Dumping Mapper

Detection and Classification of Illegal Dumping Using Deep Learning and Computer Vision

In affiliation, with

University of Twente, Enschede

Developed in scope of the Design Project of Bachelor of Technical Computer Science

By

Group 2

Job Römer s2307103

Illya Averchenko s2207834

Onen Ege Solak s2301938

Wishal M Sri Rangan s2323680

Mian Tashfeen Shahid Anwar s2039761

Supervised by

Andreas Kamilaris

11/11/2022

Contents

Product Description

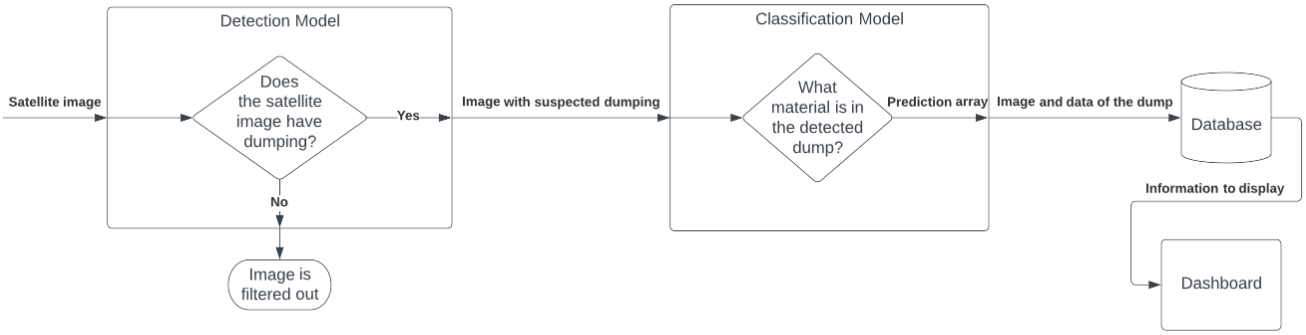
Problem Statement

Many countries across Europe suffer from illegal dumping in areas which are not designated for dumping disposal. Large amounts of waste in these areas has a significant impact on the environment, resulting in contaminated water and soil and diminishing human health due to food-safety issues and spreading of disease through animals attracted to the waste. Additionally, tourism is negatively affected, hurting the economy of the countries. Detecting and cleaning these illegal dumps timely is therefore of great importance and a service that facilitates this desire should stimulate economic growth and waste reuse.

Introduction

Our proposed solution to the problem is called Dumping Mapper. It is a system that uses two deep learning models to detect dumping and classify the materials that a dumping consists of in high resolution satellite images taken from the country of Cyprus. The output of these models is then displayed on an easy to use web application intended for the use of workers of municipalities in Cyprus. Statistics and functionality to monitor dumping sites and cleaning operations are also included.

Below is a simple flowchart of the entire system that represents a high level illustration of how it works:



This report consists of two parts. The first part, Product Description, gives the overview of the project and the design of the end product. The second part, Technical Breakdown, covers the technical aspects of the project and an explanation of the decisions that were made during the project.

Related Work

The main and one of the most recent works that were used as the guidance to the problem of the detection of dumping was done by Nandini Kannamangalam Sundara Raman and Hrushikesh N. Kulkarni (Waste Object Detection and Classification, reference). Nandini K. and Hrushikesh N. achieved F1-score 0.78 which was considered to be a good result. Our goal was to, first apply the same principles of the detection model but for the different types of data, and to improve the performance.

Methodology and Contributions

To evenly divide the work and make the most of the ten week time frame, the team split the project into five separate tasks: **Data preparation**, **Detection Model**, **Classification Model**, **Backend** and **Frontend**. Each team member was assigned to be responsible for one of these tasks while helping out other members of the team wherever necessary.

Job was responsible for the data preparation, which included the annotation of the satellite data and the creation of synthetic data. The satellite data was used by Illya, who was responsible for the creation and optimisation of the detection model, and the synthetic data was used by Tashfeen, who was responsible for the creation and optimisation of the classification model. The output of both models was then sent to Ege, who was responsible for the backend and database, and to Wishal, who was in charge of creating the web application dashboard and organizing the full stack end point connections.

The team worked on a week-to-week basis, loosely following the SCRUM architecture, and had a meeting with the client every week to discuss the team's progress, what the next steps for each task should be and to answer any questions that the team had. More technical questions were delegated to the client's associates. Moreover, the team met every Thursday to internally discuss problems and prepare the aforementioned questions for the client and met on short notice to collaborate and connect parts of the project.

Model Development

Detection model

The initial plan for the detection model was to use the Faster R-CNN state of the art model. In addition, it was planned to use the transfer learning approach and retrain the model with the new data. It was later discovered that, first, the pretrained model does not fit our data parameters, and second, the model implementation was too complex for the scope of the project.

Thus, a CNN algorithm, represented by VGG-16 was implemented with two classes, dumping and no dumping. The CNN algorithm is the first part of the Faster R-CNN algorithm, which later includes the Region Propose Network (RPN). The main module, the CNN, is a deep convolutional network that proposes regions of interest. The second, RPN, utilizes the proposed regions for classification of the objects and applying bounding boxes that represent positioning of the objects on an initial image. It was decided that the first module satisfies the requirements of the project and provides enough details with regards to the content of the images.

Classification model

Development of the classification model consisted of two parts: data and the model.

Data

The team decided to train the model on synthetically generated images with dimensions of 600 by 600 pixels, since materials in a dump could not be ascertained from real satellite pictures. These pictures only contain one type of dumping, material-wise. This was done using a Python script. For more information on this script, please refer to the *Data Preparation* section in the second part of this report. The team also generated images with dimensions of 32 by 32 pixels, as can be seen in Figure 3.0, yet these were only used to initially test the models in a small dataset of 60 images. The 32x32 images lost important information in relative size of trash objects, so the team decided to use the 600x600 images instead (Figure 3.1).

Model

The team was sure from the start that a convolutional neural network (CNN) model would be the best fit for this project. Once the team had a sufficient amount of synthetically created data, the development of the CNN started. The Google Colab environment was used to develop and host the model. The data was uploaded into a Google Drive that was then mounted to the environment. After this, the layer structure of the CNN was designed. A combination of Convolutional layers , Max Pooling Layers, Fully Connected Layers and Flattening layers were considered. The activation function used is called RELU and for the output, a SOFTMAX layer is used.

The model can classify four different types of waste (cardboard, wood, metal and plastic), but this number can be increased in the future. The output of the model is in the form of confidence values in percentages for each class (Fig 3.2).

After this, the model was compiled and the hyperparameters were passed in, along with an optimizer for the learning rate. There are many potential optimizers that could be used. The team tried using the Adam optimizer and RMSprop optimizer.

Model Training

Detection Model

The model was trained on the 'dump_sorted' dataset with over 1000 images. The number of iterations, also known as epochs, was 20. The proportion of the dataset is 80/20, where the first are no_dumping and the last are dumping images. In order to counter data disbalance, the data augmentation was applied, which then changed the data proportions to 65/35 respectively. The model architecture was developed according to the VGG-16, with 5 blocks. The model was trained with 20, 25 and 50 epochs. The best result was achieved with 20 epochs and 70/20/10 ratio of train, validation and test sets respectively. The model training progress and the results of the model are presented in the Model Testing section.

Classification model

The model was initially trained on the 32 pixel by 32 pixel images. A training set of 60 images differing in material was used to train the model. These images were small in size, so the model did not need a lot of layers to learn their features. The team initially tried with a convolution layer taking in a size 4x4 for the convolution filters and a size of 2x2 for the maxpool layer. The Keras Adam with a learning rate of 0.01 was used. Moreover, the model learns better if it can train on bigger batch sizes. The limitation of this is that the system runs out of ram if too many images are processed.

During the later weeks of the projects, the model was trained on a dataset of 600 by 600 pixel images spanning more locations. Near 100 images were created for this purpose, almost 20 pictures of each type of trash. This data, like with the 32x32 images, was split into training, testing and validation sets (Fig 3.3).

The images are then normalized, meaning the pixel values of the images are divided by 255 so that all the images color values are from 0 to 255 pixels and then we import the dataset from the drive and make a dictionary of that data which contains both our image data and their labels. Then the model is fit to the data. With the bigger pictures that were used comes the fact that CNN needs different layers and more time to learn from it , also the model also has to be tuned. Moreover, the model also needs a lot of images to actually have better accuracy, rather than only a few images. The team has tried training with almost 70 pictures and have noticed that

the network starts to converge but cannot get a good accuracy. The only problem that the team faces right now is that there is not enough time to make more data.

