Analytical Service Development
Trash Sorting AI - Report

Group 9 - Sergio Cristia & Ken Oshiro

Big Data Analytics 2020-2021

Arcada University of Applied Sciences

I. Introduction

The recycling rate in Finland remains below the target of 50%.Increasing garbage sorting efficiency and reducing its cost would prove helpful to reach this objective.

To answer this business problem, we propose an AI automated trash sorting system based on computer vision to improve the overall efficiency of the recycling process, reducing the cost of sorting and, by doing so, creating incentives to increase the share of recycled materials. It is capable of detecting and classifying up to 12 types of garbage in real time from a camera feed.

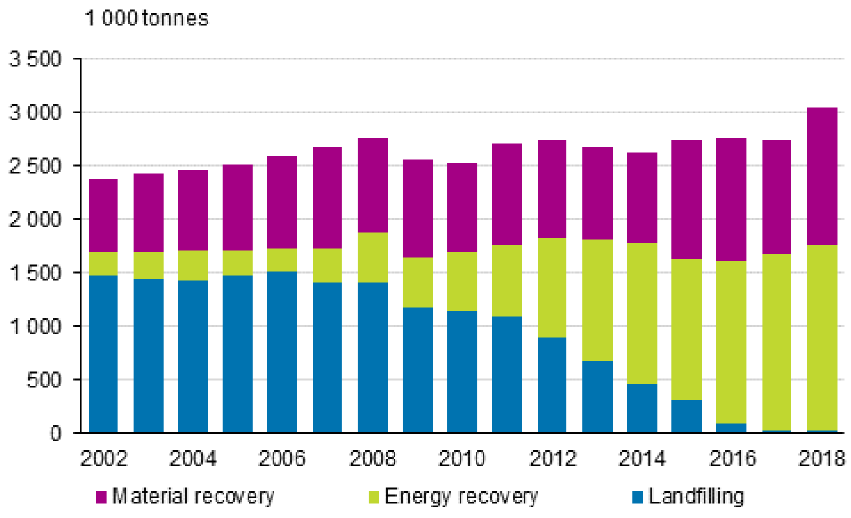
In this report, we will first present the initial problem and our solution. Secondly, we introduce the technicality of the model and the pipeline we used. Finally, we show how it can be used in the real world. The code is included at the end of this report.

II. Problem

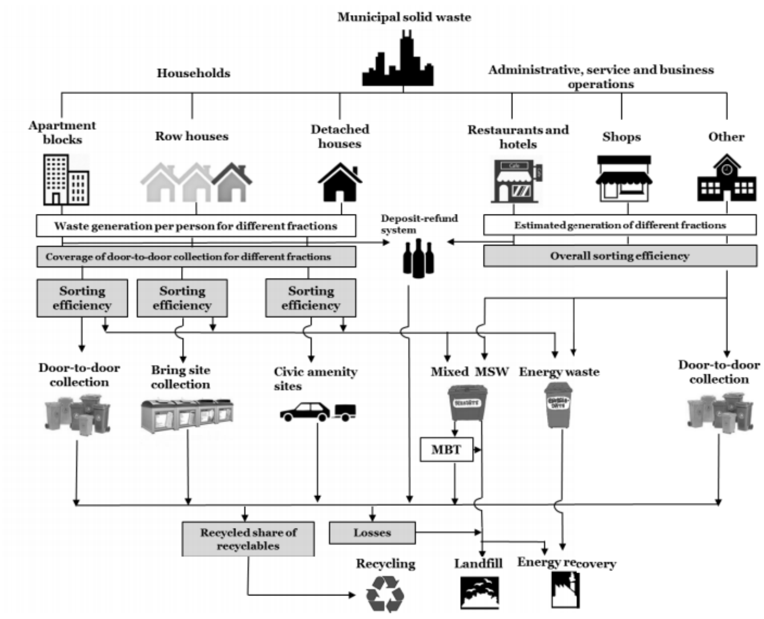
1- Achieving the recycling rate target as defined by the European Union

"With the current sorting efficiencies, a 65% recycling rate would not be reached even if door-to-door collection of all main recyclables covered all the inhabitants in Finland – including over 2.5 million inhabitants in detached houses."

Sahimaa, Olli. "Recycling potential of municipal solid waste in Finland." (2017).

