) and Gaussian Mixture Models (GMM) are frequently used machine learning models ([9], [41], [45]), however, other deep learning methods has been applied successfully as well ([26], [24]).

Using machine learning for recognizing the source of a sound with regards to material has been well studied. Perhaps the most relevant is the earlier mentioned article by Korucu et al. [19] that achieved a 97.7% accuracy using HMM. Luo et al. [25] successfully developed a deep learning model that with 91.5% accuracy is able to recognize which object is being struck by a marker pen. Gong et al. [13] developed an SVM classifier that could recognize the material of an object (98% accuracy) using data generated by the microphone, gyroscope and accelerometer on a smartphone when knocked against the object.

Using CNNs in sound recognition is not new either. There is no doubt of the performance of CNNs when it comes to image recognition ([21]), so that applying a CNN on an "image" of a sound (e.g MFCCs and/or Mel Spectrograms described in the next section) could very well be applicable. Hersey et al. [17] show promising results using CNN in sound recognition using MFCC. CNNs are better at generalizing by ignoring local variations ([7]) compared to traditional models, such as HMMs. CNNs may therefore perform better in chaotic environments ([43]) where the distinguishing features are subtle, for example when unloading a bin of trash bags into the back of a waste collection truck. A lot of irregular acoustic noise can be expected when collecting waste, which a CNN model may be better at filtering out.

## 2.3 Sound Preprocessing

When using sound for machine learning, the sound clips must be preprocessed into something the algorithm can use. The usual approach is to convert the sound to spectrograms that describe features of the sound in terms of values. Two types of spectrograms are usually applied, the Mel-Spectrogram and the Mel Frequency Cepstral Coefficient spectrogram (MFCCS for short). In addition to converting sound to values in a matrix, this processing can also highlight and scale the information contained in the audio in different ways. Depending on the method used this can for instance highlight changes in the sound, which might be an important factor when distinguishing between different sounds.

### 2.3.1 Mel-Spectrogram

The Mel-spectrogram describes the strength of frequencies at specific times in an audio clip. The spectrogram is created by first dividing the sound clip into time segments. A fourier transform is then applied to each of these time segments, which yields information about the strength of each frequency present in the time segment. A plot of a fourier transform on a single segment can be seen in figure 13. The frequency of the plot is along the x-axis and the strength along the y-axis.

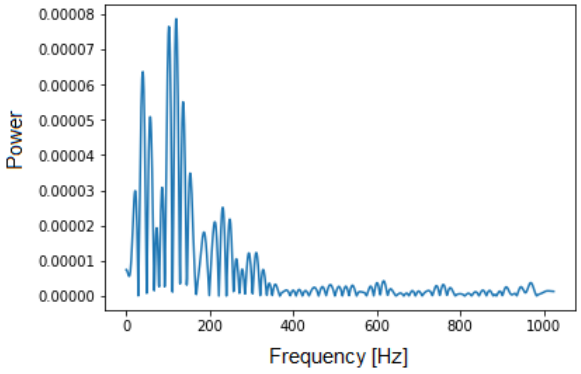



Figure 13: Fourier transform of single time segment [12]

Each of these fourier transforms output a vector with length equal to the number of frequencies described. Each index of the vector corresponds to a frequency, and the value at that index the strenght of that frequency. By transposing each of the vectors and combining them into a 2D-matrix, a representation of frequency strengths at different time segments in the audio clip is created. A 2D-plot of such a matrix is shown in figure 14, this is a frequency spectrogram where the color of each pixel represents the strenght of the corresponding frequency at one time segment.

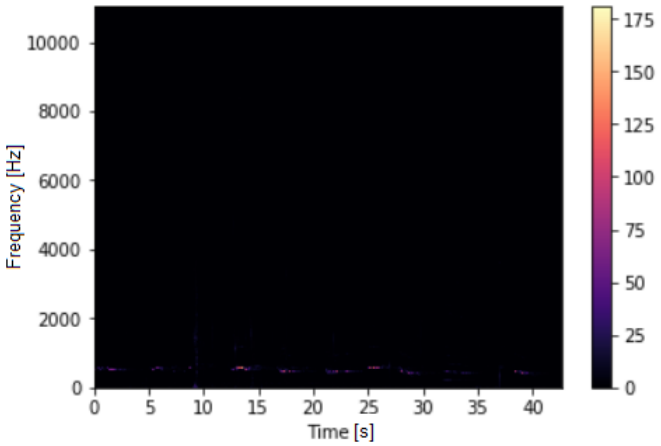



Figure 14: Frequency spectrogram [12]

As can be seen in figure 14, the high frequencies are almost non-existing compared to the low frequencies. This is due to the power scaling being linear. In the Mel-Spectrogram, the y-axis and amplitude (color axis) are scaled. The frequencies are log-scaled and the amplitudes are scaled to Decibels which is the most common way of scaling the volume of sound. Applying this scaling results in figure 15.

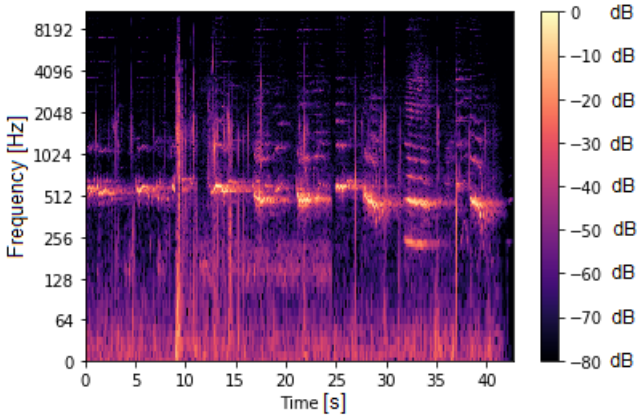



Figure 15: Log-scaled spectrogram [12]

A Mel-Spectrogram is almost the same as the one in figure 15, but with a slightly different log-scaling of the y-axis. The scaling used is the Mel-scale, created by Volkmann, Stevens and Newman [40]. This scale is based on which increments in frequency that listeners observed as equal increments in pitch. The Mel-Spectrogram, which also includes power scaling in dB, is shown in

figure 16.

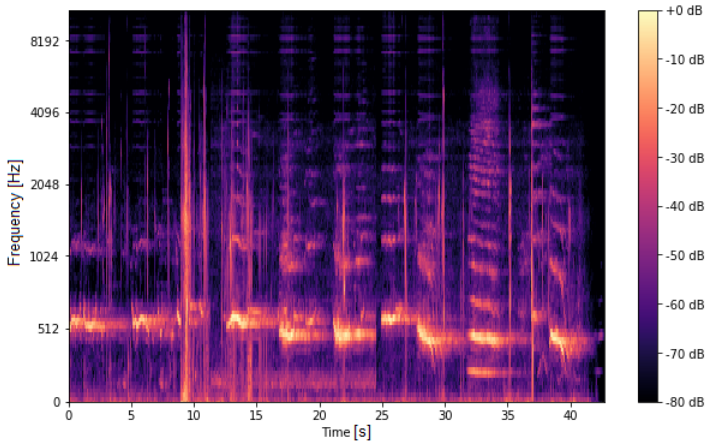



Figure 16: Mel-Spectrogram [12]

### 2.3.2 Mel Frequency Cepstral Coefficient spectrogram

The MFCC spectrogram is obtained by applying a linear cosine transform to the Mel-spectrogram. The result is something called a Cepstrum. The Cepstrum shows peaks where there are periodic elements in the sound clip [27]. The y-axis of this plot has changed due to the operations performed, and are now in the "Quefrency" domain, a coin termed by Bogert et al. [29]. This spectrogram is complicated to understand, but is frequently used in deep learning when dealing with sound. This is especially true for speech recognition. Figure 17, 18, and 19 show a sound clip used in this thesis plotted as a waveplot, a Mel-spectrogram, and a MFCC spectrogram respectively.

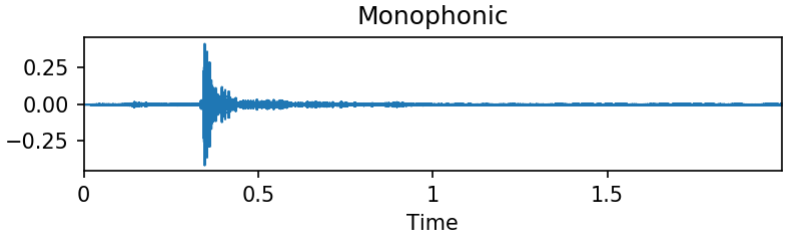



Figure 17: Waveplot

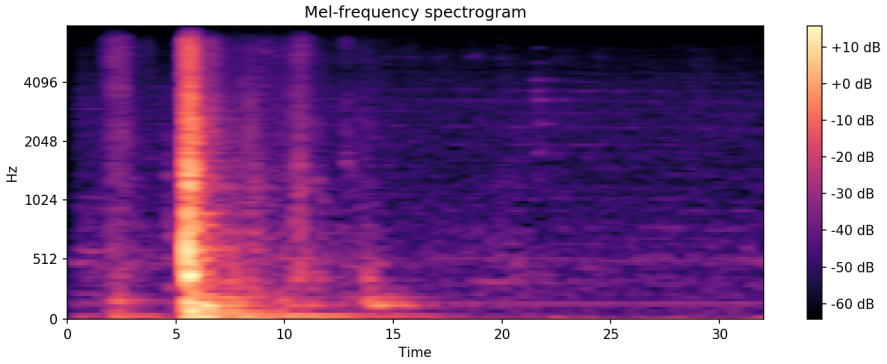



Figure 18: Mel-spectrogram

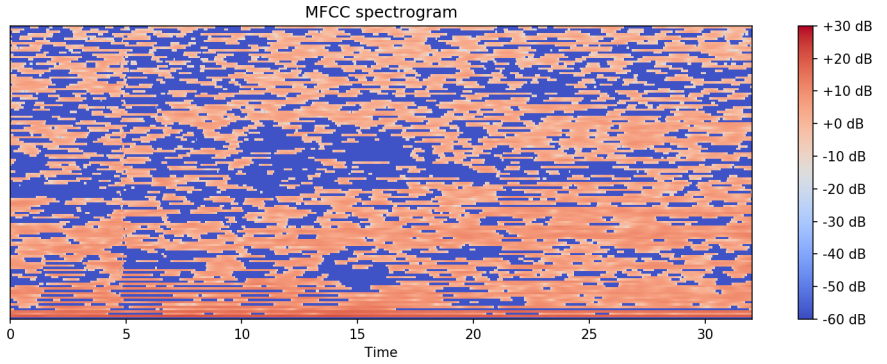



Figure 19: MFCC spectrogram

## 2.4 Metal Detection

The metal detector used in this project is derived from a DIY project found at All About Circuits by Evan Kale [18].

The detector consists of two main components, a Colpitts oscillator and an Arduino. The main function of the oscillator is to create a frequency that the Arduino can compare to a stored frequency, where any deviance will indicate a presence of metal. The Colpitts oscillator is a circuit that utilizes a combination of inductors and capacitors to produce an oscillating voltage at a certain frequency depending on the inductance and capacitance of components used. The frequency can be determined using formula 4 [31]. The main component of this ciruit is called the Tank Circuit, shown in figure 20 below.

$$f = \frac{1}{2 * \pi * \sqrt{L * C}}[Hz] \tag{4}$$

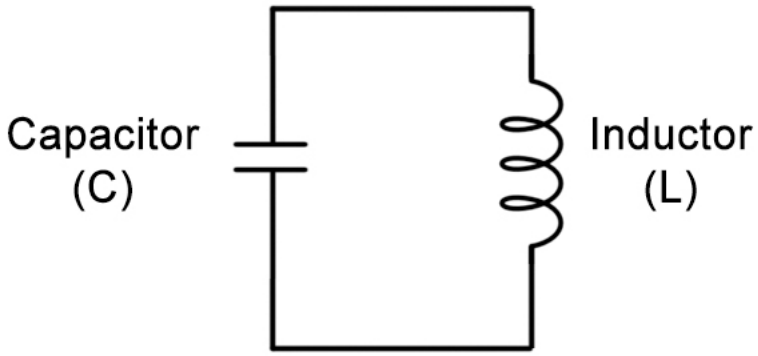



Figure 20: Tank Circuit [18]

The capacitor will discharge causing the coil to develop a magnetic field. Once the magnetic field has more energy than the capacitor, the coil will begin to induce a current which will charge the capacitor. This causes the current to oscillate back and forth which creates a fluctuating voltage in the circuit. In theory this could continue indefinitely, but the internal resistance in the components will cause some heat development resulting in energy being lost. Because of this it is required to continuously feed the circuit from an external power source. This is accomplished using a BJT inverting amplifier. The circuit for the oscillator used in this project is shown in figure 21.

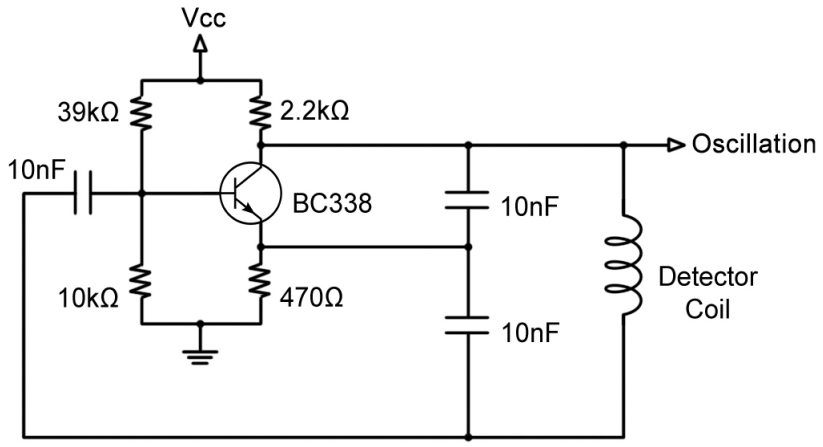



Figure 21: Colpitts Oscillator Circuit [18]

When the arduino setup runs, the frequency of the Colpitts oscillator is measured and stored. While the program runs it will continously compare the oscillator frequency to the stored frequency and output the difference. Deviance in the frequency will occur if metal is present near the coil as it will change its inductance and in turn change the frequency of the oscillation. The reason for change in inductance happens for one of two reasons. Either a ferromagnetic metals magnetic field aligns with the magnetic field of the coil, increasing the inductance, or a non-magnetic metal decreases the inductance due to Eddy Currents being induced in the metal which counteract the field. The coil (yellow wiring) used is shown in figure 22.

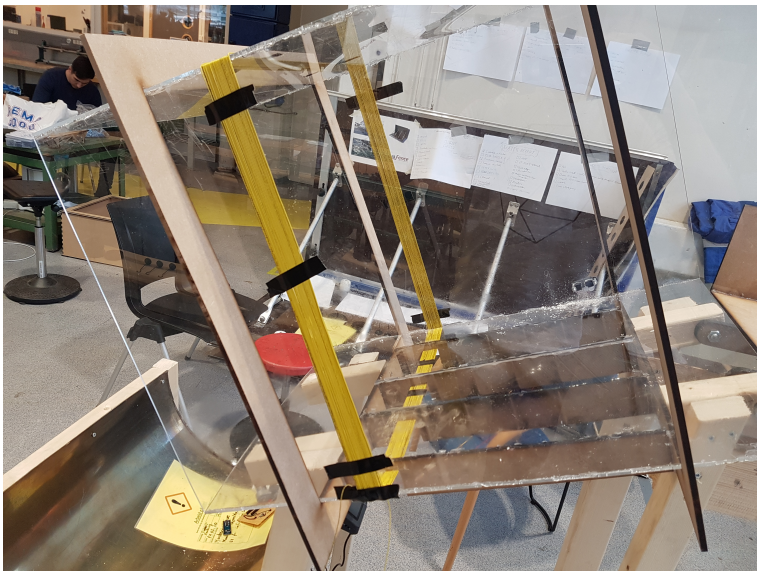
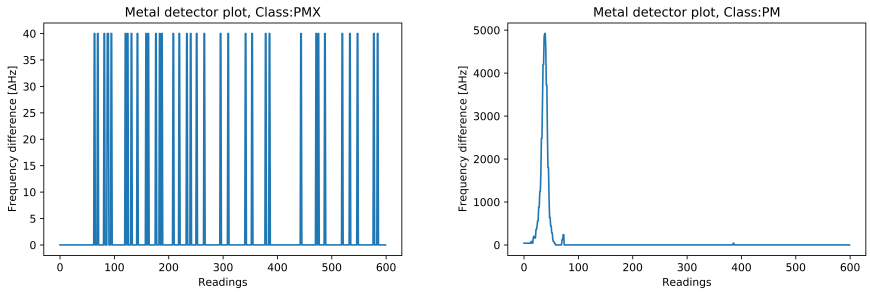


Figure 22: Metal detection coil on test setup

The inductance change in the coil is largely affected by three factors. Type of metal, amount, and orientation of the object. Different metals have different densities and magnetic properties, and therefore affect the inductance differently. Higher amounts have stronger effects, and objects oriented in the same plane as the coil will affect the inductance more than objects normal to the plane. Because of this, it is not possible to determine any of these factors from the data alone. Therefore, the metal detector is most effective at determining whether metal is present or not. Some idea of the properties can be deduced however. An object normal to the plane will cause a smaller reading than the same object aligned with the plane, but the reading will be spread over a longer time period. This change is very small, and hard for a human operator to interpret, but a computer might be able to distinguish between cases.

Examples of readings from the detector in figure 22 are shown in figure 23, notice the difference in the y-axis in the two plots. The spikes in the mixed waste plot are caused by unidentified noise, but are small and should not prove significant when machine learning is applied.




(a) Reading from mixed waste

(b) Reading from pure metal waste

Figure 23: Metal detector readings

# 3 Development

This section aims at giving both a description of the development as well as the reasoning behind any choices made during the process. Intermediate results will be presented in this section as the development process is based on an iterative procedure. The system involves the development of two main components as described in Section 1, and will be presented accordingly.

## 3.1 Trash Collection Simulation

When training a machine learning model it is necessary to have many examples of each class so the model can learn to differentiate between the data belonging to each class. The best way to obtain this data would be to implement the system at the location it would later be used, which is at the back of a garbage truck. As we did not have access to a truck for as long a period as we would need, we decided instead to build a measuring rig that would mimic the emptying of trash bins into the truck. This rig included all the sensors we would like to include in the final system, or a suitable replacement. These sensors are as follows:

- Weight sensor
- Proximity sensor
- Metal detector
- Sound recorder
- Video recorder

For all listed components, except the metal detector, existing sensors were chosen. The metal detector was built locally and is presented in Section 3.1.1. In

addition to these components, an Arduino was included to automate the recording process, as manually controlling the recording equipment would significantly slow down the operation and require a lot of post-processing of the data. All components are shown in figure 24.

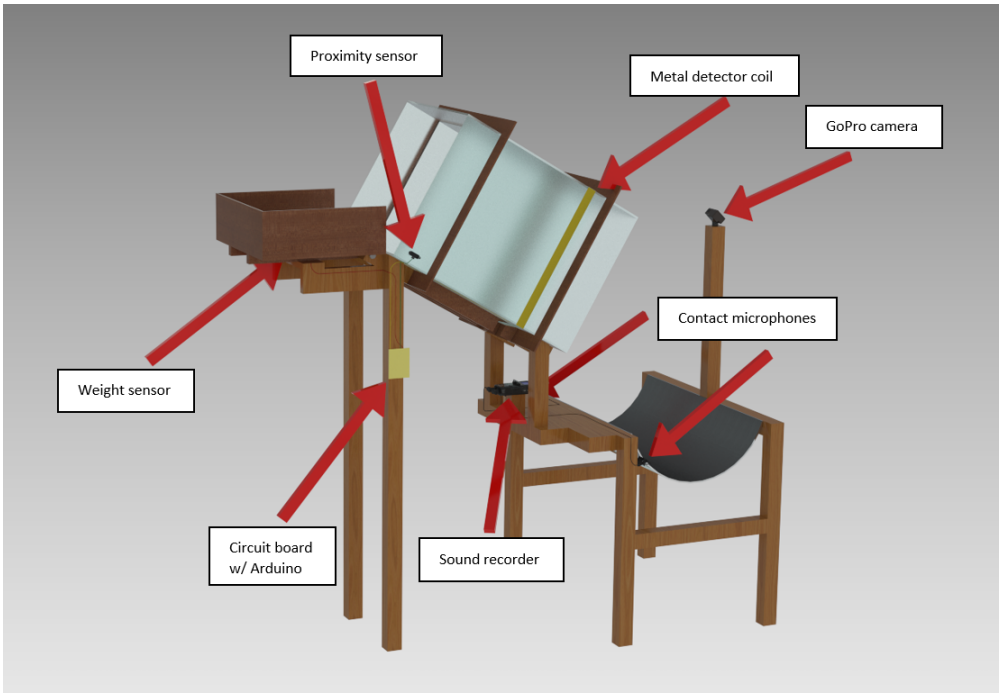



Figure 24: Test rig

The system operates as follows. A computer is connected to the Arduino and sound recorder via usb cable, and to the GoPro camera through WiFi. A program specifically made for this operation is run on the computer, which launches a graphic user interface to assist in the measurements (see section 3.1.6). Next a trash bag is placed in the tray at the back of the rig (left side in figure 24). The weight sensor will record the weight of the bag and a cue light will show in the GUI when the tray is ready to be flipped. At the same time the camera will start recording. The operator then flips the tray which causes the bag to fall down the chute and trigger the proximity sensor. This in turn triggers a python script on the computer that starts recording sound from microphones, and the arduino to send data from the metal detector. The bag lands in the metal tray, and the sound and video recording stops. These recordings are set to stop after a specified time has passed since the proximity sensor was triggered. All recorded data is temporarily stored, and when ready the GUI shows a spectrogram plot of the sound recorded and a plot of the data from the metal detector. If the operator is satisfied with the measurements,

he presses a button in the GUI which saves the data to a folder structure on the computer. During this save process the video recording is cut to remove the part from before the proximity sensor was triggered. When all the data is saved, the rig is ready for the next measurement. Each component is explained in more detail in the sections below, and figure 25 contains a flowchart of all the stages in the program.

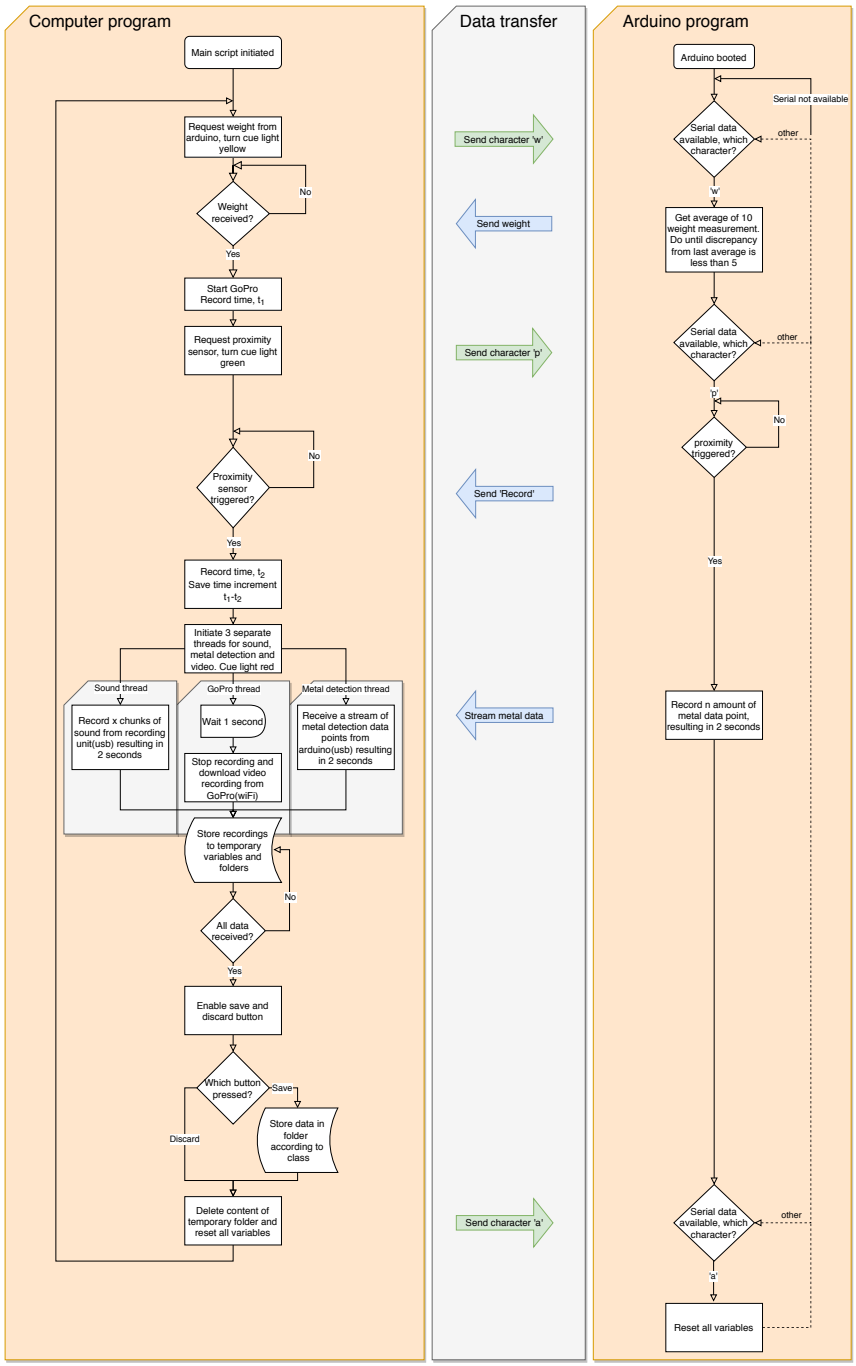
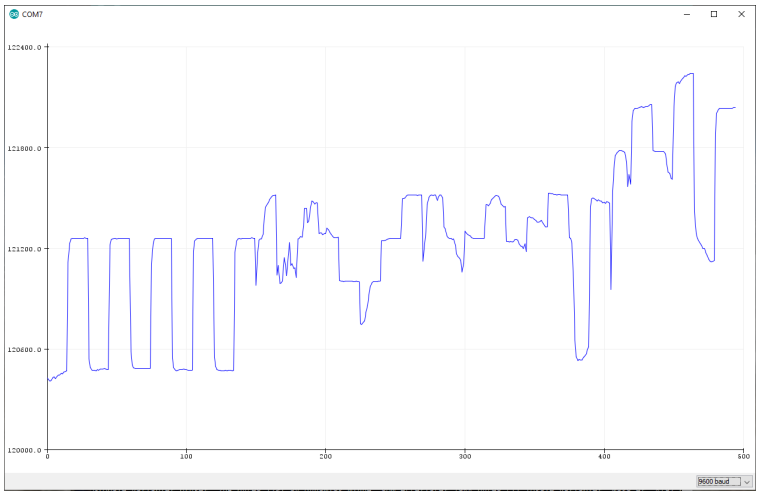



Figure 25: Computer- and Arduino Interaction

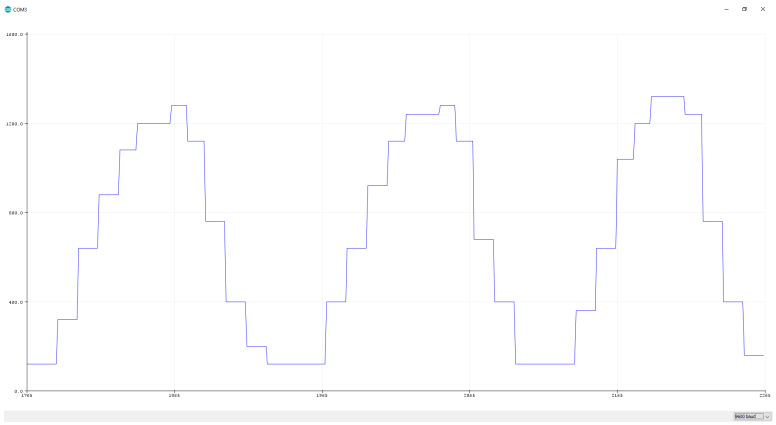
### 3.1.1 Metal Detector

The original plan was to use a Pulse Induction metal detector for this project. This detector is described in detail in the project thesis A.3 which the masters thesis is based on. The previous detector used a small coil with very limited range, so attempts were made to scale it up as a greater range or size would be required for it to function due to the size of the trash bags. None of these attempts were successful. The detection range was limited to a few cm from the windings of the coil, and attempts to change coil diameter, number of windings, or increased amperage or voltage did not seem to have any measurable effect. Therefore another type of detector was tested, which yielded much better results. The new detector type is called a beat-frequency oscillation detector, and the working principle is described in the theory section (2.4).

The detector exhibits a larger range of about 10cm, and also produced more stable readings than the previous type used. The readings proved consistent, and noise is minimal, resulting in very clear indications when metal is present. Two images comparing the old and new detector readings are shown in figure 26.




(a) Old metal detector readout as shown in the Arduino serial plotter




(b) New metal detector readout as shown in the Arduino serial plotter

Figure 26: Old vs new metal detector

The old detector was erratic in that the value when no metal was present was constantly changing. This could cause confusion for the machine learning algorithm and decrease its performance. The output was also much less consistent when the same metal object was applied, so the new detector was a big improvement. Readings obtained from metal object generally produce amplitudes between 500 and 5000, where the amplitude refers to the change in oscillation frequency. Any noise experienced generally produced amplitudes of around 40.

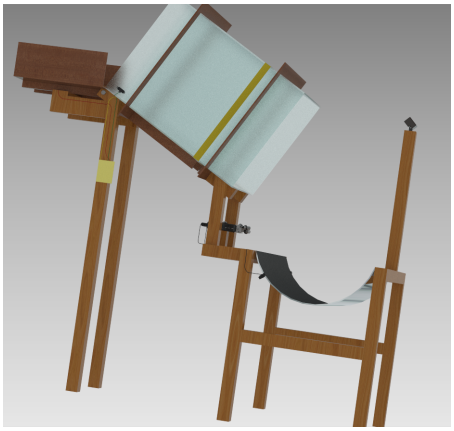
### 3.1.2 Test Rig

*Construction:*

As using a collection truck for the data collection was deemed unfeasible for initial testing, the testing rig was built to imitate the event of emptying a trash bin into a collection truck. This way, the acquired data would be representative while giving the experimenter more time and flexibility when testing and gathering the data.

To mimic the trash bin, four 3 mm thick plexi glass plates were cut with finger joint patterns at the edges and glued together using Super Glue. A transparent material such as plexi was chosen to give the experimenter vision while the bag was sliding through the chute and thereby better control of the experiment. The plates were cut with a width of 50cm and a length of 70cm making the chute a 50x50x70cm cube with openings at the top and bottom. With the added length of 40cm from the load tray (which will be covered later), it would closely resemble a trash bin (58x74x107) and thereby approximate the time taken from trash bin flipping to the bags landing in the truck. Mimicking this time was necessary in order to design a system that could work on a service car, where timings are of high importance.

(a) Picture taken from garbage truck



(b) Test rig comparison

Figure 27: Real life vs test rig

As for the landing tray in the back of the truck, an old metal hot water tank was re-purposed to mimic the landing tray in a truck while also producing a similar sound. The hot water tank was cut in half resulting in an open half-cylinder shape as found in the back of the service car. With a width of 80cm and a radius of 25cm it did not accurately resemble the dimensions of the metal tray, which is close to 2m in length and 0.5m in radius, in the collection truck.

However, it is the sound of impact that is essential to mimic for the data to be representative, and since the simulation was a rather small scale of reality (i.e flipping 1-2 bags and not a full bin) the dimensions were deemed acceptable. A comparison between the landing tray and the tray of a collection truck can be seen in figure 29 at the end of this section.

Attached to the back of the chute is the loading tray, shown in figure 28. Its function is, as mentioned, to initiate the bag sliding when being tilted by the operator. A box of dimensions 46x40x15 with an open top and front (front facing the chute), was constructed by cutting out four 6mm thick MDF plates with finger joint edges and gluing them together with hot glue. Due to the more coarse nature of the surface of MDF compared to plexi, hot glue was sufficient. Attached underneath the box was two elongated beams also cut from 6mm MDF plates with length 40cm, attached with a 15.5cm space between each other. The beams were placed such that 5cm was sticking out in the front of the box. With horizontal holes at the tips, they functioned as attachments for axles and thus giving the loading tray the ability to be flipped. The material for the load tray did not need to be transparent as it was open in the top and front and had low walls making the visibility sufficient. A tiltable loading tray was chosen to mimic the initial speed the bags obtain when trash bins are tilted and emptied in a service car. Approximating the same speed was regarded as necessary as it would influence both the metal detector readings as well as the sound of impact upon the metal tray.

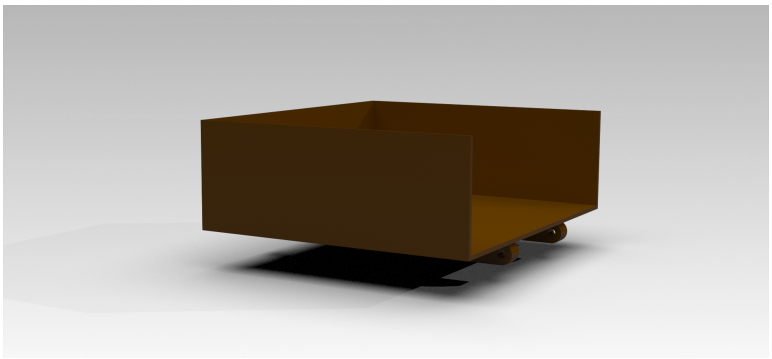


Figure 28: Loading tray

The support frame was built using 2" by 4", and 2" by 2" profiled wood. This material was chosen due to availability and ease of forming. Additionally, the material provided more than enough structural strength for the rig during operation. A framework was built around the metal tray to elevate it from the ground and to hinder the sound from propagating through the floor rather than through the tray. The elevation also provided a more comfortable level to work with for the operator. The chute was further elevated about 30cm from the

edge of the metal tray such that the bags would have a fall height of 30-55cm and thereby creating a sound loud enough for sufficient details to appear in a spectrogram of the impact. The chute was installed at a 25° angle relative to the horizontal plane to approximate the angle of a trash bin being emptied in a collection truck.

*Metal Detector:*

The metal detector is placed in two main locations. First, the search coil is wound around the plexiglass chute to register the bags passing through. The coil is connected to the circuit board which contains the rest of the detector circuit described earlier, and the Arduino Nano responsible for the data recording and transfer to an external computer. All components are indicated in figure 24 shown previously. The detector also needs an external power supply as the Arduino is not powerful enough. The one used for the rig delivers 5.7V and 300mA.

*Weight Registration:*

The weight sensor is located under the loading tray, and is shown in figure 24. The load cell is centered under the middle of the tray, and is connected to a HX711 load cell amplifier mounted on the circuit board. The amplifier is connected to the Arduino which stores and transfers the measurements to the external computer.

*Sound Recording:*

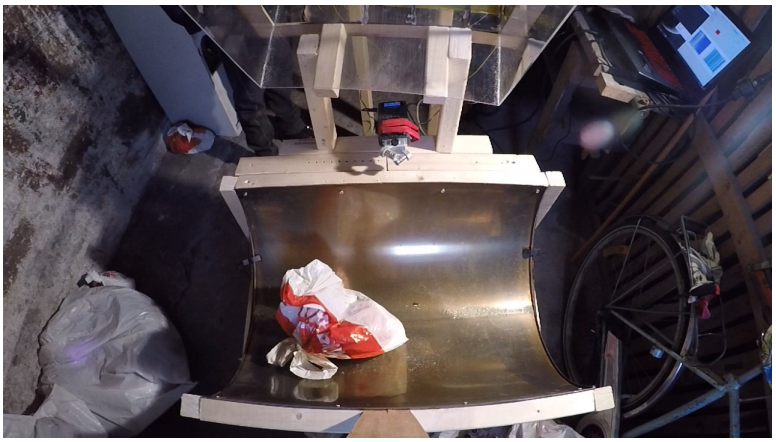
The sound recording system consists of microphones, a recording unit, and an external computer. There are four microphones; two contact microphones and two condenser microphones. The contact microphones are Korg CM300-BK clip-on microphones, which are located on each side of the landing tray, as indicated on figure 24. The condenser microphones are components for the Zoom recorder, and both are contained together in one unit. The microphones are identical, but faces in separate directions. All microphones are connected to a Zoom H6n recorder unit which performs all necessary processing of the sound, like amplification and digitalization. The recorder acts like a sound board for the computer, and its location is indicated in figure 24. By selecting the Zoom unit as the sound input for the computer, the sound can be recorded in any way desired. For this project the sound was recorded by a python script using the Pyaudio package. The main reason for using python was that the recording could easily be syncronized with other parts of the rig, i.e. the proximity sensor. The sound clips were recorded at 48,000 samples per second, with a bitrate of 16.

*Camera:*

The camera is placed overlooking the tray so it captures both the tray and the end of the chute. The placement is meant to capture both the whole tray

and the bag as it falls from the chute, to include the possibility of using pictures from the bags both in free fall and after landing. The camera was set to record video at 720p resolution at 120 frames per second. The high frame rate is required to get a clear picture of a falling bag. Even higher frame rate would be recommended, as long as it does not reduce image resolution below 720p. REN already has cameras installed that capture still pictures, and a comparison between pictures from the test rig and REN is shown in figure 29.

(a) Still picture taken by REN from garbage truck



(b) Video frame captured by test rig

Figure 29: Picture comparison

*Arduino Circuit:*

After completing the circuit and ensuring everything worked, it was permanently soldered together on a single circuit board. The location of the board is indicated in figure 24, and figure 30 shows the wiring diagram. The Arduino

code can be found in Appendix A.1.1.

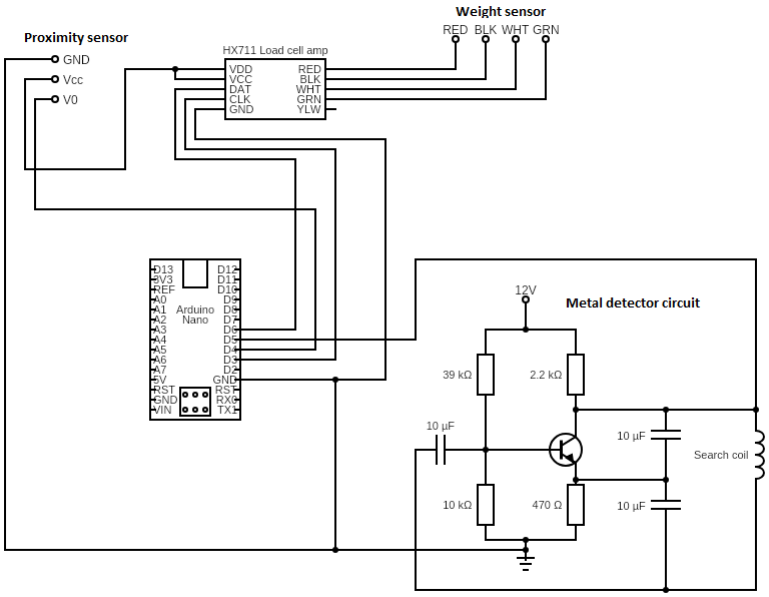



Figure 30: Circuit

### 3.1.3 Trash Generation

To gather the training data it was necessary to produce trash bags to be measured with the test rig. Trash bags were created and categorized in six categories; Pure metal (PM), pure glass (PG), metal and mixed waste (MMX), glass and mixed waste (GMX), glass and metal (GM), and pure mixed waste (PMX). Mixed waste refers to all other materials that are frequently deposited in the trash, not including hazardous materials. The materials used were collected from the recycling department at NTNU. Care was taken to ensure that all bags were different in size, weight and composition within each class to simulate real life conditions. Several types of metal and glass in different shapes and materials were used, as well as a high variety of items for the mixed waste. The categories MMX and GMX were made sure to include both samples of high, medium and low metal/glass content. Due to the extensive time requirements of producing unique bags for the entire training set, each bag was reused several times. In total there were about 25 unique bags present for each class used for the training data set. The argument for reusing bags for training was that the bags would fall with different orientations, resulting in different sound recordings and metal detector readings each time. For end testing, a separate set of measurements was recorded, containing only single measurements of unique

bags, this set containing 10 measurements from each class.

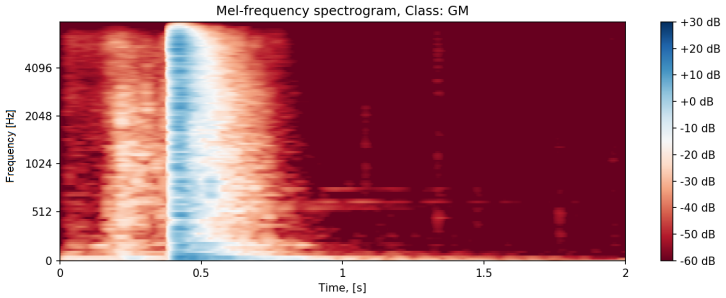
About half the measurements of MMX and GMX for training were performed by including glass or metal in a previously measured bag in the PMX category. The reason for this was to ensure that the program could separate between very similar bags where the only difference was the inclusion of glass or metal. This approach might force the program to focus on the unique signatures created by glass and metal, and focus less on the differences between the bags as a whole.

### 3.1.4 Data Collection

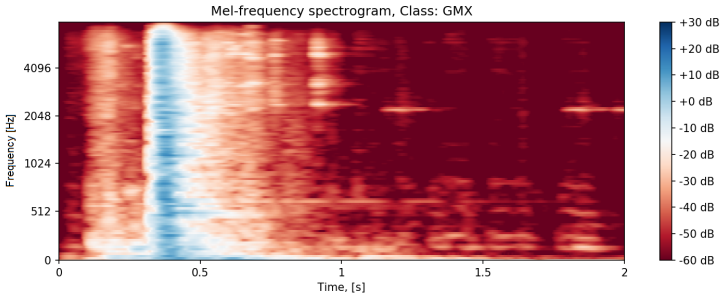
The data gathered from each bag were stored as numpy arrays (with the exception of video) in the following categories:

- Mel spectrograms
- MFCC spectrograms
- Weight data
- Metal detector data
- Video clip

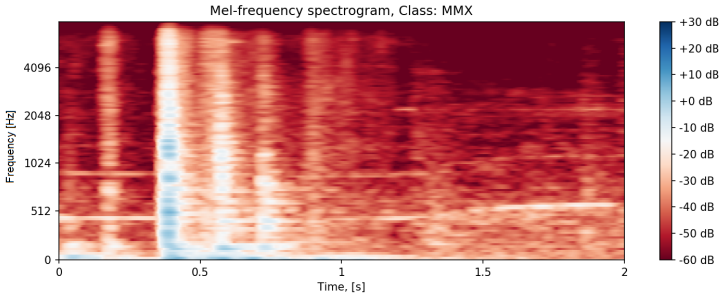
The figures below show comparisons of this data between the classes MMX, GMX, GM and PMX. A different colorscheme than in other sections has been selected for the spectrograms for better comparison, but the data remains the same. Mel spectrograms are shown i figure 31, MFCC spectrograms in figure 32, metal detector measurements in figure 33, and weight measurements in figure **??**.



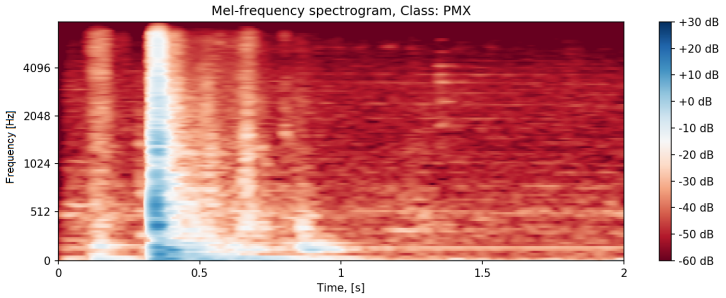

(a) Glass and metal Mel spectrogram




(b) Glass and mix Mel spectrogram

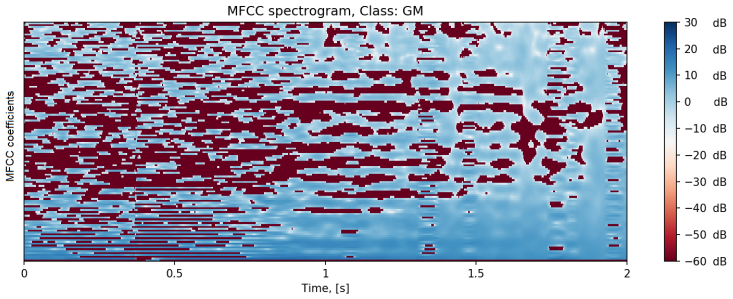



(c) Metal and mix Mel spectrogram

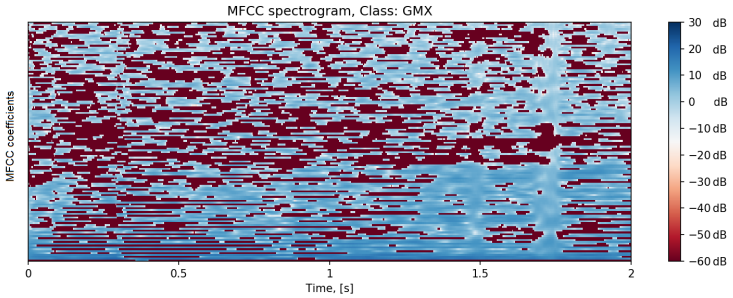



(d) Pure mix Mel spectrogram

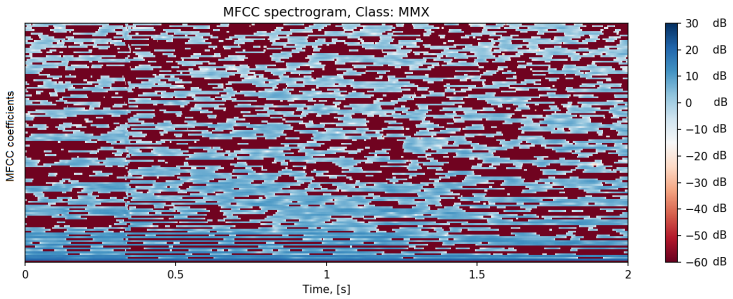
Figure 31: Mel spectrogram comparison



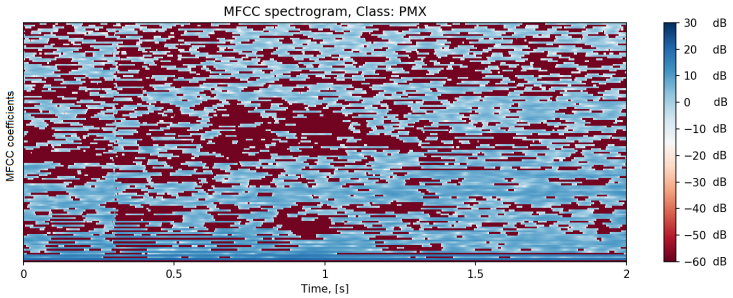

(a) Glass and metal MFCC spectrogram




(b) Glass and mix MFCC spectrogram

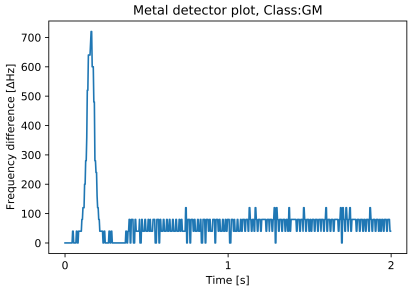



(c) Metal and mix MFCC spectrogram

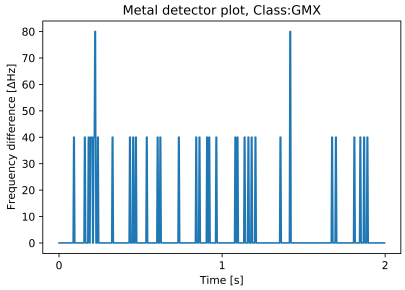



(d) Pure mix MFCC spectrogram

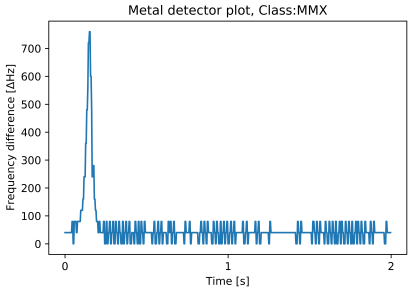
Figure 32: MFCC comparison



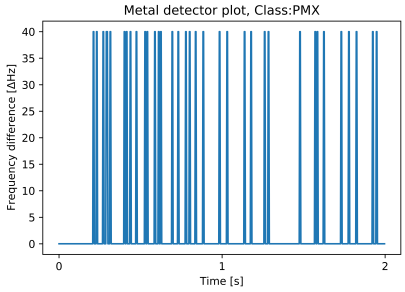

(a) Metal detector output, Glass and metal




(b) Metal detector output, Glass and mix




(c) Metal detector output, Metal and mix




(d) Metal detector output, Pure mix

Figure 33: Metal detector output comparison, (note varying y-axis scale)

Due to the machine learning model requiring that all input data has the same format, the metal data had to be reshaped to (256,256). To achieve this, the time axis was first interpolated to 256 pixels using bilinear interpolation, and then the data was repeated 256 times for the second domain. A visualization of the data after these changes is shown in figure 34.

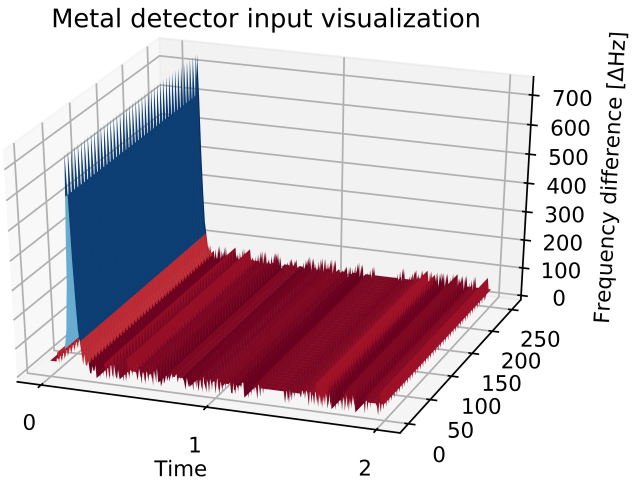



Figure 34: Metal data input visualization

### 3.1.5 Data Exploration

In this section, the data gathered from the waste collection simulation will be explored and analyzed. The purpose is to investigate the distinctions between the classes and explore the feasibility of applying machine learning methods or even simpler deterministic methods for glass and/or metal detection. Classes, PM and PG have been left out of this analysis as they have been deemed unlikely to occur in reality [4]. This is further discussed in Section 3.2.1.1.

#### 3.1.5.1 Sound

Figure 35 presents the average measured sound power over frequency for each of the four categories, at the point of impact, including the standard deviation at certain frequencies. Figure 35 shows that there is a distinctive spread between all four classes. As can be seen, nearly all error bars overlap each other. In fact, every frequency step overlaps between all classes, except between GM-MMX and GM-MX where 80 and 3 steps overlap respectively. The differences between the classes are all significant ($p \leq 0.05$) on all frequencies except in between GMX-MMX. For this comparison, among a total of 128 frequency steps, 28 of them showed insignificant differences. These are mainly found in the mel-frequency ranges of 60-300, and 1300-1550 Mels. This can also be seen from the figure, as the two curves tend to overlap in these regions.

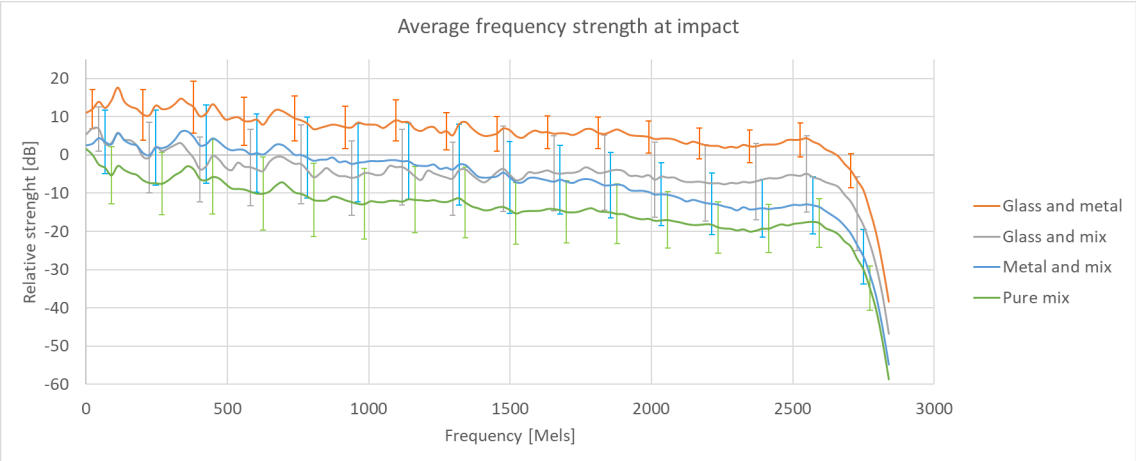



Figure 35: Average strength of sound at different frequencies (4 classes) from the condenser microphone

The above figure suggests the trash bags containing glass and metal simply creates a louder noise, such that it may be possible to distinguish GM and MMX from each other with high accuracy by simply comparing the strength in nearly whichever frequency. Additionally GM and MMX may be distinguished from each other by comparing strengths in frequencies between 1788-2840 Mels where they do not overlap. Regarding the other 4 comparisons, there are considerable overlaps such that it would seem unfeasible to distinguish them from mere sound strength.

Figure 36 shows the average frequency strengths at impact for the left contact microphone. It is clear that general strength level is lower than for the condenser microphone. The average strength is about 3.5-11.2 dB lower across all classes. Additionally the overall standard deviation is substantially larger for the contact microphone, with about 2.9-7.3 dB across all classes. This is also reflected in the amount of classes overlapping each other. Here only one comparison has non-overlapping error bars, which is GM vs. MX, where 28 frequency steps overlap. These are evenly spread out across the 128 steps. All classes are significantly different from each other ($p \leq 0.05$) in most frequencies here as well, with the exception of 13 frequencies in GMX vs. MMX and one frequency in GMX vs. MX.