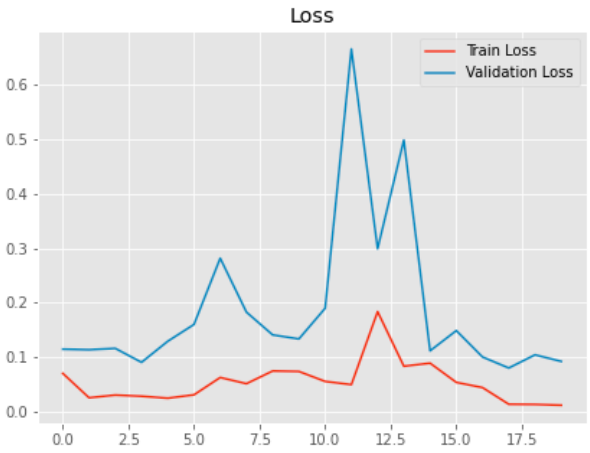
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 597, 597, 64)      3136

 max_pooling2d (MaxPooling2D  (None, 298, 298, 64)     0
 )

 conv2d_1 (Conv2D)           (None, 295, 295, 128)     131200

 max_pooling2d_1 (MaxPooling  (None, 147, 147, 128)    0
 2D)

 conv2d_2 (Conv2D)           (None, 144, 144, 256)     524544

 max_pooling2d_2 (MaxPooling  (None, 72, 72, 256)      0
 2D)

 flatten (Flatten)           (None, 1327104)           0

 dense (Dense)               (None, 64)                84934720

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 85,593,860
Trainable params: 85,593,860
Non-trainable params: 0
```

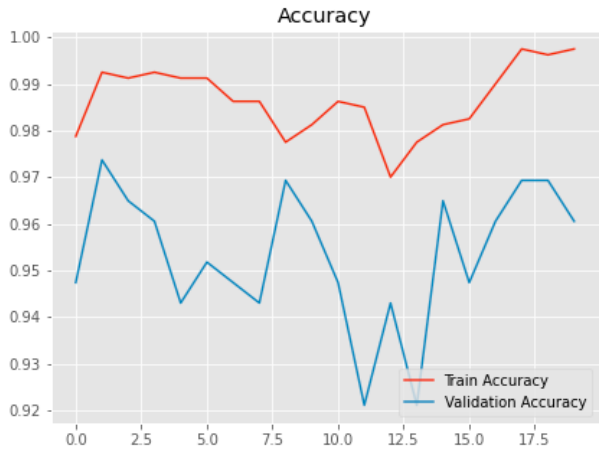
Note : The number of training parameters has been kept small due to the lack of available ram resulting in a crash.

Model Testing

Detection model

The training progress of the best result is represented below.





Here, the validation accuracy of the model reaches 96% and the validation loss rate is below 0.05.

Additionally, the Precision, Recall and F1-score were used to assess the performance of the model.

	precision	recall	f1-score	support
0	0.96	0.99	0.97	76
1	0.97	0.92	0.95	38
accuracy			0.96	114
macro avg	0.97	0.95	0.96	114
weighted avg	0.97	0.96	0.96	114

Precision - is a measure of how many of the positive predictions made are indeed correct. Recall - is a measure of how many of the positive cases the model correctly predicted, over all the positive cases in the data. F1-score - is the metric combining both the precision and recall, can be referred to as the average of the two.

$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP}
\end{aligned}
$$

Classification model

For testing the model, the team used synthetic data that was intentionally withheld from the model. For now we have made data with only one type of trash in a location. But our goal is to make synthetic data which contains all types of trash and alongside that it also contains the percentages of all of the different types of trash in the label.

```
Found 20 images belonging to 4 classes.
20/20 [==============================] - 3s 137ms/step
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.00      0.00      0.00         0
           2       1.00      0.25      0.40        20
           3       0.00      0.00      0.00         0

    accuracy                           0.25        20
   macro avg       0.25      0.06      0.10        20
weighted avg       1.00      0.25      0.40        20

[[0 0 0 0]
 [0 0 0 0]
 [5 5 5 5]
 [0 0 0 0]]
```

The output of the classification model looks as follows.