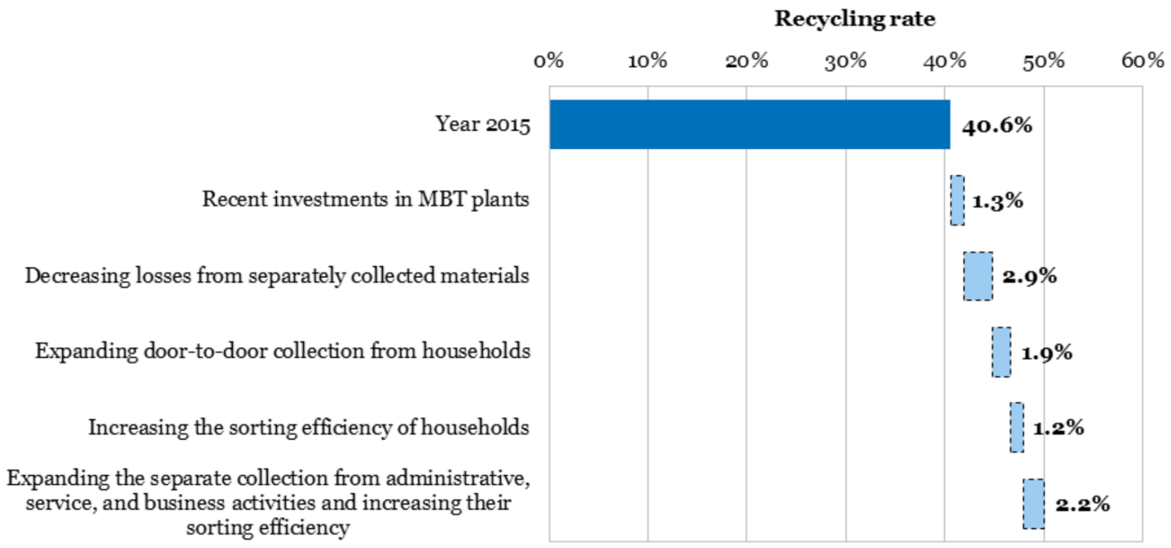


Statistics Finland



Sahimaa, Olli. "Recycling potential of municipal solid waste in Finland." (2017).

In order to reach a recycling rate of 50%, different strategies are planned. Among them, increasing the sorting efficiency of households and business/administrative units represents the largest contribution to the goal, with a combined 3.4% out of the necessary 9.4%.



Sahimaa, Olli. "Recycling potential of municipal solid waste in Finland." (2017)

2- Reducing cost linked to garbage sorting

If the cost of recycling far outweighs the selling price of the recycled commodity, there is little incentive to increase the share of recycled materials in the economy.
Some recycled products are inherently profitable compared to their virgin counterpart (such as metals), some are always unprofitable (glass), while others depend on market prices (plastic).

Reducing recycling cost would allow recycled products to be more competitive on the market and provide more incentives for private companies to operate in the recycling business, as well as being less demanding on collectivities' public finances.

Recycling is expensive mostly due to the associated labor costs. Sorting waste is still a labor intensive process despite the current level of technology.

III. Solution

Studies suggest the cost of sorting garbage could be reduced by 15 to 50% with automatization. (US Environmental Protection Agency). AI sorting systems would allow for a larger proportion of recycled garbage, and of higher quality. There are other benefits as well: faster processing time, and no health risks for workers.

It does make sense to improve the sorting efficiency of garbage by implementing an automatic garbage identification system, to reduce processing costs and reach the recycling target rate of 50%.

Two main goals:
- Classification
- Video object detection

Two primary applications:

Pre-collection: households, shops, restaurants, etc.
In this case the sorting system helps users to put the trash in the right bin. It can be a sensor positioned on top which detects if the trash fits the corresponding bin while emitting light or sounds, or a more complex object which puts the trash automatically.

Post-collection: waste management facilities can use this classification system using a live feed from a camera.

IV. Data

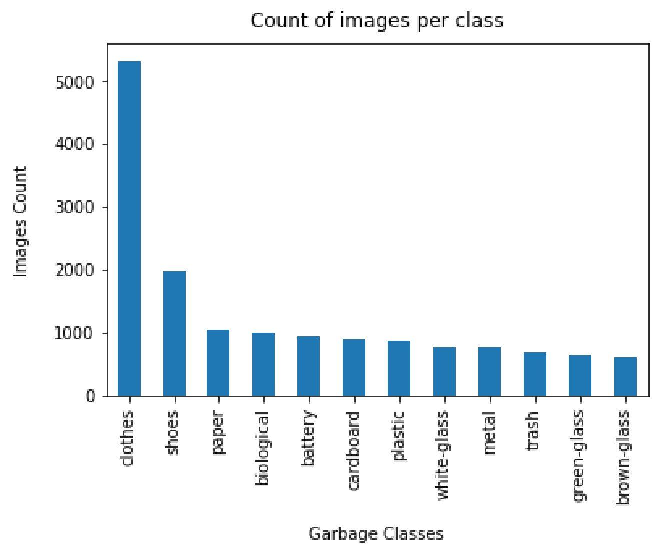
Dataset of garbage is available on Kaggle (15k images) :

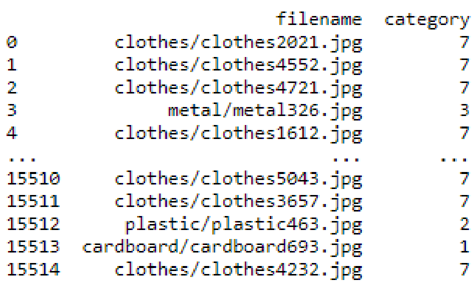
https://www.kaggle.com/mostafaabla/garbage-classification

The dataset contains 12 different classes, from different garbage categories: clothes, shoes, paper, biological waste, batteries, cardboard, plastic, white glass, green glass, brown glass, metal and burnable trash.

The classes are unbalanced, ⅓ of the dataset being images of clothes. The rest tops at around 1k images out of the total of 15k.

Each folder contains images of a particular class, which makes it easy to use during the preprocessing phase by assigning a category number based on the folder name.