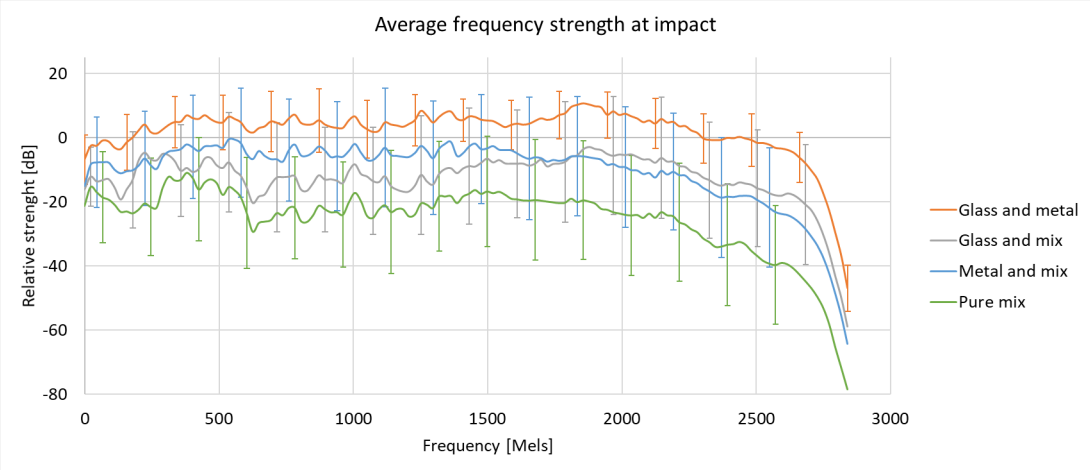



Figure 36: Average strength of sound at different frequencies (4 classes) from the contact microphone

As can be seen in the figure above, the contact microphone shows some of the same trends as the condenser microphone. The GM class is quite distinguishable from MX and the curves of GMX and MMX tend to cross and overlap each other.

#### 3.1.5.2 Metal

Figure 37 shows the maximum metal detector readings for the different classes. GM and MMX have higher maximum metal detection readings than GMX and PMX. This is expected as PMX should not contain any metal and should thereby have around zero maximum readings. GMX should also exhibit low metal detector readings except for the occasional metal lid. PMX does display some rouge readings where maximum values reaches 1000 to 3000. And example of one of these readings is shown in Figure 38. Furthermore, all differences between classes are significant($p \leq 0.05$).

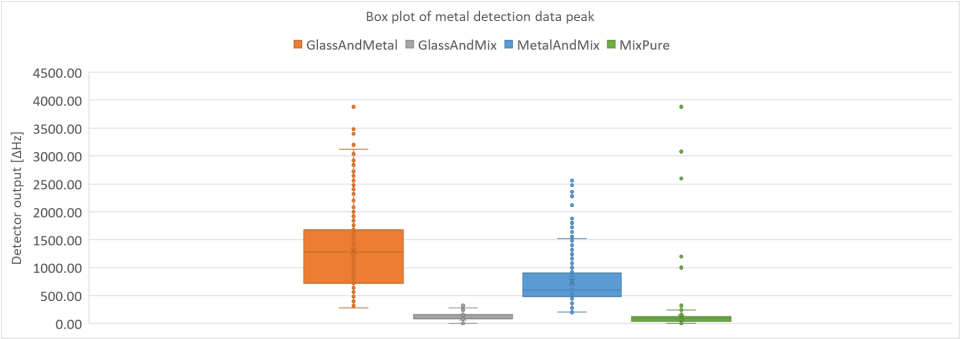



Figure 37: Metal detector: Average reading at peak (4 classes)

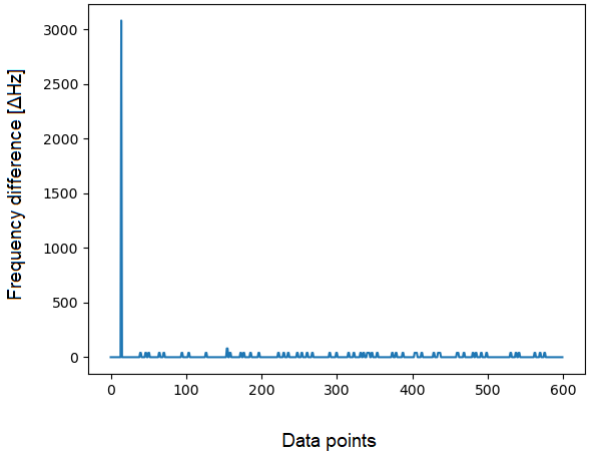



Figure 38: Sporadic metal detector reading on a sample from the PMX class

It should be possible to simply apply a threshold to the metal detection reading at 200-300 to determine the presence of metal. However, some mistakes will be made due to sporadic readings. If a filter were applied, and sudden jumps were removed, a deterministic approach could be used. A simple algorithm for thresholding metal detection data was consequently tested. It was found that a threshold of 250 would result in detecting the presence or absence of metal with 98.4% accuracy. The code for this program can be found in Appendix A. However, this approach is not able to detect the presence of glass.

It may be possible to make further distinctions between the classes by combining the sound and metal detection data. Figure 39 shows a scatter plot of

all measurements with the average strength[dB] across all frequencies along the x-axis, and metal detector reading along the y-axis.

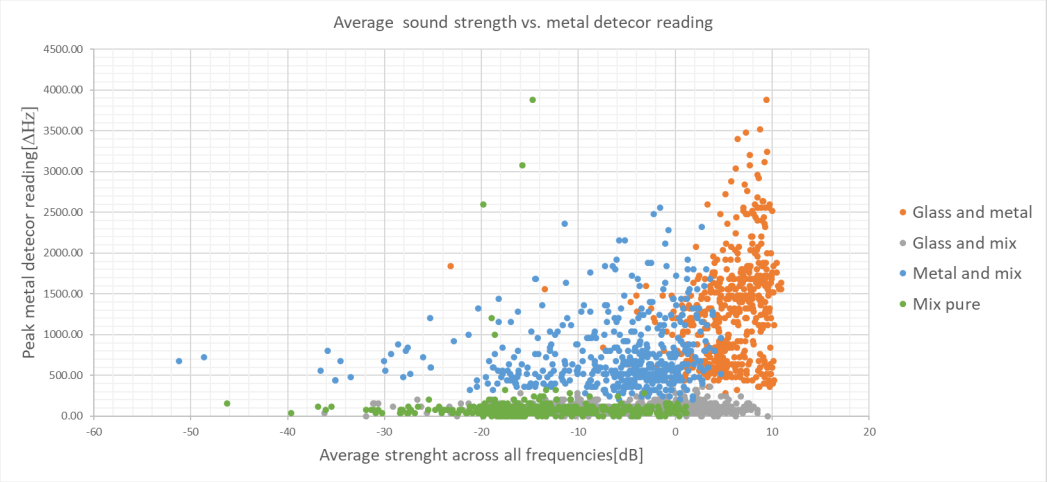



Figure 39: Metal detector peak vs. average sound strength at impact

There does seem to be some distinct areas for the different classes as shown in Figure 40. Specifically GM and MMX seems distinguishable. GMX and PMX overlaps substanially on both axes making them hard to differentiate, however, a machine learning model might be able to capture the differences using a non-linear solution.

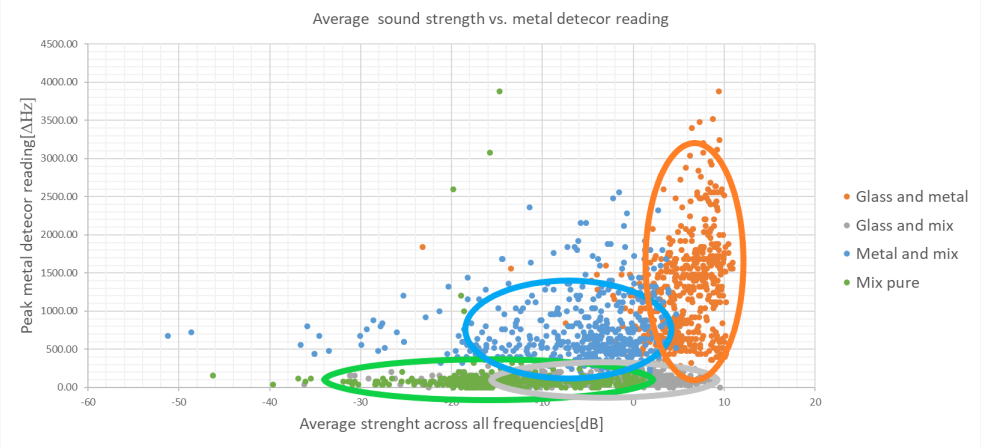



Figure 40: Metal detector peak vs. average sound strength at impact, with indications of grouping

#### 3.1.5.3 Weight

Figure 41 shows the weights of the different classes. All comparisons, except between GMX-MMX and MMX-PMX, are significantly different from each other($p \leq 0.05$). They do still exhibit extensive overlapping. There seems to be little to be said from the weight alone, except that if it is very high, it is likely to be in the GM class. The weight does not support much distinction between GMX and PMX even though the differences are significant ($p = 0.0044$).

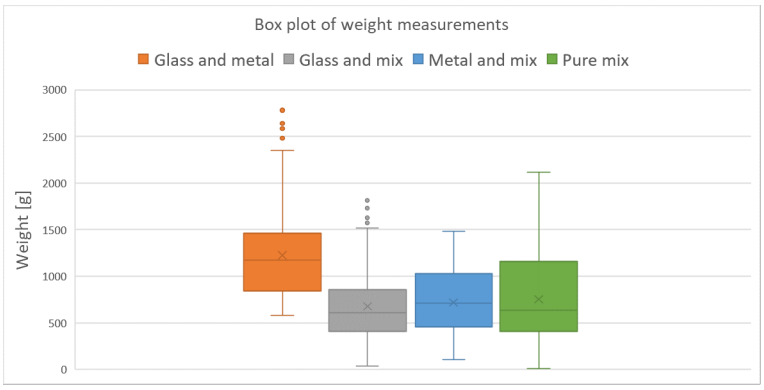



Figure 41: Metal detector: Average reading at peak (4 classes)

The analysis shows that there are differences between the classes, which may indicate that a machine learning model can find the non-linear relations between them. There may lie patterns in the combination of strengths of frequencies, that is, there might be a certain profile to each class. However, it would require a deeper understanding of sound analysis to use any deterministic approach on the data gathered here alone. A machine learning model, may potentially learn these patterns on its own without the need for expertise. The fact that most differences in sound strengths of classes are significant may point to that a machine learning model can find a pattern in the spectrograms. Had not the differences in so many of the frequencies been significant, it is possible that a machine learning model could be blind to the distinction. This is hard to say, however, as a machine learning model is a black box that may find patterns invisible to any human. A machine learning model may also find patterns in the combination of the different sensor readings. Additionally, the analysis may help explain certain results achieved using CNN.

### 3.1.6 GUI

To aid the experimenter during data collection, a simple graphical user interface (GUI) was developed. The software's primary task was to function as a control panel that presents certain sensor recordings in between measurements. When recordings are presented, the experimenter is given the choice of either saving,

in the case of satisfactory sensory readings, or discarding, in the case of faulty sensory readings. The code for the GUI can be found in Appendix A.1.2.

Such a GUI was deemed necessary for the data collection system to ensure valid recordings while keeping up the efficiency. Sensory reading could alternatively be manually validated after each single recording, though it would require the experimenter to pause the main script between every measurement and run a separate script showing the data. This would greatly increase the time consumption of the data collection. Additionally, neglecting to validate sensor readings could result in erroneous recordings within the data set causing the machine learning model later developed to learn on wrong premises. Worst case, a whole collection session could be wasted due to an unseen bug. Also, acquiring the data for machine learning tasks is often the most cumbersome and time-consuming process. A GUI displaying sensor readings would greatly increase the efficiency and control of the data collection.

Python was chosen as the programming language due to the wide availability of libraries as well as flexibility with regards to testing certain functions. Librosa enables easy creation and display of spectrograms, Numpy offers great flexibility when handling arrays and matrices and Tkinter is a relatively easy to use library for developing graphical user interfaces. Also, Python is an interpreted language, such that tests on individual components of the program can be conducted without the whole program functioning properly.

#### 3.1.6.1 Page 1

When launching the GUI, the operator is first met with Page 1 as shown in Figure 42.

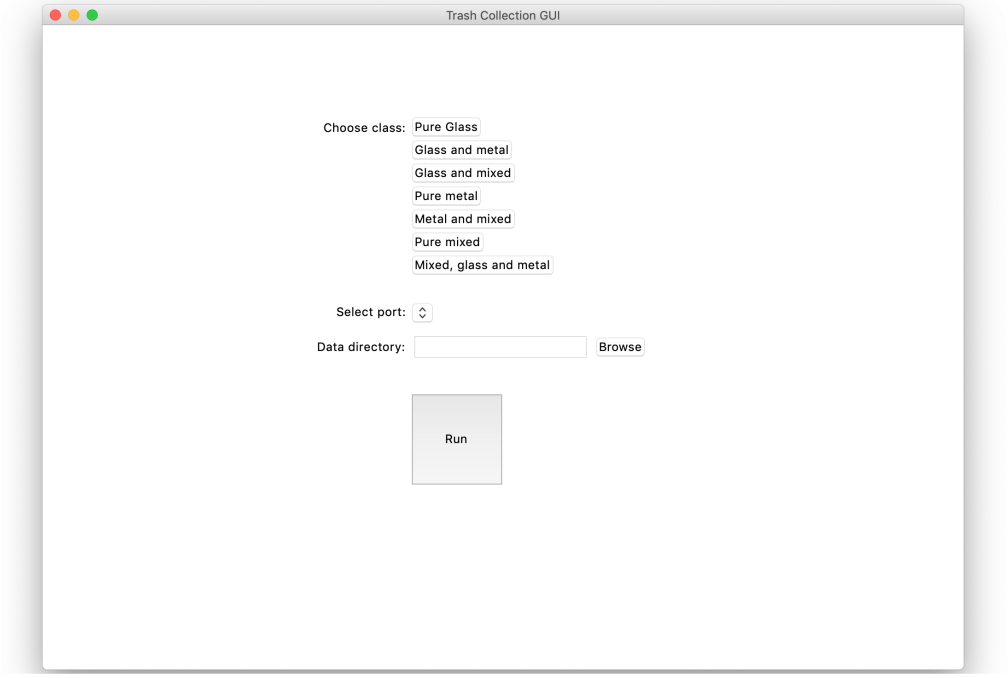



Figure 42: GUI: Page 1

First, the operator must select the USB-port the arduino is connected to. If an arduino is connected, the port will show in the drop down menu labeled "Select port" in Figure 43.

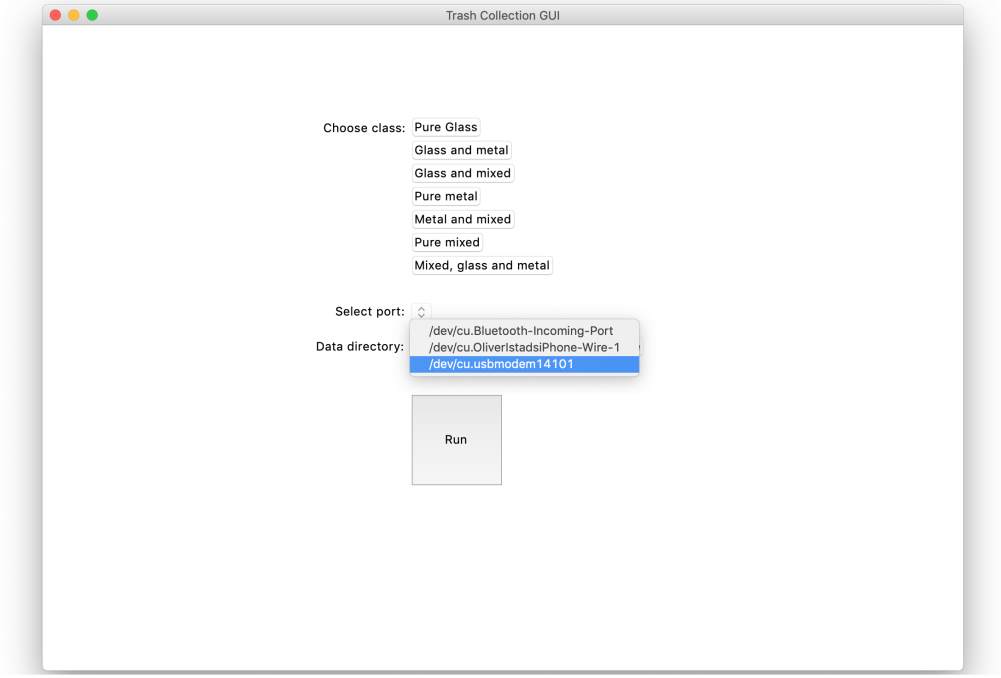



Figure 43: GUI: Page 1, USB Port

By clicking the browse button, a window pops up showing the computer's folder system. The operator now selects where all collected data should be stored.

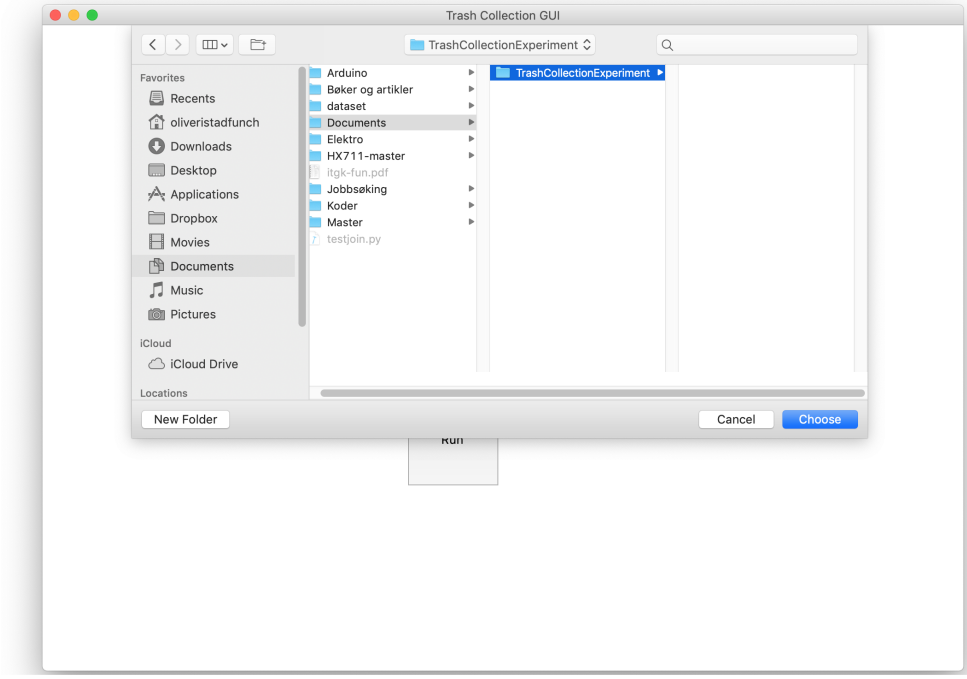



Figure 44: GUI: Page 1, Data directory

It is not until the operator has chosen the USB-port and data directory that he/she is able to click the "Run" button. If for instance a location was not specified and measurements had been initiated, either the program would crash or the recordings would have been lost. By simply disabling the button until the above mentioned options have been set, it is ensured that the program starts with a connection to the Arduino and a location for the data. This is the purpose of having an introduction window, to ensure valid configuration when initiating measurements and thereby avoid program crashes or lost measurements. Additionally, error handling does not have to be programmed simply by not allowing the error to happen.

#### 3.1.6.2 Page 2

The operator has now clicked the "Run" button and is met with Page 2, which is the main window of the application (Figure 45). If not already in place, a folder system is automatically created, containing all the class folders as well as a temp folder for intermediate storage. Additionally, a file named "storageLocation.txt" containing the selected data directory is created at the same location as the python file, such that the directory field is automatically filled on next start-up.

The yellow cue light indicates that a bag should be placed in the loading tray (yellow square in figure 45).

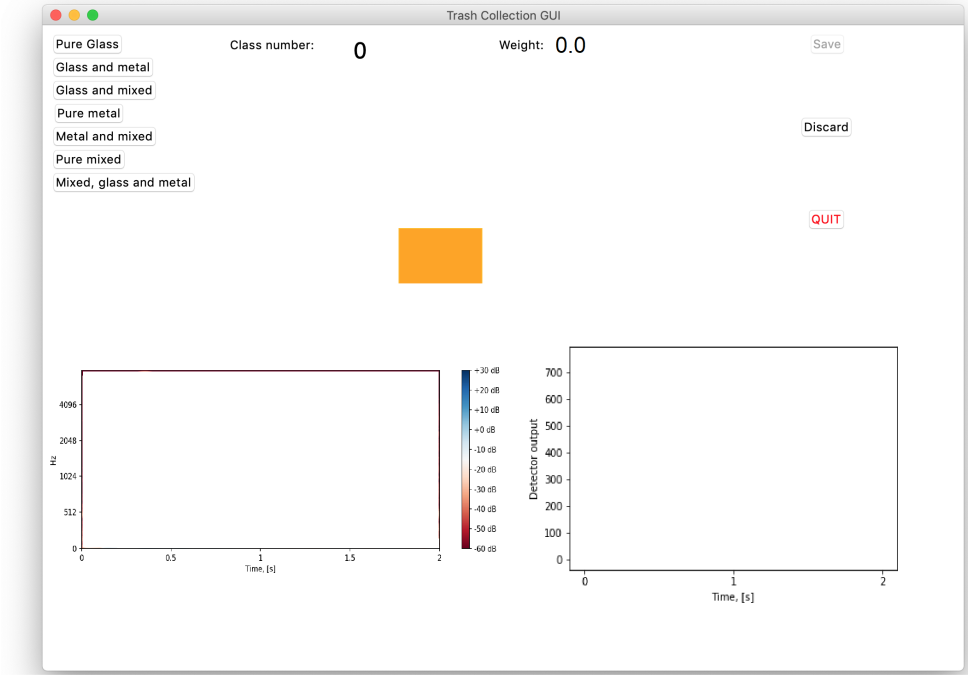



Figure 45: GUI: Page 2

Next, the weight is registered and displayed by the label "Weight". In this case 601.5 grams. Then the cue light turns green, telling the operator to flip the tray. This is also the moment video recording is initiated.

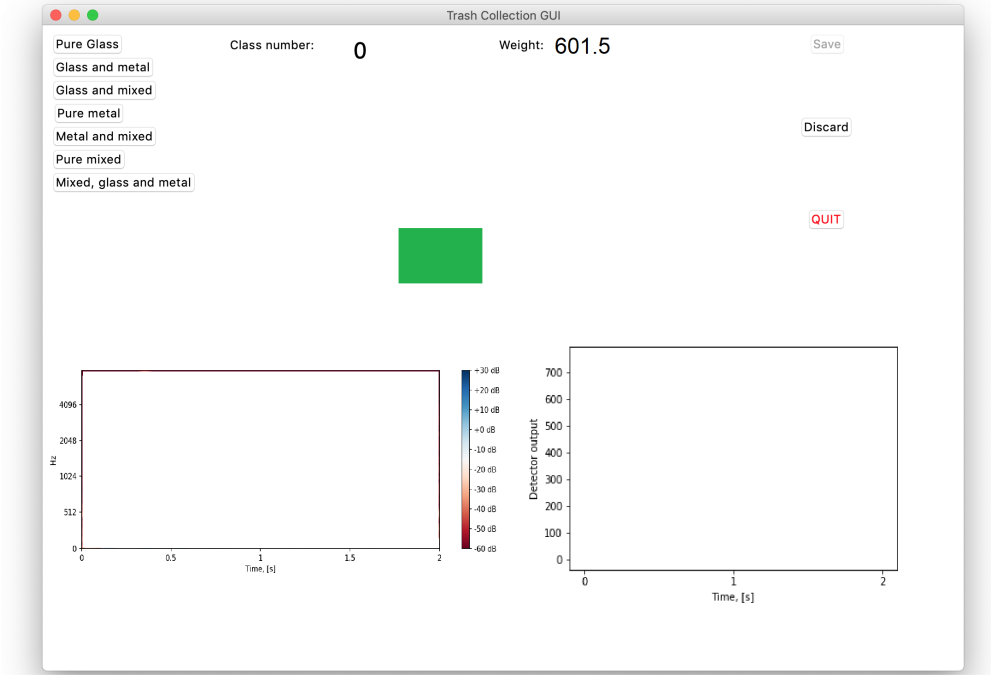



Figure 46: GUI: Page 2, Ready to flip

When the video, sound and metal recordings have been completed after the bag has landed in the metal tray, some of the recorded data is displayed in the GUI (See Figure 47). The cue light is turned red, indicating that the operator should not place any bags in the loading tray yet. The video recording must be downloaded from the GoPro via WiFi connection which may take some time, but this download is queued once the recording is stopped. The metal data is shown on the right, while the right-side condenser microphone data is shown on the left. Now the operator is able to check the sensor readings before continuing. It was assumed that if one of the sides of the condenser microphone gave valid recording, all microphones did. Also, to maintain the flow and uphold the efficiency, it was decided not to wait for the video download to finish to show a frame of it as well. Notice that the "Save" button is disabled until metal and sound recordings are completed. This, again, stops the operator from clicking the button prematurely causing a program crash or lost measurements. The discard button is always enabled, as clicking this resets all variables and restarts the program loop. Before saving, the operator can change the current class by toggling one of the buttons on the left side, though it is not required in enabling the save-button. Chosen class number is displayed next to "Class number" label, which by default is set to zero at startup.

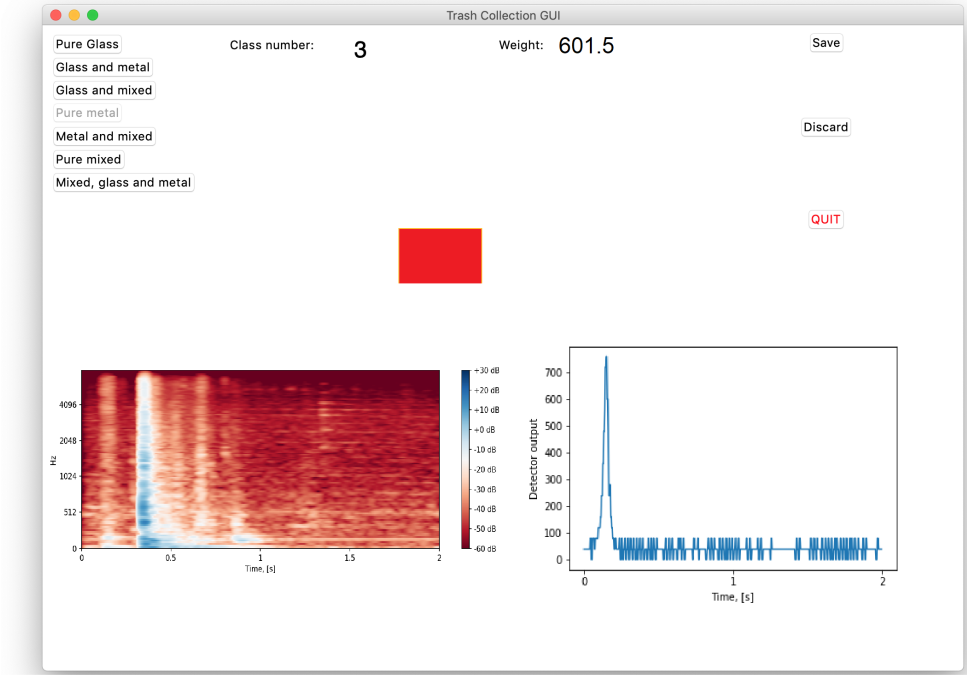



Figure 47: GUI: Page 2, Recordings presented

After the save button is pressed, a folder with the current measurement number as name, is created. The program knows the current measurement number, by reading the file "measurementNumber.txt", which is created when the save button is clicked for the first time. After sensor readings and video file is stored in the newly created folder, the "measurementNumber.txt" file is incremented by one. By storing this number externally it is possible to quit the program, and continue at a later instance. Once all saving is completed, the loop starts all over. This is indicated by the cue light once again turning yellow (see Figure 48).

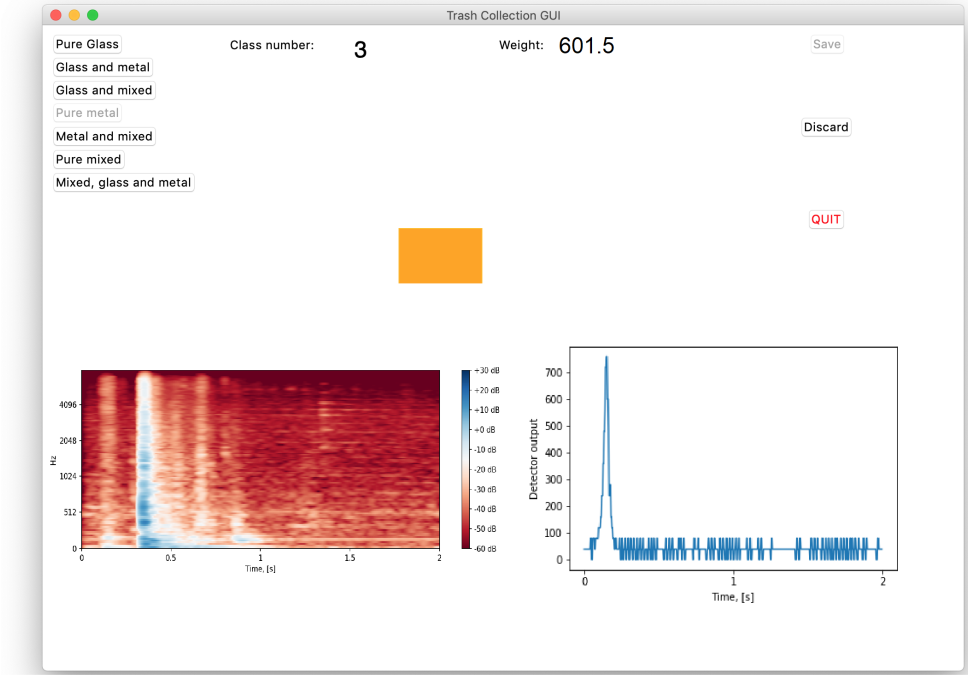



Figure 48: GUI: Page 2, Save button pressed

## 3.2 Model Development

After a sufficient amount of data had been gathered from the trash collection simulation experiment, development and testing of CNN models were initiated. This stage started with getting to know the framework. Keras was chosen due to its high-level API (2.2.3) allowing rapid prototyping. As such, several models could be developed and tested in this Master's Thesis. (2.2.2).

An architecture based on common CNN models ([23], [20], [37]) has been used as a baseline model (Section 3.2.1), and have been the basis of an ablation study conducted to determine which sensors contribute to increased model performance. The results of the ablation study will be presented consecutively as they form the foundation of further model development.

Three models have been developed and tested on the input data resulting from the ablation study. All models use a multi-class approach. These differ in layer count, filter count, kernel sizes and pool sizes. Additionally, all models will be tested using multi-class and multi-label classification labelling schemes (see Section 2.2).

### 3.2.1 Initial Model

As described in Section 2.2.2, a CNN commonly consists of convolutional blocks followed by one or more fully connected layers. The amount of blocks and fully connected layers seem to vary among successful CNN based sound classifiers([22],[43],[17]). A simple configuration containing 5 convolutional blocks and one fully connected layer as shown in Table 5 was chosen for the initial model such that the effect of input data could be the focus.

| *Layer #* | *Layer Type* | *Hyper-parameters* |
|---|---|---|
| 1 | Convolutional | Filters = 32, Kernel size = (3,3), Activation function = ReLu |
| 2 | Max Pooling | Pool size = (2,2) |
| 3 | Dropout | Dropout rate = 0.2 |
| 4 | Convolutional | Filters = 64, Kernel size = (3,3), Activation function = ReLu |
| 5 | Max Pooling | Pool size = (2,2) |
| 6 | Dropout | Dropout rate = 0.2 |
| 7 | Convolutional | Filters = 128, Kernel size = (3,3), Activation function = ReLu |
| 8 | Max Pooling | Pool size = (2,2) |
| 9 | Dropout | Dropout rate = 0.2 |
| 10 | Convolutional | Filters = 256, Kernel size = (3,3), Activation function = ReLu |
| 11 | Average Pooling | Pool size = (2,2) |
| 12 | Dropout | Dropout rate = 0.2 |
| 13 | Convolutional | Filters = 256, Kernel size = (3,3), Activation function = ReLu |
| 14 | Average Pooling | Pool size = (2,2) |
| 15 | Dropout | Dropout rate = 0.2 |
| 16 | Global Average Pooling | |
| 17 | Fully Connected | Units = 4, Activation function = Softmax |

Table 5: Model 1 Architecture

The model has uniformly sized kernels (3x3) through all convolutional layers, each pooling layer halves both the height and the width of each feature map

fed to the layer and all dropout layers blocks 20% of input units. Since it is argued that it is the convolutional layers that causes the high performance [21] when extracting features of matrices, it was decided to include only one fully connected layer to keep it simple and reduce amount of network computations. As such, the model goes straight from the last convolution to classification.

#### 3.2.1.1 Input Data

The model was initially trained using all 6 classes recorded in the trash collection simulation (see Section 3.1). In reality, trash bags having only metal or only glass would very rarely find their way into a mixed trash bin, according to employees at REN [4]. PM and PG was nevertheless included as classes while recording as it was hypothesized that it could improve the models ability to find distinguishing features. Having bags with only metal or only glass could accentuate those features, such that the model could find them in smaller scale on more realistic bags. However, it was quickly realized that the two classes PM and PG did not improve accuracy, specifically if excluded from only the validation data. The included classes are shown in Table 6.

| *Class* | *Samples* |
|---|---|
| Pure mix(PMX) | 502 |
| GlassAndMix(GMX) | 512 |
| MetalAndMix(MMX) | 503 |
| GlassAndMetal(GM) | 502 |

Table 6: Included classes and amount of samples

Furthermore, testing also showed that the weight data did not improve accuracy, but rather constituted a source of confusion for the model. As shown in Figure **??**, weight varies greatly among all classes. Specifically, it can be seen that weight measurements in PMX class deviates both beneath and above GM, PM and MMX, such that a model finds it hard to conclude based on the weight. It was decided to exclude the weight for all further model development to reduce the input size and increase accuracy. The included data sources are shown in Table 7.

| *Sensors* | *Data* |
|---|---|
| Condenser mic R | Mel-spectrogram |
| Condenser mic L | Mel-spectrogram |
| Contact mic R | Mel-spectrogram |
| Contact mic L | Mel-spectrogram |
| Condenser mic R | MFCC |
| Condenser mic L | MFCC |
| Contact mic R | MFCC |
| Contact mic L | MFCC |
| Metal detector | Stacked 1D-array |

Table 7: Included data sources and their data type

Each of the data source listed above was designated a separate channel since they all represent the same event, resulting in 9 channels. All spectrograms where resized to 256x256. The metal data was interpolated to 256 data points and stacked 256 times on top of each other along the second axis, as the CNN requires a constant input shape for all channels 2.2.2. This approach was chosen as during training it was observed that zero padding the metal detection data would result in the model effectively ignoring this input. As such, the initial shape of the input tensor was (2019,256,256,9).

#### 3.2.1.2 Training and Testing

Using the input data shown in Table 7, Model 1 was trained on a 80/20 training/validation split. Due to random initialization of weights and a random selection of dropped units (Section 2.2.2) the result may vary every time a model is trained and validated on the exact same data samples. Because of this, the model was fitted 20 times to find the standard deviation and average validation accuracy so the different input variations would be comparable. Reaching a maximum accuracy of 96.54.% and an average of 94.94%, it seemed too good to be true. It was hypothesized that it may have been overfit, even though the validation accuracy also being high should prove otherwise. The reason it still could be overfit was that individual measurements of the same class were very similar to each other, likely because the same exact bag was used many times. This is a possible issue that will be discussed further in detail in Section 4.
// By collecting data from an additional 10 new trash bags of every class, the

possible problem of overfit could be addressed. Every single bag of these new measurements were unique, and no bags were reused. This ensured that a model could not by chance either be good or bad at classifying one specific bag and thereby reach a high or low score. The test would show its ability to classify 10 new bags within each class. The newly gathered data set was called the test set, and was further used in conjunction with validation accuracy to benchmark models developed.

By loading the model weights for the best trained model (96.54% accuracy) and performing inference on the newly gathered data set, an accuracy of 80% was reached, which was significantly lower than on the validation set. The reason for this might be that the new data set was captured in different circumstances as the rig was moved to a new location at one point. However, about half of the training set was collected in the new location, such that the lower score might simply be due to the new data set containing trash bags that have not been used to generate data points for training. Either way, it may point to that the model is specialized on the validation set because it resembles the training set more than the newly gathered test set resembles the training set. Which in turn may mean that the data set gathered is not varied enough.

### 3.2.2 Testing More Models

Additional models where created to enable comparison of different parameters. Two more models where created.

#### 3.2.2.1 Model 2

The second model developed has an additional convolutional block and three more fully connected layers. As such, the effect of depth of a CNN could be assessed as well as the contribution of the fully connected layers. Its architecture is shown in Table 8.

| *Layer #* | *Layer Type* | *Hyper-parameters* |
|---|---|---|
| 1 | Convolutional | Filters = 16, Kernel size = (9,9), Activation function = ReLu |
| 2 | Max Pooling | Pool size = (2,2) |
| 3 | Dropout | Dropout rate = 0.4 |
| 4 | Convolutional | Filters = 32, Kernel size = (7,7), Activation function = ReLu |
| 5 | Max Pooling | Pool size = (2,2) |
| 6 | Dropout | Dropout rate = 0.2 |
| 7 | Convolutional | Filters = 64, Kernel size = (5,5), Activation function = ReLu |
| 8 | Max Pooling | Pool size = (2,2) |
| 9 | Dropout | Dropout rate = 0.2 |
| 10 | Convolutional | Filters = 128, Kernel size = (3,3), Activation function = ReLu |
| 11 | Max Pooling | Pool size = (2,2) |
| 12 | Dropout | Dropout rate = 0.2 |
| 13 | Convolutional | Filters = 256, Kernel size = (1,1), Activation function = ReLu |
| 14 | Max Pooling | Pool size = (2,2) |
| 15 | Dropout | Dropout rate = 0.2 |
| 16 | Convolutional | Filters = 512, Kernel size = (1,1), Activation function = ReLu |
| 17 | Max Pooling | Pool size = (2,2) |
| 18 | Dropout | Dropout rate = 0.2 |
| 19 | Global Average Pooling | |
| 20 | Fully Connected | Units = 128, Activation function = ReLu |
| 21 | Fully Connected | Units = 64, Activation function = ReLu |
| 22 | Fully Connected | Units = 32, Activation function = ReLu |
| 23 | Fully Connected | Units = 16, Activation function = ReLu |
| 24 | Fully Connected | Units = 4, Activation function = Softmax |

Table 8: Model 2 Architecture

The filter count starts at 16 and doubles on every new convolutional layer, ending at 512. The kernel sizes starts with 9x9 and descends through convolutional layers. Having bigger kernels in the first layers and smaller kernels in the later layers is said make the network able to extract both overarching patterns as well as smaller detailed features([22]). A bigger kernel will include more of a matrix element's surrounding neighbours in kernel convolutions (See section 2.2.2), such that more information is included into the resulting feature map. Additionally, the dropout rate is set to 0.4 in the first layer.

Model 2 was trained on the same data set as the previously trained Model 1. Reaching a maximum and average validation accuracy of 93.54% and 89.62% respectively, it scored lower on the validation set than Model 1. On the test set, the model that reached 93.54% validation accuracy, achieved 77.5% test accuracy. It was hypothesized that the model was simply too complex to learn from the amount of data gathered and that it would require more to reach a performance similar to model 1.

#### 3.2.2.2 Model 3

A third and final model was developed and tested. This time with less layers than model 2 as it seemed to be too deep. It's architecture is shown in Table 9.

| *Layer #* | *Layer Type* | *Hyper-parameters* |
|---|---|---|
| 1 | Convolutional | Filters = 16, Kernel size = (4,8), Activation function = ReLu |
| 2 | Max Pooling | Pool size = (2,2) |
| 3 | Dropout | Dropout rate = 0.2 |
| 4 | Convolutional | Filters = 32, Kernel size = (5,5), Activation function = ReLu |
| 5 | Max Pooling | Pool size = (2,2) |
| 6 | Dropout | Dropout rate = 0.2 |
| 7 | Convolutional | Filters = 64, Kernel size = (3,3), Activation function = ReLu |
| 8 | Max Pooling | Pool size = (2,2) |
| 9 | Dropout | Dropout rate = 0.2 |
| 10 | Convolutional | Filters = 128, Kernel size = (3,3), Activation function = ReLu |
| 11 | Max Pooling | Pool size = (2,2) |
| 12 | Dropout | Dropout rate = 0.2 |
| 13 | Convolutional | Filters = 256, Kernel size = (1,1), Activation function = ReLu |
| 14 | Max Pooling | Pool size = (2,2) |
| 15 | Dropout | Dropout rate = 0.2 |
| 16 | Global Average Pooling | |
| 17 | Fully Connected | Units = 128, Activation function = ReLu |
| 18 | Fully Connected | Units = 64, Activation function = ReLu |
| 19 | Fully Connected | Units = 4, Activation function = Softmax |

Table 9: Model 3 Architecture

As can be seen, a slight different approach with regards to kernel sizes was tested. Here the first kernel is wider than it is tall, as such, including more information along the x-axis (time domain) while being more specific along the y-axis (the frequency band). It was hypothesized that this would let the kernel look at two frequency bands at a time and thereby focusing on finding individual patterns in the time domain for each frequency. Additionally, all data is padded (2.2.2) on each convolutional layer. If not padded, the outer rows are

excluded each complete convolution resulting in whole frequency bands being lost. In image recognition this might not be important as the object usually resides in many more rows. However, if there are distinguishing features in the outer frequencies, these should be included. Figure **??** shows the average strength in each frequency for all classes. The lowest frequency does seem to have lowest differences between the classes, while differences are reducing with high frequencies as well. Even though the differences are largest in the middle, there does seem to be a significant difference in the top frequency.

Being trained on the same data set as model 1 and 2, model 3 reached a maximum and average validation accuracy of 96.02% and 94.95%. The model that achieved the best validation score, achieved 77.5% test accuracy. As of now, Model 1 shows most promise and will be used further in an ablation study to determine which sensors contribute to increased detection.

### 3.2.3 Ablation Study

The following section will compare the results of training a model while varying the input data to try to understand which of the inputs contribute the most to successful identification.

As previously described, the input data consists of mainly sound and metal detector readings. The sound recordings come from separate microphones of two different types, and the sound is then fed to the deep learning algorithm after preprocessing. This preprocessing yields two separate spectrograms, as described in section 2.3. Some of these inputs may contribute more to identification than others, and some may even cause the classification to achieve worse results due to added complexity without adding relevant information.

The following tests were performed using four separate classes; GM, GMX, MMX and PMX. The model architecture used was the same for all runs of training. As some combinations could require more or less epochs to train successfully than others, the tests were run with 50 epochs and 70 epochs for each combination. At first another set of runs at 100 epochs was included, but this seemed to result in overtrained models so instead the fewer epochs were run more times. The standard deviation between runs is high enough to require many training runs for comparable results, so most of the tests were done using 20 runs for each epoch number, totalling in 40 runs for each combination. After performing the tests, the results were recorded and analysed using a two-tailed t-test to determine if a difference in total test score was statistically significant, meaning that any observed accuracy differences were not due to random variations.

#### 3.2.3.1 Mel-spectrogram vs MFCC

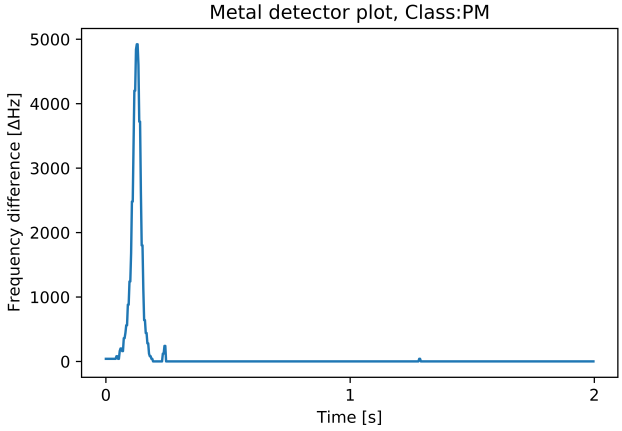
Firstly, tests were performed to see if there was a difference between using both the mel-spectrogram and the MFCC spectrogram, and using only one at a time. Results showed that using both spectrograms yielded a better test score compared to a single one, with significant P-values in both cases. Table 10 shows the average scores for each case and the calculated P-value compared to using both spectrograms.

| *Input type* | *Average test score* | *Number of tests* | *P(T<=t)* |
|---|---|---|---|
| **Mel + MFCC** | **79.3%** | 60 | Reference |
| Only Mel | 75.3% | 60 | 0.006 |
| Only MFCC | 73.1% | 60 | 0.0001 |

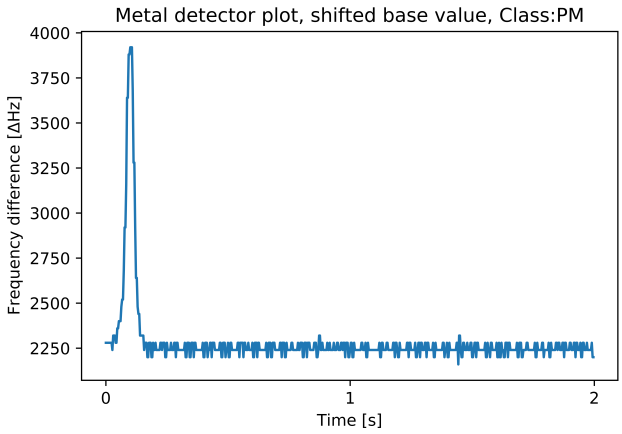
Table 10: Test of spectrogram test score influence

#### 3.2.3.2 Adjusting Erratic Metal Detector Data

For unknown reasons, the metal detector would in a few rare cases have a base value other than zero. This wasn't always discovered during data collection so some of the measurements have readings like shown in figure 49 below. A normal reading is also shown for comparison. The deviation, as seen in the second plot, is that y-axis values start at 3500 instead of the normal 0 value. The difference between minimum and maximum values in the measurements were still in the same range as the regular readings, the readings were only shifted by a constant value.



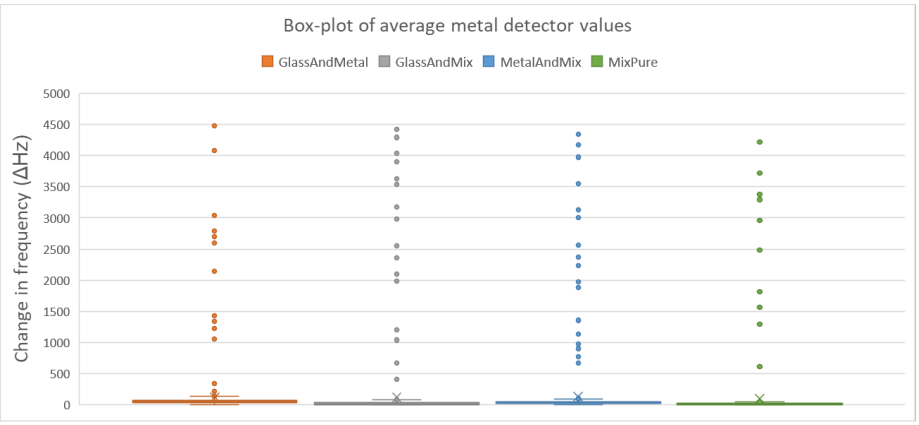

(a) Normal metal detector plot



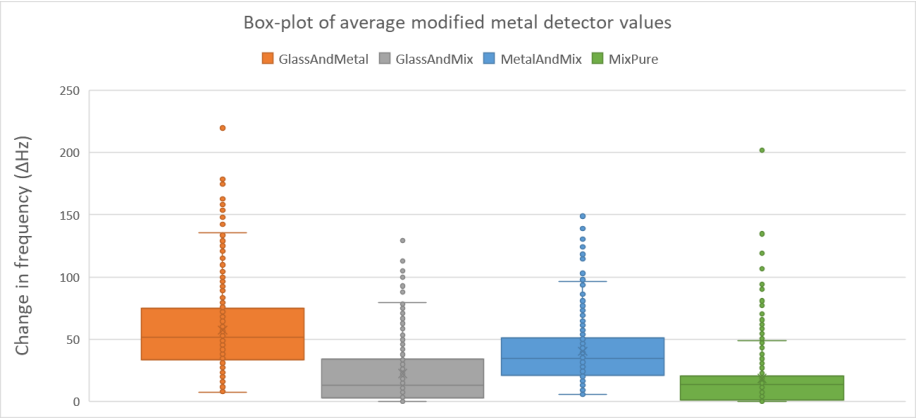

(b) High base value metal detector plot

Figure 49: Metal detector plots

The differences are very clear if the data is shown in a box-plot of the average values from each measurement. These are shown in figure 50. The plot shows that there are some average values that are far higher than total average, and are off by many standard deviations. The second plot shows the values after an adjustment has been made. There are still some high values, but these are now much closer to the global average value.




(a) Normal metal detector box-plot




(b) Adjusted metal detector box-plot

Figure 50: Metal detector box-plots

The adjustment was performed as follows: A python script was used to import the data and subtract the lowest value in the vector from all values in the vector, so as to move the plot closer to the zero value without causing negative values. Afterwards, a model was trained 20 times with 50 epochs and 20 times with 70 epochs, and the test scores recorded for comparison with previously trained models, all with identical parameters and same number of epochs and runs. The results are shown in table 11.

| *Input type* | *Average test score* | *Number of tests* | *P(T<=t)* |
|---|---|---|---|
| **Unaltered metal readings** | **80%** | 40 | Reference |
| Altered metal readings | 78% | 40 | 0.193 |

Table 11: Metal data input comparison

The two-tailed t-test shows that there is no significant difference between the results as it is higher than 0.05. It was therefore concluded that the adjustment does not affect the results. A possible explanation for this might be that the model doesn't emphasize the values of the data, but instead focuses on the change between values. This change is not affected by the shifted base value, as the amplitude of the change is independent from it.

#### 3.2.3.3 Difference Between Microphone Types

As described earlier the measurements were performed using both condenser microphones and contact microphones. It was unclear which of these were most suited for our use, so tests were performed to assess the performance of the model. Several runs of training were performed while excluding one type at a time, and the results were then compared to earlier runs where both microphones types were present. The results are shown in table 12.

| *Input type* | *Average test score* | *Number of tests* | *P(T<=t)* |
|---|---|---|---|
| Both microphones | 80% | 40 | Reference |
| **Only condenser microphone** | **83%** | 40 | 0.032 |
| Only contact microphone | 72% | 40 | 0.00004 |

Table 12: Microphone type comparison

As P-values are below 0.05, the conclusion was that the condenser microphone performs better than the contact microphones. Therefore the contact microphones should not be used as input data. It should be noted that the contact microphones used for this project were not very expensive, so it is possible that another model would perform better.

#### 3.2.3.4 Spectrogram Alterations

As can be seen from figure 51, the time frame where the bag lands in the tray is a very short part of the spectrogram. The bag hits at approximately x = 0.4 in the spectrogram.

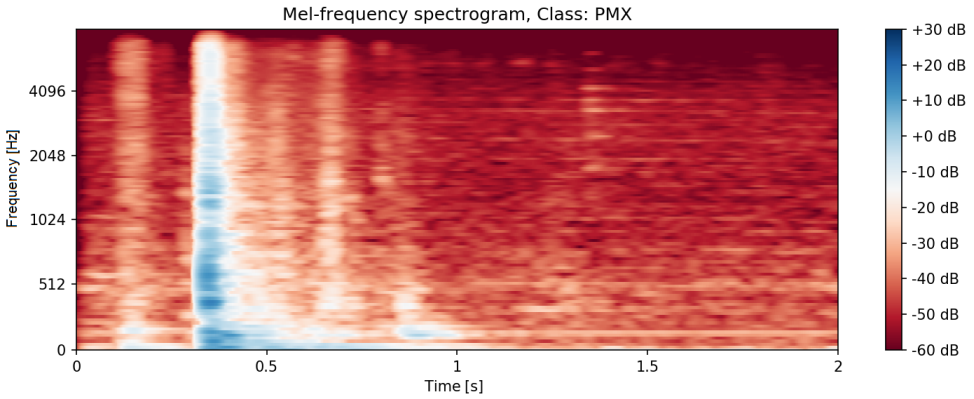



Figure 51: Unaltered Mel-spectrogram

This means that the spectrogram contains a lot of unnecessary information that does not relate to the measured bag. It was therefore attempted to remove most of the unwanted parts so the remaining information would be more relevant for training the model. A script was writted to identify the peak, and a portion of 64 columns (or pixels, representing a timestep in the recording) to the left and 192 to the right from this peak was selected, resulting in an x-axis length of 256 columns. The original length was 1379 columns. The reason for having more columns on the right side of the impact can be seen in the spectrogram, as it appears to be more valuable information to the right than to the left. The same spectrogram with this alteration is shown in figure 52.

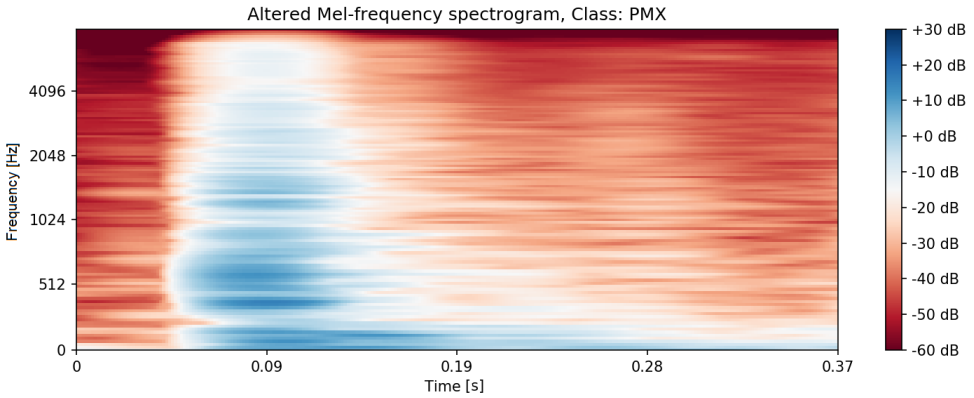



Figure 52: Altered Mel-spectrogram

Tests were performed in the same fashion as earlier to see if the alterations had any effect. Earlier modifications were also included, so both reference models

and new models were trained using only condenser microphones. The results are shown in table 13.

| *Input type* | *Average test score* | *Number of tests* | *P(T<=t)* |
|---|---|---|---|
| Unaltered spectrograms | 83% | 40 | Reference |
| **Altered spectrograms** | **88%** | 40 | 0.001 |

Table 13: Altered spectrogram comparison

The tests show a significant increase in test accuracy, so this alteration should be included in the pre-processing of the collected data. An added advantage is that the spectrogram matrix used for training the model will be much smaller, reducing the computing power needed to train the model.

#### 3.2.3.5 Weight Measurements

It was discovered early that excluding the weight measurements from the model seemed to yield better results. Due to incomplete data, a new test was run to compare results with and without the weight included to ensure that this decision was correct. This test was conducted with the earlier discussed changes in place, like altered spectrograms and no contact microphone. The results are shown in table 14.

| *Input type* | *Average test score* | *Number of tests* | *P(T<=t)* |
|---|---|---|---|
| Weight included | 82% | 40 | Reference |
| **No weight data** | **88%** | 40 | 0.0002 |

Table 14: Weight data effect

The results are clearly worse when the weight data is included. There are a couple different possible reasons for this. The most likely is that the weight does not contribute to any pattern, as the addition of glass and/or metal to a bag does not mean that the bag must be lighter or heavier than a bag with other materials. Another possibility is that the method for including the data in the program was poor. As the weight only produces a single data point, this value had to be added to all indices of a matrix the same shape as the spectrogram and metal measurements. This is because the program will not accept inputs of different shapes. A possible use for the weight was proposed, but not tested fully due to time restrictions. If the surface area or volume of the bag can be measured using video footage, the density of the bag can be calculated. The possibility and use for this will be discussed later in the thesis. Unless such a system is implemented, it is recommended to leave out the weight data from the machine learning model.