```
{'cardboard': 0, 'metal': 1, 'plastic': 2, 'wood': 3}
[[0.21260688 0.29554123 0.27224338 0.21960849]
```

Experiments

Detection model

The model was trained on the two datasets of different size, 1000 and 2000 images. The main parameters that were changed for the training are: dataset distribution into train, test and validation; augmentation parameters and number of epochs. The best performance was achieved with the dataset of about 1000 images, with 20 epochs, and 70:20:10 ratio of the dataset. The example predictions are represented in Appendix A (Figure 2.3 - Predictions).

Classification model

When the model was trained on 32 by 32 pixel images, our training dataset consisted of only 70 images, yet the classifier managed to achieve validation accuracies **between 40% and 50%**. The accuracy of the model can be enhanced further with a bigger dataset for training the model.

As for the 600 by 600 pixel images, the model performed less desirably, achieving validation accuracies of only **25% - 30%**. The main reason for this is lack of sufficient amounts of data, as we tried with almost 80 pictures of bigger sizes. This could be rectified in the future to achieve better accuracy. This and other improvements are discussed under the Conclusion and Future Work section.

Result and Analysis

Detection model:

The detection model achieved 96% accuracy and 0.96 F1-score. The training time of the model, using Google Collab, is under 3 minutes. With improved data augmentation and larger datasets, it is possible to improve the performance even further.

Classification model:

For now, this model does not have enough data, but we have finished the scripts and the model so that anyone can generate the data from Blender and train the CNN. The maximum accuracy we achieved was around **50%** using 32 by 32 pixel images. For the 600 by 600 pixel images, more computing power and data is needed to reach accuracies higher than **30%**.

Conclusion and Future work

Detection model:

There are still possibilities to improve the output of the model even further. Firstly, as was mentioned, better data augmentation and larger dataset can improve the prediction results. Secondly, the fact that there is a high proportion of False Negatives requires a further investigation about what is causing these False Negatives and, perhaps, improvement of the dataset, both quality and quantity wise, to mitigate the issue.

Classification model:

For the future, more synthetic images should be generated using Blender, which in the ideal situation would make use of all the different locations which are used and detected from the Detection Model. The aim would be to make around 2000 - 3000 images of each type of trash in different locations and create images in which more than one type of material is present. These images would be accompanied by the percentages of trash material in the labels. The accuracy of this model all depends on how many synthetic data pictures are used and how many different locations are used.

Technical Breakdown

Data Preparation - Job Römer

The data preparation part of the project consisted mainly of two tasks: **the annotation of satellite images provided by the client** and **the creation of synthetic images to make classification possible**. An additional script was developed for utility purposes: an OpenCV script to crop images to the center of an image.

Annotation

The first 3 weeks were spent getting access to the images on the OneDrive of one of our client's associates, since there were some technical difficulties with their account and permissions in the OneDrive. The client had provided us with a total of 15.000 satellite images, divided in sets of 15 between locations from the country of Cyprus.

Annotation of these images consisted of sorting them based on whether dumping occurs in the picture or not and dividing the pictures between two respective folders on a Google Drive shared by the team members. The reactive nature of this process meant that a larger number of these images were prepared in advance and based on input from the team members working on the models, more pictures were annotated (or created in the case of the classification model).

To make the process more efficient and streamlined, the locations were initially divided between the team members, where Job took more locations to annotate. The table in Appendix B shows how the locations were specifically divided between team members and how far each member got with the annotation.

One important aspect to note is the importance of the title of each picture; this name cannot be changed as it contains important data on the longitude and latitude of the location which can be used to trace back where on Cyprus the dumping occurred. However, due to the cropping that was used for the pictures, the longitude and latitude may not be exactly correct but this is not an issue that makes the system unusable, but rather a small inconvenience that could be rectified in future work if deemed necessary.

During week 6, the team also made a decision to crop the satellite images to their center, which would have dimensions of 600 pixels in both width and height. This decision was made to make model training faster and future annotation easier and more efficient. An OpenCV Python script was made for this purpose, which is also on the team's Github repository.

After 6 weeks of annotating, the decision was made to halt the annotation of the locations of the other team members, partly due to leftover work on their parts of the project but mostly because the type of annotation that was required for the detection model unexpectedly changed to bounding boxes before

changing back to the original sorting annotation. The *Difficulties* section provides some more information about this problem.

The amount of images that were annotated during the course of this project totals 4.634 (only 1000 were used due to resource constraints). The plan was to incorporate more verification of the annotations and to do more annotations, but time constraints made this difficult to realize. Additionally, another 765 images were annotated with bounding boxes using the label-studio software and free trial of the V7 annotator service, though these annotations remain unused in the final version of the models.

Synthetic Data Creation

It is impossible for the human eye to determine what materials are present in a dump from a high resolution satellite image, so the team decided to train the classification model on synthetic data that was also generated by the team. The advantage of this is that the materials of the placed objects are known and the picture can thus be labeled and used for training.

The first task was deciding which software would be used to create the most realistic looking synthetic data. The most reasonable options were scripts using **the OpenCV python library** or **python scripts using Blender and its built-in module and runtime environment**.

Using OpenCV would be easiest and most time efficient, due to the fact that dumping can be simulated with a collection of white dots scattered in an area. Blender would be more time consuming, but would allow the creation of more realistic looking images, since 3D models of real trash objects could be used to create realistic dumping sites that could be placed on satellite images.

Below are two pictures comparing the synthetic dumping sites that were created by both scripts. OpenCV is on the left and Blender is on the right.

The decision to use Blender was quickly made; the trade-off between the efficiency and realism of the pictures was important for the team. The desire was to have the most realistic images possible, to

hopefully more successfully train the classification model. Moreover, another downside of OpenCV is that it ironically runs into the problem that the team is trying to fix by creating synthetic data, namely that it is impossible to label the images created by OpenCV. It is unknown what the white dots represent and therefore the material of the “dump” cannot be ascertained, meaning this picture would be impossible to label, like the satellite images.

A workaround for this would be to create custom “objects” by using the dots or other shapes that OpenCV provides and placing those on the map instead. The team felt that this would take too much time for a sub-par result and therefore chose to go with Blender instead.

Blender

This script was adapted from one that our client shared with us and it works in a straightforward way, in that it chooses a spot on a background and places specific trash objects in this spot to create fake dumps. Satellite pictures without dumping were carefully curated for potential spots where dumping would most likely occur. From observations made during the annotation combined with logical thinking, the team determined that dumps most often occur in places that are easily reachable by car, meaning near roads or open areas connected to roads. The previously shown pictures demonstrate this observation and its practical application as well.

The curated places were point-annotated using label-studio, until a coordinate dataset was created of potential dumping locations. The Blender script then iterates over this dataset and corresponding backgrounds, which are mapped to a background plane, and places dumping objects of certain materials on this satellite image. Variables determine the number of objects in the dump, their scale and rotation and how far apart the objects can appear from each other, which proved invaluable in experimenting with the objects.

The objects that were used were mostly free objects curated from sketchfab.com, a website hosting downloadable Blender objects. Most of them did not have textures/materials and differed in scale, so the sizes of objects were manually altered to be more in line with actual real-life trash objects and materials were created to simulate colors commonly found on trash objects (red soda cans and beige cardboard boxes for example).

Once the objects are placed on a background mapped to the plane, the picture is rendered from a top down perspective. While there is no implementation for it in this system, a future improvement would be to add the material present in the dumps to the end of the title of the picture automatically, thus labeling it. Within the duration of development of this project, these pictures were manually renamed by the team.

The pictures are rendered with dimensions of 600x600 pixels and labeled through their title. As explained, the team also experimented with manually cropped 32x32 pixel images created from the generated synthetic images for the classification model, but due to loss of relative size of the objects (the size of objects related to a house for example), 600x600 pixel images were used instead.

In total, 87 600x600 images were generated to train, test and validate the classification model, which admittedly is a very low amount of images. The *Difficulties* section sheds some light on why there are not many images. 9 more 600x600 images were generated and curated with the script to show the realism of the data that could be created with the script. These contain many types of trash objects placed in spots that dumping would usually occur. Lastly, 80 32x32 pixel images containing one type of waste were created for the classification model as well, but these remain unused for the final version of the models.

Difficulties

Bounding boxes versus sorting labeling

During week 6 of the project, the team came to a dilemma about the detection model, or rather the data that would be used to train it. Due to a miscommunication between the team and the client, there was discourse in the way to proceed: the model that was being developed was a deep learning model and it used a dataset that required bounding boxes to work, while the client believed the team was working on a contrastive learning model for which labeling by sorting would suffice, resulting in two different sides of what the team believed to be correct. In the end, the team decided to switch to bounding box annotation since this would require less effort than researching and writing a completely different model.

One week later, the team found another dataset that would work with the sorting annotation and switched back to the prior annotation method. This detour significantly reduced the time the team was able to spend on the annotations and generation of synthetic data, explaining the relatively little number of images.

Blender

The team had trouble understanding and using the Blender script, as none of the members were familiar with the use of this software nor its coding environment. It took 4 weeks to get a preliminary piece of data out of the script, partly because there was little to no documentation or explanation and the fact that questions needed to be referred to our client's associates, which would introduce unavoidable delay since they would not be available at all times. This process made using the Blender script very time consuming and this constitutes the main reason as to why only a small number of realistic images could be generated.

Moreover, the pictures created by this script are not completely realistic due to weird mapping/skewing of the background images. These transformations of the images make it very hard, if not impossible, to accurately place trash objects in very specific places (resulting in objects in places where the coordinates were not placed in the dataset, like on top of trees), as well as disrupt the relative size of roads and cars.

In the interest of time, it would probably have been better if the team had chosen OpenCV to create the synthetic data, but the team does not regret its decision, as the Blender images show much more potential than the OpenCV images, especially if these images are improved further.

Future Work

In the future, multiple aspects of the data for this system could be improved. The most prominent of these improvements lies in the amount of data. Generally, more data is better for deep learning models and the relatively small datasets that were used for this system hurts their potential.

Annotation

While a large number of images were annotated, there is always the possibility of human error influencing the models. The team had used a verification process for the annotation, but due to time constraints and temporary switch to bounding box annotation, this process was abandoned during the sixth week of the project. To reduce the amount of human error in the dataset, a fully adopted verification step where team members verify the annotations made by other team members would be beneficial for future use of the models and dataset.