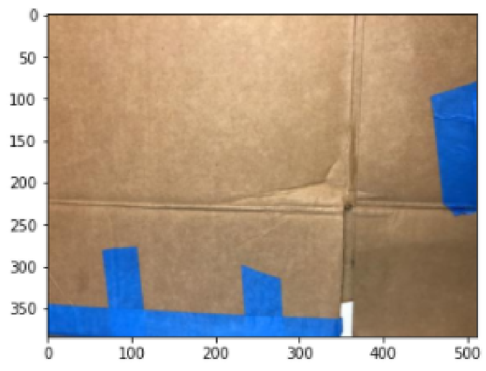




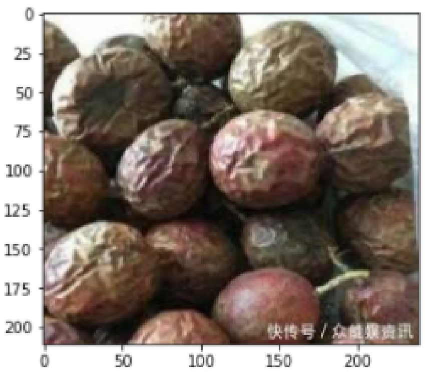
```
                        filename  category
0          clothes/clothes2021.jpg         7
1          clothes/clothes4552.jpg         7
2          clothes/clothes4721.jpg         7
3              metal/metal326.jpg         3
4          clothes/clothes1612.jpg         7
...                          ...       ...
15510      clothes/clothes5043.jpg         7
15511      clothes/clothes3657.jpg         7
15512        plastic/plastic463.jpg         2
15513  cardboard/cardboard693.jpg         1
15514      clothes/clothes4232.jpg         7
```

Images are similar to trash that can be found in bins, being rotten, damaged or in overall poor condition.

```
cardboard/cardboard199.jpg

<matplotlib.image.AxesImage at 0x1c08f8b17c8>
```



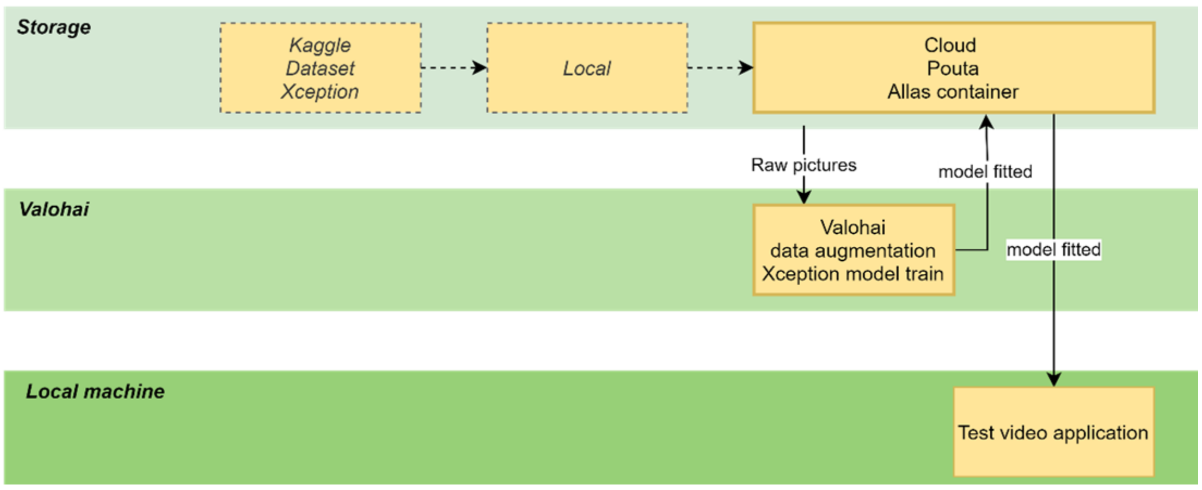
```
biological/biological252.jpg

<matplotlib.image.AxesImage at 0x1c093c88948>
```



V. Pipeline

From the perspective of this course's project, the pipeline goes as follows:

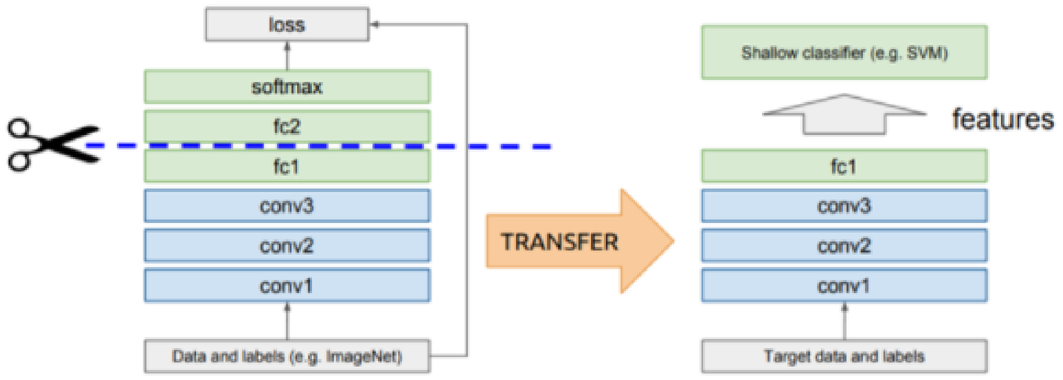
- Data is download locally and a portion is used for testing
- Tests are performed locally
- Final code is stored in github
- Whole dataset and pre-trained network to Pouta/Allas
- Valohai to run the training
- Download trained model to Pouta/Allas
- Test locally real-time video recognition with the trained model



VI. Model

The nature of the problem, computer vision, means it is possible to use a pretrained model in order to reduce training time and improve accuracy.