## 3.3 Preliminary Evaluation of Models

### 3.3.1 CNN, Multiclass Classification Results

The best results obtained from the CNN single labeling model are discussed in this section. Due to variations between training runs, the model was trained several times using the same train/test split with both 50 and 70 epochs to assess the accuracy and stability of the model. The best results were obtained when selecting a section of 256 x 128 pixels centered around the peak of the spectrograms, as described in the ablation study section. Also, the model uses only the condenser microphone. The results are shown in table 15 and table 16, where each column is a separate training run. The values show the number of successful classifications for that class. The classifications were performed on the separate test set consisting of unique bags not used for training, as described in the development section, consisting of 10 sample bags for each class.

| | *50 epochs* | | | | | | | | | | | | | | | | | | | | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Training run* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | |
| *GM* | 10 | 10 | 10 | 10 | 10 | 10 | 9 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 9 | 9 | 10 | 9 | 10 | 98 % |
| *GMX* | 10 | 10 | 10 | 10 | 10 | 9 | 10 | 10 | 9 | 10 | 9 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 99 % |
| *MMX* | 7 | 9 | 9 | 9 | 8 | 9 | 9 | 6 | 8 | 7 | 5 | 6 | 9 | 9 | 6 | 8 | 9 | 5 | 7 | 8 | 77 % |
| *PMX* | 8 | 6 | 8 | 9 | 5 | 8 | 8 | 8 | 8 | 7 | 8 | 7 | 7 | 8 | 7 | 9 | 6 | 7 | 9 | 7 | 75 % |
| *Accuracy* | 88 % | 88 % | 93 % | 95 % | 83 % | 90 % | 90 % | 85 % | 88 % | 85 % | 80 % | 83 % | 90 % | 93 % | 83 % | 90 % | 85 % | 80 % | 88 % | 88 % | |
| *Average* | 87,00 % | | | | | | | | | | | | | | | | | | | | |
| *Std. deviation* | 4,08 % | | | | | | | | | | | | | | | | | | | | |

Table 15: Results with 50 epochs

| | *70 epochs* | | | | | | | | | | | | | | | | | | | | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Training run* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | |
| *GM* | 10 | 10 | 9 | 10 | 10 | 10 | 10 | 8 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 9 | 10 | 10 | 98 % |
| *GMX* | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 9 | 9 | 10 | 10 | 9 | 99 % |
| *MMX* | 5 | 6 | 10 | 8 | 9 | 9 | 9 | 9 | 9 | 7 | 7 | 8 | 9 | 9 | 5 | 9 | 8 | 9 | 8 | 7 | 80 % |
| *PMX* | 7 | 7 | 8 | 9 | 7 | 9 | 9 | 9 | 8 | 7 | 9 | 7 | 8 | 7 | 7 | 9 | 8 | 8 | 7 | 7 | 79 % |
| *Accuracy* | 80 % | 83 % | 93 % | 93 % | 90 % | 95 % | 95 % | 90 % | 93 % | 85 % | 90 % | 88 % | 93 % | 90 % | 80 % | 93 % | 88 % | 90 % | 88 % | 83 % | |
| *Average* | 88,75 % | | | | | | | | | | | | | | | | | | | | |
| *Std. deviation* | 4,51 % | | | | | | | | | | | | | | | | | | | | |

Table 16: Results with 70 epochs

It is clear that the model seems biased towards classifying a bag as containing

glass, as the GM and GMX categories hit accuracies of 98-99%, while the other two categories have a lower accuracy score. The misidentifications in the PMX category poses a problem, as the amount of bags expected to be in this category can be assumed to be much higher than in the other categories. For the 70 epoch trained models, one out of five bags are falsely classified as containing either metal or glass, which might lead to many false positives. According to the trash analysis performed by REN in 2019 [33], 4.6% of the trash in a bin is glass and metal. This means there is a 20:1 ratio of bags without glass or metal for every bag containing either of these materials. This means that there will be 4.55 false positives for every correct identification of glass/metal, calculated using equation 5.

$$\frac{20}{1} * \frac{1 - (True\ PMX\ classification\ ratio)}{True\ classification\ ratio(MM \cup MMX \cup GMX)} \tag{5}$$

This poses a serious problem as the amount of false positives might cause false patterns to develop. On the positive side, the false positives will be spread out across all areas over time, so there is still a fair chance that certain areas will stand out if the positives occur frequently at the same locations. Taking into account what the false positives are identified as, the ratio improves. Table 17 shows a confusion matrix over the false classification distribution for all training runs using 70 epochs.

| True label | Classified as | | | |
|---|---|---|---|---|
| | *Glass and metal* | *Glass and mix* | *Metal and mix* | *Pure mix* |
| *Glass and metal* | N/A | 1.0% | 1.0% | 0.0% |
| *Glass and mix* | 1.0% | N/A | 0.0% | 0.5% |
| *Metal and mix* | 19.5% | 0.5% | N/A | 0.0% |
| *Pure mix* | 0.0% | 21.5% | 0.0% | N/A |

Table 17: False classification distribution

Two combinations stand out in this distribution. Almost all false classifications are either MMX being identified as GM, or PMX begin identified as GMX. The first presents a lesser problem, as both are categories that indicate bad sorting so they can still be counted as a true positive for this calculation. The largest issue is PMX identified as GMX, as PMX is the category that will be in majority in the trash collection. It is therefore these false positives that need to be addressed. When calculating the ratio of false vs true positives while only taking into account the PMX misclassification, the ratio looks marginally better. If again assuming a 20:1 ratio between correctly sorted bags against bags containing glass and/or metal, the false/true positive ratio becomes 4.22. Note that this is based on an average of 20 trained models, where some individual

models have a higher accuracy than the average. This might mean that one individual model might achieve a much better false vs true positive ratio.

To attempt to remedy this issue, the same models were used for reclassification, but this time using a minimum threshold for the confidence of classifications. If the confidence of a classification was below 0.7, the bag was placed into a separate "unsorted" class. The false classifications are shown in table 18.

| True label | Classified as | | | | |
|---|---|---|---|---|---|
| | *Glass and metal* | *Glass and mix* | *Metal and mix* | *Pure mix* | *Unsorted* |
| *Glass and metal* | N/A | 1.0% | 0.5% | 0.0% | 0.0% |
| *Glass and mix* | 0.5% | N/A | 0.0% | 0.0% | 1.5% |
| *Metal and mix* | 16.0% | 0.5% | N/A | 0.0% | 11.0% |
| *Pure mix* | 0.0% | 18.0% | 0.0% | N/A | 5.0% |

Table 18: False classification distribution with 0.7 threshold

For this distribution, still only taking into account the PMX misclassifications, the false/true positive ratio becomes 4.11. The ratio is slightly better, but the addition of the unsorted category means that the model is missing out on some true positives, especially in the MMX category. However, the reduction of false positives is a desirable result.

So far the results discussed have been based on the average values of several runs. This is helpful for development as it accentuates patterns, though in the end only a single model will be used. The test set used is somewhat small, and will not necessarily show the most successful model, therefore it may be best to focus on the validation accuracies of the models. The model with the highest validation accuracy may not yield the highest result on the test set, but might prove the most accurate when applied to a real setting. The best model in terms of validation accuracy had an accuracy of 99% and had a test accuracy of 87.5%. For comparison, the models with the highest test accuracy (95%) had validation accuracies of 98% and 97.3%. The confusion matrices from the first and second are shown in the tables below.

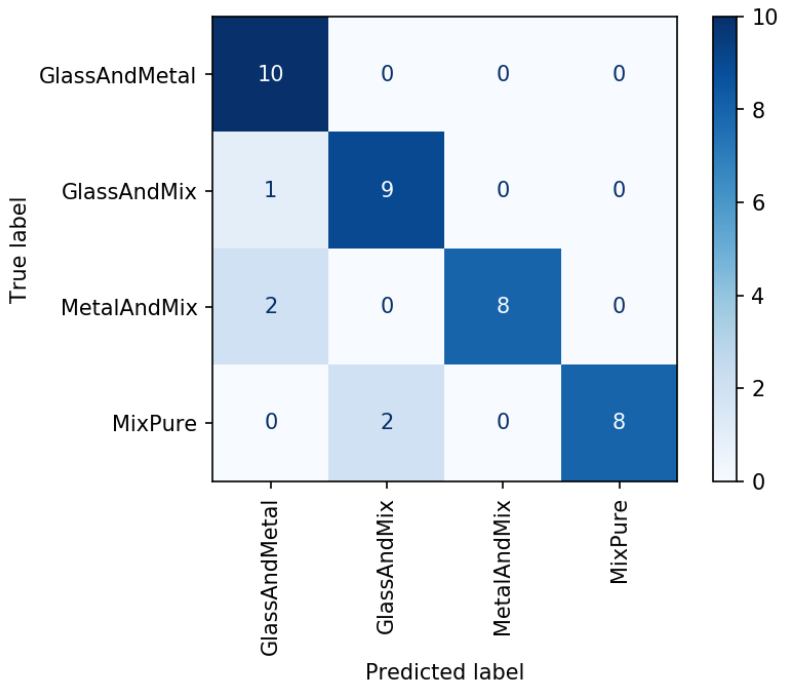



Table 19: Confusion matrix, model with highest validation accuracy

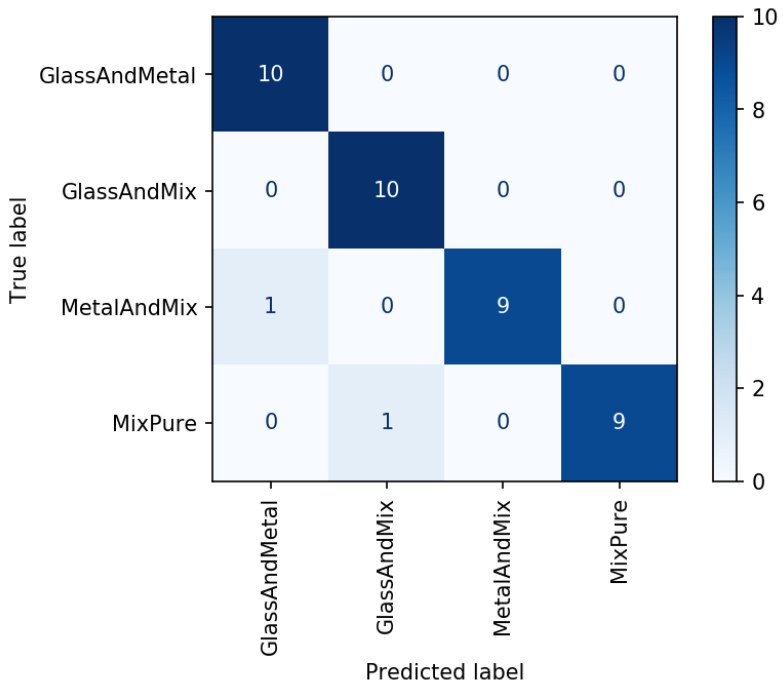



Table 20: Confusion matrix, model with highest test accuracy

Table 21 shows the results from the single best model trained with multiclass classification, as opposed to the 20 run averages presented earlier. It also shows percision and recall for each class.

| | All | | GM | | GMX | | MMX | | PMX | |
|---|---|---|---|---|---|---|---|---|---|---|
| *Model* | *Validation Accuracy* | *Test Accuracy* | *Test Precision/Recall* | | *Test Precision/Recall* | | *Test Precision/Recall* | | *Test Precision/Recall* | |
| M-1 | 99.01% | 87.5% | 76.9% | 100% | 81.8% | 90% | 100% | 80% | 100% | 80% |

Table 21: Single best model from multiclass

### 3.3.2 Multi Labelling

Another way of approaching the problem is multi-labeling. With multi-labeling (see Section 2.2) one is able ask two question at the same time:

*1: Is there **metal** in the bag?*
*2: Is there **glass** in the bag?*

The two question are independent and not mutually exclusive. As such the problem can be framed in a simpler way while maintaining the level of detail, that is, being able to specify the exact combination of materials present. Table 22 shows a truth table involving two parameters, "Metal" and "Glass", corresponding to each class as used in multiclass.

| | | Multilabel labels | |
|---|---|---|---|
| | | Metal | Glass |
| Multiclass labels | PMX | 0 | 0 |
| | GMX | 0 | 1 |
| | MMX | 1 | 0 |
| | GM | 1 | 1 |

Table 22: Individual labels corresponding to the four classes

In essence, multilabelling and multiclass are the same. It is possible to go back and forth and frame the problem in both ways after attaining results simply by assigning labels from Table 22 to all elements in the confusion matrix shown in Figure 20.

It does, however, make more sense to use multilabelling as it is related to the questions that matters for REN. Was there glass and/or metal in the trash bag? Multilabelling presents the answer to this in its simplest form. Additionally, multilabel simplifies the model output by reducing the units to two in

the last layer. This does, however, require a change in activation function(See Section 2.2.2) after the last layer, as Softmax requires the sum of output confidences to be 1. Using Sigmoid activation function enables the output of two independent confidences. How this affects the performance of the models developed is investigated further.

#### 3.3.2.1 Training Multi Labelling Models

As changing to multi label could potentially alter the accuracy of the model, Model 1, Model 2 and Model 3 as presented in Section 3, were once again trained on the data set resulting from the ablation study, 20 times each, with 70 epochs. Both individual and macro averaged accuracy, precision and recall are registered on both validation and test data on each run. As such, maximum and average scores were found for the three different architectures. The results of this will be presented here.

Table 23 shows the results achieved using multi labelling on all three models developed. The first three rows presents the model achieving the single highest validation accuracy, while the following three shows average scores obtained. Results presented under "All" are all macro averaged, as equal amount of significance is attributed to both classes.

| | All | | | Metal | | | Glass | | |
|---|---|---|---|---|---|---|---|---|---|
| *Model* | *Accuracy* | *Precision* | *Recall* | *Accuracy* | *Precision* | *Recall* | *Accuracy* | *Precision* | *Recall* |
| **M-1** | **98.14** | **99.48** | 96.78 | **100** | 100 | **100** | **96.28** | **98.95** | 93.56 |
| M-2 | 97.52 | 97.32 | **97.77** | 99.75 | 100 | 99.5 | 95.29 | 94.63 | **96.04** |
| M-3 | 97.02 | 96.62 | 97.52 | 99.75 | 100 | 99.5 | 94.29 | 93.24 | 95.54 |
| M-1(avg) | **97.24** | 97.34 | **97.22** | **98.92** | 99.19 | **98.68** | **95.56** | 95.49 | **95.77** |
| M-2(avg) | 94.98 | **97.85** | 92.09 | 98.91 | **99.95** | 97.86 | 91.05 | **95.76** | 86.31 |
| M-3(avg) | 96.25 | 97.32 | 95.25 | 98.23 | 98.99 | 97.49 | 94.27 | 95.65 | 93.02 |

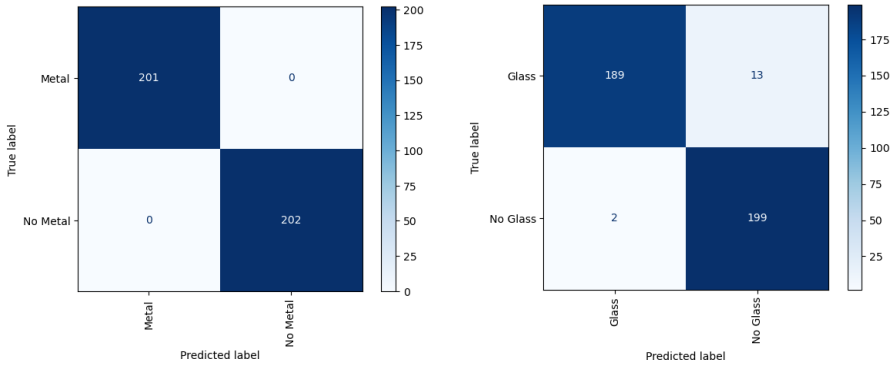
Table 23: Validation results of Model 1, 2 and 3 using multilabel

It can be seen that Model 1 (M-1) reaches the single highest accuracy of 98.14%. Model 1 is also the best performer in overall precision (99.48%), accuracy on metal (100%), recall on metal (100%), accuracy on glass (96.28%) and precision on glass (98.95%). Model 2 (M-2) does outperform M-1 on overall recall

(97.77%) and recall on glass (96.04%). Regarding the average scores, M-1 obtains the best in all categories except overall precision, precision on metal and precision on glass, where M-2 scores higher.

As multi labelling and multiclass are mostly similar, it was expected that M-1 performed best on the validation set as it did when using multi-class. The best multiclass model did get higher validation accuracy (99.01%) than that of the best multilabel model (98.14%). Additionally, multiclass obtains 0.7% higher average validation accuracy with a P-value of 0.00009.

Figure 53a and 53b shows confusion matrices for the best of the 20 trained Model 1 on metal and glass respectively. As suggested by the results presented above, every prediction regarding metal is correct. Glass, on the other hand, misses 15 times, where 13 of them are false negatives.




(a) Metal (b) Glass

Figure 53: Confusion matrices on validation data for best obtained Model 1

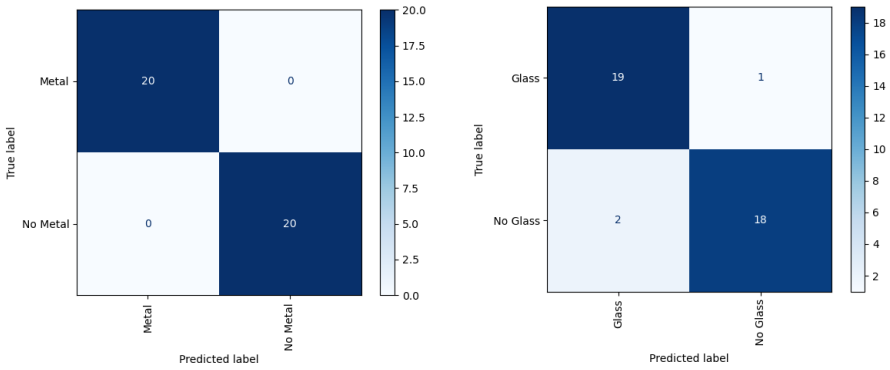
Table 24 presents the score on the test data set.

| | All | | | Metal | | | Glass | | |
|---|---|---|---|---|---|---|---|---|---|
| *Model* | *Accuracy* | *Precision* | *Recall* | *Accuracy* | *Precision* | *Recall* | *Accuracy* | *Precision* | *Recall* |
| M-1 | 96.25 | 95.24 | 97.50 | 100.00 | 100.00 | 100.00 | 92.50 | 90.48 | 95.00 |
| M-2 | 96.25 | 95.24 | 97.50 | 100.00 | 100.00 | 100.00 | 92.50 | 90.48 | 95.00 |
| **M-3** | **98.75** | **100.00** | 97.50 | 100.00 | 100.00 | 100.00 | **97.50** | **100.00** | 95.00 |
| M-1(avg) | 93.81 | 92.58 | 95.63 | 99.50 | 99.07 | 100.00 | 88.13 | 86.08 | 91.25 |
| M-2(avg) | 93.56 | **93.14** | 94.75 | 98.25 | 97.45 | 99.25 | **88.88** | **88.83** | 90.25 |
| M-3(avg) | **94.13** | 92.74 | **96.50** | **99.75** | **99.52** | 100.00 | 88.50 | 85.96 | **93.00** |

Table 24: Test results of Model 1, 2 and 3 using multilabel

On the test data, M-3 produces the best model within all categories except overall recall, accuracy, precision and recall of metal, and recall of glass, where all models are equal. M-1 and M-2 obtains the exact same results. M-3 also scores better on average in most categories, except overall precision, recall of metal, and accuracy and precision of glass. In these categories, Model 2 (M-2) obtains the best result.

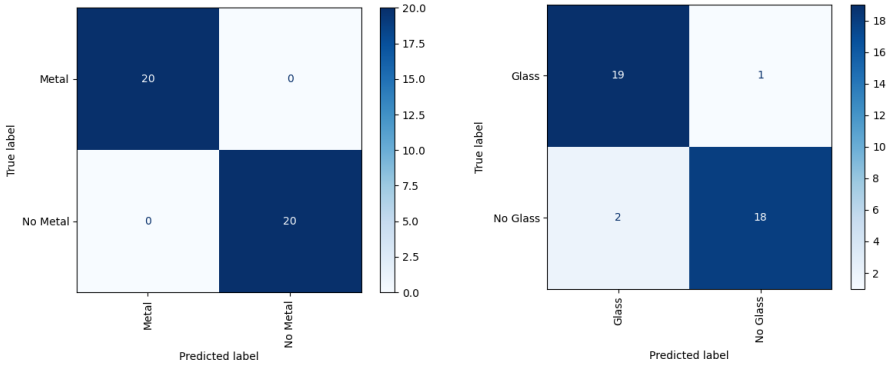
Figure 54 shows the confusion matrices of metal and glass on the test set for best obtained M-1. No mistakes were made regarding the classification of metal. On glass, 3 false predictions where made with 1 false negative and 2 false positives.




(a) Metal (b) Glass

Figure 54: Confusion matrices on test data for best Model 1

As can be seen in figure 24, M-3 achieves high results. Its confusion matrices are shown in Figure 55




(a) Confusion matrix on test data: Metal

(b) Confusion matrix: Glass

Figure 55: Confusion matrices on test data for best Model 3

#### 3.3.2.2 Optimal Threshold

When using multi-labelling a threshold must be defined in order classify a sample as either positive or negative of the two classes. In all results presented above, the default threshold has been used, which is 0.5. This makes it essentially equal to multi-class, due to the sum of all probabilities being one, once a class probability is greater than 0.5 means that it must be greater than all other classes. An algorithm was developed to find the threshold that constitutes the highest possible accuracy. It was in fact found to be 0.5.

By increasing the threshold, less positive labels would be applied by the model resulting in less true positives that in turn results in higher precision and lower recall. If decreased, more positive predictions would occur resulting in more false positives and thereby lower precision but higher recall. In other words, there is a certain trade-off between the amount of true positives and false positives and thereby precision and recall, when altering the threshold. Figure 56 shows a curve that is constructed using several thresholds from 0 to 1 and the resulting true positive rate (recall) vs false positive rate (ratio of false positives to all positives) and thereby presents this trade-off for the validation set. How or big or small this trade-off is can be described by the area under the curve (AUC), which when low constitutes a high trade-off.

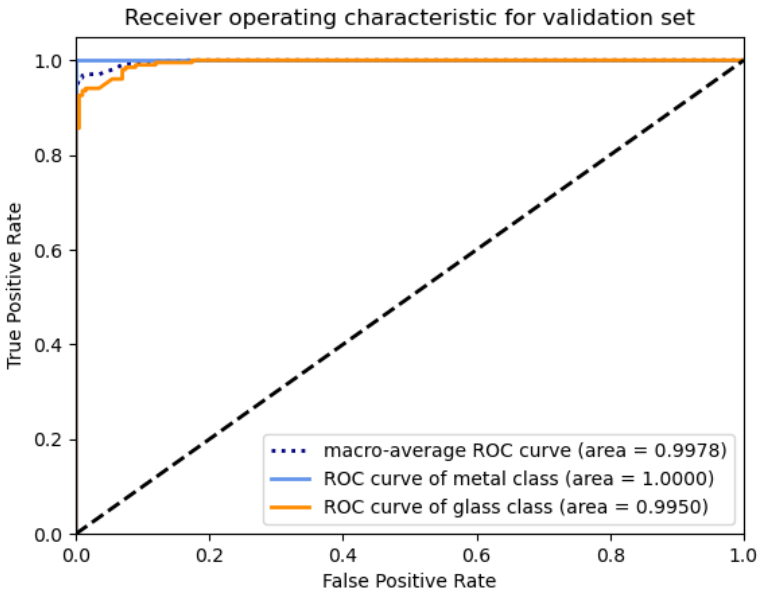



Figure 56: ROC M-1 validation set

As can be seen in the figure, there is not much of a trade-off. The model is quite fast able reach a high recall when increasing the threshold. Figure 57 shows a closer look at the slope of the curve.

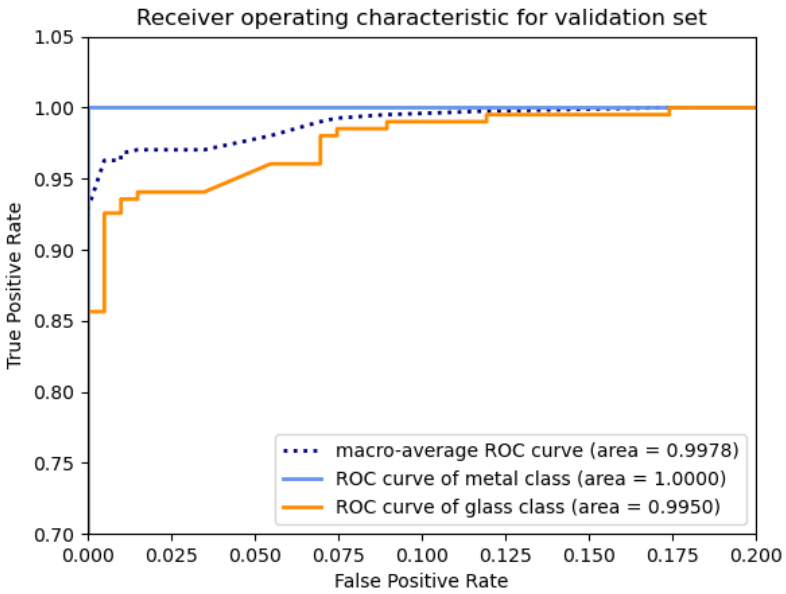



Figure 57: ROC M-1 zoomed

Figure 58 shows the ROC of the classes on the test set. Here the trade-off is larger than on the validation set, however the model still reaches quite a high recall rather quickly once increasing the threshold.

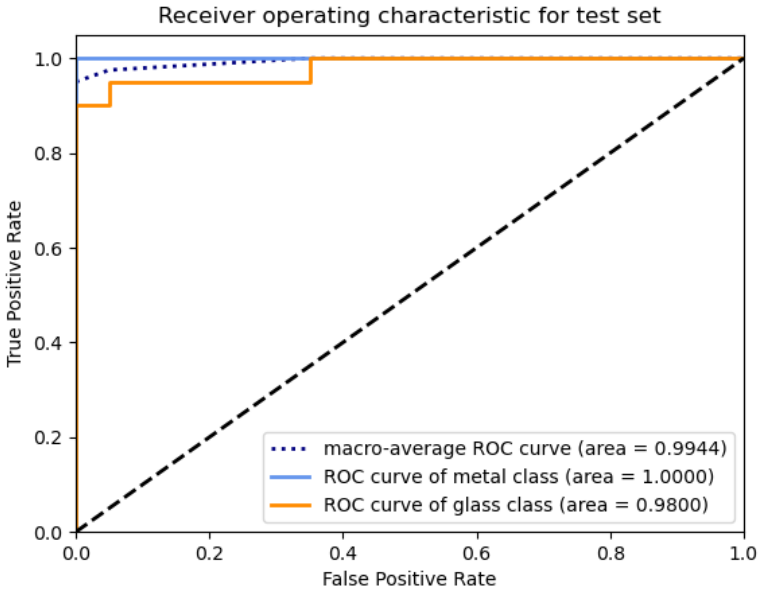



Figure 58: ROC M-1 test set

By for instance choosing the threshold that constitutes the point on the curve illustrated in Figure 59, one would achieve a false positive rate of about 0.05 and a true positive rate (recall) of 0.95. This threshold would be 0.98, meaning that the model seems quite confident of its predictions. This threshold would result in a validation accuracy of 97.14% and an unchanged test accuracy. As such, a threshold of 0.5 was chosen.

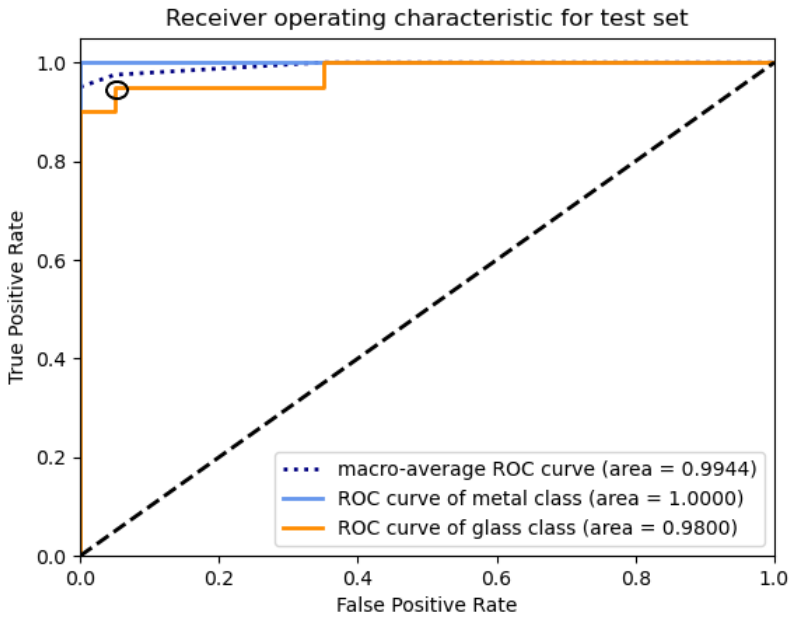



Figure 59: ROC M-1 test set, marked

# 4 Results and Discussion

In this section, results and their implications regarding the two main components of this Master's will be presented and discussed. In the first part of this section an evaluation of the Trash Collection Experiment will be given along with a discussion on its implications. The second part will contain results and discussions on the ablation study and the CNN model. For each, the results will be presented first, after which an analysis and interpretation of the results will be given.

## 4.1 Trash Collection Simulation

In the Trash Collection Experiment, a simulation of the emptying of trash bags was created successfully with moderate realism. The construction was comprised of relatively cheap and easily accessible materials and was easy to build. The framework was able to withstand loads caused by the trash bags while maintaining reliable sensor readings. The complete system worked as intended, both with regards to the mechanics and the electronics. The trash bags were flipped and sent through the chute correctly and all sensors readings where initiated at correct timings.

At first, the plexiglass used exhibited too high friction causing some bags to lag and even stop completely when sent through the chute. To remedy this, a silicone spray was applied to the surface. This allowed the bags to easily slide through the chute. A plastic material was chosen as to mimic the sound of bags sliding out of trash bins. Plexi glass and silicone are quite different materials from the trash bins(PE)) and a slightly different sound can be expected. The hope was to show the plausibility to listen to the sliding as well, and another plastic material was seen as more resembling than for instance MDF. However, in the experiment almost no sound were generated until the impact, especially when the silicone spray was applied. To get a closer resemblance, an actual trash bin could have been used instead.

Even though the metal detector was developed using rather cheap and obtainable components, it exhibited decent readings, with a clear spike if metal was present. Figure 33 shows readings of four different classes. As can be seen, the data is rather erratic. A filter could potentially be applied upon recording the data in order to get a smoother curve.

The condenser microphone was able to record a wide range of frequencies and strengths, thereby producing detailed spectrograms. Figure 35 shows that the average strength of different frequencies at impact for each class used are relatively different from each other. Almost all classes are significantly different($p < 0.05$) from each other on almost every frequency step. This indicates that the differences between the classes are effectively caught. This shows that the condenser microphone was able to record with adequate resolution and consistency.

The contact microphone was also able to record many frequencies, though on a lower strength level than the condenser microphone as shown in Figure 36. The standard deviations of the different frequencies are much larger than on the data captured by the condenser microphone, indicating that the contact microphone suffers more from inconsistency. It should still be possible to distinguish GM from PMX with high accuracy by comparing the strengths of the non-overlapping frequencies. It possible that an increase in gain on the contact microphone could be beneficial in order to capture more of the details, however, it is unknown how that would affect the variance. For this, further testing should be considered in the future.

The camera producing videos of the emptying has not yet been utilized in this masters thesis. An example of its usage will be explored in 5. It did indeed record as expected and was able to capture most of the emptying event using a wide field of view (FOV). From time to time the camera would tilt down due to heavy forces hitting the metal tray, such that a more secure fastener should have been chosen.

The weight readings showed both good consistency and high accuracy in a standalone test. For this experiment, the relative weight was seen more crucial

than what the actual weight was. It did not matter for the proof of concept if the scale was set with a non-zero unloaded value, as long as this remained the same for all measurements. When placed in the rig, the consistency diminished greatly, as shown in Figure 41. In the end, the weight data was not used in classification as it was quickly realized that it affected the models negatively.

The inconsistency of the weight readings was likely due to the how the weight was recorded. Because the loading tray was attached in one end, the weight scale may have recorded the torque as opposed to downward force. Unless the bag was placed with its center of mass directly above the weight scale each time, the recordings would vary. By filing out bigger holes for the axles leaving a wiggle room and thereby removing the upward reaction force, the consistency could be improved. This was attempted, but was likely not done thorough enough. Nonetheless, having a certain amount of inconsistency was also seen as advantageous as the same bags were used to create several readings. The sound recording and metal detection would vary inherently between every measurement as the orientation of the bag could affect the readings. In a sense, this resulted in a completely "new" reading on every measurement of the same bag. If the weight scale was perfect, the orientation of the bag would not affect the weight reading, such that even though having different sound and metal detection readings for several measurements of the same bag, the weight would be the same every time. This could potentially create biases in the network. Therefore, the inconsistency in the weight measurements was left unattended.

Using the proximity sensor as a trigger mechanism proved to work well. On very few occasions did the proximity sensor not register the trash bag sliding past. Additionally, a few ghost readings were experienced. However, if the bag was placed too close to the edge, the sensor could be triggered due to the plastic bag handle triggering the sensor. Either using a sensor with slimmer detection area or moving the sensor further down the chute would eliminate this problem.

The GUI increased the efficiency drastically. In the end, about **10-15 seconds** were required for each measurement such that **240-360 measurement** could be taken **per hour**, given all bags were prepared in advance. For coherence with multi labeling, the labeling scheme in the GUI was changed in the final version. Here one must simply check a box if the bag contains the specified material, as shown in Figure 60. Additionally, the possibility of choosing class in Page 1 has been removed. Now PureMix is set as default class. The folder system was kept the same. For a future version, control of contact microphone and video recording should be added as well. Especially regarding the camera, as it were tilted down from time to time. This would have been noticed at once if a frame from the video had been presented in the GUI.

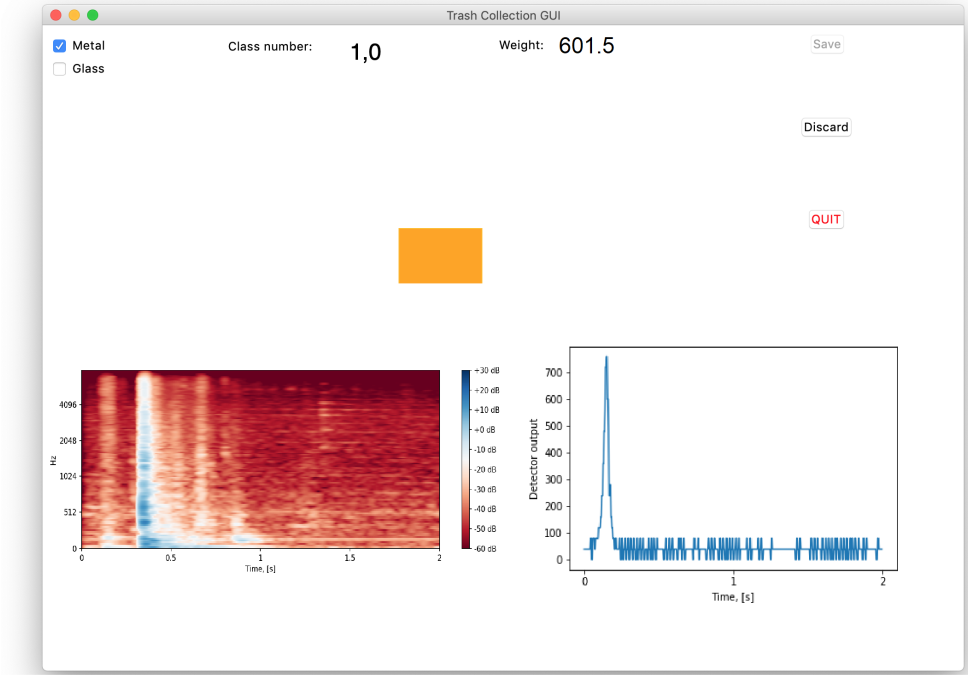



Figure 60: GUI: Page 2, Final Edition

## 4.2 CNN Model

### 4.2.1 Ablation Study

#### 4.2.1.1 Results

The ablation study (see Section 3.2.3) performed showed that not all data gathered would contribute to improved detection rate. Table 25 shows the effect of different alterations of included data and preprocessing on the test accuracy. Table 26 shows the resulting sensors and data that in conjunction with M-1, demonstrates best detection performance on the test data set.

| Test | Data | Average test score change | Number of tests | P(T<=t) |
|---|---|---|---|---|
| 0 | **All data except weight** | **0.0%** | 60 | Reference |
| 1 | Only Mel spectrogram | -4.0% | 60 | 0.006 |
| | Only MFCC spectrogram | -6.2% | 60 | 0.0001 |
| 2 | Normalized metal data | -2.3% | 40 | 0.193 |
| 3 | **Only condenser microphone** | **+3.5%** | 40 | 0.032 |
| | Only contact microphone | -7.5% | 40 | 0.00004 |
| 4 | **Frame extracted spectrograms** | **+4.6%** | 40 | 0.001 |
| 5 | Weight data included | -6.4% | 40 | 0.0002 |

Table 25: The effect of sensors and preprocessing of data on the test accuracy

| *Sensors* | *Data* |
|---|---|
| Condenser mic R | Mel-Spectrogram |
| Condenser mic L | Mel-Spectrogram |
| Condenser mic R | MFCC |
| Condenser mic L | MFCC |
| Metal detector | 2D matrix(stacked 1D arrays) |

Table 26: Sensors and data for optimal detection rate on used data set

The ablation study shows clear advantages of choosing condenser microphones over contact microphones. Additionally, a better result is achieved if including both Mel-spectrograms and MFCCs. The modification that contributed most to increased performance, was the extraction of relevant frame around the moment of impact. The weight was excluded from model development rather early, as it was seemed clear from early on that the results would improve. The ablation study later showed that this was correct.

#### 4.2.1.2 Discussion

The model achieves better results by using both Mel-spectrograms and MFCCs. It may be because it enables the CNN to extract different features from Mel-spectrograms and MFCCs that are both relevant to the classification. Additionally, by extracting the relevant frames from the spectrograms, the CNN might be able learn features more specific to the actual impact, rather than having to make account for the surrounding irrelevant noise or silence. This may have eradicated a potential bias where the location (on the horizontal axis) of the impact could be a deciding factor of which class the sample belonged to.

The favoring of condenser microphone may be due to its consistency. The contact microphones, as described in Section 3.1.5, exhibited much larger variance, which may have been the cause of its negative contribution to detection rate. A dynamic microphone might also be used, though in general condenser microphones have a larger frequency range which may be important considering analysis of the frequencies in Section 3.1.5. It should also be noted that the contact microphones used in the experiment were inexpensive and might not have been of very high quality. Installing less hardware is also desired considering the amount of trucks that will need modification, since it will reduce costs. Generally, it is recommended to perform an ablation study after implementation, as this has proved an effective and simple way of finding the best data sources.

The negative impact of the weight is likely due to large variances found within the measurements(see Section 3.1.5). As can be seen from Figure 41 it is difficult to distinguish the classes from mere weight data. It is possible that the weight could have contributed if there was information about the size of the bags as well. Since the bags could have many sizes, weight would have to be combined with its size in order to extract information about the content of the bag. If a method as will be described in Section 5 was implemented, the weight might have been more useful. This method involves extracting the amount of pixels classified as being part of the trash bag, thus giving a representation of the area and thereby adding context to the weight.

### 4.2.2 CNN Model

#### 4.2.2.1 Results

Both multi-class and multi-labelling models have been developed and tested. The multi-labelling has been chosen as the favorable labelling scheme using Sigmoid as activation function on the final classification layer. The labelling approach used is shown in Table 22.

Several models have been tested. M-1 reaches the single best (98.14%) and average (97.24%) validation accuracy. Its average validation accuracy is 2.26% higher than Model 2 ($p = 0.0003$) and 0.99% higher than Model 1 ($p = 0.00009$). The architecture of M-1 is shown in Table 27 and its results are presented in

Table 28.

| *Layer #* | *Layer Type* | *Hyper-parameters* |
|---|---|---|
| 1 | Convolutional | Filters = 32, Kernel size = (3,3), Activation function = ReLu |
| 2 | Max Pooling | Pool size = (2,2) |
| 3 | Dropout | Dropout rate = 0.2 |
| 4 | Convolutional | Filters = 64, Kernel size = (3,3), Activation function = ReLu |
| 5 | Max Pooling | Pool size = (2,2) |
| 6 | Dropout | Dropout rate = 0.2 |
| 7 | Convolutional | Filters = 128, Kernel size = (3,3), Activation function = ReLu |
| 8 | Max Pooling | Pool size = (2,2) |
| 9 | Dropout | Dropout rate = 0.2 |
| 10 | Convolutional | Filters = 256, Kernel size = (3,3), Activation function = ReLu |
| 11 | Average Pooling | Pool size = (2,2) |
| 12 | Dropout | Dropout rate = 0.2 |
| 13 | Convolutional | Filters = 256, Kernel size = (3,3), Activation function = ReLu |
| 14 | Average Pooling | Pool size = (2,2) |
| 15 | Dropout | Dropout rate = 0.2 |
| 16 | Global Average Pooling | |
| 17 | Fully Connected | Units = 2, Activation function = Sigmoid |

Table 27: Model 1 architecture

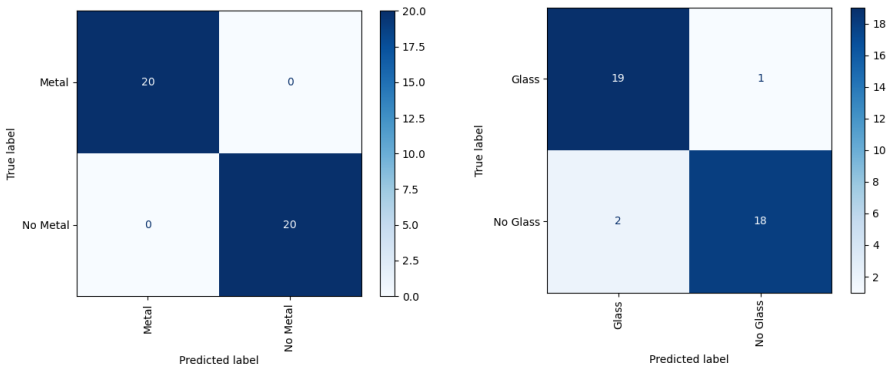
| | All | | | Metal | | | Glass | | |
|---|---|---|---|---|---|---|---|---|---|
| *Model* | *Accuracy* | *Precision* | *Recall* | *Accuracy* | *Precision* | *Recall* | *Accuracy* | *Precision* | *Recall* |
| M-1(val) | 98.14 | 99.48 | 96.78 | 100 | 100 | 100 | 96.28 | 98.95 | 93.56 |
| M-1(test) | 96.25 | 95.24 | 97.50 | 100 | 100 | 100 | 92.50 | 90.48 | 95.00 |

Table 28: M-1 Results

Table 28 shows the results of the best out of 20 trained M-1 on the validation set and test set. A validation accuracy of 98.14% means that the model can reliably determine the presence or absence of glass and/or metal in the validation data set. The positive classifications across both classes delivered by the model are correct 99.48% of the time (precision). Among all positives that should have been reported, the model reports 96.78% of them (recall). All metrics for metal are maximized, meaning that no mistakes where made when predicting presence or absence of the class. Regarding glass, 98.95% of the positives reported were correct (precision), whereas 93.56% of the total amount of samples containing glass where reported (recall). Scores are lower on the test set, though only with regards to the glass class. Recall is higher than precision, 95% and 90.48% respectively, meaning that the model favors a more complete delivery of actual positives than a precise one, on the test set.

As seen in Figure 61, the model makes no mistakes regarding the metal class. Whereas three mistakes are made on glass. As mentioned earlier, the ratio between correctly sorted bags to wrongly sorted bags is about 20:1, based on the statistics presented in section 2.1.2. Using equation 6, it can be shown that one can expect around 0.21 false positives for every true positive if using the validation score as basis. With the test score, 2.1 false positives can be expected for every true positive in real application in Oslo.

$$\frac{20}{1} * (\frac{FP}{FP+TN} \div \frac{TP}{TP+FN}) \tag{6}$$




(a) Metal (b) Glass

Figure 61: Confusion matrices on test data for best Model 1

#### 4.2.2.2 Discussion

Even though the multi-class model achieves a better validation accuracy($+0.7\%$, $p = 0.00009$), a multi-labeling approach is favored. The reason is that framing the question in such a way seems to be more coherent with reality. Additionally, if more classes are to be introduced at some point, it is easily scalable with multi-labelling by directly adding the classes, whereas the amount of classes(output neurons) needed for multi-class would increase exponentially as every combination must be a separate class. This was seen as more valuable than the 0.7% extra validation accuracy multi-class could provide on the data set used.

M-3 does achieve better on the test set with maximum 98.75% and average 94.13% accuracy as opposed to a maximum of 96.25% and average of 93.81% for M-1 as shown in Table 24. The increased score on the test set is, however, not significant($p = 0.66$) such that it can be attributed to coincidence. As such, it cannot with confidence be said that M-3 in general will perform better than M-1 in glass and metal detection. On the validation set M-1 performs better with significance ($p \leq 0.05$), such that one could argue that it has a higher probability of achieving the best performing model when several of them are trained for the real scenario.

It cannot be easily determined which metric is more important to maximize. Having a higher precision than recall means that the false negatives outnumbers the false positives. This may be favorable, as many false positives can lead to a consumer being falsely accused of poor sorting, for this a threshold even higher than 0.98 could be chosen. On the other hand, REN is likely interested in catching as many bad sorting cases as possible such that they may take measures that affects more of the people that are not sorting correctly, in that case a lower threshold should be chosen. It really comes down to what measures REN

wants to take. If they are related to money, for instance by giving fines to people that do not sort or a relief in waste collection fees to people that do sort, it may be of high interest to avoid miscarriage of justice (high precision). However, if the measures are to increase the information given to the people that do not sort, by for instance putting up posters in the immediate area, then perhaps including more of the bad sorting cases (high recall) is more important than all of them being correct. In the precision and recall scores reached in the data sets used here the trade-offs are not very big, however, it is likely that a model trained on several bags being emptied at the same time in varying locations will have substantially more FPs and FNs, as will be discussed in Section 5.1.

The metal classification reaching 100% accuracy is not a surprising result when studying the metal data collected in the experiment. The indications are so clear that this classification could be done manually as shown in Section 3.1.5, where an accuracy of 98.4% was reached. It seems, however, that the model was able to learn to ignore the sporadic readings shown in Figure 38 such that it reached an accuracy of 100% on the metal class. Additionally, many glass containers have metal lids, so the metal detector may add more information to the glass classification as well.

There is a certain difference in the corrected amount of false positives for every true positive between validation and test set. The 2.1 FP for every TP for the test set is a substantial amount. This can, however, still be seen as an acceptable number considering that the false positives will over time be spread out evenly across all locations in the real-world, while the true positives will concentrate in areas of poor sorting, assuming such trends in sorting exist.

# 5 Limitations, Applications and Future Work

In this section, the main limitations and drawbacks of the system will be discussed followed by its application. Furthermore, future potential additions will be explored. It must be noted that other applications are discussed in the article in Section 7 as well.

## 5.1 Limitations

### 5.1.1 Trash Collection Simulation

The trash collection simulation were developed with the intention of mimicking the real life scenario. It must be stated that the evaluation of resemblance given in this thesis is mostly based the authors personal experience, experience from several visits with REN and interviews with employees at REN. One of these visits included a complete ride along on two trash collection routes such that the

authors could experience trash collection at first hand giving a deeper insight into the trash collection operation. In order to get a more scientifically grounded evaluation, however, readings should have been performed on the truck as well. Recordings of bag impact from the truck and the simulation could be compared, for instance by comparing the strengths of different frequencies. As such, one could discover the potential shortcomings of the simulation and alter it thereafter.

Even though using an actual trash bin as chute could ensure resemblance, it has been convenient to have a transparent material. During operation, this only improves the experience for the operator. However, it has been helpful during the development and prototyping of the rig. The rig could have been nearly fully developed using transparent material, after which the completion could be done with an actual trash bin.

The chosen microphone was a rather expensive component, and it is unclear how well a cheaper microphone will catch the details of the impact. It is certainly something that should be tested prior to initiating any big scale development. If a sufficiently cheap microphone is not able to record with enough resolution then it would halt the project.

It must be noted that the analysis performed in Section 3.1.5 represents the differences in the obtained data set and does not necessarily reflect how every future reading of trash bags within those classes will be. There are many metal and glass objects that were not in any of the bags measured. It is expected that these objects would make sounds similar to other objects of the same material, but whether it resembles sufficiently for successful detection is unclear. Furthermore, the fact that GM has higher average metal detection peak may be simply because the bags were consistently loaded with more metal than that of the MMX class as it often got additional metal from lids of glass containers. Lastly, the PMX trash bags contained trash collected on the campus of NTNU and may not be the most representative mixed trash.

### 5.1.2 CNN Model

The use of sound recordings and metal detection with CNN seems a promising approach for detecting glass and metal when considering the results achieved. The high test accuracy of 96.25% might indicate that there is a good possibility of a high accuracy when the system is installed in the intended location. The primary purpose of the test set was to benchmark a model's applicability in a semi-real scenario. Whether or not this test set actually captured much of the varieties found in reality is not apparent. As it only contains 10 measurements of each class, there are likely many more variations. However, it was believed to be the most varying data set obtained in this project such that using it to mea-

sure the models' generality was seen as the most valuable aspect of it. There is a possibility that the test set, even though being different from the rest, resembled itself such that it simply caught one other variation. If that is the case, then perhaps it is best put to use by simply including it in the training. How much 40 measurements as opposed to 2000 would affect the weights of the model is unclear, however.

Another purpose of the test set was to possibly verify the hypothesis that one bag could effectively represent several trash bags of the same class. The fact that there was significant discrepancy between the scores on the data sets, could not refute the hypothesis either. The differences in the data sets may be due to other circumstances. For instance, two people were in the room instead of one when it was recorded, such that it may have caused changes to the acoustics and thereby the sound data. There might also have been other unknown methodical errors. In any case, an analysis of the effect of reusing bags should be performed. For instance by comparing the variances of a data set containing unique bags and a data set containing reused bags. What is certainly clear from the discrepancy between the scores, is that the training set does not vary enough to capture the reality.

It must be stated that the high accuracy scores has been achieved in a highly controlled environment (i.e. the trash collection simulation) and is likely to perform poorer at the back of a truck. Several factors can lead to the results deviating from this experiment, such as dissimilar ambient noise and different sensor placement. The largest unknown factor is that the collection trucks empty several bags at the same time. In the experiment the bags where emptied separately, so especially the sound might differ greatly compared to the real situation. Experimenting with several bags at a time on the test rig is probably not the best way to progress further, as it will only yield information on that particular subject. As there are other unknown factors, the recommended next step will be to install the system on a collection truck so the results will give a definite answer on the full applicability of the system.

Neural networks in general vary greatly and there are near infinite combinations of layers and model parameters. There is no golden standard for all tasks, let alone all sound recognition tasks. A large part in finding a suitable network constellation is trial and error. It is hard to explicitly recommend a specific model for further work as the data set representing the reality will likely differ substantially from the data set used here. All models perform quite well on the data set used, M-1 did the best, however it is unclear whether the same model would perform best in the real situation as well. The experiment does show that the task of recognizing glass and/or metal using CNN seem quite possible.

## 5.2 Application in Waste Collection Cycles

A glass and metal detection system may have many applications in a trash collection cycle, but one of the more beneficial application is perhaps early in the collection cycle, at the back of a service truck, where traceability of bags is still intact as mentioned in Section 1. Gathering the data here would result in a clearer image of people's recycling behavior. The system developed in this thesis is with good accuracy able to detect the presence of glass and/or metal in singular trash bags hitting a metal tray. In reality, bags come in batches; however, being able to classify one bag at a time was seen as a necessary first step to show the feasibility of using sound for recognition. It must be stated that the trained models presented cannot be directly used with the emptying of several bags. For this, a new model must be fitted and trained using appropriate data. It is unclear whether one could achieve similar results having a full bin of trash bags, but having the results of the models developed here in mind, it does show promise. Testing with the system mounted to a waste collection truck was desirable, but unfortunately not possible due to the COVID-19 pandemic.

If a robust model for detecting the presence of glass and/or metal when emptying trash bins were to be developed, any municipality collecting waste that should not contain glass or metal using waste collection trucks, may benefit from it. Most of the bigger cities of Norway collect glass and/or metal separate from the rest ([38],[2],[1],[32]), and may make use of such a system.

There is no doubt that more data on the recycling behavior is beneficial for improved waste management. It enables decision makers to make fact based decisions that may result in more efficient measures.

## 5.3 Including More Video Data

The video camera was included in the experiment with the aim of enabling other possible means to extract information through image recognition. One such method is to extract the area of the bags and thereby adding a much needed context for the weight measurements. As mentioned previously, one possible reason the weight did not improve detection rate, was that bags would vary much in size such that the weight varied a lot as well. With the added surface area, the weight could contribute through determining a sort of density of the trash bag, and since different materials have different densities, this could provide additional patterns for the CNN to detect.

There was an attempt at estimating the surface area of the bag using Mask R-CNN [16] with moderate success. Figure 62 shows a detection performed by Mask R-CNN. Mask R-CNN has not been covered in this Master's Thesis, such that the reader is referred to their study for more information.