Moreover, the data should have been cropped before it would be annotated. Annotating 6000x8000 pixel images added a significant amount of required time due to zooming and careful consideration of what the center of the image would be. With a cropped image, both of these parts of the process would be eliminated, saving minutes per image.

Synthetic Data

The main aspect to improve on is to make the images more realistic in a more efficient manner. While adaptations were made to have the script produce multiple images in one iteration, the most time consuming task is the curation of background images and consequent labeling using the coordinate dataset. If there would be a quicker way to label the images in this way, it would significantly increase the amount of images that could be created in a shorter time frame.

To create more realistic images, multiple things should be taken into account. The first issue to solve is the perceived issue with the skewing of the background image and mismatch of the coordinate dataset and the location where the objects end up. As of now, the team does not know what causes this issue, but it is creating less realistic images and should be researched further.

The second improvement would be to calculate the size of objects in comparison to the resolution of the background image. Currently, the size of the objects is decided by the human eye, which may cause objects to be unrealistically large or small. Given the resolution (pixel to cm ratio) of an image and a reference object/its dimensions in real life, the size of the Blender object could in theory be calculated to be as precise as possible, increasing realism of the data. This will be challenging due to the fact that Blender does not have a pixel unit system, meaning one would need to find out what 1 pixel is in the metric or imperial system that Blender supports.

The third and final improvement is to add more objects (and material types), possibly even custom objects and texture materials, to populate images with. This would likely increase the overall usefulness of the classification model (since it would have more classes) and increase its performance due to the introduction of new objects of the same material.

Special Thanks

The team's gratitude goes out to Chirag, our client's associate and specialist on data preparation, for his supervision and aid during the course of this project.

Detection Model - Illya Averchenko

Research

Before selecting a model for my task I decided to look at the nuances of the satellite images that we were provided with. I wanted to get a better understanding of the type of data we are working with and select the most appropriate model based on my observations. Thus, I outlined for myself the spatial resolution parameter, sometimes, called Ground Sample Distance. Spatial resolution describes the minimum separation distance between two objects to differentiate between them. The spatial resolution is measured in distance between the centers of two adjacent pixels. Having that in mind, I already saw a possible problem with using the majority of the models that are available. The essence of the problem is that an arbitrary model resizes images and generates a new image with a different spatial resolution. This changes the number of pixels occupied by the objects on the image. Thus, eventually, after convolutional layers and MAX-Pool layers, the objects "vanish" from the image without being detected by the model. To tackle this problem, Faster-RCNN was considered. The algorithm of detecting small images goes as follows: first, the model does the partitioning of the test image into a smaller images; then, adapts the image to the desired model image size with suitable overlap between partitioned images; import the test image into the model; finally, measure the impact of the size of the images on accuracy to decide the most suitable object size.

The graphical representation of the algorithm is provided in Appendix A *Figure 2.0 - Ground Sample Distance* and *Figure 2.1 - Faster R-CNN split*.

Initially, I intended to use the transfer learning approach - use weights from previously built models and retrain the model with our data. The assumption was that this approach could save the model development time and provide high accuracy.

After testing the Faster R-CNN model that was pre-trained on the COCO, KITTI and Open_Images datasets I realized that transfer learning is not the solution for our task. The reason for the poor performance (very low learning rate and accuracy <5%) and slow training is due to the lack of satellite images in these datasets as I discovered later. The model is simply not used to the satellite type of view.

Thus, the backbone of the Faster R-CNN model was trained on different data and therefore has weights that are not suitable for our data. The transfer learning cannot change the backbone of the model but only the detection head and so the detection model will not perform well on our dataset, as already tested.

Eventually, I trained the model from scratch. I implemented the first part of the Faster R-CNN which is a CNN model to classify the ROIs of the images into the two classes, dumping and no dumping. This model provided sufficient accuracy for our task.

Set up

Hardware:

I decided to use Google Collab Pro platform, because it provides convenient execution of code and a decent performance of the model training. In Google Collab I used the NVIDIA Tesla P100 GPU with advanced RAM consumption of the environment.

Software:

I used TensorFlow Object Detection API to set up a model faster and adjust the layers of the model more convenienlty. In addition, I used a number of libraries such as Keras, NumPy, Pandas, CV2, Matplotlib, Sklearn and OS.

Data processing

Before feeding the model with the data, the dataset had to be prepared. One of the specifics of deep learning models is the data that is passed to the model to get predictions and the data that the model was trained on have to be of the same size. The model input size was set to be 600x600px which provides enough data points for the task and does not overload the training process due to the low data size. The images that we received from the client were of approximately 6,000x8,000px. Thus, it was necessary to either scale down the images to a much smaller size or to crop the images into smaller segments for both the model training and the predictions improvement.

The idea is to split the images into sectors of 600x600px and start with only central sectors of the images for training the model. Later, expand the dataset with more sectors, including a larger “focus”. The main advantage of this is to save as much data when passing the 6000x8000px images to the model which has 600px as the minimum and 1000 as the maximum size of the image. Resizing the images to the input format of the model would cause downsizing which would change the spatial resolution of the image and result in data loss. It is also generally a good practice to reduce the size of images when training. The full size image was partitioned into 144 smaller images of same size - sectors, each of size 600x600px (Appendix A *Figure 2.2 - Image cropping*).

There were several sets of pre-annotated images. The first dataset that the model was trained on is about 1000 images separated into two folders dumping and no_dumping, according to the content of the images. The second dataset was prepared to achieve better performance of the model and about 2000 images were prepared for the training.

Besides, the data was randomly split for the model training, test and validation. Random split and shuffling provides unbiased evaluation of the model and improves the performance. The

datasets were split into 0.7:0.15:0.15 and 0.8:0.1:01 ratios for train, validation, test sets respectively.

Model Configuration

I implemented a CNN model, which has the structure of the VGG-16 network and has the following architecture:
1.InputLayer - 2. ZeroPadding2D - 3. Conv2D - 4. BatchNormalization - 5. Conv2D - 6.BatchNormalization - 7. Activation - 8. MaxPooling2D - 9. Dropout - 10. Conv2d - 11. BatchNormaliztion - 12. Activation - 13. MaxPooling2D - 14. Dropout - 15. Conv2D - 16. BatchNormaliztion - 17.Activation - 18.MaxPooling2D -19. Dropout - 20. Flatten - 21.Dense - 22. Activation (ReLU) - 23. Dense
Total: 5 blocks, 1,500,000+ parameters.

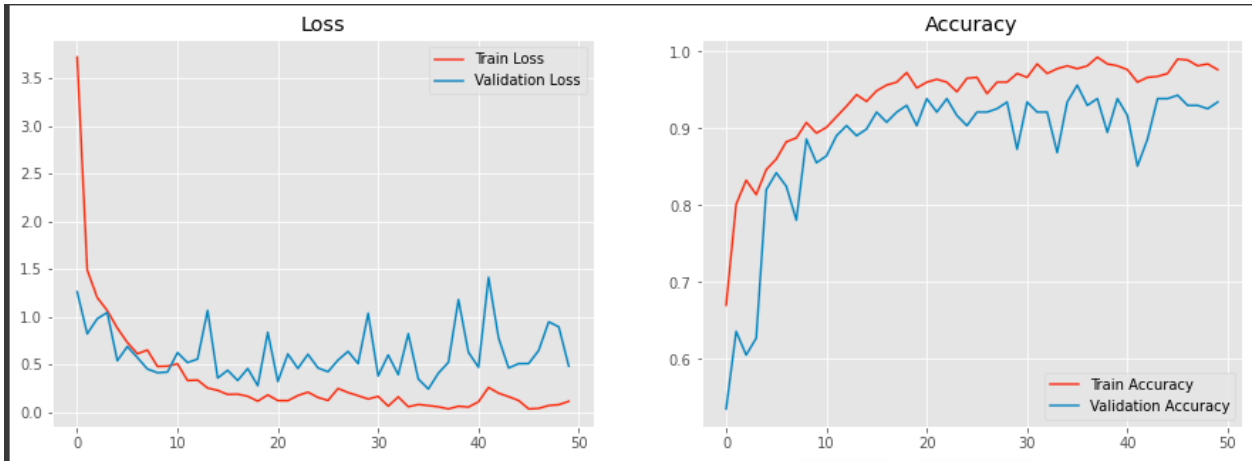
The model classifies the new data with 92-96% accuracy and still can be improved with larger datasets and augmentation (although more RAM is needed). 20-23 epochs were enough to get good results.

For the data augmentation the following functions were applied: GaussianBlur, SigmoidContrast, Fliplr, Crop, LinearContrast, Multiply, Affine.

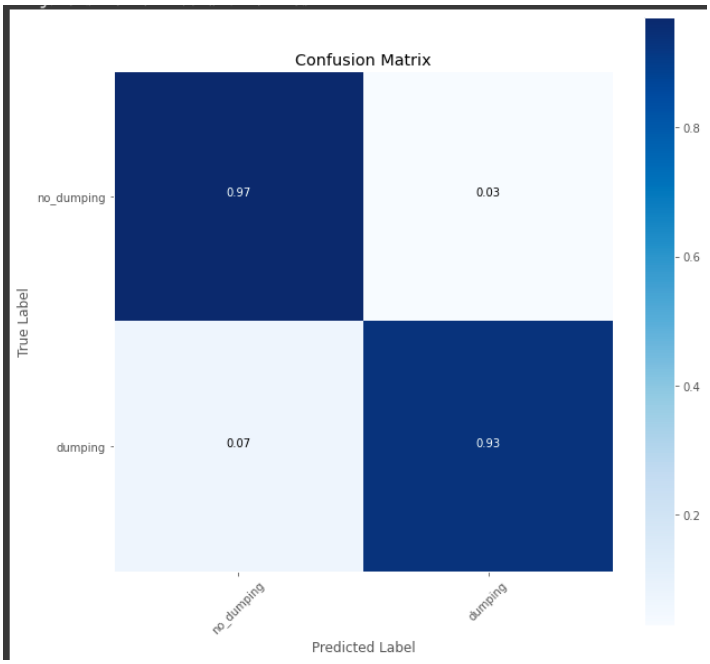
Observations

Although the model provides 96% test accuracy and 0.96 for F1-score, it still gives a lot of False Negatives. Having more False Negatives leads to skipping dumping on locations and thus, failure of the system. However, in our context it is more preferable to get a false alarm - more False Positives. Thus, if the model detects dumping on an image when in reality there is no such, then the chances that the user of the system skips dumping will be quite low.

The graph below shows the training accuracy and the loss progression through the training of 50 epochs. It is clear that after 20 epochs the accuracy of the model does not improve compared to the final result. In this example, we are particularly interested in validation accuracy which shows the prediction accuracy of the model on the test set - images that the model is new to.



In addition, the confusion matrix was created to see more detailed information about the predictions. Here, the values of True Positives and True Negatives are high, which indicates that the model correctly predicts on the classes. However, the value of False Negatives can be improved further - 0.07. Thus, there is a 7% chance that the model categorizes dumping as no dumping, and thus skips the objects of interest on the images.



Difficulties

The main difficulties that I faced during the project was to select an appropriate model architecture. So far, there are very few projects available that are aimed at detection or classification of objects from a satellite or even drone view. Most of the datasets and pre-trained models contain a horizontal view perspective, not from a “bird’s-eye view”, like from the

satellite's camera perspective. Thus, it took a lot of resources to prepare the necessary data and the model specifically for our task.

Besides, there was a lack of resources available on Google Collab. In order to train the model with more data, for example, with the second dataset that has over 2000 images, it was required to have more RAM available, which required a more expensive plan to purchase and was out of the available expenses of the project. Besides, it took a lot of time to decide on the size of data for the model and required additional research.

Future Work

There are still possibilities to improve the output of the model even further. Firstly, as was mentioned, better data augmentation and larger dataset can improve the prediction results. Secondly, the fact that there is a high proportion of False Negatives requires a further investigation about the reasons for having ones and, perhaps, to improve the data quality of the datasets which might be initially the reason for this.

Classification Model - Mian Tashfeen Shahid Anwar

Research

Research was conducted to determine what type of neural network should be used for this task. The team learned that there are three different types of neural networks ANN , RNN and CNN. The CNN network is mostly used for image classification tasks, so this makes it a good candidate for our task. The team determined that a convolutional neural network with RELU and softmax layers would be the best for this task, because convolutional neural networks are built to learn from pictures by breaking them down into smaller parts.

Apart from this, the team found out that there are three different techniques to train a CNN (Training from scratch , Transfer learning and feature extraction). In the case of this model, training from scratch would be the most obvious to use, due to the type of data that would be used.

A substantial part of the research also lies in what kind of locations the team should be using in our data, what kinds of trash would be put into those pictures and where the data would be positioned. Moreover, one of the most important parts of the research process was to determine how to label the synthetic data, because the end goal is to have percentages representing how much of one type of trash a dump consists of. For that, the team can actually make labels in the form of percentages while making the synthetic data.

At first, the model would be attempted to train on only one kind of material and if this would be successful, the model would be trained on mixed data. One of the challenges in this project is to figure out the real size of the layers because the type of trash is very small. This was also the main reason behind using 32 pixel images by 32 pixel images first.

Using Convolutional Neural Network for Image Classification | by Niklas Lang | Towards Data Science

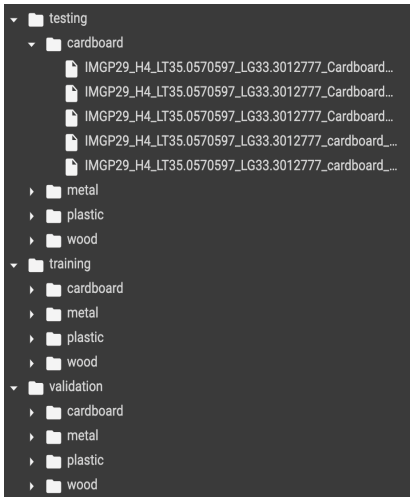
Tensorflow Course :
https://www.udemy.com/share/101Wje3@uvz6llwZRQ9zfzB69zFeErMTI1ndwY0S8C-L8IUQM1QRQWsKHakVex7GFCDBWsr-EA==/

Video Explaining CNN:
https://nl.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html?gclid=Cj0KCQiAgribBhDkARIsAASA5bv9uNwmxSqx5vvHggJuJyDnw0BNyZxnqqCerM6rLdeXx7gbj83CDMIaAk_9EALw_wcB&ef_id=Cj0KCQiAgribBhDkARIsAASA5bv9uNwmxSqx5vvHggJuJyDnw0BNyZxnqqCerM6rLdeXx7gbj83CDMIaAk_9EALw_wcB:G:s&s_kwcid=AL!8664!3!604194414418!p!!g!!convolutional%20neural%20networks&s_eid=psn_45716723653&q=convolutional%20neural%20networks

Setup

Data

For making the data, Blender was used. All pictures were collected into Google Drive and sorted into different folders (cardboard, metal, wood, plastic). Then, the data was split into 3 datasets (training, testing, validation).



The name of the image we would create was also important; we needed to keep the coordinates of the locations in the title. The pictures need to be classified and uploaded to Cloudinary. This data would need to be processed by the frontend, so a POST request is also sent to the frontend.

Code

After that, Google Colab was used. The Google Drive was mounted into the Google Colab environment and the latter was set up. Code blocks would import the different libraries that are required for the model, some of which are shown below.