Example of Transfer Learning:

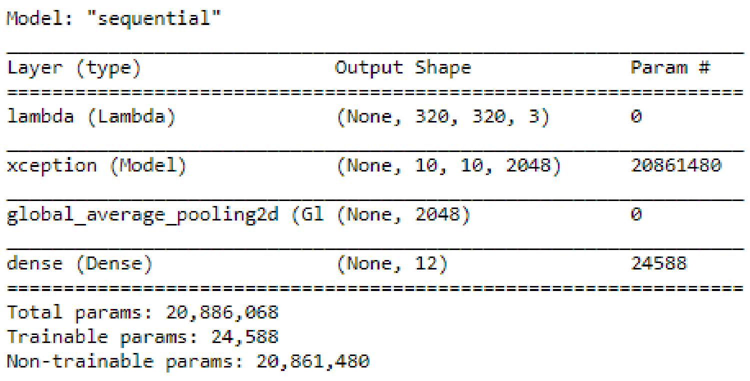


In our case we used transfer learning with Xception, a model pretrained on the ImageNet dataset. ImageNet is a very large dataset consisting of more than 14 millions images placed in 1000 different categories. This allows us to take the knowledge from the pretrained model into ours, and use that knowledge for better classification.

The steps to create the model were the following:

- Create an xception model without the last layer and load the ImageNet pretrained weights
- Add a pre-processing layer
- Add a pooling layer followed by a softmax layer at the end

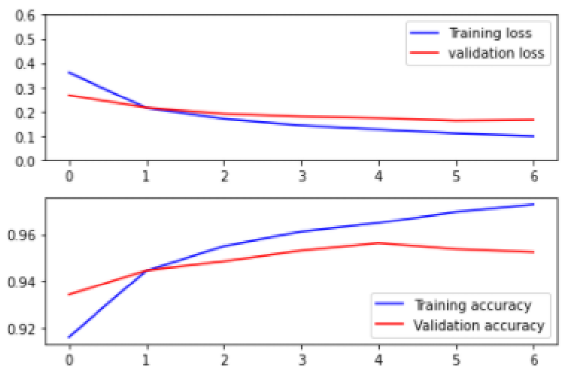
Due to the relatively low number of images from the dataset, it also made sense to use data augmentation. This technique increases the number of images by 10 times by slightly modifying existing images. For example by rotating, zooming, flipping, cropping, modifying colors or brightness.

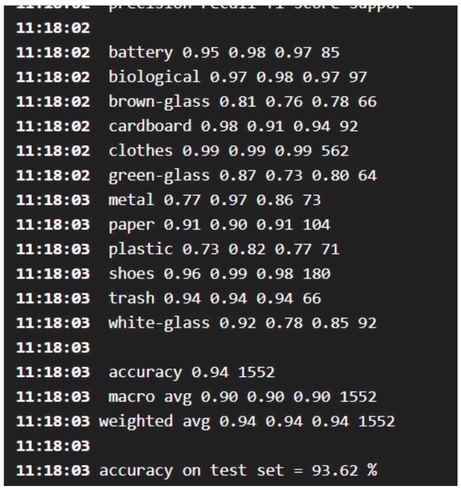


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lambda (Lambda)              (None, 320, 320, 3)       0         
_________________________________________________________________
xception (Model)             (None, 10, 10, 2048)      20861480  
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0         
_________________________________________________________________
dense (Dense)                (None, 12)                24588     
=================================================================
Total params: 20,886,068
Trainable params: 24,588
Non-trainable params: 20,861,480
```

VII. Results

Using Valohai and data augmentation, average accuracy on the test has been 93.62%. However it has unequal performance depending on the type of garbage to classify. While clothes can be identified at 99% accuracy, plastic and metal only get respectively 73% and 77% accuracy. The model is trained in around 2 hours.