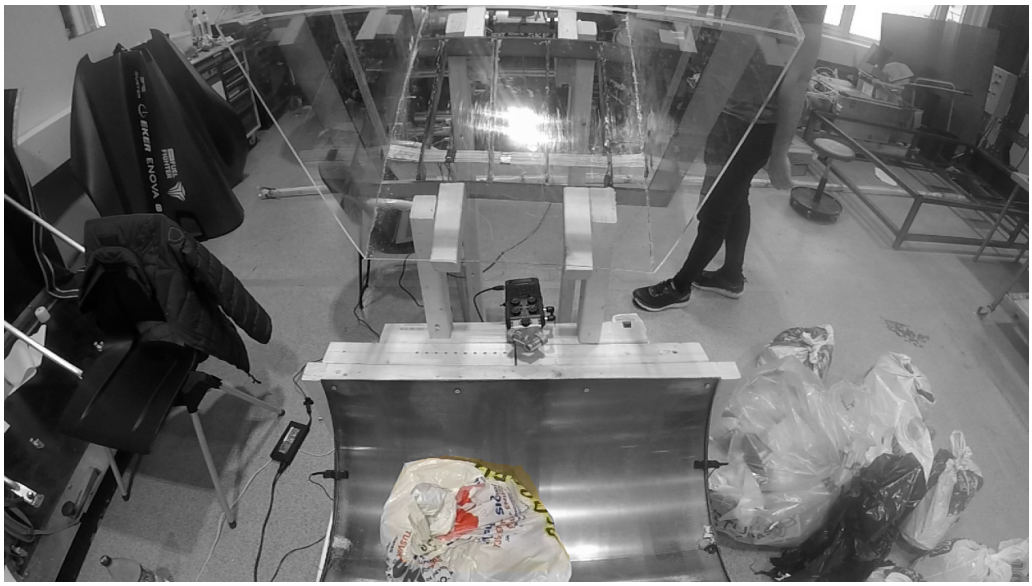


Figure 62: Mask of trash bag found

By counting the amount of pixels in the resulting mask, one could either use that number as is, or attempt to estimate the actual area by accounting for the camera placement and FOV. In this case, the 9837 pixels where classified as being part of the trash bag.

This approach was not taken further as much of the camera footage was faulty. As for instance seen in the figure above, the trash bag is slightly out of the picture. This would result in faulty area readings as well.

Extracting a picture of the bag slightly before impact was also attempted. Figure 63 shows a picture of a bag mid fall that could be used instead of the last image. However, since the camera were tilted from the forces of the impact from time to time, the camera angle was not consistent. Extracting a certain frame in a specific moment, would thereby not result in consistent area readings.

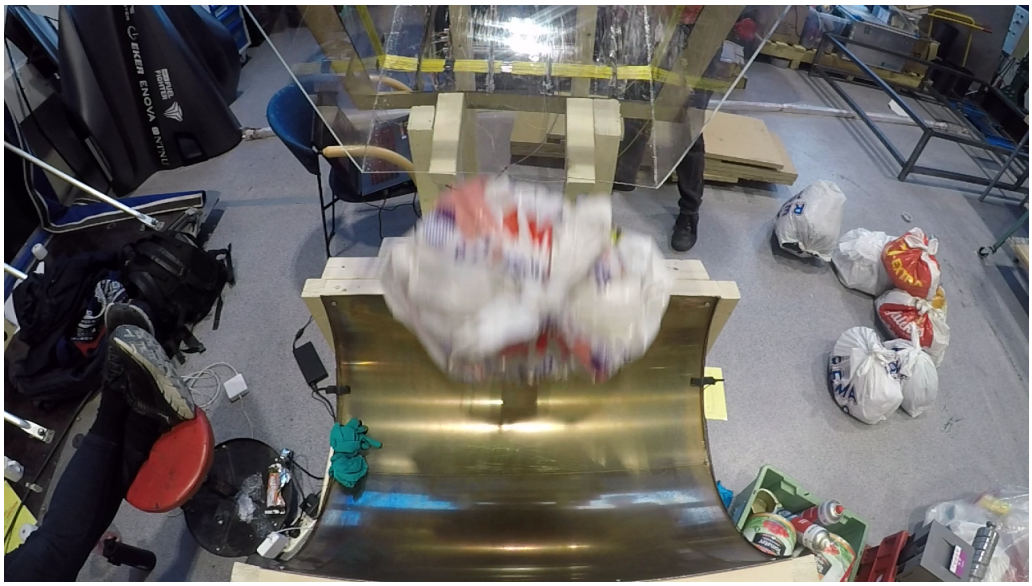


Figure 63: Trash bag mid fall

There was and idea of training a model to detect the masks of trash bags well enough, so that it could detect when the trash bag was in the middle of the picture, and first then extract the area. This could result in slightly more consistent readings, however, this would suffer from the bag not being positioned at the same distance from the camera every time, thereby still producing faulty area readings.

Focus were placed elsewhere in this thesis, but the method has potential for future work.

## 5.4 Detecting More Than Glass and Metal

As is evident by the distribution of materials shown in section 2.1.2, there are other types that could be useful to be able to detect. At the time, REN collects food waste in separate bags, but in the same bin as plastic and mixed waste. A new approach is being tested which has a separate bin for food waste, and if this project continues it will be necessary to measure the amount of food that is disposed of in the wrong place, i.e. with the mixed waste or plastic waste. Other types of sensors could possibly work well to identify food or other materials, and may also improve the functionality of the existing system.

In the paper "Multi-material classification of dry recyclables from municipal solid waste based on thermal imaging" [15], thermal imaging is used to sort between iron, stainless steel/aluminum, plastic/paper and wood. The materials

were heated before an image was taken, and machine learning was then used for classification. The accuracy obtained was between 85-96%. Some possible problems with including this in the system is that heating may be required, which takes time and energy, and the imaging may not work as well when the waste is contained inside a plastic bag.

Another method is called Hyperspectral imaging, which is a form of spectroscopy using different wavelength of light to determine the material composition of an item. This method shows promise, but as thermal imaging it has only been tested for loose objects. A study by Zheng et. al. reports a 100% classification accuracy of different plastics, although with a sample size of only 94 [44]. It is possible however that this method could work for objects inside plastic bags as some wavelengths of light can penetrate through the bag and therefore expose the objects within in the image. Some studies of transmittance in polyethylene indicate this possibility, including one by Balocco et. al. [6].

## 5.5 Other Possible Implementation Areas for the System

While the system has been designed with the goal of installing it on a garbage truck, this is not the only location it could prove useful. Another possible location could be inside the sorting facility, where the bags are separated by color. During this sorting the bags are transported between stations on conveyor belts, and over time they become separated into either single bags or small clusters of 2-3 bags. If a drop is included somewhere along the line, the sound of the bags landing can be recorded, and it would be trivial to install a metal detector there as well. This could be used to separate bags containing glass or metal before they reach their final destination, preventing unwanted materials contaminating biogas production, plastic recycling or similar. The major downside of this placement is that there is no information regarding the origin of the bag, yet a considerable advantage lies in easier installation and the possibility of measuring a single bag at a time, simplifying the operation. Also, there are many collection trucks that will need modification, but in the sorting facility only a few measuring stations will be required.

Another possible location could be in shared garbage chutes, for instance outside condominium buildings. These often have chutes that need to be unlocked with a personal RFID tag, so the bags could be measured and the measurement linked to the individual who disposed of it. An advantage here is that the chutes are generally too small to dispose of several bags at the same time, so the measurement would always read a single bag at a time. From personal experience, these chutes are about the same size as the plexiglass chute in the testrig, so the equipment could be installed in a similar way, perhaps with an impact spot along the way for the bag to generate sound before it lands among the previously disposed of bags. The trash from these shared containers are emptied by dedicated trucks, so installing the system here would not exclude the need for using it on the trucks that empty the single household bins. Therefore it would

be recommended to install it in both locations.

# 6 Implementation in Oslo

This section has the aim of enabling employees at REN to fit the system presented in this Master for their own trash collection cycle. For that, they should replicate the experiment performed in this Master, using their own service cars and trash gathered from their clients. This will hopefully result in a model capable of classifying the presence of metal or glass in a bin of trash bags, that they may use to gather information about peoples recycling habits regarding glass and metal.

An overview of an execution plan on how to develop and implement the system on the service cars is shown in Figure 64. This execution plan includes suggestions on the selection of sensors and their placement, how to gather the training data efficiently and sensibly using the GUI developed in this Master, how to develop and train a CNN model, and how to test and implement a finished model.

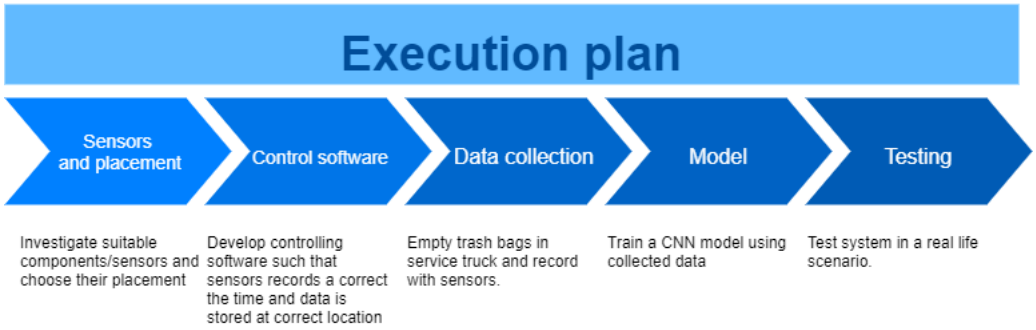



Figure 64: Suggested execution plan

## 6.1 Sensors and Placement

### 6.1.1 Sound Recording

There are many possible ways to go about sound recording, and due to none of us being experts in the field, this part will only include suggestions on how it can be solved easily. The quality of the recording is of high importance as the differences between bags containing metal and glass is very slight compared to normal mixed waste. The operating environment is harsh as it is exposed outside weather, so the equipment should be of high quality as well as being made for rough operating conditions.

A good candidate is the recorder used for the measurements in the project. The Zoom H6 can deliver multi-channel audio to a small device like a Raspberry Pi for recording. If our approach is used, the sound is recorded and stored using a short python script which is included in the ready made program. Other recorder types can be used without any changes as long as the device outputs audio in separate channels to the receiving device. The Zoom H6 supports 6 channels, while other types like the Zoom H4 supports 4 channels. This project has not investigated how many channels should be applied as this depends on the size of the tray on the back of the truck. The amount of microphones must at least be enough to pick up sound from any location in the tray. A suggestion for amount of microphones and location will be provided at the end of this section.

As concluded earlier, contact microphones have poorer performance than condenser microphones. There are several types of condenser microphones, but the safest choice would be one covering a wide range of frequencies and with directionality suited to covering the whole area. The condenser microphone used in the project was the one included with the Zoom recorder. This microphone has a range of 44.1kHz at 16-bit resolution. This has proved adequate for this project, and in lack of testing will be considered sufficient for later implementation. It should be noted that in the study "An investigation of the usability of sound recognition for source separation of packaging wastes in reverse vending machines" [19], dynamic microphones are suggested because of similar performance to condenser microphones at a lower cost.

Placement of the microphones will depend on how many are used, and their directionality. To minimize outside noise it would be beneficial to have some directionality to the microphone. The zoom microphone could be set to either 90 or 120 degrees, meaning that it registers sound in a cone within the set angle. On the test set 90 degrees was used, which was appropriate due to its placement.

A suggestion for microphone placement is shown in figure 65. These areas are not subject to any moving parts and also less prone to damage. It is not mandatory to use four microphones as shown in the picture, but generally more sensors will improve the quality of the data collected. It will probably be sufficient with two microphones if the directionality is wide enough to cover the whole tray. The placement of the Zoom recorder (Zoom H4n shown in the picture) can be selected freely, a suitable location is easier to decide once other components are considered.


Figure 65: Microphone placement suggestion (picture from REN, altered) (microphone clip-art designed by Freepik)

### 6.1.2 Metal Detection

The metal detector used in the project is cheap and easy to build, but has some issues that must be considered. The detector reacts well to metal passing through the search coil, but the range outside the coil is very limited. At about 10cm for a beverage can, the range is not sufficient to cover the required area. Therefore it could be difficult using it for truck applications. It would probably be better to use an adapted commercial detector with a range of about 0.5 meters that could output its signal directly to a Raspberry Pi or similar. This solution would prove more compact, and probably be more robust, which is of importance due to the harsh conditions it would be operating under. Employing the metal detector used in the project in the same manner would be impractical and prone to damage from debris, as illustrated in figure 66 where the yellow drawing indicates the search coil.

Figure 66: Bad choice for metal detector implementation (Original image from Joab.se, altered)

A commercial, compact metal detector could possibly be modified and mounted to the hydraulic arm that lifts the bins. This would put it in close proximity to the bags as they exit the bin and given a sufficient detection range, be able to pick up any metal as the bags pass the detector. Such a detector has to be specialized to some degree, and it would be advisable to consult with someone experienced on the subject for selecting the device to use. This solution might well prove to be the easiest and most reliable. A commercial detector typically has a search coil with diameter about 200mm. One example claims to detect a coin at about 120mm depth in the ground, which seems to be a quite common range for these. This puts the range somewhat lower than desired as the smallest bins have a width of 480mm and the larger ones are 1200mm wide. Metal objects found in the bins will however generally be larger than coin sized, so the range might be adequate as it depends on object size. The example detector here is one of the cheap models, some claim search depth of 250mm and more, depending on the size of the object. An example of a possible installation location is provided in figure 67. To cover the whole area several coils will have to be installed if the commercial circular types are employed.

Figure 67: Suggested metal detector installation area (picture from REN, altered)

### 6.1.3 Proximity Sensor

Using a proximity sensor in the same was it was employed in the test rig would be difficult on a waste collection truck. The sensor could be triggered by outside factors, like the workers, or miss the bags entirely due to the less constrained area they pass through when falling from a bin. There is however already a system in place that can be used instead of a proximity sensor. Some trucks in Oslo already has a sensor mounted to the hydraulic lifting arm that measures the angular position. The feedback from this sensor can be used in place of the proximity sensor to start recordings, and the recordings can be stopped by a time counter in the program in the same way as the test rig operates. This is probably the easiest solution as it is already developed and is not likely to be triggered by any outside factors.

## 6.2 Control Software

In order to control the initiation of each of the sensors described above, it is recommended to utilize an Arduino as it provides easy prototyping and control of sensors. It is also highly recommended to utilize the GUI developed in this Thesis as it enables the operators to control each and every measurement taken as to ensure the readings are not faulty. Both scripts are added in the Appendix and may be used and tweaked freely as seen fit by the employees at REN. The camera already installed on the trucks is connected and controlled by a Raspberry Pi. It is possible to connect to this unit through WiFi as it creates

its own network. This way, commands can be sent to it from a computer telling it to initiate video recording.

## 6.3 Data Gathering

The initial step in data gathering will be to produce bags which will be measured and used to train the machine learning model. There are several factors to consider when making these bags to ensure the system functions as intended and to insure optimal performance.

*Realism:*
Creating bags that resemble the bags found under normal operations is of highest importance. It may be tempting to just use bags that have been collected during daily operation directly, but this will require that the contents is checked for any unwanted items. If a bag is measured and labeled as containing no glass or metal, it is vital that this is actually the case. The only exception here would be very small amounts that are considered acceptable. During the project we encountered empty cardboard beverage containers that had an aluminum lining on the inside. These were registered by the metal detector, but we included them as mixed waste as we deemed them acceptable in this category. Decisions concerning if very small amounts of metal and glass should be in the accepted category must be made by the appropriate authority on the subject.
This leaves two options for the accepted bags, either go through the bags already collected to ensure no unwanted items and store them for use, or make entirely new bags. The recommended approach will be discussed in the end of this section.

*Variety:*
It is important that the bags are varied in contents, size and composition. The reason for this is to reflect the real conditions as precisely as possible. Bags containing different amounts of metal and glass should be represented, from the smallest amount deemed unacceptable to the highest amount deemed realistic. Each of these amounts must be present in several bags with varying contents to ensure that as many possible realistic combinations are represented in the training data set. Size of the bags is also important, as this is a variable encountered in real operation. Weight and size relation should be taken into account as it is probable to encounter small and heavy bags as well as large and light bags. In short, the goal should be to include as many possible combinations of these variables as possible, the only exception being if some combination will never be encountered during operation.

*Quantity:*
In general, the more measurements taken, the better. At some point though, the training set will become so large that the computing power required will present a problem. Also, the performance of the system will not improve infinitely, so collecting too much data would be a waste of time and resources.

The question then becomes how many measurements to take, which does not have a definite answer. Based on experience from this project, and a general idea from other machine learning projects, the amount of measured bags in each category should be at least between 500 and 1000. If the measurements are easy to obtain, 1000 of each category will be a good starting point. The model can then be evaluated and one can decide if more measurements are needed. One additional consideration is that each class should have about the same number of measured bags. If one class is over-represented it may lead to the model favoring this class during operation.

*Producing bags for measurement:*
It is known that REN performs a manual sorting event one or two times every year. During this event employees manually go through trash bags and record the contents to assess the quality of the household sorting. This event is perfect to combine with the production of bags for building the test set. As the bags are processed, they can be rebagged after the contents is recorded so the contents is perfectly controlled. Alternatively, the bags can be measured before the contents is checked. As most of the bags encountered during this process will probably not contain metal or glass, it is recommended that these materials are procured beforehand so they are ready to combine with bags for the test set. It is very important that all personnel participating are briefed on the factors detailed in this chapter so the bags created uphold these criteria. In addition, one must remember to tag the bags in some way to make sure that the contents is known for later.

## 6.4 Training the CNN Model

The models presented in this Master has been reached through an iterative procedure where many different constellations and settings has been tested. For REN to reach a model best fit for their system, a similar approach should be taken. However, it is sensible to first try the architectures used in this Master, though, other architectures should be tested as well. REN is currently working on machine learning solutions and possesses both knowledge and experience within the field [4] such that it is expected that they could develop their own model.

After which a round of data gathering has been performed, an ablation study should be conducted. By removing one data source at a time, which sensors that contribute to increased detection rate may be uncovered. If additional data sources are added at a new round of data gathering, all previous data sources even though deemed unnecessary by the ablation study, should be included in a new ablation study. New data sources might give context to previously used data sources, and only by including everything may this be revealed.

Every combination of included data should be saved as an array locally on the computer. This way, one may run one main script training a model on

all data sets. An example of training M-1 both with and without contact microphones is shown in Appendix A.2.5. If one wishes to train several models, additional main files can be created importing different models in the first line. A batch script may then be created running the different main files as shown in Appendix A.2.7.

# 7 Journal Paper "Detecting improperly sorted content of trash bags during waste collection using convolutional neural networks"

This section contains the paper that was written about the work performed in this thesis. The complete article is presented in the next page.

# Detecting improperly sorted content of trash bags during waste collection using convolutional neural networks

List of authors:

Oliver Istad Funch[a,*]

Robert Marhaug[b,*]

Sampsa Kohtala[c,*]

Martin Steinert[d,*]

Authors' affiliation(s):

[*]Norwegian University of Science and Technology (NTNU), Department of Mechanical and Industrial Engineering, Richard Birkelands Veg 2B, 7491 Trondheim, Norway

[a] oliver.funch@gmail.com

[b]robertmarhaug@gmail.com

[c]sampsa.kohtala@ntnu.no

[d]martin.steinert@ntnu.no

Corresponding author:

Name: Sampsa Kohtala

Email: sampsa.kohtala@ntnu.no

## Abstract

We present a proof-of-concept method for classifying the presence of glass and metal in consumer trash bags. With the prevalent utilization of waste collection trucks in municipal solid waste management, the aim of the method is to help pinpoint the locations where waste sorting quality is below accepted standards, making it possible and more efficient to develop tailored procedures that can improve the waste sorting quality where it is needed the most. Using trash bags containing various amounts of glass and metal, in addition to common waste found in households, we use a combination of sound recording and a beat-frequency oscillation metal detector as inputs to a machine learning algorithm to identify the occurrence of glass and metal in trash bags. A custom-built test rig is developed with the purpose of mimicking a real waste collection truck, which is used for testing different sensors and building the data sets. Convolutional neural networks have been trained for the classification task, achieving accuracies up to 98%. The promising results supports potential implementation in real waste collection trucks, with the method enabling location-specific and long-term monitoring of consumer waste sorting quality which can provide decision support for waste management systems, and research on consumer behavior.



**Keywords:** Trash bag classification, convolutional neural network, waste collection, sound recognition, metal detection


**Nomenclature:**

| | |
|---|---|
| BFO | Beat-Frequency Oscillation |
| CNN | Convolutional Neural Network |
| GM | Glass and Metal |
| GMX | Glass and Mixed waste |
| HMM | Hidden Markov Model |
| MFCC | Mel Frequency Cepstral Coefficients |
| ML | Machine Learning |
| MMX | Metal and Mixed waste |
| PG | Pure Glass |
| PM | Pure Metal |
| PMX | Pure Mixed waste |
| REN | Renovasjonsetaten |
| RFID | Radio Frequency Identification |
| ROC | Receiver Operating Characteristics |
| SVM | Support Vector Machine |

# 1 Introduction

The increasing focus on preserving the global environment has led to marked changes in the municipal waste management policies around the world. The EU has set targets for recycling of household waste at 50% within 2020, 55% within 2025, 60% within 2030, and 65% within 2035 according to Avfall Norge, the field association for recycling in Norway (Wilsgaard, 2018). Currently, this number is around 35% according to Renovasjonsetaten (REN), the department responsible for municipal waste management in Oslo.

According to Adhithyarasanna and Kaushal (2018), waste should be sorted at the earliest stage possible to reduce contamination during the recycling processes. Improper sorting can lead to bag rupture due to sharp edges, contamination of plastic reducing the recyclability (J. Almankaas & J. H. Ellefsen-Killerud, personal communication, 4 January 2020), contaminated biogas production (Jiang et al., 2020), and contamination during waste incineration. Recycling materials like aluminum is also a preferable alternative to incineration as the energy costs of aluminum production is much greater than the cost of recycling. The current municipal waste system in Oslo, as illustrated in Figure 1, relies on home sorting by the consumer. Consumers sort their waste into bags of specific colors, which are later separated in a waste sorting facility. Neither glass nor metal should go in these bags as a separate collection point is dedicated for these materials. For this system to function, it is vital that the consumer performs this sorting (Rousta & Ekström, 2013), and does so correctly since the sorting facility only considers the distinct bag color and not its content. Radio frequency identification (RFID) tags are fixed on the waste containers outside people's homes, which are read by the waste collection trucks to record the location of each emptying, along with a timestamp and the weight of each waste bin. The degree to which the contents of the colored bags are correctly sorted is not known during this collection process. Today, knowledge about incorrect sorting in Oslo is gathered by manually analyzing a portion of collected waste once a year. 4259kg of waste from 10 different areas were analyzed and reported in 2019 ("Avfallsanalysen 2019," 2019). The report found that in the mixed waste category that will ultimately be incinerated, only 26.6% in weight of the contents belonged in that category. Of the materials not belonging in this category, the fourth largest was glass and metal, constituting 6.1% in weight. There was also a significant difference in sorting quality between the geographical areas, where the amount of wrongly sorted trash varied from 9.8% to 63.4% in weight.

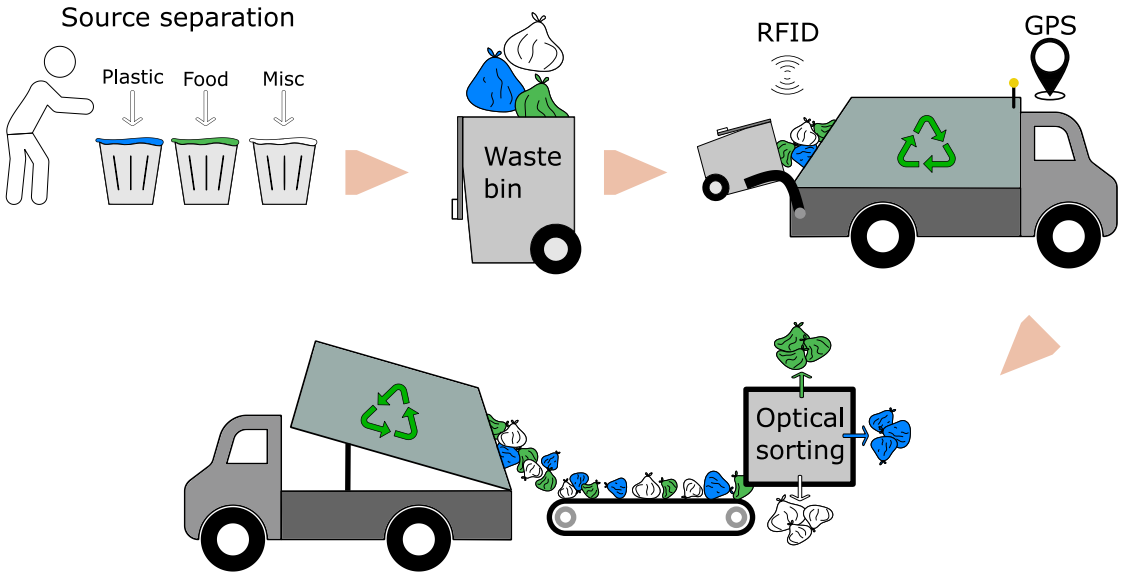



*Figure 1: The waste management cycle that is the basis for this study, which starts from the source separation by consumers and ends at the optical sorting facility.*

Attempts have been made in previous works to improve on consumer sorting, either by developing an automatic system to perform the sorting instead of the consumer or a system for quality control of the home sorting task, usually relying on some sort of classification method. The most prevalent shortcoming of these approaches is the use of single objects per datapoint (Adhithyarasanna & Kaushal, 2018), often requiring or

assuming that a dedicated sorting system is able to accurately separate piles of trash or rely on consumers to separate each trash individually. Meanwhile, convenience is one of the driving factors for source-segregation of waste in households (Bernstad, 2014; Rousta et al., 2017), such as having separate bins for different waste types. Consumer sorting requires knowledge and incentives to function properly. Being able to identify and link consumers to their trash sorting habits can help localize the areas where sorting behavior needs improvement. Rada et al. (2013) stated that uncontrolled road containers in Italy had lower sorting quality, indicating that consumers are less concerned when the trash bins are not directly relatable to them, thereby making a link between consumers and sorting quality an incentive for changing behavior.

The selected approach in this study aims to identify glass and metal in the collected waste without having to inspect the contents of the individual bags manually. Glass and metal are chosen as they are commonly sorted wrong by consumers who are not using (or are not aware of) the designated collection points for these materials. Our method utilizes a combination of sound recording and a beat-frequency oscillation metal detector as inputs to a machine learning algorithm to identify the occurrence of glass and metal in trash bags. The intended implementation for the method is in the trash collection trucks, with the aim of collecting information about the quality of sorting during the normal collection routines. As a system with RFID is already in place for the trash bins scheduled for collection, our approach aims to help pinpoint the locations where waste sorting quality is below accepted standards, making it possible and more efficient to develop tailored procedures that can improve the waste sorting quality where it is needed the most, thus improving the overall waste management system. Other potential benefits associated with collecting this type of data will also be discussed.

An experimental design has been developed including a custom-built test rig for capturing data sets of trash bags containing different levels of glass and metal. For the proof-of-concept trash-classification system, we generated bags in six different categories for training and evaluating several machine learning (ML) models. Multiple sensors are tested to find the best combination of input data for the models, and the final models are presented with results and a discussion on the implications and prospects of our approach for supporting waste management systems.

# 2 Related work

## 2.1 Solid waste classification

A common trend for classifying solid waste in literature is based on analyzing image data. Specifically, Convolutional Neural Networks (CNNs) have been heavily used in multiple applications after gaining popularity over the past few decades, mostly due to its impressive accuracy in classifying images with an increasing training speed (Krizhevsky et al., 2012). A CNN is a supervised learning method, mapping two-dimensional input data (e.g. images) to output data (classes or categories). The main idea behind CNNs is to automatically learn to extract relevant features and find patterns from the input data (LeCun et al., 1998). In simple terms, a CNN consists of the input and output layer, with hidden units in-between consisting of convolutional- and pooling layers for extracting features and a fully connected neural network for calculating class probabilities based on the features. Using the input and output examples (i.e. labeled images) the CNN is able to update its weights through backpropagation to improve the classification accuracy of the model being trained. One of the challenges with CNNs is acquiring the large amounts of data that is often needed to properly train and tune a model.

In one of the earlier works on applying machine learning to classify solid waste by Yang and Thung (2016), they generated a dataset called TrashNet with roughly 2400 images of solid waste in six different categories (glass, paper, metal, plastic, cardboard and general trash). They achieved an accuracy of 63% using an SVM (support vector machine) with a feature detection algorithm, and 22% using a CNN. Bircanoğlu et al. (2018) experimented with several different CNN architectures on the TrashNet dataset. They achieved the highest accuracy of 95% by fine-tuning a pre-trained model and using data augmentation (flipping and rotating training samples) to increase the data set size. Similarly, using randomly initiated weights for the model, Ruiz et al. (2019) achieved an accuracy of 89%. Lindermayr et al. (2018) extended the TrashNet dataset by capturing more images of trash. To resemble their application of roadside trash detection, they also generated synthetic data by segmenting the trash and adding different backgrounds to the images. Their best model

reached an accuracy of 84% for the more challenging dataset. Toğaçar et al. (2020) combined an auto encoder, CNNs and an SVM to classify images of waste as organic or recyclable from a large dataset of over 20k images, and achieved nearly 100% accuracy. Chu et al. (2018) developed a classification system utilizing images together with weight and metal detection sensors. Their method, called multilayer hybrid method, used a CNN for extracting image features and sensors to capture numerical features. Their model had an accuracy of over 90% for classifying items such as paper, plastic, metal, glass, and food waste as recyclable or not. A classification model for electronic waste including the ability to estimate the object size (object detection) was presented by Nowakowski and Pamuła (2020), showing an accuracy of over 90%. Korucu et al. (2016) used a Hidden Markov Model (HMM) and SVM to classify materials based on sound. Their approach is similar to ours, although their focus lies in source separation of packaging waste in reverse vending machines. They achieved up to 100% classification accuracy when measuring free falling impact sounds from glass, plastic, metal and cardboard, in addition to high accuracy when estimating the size of the objects as well.

Most of the waste classification methods found in literature are based on images of single objects with homogeneous backgrounds, in addition to often lacking a description of direct real-world applications. While it is valuable to develop methods for automatically classifying any kind of waste, it is important to consider their applicability in today's waste management systems in order to reach the fast-approaching goals set by the EU.

## 2.2 Sound Recognition

Several studies have shown that sound recognition is a good method for determining material properties of objects. Giordano and McAdams (2006) showed good results on material recognition by humans based on impact sounds between gross material categories, but found that the acoustical and source properties contains sufficient information for identifying even sub-categories of same materials. Consequently, a machine learning algorithm might be able to perform even better than the human by detecting small variations in the data. A study by Gong et al. (2019) demonstrated the use of microphone, accelerometer and gyroscope data generated from a smartphone to recognize objects using an SVM. By simply knocking the phone against an object they were able to identify its material. The accuracy was high when using sound, but poor from the other sensors which further indicates that sound is well suited for detecting materials.

Machine learning techniques usually employ spectrograms to process and visualize sound data and to highlight distinguishing features. Two widely used types are the Mel spectrogram and the Mel Frequency Cepstral Coefficients (MFCC) spectrogram. To create a Mel spectrogram, a Fourier transform is applied to time segments of the sound clip which shows the energy present for each frequency. The frequency axis is then scaled to a Mel scale, which is a log scale created to mimic how a human experiences sound (Volkmann et al., 1937). The energy axis is also scaled to a decibel scale as the experienced sound volume is not linear (Chapman, 2000). The MFCC spectrogram is similar to the Mel spectrogram, but includes one additional processing step using a reverse Fourier transform which results in a Cepstrum. This spectrogram has peak values where there are periodic elements in the time segment (Noll, 1967).

Traditional sound recognition often utilizes an HMM, gaussian mixture model or SVMs (Ananthi & Dhanalakshmi, 2015; Deng & Yu, 2014; Li et al., 2017; McLoughlin et al., 2015; Mesaros et al., 2010; Sharan & Moir, 2016), and several studies have also shown promising results using CNNs (Hershey et al., 2017; Khamparia et al., 2019; Kumar & Raj, 2017). CNN models often perform better in chaotic environments (Zhang et al., 2015), and are able to extract more abstract features while unaffected by local variations (Çakır et al., 2017). In the case of trash collection, a lot of irregular acoustic noise can be expected, for which a CNN model may perform well.

## 2.3 Metal detection

Metal detection is a well proven concept which is widely used in multiple applications. Common methods include Beat-Frequency Oscillation (BFO), very low frequency and pulse induction. The type we have used is a BFO detector, mainly due to its simple design. The BFO creates an oscillating frequency by using an LC-circuit. The LC-circuit causes an oscillation frequency due to the capacitor first discharging to the inductor, and thereafter being charged by the induced voltage created in the coil. The charging and

discharging are time-shifted between the components, which causes the current to rise and fall. As the internal resistance of the components cause the energy to dissipate, the circuit requires an external power supply to keep the oscillation going. The frequency depends on the inductance of the search coil of the detector, which is made by winding insulated cable in a loop. If a metal object is in close proximity to the coil, the inductance changes which in turn changes the oscillation frequency. By storing this frequency and constantly comparing it to the current frequency, any change will indicate the presence of metal. An Arduino microcontroller can be used to perform this comparison and output the detection results.

# 3 Method

## 3.1 Experimental setup and data recording procedures

A custom test rig was built for recording sound and metal detector data from trash bags. The purpose of this rig is to mimic the part of the truck where the trash bins are emptied during the normal collection cycle, making the data acquisition similar to real conditions. Figure 2 shows a comparison of the tray in a waste collection truck and the landing tray of the test rig. Apart from the difference is size, the trays are similar in appearance. This simplified approximation provides more control over the experiments in addition to enabling rapid prototyping of the classification system.

*Figure 2: The landing tray of a waste collection truck under operation is shown on the left and the test rig on the right.*

The main parts of the rig consist of a loading tray, a chute, and a steel landing tray. The chute is approximately the same size as a normal thrash bin to simulate the velocity of trash bags before impact. A distance sensor is located at the top of the chute for registering when a trash bag is sent through. Next to the landing tray is mounted a Zoom H6n recorder which functions as a sound board, including two stereo condenser microphones attached to the recorder and two additional contact microphones that are placed on each side of the landing tray. A length of wire is wound around the chute which works as the search coil for the metal detector. The setup also includes a weight sensor beneath the loading tray, a GoPro camera mounted at the end of the landing tray, and a circuit board containing an Arduino Nano and a HX711 load cell amplifier. The rig is illustrated in Figure 3.

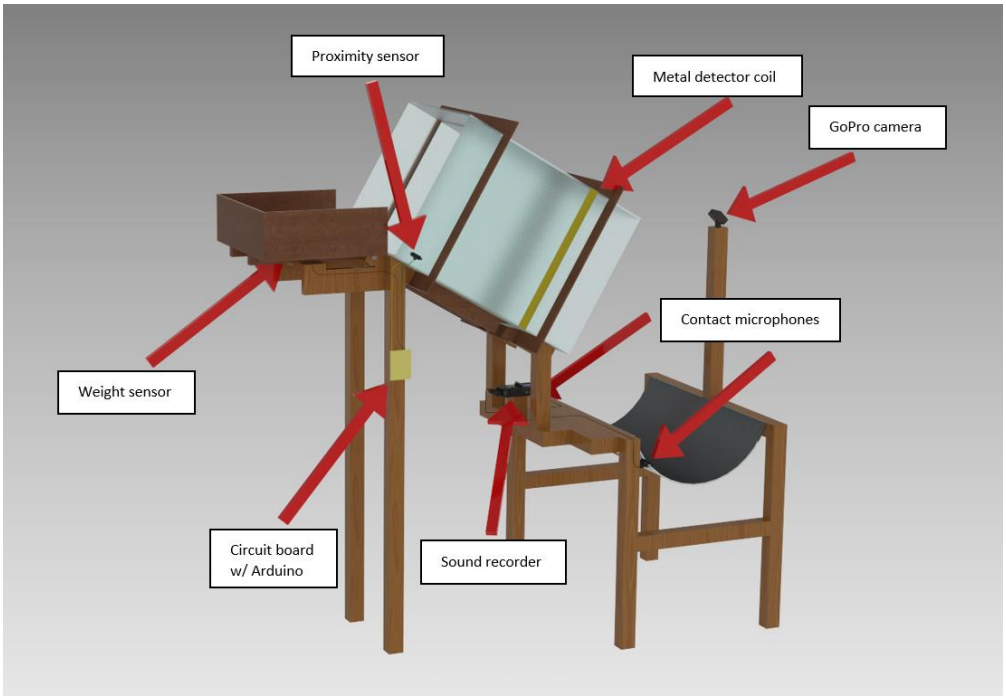



*Figure 3: 3D-model of the test rig showing all the components used for data collection.*

Trash bags were created in six different categories: Pure metal (PM), pure glass (PG), metal and mixed waste (MMX), glass and mixed waste (GMX), glass and metal (GM), and pure mixed waste (PMX). Each category contains about 500 measurements. Since glass and metal are generally found mixed with other waste in a real-world scenario, PM and PG were omitted from the data set, reducing it to about 2000 samples. Care was taken to ensure that all bags had different size, weight, and composition within each category to simulate real conditions. The glass used was mostly bottles and jars that are normally found in households. The metal is an assortment of beverage cans and tins from canned food and some scrap sheet metal. The mixed waste category is composed of waste found in trash bins around the campus of the Norwegian University of Science and Technology where the experiment was conducted. These materials were thoroughly inspected to ensure that no metal or glass were present. The categories MMX and GMX were made sure to include samples of high, medium, and low metal or glass content. Due to the extensive time requirement of producing unique bags for the entire training set, each bag was reused several times. In total there were about 20 unique bags present for each category. We hypothesize that when the same bag is used several times with random orientations, the resulting sound and metal detection characteristics will be different. To verify our hypothesis, a separate data set (test set) was created containing 40 unique bags (10 for each category), where the bags were not reused.

Each bag was recorded individually when capturing the data sets. Weight was automatically recorded after the bag was placed in the loading tray. After tilting the loading tray to initiate the recording procedure, sound and metal detector data was recorded automatically for 2 seconds after the distance sensor was triggered. Video recording started when the weight was measured, and later automatically edited to include only 2 seconds after the distance sensor was triggered. The procedure was monitored from a custom graphical interface, allowing erroneous measurements to be discarded directly, making the data collection more robust.

## 3.2 Data and model preparation

In this study, several CNN-based models are developed and tested. A simple ablation study is conducted to determine which sensors and input data contribute to model performance, including different sound recorders, sound data representations (Mel spectrogram and MFCC spectrogram), metal detector data and weight. The ablation study was performed in consecutive order, where inputs contributing to significant increase in accuracy (P-value below 0.05) are included in the next test. Consequent tests are compared to the previous best result.

Mel spectrograms and MFCCs were created using the Python package Librosa. Examples of both spectrograms, including metal detection data, are shown in Figure 4, each taken from the same sample. To reduce the sample size and computational power needed for training, an algorithm was used to detect the moment of impact to extract a smaller frame around the event. The time window of 2 seconds was originally divided into 1379 columns, or time segments. 64 segments before the impact and 192 after the impact were extracted for a total of 256 columns for each sample in the data sets, as it was assumed that more features of interest would occur following impact. The vertical axis was resized to 256 using bilinear interpolation resulting in a shape of 256x256. The metal detector data originally recorded 600 data points in a 1D-array, totaling 2 seconds of data. This is longer than required as the bags are only briefly passing the metal detector during the initial part of the recording, but it was preferred to ensure that all bags would be captured regardless of sliding speed through the chute. The data was interpolated to 256 data points and repeated along the second axis, as the CNN requires a constant input shape for all channels. This approach was chosen as during training it was observed that zero padding the metal detection data would result in the model effectively ignoring this input. The data set was split into 80% for training and 20% for validation, totaling 1616 training samples and 403 validation samples, in addition to the test set containing 40 samples. All data from each measurement was fed to the network as separate channels, where the data presented in Figure 4 results in 9 channels (two condenser- and contact microphones for each spectrogram, and metal detector data).

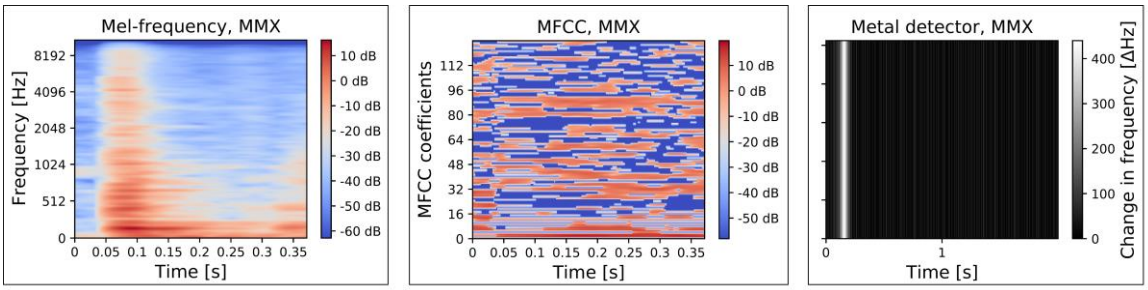



*Figure 4: Examples of a Mel spectrogram, MFCC, and the transformed metal detector data for a sample from the metal and mixed waste category, used as input for the CNN.*

Three CNN models having different numbers of layers and model parameters were tested. Model 1 (M-1) is a simple model having five convolutional layers, one fully connected layer and uniformly sized kernels (3x3). Model 2 (M-2) extends the former with two additional convolutional and fully connected layers while also having descending kernel sizes. Model 3 (M-3) is between M-1 and M-2 in layer count but uses a rectangular kernel (4x8) in the first layer. M-1 was the subject for the ablation study. Additionally, several convolutional layers were tested for M-1, where it was found that five layers was optimal for this data set. Three labelling schemes were considered (multi-class, multi-label, and binary classification), where multi-labeling was chosen due to its ability to consider each material independently, regardless of their combination in each sample.

To evaluate and discuss the classification performance, we use standard metrics commonly used to evaluate ML-models, namely accuracy, precision, and recall. Receiver operating characteristics (ROC) curves are also included which offers a simple representation of the classification thresholds and their effect on true positive- and false positive rates. The influence of the number of training samples is presented in the form of learning curves for each data set, which can be analyzed to detect the degree of bias and variance in the models.

# 4 Results

## 4.1 Ablation study with input data

Results of the ablation study are shown in Table 1. Initially, every input data was included as a reference. Weight was excluded in the first test, resulting in a better performing model. The next test (no metal) was then compared to the model trained on all the data except weight. Excluding the metal detection data did not yield a significant difference in accuracy for the validation set, although the accuracy was substantially lower for the test set. Due to the significant reduction in accuracy for the test set, metal detection data was included

in both cases going forward. Using only one of the two microphones was compared in the third test, resulting in a contradiction between the data sets. For the validation set, using only the condenser microphone gave a significant reduction in accuracy, while the test set indicates an even more significant increase in accuracy. The difference is arguably small for the validation set, while an increase of 2.25% in accuracy is beneficial for the test set. Using only the condenser microphone was therefore decided as favorable for our model. A reduction in accuracy when only using Mel spectrograms or MFCC was observed for the validation set, with no significant increase in accuracy using only the Mel spectrogram for the test set, thus showing the benefit of including both.

*Table 1: Results from the ablation study showing the change in model performance based on input data, on both the validation (valid) and test set. Every positive result is carried over to the next test, showing the cumulative change in score. Significant results are shown in bold.*

| Test | Input data | Number of runs | Average score change (valid) | P(T<=t) | Average score change (test) | P(T<=t) |
|---|---|---|---|---|---|---|
| 0 | All data | 20 | 0.00% | Reference | 0.00% | Reference |
| 1 | No weight | 20 | **+0.70%** | 0.0096 | **+2.19%** | 0.0294 |
| 2 | No metal | 20 | -0.05% | 0.3449 | **-16.06%** | < 0.0001 |
| 3 | Condenser microphone | 20 | **-0.63%** | 0.0031 | **+2.25%** | 0.0025 |
| | Contact microphone | 20 | -0.42% | 0.0550 | -2.00% | 0.0105 |
| 4 | Mel | 20 | **-1.84%** | < 0.0001 | +0.50% | 0.4794 |
| | MFCC | 20 | **-2.54%** | < 0.0001 | **-17.00%** | < 0.0001 |

## 4.2 Results on final sensors setup

M-1, M-2 and M-3 were trained using the input data resulting in the highest increase in accuracy from the ablation study. For each model, training was performed 20 times. Small variations were observed between training iterations, and the best results are presented in Table 4. All results are based on a 0.5 threshold (or confidence) for counting a prediction as valid. M-1 achieves the highest validation accuracy, which is able to correctly predict the presence of metal with 100% accuracy, and 96.28% for glass. Inference is also included for M-1 on the test set, showing a slightly lower performance compared to the validation set. The results on the test data are shown at the bottom of Table 4.

*Table 2: Accuracy (A), precision (P) and recall (R) for each model on the validation set, with the same metrics for the best performing model on the test set.*

| | | All | | | Metal | | | Glass | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Model | Data set | A | P | R | A | P | R | A | P | R |
| **M-1** | Valid | **98.14** | 99.49 | 96.77 | 100.00 | 100.00 | 100.00 | 96.28 | 98.95 | 93.56 |
| M-2 | | 97.52 | 97.28 | 97.77 | 99.75 | 100.00 | 99.50 | 95.29 | 94.63 | 96.04 |
| M-3 | | 97.02 | 96.56 | 97.52 | 99.75 | 100.00 | 99.50 | 94.29 | 93.24 | 95.54 |
| M-1 | Test | 96.25 | 95.24 | 97.50 | 100.00 | 100.00 | 100.00 | 92.50 | 90.48 | 95.00 |

Figure 8 shows the confusion matrices for M-1 on the validation and test set. It clearly shows that glass has the highest number of false positives, while metal is predicted correctly for every sample. Glass has more false positives than false negatives on the validation set, indicating a slight bias towards detecting glass.

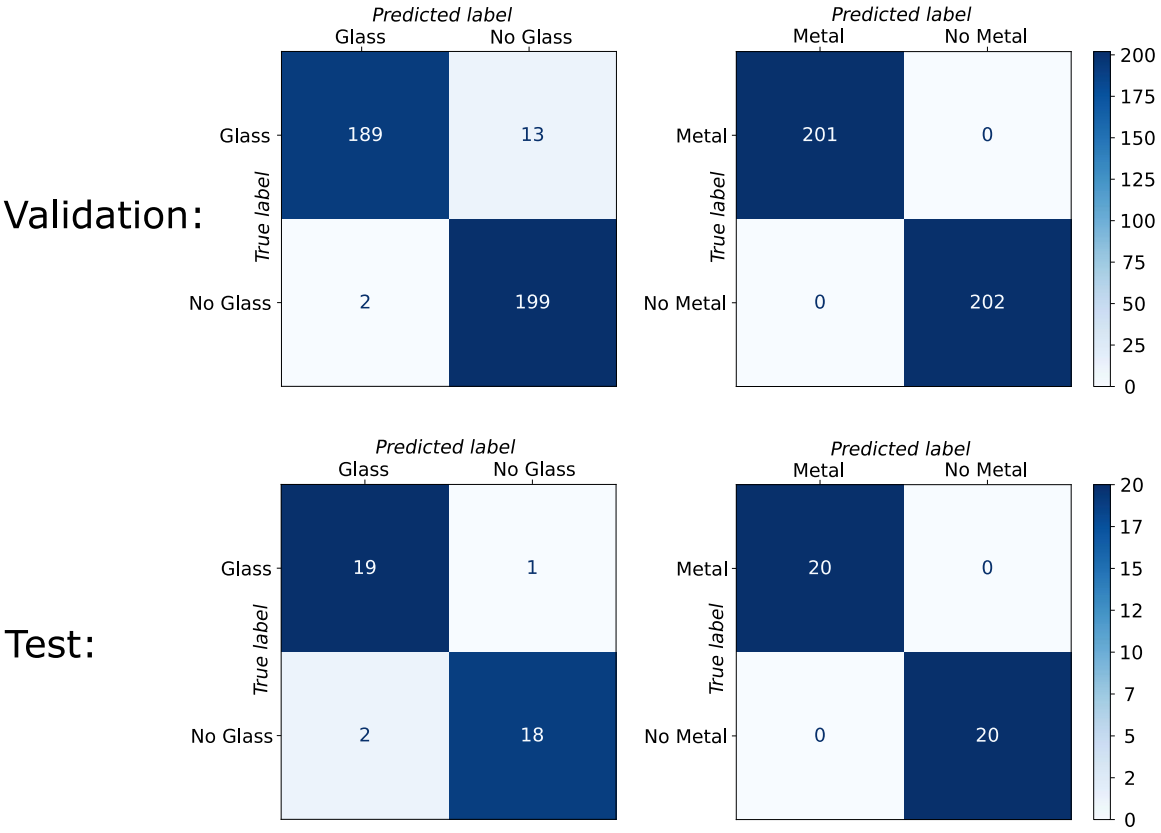



*Figure 5: Confusion matrices for glass and metal on each data set.*

Figure 6 shows the receiver operating characteristics for M-1 on both the validation- and test set. The AUCs being close to 100% shows there is little trade-off between the true positive- and false positive rate when tuning the threshold, and a high recall can be achieved without sacrificing much precision.

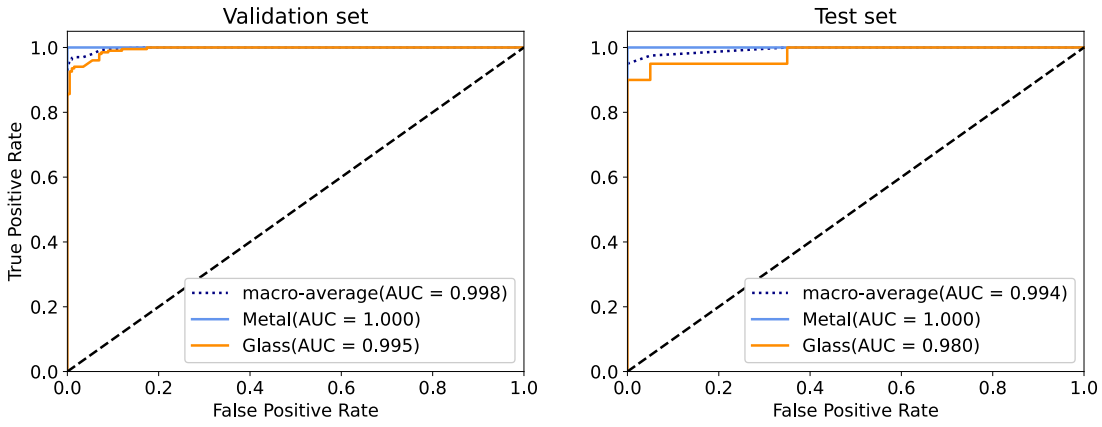



*Figure 6: ROC curves for the validation- and test set.*

The learning curves in Figure 7 seems to converge after including more than 8 samples per category when training the model. With each data set having small loss values and showing convergence, few samples are needed to train the model while showing low levels of bias and variance. Consequently, adding more training

data would not improve the model, with the optimal model performance being achieved using only 16 samples per category (64 samples in total).

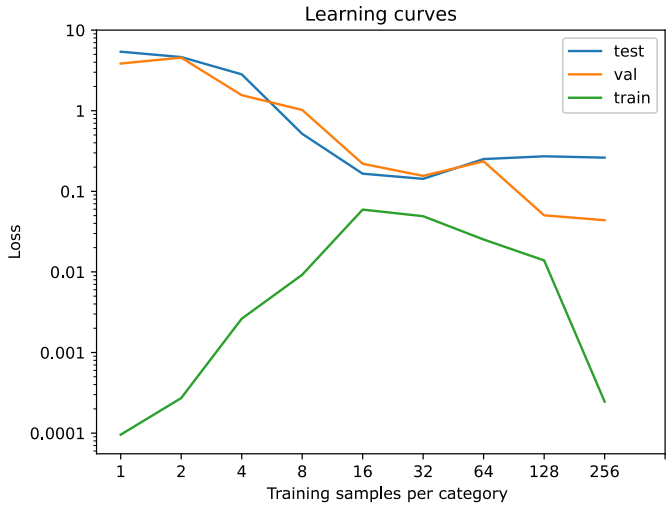



*Figure 7: Learning curves where the loss for each data set is calculated for each increment of the number of training samples used.*

# 5 Discussion

The results have shown great potential for applying CNNs to classify the presence of glass and metal in trash bags using sound and metal detection. While Korucu et al. (2016) demonstrated the applicability of using sound for detecting materials in a reverse vending machine, our approach takes it a step further by detecting materials mixed inside trash bags. With our aim to support current waste management systems relying on consumer sorting, without proposing radical changes to the status quo, the approach seems feasible based on the results.

The ablation study consolidated the benefit of using both Mel and MFCC spectrograms. We also found that using only condenser microphones, and not contact microphones, can improve the detection, which is desired since less hardware will be required to reduce cost in a potential future implementation. However, given that this is a proof-of-concept, we would still recommend testing both microphones in a realistic setting as our data set can be biased from the controlled experiment. A commercial metal detector should also be tested in a real context, where a longer range and better robustness may be required. Nonetheless, the use of metal detection data had less effect when running the models on the validation set but had a significant contribution to improving the accuracy on the test set. Event though metal was detected with 100% accuracy, making it possible to detect metal using only a simple threshold-based algorithm, it may be an important input for ML-models when detecting finer details, such as metal lids on glass jars, which were not accounted for in our data set. ML-models may also be able to differentiate between low and high metal or glass content in bags, providing more information about the consumer sorting behavior.

The model shows slightly better accuracy for the validation set, which contains bags from the same distribution as the training data where bags were reused, compared to the test set containing only unique bags. It is possible that reusing bags with random orientation have less variation than expected, causing the models to overfit on the bags used for training and validation. It is also a possibility that the test set contains slightly different features due to having been captured during a different time and location, which may have affected the sensor readings. It can be valuable to analyze the variance between the sensor readings of trash bags when using different orientations and different environments, to better understand if either approach can or should be used in the future. The benefit of reusing bags is to capture more data faster, as making unique bags is a cumbersome and time-consuming task. However, based on the learning curves in Figure 7, few training samples may be needed, and capturing samples with large variation is more important. When approaching a real-world application, we suggest collecting trash bags from consumers to build the data sets. It is then, for

example, possible to pass the collected bags through the experimental rig or a collection truck before analyzing the bags manually to determine the correct label for the data, thereby producing an even more realistic data set for training a classifier. To reduce the manual labor, an image-based classification system could also be implemented for this purpose.

Adjusting the classification threshold usually results in either an increase of the precision and reduction of the recall, or vice versa. According to the ROC plots in Figure 6, this trade-off is miniscule for our models, however, is likely to be substantially larger (smaller AUC) for real-world application. Which metric to favor depends on how the results will be used. If the goal is to plan on initiating pecuniary measures based on who sorts poorly, then perhaps avoiding miscarriage of justice (false positives) is essential, thereby maximizing precision. If the measures are, for instance, to distribute more information in areas of poor sorting, then including as many of the cases as possible may be of more interest, thereby maximizing recall.