```
import tensorflow as tf
import matplotlib.pyplot as plt
import cv2
import os
import numpy as np
from tensorflow import keras
from keras import utils
from keras import preprocessing

# import tensorflow
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import RMSprop

!pip install requests
```

Lastly, there was a problem with sending data to the server using a POST request. This is why the team pivoted to using PyCharm in addition to Google Colab.

Now, we have the code for the model and for uploading the files from the model to the Database in GoogleColab and we have the code for retrieving images from the Database and making a POST request to the frontend in Pycharm. In Future Work, these two files can be combined and the code can be run together in a different environment, excluding Google Colab. The problem with Google Colab is that it uses a hosted runtime, which means that the moment a connection with the backend is made, it is rejected automatically.

Data generation

For generating the synthetic data, a Blender script is used. Before using this, a satellite image is taken and an area without dumping is cropped a square of 600 by 600 pixels.

It is important to be specific in which locations are used. If dumping is placed in the wrong location or if the dumping overlaps too much, the data set would not be useful. On the other hand, more objects have to be made to actually get a good accuracy when the model is tested on the real dumping image.

The initial plan was to generate data of 32 by 32 pixels because we wanted to make bounding boxes around the different types of objects on the picture that would be received from the detection model (600 pixels by 600 pixels). Then getting each picture per object separately and testing it with the model. Then we could have made the model to

- Get picture from detection model
- Detect specific objects
- Take 32 by 32 pixel image of that object (Model would be trained on synthetic data 32 by 32 pixels)
- Classify it
- Then count the number of pictures classified in each category and calculate the percentages of the type of trash present

On the other hand, when 600 by 600 pixels images were used, the plan changed. The new plan was:

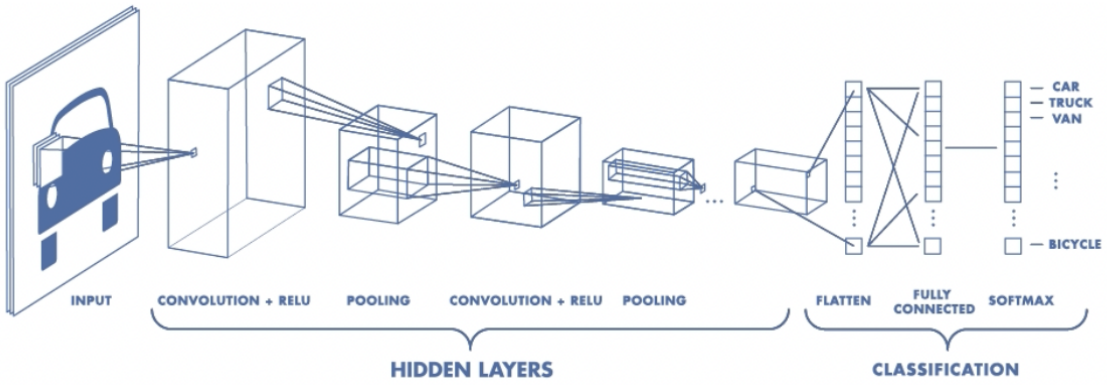
- Get picture from the detection model
- Classify it
- Get percentages

Changing the number of pixels to 600 would result in being able to classify the whole picture in one go, because differentiating each picture based on percentages would now be possible. For example if 50 cardboard objects, 50 metal objects, 50 wood and 50 plastic objects are placed in a synthetic data image, it will make the percentage of each object to 25%. This information can be used to label the image. The softmax layer already provides us with the percentages of each class. However, the labels would be given by the name of the picture.

Model training / testing

For training the model, a CNN which takes in convolutional layers, maxpool, dense and flatten layers was set up. Parameters can be given to these layers to, for example, change the size of the convolution filter from the convolutional layers, change the size of maxpool layers which basically makes the filters smaller and increase or decrease the amount of convolutional layers and maxpool layers. Lastly, a flattening layer and dense layers are used.

Experiments can also be performed with different learning rates, different optimizers, different and weight initializers. There is also a possibility to try different activation functions but in this case the best would be RELU and softmax.



During the model training, the biggest role is played by the amount of data that is used and the quality of said data. In this case, a lot of images would be needed.

A lot can be changed in the model given more images. To train the model better, very realistic images need to be put in (Figure 3.1). Otherwise, it would be very hard to make the model predict correctly on a real image.

To actually train the model, you can use the .ipynb file alongside the pictures in your own Google Drive. For testing the model, the testing dataset can be provided to the model and it will try to predict the class of all the images in the testing folder.

Difficulties

The difficulties that were faced within this task are mainly related to the generation of data. There was little time to create realistic images and a satisfactory number of them. The model was developed early but training and optimizing the model to get a better accuracy is hard without a proper amount of data, because the model is trained from scratch.

With enough time, sufficient data can be generated, yet the model still needs to be trained. The first problem that was encountered, had to do with batch sizes and required RAM. An improvement would be to make the batch size smaller because the Google Colab environment keeps on running out of ram. Since this model is being trained from scratch, it needs much more data and processing power.

Lastly, there was not enough time to generate a lot of data and train this model to get a good accuracy.

Future work

In future work, many tasks will have to be performed, which range from making more synthetic data and then trying to train and tune the model to have better accuracy until a point is reached where the model can actually be tested on a real dumping image.

Moreover, the model only detects one type of dumping at a time but a future improvement would consist of extending the model to detect dumping based on percentages of different classes. For that, the synthetic data creation and labeling techniques will have to be altered.

Backend - Onen Ege Solak

This application uses mainly MERN(MongoDB, Express, React, Node) Technologies:

- MongoDB: Database
- Express: Web Framework
- Node: Web Server
- React: Client-side application
- Cloudinary: Drive for storing the images

Moreover, the MapBox library is used for building the geocoding application. The Mongoose library is used for the connection between Node and MongoDB. All these frameworks and applications are JSON object oriented and in this project we are working with JSON objects. Some other libraries such as Bcrypt and JWT tokens are also used for this project. Cloudinary is chosen for storing the images of the detected dumps.

Database

The database consists of two collections: Municipalities (Districts) and Dump.