```
11:18:02  precision recall f1-score support
11:18:02
11:18:02  battery 0.95 0.98 0.97 85
11:18:02  biological 0.97 0.98 0.97 97
11:18:02  brown-glass 0.81 0.76 0.78 66
11:18:02  cardboard 0.98 0.91 0.94 92
11:18:02  clothes 0.99 0.99 0.99 562
11:18:02  green-glass 0.87 0.73 0.80 64
11:18:03  metal 0.77 0.97 0.86 73
11:18:03  paper 0.91 0.90 0.91 104
11:18:03  plastic 0.73 0.82 0.77 71
11:18:03  shoes 0.96 0.99 0.98 180
11:18:03  trash 0.94 0.94 0.94 66
11:18:03  white-glass 0.92 0.78 0.85 92
11:18:03
11:18:03  accuracy 0.94 1552
11:18:03  macro avg 0.90 0.90 0.90 1552
11:18:03 weighted avg 0.94 0.94 0.94 1552
11:18:03
11:18:03 accuracy on test set = 93.62 %
```

VIII. Real-time video detection

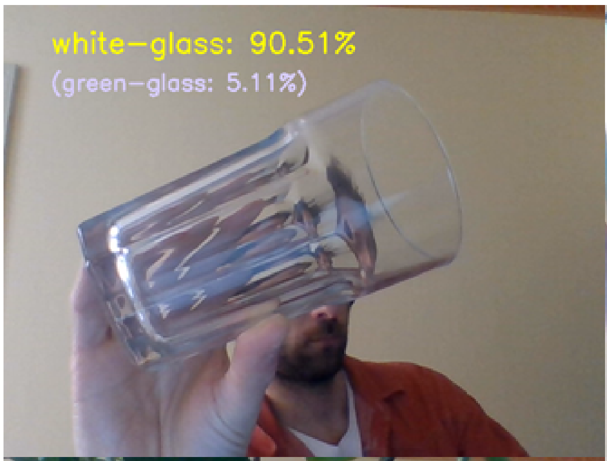
Once the model is ready, it can be used for real-time detection using a laptop and its webcam. The main goal is to prove the model's ability to predict from some real life input and we want to look at two variables: time and accuracy.

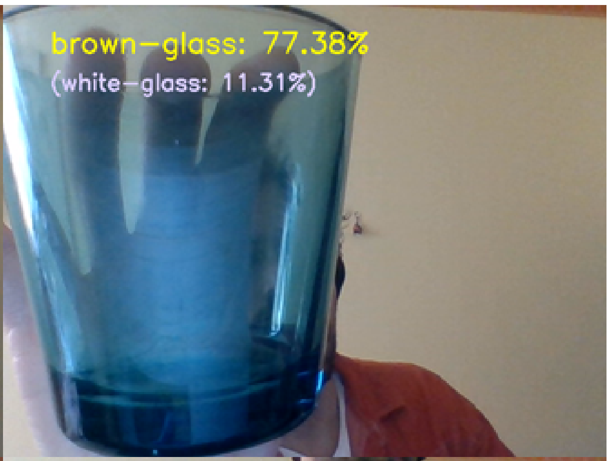
The test is performed by using the Open-cv library for python. This library is widely used for the processing of video in machine learning and object detection applications. The process goes as follows:

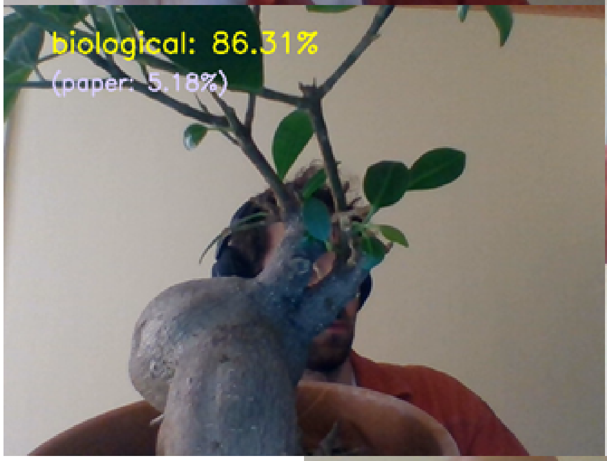
- Open a frame instance that displays video
- Take every frame and process it to be used as an input for the model
- Make a prediction
- Display results

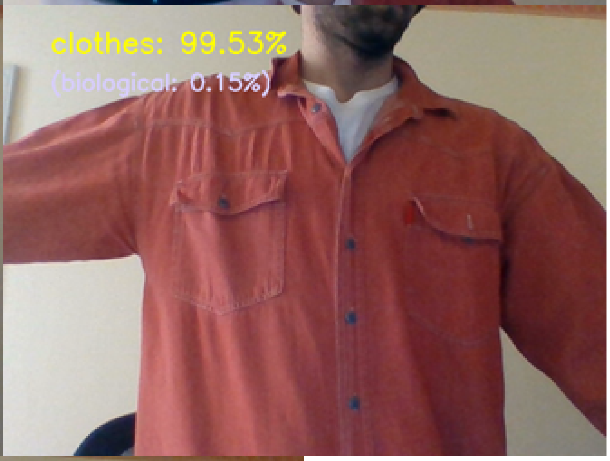
In terms of accuracy, the model seems to respond fairly accurately to whatever is in the frame for as long as it is well displayed and occupies most of the screen. In terms of processing time, the video predictions happen almost immediately with very low latency. It is worth noting that even despite the quality of the camera used, the model is performing in the test fairly well.

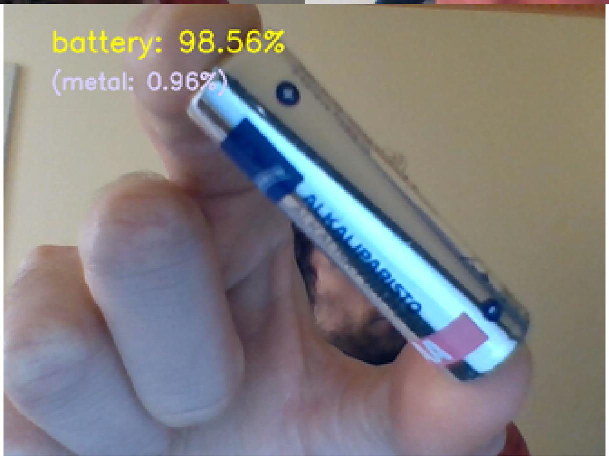
In the following pictures we can see some captions. The yellow labels represent the result with highest probability and the labels in purple the second highest.











IX. Real World Applications

From the perspective of the product we are willing to commercialize, it is possible to make a difference between post and pre-collection pipelines that may serve different applications. While both might be pretty much identical for the training stages, the remaining infrastructure might contain layers in different ways.

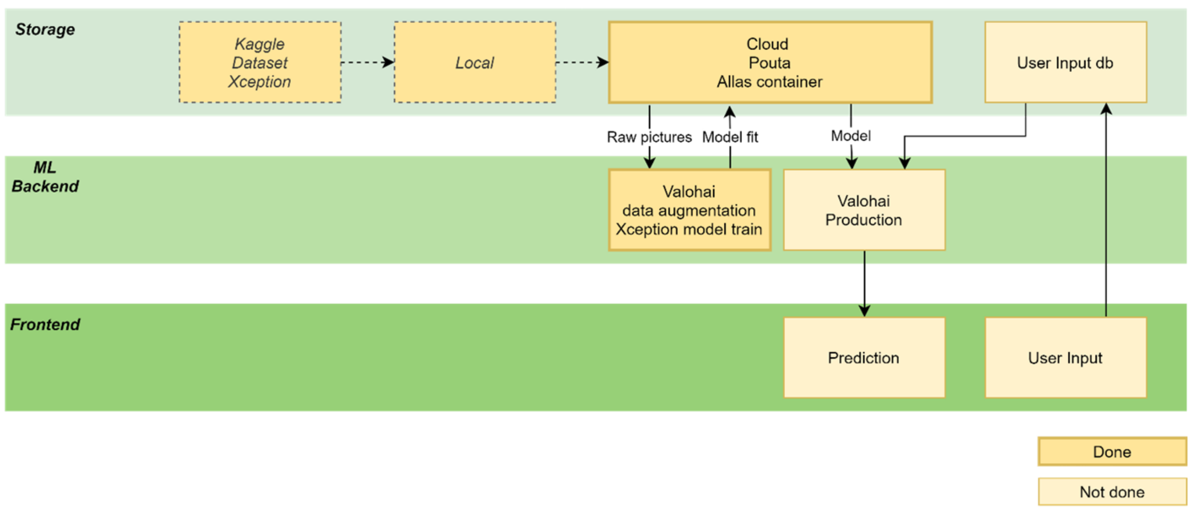
In terms of monetization, either of the two solutions can be sold to the users,e.g. households, restaurant or recycling plants, and the analytics could be sold or given also to a third party who looks after the compliance of these recycling processes like for example the city of Helsinki or the same recycling plants.

1- Pre-collection pipeline

From the pre-collection perspective, one solution could be the one where the users send an input to the cloud, the prediction is performed remotely and then sent back to the user.

In its simplest form, a phone app could help users to determine what bin the item they are about to throw away belongs to. This case is the least attractive since it can be very trivial to sort trash at least most of it and the use of this application could become marginal and eventually forgotten.

A more interesting case would use the model in a system, we can call it a "smart container", that will sort the items into separate bins. In this case the sorting system helps users to put the trash in the right bin. In the simplest way, this application can run on a phone so users can see whether or not they are sorting correctly.