An important factor to consider in a real-world setting is the ratio between correctly sorted and wrongly sorted bags, which is about 20:1 according to statistics from REN (J. Almankaas & J. H. Ellefsen-Killerud, personal communication, 4 January 2020). Using the Bayes' theorem in the case of detecting glass using the results from the test set, where we want to calculate the likelihood of finding a true positive (A) given that the model has classified the bag as positive (B), P(A|B), where we know that the probability of classifying a positive given a true positive, P(B|A), is the precision (90.48%), and the probability of finding a true positive P(A) is 1/21, with a false positive rate of 0.1, we get 31.15% likelihood that a classified positive is a true positive. In other words, we can expect 2.2 false positives for every true positive. If multiple areas of trash collection over time is considered, this is an acceptable result as the positive detections are likely to accumulate at the locations where bad sorting occurs more often, thereby increasing the likelihood of locating true positives.

The data sets used in this study was captured under highly controlled circumstances with one bag at a time. It is unclear how a model would perform on data from the actual collection routine. One possible problem relating to the collection truck is that many bags are emptied at once, which may increase the classification difficulty. If this proves a problem, a possible solution might be to include some sort of funnel on the collection truck which forces the bags to fall through one at a time, or to simply train the model with multiple bags at a time.

While the method is mainly developed for use in a collection truck, there are other areas where implementation would be easy and effective. For example, it could support the sorting facility in separating bags based on their content, although information about the source of bad sorting behavior will be lost at this stage. Also, many condominium buildings have shared trash disposal units, often hidden underground, where the chute provides an excellent location for installing sensors. These systems often have RFID in place, enabling a link between the consumer and their sorting behavior to be made.

It is no doubt that capturing data on consumers' sorting behaviors is beneficial to most waste management systems as it enables fact-based decision making. Most households in Oslo have a distance less than 300 meters to the nearest collection point, although in some cases it is further. Together with our approach it is possible to optimize the location of collection points to benefit consumers, as convenience is an important factor (Bernstad, 2014; Rousta et al., 2017). Some waste management systems also utilizes mobile recycling stations that accepts, among others, metal and e-waste, which are regularly moved. Data on consumer sorting may help to choose optimal routines and placements for the mobile units.

If successfully deployed, our approach may also support research on consumer waste sorting behavior. In general, data captured with our system may contribute to studies attempting to predict municipal solid waste generation (Adamović et al., 2018; Kannangara et al., 2018; Wu et al., 2020) as it may provide accurate location specific data. Rousta and Ekström (2013) studied the environmental, economic, and social aspects of incorrect sorting, however, concluded that future research should be conducted to uncover the driving factors of consumer sorting, for which our method may be useful. Bernstad (2014) argued that research on factors influencing participation in waste sorting has been largely inconsistent, and that the effects of promotional campaigns is often unknown due to a lack of proper monitoring. Our approach enables long-term monitoring, which can be used to find the most sustainable methods for improving consumer sorting.

# 6 Conclusion

With the increasing focus on preserving the global environment, necessary targets have been set for increasing the recycling of household waste within the next few years. Consumer sorting is widely deployed as the first step in a sustainable waste management system. Since the degree of correct sorting is not known during the normal collection process, we aim to support waste management systems in understanding consumer waste generation and behavior by enabling trash bags to be classified during this process. The proof-of-concept system is able to identify the occurrence of glass and metal in consumer trash bags with high accuracy. With an RFID system in place for the collection of municipal waste, our method can help pinpoint the areas where bad sorting occurs without having to invest in considerable changes to the current system.

The classification system comprising a combination of sound recording and a beat-frequency oscillation metal detector has been developed successfully. The trained CNN model is able to identify the occurrence of glass and metal in trash bags with 98% accuracy. Considering the experimental nature of the study, along with the high accuracies achieved with relatively small amounts of data required, the potential for applying this method in the real world is promising.

For future research, we suggest collecting more realistic data sets of consumer trash bags for training a CNN model. The use of sound recorders and metal detection has shown promising results and should be tested in more realistic settings. Enabling long-term monitoring of consumer waste sorting quality will also benefit research on consumer behavior. For waste management systems, fact-based decision making can be realized using our approach, to for example optimize the location of collection points to increase convenience for consumers.


## Acknowledgments

We would like to thank Renovasjonsetaten in Oslo for providing valuable insights to their waste management system.


## References


Adamović, V. M., Antanasijević, D. Z., Ćosović, A. R., Ristić, M. Đ., & Pocajt, V. V. (2018). An artificial neural network approach for the estimation of the primary production of energy from municipal solid waste and its application to the Balkan countries. *Waste management, 78*, 955-968. doi:10.1016/j.wasman.2018.07.012

Adhithyarasanna, S., & Kaushal, V. (2018). Survey on identification and classification of waste for efficient disposal and recycling. *International Journal of Engineering & Technology, 7*(2.8), 520-523. doi:10.14419/ijet.v7i2.8.10513

Ananthi, S., & Dhanalakshmi, P. (2015). *SVM and HMM modeling techniques for speech recognition using LPCC and MFCC features.* Paper presented at the Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014.

Avfallsanalysen 2019. (2019). Retrieved May 23, 2020, from https://www.oslo.kommune.no/avfall-og-gjenvinning/hvordan-kildesortere-i-oslo/avfallsanalysen/#gref.

Bernstad, A. (2014). Household food waste separation behavior and the importance of convenience. *Waste management, 34*(7), 1317-1323.

Bircanoğlu, C., Atay, M., Beşer, F., Genç, Ö., & Kızrak, M. A. (2018). *RecycleNet: Intelligent waste sorting using deep neural networks.* Paper presented at the 2018 Innovations in Intelligent Systems and Applications (INISTA).

Çakır, E., Parascandolo, G., Heittola, T., Huttunen, H., & Virtanen, T. (2017). Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection. *IEEE/ACM Transactions on Audio, Speech, and Language Processing, 25*(6), 1291-1303. doi:10.1109/TASLP.2017.2690575

Chapman, D. M. F. (2000). Decibels, SI units, and standards. *The Journal of the Acoustical Society of America, 108*(2), 480-480. doi:10.1121/1.429620

Chu, Y., Huang, C., Xie, X., Tan, B., Kamal, S., & Xiong, X. (2018). Multilayer hybrid deep-learning method for waste classification and recycling. *Computational Intelligence and Neuroscience, 2018*.

Deng, L., & Yu, D. (2014). Deep Learning: Methods and Applications. *Foundations and Trends in Signal Processing, 7*(3–4), 197–387. doi:10.1561/2000000039

Giordano, B. L., & McAdams, S. (2006). Material identification of real impact sounds: Effects of size variation in steel, glass, wood, and plexiglass plates. *The Journal of the Acoustical Society of America, 119*(2), 1171. doi:10.1121/1.2149839

Gong, T., Cho, H., Lee, B., & Lee, S.-J. (2019). Knocker: Vibroacoustic-based Object Recognition with Smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 3*(3), 1-21. doi:10.1145/3351240

Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, R. C., . . . Wilson, K. (2017, March 2017). *CNN architectures for large-scale audio classification.* Paper presented at the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

Jiang, P., Fan, Y. V., Zhou, J., Zheng, M., Liu, X., & Klemeš, J. J. (2020). Data-driven analytical framework for waste-dumping behaviour analysis to facilitate policy regulations. *Waste management, 103*, 285-295. doi:10.1016/j.wasman.2019.12.041

Kannangara, M., Dua, R., Ahmadi, L., & Bensebaa, F. (2018). Modeling and prediction of regional municipal solid waste generation and diversion in Canada using machine learning approaches. *Waste management, 74*, 3-15. doi:10.1016/j.wasman.2017.11.057

Khamparia, A., Gupta, D., Nhu, N., Khanna, A., Pandey, B., & Tiwari, P. (2019). Sound Classification Using Convolutional Neural Network and Tensor Deep Stacking Network. *IEEE Access, PP*, 1-1. doi:10.1109/ACCESS.2018.2888882

Korucu, M. K., Kaplan, Ö., Büyük, O., & Güllü, M. K. (2016). An investigation of the usability of sound recognition for source separation of packaging wastes in reverse vending machines. *Waste management, 56*, 46-52.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *Imagenet classification with deep convolutional neural networks.* Paper presented at the Advances in neural information processing systems.

Kumar, A., & Raj, B. (2017). Deep CNN Framework for Audio Event Recognition using Weakly Labeled Web Data. *arXiv:1707.02530 [cs]*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324.

Li, J., Dai, W., Metze, F., Qu, S., & Das, S. (2017, March 2017). *A comparison of Deep Learning methods for environmental sound detection.* Paper presented at the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

Lindermayr, J., Senst, C., Hoang, M.-H., & Haegele, M. (2018). *Visual Classification of Single Waste Items in Roadside Application Scenarios for Waste Separation.* Paper presented at the 50th International Symposium on Robotics (ISR 2018).

McLoughlin, I., Zhang, H., Xie, Z., Song, Y., & Xiao, W. (2015). Robust Sound Event Classification Using Deep Neural Networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing, 23*(3), 540-552. doi:10.1109/TASLP.2015.2389618

Mesaros, A., Heittola, T., Eronen, A., & Virtanen, T. (2010, August 2010). *Acoustic event detection in real life recordings.* Paper presented at the 2010 18th European Signal Processing Conference.

Noll, A. M. (1967). Cepstrum Pitch Determination. *The Journal of the Acoustical Society of America, 41*(2), 293–309. doi:10.1121/1.1910339

Nowakowski, P., & Pamuła, T. (2020). Application of deep learning object classifier to improve e-waste collection planning. *Waste management, 109*, 1-9.

Rada, E. C., Ragazzi, M., & Fedrizzi, P. (2013). Web-GIS oriented systems viability for municipal solid waste selective collection optimization in developed and transient economies. *Waste management, 33*(4), 785-792.

Rousta, K., & Ekström, K. M. (2013). Assessing incorrect household waste sorting in a medium-sized Swedish city. *Sustainability, 5*(10), 4349-4361.

Rousta, K., Ordoñez, I., Bolton, K., & Dahlén, L. (2017). Support for designing waste sorting systems: A mini review. *Waste Management & Research, 35*(11), 1099-1111.

Ruiz, V., Sánchez, Á., Vélez, J. F., & Raducanu, B. (2019). *Automatic image-based waste classification.* Paper presented at the International Work-Conference on the Interplay Between Natural and Artificial Computation.

Sharan, R. V., & Moir, T. J. (2016). An overview of applications and advancements in automatic sound recognition. *Neurocomputing, 200*, 22–34. doi:10.1016/j.neucom.2016.03.020

Toğaçar, M., Ergen, B., & Cömert, Z. (2020). Waste classification using AutoEncoder network with integrated feature selection method in convolutional neural network models. *Measurement, 153*, 107459.

Volkmann, J., Stevens, S. S., & Newman, E. B. (1937). A Scale for the Measurement of the Psychological Magnitude Pitch. *The Journal of the Acoustical Society of America, 8*(3), 208–208. doi:10.1121/1.1915893

Wilsgaard, S. (2018). Europa har fått nye avfallsdirektiv. Retrieved May 12, 2019, from https://www.avfallnorge.no/bransjen/nyheter/europa-har-f%C3%A5tt-nye-avfallsdirektiv.

Wu, F., Niu, D., Dai, S., & Wu, B. (2020). New insights into regional differences of the predictions of municipal solid waste generation rates using artificial neural networks. *Waste management, 107*, 182-190. doi:10.1016/j.wasman.2020.04.015

Yang, M., & Thung, G. (2016). Classification of trash for recyclability status. *CS229 Project Report, 2016*.

Zhang, H., McLoughlin, I., & Song, Y. (2015, April 2015). *Robust sound event recognition using convolutional neural networks.* Paper presented at the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

# 8 Conclusion

Through a review of relevant material and several interviews with employees at REN, one of their main roadblocks regarding their waste collection cycle has been discovered: not enough is known about the recycling habits of specific households in Oslo and little is known about the source of wrongly sorted material. This makes it challenging to introduce effective location specific measures. It is argued that with more data, as well as higher quality data able to pin point which bin a wrongly sorted material originates from, REN will be able to take more effective action against unsatisfactory recycling behavior.

In this Master's Thesis, an attempt have been made to develop a CNN-based glass-and-metal detection system using sensors recommended by the preceding Project Thesis A.3. These include microphone, weight scale, metal detector and camera. The system was meant to be installed on a waste collection truck, such that the detection of glass or metal can be linked to a location. Due to insufficient availability of a trash collection truck, a simulation of the emptying of trash bin was made locally such that a model could be trained on data resembling the real situation. The results show that detecting glass and metal using a combination of sound and metal detection is an effective approach in the classification of singular trash bags. With an accuracy of 98.14% in the validation set and 96.25% in the test set, the model developed is able to detect the presence of glass and/or metal with high precision in a considerable amount of trash bags. This does not conclusively prove the feasibility of such a model on a truck where several bags are emptied simultaneously, but provide support for further research and testing. A conclusive test will require testing and operation of the system in the actual implementation location.

The developed system is not yet fully implementable on a service truck as the CNN model must be trained on data from actual emptying of trash bins in the final implementation location. However, great efforts has been put in to enabling a replication of the experiment on a truck, and the results show promise for the system being able to function as intended.

# References


[1] Oppdatert: 20.05.2020. *Kildesortering — Stavanger kommune*. no. Library Catalog: www.stavanger.kommune.no. URL: https://www.stavanger.kommune.no/renovasjon-og-miljo/kildesortering/ (visited on 05/26/2020).

[2] BIR A/S. *Glass- og metallemballasje - BIR*. no. Library Catalog: bir.no. URL: https://bir.no/slik-sorterer-du/glass-og-metallemballasje/ (visited on 05/26/2020).

[3] Olugboja Adedeji and Zenghui Wang. "Intelligent Waste Classification System Using Deep Learning Convolutional Neural Network". en. In: *Procedia Manufacturing*. The 2nd International Conference on Sustainable Materials Processing and Manufacturing, SMPM 2019, 8-10 March 2019, Sun City, South Africa 35 (Jan. 2019), pp. 607–612. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2019.05.086. URL: http://www.sciencedirect.com/science/article/pii/S2351978919307231 (visited on 05/14/2020).

[4] Jørgen Almankaas and Jan Haakon Ellefsen-Killerud. *Interview With Employees at REN*. Norwegian. Jan. 2020.

[5] Md Zahangir Alom et al. "A State-of-the-Art Survey on Deep Learning Theory and Architectures". eng. In: *Electronics* 8.3 (2019). Publisher: MDPI AG, pp. 292–. ISSN: 2079-9292. DOI: 10.3390/electronics8030292. URL: https://doaj.org/article/2fc4ff8667364b7899b508a21a30d572 (visited on 06/27/2020).

[6] Carla Balocco et al. "Experimental transmittance of polyethylene films in the solar and infrared wavelengths". eng. In: *Solar Energy* 165 (2018). Publisher: Elsevier Ltd, pp. 199–205. ISSN: 0038-092X. DOI: 10.1016/j.solener.2018.03.011.

[7] Emre Çakır et al. "Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.6 (June 2017). Conference Name: IEEE/ACM Transactions on Audio, Speech, and Language Processing, pp. 1291–1303. ISSN: 2329-9304. DOI: 10.1109/TASLP.2017.2690575.

[8] Yinghao Chu et al. "Multilayer Hybrid Deep-Learning Method for Waste Classification and Recycling". en. In: *Computational Intelligence and Neuroscience* 2018 (Nov. 2018), pp. 1–9. ISSN: 1687-5265, 1687-5273. DOI: 10.1155/2018/5060857. URL: https://www.hindawi.com/journals/cin/2018/5060857/ (visited on 05/24/2020).

[9] Li Deng and Dong Yu. "Deep Learning: Methods and Applications". In: *Foundations and Trends in Signal Processing* 7.3–4 (June 2014), pp. 197–387. ISSN: 1932-8346. DOI: 10.1561/2000000039. URL: https://doi.org/10.1561/2000000039 (visited on 06/09/2020).

[10] Google Developers. *Introduction to TensorFlow — Machine Learning Crash Course*. en. Library Catalog: developers.google.com. 2020. URL: https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit (visited on 06/01/2020).

[11] Juan Du. "Understanding of Object Detection Based on CNN Family and YOLO". en. In: *Journal of Physics: Conference Series* 1004 (Apr. 2018), p. 012029. ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/1004/1/012029. URL: http://stacks.iop.org/1742-6596/1004/i=1/a=012029?key=crossref.aef5af21bb1bf8edebfb4bd94c9cf5ff (visited on 12/03/2019).

[12] Dalya Gartzman. *Getting to Know the Mel Spectrogram*. en. Library Catalog: towardsdatascience.com. Jan. 2020. URL: https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0 (visited on 04/29/2020).

[13] Taesik Gong et al. "Knocker: Vibroacoustic-based Object Recognition with Smartphones". en. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3.3 (Sept. 2019), pp. 1–21. ISSN: 2474-9567, 2474-9567. DOI: 10.1145/3351240. URL: https://dl.acm.org/doi/10.1145/3351240 (visited on 05/21/2020).

[14] Google. *TensorFlow*. en. Library Catalog: www.tensorflow.org. 2020. URL: https://www.tensorflow.org/?hl=nb (visited on 06/04/2020).

[15] Sathish Paulraj Gundupalli, Subrata Hait, and Atul Thakur. "Multi-material classification of dry recyclables from municipal solid waste based on thermal imaging". eng. In: *Waste Management* 70 (2017). Publisher: Elsevier Ltd, pp. 13–21. ISSN: 0956-053X. DOI: 10.1016/j.wasman.2017.09.019.

[16] Kaiming He et al. "Mask R-CNN". eng. In: (2017). URL: https://arxiv.org/abs/1703.06870 (visited on 06/28/2020).

[17] Shawn Hershey et al. "CNN architectures for large-scale audio classification". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ISSN: 2379-190X. Mar. 2017, pp. 131–135. DOI: 10.1109/ICASSP.2017.7952132.

[18] Evan Kale. *Build Your Own Metal Detector with an Arduino - Projects*. en. URL: https://www.allaboutcircuits.com/projects/metal-detector-with-arduino/ (visited on 03/30/2020).

[19] M. Kemal Korucu et al. "An investigation of the usability of sound recognition for source separation of packaging wastes in reverse vending machines.(Report)". eng. In: *Waste Management* 56 (2016). Publisher: Elsevier BV, pp. 46–52. ISSN: 0956-053X. DOI: 10.1016/j.wasman.2016.06.030.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet classification with deep convolutional neural networks". eng. In: *Communications of the ACM* 60.6 (2017). Publisher: ACM, pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (visited on 06/01/2020).

[22] Anurag Kumar and Bhiksha Raj. "Deep CNN Framework for Audio Event Recognition using Weakly Labeled Web Data". In: *arXiv:1707.02530 [cs]* (July 2017). arXiv: 1707.02530. URL: http://arxiv.org/abs/1707.02530 (visited on 06/09/2020).

[23] Y. Lecun et al. "Gradient-based learning applied to document recognition". eng. In: *Proceedings of the IEEE* 86.11 (1998). Publisher: IEEE, pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

[24] Juncheng Li et al. "A comparison of Deep Learning methods for environmental sound detection". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ISSN: 2379-190X. Mar. 2017, pp. 126–130. DOI: 10.1109/ICASSP.2017.7952131.

[25] Shan Luo et al. "Knock-Knock: Acoustic object recognition by using stacked denoising autoencoders". eng. In: *Neurocomputing* 267.C (2017). Publisher: Elsevier BV, pp. 18–24. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.03.014.

[26] Ian McLoughlin et al. "Robust Sound Event Classification Using Deep Neural Networks". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23.3 (Mar. 2015). Conference Name: IEEE/ACM Transactions on Audio, Speech, and Language Processing, pp. 540–552. ISSN: 2329-9304. DOI: 10.1109/TASLP.2015.2389618.

[27] A. Michael Noll. "Cepstrum Pitch Determination". eng. In: *The Journal of the Acoustical Society of America* 41.2 (1967). Publisher: Acoustical Society of America, pp. 293–309. ISSN: 0001-4966. DOI: 10.1121/1.1910339.

[28] Avfall Norge. *Europa har fått nye avfallsdirektiv*. no. URL: https://www.avfallnorge.no/bransjen/nyheter/europa-har-f%C3%A5tt-nye-avfallsdirektiv (visited on 12/05/2019).

[29] A. V. Oppenheim and R. W. Schafer. "From frequency to quefrency: a history of the cepstrum". eng. In: *IEEE Signal Processing Magazine* 21.5 (2004). Publisher: IEEE, pp. 95–106. ISSN: 1053-5888. DOI: 10.1109/MSP.2004.1328092.

[30] E. C. Rada, M. Ragazzi, and P. Fedrizzi. "Web-GIS oriented systems viability for municipal solid waste selective collection optimization in developed and transient economies". eng. In: *Waste Management* 33.4 (2013). Publisher: Elsevier Ltd, pp. 785–792. ISSN: 0956-053X. DOI: 10.1016/j.wasman.2013.01.002.

[31] B. Visvesvara Rao. *Electronic circuit analysis*. English. OCLC: 890449125. Chennai: Pearson, 2012. ISBN: 978-81-317-5428-3.

[32] Trondheim Renholdsverk. *Slik sorterer du*. no. Library Catalog: trv.no. URL: https://trv.no/sortere/ (visited on 05/26/2020).

[33] Renovasjonsetaten. *Avfallsanalysen*. no. Library Catalog: www.oslo.kommune.no. URL: https://www.oslo.kommune.no/avfall-og-gjenvinning/hvordan-kildesortere-i-oslo/avfallsanalysen/ (visited on 05/23/2020).

[34] Arthur L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM J. Res. Dev.* (1959). DOI: 10.1147/rd.33.0210.

[35] Roneel V. Sharan and Tom J. Moir. "An overview of applications and advancements in automatic sound recognition". eng. In: *Neurocomputing* 200 (2016). Publisher: Elsevier BV, pp. 22–34. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2016.03.020.

[36] Keras SIG. *Keras: the Python deep learning API*. 2020. URL: https://keras.io/ (visited on 06/04/2020).

[37] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". eng. In: *arXiv.org* (2015). Place: Ithaca Publisher: Cornell University Library, arXivorg. ISSN: 2331-8422. URL: http://search.proquest.com/docview/2081521649/?pq-origsite=primo (visited on 06/08/2020).

[38] Avfall Sør. *Glass- og metallemballasje*. nb-NO. Library Catalog: avfallsor.no. URL: https://avfallsor.no/hvor-skal-dette/glass-og-metallemballasje/ (visited on 05/26/2020).

[39] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". en. In: (), p. 30.

[40] J. Volkmann, S. S. Stevens, and E. B. Newman. "A Scale for the Measurement of the Psychological Magnitude Pitch". eng. In: *The Journal of the Acoustical Society of America* 8.3 (1937). Publisher: Acoustical Society of America, pp. 208–208. ISSN: 0001-4966. DOI: 10.1121/1.1915893.

[41] Tadeusz A. Wysocki et al. *Advanced Signal Processing for Communication Systems*. eng. Vol. 703. International series in engineering and computer science ; Boston, MA: Springer US, 2002. ISBN: 978-0-306-47791-1.

[42] Mindy Yang and Gary Thung. "Classification of trash for recyclability status". In: *CS229 Project Report* 2016 (2016).

[43] Haomin Zhang, Ian McLoughlin, and Yan Song. “Robust sound event recognition using convolutional neural networks”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ISSN: 2379-190X. Apr. 2015, pp. 559–563. DOI: 10.1109/ICASSP.2015.7178031.

[44] Yan Zheng et al. “A discrimination model in waste plastics sorting using NIR hyperspectral imaging system”. eng. In: *Waste Management* 72 (2018). Publisher: Elsevier Ltd, pp. 87–98. ISSN: 0956-053X. DOI: 10.1016/j.wasman.2017.10.015.

[45] 熙 古井. “An Overview of Speaker Recognition Technology”. eng. In: 1994, pp. 1–9. URL: http://t2r2.star.titech.ac.jp/cgi-bin/publicationinfo.cgi?q_publication_content_number=CTT100589789 (visited on 06/01/2020).

# A Appendix

## A.1 Trash Collection Experiment

### A.1.1 Arduino Code

```
#include "HX711.h"

#define DOUT  6  // Arduino pin 6 connect to HX711 DOUT
#define CLK  3  //  Arduino pin 5 connect to HX711 CLK

HX711 scale;

int proximityPin = 4;
bool weight_reg = false;

double weight = 0;
double current_weight = 0;
int datapoints = 600; // Number of metal detector
    readings

bool metal_sent = false;

// Number of cycles from external counter needed to
    generate a signal event
#define CYCLES_PER_SIGNAL 1000

unsigned long lastSignalTime = 0;
unsigned long signalTimeDelta = 0;

```

```
23 boolean firstSignal = true;
24 unsigned long storedTimeDelta = 0;
25 double discrepancy = 1000;
26
27 int sensorPin = A4;
28 int val;
29
30 SIGNAL(TIMER1_COMPA_vect)
31 {
32   unsigned long currentTime = micros();
33   signalTimeDelta =  currentTime - lastSignalTime;
34   lastSignalTime = currentTime;
35
36   if (firstSignal)
37   {
38     firstSignal = false;
39   }
40   else if (storedTimeDelta == 0)
41   {
42     storedTimeDelta = signalTimeDelta;
43   }
44
45   // Reset OCR1A
46   OCR1A += CYCLES_PER_SIGNAL;
47 }
48
49
50 void setup() {
51   Serial.begin(9600);
52     TCCR1A = 0b00000000;
53
54   // Set CSS(Clock Speed Selection) to 0b111 (External
       clock source on T1 pin
55   // (ie, pin 5 on UNO). Clock on rising edge.)
56   TCCR1B = 0b00000111;
57
58   // Enable timer compare interrupt A (ie, SIGNAL(
       TIMER1_COMPA_VECT))
59   TIMSK1 |= (1 << OCIE1A);
60
61   // Set OCR1A (timer A counter) to 1 to trigger
       interrupt on next cycle
62   OCR1A = 1;
63
64   pinMode(proximityPin,INPUT);
```

```
65   scale.begin(DOUT,CLK);  // initialize the scale using
        pin DOUT and CLK defined above
66   scale.set_scale(244);  // Start scale, for some reason
        1000 amounts to grams
67   scale.tare();       // Reset scale to zero
68
69
70
71 }
72
73 void loop() {
74   if(Serial.available()){
75     char decider = Serial.read();//Arduino reads the
          character received from computer
76
77
78     if (decider == 'w'){ //Initializes weight reading
79         while(discrepancy>5 || weight<3.00){//If
              discrepancy between four consectuvie readings
              is low enough, register weight
80         for(int i = 0;i<4;i++){
81           current_weight=(double)scale.get_units(10);
82           discrepancy = current_weight - weight;
83           weight = current_weight;
84           delay(100);
85         }
86         Serial.println(weight);//Sends weight data to
              computer
87     }
88
89
90     if(decider == 'p'){//Initializes proximity sensor
          reading followed by metal detection reading
91           while(val <200){ //A while loop halts the
                script untill proximity sensor is triggered
92             val = 0;
93             for(int i = 0;i<5;i++){
94                 val += analogRead(sensorPin);
95              }
96              val = val/5;//Finds the average of five
                   sensor readings
97           }
98
99           double metalValue;
100          Serial.println("Record"); //Tells computer to
                record sound data as proximity has been
```

```
                triggered and Arduino is ready to record
                metal data
101           for(int i = 0; i<datapoints; i++){
102              metalValue = (storedTimeDelta -
                     signalTimeDelta) * 10; \\ Save difference
                      between stored and measured value
103              if (metalValue>10000){
104                 metalValue = 0;
105              }
106              Serial.println(metalValue);//Sends readings
                     to the computer continuously
107            }
108
109       Serial.println("complete");//Tells computer its
              finished sending metal detector data
110     }
111
112     else if(decider == 'a'){//Resets all variables
113       weight = 0;
114       current_weight = 0;
115       discrepancy = 1000
116       val = 0;
117       storedTimeDelta = 0;
118       decider;
119       scale.begin(DOUT,CLK);
120       scale.tare();
121
122       for(int i = 0; i<datapoints; i++){
123           metal_detection_data[i] = 0.0;
124       }
125
126     }
127   }
128 }
```

### A.1.2 GUI Code

```
1
2 import atexit
3 import tkinter as tk
4 from tkinter import filedialog
5 from tkinter import *
6 import time
7 import itertools
8 import serial
9 import numpy as np
10 from matplotlib.backends.backend_tkagg import (
11     FigureCanvasTkAgg, NavigationToolbar2Tk)
12
```

```
13 from matplotlib.backend_bases import key_press_handler
14
15 from matplotlib.figure import Figure
16 import matplotlib.pyplot as plt
17
18 from tkinter import Canvas
19
20 import os
21 import threading
22
23 import pyaudio
24 import wave
25 import librosa
26 import librosa.display
27
28 from goprocam import GoProCamera
29 from goprocam import constants
30
31 import serial.tools.list_ports
32
33 from moviepy.video.io.ffmpeg_tools import ffmpeg_extract_subclip
34
35
36
37 class App(tk.Tk):
38
39     def __init__(self):
40         tk.Tk.__init__(self)
41         self._frame = None
42         self.geometry('1000x700')
43         self.show_frame1()
44         self.winfo_toplevel().title("Trash Collection GUI")
45
46     def show_frame1(self):
47
48         new_frame = Page1(self)
49         if self._frame is not None:
50             self._frame.destroy()
51         self._frame = new_frame
52         self._frame.place(relx=0, rely=0)
53
54
55     def show_frame2(self, classNum, serialPort, dataDir):
56
57         try:
58             self._frame.storageLocFile.write(dataDir)
59             self._frame.storageLocFile.close()
60         except:
61             pass
62         new_frame = Page2(self, (0,0),serialPort, dataDir)
63         if self._frame is not None:
64             self._frame.destroy()
65         self._frame = new_frame
66         self._frame.place(relx=0, rely=0)
67         self._frame.updater()
68
69     def exit(self):
```

```
70          try:
71              self._frame.ser.close()
72          except AttributeError:
73              pass
74
75  class Page1(tk.Frame):
76      def __init__(self, master):
77          tk.Frame.__init__(self, master,width=1000,height=700)
78          self.currdir = os.getcwd()
79          self.tempdir = self.currdir
80
81          try:
82              self.storageLocFile = open(self.currdir+'/
                    StorageLocation.txt','r')
83              self.storageLoc = self.storageLocFile.read()
84              if self.storageLoc =='':
85                  raise NameError("Empty storge location file")
86          except:
87              self.storageLocFile = open(self.currdir+'/
                    StorageLocation.txt','w+')
88              self.storageLoc = ''
89
90
91
92
93          self.ports = self.serial_ports()
94
95
96
97          self.portSelection = tk.StringVar()
98          self.portSelection.set('')
99
100         self.dropdownMenu = OptionMenu(self, self.portSelection, *
                self.ports)
101
102         #labels
103         self.portlabel = Label(self, text="Select port:", fg="black
                ")
104         self.dirLabel = Label(self,text="Data directory:",fg="black
                ")
105
106         #Old label
107 ##        self.classLabel = Label(self,text="Choose class:",fg="
    black")
108
109
110         self.classNum = (0,0)
111
112         self.entry = Entry(self)
113
114
115         ##OLD##
116         # Class indexes: 1 = Pure Glass
117         #                2 = Glass and metal
118         #                3 = Glass and mixed
119         #                4 = Pure metal
120         #                5 = Metal and mixed
```

```
121         #                   6 = Pure mixed
122         #                   7 = Mixed, glass and metal
123
124
125         #Old buttons in page 1, found unecessary
126 ##          button1 = tk.Button(self, text="Pure Glass", command=
    lambda: self.set_class(0))
127 ##          button2 = tk.Button(self, text="Glass and metal", command
    =lambda: self.set_class(1))
128 ##          button3 = tk.Button(self, text="Glass and mixed", command
    =lambda: self.set_class(2))
129 ##          button4 = tk.Button(self, text="Pure metal", command=
    lambda: self.set_class(3))
130 ##          button5 = tk.Button(self, text="Metal and mixed", command
    =lambda: self.set_class(4))
131 ##          button6 = tk.Button(self, text="Pure mixed", command=
    lambda: self.set_class(5))
132 ##          button7 = tk.Button(self, text="Mixed, glass and metal",
    command=lambda: self.set_class(6))
133         button8 = tk.Button(self, text="Browse", command=self.
                get_dir)
134         button9 = tk.Button(self,text="Run",command = lambda:
                master.show_frame2(self.classNum,self.portSelection.get
                (),self.entry.get()))
135 ##
136 ##          button1.place(relx=0.4, y=100)
137 ##          button2.place(relx=0.4, y=25+100)
138 ##          button3.place(relx=0.4, y=50+100)
139 ##          button4.place(relx=0.4, y=75+100)
140 ##          button5.place(relx=0.4, y=100+100)
141 ##          button6.place(relx=0.4, y=125+100)
142 ##          button7.place(relx=0.4, y=150+100)
143
144
145
146
147         self.dropdownMenu.place(relx=0.4, y=300)
148         self.portlabel.place(relx=0.4, y=300,anchor=NE)
149         self.dirLabel.place(relx=0.4, y=350,anchor=E)
150         #self.classLabel.place(relx=0.4, y=100,anchor=NE)
151         self.entry.place(relx=0.4, y=350,anchor=W)
152         button8.place(relx=0.6, y=350,anchor=W)
153         button9.place(relx=0.4, y=400,height=100,width=100)
154
155         if self.storageLoc:
156             self.entry.insert(0,self.storageLoc)
157
158     def serial_ports(self):
159         result = []
160         ports = [comport.device for comport in serial.tools.
                list_ports.comports()]
161         for port in ports:
162             try:
163                 s = serial.Serial(port)
164                 s.close()
165                 result.append(port)
166             except (OSError, serial.SerialException):
```

```
167                 pass
168         return result
169
170     def get_dir(self):
171         self.tempdir = filedialog.askdirectory(parent=self,
                initialdir=self.currdir,title='Please select a
                directory')
172         self.entry.insert(0,self.tempdir)
173
174     def set_class(self,classNum):
175         self.classNum = classNum
176
177 class Page2(tk.Frame):
178     def __init__(self, master, classNum, serialPort, dataDir):
179         tk.Frame.__init__(self, master,width=1000,height=700)
180
181         self.classNum = classNum
182         master.bind('<Return>', self.enter) #Binds enter button to
                function self.enter
183
184
185         # paths
186         self.path_measurements = dataDir+'/Measurements'
187         self.tempPath = self.path_measurements + '/temp'
188
189         #Old class list
190 ##          self.classList = ['/GlassPure/', '/GlassAndMetal/', '/
    GlassAndMix/', '/MetalPure/', '/MetalAndMix/',
191 ##                            '/MixPure/', '/MixedGlassAndMetal/']
192
193         #New class dictionary
194         self.classList = {(1,1):'/GlassAndMetal/', (0,1):'/
                GlassAndMix/',(1,0):'/MetalAndMix/',(0,0):'/MixPure/'}
195
196         #create directories if not existing
197         if not os.path.exists(self.path_measurements):
198             os.mkdir(self.path_measurements)
199
200         for C in self.classList:
201             path = self.path_measurements+self.classList[C]
202             if not os.path.exists(path):
203                 os.mkdir(path)
204
205         if not os.path.exists(self.tempPath):
206             os.mkdir(self.tempPath)
207
208         #Previous buttons
209 ##          self.button1 = tk.Button(self, text="Pure Glass", command
    =lambda: self.define_class(0, self.button1))
210 ##          self.button2 = tk.Button(self, text="Glass and metal",
    command=lambda: self.define_class(1, self.button2, ))
211 ##          self.button3 = tk.Button(self, text="Glass and mixed",
    command=lambda: self.define_class(2, self.button3, ))
212 ##          self.button4 = tk.Button(self, text="Pure metal", command
    =lambda: self.define_class(3, self.button4))
213 ##          self.button5 = tk.Button(self, text="Metal and mixed",
    command=lambda: self.define_class(4, self.button5))
```

```
214 ##          self.button6 = tk.Button(self, text="Pure mixed", command
    =lambda: self.define_class(5, self.button6))
215 ##          self.button7 = tk.Button(self, text="Mixed, glass and
    metal",command=lambda: self.define_class(6, self.button7))
216
217         self.checkVar1 = tk.IntVar()
218         self.checkVar2 = tk.IntVar()
219
220         self.checkbutton1 = tk.Checkbutton(self, text="Metal",
                variable = self.checkVar1,onvalue = 1, offvalue=0,
                command=lambda: self.define_class(0, self.checkbutton1,
                self.checkbutton2))
221         self.checkbutton2 = tk.Checkbutton(self, text="Glass",
                variable = self.checkVar2,onvalue = 1, offvalue=0,
                command=lambda: self.define_class(1, self.checkbutton1,
                self.checkbutton2))
222
223
224         self.canva = Canvas(self)
225         self.myrectangle = self.canva.create_rectangle(30, 10, 120,
                80, outline="#fb0", fill="#fb0")
226
227         self.StorageLocation = ""
228
229         #Timing-variables
230         self.currTime = 0.000
231         self.cutIncrement = 0.000
232
233         # dybamic text and label
234         self.dynamicText = tk.StringVar()
235         self.dynamicText.set("-")
236
237         self.dynamicText2 = tk.StringVar()
238
239         self.labeltitle = Label(self, textvariable=self.dynamicText
                , fg="black",font=("Helvetica", 25))
240
241         self.labeltitle2 = Label(self, textvariable=self.
                dynamicText2, fg="black",font=("Helvetica", 25))
242
243         self.weightlabel = Label(self, text="Weight:", fg="black")
244
245         self.classlabel = Label(self, text="Class number:", fg="
                black")
246
247         self.dynamicText2.set(str(self.classNum))
248
249         # save button
250         self.displayButton = Button(self, text="Save",
251                                     command=self.save)
252
253         # quit button
254         self.quitButton = Button(self, text="QUIT", fg="red", cnf
                ={}, command=master.destroy)
255
256         #discard button
257         self.discardButton = Button(self, text = "Discard", command
```

```
            = self.discard)
258
259         # boolean variables
260         self.weight_received = False
261         self.metal_received = False
262         self.proximity_received = False
263         self.metal_recorded = False
264         self.weight_requested = False
265         self.proximity_requested = False
266         self.metal_requested = False
267         self.recorded = False
268         self.rdySave = False
269         self.update = True
270         self.vidDownloaded = True
271         self.very_received = False
272         self.vidRdy = False
273
274         # array for metal data
275         self.metal_data = []
276
277         # serial
278         self.ser = serial.Serial(serialPort, 9600)
279         time.sleep(2)
280         self.ser.reset_input_buffer()
281
282         # spectrograms
283         self.mfccs1 = np.empty(shape=(0, 0))
284         self.mfccs2 = np.empty(shape=(0, 0))
285         self.mel1 = np.empty(shape=(0, 0))
286         self.mel2 = np.empty(shape=(0, 0))
287
288         # gopro
289         self.cam = GoProCamera.GoPro(constants.gpcontrol,
                mac_address = "74-70-FD-C7-72-E3")
290
291
292         # Class indexes: 1 = Pure Glass
293         #                2 = Glass and metal
294         #                3 = Glass and mixed
295         #                4 = Pure metal
296         #                5 = Metal and mixed
297         #                6 = Pure mixed
298         #                7 = Mixed, glass and metal
299
300
301         self.t = np.empty(shape=(1, 1))
302         self.fig = Figure(figsize=(5, 4), dpi=100)
303
304         self.fig.add_subplot(111).plot(self.t)
305
306         # plot
307         self.canvas = FigureCanvasTkAgg(self.fig, master=master)  #
                A tk.DrawingArea.
308         self.canvas.draw()
309
310         # spectrogram plots
311         self.specFig = plt.Figure(figsize=(5, 4))
```

```
312         self.specCanvas = FigureCanvasTkAgg(self.specFig, master=
                master)
313         self.specCanvas.get_tk_widget().place(relx=0, rely=0.4)
314         self.canvas.get_tk_widget().place(relx=0.5,rely=0.4)
315         self.canva.place(relx=0.5,rely=0.3,anchor=N)
316 
317         self.canvas.draw()
318         self.specCanvas.draw()
319         self.packing()
320 
321 
322     def cancel(self):
323         if self._job is not None:
324             self.after_cancel(self._job)
325             self._job = None
326 
327     def packing(self):
328         #Previously button placements
329 ##        self.button1.place(x=10,y=10)
330 ##        self.button2.place(x=10,y=10+25*1)
331 ##        self.button3.place(x=10,y=10+25*2)
332 ##        self.button4.place(x=10,y=10+25*3)
333 ##        self.button5.place(x=10,y=10+25*4)
334 ##        self.button6.place(x=10,y=10+25*5)
335 ##        self.button7.place(x=10,y=10+25*6)
336 
337         self.checkbutton1.place(x=10,y=10)
338         self.checkbutton2.place(x=10,y=10+25*1)
339 
340         self.weightlabel.place(x=550,y=10,anchor=NE)
341         self.classlabel.place(x=300,y=10,anchor=NE)
342 
343         self.labeltitle.place(x=580,y=10)
344         self.labeltitle2.place(x=330,y=10)
345 
346         self.displayButton.place(x=850,y=10,anchor=N)
347         self.discardButton.place(x=850,y=100,anchor=N)
348         self.quitButton.place(x=850,y=200,anchor=N)
349 
350 
351 
352     def enter(self, event=None):
353         if self.rdySave:
354             self.save()
355 
356     def run_script(self): #Main govenrning function. Loops
            constantly until program is shut down.
357         # get the weight
358         if not self.weight_received and self.vidDownloaded:
359             self.canva.itemconfig(self.myrectangle,fill='orange')
360             if not self.weight_requested:
361                 self.ser.reset_input_buffer()
362                 self.ser.reset_output_buffer()
363                 self.ser.write("w".encode())
364                 self.weight_requested = True
365                 return
366             elif self.ser.in_waiting:
```

```
367                 b = self.ser.readline().decode()
368                 self.weight = float(b)
369                 self.dynamicText.set(str(self.weight))
370                 if not self.weight == 0:
371 
372                     self.weight_received = True
373                     self.currTime = time.time()
374                     self.cam.shutter(constants.start)
375                     self.canva.itemconfig(self.myrectangle,fill='
                            green')
376 
377             return
378 
379         # get proximity signal
380         elif not self.proximity_received and self.weight_received
                == True:
381             if not self.proximity_requested:
382                 self.ser.flush()
383                 self.ser.write("p".encode())
384                 self.proximity_requested = True
385                 return
386             elif self.ser.in_waiting:
387                 self.canva.itemconfig(self.myrectangle, fill='red')
388                 b = str(self.ser.readline().decode().strip("\r\n"))
389 
390                 if b == "Record":
391                     self.vidRdy = False
392                     x3 = threading.Thread(target = self.
                            receiveMetal)
393                     x2 = threading.Thread(target = self.gopro)
394                     x1 = threading.Thread(target=self.record)
395 
396                     self.cutIncrement = time.time()-self.currTime
                            -1.00
397 
398                     #x2.start()
399                     x3.start()
400                     x1.start()
401                     self.proximity_received = True
402 
403             return
404 
405         elif self.metal_received and not self.very_received:
406             self.cam.shutter(constants.stop)
407             self.t = np.asarray(self.metal_data)
408             self.fig.add_subplot(111).plot(self.t)
409             self.canvas.draw()
410             self.very_received = True
411 
412         if self.recorded and self.metal_received:
413             self.displayButton.configure(state=NORMAL)
414             self.rdySave = True
415 
416             librosa.display.specshow(self.mel3, x_axis='time', ax=
                    self.specFig.add_subplot(111))
417             self.specCanvas.draw()
418             self.update = False
```

```
419
420     def receiveMetal(self):
421         if self.ser.in_waiting:
422             complete = False
423             while not complete: #read until Arduino sends complete
                    signal
424                 b = self.ser.readline()
425                 b = b.decode()
426                 try:
427                     self.metal_data.append(float(b.strip("\r\n")))
                            #try to append metal data
428                 except: #if not successfull, then arduino has sent
                        chars
429                     if b.strip("\r\n")=="complete":
430                         complete = True
431                         self.metal_received = True
432                         continue
433                     self.metal_data.append(0)
434             self.ser.flush()
435
436     def save(self):
437         self.reset_data()
438
439         # Read last measurement number from txt file
440         measurementNumberPath = self.path_measurements + '/
                MeasurementNumber.txt'
441         NumberFile = open(measurementNumberPath, 'r')
442         MeasurementNumber = int(NumberFile.read())
443         NumberFile.close()
444
445         # Increment measurement number
446         MeasurementNumber += 1
447         NumberFile = open(measurementNumberPath, 'w')
448         NumberFile.write(str(MeasurementNumber))
449         NumberFile.close()
450
451
452         # Create recording folder
453         self.StorageLocation = (self.path_measurements + self.
                classList[self.classNum] + str(MeasurementNumber) + '/'
                )
454         os.mkdir(self.StorageLocation)
455
456
457
458         # saving data
459         np.save(self.StorageLocation + 'weight.npy', self.weight)
460
461         np.save(self.StorageLocation + 'metal_data.npy', self.t)
462
463         np.save(self.StorageLocation + 'MFCCS1.npy', self.mfccs1)
464         np.save(self.StorageLocation + 'MelSpectrogram1.npy', self.
                mel1)
465
466         np.save(self.StorageLocation + 'MFCCS2.npy', self.mfccs2)
467         np.save(self.StorageLocation + 'MelSpectrogram2.npy', self.
                mel2)
```

```
468
469         np.save(self.StorageLocation + 'MFCCS3.npy', self.mfccs3)
470         np.save(self.StorageLocation + 'MelSpectrogram3.npy', self.
                mel3)
471
472         np.save(self.StorageLocation + 'MFCCS4.npy', self.mfccs4)
473         np.save(self.StorageLocation + 'MelSpectrogram4.npy', self.
                mel4)
474
475         # moving wav-files
476         os.rename(self.tempPath + '/Recording1.wav', self.
                StorageLocation + 'Recording1.Wav')
477         os.rename(self.tempPath + '/Recording2.wav', self.
                StorageLocation + 'Recording2.Wav')
478         os.rename(self.tempPath + '/Recording3.wav', self.
                StorageLocation + 'Recording3.Wav')
479         os.rename(self.tempPath + '/Recording4.wav', self.
                StorageLocation + 'Recording4.Wav')
480
481         # running cut vid thread
482         x4 = threading.Thread(target=self.cutVid)
483         x4.start()
484
485
486
487
488
489
490     def cutVid(self):
491         os.chdir(self.tempPath)
492         while not self.vidRdy:
493             pass
494         #deleting gopro contents
495         for file in os.listdir('.'):
496             if file[:8] ==”101GOPRO”:
497                 ffmpeg_extract_subclip(file, self.cutIncrement,
                        self.cutIncrement+2, targetname=self.
                        StorageLocation+”test.mp4”)
498                 os.remove(file)
499         self.vidDownloaded = True
500
501
502
503     def updater(self):#Function causing self.run_script() to loop
504         self.run_script()#Runs script
505         if self.update:
506             self._job = self.after(100, self.updater)#Schedule to
                    run self.updater after 100ms
507
508     def reset_data(self):
509
510         # resetting arduino variables
511         self.ser.write(”a”.encode())
512
513         # resetting variables
514         self.vidDownloaded = False
515         self.weight_received = False
```

```
516         self.metal_received = False
517         self.proximity_received = False
518         self.metal_recorded = False
519         self.metal_data = []
520         self.weight_requested = False
521         self.proximity_requested = False
522         self.metal_requested = False
523         self.recorded = False
524         self.rdySave = False
525         self.update = True
526         self.very_received = False
527 
528         self.ser.flush()
529         self.fig.clf()
530         self.specFig.clf()
531         librosa.cache.clear()
532         try:
533             self.canvas.pack_forget()
534         except:
535             pass
536         try:
537             self.specCanvas.pack_forget()
538         except:
539             pass
540 
541 
542 
543         # disabling save button
544         self.displayButton.configure(state=DISABLED)
545         self.updater()
546 
547     def discard(self):
548         self.reset_data()
549         #self.cam.delete("all")
550         try:
551             os.chdir(self.tempPath)
552             for file in os.listdir('.'):
553                 if file[:8] =="101GOPRO":
554                     os.remove(file)
555                     self.vidDownloaded = True
556         except:
557             pass
558 
559         try:
560             self.cam.shutter(constants.stop)
561         except:
562             pass
563         try:
564             os.remove(self.tempPath+'/Recording1.wav')
565             os.remove(self.tempPath + '/Recording2.wav')
566             os.remove(self.tempPath + '/Recording3.wav')
567             os.remove(self.tempPath + '/Recording4.wav')
568         except:
569             pass
570 
571     def define_class(self, classNum, button1,button2):
572         #For old buttons
```

```
573 ##         for B in (self.button1, self.button2, self.button3, self.
    button4, self.button5, self.button6, self.button7):
574 ##             B.configure(state=NORMAL)
575 ##         aButton.configure(state=DISABLED)
576 ##         self.classNum = classNum
577 
578         classLabels = (self.checkVar1.get(),self.checkVar2.get())
579         self.dynamicText2.set(str(classLabels[0])+","+str(
                classLabels[1]))
580 
581     def gopro(self):
582 
583         time.sleep(1)
584         self.cam.shutter(constants.stop)
585         os.chdir(self.tempPath)
586         self.clip = self.cam.downloadLastMedia()
587         self.vidRdy = True
588 
589 
590     def record(self):
591         chunk = 256  # Record in chunks of 256 samples
592         sample_format = pyaudio.paInt16  # 16 bits per sample
593         Channels = 4
594 
595 
596         fs = 48000 #Record at 48 000 frames per second
597         seconds = 2
598         filename1 = (self.tempPath + '/Recording1.wav')
599         filename2 = (self.tempPath + '/Recording2.wav')
600         filename3 = (self.tempPath + '/Recording3.wav')
601         filename4 = (self.tempPath + '/Recording4.wav')
602 
603         p = pyaudio.PyAudio()  # Create an interface to PortAudio
604 
605         stream = p.open(format=sample_format,
606                         channels=Channels,
607                         rate=fs,
608                         frames_per_buffer=chunk,
609                         input=True)
610 
611         frames1 = []  # Initialize array to store frames
612         frames2 = []  # Initialize array to store frames
613         frames3 = []  # Initialize array to store frames
614         frames4 = []  # Initialize array to store frames
615 
616         # Store data in chunks for 2 seconds
617         for i in range(0, int(fs / chunk * seconds)):
618             data = stream.read(chunk)
619             # Convert string to numpy array
620             dataArray = np.frombuffer(data, dtype='int16')
621             # Deinterleave channels
622             channel1 = dataArray[0::Channels]
623             channel2 = dataArray[1::Channels]
624             channel3 = dataArray[2::Channels]
625             channel4 = dataArray[3::Channels]
626             # Convert back to string
627             dataChannel1 = channel1.tostring()
```

```
628             dataChannel2 = channel2.tostring()
629             dataChannel3 = channel3.tostring()
630             dataChannel4 = channel4.tostring()
631             frames1.append(dataChannel1)
632             frames2.append(dataChannel2)
633             frames3.append(dataChannel3)
634             frames4.append(dataChannel4)
635
636         # Stop and close the stream
637         stream.stop_stream()
638         stream.close()
639         # Terminate the PortAudio interface
640         p.terminate()
641
642         # Save the recorded data as a WAV file
643         wf = wave.open(filename1, 'wb')
644         wf.setnchannels(1)
645         wf.setsampwidth(p.get_sample_size(sample_format))
646         wf.setframerate(fs)
647         wf.writeframes(b''.join(frames1))
648         wf.close()
649
650         wf = wave.open(filename2, 'wb')
651         wf.setnchannels(1)
652         wf.setsampwidth(p.get_sample_size(sample_format))
653         wf.setframerate(fs)
654         wf.writeframes(b''.join(frames2))
655         wf.close()
656
657         wf = wave.open(filename3, 'wb')
658         wf.setnchannels(1)
659         wf.setsampwidth(p.get_sample_size(sample_format))
660         wf.setframerate(fs)
661         wf.writeframes(b''.join(frames3))
662         wf.close()
663
664         wf = wave.open(filename4, 'wb')
665         wf.setnchannels(1)
666         wf.setsampwidth(p.get_sample_size(sample_format))
667         wf.setframerate(fs)
668         wf.writeframes(b''.join(frames4))
669         wf.close()
670
671         # Create spectrogram and store as file
672         # Channel 1
673         try:
674             audio, sample_rate = librosa.load(filename1, res_type='
                    kaiser_fast')
675             self.mfccs1 = librosa.feature.mfcc(y=audio, sr=
                    sample_rate, hop_length=32, n_mfcc=200)
676             self.mfccs1 = librosa.power_to_db(self.mfccs1)
677
678         except Exception:
679             print("Error encountered while parsing file: ",
                    filename1)
680
681         try:
```

```
682             audio, sample_rate = librosa.load(filename1, res_type='
                    kaiser_fast')
683             self.mel1 = librosa.feature.melspectrogram(y=audio, sr=
                    sample_rate, n_fft=2048, hop_length=32)
684             self.mel1 = librosa.power_to_db(self.mel1)
685
686         except Exception:
687             print("Error encountered while parsing file: ",
                    filename1)
688
689         # Channel 2
690         try:
691             audio, sample_rate = librosa.load(filename2, res_type='
                    kaiser_fast')
692             self.mfccs2 = librosa.feature.mfcc(y=audio, sr=
                    sample_rate, hop_length=32, n_mfcc=200)
693             self.mfccs2 = librosa.power_to_db(self.mfccs2)
694
695         except Exception:
696             print("Error encountered while parsing file: ",
                    filename2)
697
698         try:
699             audio, sample_rate = librosa.load(filename2, res_type='
                    kaiser_fast')
700             self.mel2 = librosa.feature.melspectrogram(y=audio, sr=
                    sample_rate, n_fft=2048, hop_length=32)
701             self.mel2 = librosa.power_to_db(self.mel2)
702
703         except Exception:
704             print("Error encountered while parsing file: ",
                    filename2)
705
706         # Channel 3
707         try:
708             audio, sample_rate = librosa.load(filename3, res_type='
                    kaiser_fast')
709             self.mfccs3 = librosa.feature.mfcc(y=audio, sr=
                    sample_rate, hop_length=32, n_mfcc=200)
710             self.mfccs3 = librosa.power_to_db(self.mfccs3)
711
712         except Exception:
713             print("Error encountered while parsing file: ",
                    filename3)
714
715         try:
716             audio, sample_rate = librosa.load(filename3, res_type='
                    kaiser_fast')
717             self.mel3 = librosa.feature.melspectrogram(y=audio, sr=
                    sample_rate, n_fft=2048, hop_length=32)
718             self.mel3 = librosa.power_to_db(self.mel3)
719
720         except Exception:
721             print("Error encountered while parsing file: ",
                    filename3)
722
723         # Channel 4
```

```
724         try:
725             audio, sample_rate = librosa.load(filename4, res_type='
                kaiser_fast')
726             self.mfccs4 = librosa.feature.mfcc(y=audio, sr=
                sample_rate, hop_length=32, n_mfcc=200)
727             self.mfccs4 = librosa.power_to_db(self.mfccs4)
728 
729         except Exception:
730             print("Error encountered while parsing file: ",
                filename4)
731 
732         try:
733             audio, sample_rate = librosa.load(filename4, res_type='
                kaiser_fast')
734             self.mel4 = librosa.feature.melspectrogram(y=audio, sr=
                sample_rate, n_fft=2048, hop_length=32)
735             self.mel4 = librosa.power_to_db(self.mel4)
736 
737         except Exception:
738             print("Error encountered while parsing file: ",
                filename4)
739 
740         self.recorded = True
741 
742     def __del__(self):
743         self.ser.close()
744 
745 
746 
747 
748 
749 if __name__ == "__main__":
750 
751     app = App()
752     atexit.register(app.exit)
753     app.mainloop()
754     app.exit()
```

## A.2 CNN Model

All code regarding CNN models are presented here. The complete model class is only presented for Model 1, as only "def build_model(self)" differs between them. Complete code is presented for multi-class Model 1 as well, since it differs in the way it creates confusion matrices.