```
Municipality:

    name: {
        type: String,
        required: true,
    },
    id: {
        type: String,
        required: true,
    },
    username: {
        type: String,
        required: true,
        unique:true,
    },
    password: {
        type: String,
        required: true,
    },
    Respworkers: {
        type: Array,
        workers: [{
            Type: String,
          }],
        required: true,
    },
});
```

```
  Dump:

  long: {
    type: Number,
    required: true,
  },
  lat: {
    type: Number,
    required: true,
  },
  districtId: {
    type: String,
    required: true,
  },
  imageurl: {
    type: String,
    required: true,
  },
  type: {
    type: Array,
    class: [
      {
        Type: String,
      },
    ],
    required: true,
  },
  status: {
    type: String,
    required: true,
  },
  address: {
    type: String,
    required: true,
  }
});
```

The Municipalities (Districts) collection and the Dump collection are connected through the “districtId” field where the id of the municipality is stored within each dump data. In the dump collection, the information of the url of the image inside the Cloudinary is stored in the “imageurl” field and the type of the dump that is gathered from the classification models is stored in the “type” field. The longitude and latitude of the is used for identifying the dump since It is unique to one dump. That is why the long and lat field is requested when calling a route.

Connection with the Front-end

The connection between front-end and back-end is made with Node and Express which are the application tier frameworks of the MERN stack. First, the setup of the server is implemented using Node.js. The server started at localhost and on port 3001. This way the environment for the connection is set. For testing the application side of the project, Postman was used.

```
app.listen(3001, () => {
  console.log("SERVER IS RUNNING");
});
```

After that, the routes were implemented to create a communication between the database and the client side server using Express.js. The middleware for the api’s are implemented so that the routes can be taken to another folder called “routes”. In that table, there are two files: one for the workers and one for the dumps. The list of API calls of the workers file is as follows:

-”./workers”: for retrieving all the Respworkers in a district
-”./delworker”: Deletes a worker value inside the Respworkers field.
-”./addworker”: Adds a worker value inside the Respworkers field.
-”./login”: Provides credentials for logging in a district page.

“./workers” route returns all the workers currently working in the municipality. The route constructs a “muni” document by finding the requested username in the database and once it is found the function returns all the workers that the municipality has with a status(200) message.**See Figure 4.0**

“./addworkers” and “./delworkers” routes work in a similar way. The routes find the document using “.findOne()” and constructs a “muni” document that has the same values as the found document and for adding the worker value it uses ”.push()” function and for deleting uses the “.pull()” function to edit the values inside a field. The “./addworkes” route also constructs another document called “check” with the ".findOne()” function to check if the added worker exists inside the document.**See Figure 4.1**

“./login” route uses JWT to communicate the credentials between the MongoDB and React client server and the password fields for the Districts are manually hashed and added as a value. The route uses JWT token procedure to provide security for the login. With the “./login” route the username (for municipality) and password (for district) are requested from the user to construct a municipality parameter called ”muni” by finding the table according to the provided username inside the database. When the municipality corresponds to the requested username found, it retrieves its hashed password from the database and compares the requested password with the hashed password inside the database using the bcrypt.compare() function. Bcrypt.compare() function hashes the requested password and compares it with the password in the database that is already hashed. If the passwords match, the JWT token is sent to the frontend as well as the login successful message.

```
router.post("/login", async (request, response) => {
  // check if email exists
  const { username, password } = request.body;
  const muni = await municipalities.findOne({ username: username });
  // if username exists
  if (muni) {
    // compare the password entered and the hashed password found
    const passwordCheck = await bcrypt.compare(password, muni.password);
    // if the passwords match
    if (passwordCheck) {
      //   create JWT token
      const token = jwt.sign(
        {
          municipalitiesId: muni.id,
          municipalitiesName: muni.username,
        },
        "RANDOM-TOKEN",
        { expiresIn: "24h" }
      );
      //   return success response
      return response.status(200).send({
        message: "Login Successful",
        username: muni.username,
        name: muni.name,
        id: muni.id,
        token: token,
      });
    } else {
      return response.status(404).send({
      });
    }
    // catch error if password does not match
  } else {
    // catch error if username does not exist
    return response.status(400).send({
    });
  }
});
```

As for the “dumps” file the list of API calls are the following:

-”./getdumps”: Returns all the values of the dump collection

-”./deldumps”: Deletes a dump document from the dump collection

-”./createdump”: Creates a dump document to dumps collection

-”./confirmdump”: Updates the status field to “confirmed”

-”./suspectdump”: Updates the status field to “suspected”

“./getdemps” route constructs a document called “dump” and finds all the documents in the collection using find() without parameters and returns them. **See Figure 4.2**

“./deldump” route constructs a document called dump by requesting latitude, longitude and imageurl. It then calls the function “.findOneandDelete()”. This function uses longitude and latitude as parameters. Cloudinary library on the other hand uses the “imageurl” to erase the image of the dump from the drive. **See Figure 4.2**

“./confirmdump” and “./suspectdump” routes use the same methodology. They first define the field value to be updated and they construct a variable called “dump” with the requested parameters with the “.findOneandUpdate()” function and they update the value that is defined. **See Figure 4.3**

“./createdump” route requests longitude, latitude, imageurl and type fields and calls a function called “reverseGeocoding”. “reverseGeocoding” function connects to the MapBox site to retrieve the address information for the “address” field. It also retrieves the district information from the longitude and latitude which is to be used as a “districtId” to group the dumps according to districts. The dumps are automatically set as “suspected” in their “status” field and the rest of the field values are filled from the request body.

```
function reverseGeocoding(req, res, next) {
  const { lat, long, imageurl, type } = req.body;
  console.log(req.body);
  if(lat,long,imageurl,type){
  var url =
    "https://api.mapbox.com/geocoding/v5/mapbox.places/" +
    long +
    ", " +
    lat +
    ".json?access_token=" +
    ACCESS_TOKEN;

  request({ url: url, json: true }, function (error, response, next) {
    if (error) {
      res.send("Unable to connect to Geocode API");
    } else if (response.body.features.length == 0) {
      res.send("Unable to find location. Try to" + " search another location.");
    } else {
      const address = response.body.features[0].place_name;
      const place =
        response.body.features[response.body.features.length - 2].place_name;
      const district = place.substring(0, place.indexOf(","));
      Dumps.insertMany({
        long: long,
        lat: lat,
        districtId: district,
        imageurl: imageurl,
        status: "suspected",
        address: address,
        type: type,
      });
    }
  });
  next();
}else{
  res.send("Provide all images");
}
}
```