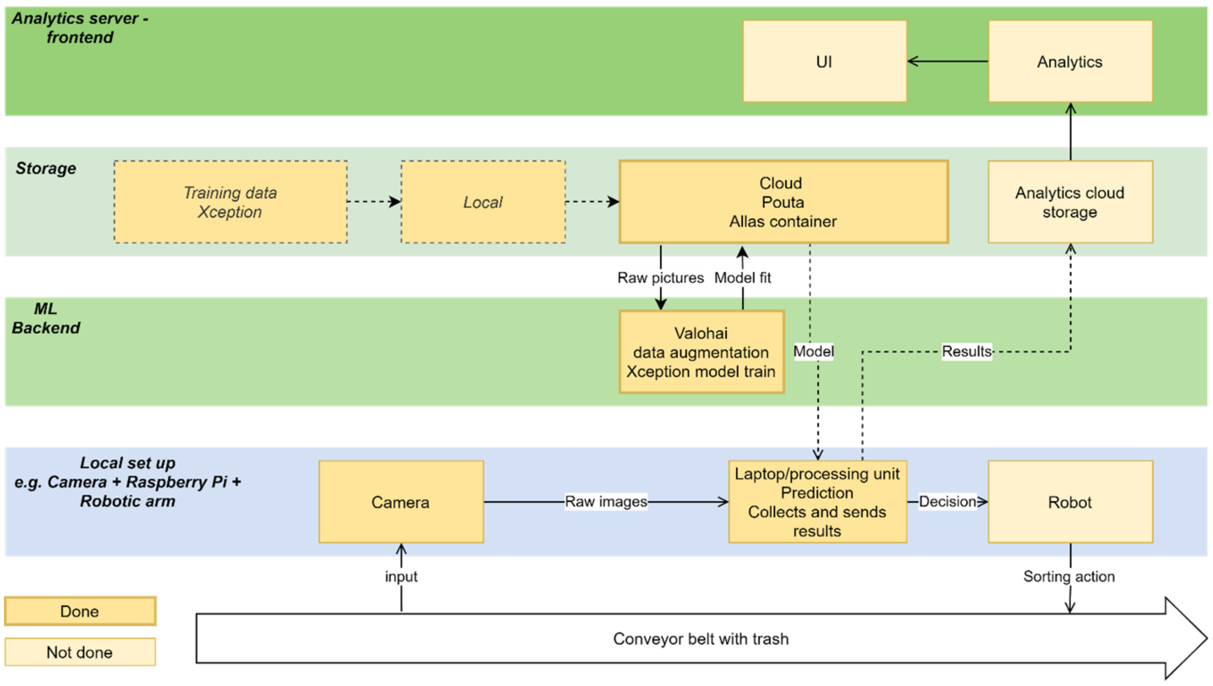


2- Post-collection pipeline

In the post-collection scenario the boxes change a little and there are some extra elements added to the flowchart.

The predictions in this case are performed locally at the “local set-up”. Inside this layer there is a processing element, e.g. laptop or raspberry pi, that receives input as raw images from a camera installed on top of a conveyor belt. The processing unit processes the input, makes a prediction and sends a decision to a sorting entity, e.g. robot arm, that takes a sorting action. The model is very lightweight, a little less than 100Mb, so there is very little restriction to where it could be integrated.

The processing element sends the predictions and other data back to the storage layer to a separate database than the training. This data can then be picked up from an analytics server that creates some insights and KPIs and those can be used from other users through a dashboard.



Analytics server - frontend
UI
Analytics
Storage
Training data Xception
Local
Cloud Pouta Allas container
Analytics cloud storage
Raw pictures
Model fit
ML Backend
Valohai data augmentation Xception model train
Model
Results
Local set up e.g. Camera + Raspberry Pi + Robotic arm
Camera
Raw images
Laptop/processing unit Prediction Collects and sends results
Decision
Robot
input
Sorting action
Done
Not done
Conveyor belt with trash

X. Conclusion

During this work we could probe the concept of using AI for trash identification through transfer learning and the use of a real-time image detection system for trash classification. The model seems to be fairly accurate and its applicability on video recognition was so far successful. However, there is a few points to improve or develop further:

- Training classes. the different classes used in this exercise might have to be contrasted to those of the final user/application as well as re-balanced the amount of items per class during training.

- Model accuracy. In line with the previous point, it would be interesting to improve the accuracy of the model.

- Video detection test set-up. The video collection set-up was performed this time with a laptop and its webcam. The process followed was simply by showing stuff to the camera. In this sense, it would be interesting to simulate a “real” production environment, e.g. a conveyor belt + camera + raspberry pi.

- Sorting entity. In line with the previous point, during this exercise it was not possible to simulate a sorting system however, for such application, there is a few things to consider:

    - If we imagine a conveyor belt, how could we locate the non-belonging item in a section with correctly sorted ones. In such conditions, a location detection system needs to be designed and that could even include the implementation of object detection. Perhaps a system that crops the image into many sections, detects the item, and then identifies the exact location of the object.

    - The nature of the sorting entity needs to be defined too since one could imagine that grabbing glass requires another mechanism than grabbing clothes or bio.

All things considered, from the authors of this work, the project was an enrichment experience and we see there is some potential in such technology and its applications.

XI. Code

Link: https://github.com/sergiocristia/ASD_trash.git

Training