### A.2.1 Model 1

```
1 from tensorflow.keras.models import Sequential, Model
2 from tensorflow.keras.layers import Conv2D, MaxPooling2D,
      AveragePooling2D, GlobalAveragePooling2D, Dropout, Dense,
      Activation
3 from tensorflow.keras.optimizers import Adam, RMSprop
4 from tensorflow.keras.callbacks import ModelCheckpoint,
      ReduceLROnPlateau
5 from tensorflow.keras.mixed_precision import experimental as
      mixed_precision
```

```
6 from sklearn.metrics import accuracy_score, hamming_loss,
      multilabel_confusion_matrix, ConfusionMatrixDisplay,
      precision_score
7 import matplotlib.pyplot as plt
8 import sys
9 from utils2 import Logger
10 import os
11 import datetime
12 from contextlib import redirect_stdout
13 import numpy as np
14
15 class trashDetCNN:
16     def __init__(self,input_shape, num_classes,save_dir,batch_size,
          epoch_count):
17         self.input_shape = input_shape
18         self.batch_size = batch_size
19         self.epoch_count = epoch_count
20         self.num_classes = num_classes
21         self.model_dir = os.path.join(save_dir,datetime.datetime.
               now().strftime('%d-%m-%Y_%H-%M-%S'))
22         os.mkdir(self.model_dir)
23         sys.stdout = Logger(self.model_dir)
24         self.build_model()
25         print(self.input_shape)
26         print(self.batch_size)
27         print(self.epoch_count)
28         print(self.num_classes)
29         print(self.model_dir)
30
31     def build_model(self):
32         print('Building model...')
33
34         ### Convolutional blocks
35         self.model = Sequential()
36         self.model.add(Conv2D(filters=32, kernel_size=(3,3),
               strides = (1,1), data_format = "channels_last",
               input_shape=self.input_shape, activation='relu'))
37         self.model.add(MaxPooling2D(pool_size=(2,2)))
38         self.model.add(Dropout(0.2))
39
40         self.model.add(Conv2D(filters=64, kernel_size=(3,3),
               activation='relu'))
41         self.model.add(MaxPooling2D(pool_size=2))
42         self.model.add(Dropout(0.2))
43
44         self.model.add(Conv2D(filters=128, kernel_size=(3,3),
               activation='relu'))
45         self.model.add(MaxPooling2D(pool_size=2))
46         self.model.add(Dropout(0.2))
47
48         self.model.add(Conv2D(filters=256, kernel_size=(3,3),
               activation='relu'))
49         self.model.add(AveragePooling2D(pool_size=2))
50         self.model.add(Dropout(0.2))
51
52         self.model.add(Conv2D(filters=256, kernel_size=(3,3),
               activation='relu'))
```

```
53          self.model.add(AveragePooling2D(pool_size=2))
54          self.model.add(Dropout(0.2))
55
56
57
58          #model.add(Flatten())
59
60          self.model.add(GlobalAveragePooling2D())
61
62          ## Softmax Output
63          self.model.add(Dense(self.num_classes, name='preds'))
64          self.model.add(Activation('sigmoid'))
65          #kernel_regularizer=regularizers.l2(0.01)
66          opt = Adam(lr=0.001)
67          #opt = RMSprop(lr=0.0005)  # Optimizer
68
69          self.model.compile(
70                  loss='binary_crossentropy',
71                  optimizer=opt,
72                  metrics=['accuracy']
73              )
74
75          print(self.model.summary())
76
77
78      def train_model(self,x_train, y_train, x_val, y_val,x_test,
            y_test):
79          checkpoint_callback = ModelCheckpoint(r'G:\workspace_python
                \ml_modeller\cnn\cnn2\weights\weights.best.h5', monitor
                ='val_accuracy', verbose=2, save_best_only=True, mode='
                max')
80
81          reducelr_callback = ReduceLROnPlateau(
82                      monitor='val_accuracy', factor=0.5, patience
                            =10,
83                      verbose=2
84                  )
85          callbacks_list = [checkpoint_callback, reducelr_callback]
86
87          # Fit the model and get training history.
88          print('Training...')
89          #with open(os.path.join(model_dir,'log.txt'), 'w') as f:
90            #  with redirect_stdout(f):
91          self.history = self.model.fit(x_train, y_train, batch_size=
                self.batch_size, epochs=self.epoch_count,
                validation_data=(x_val, y_val), verbose=2, callbacks=
                callbacks_list)
92          self.save_model(x_train, y_train, x_val, y_val,x_test,
                y_test)
93
94      def save_model(self,x_train,y_train,x_val,y_val,x_test,y_test):
95
96          self.model.save(os.path.join(self.model_dir,'Model.h5'))
97          with open(os.path.join(self.model_dir,'modelsummary.txt'),
                'w') as f:
98              with redirect_stdout(f):
99                  self.model.summary()
```

```
100         confidences = self.model.predict(x_val)
101         np.save(os.path.join(self.model_dir,"confidences"),
                confidences)
102
103         confidences2 = self.model.predict(x_test)
104         np.save(os.path.join(self.model_dir,"confidences_test"),
                confidences2)
105
106         predictions = confidences
107         for i in range(len(confidences)):
108             for n in range(len(confidences[i])):
109                 if confidences[i,n] >0.8:
110                     predictions[i,n] = 1
111                 else:
112                     predictions[i,n] = 0
113
114         with open(os.path.join(self.model_dir,'accuracy.txt'), 'w')
                as f:
115             f.write(str(max(self.history.history['val_accuracy'])))
116
117         conf = multilabel_confusion_matrix(y_val,predictions)
118
119         disp = ConfusionMatrixDisplay(conf[0],['Metal','No Metal'])
120         disp = disp.plot(xticks_rotation='vertical')
121         plt.savefig(os.path.join(self.model_dir,'
                ConfusionMatrixMetal.png'),bbox_inches='tight')
122
123         disp = ConfusionMatrixDisplay(conf[1],['Glass','No Glass'])
124         disp = disp.plot(xticks_rotation='vertical')
125         plt.savefig(os.path.join(self.model_dir,'
                ConfusionMatrixGlass.png'),bbox_inches='tight')
126
127         plt.clf()
128
129         plt.plot(self.history.history['accuracy'])
130         plt.plot(self.history.history['val_accuracy'])
131         plt.title('model accuracy')
132         plt.ylabel('accuracy')
133         plt.xlabel('epoch')
134         plt.legend(['train', 'val'], loc='upper left')
135         plt.text(self.epoch_count*0.56, (max(self.history.history['
                accuracy'])*0.8), ('Max accuracy: ', round(max(self.
                history.history['accuracy']), 5)), bbox=dict(facecolor=
                'C0', alpha=0.5))
136         plt.text(self.epoch_count*0.56, (max(self.history.history['
                accuracy'])*0.7), ('Max accuracy: ', round(max(self.
                history.history['val_accuracy']), 5)), bbox=dict(
                facecolor='C1', alpha=0.5))
137         plt.savefig(os.path.join(self.model_dir,'Accuracy.png'))
138         plt.clf()
139
140
141         # Summarize history for loss
142         plt.plot(self.history.history['loss'])
143         plt.plot(self.history.history['val_loss'])
144         plt.title('model loss')
145         plt.ylabel('loss')
```

```
146         plt.xlabel('epoch')
147         plt.legend(['train', 'val'], loc='upper left')
148         plt.text(self.epoch_count*0.56, (max(self.history.history['
                loss'])*0.5), ('Min loss: ', round(min(self.history.
                history['loss']), 5)), bbox=dict(facecolor='C0', alpha
                =0.5))
149         plt.text(self.epoch_count*0.56, (max(self.history.history['
                val_loss'])*0.1), ('Min loss: ', round(min(self.history
                .history['val_loss']), 5)), bbox=dict(facecolor='C1',
                alpha=0.5))
150         plt.savefig(os.path.join(self.model_dir,'Loss.png'))
151         plt.clf()
152
153         np.save(os.path.join(self.model_dir,"loss"),self.history.
                history['loss'])
154         np.save(os.path.join(self.model_dir,"val_loss"),self.
                history.history['val_loss'])
155         np.save(os.path.join(self.model_dir,"acc"),self.history.
                history['accuracy'])
156         np.save(os.path.join(self.model_dir,"val_acc"),self.history
                .history['val_accuracy'])
```

### A.2.2 Model 2

```
1     def build_model(self):
2         print('Building model...')
3
4         ### Convolutional blocks
5         self.model = Sequential()
6         self.model.add(Conv2D(filters=16, kernel_size=(9,9),
                strides = (1,1), data_format = "channels_last",
                input_shape=self.input_shape, activation='relu'))
7         self.model.add(MaxPooling2D(pool_size=(2,2)))
8         self.model.add(Dropout(0.4))
9
10        self.model.add(Conv2D(filters=32, kernel_size=(7,7),
                activation='relu'))
11        self.model.add(MaxPooling2D(pool_size=2))
12        self.model.add(Dropout(0.2))
13
14        self.model.add(Conv2D(filters=64, kernel_size=(5,5),
                activation='relu'))
15        self.model.add(MaxPooling2D(pool_size=2))
16        self.model.add(Dropout(0.2))
17
18        self.model.add(Conv2D(filters=128, kernel_size=(3,3),
                activation='relu'))
19        self.model.add(MaxPooling2D(pool_size=2))
20        self.model.add(Dropout(0.2))
21
22        self.model.add(Conv2D(filters=256, kernel_size=(1,1),
                activation='relu'))
23        self.model.add(MaxPooling2D(pool_size=2))
24        self.model.add(Dropout(0.2))
25
26        self.model.add(Conv2D(filters=512, kernel_size=(1,1),
                activation='relu'))
```

```
27          self.model.add(MaxPooling2D(pool_size=2))
28          self.model.add(Dropout(0.2))
29
30          #model.add(Flatten())
31
32          self.model.add(GlobalAveragePooling2D())
33          ## Softmax Output
34          self.model.add(Dense(128, activation = 'relu', name='dense1
                ''))
35          self.model.add(Dense(64, activation = 'relu', name='dense2'
                ))
36          self.model.add(Dense(32, activation = 'relu', name='dense3'
                ))
37          self.model.add(Dense(16, activation = 'relu', name='dense4'
                ))
38          self.model.add(Dense(self.num_classes, name='preds'))
39          self.model.add(Activation('sigmoid'))
40          #kernel_regularizer=regularizers.l2(0.01)
41          opt = Adam(lr=0.001)
42          #opt = RMSprop(lr=0.0005)  # Optimizer
43
44          self.model.compile(
45                  loss='binary_crossentropy',
46                  optimizer=opt,
47                  metrics=['accuracy']
48              )
49
50          print(self.model.summary())
```

### A.2.3 Model 3

```
1       def build_model(self):
2           print('Building model...')
3
4           ### Convolutional blocks
5           self.model = Sequential()
6           self.model.add(Conv2D(filters=32, kernel_size=(4,8),
                strides = (1,1), data_format = "channels_last",
                input_shape=self.input_shape, padding = 'same',
                activation='relu'))
7           self.model.add(MaxPooling2D(pool_size=(2,2)))
8           self.model.add(Dropout(0.2))
9
10          self.model.add(Conv2D(filters=64, kernel_size=(5,5),padding
                = 'same', activation='relu'))
11          self.model.add(MaxPooling2D(pool_size=2))
12          self.model.add(Dropout(0.2))
13
14          self.model.add(Conv2D(filters=128, kernel_size=(5,5),
                padding = 'same', activation='relu'))
15          self.model.add(MaxPooling2D(pool_size=2))
16          self.model.add(Dropout(0.2))
17
18          self.model.add(Conv2D(filters=256, kernel_size=(3,3),
                padding = 'same', activation='relu'))
19          self.model.add(MaxPooling2D(pool_size=2))
20          self.model.add(Dropout(0.2))
```

```
21
22          self.model.add(Conv2D(filters=256, kernel_size=(1,1),
                padding = 'same', activation='relu'))
23          self.model.add(MaxPooling2D(pool_size=2))
24          self.model.add(Dropout(0.2))
25
26
27          #model.add(Flatten())
28
29          self.model.add(GlobalAveragePooling2D())
30          ## Softmax Output
31          self.model.add(Dense(128, activation = 'relu', name='dense1
                '))
32          self.model.add(Dense(64, activation = 'relu', name='dense2'
                ))
33          self.model.add(Dense(self.num_classes, name='preds'))
34          self.model.add(Activation('sigmoid'))
35          #kernel_regularizer=regularizers.l2(0.01)
36          opt = Adam(lr=0.001)
37          #opt = RMSprop(lr=0.0005)  # Optimizer
38
39          self.model.compile(
40                  loss='binary_crossentropy',
41                  optimizer=opt,
42                  metrics=['accuracy']
43              )
44
45          print(self.model.summary())
```

### A.2.4 Multi-class model(model 1)

```
1  from tensorflow.keras.models import Sequential, Model
2  from tensorflow.keras.layers import Conv2D, MaxPooling2D,
       AveragePooling2D, GlobalAveragePooling2D, Dropout, Dense,
       Activation
3  from tensorflow.keras.optimizers import Adam, RMSprop
4  from tensorflow.keras.callbacks import ModelCheckpoint,
       ReduceLROnPlateau
5  from tensorflow.keras.mixed_precision import experimental as
       mixed_precision
6  from sklearn.metrics import accuracy_score, hamming_loss,
       confusion_matrix, ConfusionMatrixDisplay, precision_score
7  import matplotlib.pyplot as plt
8  import sys
9  from utils2 import Logger
10 import os
11 import datetime
12 from contextlib import redirect_stdout
13 import numpy as np
14 import json
15
16 class trashDetCNN:
17     def __init__(self,input_shape, num_classes,save_dir,batch_size,
           epoch_count):
18         self.input_shape = input_shape
19         self.batch_size = batch_size
20         self.epoch_count = epoch_count
```

```
21          self.num_classes = num_classes
22          self.model_dir = os.path.join(save_dir,datetime.datetime.
                now().strftime('%d-%m-%Y_%H-%M-%S'))
23          os.mkdir(self.model_dir)
24          sys.stdout = Logger(self.model_dir)
25          self.build_model()
26          print(self.input_shape)
27          print(self.batch_size)
28          print(self.epoch_count)
29          print(self.num_classes)
30          print(self.model_dir)
31
32      def build_model(self):
33          print('Building model...')
34
35          ### Convolutional blocks
36          self.model = Sequential()
37          self.model.add(Conv2D(filters=32, kernel_size=(3,3),
                strides = (1,1), data_format = "channels_last",
                input_shape=self.input_shape, activation='relu'))
38          self.model.add(MaxPooling2D(pool_size=(2,2)))
39          self.model.add(Dropout(0.2))
40
41          self.model.add(Conv2D(filters=64, kernel_size=(3,3),
                activation='relu'))
42          self.model.add(MaxPooling2D(pool_size=2))
43          self.model.add(Dropout(0.2))
44
45          self.model.add(Conv2D(filters=128, kernel_size=(3,3),
                activation='relu'))
46          self.model.add(MaxPooling2D(pool_size=2))
47          self.model.add(Dropout(0.2))
48
49          self.model.add(Conv2D(filters=256, kernel_size=(3,3),
                activation='relu'))
50          self.model.add(AveragePooling2D(pool_size=2))
51          self.model.add(Dropout(0.2))
52
53          self.model.add(Conv2D(filters=256, kernel_size=(3,3),
                activation='relu'))
54          self.model.add(AveragePooling2D(pool_size=2))
55          self.model.add(Dropout(0.2))
56
57          #model.add(Flatten())
58
59          self.model.add(GlobalAveragePooling2D())
60          ## Softmax Output
61
62          self.model.add(Dense(self.num_classes, activation = '
                softmax', name='preds'))
63
64          #kernel_regularizer=regularizers.l2(0.01)
65          opt = Adam(lr=0.001)
66          #opt = RMSprop(lr=0.0005)  # Optimizer
67
68          self.model.compile(
69                  loss='categorical_crossentropy',
```

```
70                  optimizer=opt,
71                  metrics=['accuracy']
72              )
73
74          print(self.model.summary())
75
76
77      def train_model(self,x_train, y_train, x_val, y_val,x_test,
            y_test):
78          checkpoint_callback = ModelCheckpoint(r'G:\workspace_python
                \ml_modeller\cnn\cnn2\weights\weights.best.h5', monitor
                ='val_accuracy', verbose=2, save_best_only=True, mode='
                max')
79
80          reducelr_callback = ReduceLROnPlateau(
81                      monitor='val_accuracy', factor=0.5, patience
                            =10,
82                      verbose=2
83                  )
84          callbacks_list = [checkpoint_callback, reducelr_callback]
85
86          # Fit the model and get training history.
87          print('Training...')
88          #with open(os.path.join(model_dir,'log.txt'), 'w') as f:
89            #  with redirect_stdout(f):
90          self.history = self.model.fit(x_train, y_train, batch_size=
                self.batch_size, epochs=self.epoch_count,
                validation_data=(x_val, y_val), verbose=2, callbacks=
                callbacks_list)
91          self.save_model(x_train, y_train, x_val, y_val,x_test,
                y_test)
92
93      def save_model(self,x_train,y_train,x_val,y_val,x_test,y_test):
94
95          self.model.save(os.path.join(self.model_dir,'Model.h5'))
96          with open(os.path.join(self.model_dir,'modelsummary.txt'),
                'w') as f:
97              with redirect_stdout(f):
98                  self.model.summary()
99          confidences = self.model.predict_classes(x_val)
100         rounded_labels=np.argmax(y_val, axis=1)
101         rounded_labels[1]
102
103         rounded_labels2=np.argmax(y_test, axis=1)
104         rounded_labels2[1]
105         np.save(os.path.join(self.model_dir,"confidences"),
                confidences)
106         predictions = confidences
107
108
109
110         conf = confusion_matrix(rounded_labels, predictions)
111         #conf = multilabel_confusion_matrix(y_val,predictions)
112
113         disp = ConfusionMatrixDisplay(conf,['MixPure', 'MetalAndMix
                ','GlassAndMetal','GlassAndMix'])
114         disp = disp.plot(xticks_rotation='vertical')
```

```
115         plt.savefig(os.path.join(self.model_dir,'ConfusionMatrixVal
                .png'),bbox_inches='tight')
116
117         plt.clf()
118
119         confidences2 = self.model.predict_classes(x_test)
120         conf2 = confusion_matrix(rounded_labels2, confidences2)
121         disp2 = ConfusionMatrixDisplay(conf2,['MixPure', '
                MetalAndMix','GlassAndMetal','GlassAndMix'])
122         disp2 = disp2.plot(xticks_rotation='vertical', cmap = '
                Blues')
123         plt.savefig(os.path.join(self.model_dir,'
                ConfusionMatrixTest.png'),bbox_inches='tight')
124
125         plt.clf()
126         print(confidences2.shape)
127         print(y_test.shape)
128         print(x_test.shape)
129         con = np.argmax(confidences2, axis=0)
130         print(confidences2.transpose())
131         print(rounded_labels2)
132
133         with open(os.path.join(self.model_dir,'accuracy.json'), 'w'
                ) as f:
134             json.dump({'valAcc':str(max(self.history.history['
                    val_accuracy'])),'testAcc':str(accuracy_score(
                    rounded_labels2.transpose(),confidences2.transpose
                    ()))},f)
135
136         plt.plot(self.history.history['accuracy'])
137         plt.plot(self.history.history['val_accuracy'])
138         plt.title('model accuracy')
139         plt.ylabel('accuracy')
140         plt.xlabel('epoch')
141         plt.legend(['train', 'val'], loc='upper left')
142         plt.text(self.epoch_count*0.56, (max(self.history.history['
                accuracy'])*0.8), ('Max accuracy: ', round(max(self.
                history.history['accuracy']), 5)), bbox=dict(facecolor=
                'C0', alpha=0.5))
143         plt.text(self.epoch_count*0.56, (max(self.history.history['
                accuracy'])*0.7), ('Max accuracy: ', round(max(self.
                history.history['val_accuracy']), 5)), bbox=dict(
                facecolor='C1', alpha=0.5))
144         plt.savefig(os.path.join(self.model_dir,'Accuracy.png'))
145
146
147
148         # Summarize history for loss
149         plt.plot(self.history.history['loss'])
150         plt.plot(self.history.history['val_loss'])
151         plt.title('model loss')
152         plt.ylabel('loss')
153         plt.xlabel('epoch')
154         plt.legend(['train', 'val'], loc='upper left')
155         plt.text(self.epoch_count*0.56, (max(self.history.history['
                loss'])*0.5), ('Min loss: ', round(min(self.history.
                history['loss']), 5)), bbox=dict(facecolor='C0', alpha
```

```
            =0.5))
156         plt.text(self.epoch_count*0.56, (max(self.history.history['
                loss'])*0.1), ('Min loss: ', round(min(self.history.
                history['val_loss']), 5)), bbox=dict(facecolor='C1',
                alpha=0.5))
157         plt.savefig(os.path.join(self.model_dir,'Loss.png'))
```

### A.2.5 Main

```
1  from model_CNN1 import *#defines which model to train
2  import os
3  import json
4
5
6  batch_size = 32
7  epoch_count = 70
8
9
10 def train_model(root_dir,save_dir, iterations):
11     if not os.path.exists(save_dir):
12         os.mkdir(save_dir)
13
14     x_train = np.load(os.path.join(root_dir,'x_train.npy'))
15     y_train = np.load(os.path.join(root_dir,'y_train.npy'))
16     x_val = np.load(os.path.join(root_dir,'x_val.npy'))
17     y_val = np.load(os.path.join(root_dir,'y_val.npy'))
18     x_test = np.load(os.path.join(root_dir,'x_test.npy'))
19     y_test = np.load(os.path.join(root_dir,'y_test.npy'))
20
21
22     input_shape = (x_train.shape[1],x_train.shape[2],x_train.shape
           [3])
23     num_classes = y_train.shape[1]
24
25     for i in range(iterations):
26         print("Training session ",i+1)
27         model = trashDetCNN(input_shape,num_classes,save_dir,
               batch_size,epoch_count)
28         model.train_model(x_train,y_train,x_val,y_val,x_test,y_test
               )
29
30
31     x_train = None
32     y_train = None
33     x_val = None
34     y_val = None
35
36 save_dirs = [r'E:\Master\Models\Model1\1100110010',r'E:\Master\
       Models\Model1\1111111110']
37
38 data_dirs = [r'E:\Master\SavedData\1100110010',r'E:\Master\
       SavedData\1111111110']
39
40
41 for save_dir,data_dir in zip(save_dirs,data_dirs):
42     train_model(data_dir,save_dir,20)
```

### A.2.6 Saving Arrays

This code was used to load the saved data into arrays such that they could be saved.

```
from metalInterpolator import *
import numpy as np
from cv2 import resize
import random
import os
import matplotlib.pyplot as plt
import librosa
import librosa.display
from tensorflow.keras.utils import to_categorical
from sys import getsizeof
import pickle
import json
import sys
from sklearn.preprocessing import MultiLabelBinarizer


def listdir_fullPath(path):
    return[os.path.join(path,s) for s in os.listdir(path)]

def getDataList(data_path, included_data):
    num2label = {0:'MelSpectrogram1.npy',1:'MelSpectrogram2.npy',2:
        'MelSpectrogram3.npy',3:'MelSpectrogram4.npy',4:'MFCCS1.npy
        ',5:'MFCCS2.npy',6:'MFCCS3.npy',7:'MFCCS4.npy',8:'
        metal_data.npy',9:'weight.npy'} #Dictionary used to
        detemine what data to include.
    labelNums = [i for i in range(len(included_data)) if
        included_data[i]==1] #Uses included_data to determine what
        data to include
    dataList = []
    for labelNum in labelNums:
        dataList.append(os.path.join(data_path,num2label[labelNum])
            )
    return dataList

def getDataList_fromArray(data_paths,included_data):
    return [getDataList(data_path,included_data) for data_path in
        data_paths]

def getPathList(root_dir,measurements_per_class,split, classList,
    included_data):
    classDict = {'MetalPure':(0,),'GlassPure':(1,),'MixPure':(),'
        GlassAndMetal':(0,1),'MetalAndMix':(0,),'GlassAndMix':(1,)}
         #Label data according to classDict
    print("Gathering data paths...")
    classList = classList
    if measurements_per_class == 'all':
        measurements_per_class = None
        start = None
    elif type(measurements_per_class) == int:
        measurements_per_class = measurements_per_class
        start = 0
    else:
```

```
42          print("Provide 'all' or an integer for input '
                images_per_class'")
43          return
44
45      train_split = split
46
47      y_train = []
48      y_val = []
49
50      x_train = []
51      x_val = []
52
53      mlb = MultiLabelBinarizer()
54
55      for n, label in enumerate(classList):
56          tmp = listdir_fullPath(os.path.join(root_dir,label))[start:
                measurements_per_class]
57          random.Random(42).shuffle(tmp)
58          x_train += getDataList_fromArray(tmp,included_data)
59          for i in range(int(round(train_split*len(tmp)))):
60              y_val.append(classDict[label])
61              x_val.append(x_train.pop())
62          y_train +=[classDict[label]]*int(len(tmp)-round(len(tmp)*
                split))
63      return x_train, mlb.fit_transform(y_train), x_val, mlb.
            fit_transform(y_val)
64
65  def load_data_from_paths(x,shape,t1,t2,time_frame):
66      if len(x[0])>1:
67          x_dat = np.zeros([len(x),shape[0],time_frame,len(x[0])],
                dtype='float16')
68          for n in range(len(x)):
69              tmp =x[n][0].split('\\')[0:-1]
70              tmp.append('MelSpectrogram1.npy')
71              a,b = getImpactX(resize(np.load('\\'.join(tmp)),(shape
                    [1],shape[0])),t1,t2)
72              for i in range(len(x[0])):
73                  if x[n][i][-5:] in ['1.npy','2.npy','3.npy','4.npy'
                        ,'t.npy']:
74                      x_dat[n,:,:,i] = resize(np.load(x[n][i]),(shape
                            [1],shape[0]))[:,a:b]
75                  else:
76                      x_dat[n,:,:,i] = interpolate(np.load(x[n][i]),(
                            time_frame,shape[0]))
77          return x_dat
78
79      elif len(x[0])==1:
80          if x[0][-5:] in ['1.npy','2.npy','3.npy','4.npy','t.npy']:
81              for n in range(len(x)):
82                  tmp =resize(np.load(x[n][0]),(shape[1],shape[0]))
83                  a,b = getImpactX(tmp,t1,t2)
84                  x_dat = tmp[:,a:b]
85          else:
86              for n in range(len(x)):
87                  x_dat = interpolate(np.load(x[n][0]),(time_frame,
                        shape[0]))
88          return x_dat
```

```
89      else:
90          print("No data included")
91          return
92
93  def load_data(root_dir,save_dir,
94                measurements_per_class=None,split=None,
95                classList=None,included_data=None,shape=None,
96                t1=None,t2=None):
97      return_classList = False
98      if 'dataSetup.json' in os.listdir(save_dir):
99          measurements_per_class,split,classList,included_data,shape,
                t1,t2,mode = load_setup(save_dir)
100         split = 0
101         return_classList = True
102     if t1 and t2 is None:
103         time_frame = shape[1]
104     else:
105         time_frame = t1+t2
106     x_train, y_train, x_val, y_val = getPathList(root_dir,
            measurements_per_class,split, classList,included_data)
107     x_train = load_data_from_paths(x_train,shape,t1,t2,time_frame)
108     try:
109         x_val = load_dict[mode](x_val,shape,t1,t2,time_frame)
110     except:
111         print("No validation data")
112         x_val = None
113         y_val = None
114
115     if return_classList:
116         return x_train,y_train,classList
117
118     #save_setup(save_dir,measurements_per_class,split,classList,
            included_data,shape,t1,t2,mode)
119     return x_train, y_train, x_val, y_val
120
121 def getImpactX(spec,t1,t2):
122     if t1 and t2 is not None:
123         xMax = int(round(np.mean(np.argmax(spec,axis=1))))
124         return xMax-t1,xMax+t2
125     else:
126         return None, None
127
128
129 def save_setup(save_dir,measurements_per_class,split,classList,
        included_data,shape,t1,t2,mode):
130     setup = {'measurements_per_class':measurements_per_class,
131              'split':split,
132              'classList':classList,
133              'included_data':included_data,
134              'shape':shape,
135              't1':t1,'t2':t2,
136              'mode':mode}
137
138     with open(os.path.join(save_dir,'dataSetup.json'),'w') as file:
139         json.dump(setup,file)
140
141 def load_setup(save_dir):
```

```
142     with open(os.path.join(save_dir,'dataSetup.json'),'r') as file:
143         d = json.load(file)
144     return d['measurements_per_class'],d['split'],d['classList'],d[
        'included_data'],d['shape'],d['t1'],d['t2'],d['mode']
145
146 class Logger(object):
147     def __init__(self,save_dir):
148         self.terminal = sys.stdout
149         self.log = open(os.path.join(save_dir,"logfile.log"), "a")
150
151     def write(self, message):
152         self.terminal.write(message)
153         self.log.write(message)
154
155     def flush(self):
156         #this flush method is needed for python 3 compatibility.
157         #this handles the flush command by doing nothing.
158         #you might want to specify some extra behavior here.
159         pass
160
161
162 x_train, y_train, x_val, y_val = load_data(r'E:\Master\Measurements
        ',r'E:\Master','all',0.2,['MixPure', 'MetalAndMix','
        GlassAndMetal','GlassAndMix'],[1,1,0,0,1,1,0,0,1,0],
163                                            shape =(256,1379),t1 =
                                                128-64, t2 = 128+64)
164
165 x_test, y_test, _, _ = load_data(r'E:\Master\UnikeMeasurements\
        Measurements',r'E:\Master','all',0,['MixPure', 'MetalAndMix','
        GlassAndMetal','GlassAndMix'],[1,1,0,0,1,1,0,0,1,0],
166                                            shape =(256,1379),t1 =
                                                128-64, t2 = 128+64)
167
168
169 save = r'E:\Master\Paper\Model\Multilabel2classes\Ablation\
        NoWeightSmoothMetal'
170 if not os.path.exists(save):
171     os.mkdir(save)
172
173 np.save(os.path.join(save,'x_train'),x_train)
174 np.save(os.path.join(save,'y_train'),y_train)
175 np.save(os.path.join(save,'x_val'),x_val)
176 np.save(os.path.join(save,'y_val'),y_val)
177 np.save(os.path.join(save,'x_test'),x_test)
178 np.save(os.path.join(save,'y_test'),y_test)
```

### A.2.7 Batch File

```
py -3 main1.py
py -3 main2.py
py -3 main3.py
py -3 main4.py
py -3 main5.py
```

### A.3 Project Thesis

NTNU
Norwegian University of
Science and Technology

Project Thesis

# A Study of Data Gathering and Utilization For Improvement Purposes In Household Trash Collection

*Robert Marhaug and Oliver Istad Funch*

supervised by
Martin Steinert
Sampsa Matias Ilmari Kohtala
Heikki Sjöman

December 2019

**Abstract**

This project thesis' purpose has been to explore problems related to the trash collection cycle in Oslo, as well as develop methods for measuring the magnitude of the identified problems to estimate their significance. A brief theoretical background is given on the concepts used during the development. Through a Wayfaring process, several concepts has been tested through prototyping. The concepts that showed promise which will be given further attention is: metal detection to determine metal content of trash bags, object detection(machine learning) to count bags and detect and classify loose objects, sound detection(machine learning) to detect the presence of glass and spectroscopy using a dual-paraboloid constellation in order to create a concentrated beam. The final recommendation given to Renovasjonsetaten is to combine the aforementioned methods into a single system placed on the back of the trash collection truck. The back of the truck is seen as the location with most potential due to possibility of implementation as well as traceability to the household.

# Contents

# List of Figures

# 1 Introduction

The project is funded by Renovasjonsetaten in Oslo, the department for collection of household garbage. Sorting and recycling is performed by a separate department, but this assignment can also include these areas as the departments are soon to merge. The initial problem statement from the customer is as follows: Too little information is known about the garbage collected, especially considering to what degree the consumer is sorting their garbage properly. It is desired to collect more information to determine if this data might reveal useful insights for future improvement of the current collection system.

The task is not very specific, as the starting goal was to collect information of any kind from any place in the cycle deemed appropriate. The information gathered does not necessarily need to be targeted for a specific end use, as it can be hard to determine what kind of information will prove most valuable. The overall goal for Renovasjonsetaten is to increase the amount of trash that is recycled, in contrast to incinerated or deposited. The EU has set target goals for recycling at 50% within 2020, 55% within 2025, 60% within 2030, and 65% within 2035 according to Avfall Norge, the field association for recycling in Norway [18]. According to the customer, the current level is at about 35%.

## 1.1 Information gathering

Information from the customer might not yield the greatest insights into the problem, and the surroundings are best understood by seeing them in person. Therefore we went to Oslo to meet with Renovasjonsetaten and take part in their daily activity. The task was discussed in detail, and we took a tour of the sorting facility where the bags of different color are separated. As we would probably involve a garbage truck in our work, we inspected one thoroughly to assess possibilities for adding equipment as well as possible problem areas. We also spent half a day working as trash collectors, joining with one truck each on their daily collection route. Underway we collected video footage as well as talking to the workers to gain insights from them. The images in figure 1 are taken from this video footage.

(a) Side view from video footage

(b) Rear view from video footage

Figure 1: Video footage snapshots

## 1.2 Collection cycle

It is useful to have a clear overview of the whole garbage cycle, so the cycle was separated into stages, illustrated, and known problems listed.

Stage 1 in figure 2 shows home sorting done by the consumer. There are three categories: plastic, food waste, and miscellaneous/mixed waste. All plastic in the plastic bin should be clean. If it is not possible to clean it properly it should be deposited in the mixed waste. Metal, glass and hazardous waste are deposited in separate containers at specified locations. Problems here include indifference, meaning the consumer does not care about correct sorting, lack of knowledge about sorting, and discarding contaminated plastic without cleaning it first.

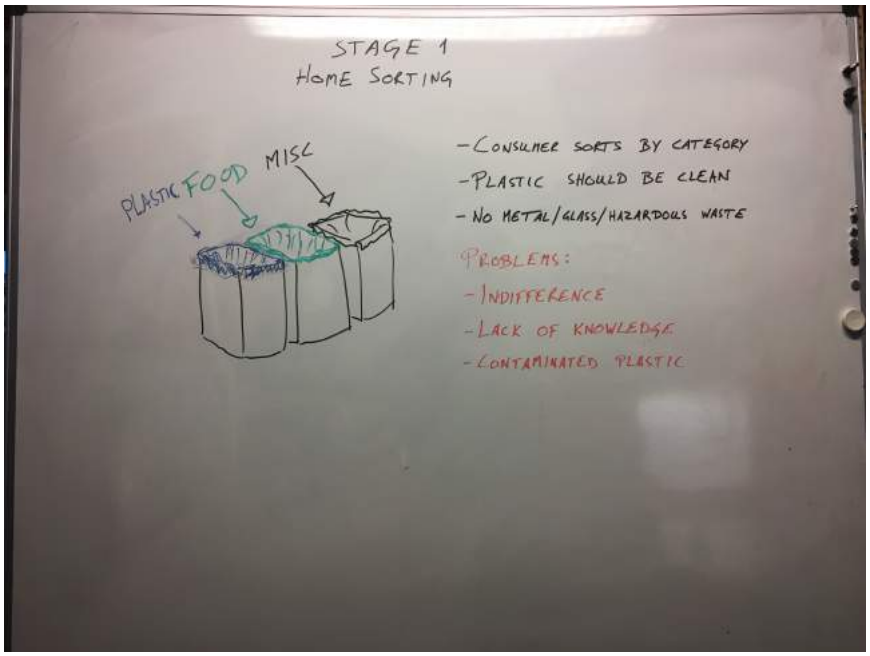



Figure 2: Collection cycle overview, Stage 1

In Stage 2 in figure 3, the consumer deposits the different colored bags into the same bin. This can be a private bin connected to their household, or a shared bin for an apartment complex. The bags should be tied with a double knot to prevent them from opening. Problems here include items that are not placed in bags, which should not occur, and items that do not belong, for instance garden waste and glass.

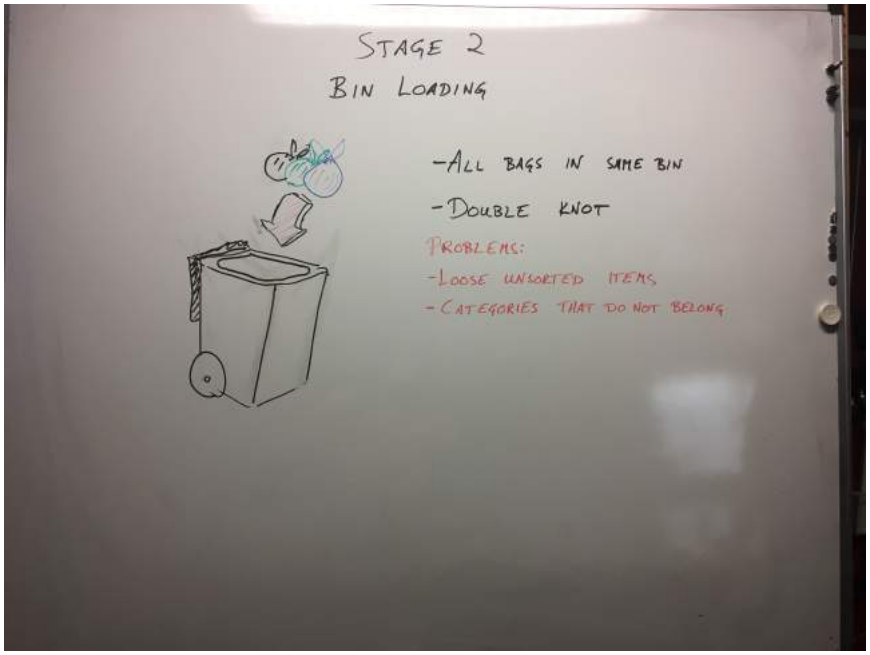



Figure 3: Collection cycle overview, Stage 2

At stage 3 in figure 4, the bins are unloaded into the back tray of the garbage truck. The unloading is performed by an operator that places the bin next to the loading arm, and operates the hydraulic arm to turn the bin upside down over the tray. The bins have RFID (Radio-Frequency Identification) tags which are read by a scanner on the loading arm, and the weight of the bin is also measured by equipment attached to the arm. The bags might be damaged at this stage when falling from the bin into the tray. This is also the last stage where the bags can be traced to a specific bin, which can be done using the information in the RFID tag.

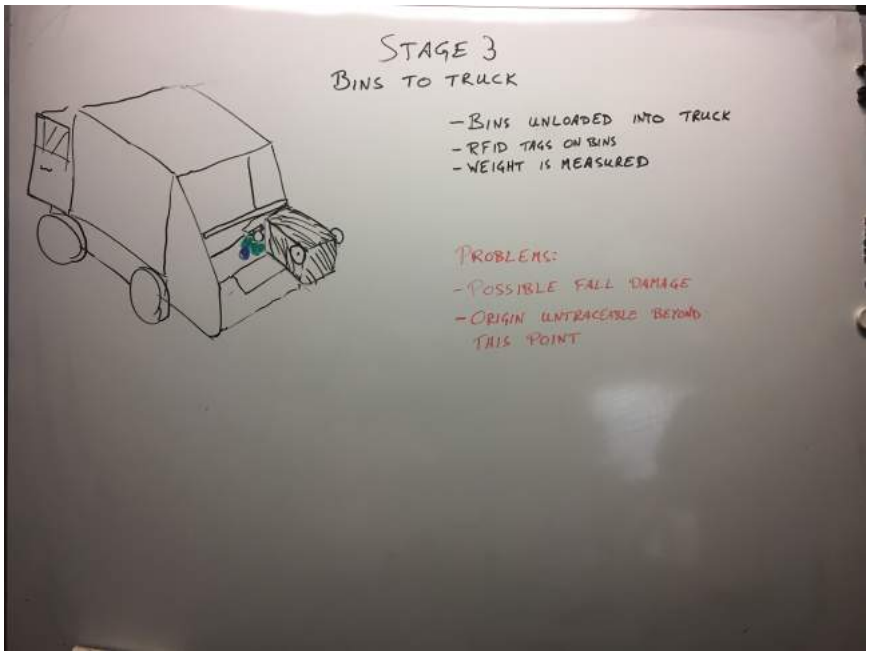



Figure 4: Collection cycle overview, Stage 3

At stage 4 in figure 5, the trash is moved by a hydraulic arm to the interior of the truck. Inside the compartment there is a wall that the trash is compacted against. As the amount of bags increase, the wall moves to make room for more. The wall automatically moves when a certain pressure against it is registered. Problems here include high forces from the compression, which could lead to bag rupture, and potential contact with sharp objects both from parts of the truck and objects in the bags. Sharp objects could be a cause of bag rupture.

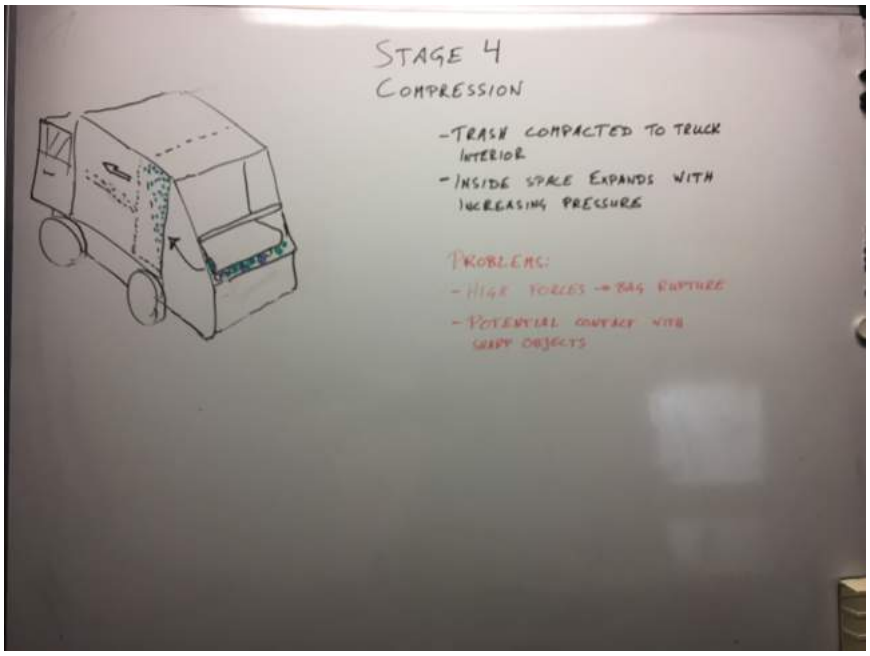



Figure 5: Collection cycle overview, Stage 4

In stage 5 figure 6, the truck unloads its cargo into the sorting facility. The back of the truck opens and the interior piston pushes the bags out of the truck. The bags might receive fall damage during the unloading, or from being pushed out of the truck. There is also the potential for damage caused by sharp objects.

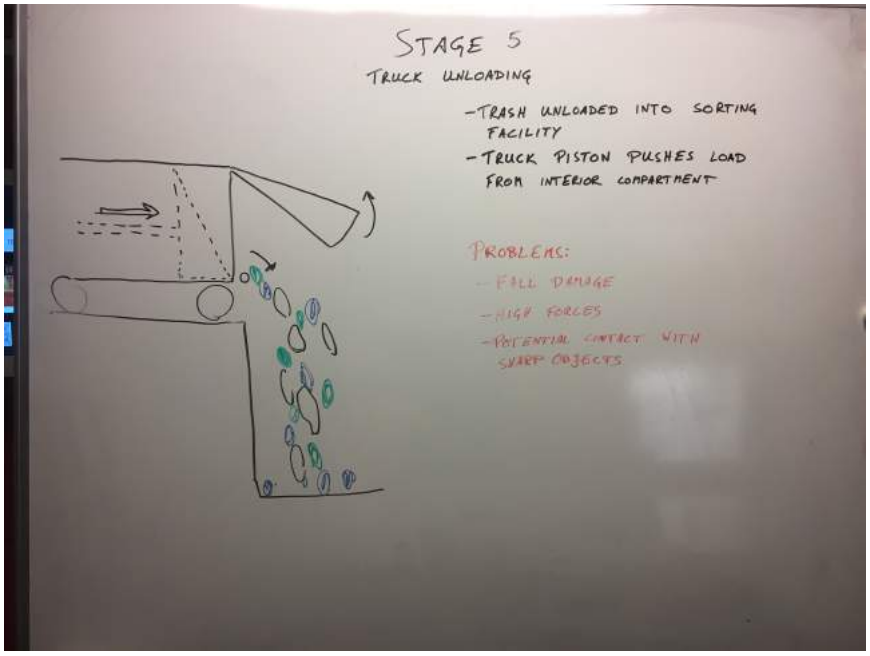



Figure 6: Collection cycle overview, Stage 5

The trash is then moved into the sorting facility in stage 6, figure 7. This is done on a surface where separate segments move at different times. This surface is called a "walking floor". In this stage there is much movement of the bags relative to each other which may cause damage and/or rupturing. The pile of trash is very large, so it is subjected to high forces during the movement.

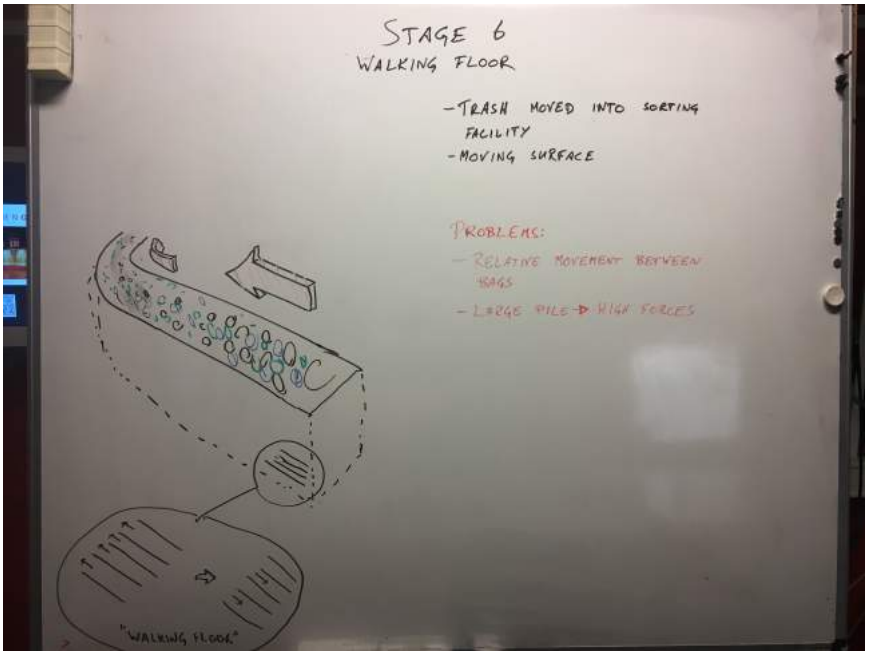



Figure 7: Collection cycle overview, Stage 6

In stage 7, figure 8, the trash passes through a filtering system designed to separate the objects by size. The tilted surface shakes, causing the items to move downward. Along the surface there are gaps of different sizes where the objects fall through if they are smaller than the gap. Many loose objects are removed at this stage as they are smaller than the bags. Large objects are also separated as they remain at the end of the stage. There is much movement between objects, possibly causing damage, and very small bags might be filtered out as loose objects and therefore not go through the appropriate sorting system later.

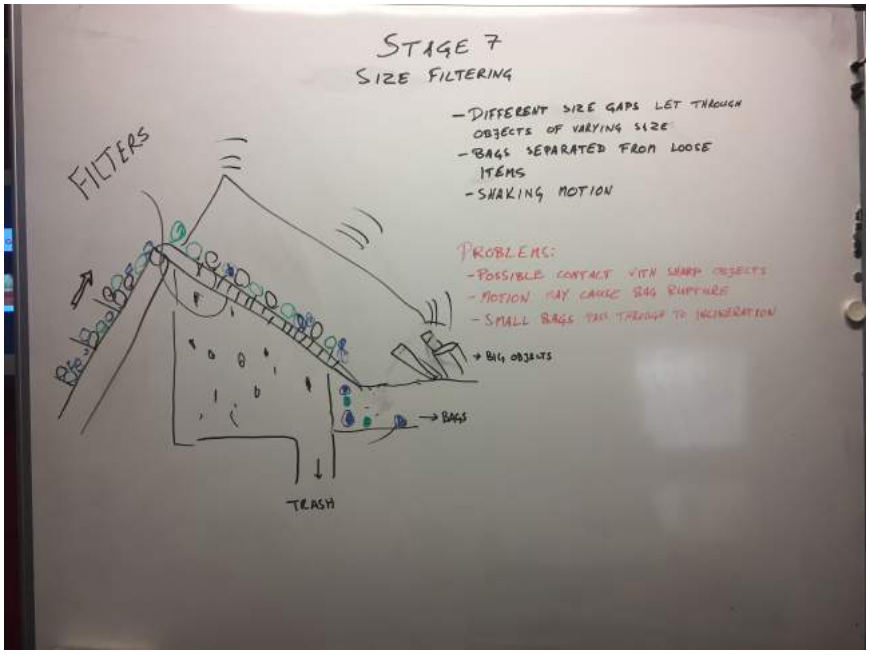



Figure 8: Collection cycle overview, Stage 7

At stage 8 figure 9, the bags are sorted according to color. The green bags are sent to bio gas production and the blue bags go to another sorting step shown in stage 9.1. The bags with mixed waste continue through. These sorters are not perfect and sometimes miss bags of the correct color. The bags are removed using a moving mechanical arm that shoves them from the conveyor belt, and if the bags are close to each other, a wrong bag may accompany the bag that is to be removed.

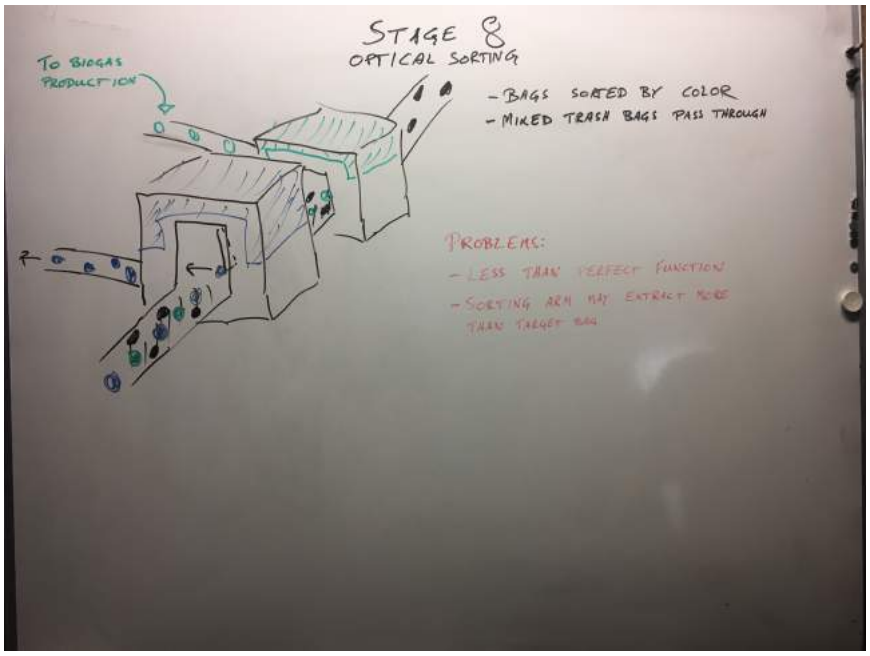



Figure 9: Collection cycle overview, Stage 8

Stage 9.1 figure 10 shows the next step in the sorting for the blue bags containing plastic. The bags pass by a fan blowing air. If the bag is light it is blown into a chute, and if it is heavy it drops down onto a conveyor belt that transports it to the mixed trash path. The bags passing through the chute are considered pure plastic and packed before shipping to Germany for recycling. The purpose of this stage is to remove heavy bags, as the added weight is often caused by other materials than plastic being present. The criteria here is, according to Renovasjonsetaten, too strict and often removes bags containing only plastic, so some recycling opportunity is lost. Also, the fan often gets contaminated by debris so its function degrades. Therefore it requires regular cleaning for it to operate satisfactory. This sorting method also does not account for other light materials that should not be present, like paper or cardboard.

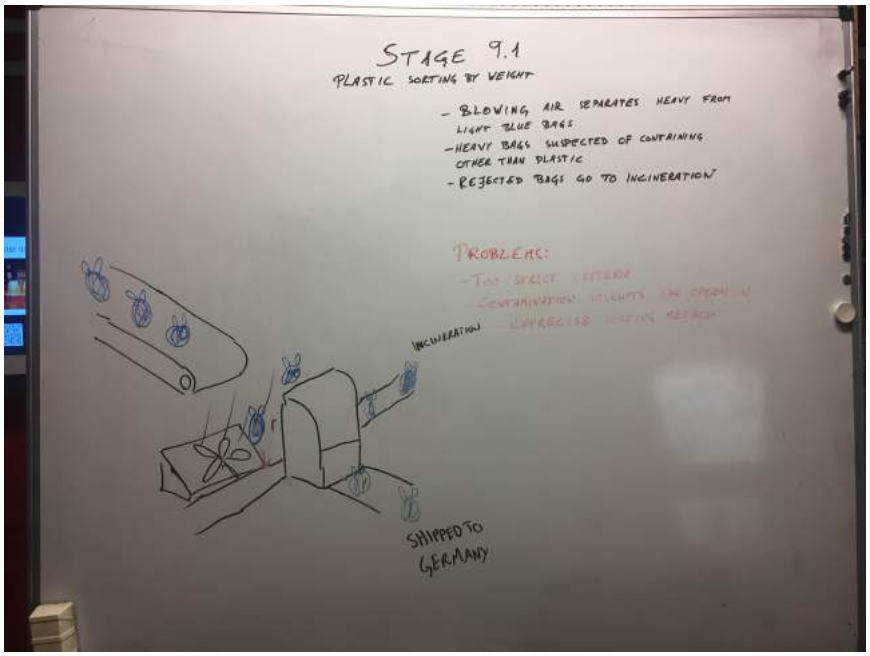



Figure 10: Collection cycle overview, Stage 9.1

Stage 9.2 in figure 11 is the final stage of the sorting. Here, everything not sorted into plastic or food waste earlier is passed through to an incinerator. Due to insufficient sorting at earlier stages, there are items at this stage that should not have ended up in the incinerator. These include unsorted bags, garden waste, electronics, and more. Some of these items will not be thoroughly incinerated and will remain afterwards, like for instance glass which has a high melting temperature and even higher burn temperature. The loose items from ripped bags also end up at this stage.