```
router.post("/createdump", reverseGeocoding, async (req, res) => {
  res.redirect(200,"Data Created");
});
```

Future Work

New functionalities such as messaging or improvement of the statistics supported by the backend can be implemented. Some bug fixes and protection against cyber attacks can be improved. The application can be modified and further be used for more local governments such as cities or villages.

Front-end- Wishal M Sri Rangan

Design Methodology

The web application was designed following the atomic design pattern approach. The main reason for using this design methodology when organizing a React application is to isolate the environments of each feature component. When isolated, code becomes a lot more readable and modular. A single instance of a feature will make testing more straightforward, thus improving the overall quality of the dashboard.[1] **See Figure 5.0.**

The design pattern approach elements are as follows along with their implementation in this web application:

- Atoms: Consists of basic elements. In this web application this includes buttons, icons, input fields, scroll, text and headers.
- Molecules: Grouping atoms to build functionality. This includes pie charts, bar charts, navigation tabs and dumping site view tabs.
- Organisms: Combining molecules to form an organism that makes up a section of the website. In this instance, the home, messages, navigation, profile and statistics component are part of organisms.
- Templates: Combining organisms to form a page. In this project this would be the authentication and client page.
- Pages: An ecosystem that views different template renders. In this application this is the *App.jsx* file.

Moreover, the dashboard was designed to incorporate several React libraries that help with the visualization and ease of use of the application. **The libraries and frameworks used are listed as follows**:

[1] https://medium.com/@janelle.wg/atomic-design-pattern-how-to-structure-your-react-application-2bb4d9ca5f97

1. Formik[2]: A form library that handles the tracking of values, errors and visited fields, aiding validation and taking care of form submission processes. This library will be used to help build the authentication page and the employee administration section.
2. Mapbox GL JS[3]: A client-side Javascript library that renders 2D Mapbox maps as dynamic visual graphics with OpenGL in any compatible web browser, without using additional plugins. It comes with several user interface elements notably markers which will primarily be used in visualizing dumping locations on the interactive maps. Moreover, there are optional controllers that were implemented on the map such as zoom and scale controls.
3. Flux: React Flux was needed to make use of the dispatcher which helped keep track of navigation tabs.
4. React-Chartjs-2[4]: A Javascript charting that was used to implement bar charts and pie charts to visualize data received from the backend. Chart.js makes use of interactive charts that make data visualization very convenient.
5. React Select[5]: A select input control for React.js with a multi select feature that aids in the implementation of the map's filtering feature.
6. Axios[6]: A promise based HTTP client for the browser and node.js
7. Fortawesome: A font and icon framework used to implement the icons on the dashboard.

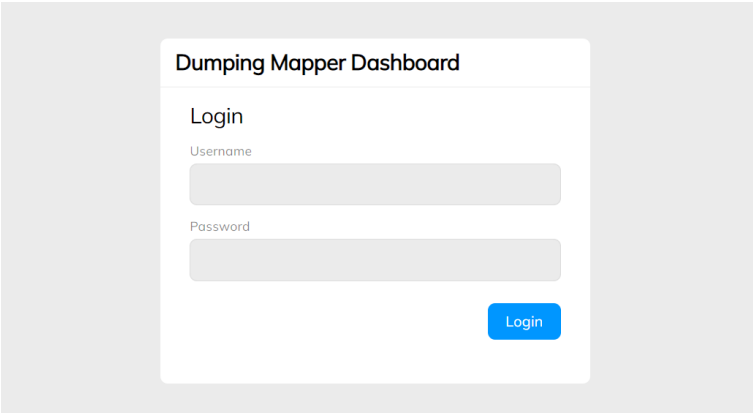
As the atomic design pattern approach was used to design the dashboard, the application was split into two separate pages:

1. Authentication: This consists of the sign up and login process for the employees of the municipality of each of Cyprus' districts.
2. Dashboard: The main page visualizing the data received from the back-end. The navigation will consist of the home, messages, statistics and profile section. In addition, there will be a night mode and logout functionality.

Authentication

The authentication page will handle the login process for the district employees of the dashboard. After discussion, it was decided that it is best that only one account per district existed and there is no registration process to create new accounts. In order to simplify validation and handling submission of input values, the form library Formik is used. This will also speed up the testing process for the authentication page further in development. The authentication process is also handled by axios making the necessary POST request with a user's username and password in order to access the dashboard as well as obtaining an access code, district identifier as well as the district name which will be needed in order to get specific information about districts or handle employee management. **See Figure 5.1**.

[2] https://formik.org/
[3] https://docs.mapbox.com/mapbox-gl-js/api/
[4] https://react-chartjs-2.js.org/
[5] https://react-select.com
[6] https://github.com/axios/axios



Client

The client page will consist of 4 sections:

Home

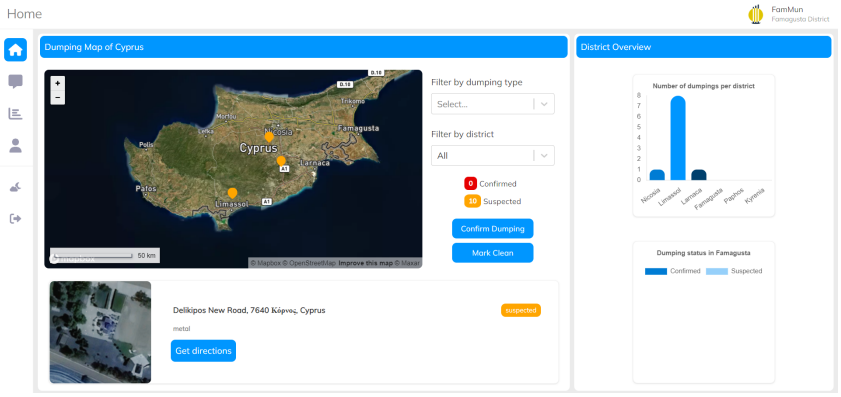
The home page consists of two subsections, the map display as well as the basic statistics display.