```
def main():
    # function to get paths to the different directories containing the
images, the pre-trained model and others
    paths,labels,path_to_model,archive_path = get_paths()
    df = pd.DataFrame({
    'filename': paths,
    'category': labels
    })

    # Shuffle the dataframe
    df = df.sample(frac=1).reset_index(drop=True)
    print('number of elements = ' , len(df))
    print(df)

    # Modelling<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
    # Defining constants
    IMAGE_WIDTH = 320
    IMAGE_HEIGHT = 320
    IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
    IMAGE_CHANNELS = 3

    from keras.models import Sequential
    from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense,
Activation, BatchNormalization
    import keras.applications.xception as xception

    xception_layer = xception.Xception(include_top = False, input_shape =
(IMAGE_WIDTH, IMAGE_HEIGHT,IMAGE_CHANNELS),
                        weights = path_to_model)

    # We don't want to train the imported weights
    xception_layer.trainable = False


    # initialize object
    model = Sequential()

    # input of shape 320x320x3
    model.add(keras.Input(shape=(IMAGE_WIDTH, IMAGE_HEIGHT,
IMAGE_CHANNELS)))
```

```
    #create a custom layer to apply the preprocessing
    def xception_preprocessing(img):
        return xception.preprocess_input(img)

    model.add(Lambda(xception_preprocessing))
#Lambda(xception_preprocessing)

    model.add(xception_layer)
    model.add(tf.keras.layers.GlobalAveragePooling2D())
    model.add(Dense(len(set(labels)), activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['categorical_accuracy'])

    # We first split the data into two sets and then split the validate_df
to two sets, test and validation
    train_df, validate_df = train_test_split(df, test_size=0.2,
random_state=42)
    validate_df, test_df = train_test_split(validate_df, test_size=0.5,
random_state=42)

    train_df = train_df.reset_index(drop=True)
    validate_df = validate_df.reset_index(drop=True)
    test_df = test_df.reset_index(drop=True)

    total_train = train_df.shape[0]
    total_validate = validate_df.shape[0]

    early_stop = EarlyStopping(patience = 2, verbose = 1,
monitor='val_categorical_accuracy' , mode='max', min_delta=0.001,
restore_best_weights = True)
    callbacks = [early_stop]
    print('call back defined!')

    print('train size = ', total_train , 'validate size = ', total_validate,
'test size = ', test_df.shape[0])

    base_path = archive_path

    # data augmentation set up
    batch_size=64
    train_datagen = image.ImageDataGenerator(

        ###  Augmentation Start  ###

        rotation_range=30,
        shear_range=0.1,
        zoom_range=0.3,
        horizontal_flip=True,
        vertical_flip = True,
        width_shift_range=0.2,
        height_shift_range=0.2

        ##  Augmentation End  ###
    )

    # get input from train dataset
    train_generator = train_datagen.flow_from_dataframe(
        train_df,
        base_path,
        x_col='filename',
        y_col='category',
        target_size=IMAGE_SIZE,
        class_mode='categorical',
        batch_size=batch_size
    )

    # augmentation is not applied to the validation dataset
    validation_datagen = image.ImageDataGenerator()

    validation_generator = validation_datagen.flow_from_dataframe(
        validate_df,
        base_path,
        x_col='filename',
        y_col='category',
        target_size=IMAGE_SIZE,
        class_mode='categorical',
        batch_size=batch_size
    )

    # fit the model
    EPOCHS = 10
    history = model.fit_generator(
        train_generator,
        epochs=EPOCHS,
        validation_data=validation_generator,
        validation_steps=total_validate//batch_size,
        steps_per_epoch=total_train//batch_size,
        callbacks=callbacks
    )