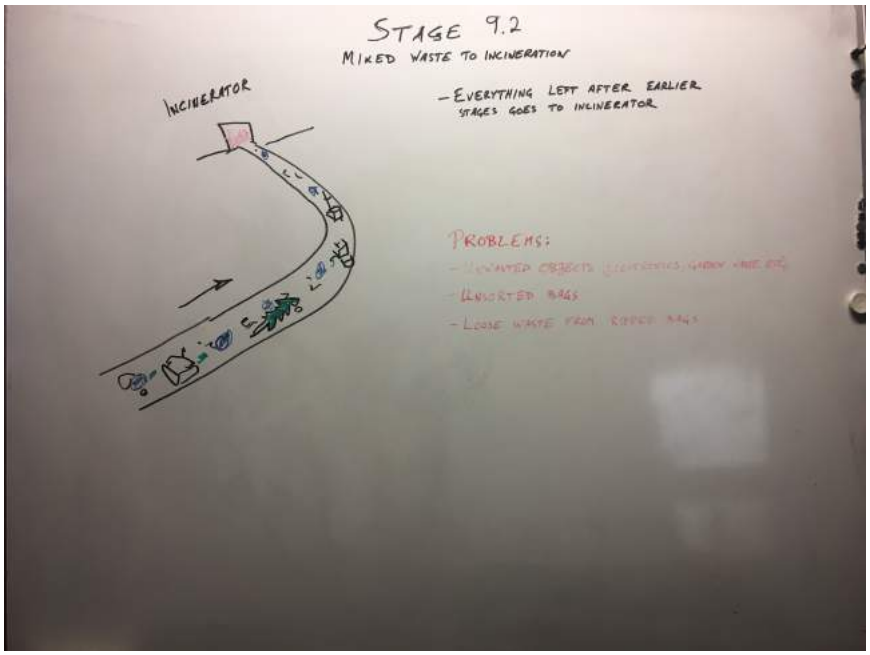



Figure 11: Collection cycle overview, Stage 9.2

## 1.3 Revised problem statement

The problems described in each stage in section 1.2 are mainly based on discussions with employees at Renovasjonsetaten but also on observations made when working as trash collectors and visiting the sorting facility. It is however unknown to REN which problems are the most prominent and which areas should be focused on. A fact-based decision may only take you as far as the facts go. As such, this projects mission has been to explore and develop methods for gathering information at several stages in order to determine the proposed problems magnitudes.

Detecting the magnitude of several of these proposed problems rely on the same fundamental sub-problem being solved:

*Wrongly sorted material:*
The significance of the indifference and contaminated plastic in stage 1, categories that do not belong in stage 2(which leads to categories not belonging in stage 3), too strict criteria in stage 9.1 as well as unwanted objects in stage 9.2, all relies on classification of material inside and outside of bags. The reasons these are listed as problems are that when non-coloured bags contain plastics or food waste that should have been sorted in their respective bags, it's a lost opportunity to increase the percentage recycled. Furthermore, plastic bags containing uncleaned plastics or other items causes the plastic recycling facility in Germany that receives our plastics to down-prioritize it. This results in a lower return for the shipped plastic bags.

*Bag ruptures:*
Having a system for determining forces, pressures, impacts and ruptures on bags as well as where it happened may determine the significance of problems in stage 3, 4, 5 and 6. The reason forces, pressures, impacts, relative motion and possible contact with sharp objects are listed as problems in these stages, is that if blue and green bags are ruptured such that their content spills out, they are not being sorted. Additionally, as noted by employees at REN, when green bags rupture food waste spills over plastic bags and thereby lowers its recyclability as mentioned above.

*Loose objects:*
Lastly, the magnitude of the problem of loose items in stage 2 (leading to loose items in later stages) may be estimated by somehow determining the amount of loose items vs trash bags on several stages. Loose items were listed as a problem due to all of it going to incineration regardless of its material, so items that could have been recycled will be burned instead; again a lost opportunity to increase the percentage recycled. Also, it was noted that there was a high amount of loose trash inside the sorting facility that was not contained in a bag. This might be the source of ruptured bags though it can also cause problems for the machinery in the facility due to clogging and contamination. An example

of this problem can be seen in figure 12.

Figure 12: Loose trash in sorting facility

The problems listed in the stages in Section 1.2 are largely interrelated. Also, some of the problems listed are rather root causes to other problems, see figure 13.

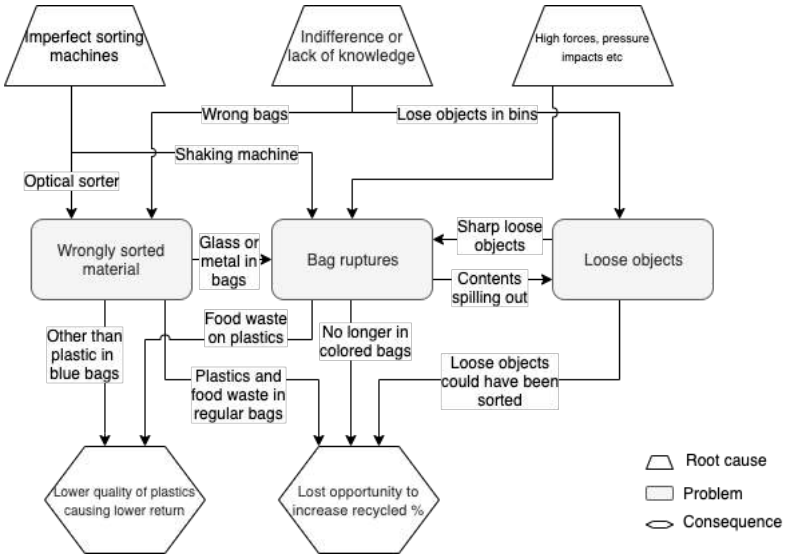



Figure 13: Root cause of problems and their consequences

If one could determine the magnitude of the problems described in the rounded squares in figure 13, one could possibly gain insights into which area(s) one should focus on improvements. Note that all the problems, causes and consequences depicted in this figure were not known at the onset of the project, but was rather discovered throughout the development. Also, the figure does not presume to illustrate the whole problems-, root causes- and consequences-situation at Renovasjonsetaten, but is a proposal of areas that should be explored further.

# 2 Background and Theory

This chapter aims to give a basic understanding of the development process used as well as the theory behind concepts that are presented throughout this document. Additionally, it provides meaning to terms that will be used in later chapters.

## 2.1 Design Methodology

### 2.1.1 Prototyping

The Greek word "prototype" meaning "primitive form" has been used extensively in relation to many design tasks lately. It can be described as the primitive version or first("protos") impression("typos") of a product. It has been commonly associated with a physical representation of a product that tests its physical properties. Lately, both when computer software rose and CAD software were introduced, the meaning has changed. A prototype according to Ulrich, Eppinger and Yang[28] is "an approximation of the product along one or more dimensions of interest" with the dimensions of interest being focused vs. comprehensive and analytical vs. physical. A focused prototype aims at testing a few functions extensively rather than the interplay of properties as a comprehensive prototype does. A physical prototype is, well, physical, while an analytical prototype may be a computer simulation of the product or a sketch for that matter. In other words, a prototype according to Eppinger et al. aims to test or get an impression of one or more aspects of a product, be it design, function(s), form, integration etc. "Prototyping" is then the act of creating such an approximation.

In the sentence "an approximation of the product..." it is assumed that the idea of the product already exists and one knows how much the prototype approximates it. As such, this definition may refer to prototyping in a relatively late stage of the product development process. Lim et al. (2008) [17] defines prototyping as follows: "...a prototype is fundamentally different from the final product, whether or not it is identical to the final product. Prototypes are means and tools for design and are not the ultimate target for design.". In other words, prototyping can be described as a means to define the product itself, to explore possibilities and clarify problems.

### 2.1.2 The Hunter-Gatherer Model based on Wayfaring

The hunter-gatherer model based on wayfaring [25] addresses the issue of managing innovation in the fuzzy front end stage. That is, before requirements are set, before the great idea is discovered and before it is known what to actually make. This stage cannot be guided by a step-by-step process often mentioned when talking about product development. The name of the Hunter-Gatherer Model comes from an analogy to when humans lived in a hunter-gatherer society. When hunting, little was known of the prey's exact location. The hunters

would have to venture out and find their way as they went, tracking down the prey. The prey is the "next big idea" in this regard. The location of this idea, or rather what the idea is, cannot be known before it is discovered. This unknown requires abduction to be discovered, that is, it starts with observation followed by a search for an explanation. As such, designers should embark on an iterative journey of cycles with abduction/prototyping, testing and learning in order to get ever closer to the next big idea. Figure 14 illustrates this journey.

In the model, three rules are proposed:

*Human Rule, 1) 'never go hunting alone'*: Hunting in teams consisting of people with various skills (tracking, killing, weather prediction, etc.) increases the chance of success. In terms of product development, having small agile teams with a diverse set of skills that fulfill each other, all equipped with tools and training increases the chances of finding the idea.

*Ambiguity Rule, 2) 'never go home prematurely'*: Even though the hunt may have seemed to yield no result, the prey might lie around the next corner, or, a new prey may present itself. On the product development side, this means that the team should not settle for a 'thank you' result even though the long journey and ambiguity starts to puncture their spirits. Rather, change focus and enter an entirely new concept and solution space and keep hunting for the big idea.

*Re-Design Rule, 3) 'bring it home'*: The prey is located and the killing blow is struck, yet the hunt is not over. The prey must be brought home. Now that the big idea is located, it is time for the well known linear step-by-step optimization of the product to begin.

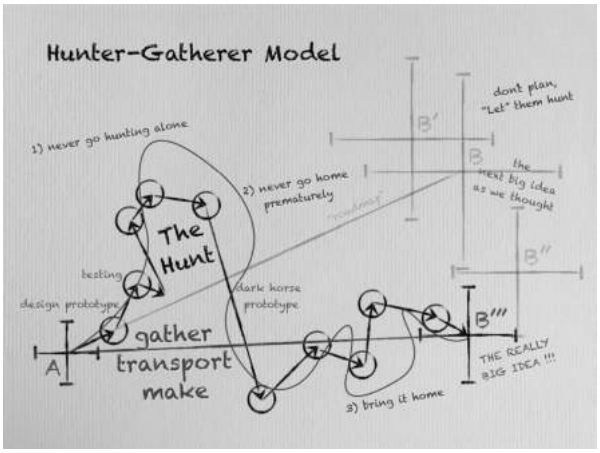



Figure 14: Hunter-Gatherer Model

## 2.2 Technology Review and Theory

### 2.2.1 Machine learning - Object Detection

Machine learning has become a popular tool in recent years. Although it has existed for a while, it isn't until now that the increased computing power as well as the accessibility of large amounts of data has unlocked its true potential[15].Paraphrasing of Arthur Samuel, generally regarded as the pioneer of machine learning, often goes as follows: "Machine learning is the field of study that gives computers the ability to learn without being explicitly told" [22]

It has become commonplace to distinguish between supervised- and unsupervised learning forms of machine learning. Both aim at making meaning out of data given to it. With unsupervised learning algorithms there are no specific labels or categories such that there is no answer to be given in advance. This type of algorithm tries to make meaning out of the data by finding rules, detecting patterns and grouping data points. As supervised algorithms are more common and are commonly used when performing object detection, unsupervised algorithms will not be discussed further.

With supervised learning, the algorithm is given earlier occurrence of what it tries to predict. When training such an algorithm, the correct answer to a question is given in advance such that the code may "learn" how to reach the answer on its own. In other words, it receives a series of inputs with their associated outputs such that it may create "rules" in order to map the inputs to the outputs in a way that best fits all the data given.

There are two sub categories of supervised learning, regression and classification. With regression, the outputs are continuous values whereas with classification, they are discrete. As exemplified by Andrew Ng on Coursera [24], predicting house prices based on their sizes is a regression problem, whereas whether the house sells for more or less than the asking price, is a classification problem.

With a neural network(NN) the learning described above takes place in the form of altering the synaptic strength(i.e. weight, w) between nodes across layers in a neural network, see Figure 15. This weight, for instance $w_{11}^1$, controls the influence the input $x$ has on $a_1^2$.

Following figure 15 the value of $a_1^2$ is determined by first multiplying its inputs with their respective weights and adding them together $z_1^2 = xw_{11}^1 + b_1^2$, and then processing the result in an activation function $a_1^2 = \sigma(z_1^2)$. This will in turn be the input for nodes in the next layer.

There are several different types of activation functions. Its main objective is to introduce non-linearity such that the network may solve more than trivial

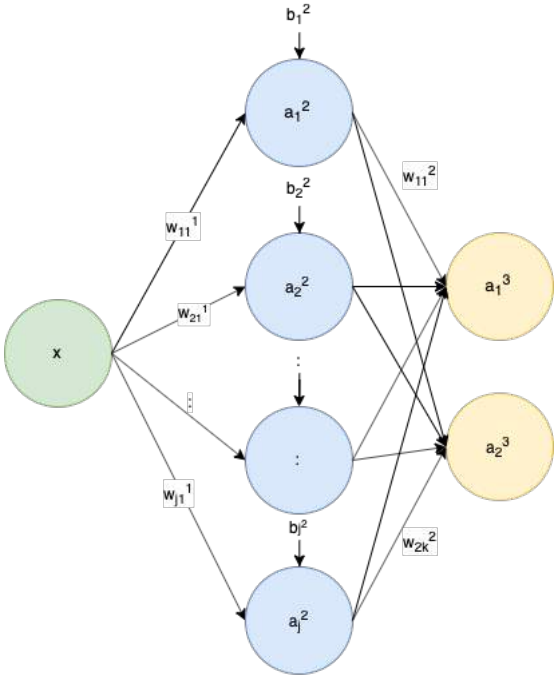



Figure 15: Single Layer, Fully connected neural network

tasks.

The convolutional neural network(CNN) has been the state-of-the-art network within image classification ever since Krizhevsky et al. [15] outperformed all others in the annual LSVRC in 2012 (They go by the name "SuperVision", but often referred to as AlexNet)[16]. Its essential building blocks consist of convolutional layers, activation layer rectified linear units(ReLu), pooling layers and fully connected layers[7].

The convolutional layers are the cornerstones of the CNN. They contain a specific amount of filters (small in size) that are not pre-defined but are to be learned. For instance, a Sobel filter/kernels accentuates contours in an image by iterating through all pixels of an image and calculating the approximate derivative of pixel values by using two 3x3 filters (one for x and one for y-direction) and the local 3x3 pixels found at present iteration. These calculated values are then added and put in a new matrix which is the resulting convolution, see appendix figure 44. Such a filter is pre-defined, appendix figure 43. In a CNN network, these filters are defined based on the data passed on to them during training and will usually be able to detect many different types of features. Here, there are several filters, all producing an output matrix. These matrices are processed in an activation function (ReLu being popular) which results in an activation

map. These activation maps are stacked together producing the output and thereby the input to the next layer, which eventually is a new filter. As filters are succeeded by new self-learned filters, they become more complex and able to detect more abstract patterns in the image. The reason a filter takes a specific form, is to support the final decision at the last layer. Hence, the filters can be seen as the weights between convolutional layers. The stronger a filter reacts to a specific part of the image the more likely the numerical value is to propagate further through the network. As such, these filters extracts features worth examining in order to classify objects.
The pooling layers "pools" together values in the output matrix from a convolutional layer, for instance by calculating the average value of a 2x2 matrix and substituting it with one pixel value. They lower the resolution such that small changes to an image containing the same feature would not result in a completely different activation map[5] and thereby reduces overfitting.

Overfitting happens when a network is highly specialized for its input causing the algorithm to be very good at predicting output of training data but cannot be used on other data. Dropout layers and data augmentation [20] are two ways of reducing this phenomenon. In order to test for overfitting, often 20% of the data is set aside before training such that the algorithm may test it self on never-seen-before data.

The final fully connected layers as shown in Figure 15 will determine the probabilities each image has of belonging in each class. Classification happens when a probability surpasses a pre-defined threshold set by the programmer. As such, object detection through CNN is a regression problem until the very end.

The AlexNet[15] as well as the LSVRC challenge only concerns itself with image classification, that is, detecting what classes might reside in the image and not their locations. Object localization is another task that may also be performed by a CNN network. This is usually done by segmenting the image into sub-images containing one object each, and then classifying these sub-images.

You Only Look Once(Yolo) is a method for localizing objects and classifying them (thereby object detection) that is based on a CNN[20]. Though here, the whole network performs both tasks in one network iteration (hence the name) by pre-segmenting the image and calculating both class probabilities and bounding box probabilities. The total probability of an object and its location is the two probabilities multiplied. The model uses 24 convolutional layers followed by two fully connected layers. The convolutional layers extracts the features while the fully connected layers predicts the objects present and their bounding boxes. Its architecture is shown in Figure 16.

When predicting the bounding boxes, four parameters are predicted per object. $x, y, w$ and $h$ describes the position$(x, y)$ of a box defined by the width$(w)$

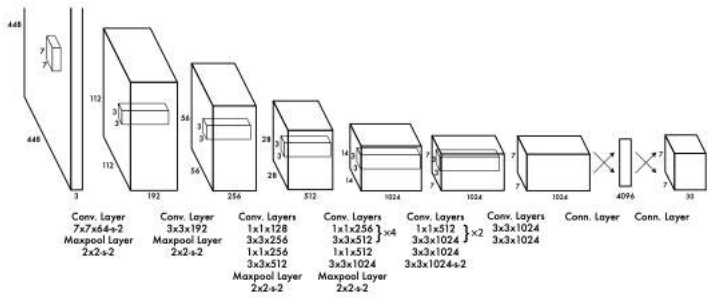



Figure 16: You Only Look Once Architecture

and the height($h$) of the box. When training on a data-set, these parameters must be given in advance for each object in each image. Yolomark is an intuitive GUI[2] made to mark images and autogenerate a .txt document containing these parameters used to train. See Figure 17.

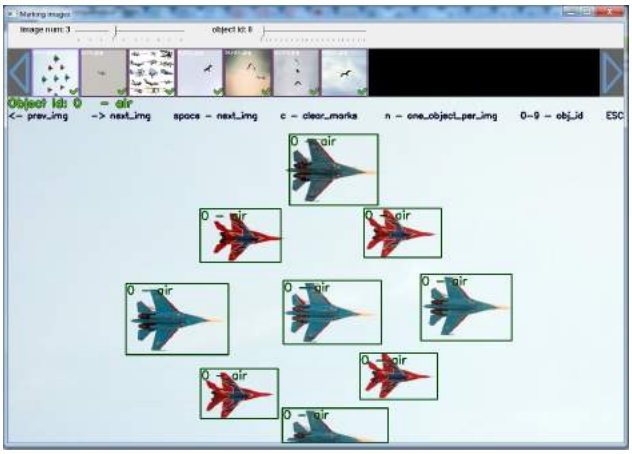



Figure 17: Yolomark GUI

During training, the algorithm tries to minimize or optimize for a certain loss function. In statistics, a loss function maps the relationship between predicted values and actual values. If the loss is small then the prediction is accurate. In Figure 18 the red arrows illustrates the loss. As the function is fitted to the data the red arrows(the loss) becomes smaller(right). Yolo optimizes for the sum squared error function.

The precision of an algorithm is usually of interest. It is described as correct

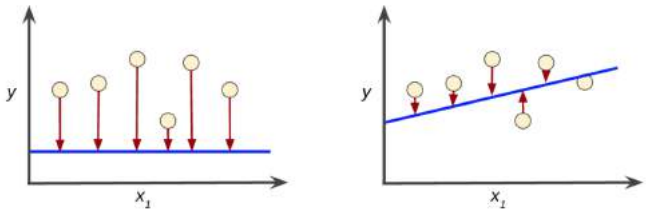



Figure 18: Loss [6]

predictions vs all predictions that should have been made, or, true positives divided by (true positives + false positives). As such, the closer to 100% the better. In Yolo, precision is based on the same principle, though here it is the area of the intersection between the detected box and the actual box (the area that was correct) divided by the area of the actual box. See Figure 19.

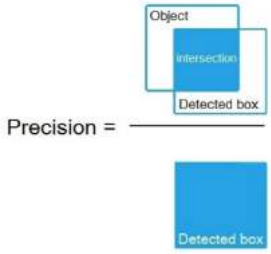



Figure 19: Precision [1]

Mean average precision(mAP) is a much used metric in order to express a networks performance. It is specifically used to determine the performance in the PASCAL VOC challenge [27]. mAP is much of the same as described above, although it is an average of all the average precisions of the individual classes.

### 2.2.2 Machine learning - Sound Identification

Sound identification works much in the same way as object identification described in the section above. The difference lies in converting sound files to a format that the machine learning algorithm can use, and this is done by creating an image that describes the sound file. The process used is derived from a project by Mike Smales, "Classifying Urban sounds using Deep Learning" [23]. The complete code used can be reviewed in the appendix. The code converts each sound file to an image which is a Mel-Frequency Cepstral Coefficients Spectrogram [30], hereby referred to as an MFCCS. The MFCCS shows the energy in different frequency bands at different time segments in the audio file, see figure 20 for an example. This conversion is performed using the python

package Librosa. Setting parameters will result in a consistent x-axis for every image, but the y-axis will differ based on the length of the sound file. This will present a problem at a later stage because the algorithm requires similar divisions for comparison, and must therefore be corrected using the function defined "interpolate". This segment changes the dimensions of the y-axis to a consistent value by either adding new values derived from the current ones, or compressing the existing values. Further processing is done in the same way as for image classification described earlier.

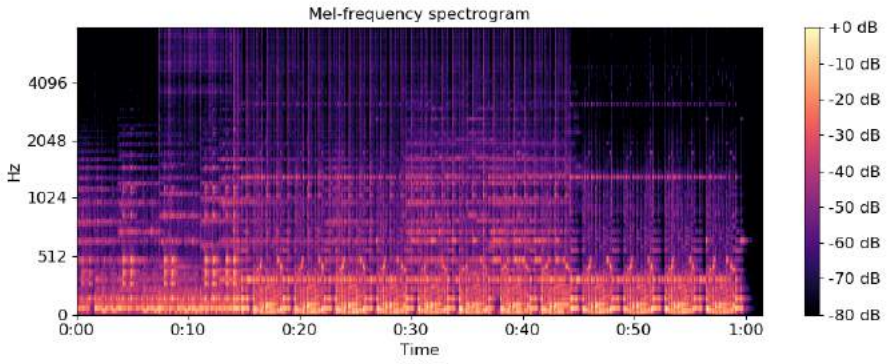



Figure 20: MFCC Spectrogram example [26]

### 2.2.3 Metal detection

Metal detection in this project is based an instructables.com tutorial by user rcgo [21]. The sensor consists of an Arduino, a diode, a resistor, a capacitor and a coil. The circuit is shown in figure 21. When the coil experiences a change in current, a magnetic field is induced in the coil. According to Faraday's law of induction [8], this changing field induces an electric field that opposes the change. The voltage from this field is related to the coils inductance which is indirectly what the circuit is measuring. The voltage relation to the inductance is described by equation 1, derived from Faraday's law where v is voltage in Volts, i is current in Amperes and L is inductance in Henry's.

$$v(t) = L\frac{di}{dt} \tag{1}$$

If the coil is in near proximity to metal, the inductance will change depending on the metal. Ferromagnetic metals will increase the inductance as their magnetic field will align with and add to the field of the coil. Non-magnetic metals will cause it to decrease due to induced eddy currents in the metal which opposes the field of the coil.
Measuring the inductance is done using the coil as part of a high-pass filter and sending a block-wave current through from the A0 pin. The current creates a

magnetic field in the coil which will induce a voltage when the current is switched off. This voltage spike has a duration proportional to the coils inductance. Because these spikes are very short they are difficult to read with precision due to the Arduinos clock frequency of 16MHz. To overcome this problem, the pulses are instead used to charge the capacitor and this charge is measured on the A1 pin. After the measurement is made, the capacitor is discharged before the next measurement. For greater accuracy, several measurements are recorded and the result is averaged. The charge is not linear to the inductance of the coil, and the inductance is also very dependent on the orientation and proximity of the metal. Therefore the measured value cannot be used to discern the type of metal or the mass of the object. The detector is therefore only usable for detecting if metal is present or not.

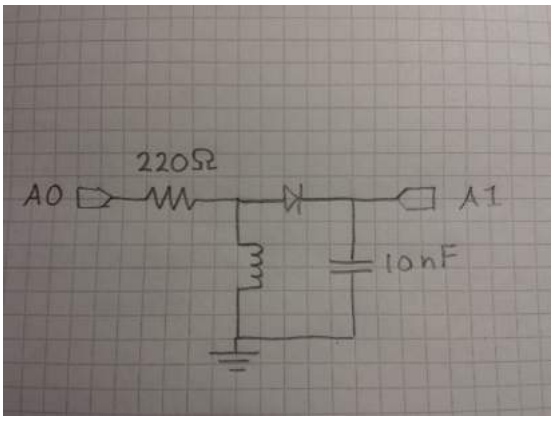



Figure 21: Metal detector circuit diagram [21]

### 2.2.4 Force sensing

Force or strain sensing can be performed using a composite material consisting of chopped carbon fibers in a silicone rubber matrix. The procedure used is similar to the one described in "Highly Sensitive, Stretchable Chopped Carbon Fiber/Silicon Rubber Based Sensors for Human Joint Motion Detection" [4]. Carbon fiber is first heated to 450°C to remove epoxy, which is not a desired part of the composite. The fibers are then chopped into pieces with length around 1mm, as the desired result is randomly oriented strands of fiber. These pieces are then mixed with silicone rubber and cast in a mould to the desired dimensions. This composite functions as a piezoresistive sensor. When the material is at rest there exists connections between the randomly distributed carbon fiber pieces, and when the composite is subjected to strain these connections change, causing the resistance of the material to change. This is illustrated in figure 22. By passing a current through the composite the resistance can be measured and used to determine the applied strain, which in turn can be used

to determine the force applied.

According to the tests performed in the paper, the minimum amount of carbon fiber is about 1.47wt%. Amounts less than this value resulted in connection loss through the composite, due to the conducting fibers being too far apart. There is also a maximum strain value, where exceeding the value will not result in changing resistance. This is because the fibers move too far apart for conduction to happen, and is shown in the far right part of figure 22.

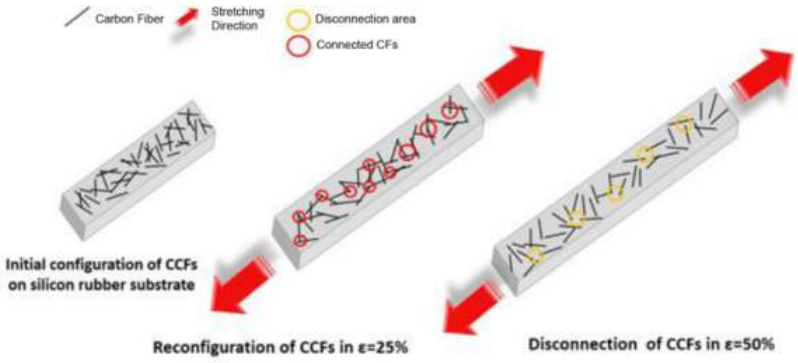



Figure 22: Reconfiguration of fibers due to strain application [4]

### 2.2.5 ESP8266 NodeMCU v3

ESP8266 is a microchip produced by Espressif Systems capable of WiFi-communication through TCP/IP. By connecting it to a microcontroller it allows the microcontroller connection to internet. There are several different modules of the ESP8266, the first being ESP01. This module, as most of the modules, has its own sets of inputs and outputs, and is therefore capable to act as a microcontroller itself. The NodeMCU v3 is a later module also equipped with a set of inputs and outputs.
These microcontrollers are as easy to use as an Arduino. In fact, they may be programmed through the Arduino IDE by installing the ESP8266 Add-On, after which the regular Arduino WiFi [11] library may be used. Through Attention(AT) commands, which is a specific command language used to produce operations at wireless modems, one can configure the ESP8266 to either be an access point(server), a client, or both an access point and a client [10].

On an ESP8266 configured as an access point setting up the network is easily done with the following line of code:

```
1        WiFi.softAP("TrashNet", "abcd1234");
```

Where "TrashNet" refers to the SSID given to the network while "abcd1234" refers to the password. On an ESP8266 configured as a client the following line of code

```
1        WiFi.begin("TrashNet", "abcd1234");
```

will connect it to the network created by the server.

Received signal strength indication(RSSI) is a measure of the strength of the signal. The higher the number, the stronger the signal. Thus, when values are represented in the negative form, values closer to zero amounts to a higher RSSI. This value can be displayed using the line:

```
1        Serial.println(WiFi.RSSI());
```

The ESP8266 produces a 2,4GHz signal which is about 12,5cm wavelength.

### 2.2.6 Spectroscopy

Spectroscopy refers to the study of the interaction between materials and electromagnetic radiation. There are several types of spectroscopy. One may for instance concern oneself with what happens when a material is beamed by a specific radiation. When radiation is incident upon an object, a part of the radiation energy is reflected, another part is transmitted and the last part is absorbed [9]. That is

$$\rho_\lambda + \tau_\lambda + \alpha_\lambda = 1 \tag{2}$$

where $\rho_\lambda$ refers to the amount reflected, $\tau_\lambda$ refers to the amount transmitted and $\alpha_\lambda$ refers to the amount absorbed. The sub-scripted $\lambda$ denotes the dependency this balance has on the wavelength of the incident radiation. In other words, the amount reflected, transmitted and absorbed by the object may be different

when for instance microwaves are the incident radiation as opposed to when infrared waves hit the object. These values usually differ with different materials such that a specific profile or balance may be obtained for a material if these properties can be determined.

### 2.2.7 Parabolic Reflector

A parabolic reflector is a shape often used to collect and focus incoming energy into a single point. The purpose can be to strengthen radio waves coming from a specific direction or it may be used for other purposes, such as focusing heat waves in order to generate heat for a heat stove. Through its parabolic shape described by the equation

$$4fy = x^2 \tag{3}$$

where x and y are Cartesian coordinates and f is the focal length, it focuses the incoming radio waves upon the focal point F with a distance f from origin. See figure 23.

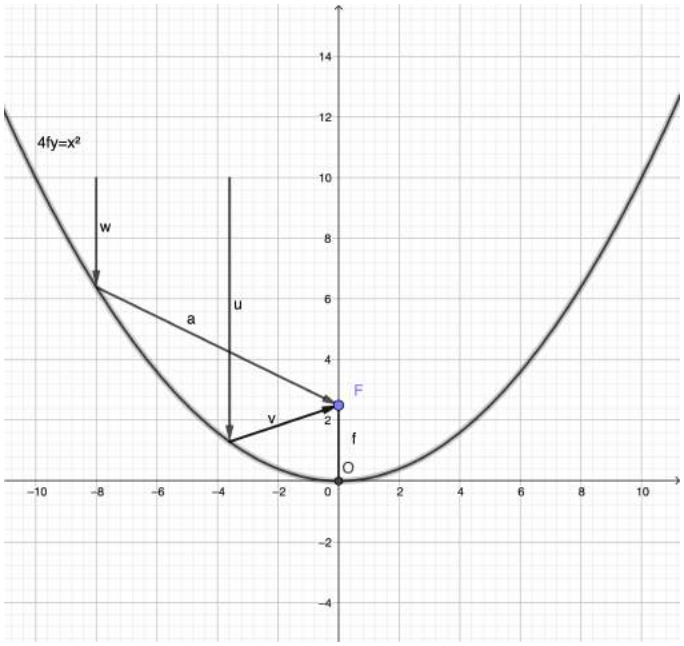



Figure 23: Parabola with incoming rays,
horizontal axis: x, vertical axis: y

Due the nature of waves having the same outgoing angle as the incoming angle when reflecting off of surfaces [9], combined with the shape of the parabola, the vectors w and u are directed towards F. This is true for all rays travelling parallel to the y-axis hitting the parabola. Following this, the opposite must be

true as well. A circular wave generated at point F is redirected along the y-axis, forming a collimated beam. That is, parallel rays that will spread minimally as it propagates along the y-axis. See figure 24.

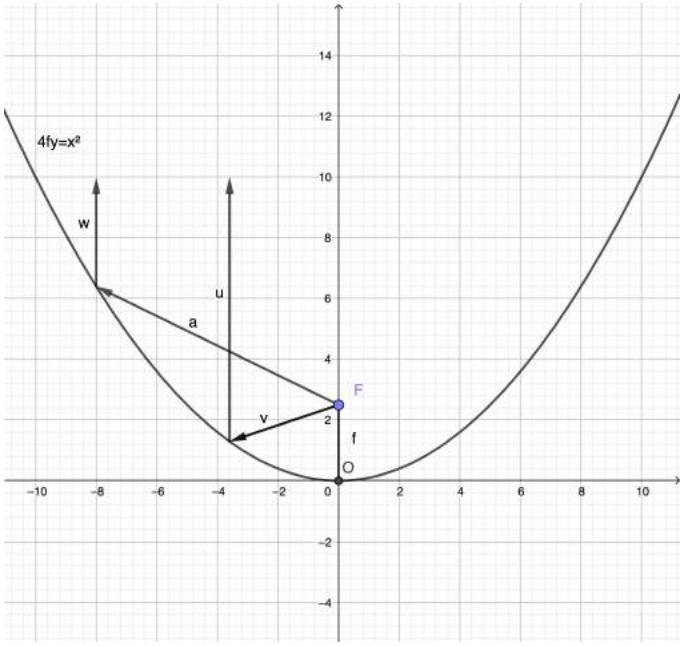



Figure 24: Parabola with outgoing rays,
horizontal axis: x, vertical axis: y

By revolving the parabola around the y-axis, a paraboloid is created. This is a 3-dimensional version of a parabola holding the same properties as described above. Here a spherical wave generated at point F can be redirected to form a collimated beam.

# 3 Development and Discussion

In this chapter several prototypes will be presented as well as the motivation for creating them and a discussion of their application. It should be mentioned that when using the terms "prototype" and "prototyping" it refers to the definition described by Lim, Stolterman and Tenenberg[17] in Section 2.1. The Hunter-Gatherer Model described in Section 2.1.2 has been applied to the development process to a large degree. That is, prototyping has been used extensively to explore the solution space, generate knowledge and provided guidance for the next step.

## 3.1 Metal detector

### 3.1.1 Motivation

In general, there should be no metal present in the trash collected in Oslo as metal is supposed to be recycled at specified collection points. Some metal is to be expected in the bags containing mixed waste, but at present there are no good estimates for the amount. Metal content in the blue bags will result in contamination of the plastic to be recycled, and metal in the green bags will contaminate the process of producing bio gas from food waste. Sharp metal can also cause the bags to rupture which is undesirable. Also, the incineration process of mixed waste happens at a temperature too low to incinerate metal, so any metal present will remain afterwards. It is therefore of interest to be able to quantify if and how often metal is present in the system.

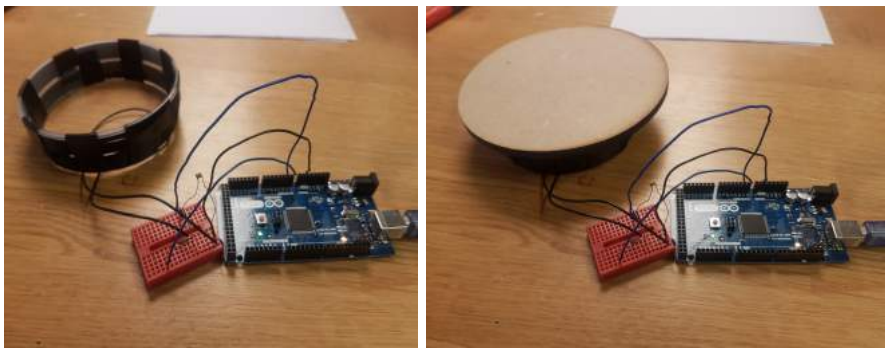
### 3.1.2 Prototyping

For this application, a simple metal detector was built with the purpose of logging the presence of metal, see figure 25a. The detector in question was built according to an instructables.com guide by user rcgo [21] and its working principle is covered in section 2.2.3. The development was simple, using insulated wire wrapped around a round piece of plastic cut from a cup as the coil. After completing the circuit the code was slightly altered to only print when a change was registered in the readings.

### 3.1.3 Testing

Initial testing revealed that the sensor gave very reliable readings, being completely stable until metal was brought in close proximity. Adapting the Arduino code to only register when the measurement changed by a minimum value yields a detector that can reliably sense when a metal object passes through or close to the coil. The code used can be found in the appendix. The test setup used is shown in figure 25b. A simple cover was placed on the detector to ensure the test bags were placed in equal proximity to the sensor. The bags used were identical, with different contents. One filled with metal pieces, one containing a glass jar with a metal lid and one containing MDF and plastic. The test

results are shown in figure 26. Note that the bag with MDF and plastic gave no indication and is therefore not present in the figure. The indicated change is larger for the bag with metal pieces, but this is mostly due to proximity to the coil and not due to the material or amount of metal present.



(a) Metal detector (b) Metal detector with cover

Figure 25: Metal Detector Setup

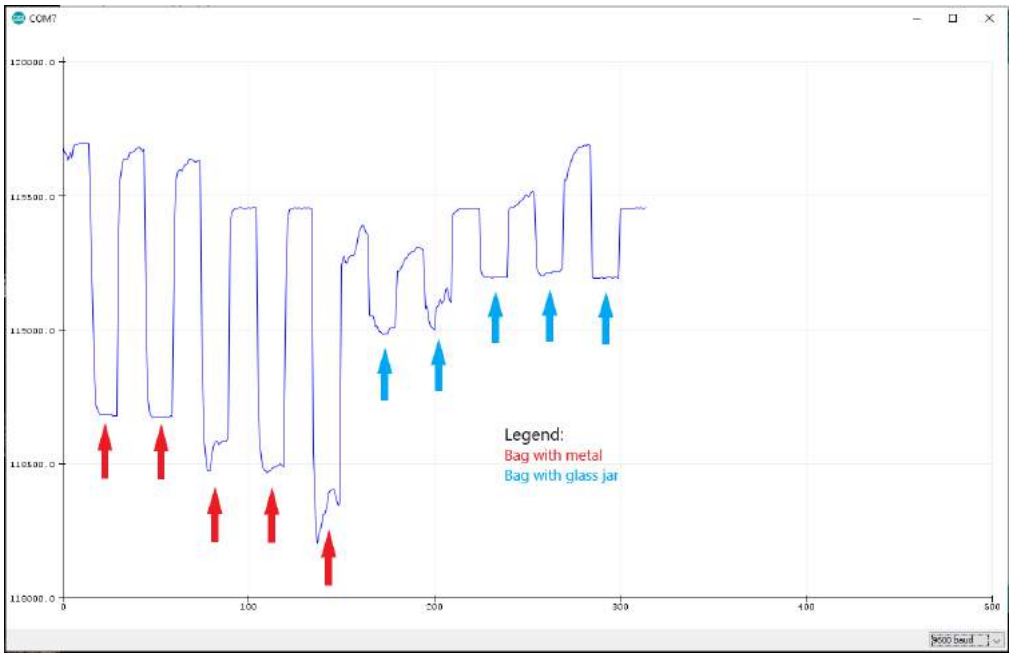



Figure 26: 5 Readings of metal followed by 5 readings of glass jar with metal lid

### 3.1.4 Viability and implementation

As the readings from this detector varies with orientation, size and material of the object, its use is limited to detecting the presence of metal. Depending on where the sensor is applied, this does not have to be a disqualifying characteristic. One possible application is detection of metal in bags exclusively for plastic waste. In this case any presence would lead to the bag being removed from the plastic recycling chain, regardless of the metal amount or composition. This also applies to the green bags for food waste. It is also useful in combination with other sensors, as the combined output can be used both for identifying false positives or negatives, and possibly identify common objects that combine metal and other materials like for instance glass jars with metal lids. The detector can also be applied when emptying the bins into the truck. As the bins are tagged with an RFID, a frequent occurrence of metal can reveal if a single source is not performing the sorting properly.
If more accurate readings showing for instance metal amount is desired, a possible solution might be to include several small coils in a grid pattern or similar, as a small coil might be less affected by the orientation of the material. This method would require additional testing to determine its viability.

## 3.2 Sensor ball

### 3.2.1 Motivation

During the process of collecting and sorting trash, the bags collected will be subjected to varying forces at different locations. This subjection is a possible reason for the bags rupturing during the process, causing the bags to spill their contents and reducing the quality of the sorting as described in 1.3. At the time, the magnitude of the forces and their locations are unknown, so a method for measuring this is desired.

### 3.2.2 Prototyping

A force sensor was developed with the intention of logging the forces applied during the trash collection cycle. The working principle of this sensor is described in detail in section 2.2.4. Using the previously described composite material, tests were performed to assess if it was applicable to this problem. Preliminary tests indicated that the readings were consistent with the magnitude of the force applied, but the samples used were not directly applicable to the intended use. Therefore a new sample was made by wrapping a sphere of rigid foam in the composite material, see figure 27. The reason for the spherical shape was for the sensor to give consistent readings regardless of its orientation. The ball was connected to an Arduino to obtain readings and subjected to testing using a specialized test setup as shown in figure 28.

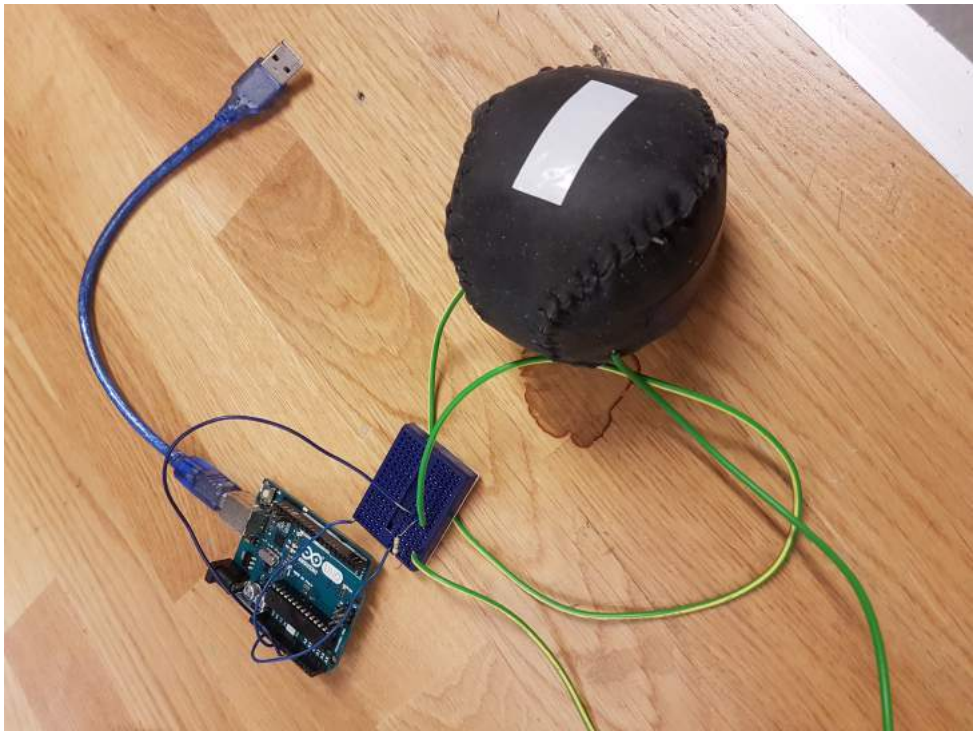


Figure 27: Sensor ball

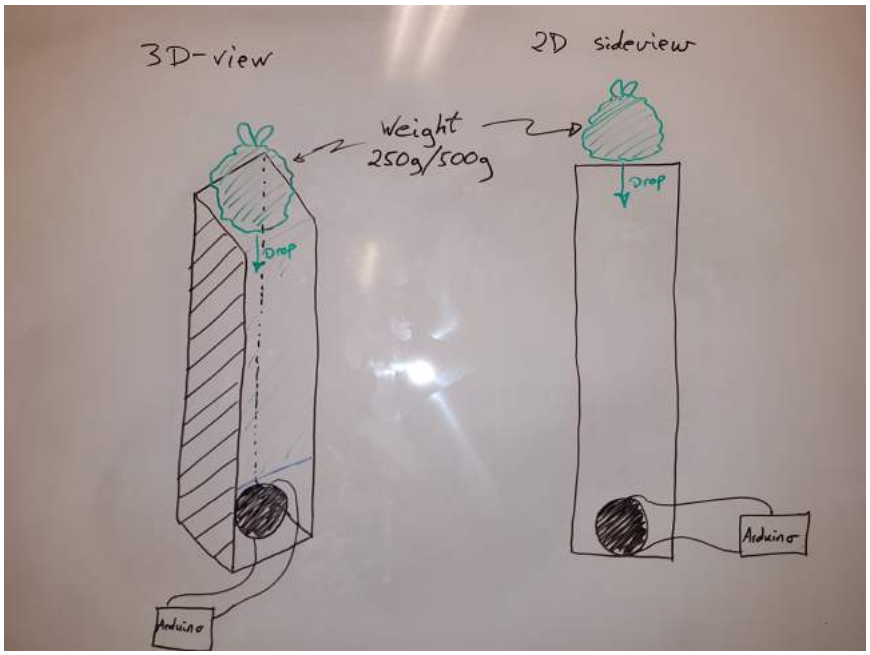



Figure 28: Sensor ball test setup

### 3.2.3 Testing

The test was conducted by placing the sensor ball at the bottom of the tower and dropping a bag of known weight from a set height. The readings were recorded and are shown in figure 29. The separate tests relate to different orientations of the ball. As seen in the graphs, the difference between 250 and 500 grams is not significant, which lead to the conclusion that the sensor could not be trusted without making changes to the design. It should be noted though that the readings are very consistent between each test. Further testing revealed that the sensor had a maximum threshold for applied load, probably due to the silicone matrix reaching its maximum deformation. It is possible that increasing the material thickness or hardness would increase this threshold so the sensor can differentiate between loads in the desired spectrum. At the time, the magnitude of the loads that can be expected in the trash collection are unknown and will require extensive trial and error to ascertain.

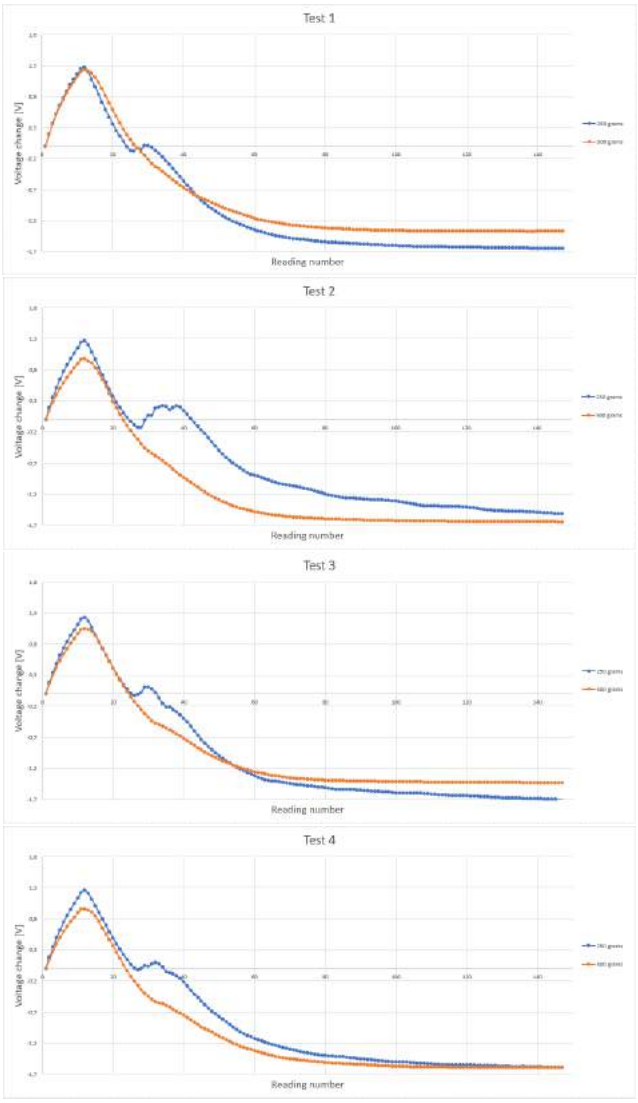



Figure 29: Sensor ball test data

### 3.2.4 Viability and implementation

As the sensor will require a lot of rework, and must be extensively tested during development, it was decided at the current stage to pause its development and explore other avenues in the project. In addition to tuning its applicable sensing area, it will be necessary to place its location in the collection cycle as it is desirable to know both the magnitude of force and the place where it was measured. This means that some way of location tracking must be implemented which has challenges of its own. Several locations, including the garbage truck and sorting facility, may be shielded from GPS and/or network signals which would make tracking more difficult. Also, some way of extracting the device at the end of the cycle is desirable, as failure to do so would result in it being incinerated. In conclusion, this device requires extensive work and the results might not be worth the effort.

## 3.3 Trash-detection through object-detection

### 3.3.1 Motivation

Gathering information about types of garbage collected can be of great interest for Renovasjonsetaten. The ratio of blue, green and regular bags may reveal important insights about the amount of material recycled by the consumer. For instance, if the number of blue and green bags are kept consistently low over an extended period of time at a specific household, it may indicate an absence of recycling initiatives adding to the severity of "wrongly sorted material" in figure 13. Counting bags may also aid in the estimation of the amount of ripped bags by comparing the amount before and after a stage. Estimating the amount of loose items and classifying them within each stage of the collection cycle may also provide useful insights as to how significant that problem is. An object detection algorithm may provide information on all three problems.

### 3.3.2 Prototyping

At first, focus was on training a network to detect blue and green bags in the back of the truck in order to determine its viability. Darknet was chosen as the neural network framework due to its simplicity as well as its ability to support GPU calculations making the training speed up to 500 times faster[19]. Furthermore, yolomark was used to mark and classify the bags [2]. Training the algorithm to detect blue and green bags in the garbage truck proved to be easy and required less than 50 pictures to be marked before it reached decidedly high precision. Additionally, it was experimented with other camera angles, such as a side view, see Figure 31. The goal was to uncover bags that are left undetected by the front camera due to other material blocking the view and as such, acquire more accurate data. A challenge then would be to combine the two algorithms, preventing the same bag being counted twice.


Figure 30: Front view bag detection


Figure 31: Side view bag detection

As it was clear that implementing object detection was viable, it was decided to take it a step further. Yang and Thung[31] attempted to train an

algorithm to classify pictures (object classification, not object detection) containing six different kinds of waste with moderate success. They tested with both AlexNet[15], a CNN based model, as well as an SVM model. The achieved accuracy was 63% and 22% respectively. Accuracy is a metric that is mostly useful in image classification, and it's simply described as the percentage of correctness in the predictions. Between 300 to 500 pictures within each class were used to train the models. The data was obtained manually by acquiring different kinds of waste and photographing each individual object on a white background. This was described as the most tedious and time-consuming part of the project. Luckily, the pictures were made available through a git-hub for others to use[kilde github]. [kilde til de andre som prøvde] tested several deep learning models on the same data set, including Densenet121, DenseNet169, InceptionResnetV2, MobileNet, Xception. They achieved a test accuracy rate of 95% using Densenet121 as the best performing architecture.

It was decided to utilize the data set made available by [31] to attempt to train a yolov3 model using Darknet for object detecting even though the images were meant for image classification. To train, bounding boxes had to be placed in each of the 2527 pictures in a manner described in section 2.2.1. As it seemed like a tedious job, a code was developed in C++ using OpenCV to automatically detect contours in each picture and thereby assign a bounding box [referanse til kode i Appendix]. This was possible due to the white background of the pictures which was one of the advantages of the highly standardized data-set. To avoid including contours that occurred due to shadows, each picture had to be blurred prior to detecting contours. That is, averaging out pixel values based on the surrounding pixel values. Additionally, a threshold for what constitutes a contour could also be altered based on the object in the picture. These parameters were not the same for all pictures. One combination would for instance be perfect for colored glass, though leave out much of a transparent glass. As such, yolo mark was used to scan through all bounding boxes to correct those that were poorly placed by the code.

The algorithm was set to train overnight using the .cfg file shown in [appendix] with 20% of the data within each category set aside for testing. The algorithm reached a mAP of about 84%, see figure 32, which compared to Yolov3 performance in PASCAL VOC is a relatively high precision. Additionally, the loss function seems to have converged to a fairly low number.

### 3.3.3 Testing

Even though the results seemed promising, the object detection algorithm performed poorly when presented with self made data. It especially had troubles distinguishing clear glass from plastic as shown in Figure 33.

Live testing using a web-camera was also performed. As expected it also performed poorly. It is likely that the trained network suffers from overfitting described in Section 2.2.1.

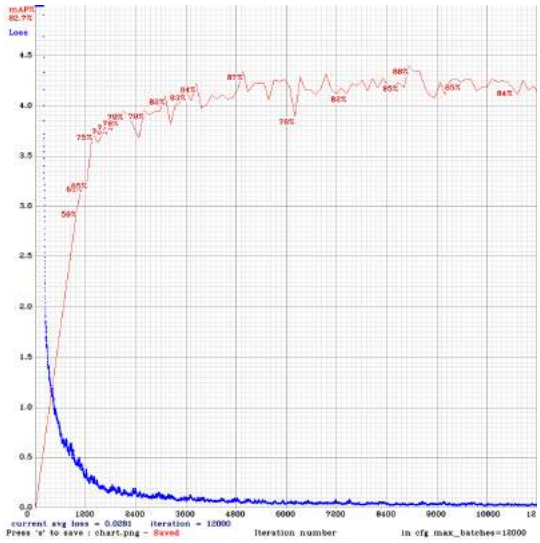



Figure 32: Performance Chart Yolov3 Trash Detection


Figure 33: Glass detection test

### 3.3.4 Viability and implementation

The bag detection algorithm was relatively successful in that it reached a high accuracy with only a few images. As all data given to this algorithm would be from the same angle (the camera is stationary) with a constant background (truck) the problem of specialized data would not be a big problem. As such,

it is not necessary for this specific network to be able to detect bags in other circumstances, which makes the training simpler and the likelihood of success higher.

If one wishes to count bags in other areas as well, one could simply train a separate network for that exact location. As most areas of interest in the trash collection cycle has a constant background this is likely an easy implementation. Such locations might be before and after stage 3 to stage 5 to estimate the amount of ripped bags in the truck. It may also be before and after stage 7 to estimate the amount of ripped bags in the filtering machine.

With regards to classifying and localizing different types of trash, more testing and more data gathering should be conducted before landing on a decision. It is likely that since the images used where only meant for image classification the network had a hard time detecting several objects, see Appendix A.5 Figure 45d. Images from the exact location where the object detection is supposed to happen, should be collected. It might be of interest to focus only on few classes or just loose items in general. For instance, one might train a network to estimate the amount of blue, green and regular bags, as well as estimate the volume of loose items, that is, all other objects than bags. Such a network might be useful in order to determine the severity of loose items at several stages. It might also be interesting to place it before and after stages, as described above, in order to correlate loose items to ripped bags. As such, one might get a sense of which root cause of ripped bags might be the more critical.