    # serialize model weights
    outputs_dir = os.getenv('VH_OUTPUTS_DIR', './outputs')
    if not os.path.isdir(outputs_dir):
        os.makedirs(outputs_dir)
    save_path = os.path.join (outputs_dir, "model.h5")
    model.save_weights(save_path)
    # serialize model to JSON
    model_json = model.to_json()
    save_path = os.path.join (outputs_dir, "model.json")
    with open(save_path, "w") as json_file:
        json_file.write(model_json)
    # Apply model to test and calculate accuracy
    test_datagen = image.ImageDataGenerator()

    test_generator = test_datagen.flow_from_dataframe(
        dataframe= test_df,
        directory=base_path,
        x_col='filename',
        y_col='category',
        target_size=IMAGE_SIZE,
        color_mode="rgb",
        class_mode="categorical",
        batch_size=1,
        shuffle=False
    )

    # Accuracy
    filenames = test_generator.filenames
    nb_samples = len(filenames)
    _, accuracy = model.evaluate_generator(test_generator, nb_samples)

    # We defined at the beginning of this notebook a dictionary that maps
the categories number to names, but the train generator
    # generated it's own dictionary and it has assigned different numbers to
our categories and the predictions made by the model
    # will be made using the genrator's dictionary.

    gen_label_map = test_generator.class_indices
    gen_label_map = dict((v,k) for k,v in gen_label_map.items())
    print(gen_label_map)

    # get the model's predictions for the test set
    preds = model.predict_generator(test_generator, nb_samples)

    # Get the category with the highest predicted probability, the
prediction is only the category's number and not name
    preds = preds.argmax(1)

    # Convert the predicted category's number to name
    preds = [gen_label_map[item] for item in preds]

    # Convert the pandas dataframe to a numpy matrix
    labels = test_df['category'].to_numpy()
    print(classification_report(labels, preds))
    print('accuracy on test set = ',  round((accuracy * 100),2 ), '% ')

    return model
```

Video application

```
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
import numpy
import os

# Change working directory
path = 'C:\\Users\\...'
os.chdir(path)
# create network architecture
import keras
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense,
Activation, BatchNormalization
import keras.applications.xception as xception
from keras.layers import Lambda
# in the following lines we rebuilt the architecture of the neural
network
# then we use the weights we saved from the training to complete the
model
# Defining constants
IMAGE_WIDTH = 320
IMAGE_HEIGHT = 320
IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
IMAGE_CHANNELS = 3
path_to_model =
"./data/xception_weights_tf_dim_ordering_tf_kernels_notop.h5"
xception_layer = xception.Xception(include_top = False, input_shape =
(IMAGE_WIDTH, IMAGE_HEIGHT,IMAGE_CHANNELS),
                    weights = path_to_model)
# We don't want to train the imported weights
xception_layer.trainable = False

# initialize object
model = Sequential()
```

```python
# input of shape 320x320x3
model.add(keras.Input(shape=(IMAGE_WIDTH, IMAGE_HEIGHT,
IMAGE_CHANNELS)))
#create a custom layer to apply the preprocessing
def xception_preprocessing(img):
    return xception.preprocess_input(img)
model.add(Lambda(xception_preprocessing))
#Lambda(xception_preprocessing)
model.add(xception_layer)
model.add(tf.keras.layers.GlobalAveragePooling2D())
model.add(Dense(12, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['categorical_accuracy'])
# load weights into new model
model.load_weights("./data/model.h5")
print("Loaded model from disk")
model.summary()
```

```python
import numpy as np
import cv2
from PIL import Image
import pandas as pd

cap = cv2.VideoCapture(0)
img_counter = 0

while True:
    ret, frame = cap.read()
    #Convert the captured frame into RGB
    im = Image.fromarray(frame, 'RGB')
    #Resizing into 128x128 because we trained the model with this image
size.
    im = im.resize((320,320))
    img_array = np.array(im)
    img_array = np.expand_dims(img_array, axis=0)

    img_class = model.predict(img_array)
    prediction = img_class[0]
    classes = "battery biological brown-glass cardboard clothes
green-glass metal paper plastic shoes trash white-glass".split()
    prob_result = numpy.amax(prediction)
    loc_result = numpy.where(prediction == numpy.amax(prediction))
    class_result = classes[loc_result[0][0]]
    # Pandas with the results from the prediction
    result_pred = pd.DataFrame()
    result_pred["Values"] = prediction
    result_pred["Class"] = classes
    result_pred.sort_values(by="Values",ascending=False, inplace=True)
    # Use putText() method for
    # inserting text on video
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(frame,
                str(result_pred.iloc[0,1])+':
'+str(round(result_pred.iloc[0,0]*100,2))+'%',
                (50, 50),
                font, 1,
                (0, 255, 255),
                2,
                cv2.LINE_4)
    cv2.putText(frame,
                "(" + str(result_pred.iloc[1,1])+':
'+str(round(result_pred.iloc[1,0]*100,2))+'%)',
                (50, 90),
                font, 0.8,
                (255, 205, 215),
                2,
                cv2.LINE_4)
    # End camera and take screenshot
    cv2.imshow('frame',frame)
    if cv2.waitKey(1) == ord('q'):
        break
    elif cv2.waitKey(1) == ord('s'):
        img_name = "opencv_frame_{}.png".format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1
cap.release()
cv2.destroyAllWindows()
```