## 3.4 Trash-detection through sound-detection

### 3.4.1 Motivation

One material that is not supposed to be discarded into the regular garbage is glass. Glass is supposed to be deposited at separate collection points that are emptied by dedicated trucks at certain times. Despite this, glass often appears in the mixed trash bags, and possibly also in the plastic and food waste bags. This is a problem for several reasons. Glass has a higher melting point than the temperature in the incinerator, and must therefore be removed at regular intervals. Broken glass is also sharp, and may be a cause of ripped bags during the collection cycle. Lastly, glass present with the plastic waste will be a problem in the plastic recycling process, and glass in food waste causes issues with the bio gas production. We knew from talking to the people employed at Renovasjonsetaten that glass could be heard from time to time during trash collection, so there is a possibility of recording and recognizing these sounds with a machine learning algorithm.

### 3.4.2 Prototyping and testing

The python code described in section 2.2.2 was used to determine if sound classification was a possible way to gain information about trash bag contents.

The first task was to ensure that the program functioned as intended, and for this purpose the data set used in the reference project [23] was imported for initial testing. Figure 34 shows the training and test accuracy using the UrbanSounds8K data set. A test accuracy of 92.9% shows promise, though it does not guarantee that the model will function well when trained using other input data.

| Epochs | Training accuracy | Test accuracy |
|---|---|---|
| 100 | 97,1% | 90,9% |
| 150 | 98,9% | 92% |
| 200 | 99,5% | 92,9% |

Figure 34: UrbanSound8K training data

After the proper function of the program was confirmed, a new test set was made to assess its accuracy in classifying mixed trash of different types. Three separate bags were made, one containing two glass jars, the second containing MDF and plastic, and the third containing plastic and metal pieces. The test set was created using a Zoom microphone, and repeatedly dropping the bags on a table. A sound editing program was used to separate the sound file into short segments by splitting into segments whenever a period of silence was detected. The resulting set consists of 220 glass sounds, 272 metal sounds and 210 plastic/MDF sounds. Two classes were established for training, "glass" and "rest". The goal for this test was to see how good the algorithm is at distinguishing between the two classes. Figure 35 shows training and test accuracy using this data set.

| Epochs | Training accuracy | Test accuracy |
|---|---|---|
| 100 | 98% | 90,8% |
| 150 | 99,3% | 90,8% |
| 200 | 98,9% | 92,9% |

Figure 35: Self-made training data

Although the training shows high accuracy and also high precision in classifying the allocated test set, there are important factors to consider. The test set is small and very specialized. The size of the test set limits the variation available for training, and this can result in the model becoming overfitted [3]. An overfit model will have difficulties in classifying sounds that are different from the ones encountered in the data set, for instance with other background noises present. This same problem is made even worse by the set being so specialized. Specialized in this setting means that the sounds are recorded in very

controlled circumstances with barely any noise in the background. Any random noise will therefore cause much confusion for the model, and possibly lead to false classification. One advantage though is that the sounds recorded are very similar, as most of the sound made comes from the table when the bags hit. As the sounds are similar, there is little difference left for the model to distinguish between, and the high test accuracy shows that this is not a problem. The conclusion then is that there are slight differences that the program can differentiate between with good accuracy, and in the real setting that is what will be required because of high amounts of background noise.

### 3.4.3 Viability and implementation

This approach presents several difficulties to overcome. Several factors affect the end performance, the most important being the data set used for training, and the microphones used.

The data set has to be of sufficient size to ensure that the program will be able to find the distinguishing factors between the different sounds produced by different materials found in the trash. A higher number of sounds distinguished requires a larger data set, and a larger set will also reduce the chance of false readings as it will have a higher chance of containing all random noises encountered from different sources. Obtaining a good set is time consuming, as it is necessary to ensure that the sound clips are classified correctly; for instance a glass sound has to be labeled correctly as glass and other sounds must have their corresponding labels. The set must therefore be created in controlled circumstances. For optimal performance, the data set must also be recorded in a setting as close as possible to the real life situation. Any noise encountered in an actual situation should be present when making the recordings, thus requiring the data collection to be performed on site, preferably under normal operation. If the only distinction desired is between glass and other sounds, several thousand clips where glass is known to be present must be recorded while the other part of the set can be recorded randomly during normal operation. Any recordings done from normal operation should be controlled to ensure they contain no sounds from glass, and this will probably take a lot of work. Once the data set has been made however, the work does not have to be repeated unless major changes to the collection process are implemented which change the overall sound picture.

Another important factor to consider is the type and placement of the recording microphones. Tests have to be performed to make sure the sound recorded yields a high difference between the desired sounds so they are easier to distinguish. There is possibly a high difference between using a contact microphone compared to one picking up from the surrounding air, and some types are more dependent on direction than others. Placement and quantity is also to be considered, and possibly merging several recordings to one clip to obtain an averaged result.

The custom data set was tested on other recorded sounds using a different microphone, but with the same bags used to make the data set. This test revealed that using the same microphone in the same placement is vital as the results from this test was poor compared to the results from earlier. Sounds were often misidentified though in general they were closer to their real classification than not, giving some confidence to the viability of the method.

## 3.5 Infrared trash classification

### 3.5.1 Motivation

As indicated in 2.2.6 it may be possible to obtain a material specific profile by determining its reflectivity, absorptivity and its transmissivity on a specific wavelength, for instance in the infrared waves region. As determining bag contents as well as classifying loose items is a need for REN, beaming bags or objects with infrared waves might prove to be useful.

### 3.5.2 Prototyping and testing

The initial idea was to point an infrared diode on an object and measure the amount received at the other end. Such a diode where plugged in the Arduino as well as an infrared photoresistor, which is a resistor that changes its resistance based on amount of incident infrared light. As such one could hopefully correlate the data with what type of material was being scanned.

Initial tests of this showed that almost no radiation would pass typical objects found in trash bags. However, there might still have been a difference in reflection. As such, the receiver was placed next to the emitter with isolation between. This did not prove to be a viable method either as the received signal was quite low and the output values did not change sufficiently. A reason for this might have been a lack of power from the infrared diode. Therefore, a high power infrared diode was tested as well, though with the same result.

### 3.5.3 Viability and implementation

It was concluded that more precise equipment would be needed in order to utilize infrared light to determine the type of material. NIR [12] is a proven method for determining different types of plastics using infrared light.

## 3.6 WiFi trash classification

### 3.6.1 Motivation

The motivation for a WiFi trash classification system is much of the same as with Infrared trash classification. If the amount of radiation passing through a bag differs based on the material residing inside, one might gain information about the contents of a bag by reading the received signal on the other side.

### 3.6.2 Prototyping

By using two ESP8266 configuring one to be the server and the other to be the client, one could possibly send a WiFi signal through a bag, receive it at the other end, and display the signal strength using RSSI(), see section 2.2.5. Simply placing different objects between the two did not alter the signal strength. A reason for this might be that the signals reflect off other surfaces in the room and back to the receiver, inhibiting a change in the RSSI.

By shielding the signal from the surroundings one could achieve less reflections. The initial test of this concept was done by inserting one of the ESP8266 in a steel cylinder (vaccuum chamber at TrollLabs). The RSSI clearly increased when the other ESP8266 was held in front of the opening as well as decreased when placed next to the cylinder.

Additionally, it was hypothesized that if one could concentrate the signal into a pipe sized beam (with the diameter of a trash bag), one could possibly reduce the reflections from the surroundings simply by letting less of the radiation go to the surroundings. Such a beam of radiation is called a collimated beam as described in section 2.2.7 and could be achieved using a paraboloid. Two paraboloids were 3D printed according to the model in figure 36. Both where laced with aluminum foil as to increase reflectivity of the parabola.

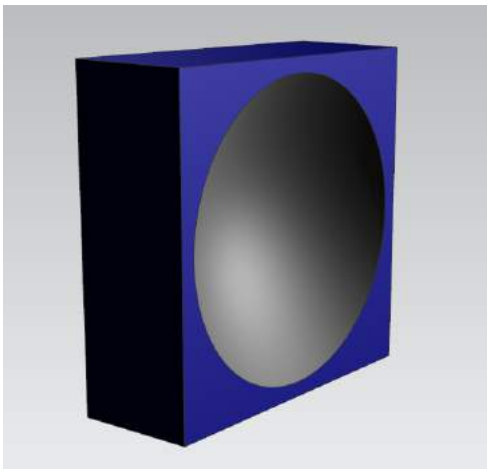


Figure 36: 3D printed parabola

### 3.6.3 Testing

An elongated plate with glued on slots was created in order to ensure the same distance and orientation with every measurement. Additionally, brackets for

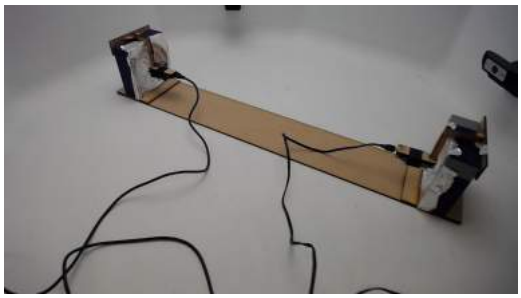


Figure 37: WiFi classification test setup

holding the ESP8266 were made in order to keep the sender and receiver at the focal point of each paraboloid, ref section 2.2.7. The whole test setup is shown in figure 37.

Seven different cases were tested. Nothing(air), bag containing plastic, bag containing glass, metal, a slice of pizza, a water bottle and a hand. Pictures of these tests can be found int he appendix, see figure 45. The RSSI was recorded in each case when it had reached a stable value. The results are presented in figure 38.

| Material | Reading in dB |
|---|---|
| Air | -23 |
| Plastic | -23 |
| Glass | -22 |
| Metal | -23 |
| Pizza | -24 |
| Water bottle | -33 |
| Hand | -27 |

Figure 38: RSSI values with various objects blocking the signal

### 3.6.4 Viability and implementation

From the test results it seems there is a certain reaction from some objects blocking the beam. There is a change present for the hand and the water bottle, which block or absorb much of the beam. The explanation for this might be a high absorptivity of radio waves in liquid. However, other material might have been present during the tests such that much more data must be collected in order to safely assume a specific reaction to a specific material. At least the data proves that one might get a reaction in the RSSI when creating a collimated beam and blocking it. When not using paraboloids, there would be little to no change in this value. As such, there is potential in this device, and it should be

explored further.

It should be mentioned that the beam cannot be perfectly collimated because one cannot make a perfect paraboloid. Additionally, the signal is not generated at exactly one point and will thereby cause some radiation to reach the surroundings. As such, a change in the surrounding geometry may lead to a change in received signal strength. An idea could be to surround parts of the outgoing and receiving beam with an absorbing material causing less radiation to escape the surroundings, and less radiation reflected to reach the receiver.

The experiment shows promise for a dual-paraboloid constellation in conjunction with microwaves for spectroscopy. Other frequencies should be tested with this device as well, such as radio waves lower than the WiFi frequencies or UV-frequencies. A placement for this device could be in the trash bin or in the truck. That is, in Stage 2 or stage 3 such that content could be detected prior to losing traceability. Additionally, it could be placed before Stage 9.1 in order more accurately filter out contaminated plastics, or at any stage where classification is desired.

## 3.7 Bagrip detection

### 3.7.1 Motivation

Some prototyping was performed at an early stage to attempt to make a bag that would record if it was ripped open. The motivation for this was the problem with loose objects inside the sorting facility. A significant part of the loose items originate from bags that are ripped open during the collection cycle, evidenced by there being found many empty, open bags inside the sorting facility. This leads to lots of material being incinerated instead of being sorted correctly. Also, the plastic waste is contaminated by loose waste, such as food waste especially, and this degrades its quality considering later recycling. As previously mentioned, it is not known where and why most bags are ripped, so specialized bags that can detect where this occurs is of interest.

### 3.7.2 Prototyping

One idea consisted of making a mesh of conducting material embedded in the bag, see figure 39. Ripping a hole would change this mesh, severing connections that cause the resistance to either drop or increase. Placing all the electronics on the inside and filling the rest of the bag with waste, and possibly other sensors, would allow it to behave exactly like a regular trash bag, and thereby giving good feedback on where ripping occurs. This would of course require several bags and several trips through the system to gain enough data points. The idea of embedded conducting material comes with several challenges. The strength of the material must not be changed significantly, as this could lead to no ripping, or at a different location than usual. Also, the conducting material

has to tear at the same time as the bag, or it could result in false readings. A conducting coating was suggested, but it would need to be flexible and bond well to the plastic. Also, any change in resistance is difficult to measure if the value is very low. All these constraints make material selection very challenging.

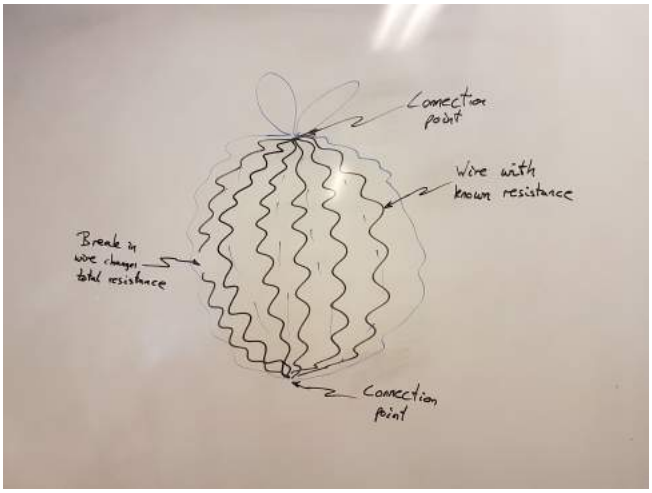



Figure 39: Bag rip detection, resistance monitoring

Another solution that would be easier to implement, is using a gas sensor inside a sealed bag, see figure 40. The bag can either be filled with a gas that the sensor can detect, causing the amount to drop when the bag is ripped, or it can react to any gases that can reliably be found in the collection cycle. Some initial testing showed that the sensors easily available were not sensitive enough, but there are sensors on the market that would be adequate, although at a much steeper price. Additionally, most of the sensors are made to detect flammable gases, which we do not want to make use of due to health and safety concerns.

### 3.7.3 Viability and implementation

Other features that will have to be implemented is some way of knowing the location of the bag when a rip is detected, and preferably a way of recovering it before it ends up in the incinerator. Wireless transmission of the data would render this unnecessary, but it would still be optimal if the bag could be reused. In general this product seems achievable, but it requires a lot of work for it to function as intended and the gain might not be worth it compared to other areas of interest. Due to other projects looking more promising, further development of these prototypes was stopped.

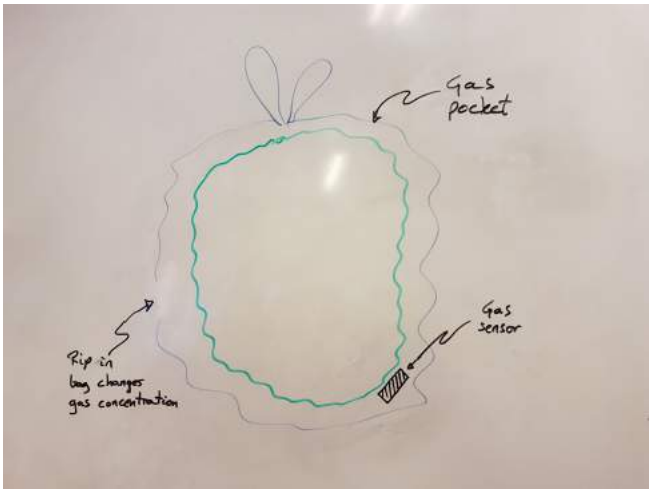



Figure 40: Bag rip detection, gas monitoring

# 4 Application and future work

In this chapter the findings applications will be discussed as well as reflections on possible future work. Lastly, a ranking on the methods and concepts will be given. It must be mentioned that the hunt2.1.2 is by no means over such that the methods recommended should be further investigated. Additionally, there is likely many more stones left unturned such that the eyes should be kept open for new discoveries.

## 4.1 Non-focused information gathering

The initial request made by the customer was to use an array of different sensors placed strategically at a point in the collection cycle with the goal of collecting information without a predetermined purpose. While this method could prove valuable, it also comes with very high risk. The array could consist of for instance microphones, cameras, IR emitters and sensors, UV emitters and sensors, several radio wave transmitters and receivers. Another possibility in this regard would be to make a sensing unit that travels through the system packed with several sensors, but without any consideration for how the sensors are placed or what they should be tuned for. This information gathering could potentially reveal some interesting information over time, but the drawbacks are large and there is a high probability that the collected data would not be useful. One major reason for this is that many sensors require tuning depending on what they are supposed to detect. This is made clear with the sensor ball discussed earlier, which could not differentiate between 250 grams and 500 grams dropped from the same height. If this was placed directly into the system without any prior testing, the results would probably be chaotic and yield little valuable information. The readings peaked at very low forces, so the sensor would probably show high impacts through the entire collection cycle. As this situation applies to other sensors as well, the conclusion would be that a lot of work has been done for very little or no gain.

This issue can be overcome by front-loading, which is also a very economic approach. By running tests on each sensor in the lab, situations similar to the real world application can be used to tune the sensors before installation. This approach is cheap and quick, and increases the chance of gaining good data. The sensors would most likely require additional tuning after applying them to the end stage, but this tuning is made easier after the coarse testing done in the lab. Doing this would still work with the same approach as originally intended, but have a much higher possibility of yielding useful information.

It is however worth mentioning that machine learning has the possibility of detecting subtle differences in the data that are invisible otherwise. Not doing any sensor testing beforehand has a much higher chance being successful when applying machine learning to interpret the collected data. This does add the need for data classification discussed earlier, which is very time consuming. Also,

preliminary tuning of the sensors does not necessarily affect the machine learning in a negative way, and also adds the possibility of producing understandable data for a human interpreter. Therefore preliminary testing and tuning is still a preferable approach.

In general, the non-focused approach is not recommended in the initial phase, as an approach designed to look for specific targets will have a higher probability of yielding useful results. Over time however, when a sensor apparatus is already implemented, it could be interesting to see if the data could be used in other ways than at first intended. Different combinations or applications can possibly result in new discoveries not originally planned for.

## 4.2 Possible areas for further work

### 4.2.1 Sensor bag

One idea discussed consists of making a sensor bag, which is a sensor array that mimics a trash bag to be sent through the system and collect information along the way. This product would preferably be approximately the same size and weight as an average trash bag, and have more than a single function. Previously discussed development could be implemented in this device, such as the sensor ball, and some sort of bag rip detection. Other sensors include accelerometers, temperature, moisture and gas. This array could provide many useful insights, making it possible to eliminate known problems like bags being ripped apart by identifying the stages where this happens. There are some problems to overcome if this device is to be created. The information is a lot more useful if the location of the bag can be pinpointed at all times during the cycle. This is because one needs to know at what location the readings were made for the data to be useful, and also to correct for expected circumstances like the weight of trash above the sensor.

Positioning was explored to some degree, and found that this was not easy to implement. A rough positioning is achievable using for instance GPS or RFID scanning at specific locations. The GPS positioning could be problematic due to the sensor being underneath piles of trash and not getting a signal to the positioning satellites. Parts of the cycle take place inside a truck or a building containing heavy machinery and this would probably disrupt the signal. Using RFID is possible, but would the range of these devices is very limited and would require the sensor to pass close to an antenna at each stage. High power antennas and the use of several at key positions could be adequate for this to work. This does not however give very accurate positioning, like for instance the vertical position of the sensor inside the garbage truck. Triangulation was explored for this purpose, but this is also problematic due to varying amounts of garbage around the sensor. Triangulation usually depends on comparing the signal strength of several sources, and this strength would vary with the material present between source and sensor bag. As the amount of debris in between

is unknown, it is impossible to account for this signal distortion. A solution using a grid of narrow beam transmitters was proposed, where the sensor bag could infer its position based on which of the transmitters were visible, but this solution is complex and would require many components to be implemented.

In general the sensor bag would possibly yield a high amount of useful information, but there are many difficult design challenges to overcome for it to work. After discussing pros and cons of the possible angles to approach, this device was rejected due to its complexity. It is however an interesting possibility that could be explored further in the future.

### 4.2.2 Trash Classification in sorting facility

In the case of trash classification, which in general refers to detecting the contents of a bag and/or loose items, there are several places such a system could be implemented. One of these is inside of the sorting facility. The quality of the sorting could be dramatically improved depending on the accuracy of the system, and other problems caused by unwanted items could also be avoided. An assessment of the possible implementation areas was made to decide whether such a system should be placed somewhere inside the facility, or at another location for the most impact. The first point to discuss is whether the system should be placed before the color sorting of the bags, or at a later stage. At the early stage the system can be used to separate loose items, eliminating their negative impact on the machinery in later stages like debris clogging. There is also the possibility of extracting bags that contain wrongly sorted material by scanning their contents in combination with color detection. The bags at this stage are however less separated than later, so there could arise problems with several bags being detected at the same time, confusing the sensors as to which bag contains the unwanted contents. It would also be required to scan the color so the system knows what contents is acceptable, something that would be unnecessary if the system is implemented after color sorting. Since the color sorting is already in place, it seems logical to place the sensors at a later stage. A less sophisticated system could be used before sorting just to single out loose objects or similar. After sorting the bags are less frequent, minimizing confusion, and the criteria for acceptance are clearer.

One specific location is of special interest inside the sorting facility. At stage 9.1 in figure 10, the plastic bags are separated by weight as the heavy bags are suspected of containing other than plastic. The mentioned classification system could potentially improve this, gaining a higher accuracy resulting in less wasted pure plastic and removing more contaminated plastic. This would work in a similar fashion for the green bags, as the biogas production is less efficient if the food waste contains unwanted material.

As for the mixed waste that goes to incineration, the contents are less im-

portant, but it would still be of interest to be able to remove particular objects. For instance glass and metal has too high burn temperature and won't be incinerated in the process, so it has to be removed at regular intervals. Also, food waste is hard to incinerate due to high moisture content, and will therefore inhibit the incineration process.

Additionally, there are several stages that might be suitable for object detection. In general, the amount of loose items vs bags might be of interest as all loose items goes to incineration. If the amount of loose items proves to be high, then there might be much to be gained by reducing it with regards to the percentage recycled waste. Loose items might stem from bags that are ruptured somewhere in the process and/or from people throwing loose items without sorting them. By attaching a camera at several stages in the sorting facility, for instance, at the intersection of stage 6 and 7 in figure 7 and 8 as well as on the beginning of stage 8 9 one might simply count the number of green and blue bags within a specific amount of time on each stage and determine the discrepancy. This discrepancy could be attributed to the amount of ruptured bags, though it might also indicate a difference in precision of the algorithms. Even though the bags are more spaced out at the intersection of stage 6 and 7 than at the beginning of stage 6, they are more cramped together here than at the beginning of stage 8. This may cause the algorithm at stage 6-7 to be less precise due to more disturbances in the pictures. Whether or not this will cause more false positives or less true positives is likely dependent on the types of other items found in the stage. Determining the accuracy of this method can easily be done by manual counting and comparing it to the output from the algorithm.

In a wider aspect, the sorting facility is a good candidate for implementation of a classification system. There are some drawbacks compared to other locations though. When wrongly sorted material is found, there is no way of tracing its origin. If the origin of the bag was known, the information could be used to map which areas or households need more information about correct home sorting. Implementing the system into the facility is also harder compared to for instance a collection truck, as there is a lot of machinery in place already and the space is limited.

### 4.2.3 Trash classification in truck

Placing a trash classification system in the truck has both advantages and disadvantages compared to the sorting facility. There are only two possible placements on the truck for such a system, and that is in and around the tray where the bins are unloaded, and inside the storage compartment where the trash is compressed.

The tray is located at the back of the truck and is open to the outside environment, see figure 41 below for reference. Installation here would require shielding the sensors from rain and cold, but this is a very small challenge compared to

all the other factors that need to be resolved. All equipment will need to be placed away from moving machinery, and not inhibit the daily operation of the workers in a significant way. Fortunately there are plenty of spaces available for mounting equipment, which facilitates fast prototyping in the early stage. The largest selling point for this location is trace-ability. There is already a system implemented using RFID where a chip in the bin is scanned when it's emptied. If something notable is recorded by the classification system it can automatically be connected to the bin where the trash came from, and this enables Renovasjonsetaten to take action to avoid similar issues in the future. In addition to the RFID system, there is also a camera system already in place at this location, and this can readily be used with object recognition. The preexisting system has already implemented wireless transfer of its data to a central server so any heavy computing does not need to be performed by equipment mounted on the truck. There are some drawbacks to this position however. In the sorting facility there are only a few locations where sensors must be installed while there are a large number of trucks in use, so the equipment must eventually be mounted on all of them. The only issue with this is price however, as the development can be done on only one truck until it is finished. Additionally, after the bin is dumped into the tray the collection of bags is very chaotic, so when a detector senses something of interest it is difficult to know which bag contains the object. A possibility here is to do measurements while the bags are moving between bin and tray, though the time to collect a reading is very limited. If the readings are collected fast enough, it might be possible for the object detection program to detect which bag was passing the sensor at the moment of the reading. As for detection in the tray, a sensor array that passes over the trash could be able to use location and object detection to sync the sensor readings to a specific bag.

The sensor array might also be placed so it scans the bin as it is hooked on to the loading arm on the back of the truck. The problem with this solution is that it would be impossible to combine with a camera feed, so detecting trash sorted in the wrong bags would be impossible. Metal and glass might still be detected with this approach, so a possibility is placing those sensors at this location and other sensors around the tray. A thorough assessment including testing should be done for several locations to ensure the optimal location for each type of sensor.

As for the interior compartment of the truck (figure 42, this location has several problems and no clear advantages over the outside placement. The compartment is very cramped due to the compression of the trash, so it is hard to place equipment and camera footage would be impossible. The amount of trash increases over time, so the readings will be harder to distinguish from another and trace-ability to the bins is lost.

In general, the truck is a good location especially because of trace-ability to the bins, and due to the pre-existing equipment already in place that the new equipment can be connected to. Access to the trucks is easier than to the sorting

facility, and there are many possible mounting points.


Figure 41: JOAB garbage truck [13]

### 4.2.4 Trash classification stage 1

There are other possible areas for classification apart from the sorting facility and garbage truck. Some of these possibilities have been discussed and are mentioned here. The first possibility is in the home of the consumer, at stage 1 described in figure 2. A potential issue to be solved at this stage is consumer lack of knowledge, resulting in sub optimal sorting. The sorting could be done automatically, by for instance a trash can consisting of sensors combined with machine learning and a way of separating the objects into selected containers similar to [29]. Objects can also be discarded if they do not belong in the three categories described, which are plastic, food waste and mixed waste.

This area is not considered promising due to several factors. It is hard to ascertain how effective this sorting system would be compared to the consumer sorting manually. It is highly probable that a well-informed consumer will sort just as good or better than an automatic system due to the high variation of waste that exists. It may therefore be preferable to identify consumers bad at sorting and providing them with information on the subject. It is also a very expensive solution, as a large amount of households would need these installed, including having access to power.


Figure 42: Interior compartment [14]

### 4.2.5 Trash classification stage 2

Another area to consider is the trash bins outside of households. These would only provide information about the contents and would be quite similar to the system discussed on the garbage trucks. The sensor data obtained would have to be collected, for instance by the garbage truck during emptying or by wireless transfer. This area faces many of the same difficulties as with the in-house sorting system. Power to the sensors and collection interface must be considered, provided by for instance a battery or wirelessly from the truck during emptying. Also there are expenses to consider as this solution would require every trash bin to be equipped, rather than on the trucks, and the equipment would have to withstand harsh conditions like snow and rain. There are some advantages however, as this system is related to a specific container which is connected to a specific household. This makes it easy to follow up on issues like bad sorting. Some of these problems are also avoided if this type of system is applied only to large containers placed at for instance apartment complexes. These are fewer than the single household containers, so the implementation costs and challenges would be lower in total. In this case it becomes significantly more promising, as it is the last point where trace-ability to a specific household can be achieved. Also, due to its size, it is easier to defend costs and difficulties related to powering the sensing system.

## 4.3 Combination of sensory outputs and machine learning

At several points during the assignment, using many sensors at the same location has been mentioned. A large disadvantage with using a single sensor is the problem with false positives. When a sensor output is received, the data is often not specific enough to tell exactly what material or object it has detected. This is for instance the case when applying radio waves (see section 3.6), as the output received has very low resolution. The output gives some information about the object in between, but not enough to tell exactly what it is. As trash generally contains a huge variety of different objects and materials, it is very hard for any single sensor to yield a high accuracy classification.

This problem can be overcome with combining the output of several sensors. A good way of doing this is by using machine learning. For each sensor added, a new dimension of information is added to the data set the algorithm uses to learn. While the combinations of several sensor outputs can be hard for a human to read, it can provide a pattern that the machine learning algorithm is able to recognize. In a case where the algorithm has trouble distinguishing between two possible classifications, a new sensor input might give definitive answer for which of the two classifications is correct. How well this combination works in practice is near impossible to tell beforehand, and must be tested to assess its validity. It should be noted though that in many cases the multiple dimensions of information available is one reason humans are good at distinguishing between different objects. For instance, deciding which material an object consists of is not necessarily easy to tell just by sight, while adding the weight and feel of the object might make it trivial.

Combining the inputs might provide some challenge as most available algorithms are specialized for one type of information. Most existing programs are made to interpret images or speech, and it might possibly require expert knowledge of the subject to combine inputs which are not directly comparable. The possible gain however is vast, and is also the most promising way of gaining the desired information the customer wants.

## 4.4 Ranking of methods and implementation areas

The following list shows the recommended solutions, starting with the most promising method. This prioritization is based on the discussion in section 4.

1. Trash classification by machine learning
    (a) Image classification
    (b) Metal detection
    (c) Sound classification
    (d) Radio wave classification
    (e) Infrared classification

2. Sensor bag
    (a) Bag rip detection
    (b) Force sensing
3. Smart sorter for apartment complexes
4. Smart home sorter

Below follows a list of the recommended implementation areas in prioritized order.

1. Collection truck
2. Sorting facility
3. Trash bins, exterior
4. Trash bins, interior

# 5 Conclusion

During testing and development some methods have distinguished themselves by results and possibilities. In general, machine learning combined with several sensor inputs has many possible uses, for instance better sorting inside the facility or determining how well consumers sort at home. Because of this versatility, this method has been selected as the most promising for future development. As for the implementation area, the back of the collection truck has been identified as most promising. The primary reason for this is the trace-ability to individual consumers, as this has the potential to greatly improve home sorting which could eliminate the need for better sorting in later stages. The recommended system is then: *a machine learning powered trash classification system based on a wide variety of sensor outputs placed in the back of the collection truck. Recommended machine learning inputs include camera feed, sound recordings, metal detection and spectroscopy through electromagnetic radiation.*

# References


[1] Alexey. *AlexeyAB/darknet*. original-date: 2016-12-02T11:14:00Z. Dec. 17, 2019. URL: https://github.com/AlexeyAB/darknet (visited on 12/17/2019).

[2] Alexey. *AlexeyAB/Yolo_mark*. original-date: 2016-12-17T21:25:07Z. Dec. 17, 2019. URL: https://github.com/AlexeyAB/Yolo_mark (visited on 12/17/2019).

[3] Tensorflor authors. *Overfit and underfit — TensorFlow Core*. TensorFlow. URL: https://www.tensorflow.org/tutorials/keras/overfit_and_underfit (visited on 12/10/2019).

[4] M. Azizkhani et al. "Highly Sensitive, Stretchable Chopped Carbon Fiber/Silicon Rubber Based Sensors for Human Joint Motion Detection". In: *Fibers and Polymers* 20.1 (2019), pp. 35–44. ISSN: 1229-9197. DOI: 10.1007/s12221-019-8662-0.

[5] Jason Brownlee. *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*. Machine Learning Mastery. Apr. 21, 2019. URL: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (visited on 12/17/2019).

[6] *Descending into ML: Training and Loss — Machine Learning Crash Course*. Google Developers. URL: https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss (visited on 12/17/2019).

[7] Juan Du. "Understanding of Object Detection Based on CNN Family and YOLO". In: *Journal of Physics: Conference Series* 1004 (Apr. 2018), p. 012029. ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/1004/1/012029. URL: http://stacks.iop.org/1742-6596/1004/i=1/a=012029?key=crossref.aef5af21bb1bf8edebfb4bd94c9cf5ff (visited on 12/03/2019).

[8] Edward C. Jordan. *Electromagnetic waves and radiating systems*. In collab. with Keith G. Balmain. 2nd ed. Prentice-Hall electrical engineering series. Englewood Cliffs, N.J: Prentice-Hall, 1968. xiii+753. ISBN: 978-0-13-249995-8.

[9] Frank P. Incropera et al. *Incropera's Principles of heat and mass transfer*. 8. utg. Global ed. Singapore: Wiley, 2017. xxi+978. ISBN: 978-1-119-38291-1.

[10] Fuho. *ESP8266 - AT Command Reference · room-15*. URL: https://room-15.github.io/blog/2015/03/26/esp8266-at-command-reference/ (visited on 12/11/2019).

[11] Arduino Referance Guide. *Arduino - WiFi*. URL: https://www.arduino.cc/en/Reference/WiFi (visited on 12/11/2019).

[12] Hamed Masoumi, Seyed Mohsen Safavi, and Zahra Khani. *Identification And Classification Of Plastic Resins Using Near Infrared Reflectance Spectroscopy*. 2012. URL: http://dx.doi.org/10.5281/zenodo.1076915 (visited on 12/12/2019).

[13] JOAB. *JOAB Anaconda HD*. Joab. URL: https://www.joab.se/en/product/joab-anaconda-hd/ (visited on 12/10/2019).

[14] FAUN Umwelttechnik GmbH \& Co. KG. *FAUN VARIOPRESS Hecklader Animation*. URL: https://www.youtube.com/watch?v=UGYsEi4D5Dg (visited on 12/12/2019).

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (visited on 12/09/2019).

[16] Stanford Vision Lab. *ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012)*. 2012. URL: http://image-net.org/challenges/LSVRC/2012/results.html (visited on 12/17/2019).

[17] Youn-kyung Lim, Erik Stolterman, and Josh Tenenberg. *7The Anatomy of Prototypes: Prototypes as Filters, Prototypes as Manifestations of Design Ideas*.

[18] Avfall Norge. *Europa har fått nye avfallsdirektiv*. URL: https://www.avfallnorge.no/bransjen/nyheter/europa-har-f%C3%A5tt-nye-avfallsdirektiv (visited on 12/05/2019).

[19] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. 2013. URL: http://pjreddie.com/darknet/.

[20] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA: IEEE, June 2016, pp. 779–788. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.91. URL: http://ieeexplore.ieee.org/document/7780460/ (visited on 12/19/2019).

[21] rgco. *Simple Arduino Metal Detector*. Instructables. URL: https://www.instructables.com/id/Simple-Arduino-Metal-Detector/ (visited on 11/18/2019).

[22] A. L. Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of Research and Development; Armonk* 44.1 (1959), pp. 206–226. ISSN: 00188646. URL: https://search.proquest.com/docview/220681210/abstract/4997F25972114809PQ/1 (visited on 12/17/2019).

[23] Mike Smales. *mikesmales/Udacity-ML-Capstone*. original-date: 2018-12-11T10:36:47Z. Nov. 13, 2019. URL: https://github.com/mikesmales/Udacity-ML-Capstone (visited on 11/28/2019).

[24] Stanford. *Machine Learning*. Coursera. URL: https://www.coursera.org/learn/machine-learning?utm_source=gg&utm_medium=sem&utm_content=07-StanfordML-ROW&campaignid=2070742271&adgroupid=80109820241&device=c&keyword=machine%20learning%20mooc&matchtype=b&network=g&devicemodel=&adpostion=1t1&creativeid=369041663186&hide_mobile_promo&gclid=Cj0KCQiA89zvBRDoARIsAOIePbBCOAb26hmmN9KdrH5K1bmdTspDedsMUgOhU9r8BABULktMzVuswcB (visited on 12/16/2019).

[25] Martin Steinert and Larry Leifer. "'Finding One's Way': Re-Discovering a Hunter-Gatherer Model based on Wayfaring". In: *International Journal of Engineering Education* 28 (Jan. 1, 2012), pp. 251–252.

[26] Librosa development team. *librosa.feature.melspectrogram — librosa 0.7.1 documentation*. URL: https://librosa.github.io/librosa/generated/librosa.feature.melspectrogram.html#librosa.feature.melspectrogram (visited on 12/12/2019).

[27] *The PASCAL Visual Object Classes Homepage*. URL: http://host.robots.ox.ac.uk/pascal/VOC/ (visited on 12/17/2019).

[28] Karl T. Ulrich and Steven D. Eppinger. *Product design and development*. Sixth edition. New York, NY: McGraw-Hill Education, 2016. 432 pp. ISBN: 978-0-07-802906-6.

[29] Wevolver. *Sorting Marshmallows with AI - Using Coral and Teachable Machine*. Wevolver. URL: https://www.wevolver.com/lucas.ochoa/sorting.marshmallows.with.ai.-.using.coral.and.teachable.machine/master/ (visited on 12/09/2019).

[30] M. Xu et al. "HMM-based audio keyword generation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3333 (2004), pp. 566–574. ISSN: 0302-9743.

[31] Mindy Yang and Gary Thung. "Classication of Trash for Recyclability Status". In: (), p. 6.

# A Appendix

## A.1 Arduino code for metal detector

```
// Metal detector
// Runs a pulse over the search loop in series with resistor
// Voltage over search loop spikes
// Through a diode this charges a capacitor
// Value of capacitor after series of pulses is read by ADC

// Metal objects near search loop change inductance.
// ADC reading depends on inductance.
// changes wrt long-running mean are indicated by LEDs
// LED1 indicates rise in inductance
// LED2 indicates fall in inductance
// the flash rate indicates how large the difference is

// wiring:
// 220Ohm resistor on D2
// 10-loop D=10cm seach loop between ground and resistor
// diode (-) on pin A0 and (+) on loop-resistor connection
// 10nF capacitor between A0 and ground
// LED1 in series with 220Ohm resistor on pin 8
// LED2 in series with 220Ohm resistor on pin 9

// First time, run with with serial print on and tune value of npulse
// to get capacitor reading between 200 and 300

const byte npulse = 10;

const byte pin_pulse=A0;
const byte pin_cap  =A1;

void setup() {
  Serial.begin(9600);
  pinMode(pin_pulse, OUTPUT);
  digitalWrite(pin_pulse, LOW);
  pinMode(pin_cap, INPUT);
}

const int nmeas=256;  //measurements to take
long int sumsum=0; //running sum of 64 sums
```

```
39 long int skip=0;   //number of skipped sums
40 long int diff=0;        //difference between sum and
     avgsum
41 long int flash_period=0;//period (in ms)
42 long unsigned int prev_flash=0; //time stamp of previous
     flash
43 double value = 0;
44 double a = 0.2;
45
46 void loop() {
47
48   int minval=1023;
49   int maxval=0;
50   double first;
51   double timer;
52   double timer1;
53
54   //perform measurement
55   long unsigned int sum=0;
56   for (int imeas=0; imeas<nmeas+2; imeas++){
57     //reset the capacitor
58     pinMode(pin_cap,OUTPUT);
59     digitalWrite(pin_cap,LOW);
60     delayMicroseconds(20);
61     pinMode(pin_cap,INPUT);
62     //apply pulses
63     for (int ipulse = 0; ipulse < npulse; ipulse++) {
64       digitalWrite(pin_pulse,HIGH); //takes 3.5
            microseconds
65       delayMicroseconds(3);
66       digitalWrite(pin_pulse,LOW);  //takes 3.5
            microseconds
67       delayMicroseconds(3);
68     }
69     //read the charge on the capacitor
70     int val = analogRead(pin_cap); //takes 13x8=104
          microseconds
71     minval = min(val,minval);
72     maxval = max(val,maxval);
73     sum+=val;
74   }
75   value = value*a + (1-a)*sum;
76
77   while (millis() > 5000){
78     if (timer == 0) {
79       first = value;
```

```
80      }
81      sum = 0;
82      for (int imeas=0; imeas<nmeas+2; imeas++){
83        //reset the capacitor
84        pinMode(pin_cap,OUTPUT);
85        digitalWrite(pin_cap,LOW);
86        delayMicroseconds(20);
87        pinMode(pin_cap,INPUT);
88        //apply pulses
89        for (int ipulse = 0; ipulse < npulse; ipulse++) {
90          digitalWrite(pin_pulse,HIGH); //takes 3.5
              microseconds
91          delayMicroseconds(3);
92          digitalWrite(pin_pulse,LOW);  //takes 3.5
              microseconds
93          delayMicroseconds(3);
94        }
95        //read the charge on the capacitor
96        int val = analogRead(pin_cap); //takes 13x8=104
            microseconds
97        minval = min(val,minval);
98        maxval = max(val,maxval);
99        sum+=val;
100     }
101     value = value*a + (1-a)*sum;
102     timer1 = millis();
103     timer = millis();
104     if ((abs(value - first)) >= 200) {
105       while (timer < (timer1 + 1000)) {
106         sum = 0;
107         for (int imeas=0; imeas<nmeas+2; imeas++){
108           //reset the capacitor
109           pinMode(pin_cap,OUTPUT);
110           digitalWrite(pin_cap,LOW);
111           delayMicroseconds(20);
112           pinMode(pin_cap,INPUT);
113           //apply pulses
114           for (int ipulse = 0; ipulse < npulse; ipulse++)
                {
115             digitalWrite(pin_pulse,HIGH); //takes 3.5
                  microseconds
116             delayMicroseconds(3);
117             digitalWrite(pin_pulse,LOW);  //takes 3.5
                  microseconds
118             delayMicroseconds(3);
119           }
```

```
120           //read the charge on the capacitor
121           int val = analogRead(pin_cap); //takes 13x8=104
                  microseconds
122           minval = min(val,minval);
123           maxval = max(val,maxval);
124           sum+=val;
125         }
126         value = value*a + (1-a)*sum;
127         Serial.println(value);
128         //Serial.print(",");
129         timer = millis();
130       }
131       //Serial.println();
132       timer = 0;
133     }
134   }
135 }
```

## A.2 Image processing and contour detecting code

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdlib.h>
#include <stdio.h>
#include <opencv2/core/types.hpp>
#include <fstream>


using namespace cv;
using namespace std;

/*Mat src_gray;
int thresh = 100;*/
RNG rng(12345);

Mat src, src_gray;
Mat dst, detected_edges;
Mat myImage;

int edgeThresh = 1;
int lowThreshold=13;
```

```
int const max_lowThreshold = 100;
int ratio = 3;
int kernel_size = 3;
String window_name = "someWindow";
String window_name2 = "someWindow2";

double minVal;
double maxVal;
Point minLoc;
Point maxLoc;

double minVal1;
double maxVal1;
Point minLoc1;
Point maxLoc1;

double imWidth;
double imHeight;

double boxWidth;
double boxHeight;
double boxCenterX;
double boxCenterY;

string imFile;
string txtFile;
string label;
string folder;

vector <vector<Point>>  contours;

//void thresh_callback(int, void*);


void CannyThreshold(int, void*)
{
        /// Reduce noise with a kernel 3x3
        blur(src_gray, detected_edges, Size(7, 7));

        /// Canny detector
        Canny(detected_edges, detected_edges, lowThreshold, lowThreshold*ratio,

        /// Using Canny's output as a mask, we display our result
        dst = Scalar::all(0);
```

```
        src.copyTo(dst, detected_edges);

        //imshow(window_name, dst);
}


void writeToFile() {

                ofstream outFile{ txtFile };
                if (!outFile) { cerr << "Can't open file" << endl; }

                        outFile << label<<" "<<boxCenterX / imWidth << " " << bo
                        outFile.close();
}


int main(int argc, char** argv)
{
        if (argc != 2)
        {
                cout << " Usage: display_image ImageToLoadAndDisplay" << endl;
                return -1;
        }
        folder = argv[1];

        for (int n = 1; n < 11; n++) {

                imFile = folder + "/" + folder +to_string(n)+ ".jpg";

                txtFile = imFile;


                txtFile.erase(txtFile.end() - 3, txtFile.end());

                txtFile = txtFile + "txt";

                cout << txtFile << endl;

                cout << "Char:" << imFile[0] << endl;


                switch (imFile[0])
                {
                case 'c':
                        label = "0";
```

```
            break;
    case 'g':
            label = "1";
            break;
    case 'm':
            label = "2";
            break;
    case'p':
            if (imFile[1] == 'a') {
                    label = "3";
                    break;
            }
            else {
                    label = "4";
                    break;
            }
    case 't':
            label = "5";
            break;
    default:
            cout << "Uncorrect image or file" << endl;
            break;
    }

    cout << "Label:" << label << endl;

    src = imread(imFile, CV_LOAD_IMAGE_COLOR);

    imWidth = src.cols;
    imHeight = src.rows;

    if (!src.data)
    {
            cout << "Could not open image" << endl;
            return -1;
    }

    /// Create a matrix of the same type and size as src (for dst)
    dst.create(src.size(), src.type());

    /// Convert the image to grayscale
    cvtColor(src, src_gray, CV_BGR2GRAY);
    blur(src_gray, src_gray, Size(4, 4));
    /// Create a window
    namedWindow(window_name, CV_WINDOW_AUTOSIZE);
```

```
/// Create a Trackbar for user to enter threshold
//createTrackbar("Min Threshold:", window_name, &lowThreshold, n

/// Show the image
CannyThreshold(0, 0);
//minMaxLoc(detected_edges, &minVal, &maxVal, &minLoc, &maxLoc);

//minMaxLoc(detected_edges, &minVal, &maxVal, &minLoc, &maxLoc);


cout << "min loc : " << minLoc << endl;
cout << "max loc: " << maxLoc << endl;
cout << "amount: " << detected_edges.total() << endl;
cout << "Type: " << detected_edges.type() << endl;

vector<Vec4i> hierarchy;
cout << "The test" << endl;
findContours(detected_edges, contours, RETR_EXTERNAL, CHAIN_APPR
cout << "Got through" << endl;
//Mat drawing = Mat::zeros(detected_edges.size(), CV_8UC3);
/*for (int i = 0; i < contours.size(); i++)
{
        Scalar color = Scalar(255, 255, 255);
        drawContours(drawing, contours, i, color, 2, 8,0, 0, Poi
}*/
Scalar color = Scalar(255, 255, 255);


//drawContours(drawing, contours, 0, color, 2, 8, 0, 0, Point())

//Rect rec = minAreaRect(contours).boundingRect();
if (contours.size() == 0) { continue; }
Point min = contours[0][0];
Point max = contours[0][0];
cout << "Got through" << endl;

for (int i = 0; i < contours.size(); i++) {
        for (int j = 0; j < contours[i].size(); j++) {
                //cout << contours[i][j] << endl;
                if (contours[i][j].y < min.y) {
                        min.y = contours[i][j].y;
                }
                if (contours[i][j].x < min.x) {
                        min.x = contours[i][j].x;
                }
                if (contours[i][j].y > max.y) {
```

```
                                        max.y = contours[i][j].y;
                                }
                                if (contours[i][j].x > max.x) {
                                        max.x = contours[i][j].x;
                                }
                        }

                }

                cout << boxCenterX / imWidth << " " << boxCenterY / imHeight <<

                rectangle(src, min, max, Scalar(0,0,0), 2, 8, 0);

                boxWidth = max.x - min.x;
                boxHeight = max.y - min.y;
                boxCenterX = boxWidth / 2 + min.x;
                boxCenterY = boxHeight / 2 + min.y;

                /*boxWidth = imWidth;
                boxHeight = imHeight;
                boxCenterX = boxWidth / 2;
                boxCenterY = boxHeight / 2;*/

                //Normalizing positions according to image size and writing to f

                writeToFile();
                cout << endl;
                imshow(txtFile, src);
                imFile = "";
                txtFile = "";

        }
        //imshow(window_name2, src);

        /// Wait until user exit program by pressing a key
        waitKey(0);

}



        /*if (!src.data)                                  // Check for invalid input
        {
                cout << "Could not open or find the image" << std::endl;
                return -1;
```

```
        }

        cvtColor(src, src_gray, COLOR_BGR2GRAY);
        GaussianBlur(src, src, Size(3, 3), 0, 0, BORDER_DEFAULT);
        const char* source_window = "Source";
        namedWindow(source_window);
        imshow(source_window, src);
        const int max_thresh = 255;
        createTrackbar("Canny thresh:", source_window, &thresh, max_thresh, thre
        thresh_callback(0, 0);
        waitKey();
        return 0;
}*/
```

### A.3 Machine learning code for sound classification

```
# Load various imports
import pandas as pd
import os
import librosa
import librosa.display
import numpy as np

# Load various imports


# Set the path to the full UrbanSound dataset
fulldatasetpath = "D:/Prosjektoppgave/Maskinlaering/EgetDatasett/"

metadata = pd.read_csv(fulldatasetpath + "/metadata/Metadata.csv")

features = []

global mfccs

def extract_features(file_name):
    global mfccs
    try:
        audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
        mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)

    except Exception:
        print("Error encountered while parsing file: ", file_name)
        return None
```

```
    return mfccs

def rebin(a, xrange, bins):
    mfccsrebinned = np.zeros([xrange,bins])
    global mfccs
    for i in range(xrange):
        amplitude = max(mfccs[i] - min(mfccs[i]))
        step = amplitude/bins
        mfccsbinned = np.zeros(bins)
        for j in range(bins):
            mfccsbinned[j] = np.mean(mfccs[i][(mfccs[i] > (min(mfccs[i])+step*j)
            mfccsbinned[j] = np.nan_to_num(mfccsbinned[j])
        mfccsrebinned[i] = mfccsbinned
    return mfccsrebinned


# Iterate through each sound file and extract the features
for index, row in metadata.iterrows():

    file_name = os.path.join((fulldatasetpath),str(row["fold"])+'/'+"cut",str(ro

    class_label = row["class"]
    data = rebin(extract_features(file_name), 40, 40)

    features.append([data, class_label])

# Convert into a Panda dataframe
featuresdf = pd.DataFrame(features, columns=['feature','class_label'])

print('Finished feature extraction from ', len(featuresdf), ' files')

from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

# Convert features and corresponding classification labels into numpy arrays
X = np.array(featuresdf.feature.tolist())
y = np.array(featuresdf.class_label.tolist())

# Encode the classification labels
le = LabelEncoder()
yy = to_categorical(le.fit_transform(y))

# split the dataset
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(X, yy, test_size=0.2, random
```

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D

num_rows = 40
num_columns = 40
num_channels = 1

x_train = x_train.reshape(x_train.shape[0], num_rows, num_columns, num_channels)
x_test = x_test.reshape(x_test.shape[0], num_rows, num_columns, num_channels)

num_labels = yy.shape[1]
filter_size = 2

# Construct model
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2, input_shape=(num_rows, num_columns,
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=32, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.2))
model.add(GlobalAveragePooling2D())

model.add(Dense(num_labels, activation='softmax'))

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

# Display model architecture summary
model.summary()

# Calculate pre-training accuracy
score = model.evaluate(x_test, y_test, verbose=1)
accuracy = 100*score[1]
```

```
print("Pre-training accuracy: %.4f%%" % accuracy)

from keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 500
num_batch_size = 200

checkpointer = ModelCheckpoint(filepath='D:/Prosjektoppgave/Maskinlaering/saved_
                               verbose=1, save_best_only=True)
start = datetime.now()

model.fit(x_train, y_train, batch_size=num_batch_size, epochs=num_epochs, valida


duration = datetime.now() - start
print("Training completed in time: ", duration)

# Evaluating the model on the training and testing set
score = model.evaluate(x_train, y_train, verbose=0)
print("Training Accuracy: ", score[1])

score = model.evaluate(x_test, y_test, verbose=0)
print("Testing Accuracy: ", score[1])
```

## A.4 Code for Sensor ball

```
1  int readPin = A0;
2  double value;
3  double a = 0.9;
4  double Time1;
5  double Time2;
6  double saved1;
7  double saved2;
8
9  void setup() {
10   Serial.begin(9600);
11   value = analogRead(readPin);
12 }
13
14 void loop() {
15   value = value*a + (1-a)*analogRead(readPin);
                    //Sensor values with filtering
16   if (saved1 == 0) {
                                                  //Set start
```

```
      time and reference value
17    saved1 = value;
18    Time1 = millis();
19  }
20  if (millis() > (Time1 + 0.01)) {
                                  //Check if time has
      increased enough for calculating value change
21    saved2 = value*a + (1-a)*analogRead(readPin);
22    if (abs(saved2 - saved1) > 0) {
                                  //If value has increased
        enough, start output
23      Time1 = millis();
24      while (Time2 < (Time1 + 1000)){
                                   //Output sensor data for
          one second, then reset references.
25        value = value*a + (1-a)*analogRead(readPin);
26        Serial.println(abs(value));
27        //Serial.print(",");
                                                    //
            Commaseparated data for easy Excel import
28        Time2 = millis();
29        }
30      saved1 = 0;
31      Serial.println();
                                                     //Make a
          space in between sensor data chunks
32    }
33  }
34 }
```

## A.5 Additional Figures

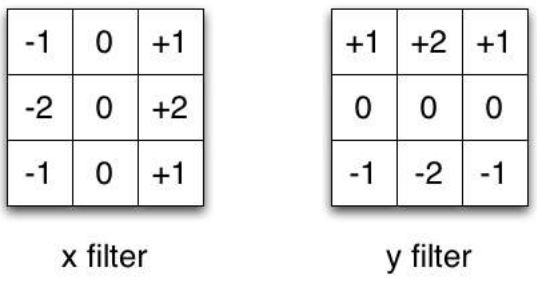



Figure 43: Sobel filter operator

Figure 44: Image before and after Sobel filter application

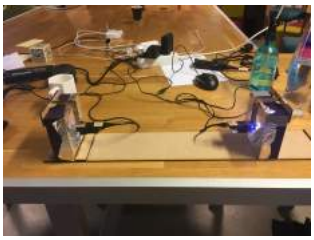
(a) Aluminum

(b) Plastic

(c) Glass

(d) Several Objects

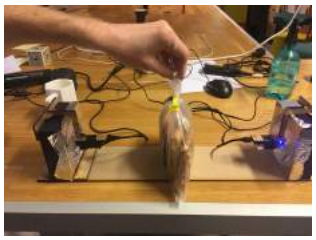
Figure 45: Machine Learning Tests



(a) Air

(b) Bag containing plastic

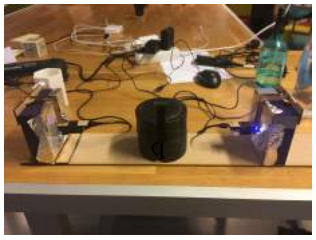
(c) Water Bottle



(d) Slice of pizza

(e) Bag containing glass

(f) Hand



(g) Metal

Figure 46: WiFi test setup

Oliver I. Funch and Robert Marhaug

# Detecting Improperly Sorted Materials in Trash Bags

NTNU
Norwegian University of
Science and Technology