The map display includes the interactive map that has been implemented using Mapbox GL JS. As was aforementioned, the interactive map will be displaying markers where users will be able to spot confirmed and suspected dumping locations in the form of red (confirmed) and orange (suspected) markers. An alternative solution for the map display would have been the Google Maps API but it is currently under a paywall and therefore MapBox was chosen to be the free alternative. Furthermore, functionalities to report and clean each of these dumping spot markers has been implemented. This was done by making use of the axios library using GET and POST HTTP requests to the backend. When a marker is selected, a satellite image of the selected location will be displayed along with relevant dumping details such as the address, dump type and its confirmed or suspected status. With the use of the React Select library, users can filter dumping based on two parameters: district name and dumping type.

The basic statistics subsection includes a bar chart that shows the total number of dumping occurrences in each district of Cyprus. Moreover, this section includes a pie chart that shows the number of confirmed and suspected cases of dumping in the district from where a user is logged in.

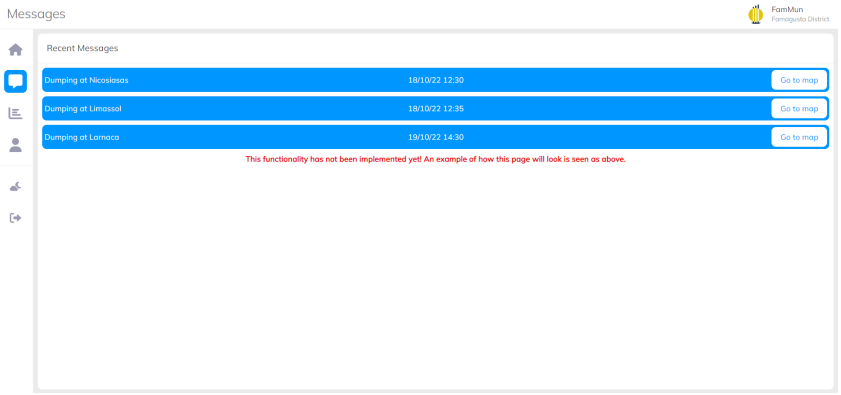
Finally, a pie chart of the percentages of detected, reported and cleaned dumping locations and a bar graph of the number of new dumping per week in the district that the employee works in will be displayed. The library that will be used for this is Chart.js as I am highly familiar with how the library functions as well as its simple charting functionalities. I have used the react-select library to create a dropdown menu with possibilities to display markers on the map based on the type of dumping detected as well as Cyprus districts. The algorithm for the filtering process using React Select can be found in **Figure 5.2**.

As mentioned previously, when a marker is selected, a site view card is displayed showing all relevant information about a dumping including an image. This image is retrieved from the Cloudinary storage where all dumping images are stored.



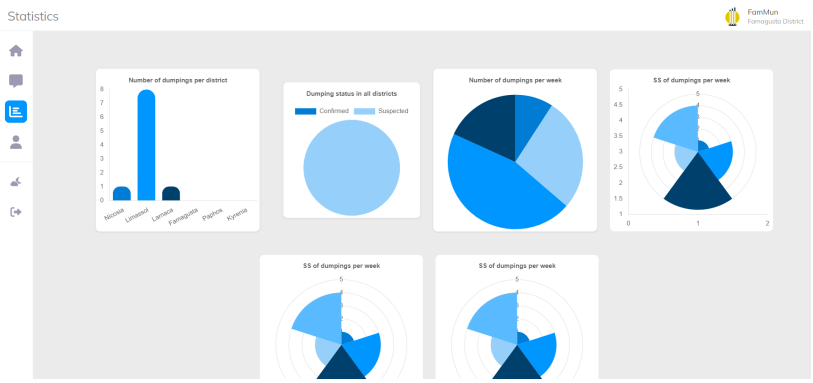
Messages

This section was designed to display incoming alerts of new dumping sites as well as messages regarding the status of these dumping sites. Moreover, an additional functionality this page will include is the ability to show the dumping on the map when the button “Show on Map” is clicked. The functionality of this section of the dashboard unfortunately can not be implemented as the current data received from the models and back-end do not include any information on time. Therefore this will be kept as a future possible functionality to add to the dashboard.



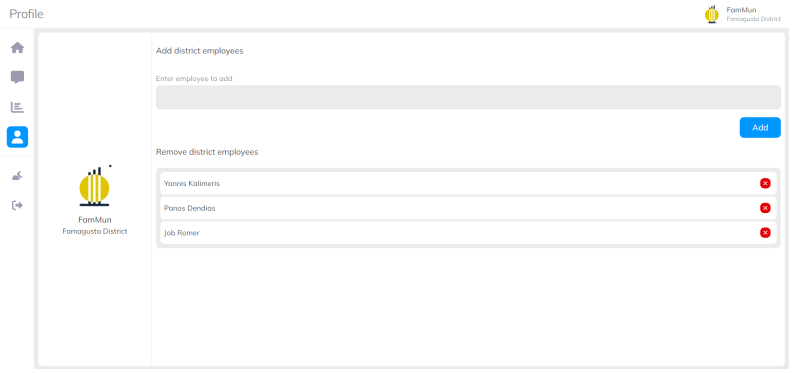
Statistics

This section will be displaying all possible metrics of dumping data. This would include molecules such as pie charts, bar charts and polar charts. Similar to the problem stated in the previous section, the readiness of the statistics page is lacking due to the missing time information on dumpings and therefore the full implementation could unfortunately not be completed.



Profile

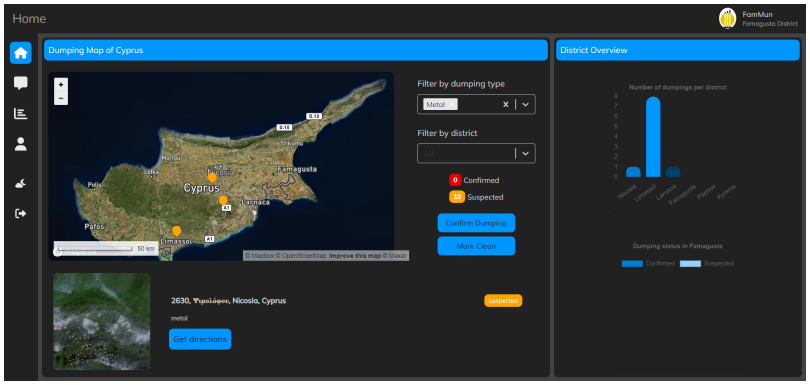
The profile section will include basic district information such as the username as well as the associated district the user has authority over. Additionally, the profile section has a functionality to add and remove district workers from the database. This was implemented using the help of Formik to handle the form submission and validation as well as axios to make HTTP requests to the backend of the application in order to add or remove workers.



Miscellaneous

Dark mode

Dark mode was implemented in order to provide users with options for their preferred theme of the dashboard.



Logout

The logout functionality was implemented making use of local storage. When a user logs in from the authentication page, they are provided with an access token which decides whether a user is allowed access to the Client page of the app. This access token is stored in the local storage. When logging out, the local storage is cleared, switching the viewport to the Authentication page.

Difficulties

As mentioned in the section above, the Messages and Statistics page could unfortunately not be implemented completely due to the lack of time information. The filtering options for different dumping types also are not fully functional as a bug occurs when a specific type of dumping is selected to filter the map. Furthermore, the Cascading Style Sheets of the dashboard is currently not optimized for a mobile application or a mobile viewport as more time had to be devoted into the connection with the backend and handling data received from it.

Future Work

In regards to future work, the main priority would be to fix the existing bugs and unoptimized features of the dashboard as well as having the full functionality of the messages and statistics page to be complete so that the dashboard is ready to use for an end consumer/user.

Appendix A

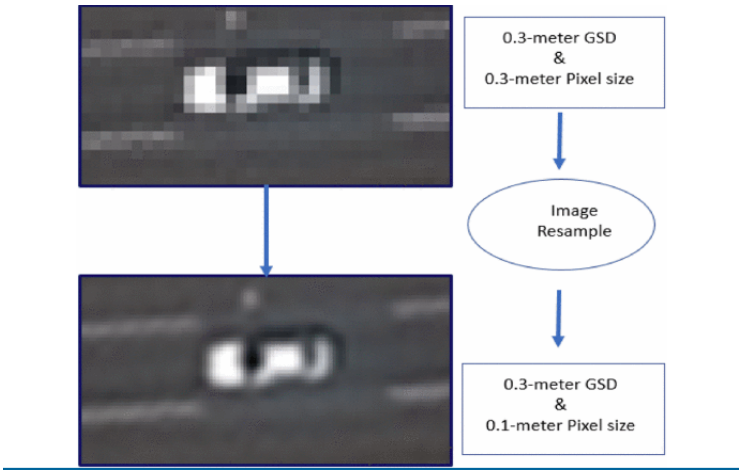


Figure 2.0 - Ground Sample Distance

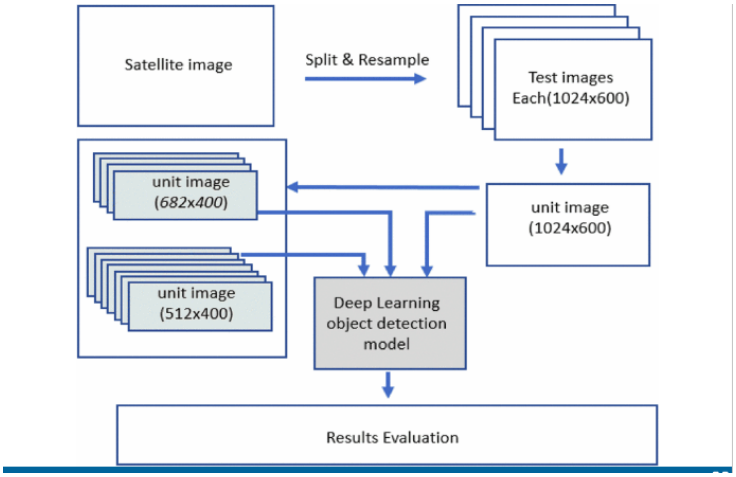


Figure 2.1 - Faster R-CNN split

Figure 2.2 - Image cropping

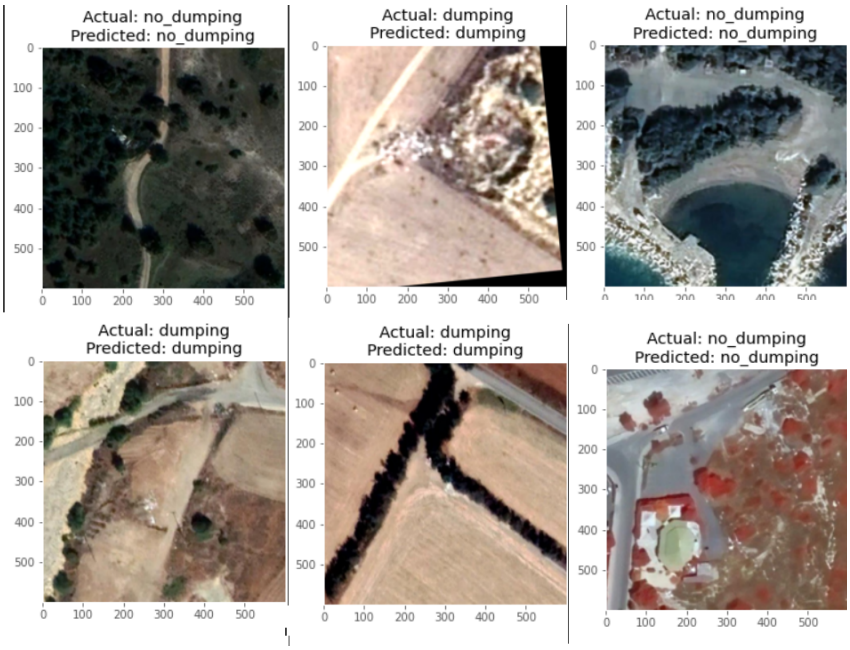


Figure 2.3 - Predictions

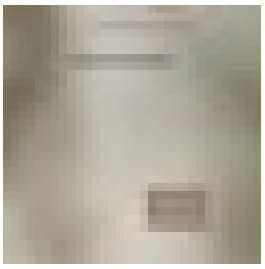


Figure 3.0 - 32 pixels by 32 pixels image.

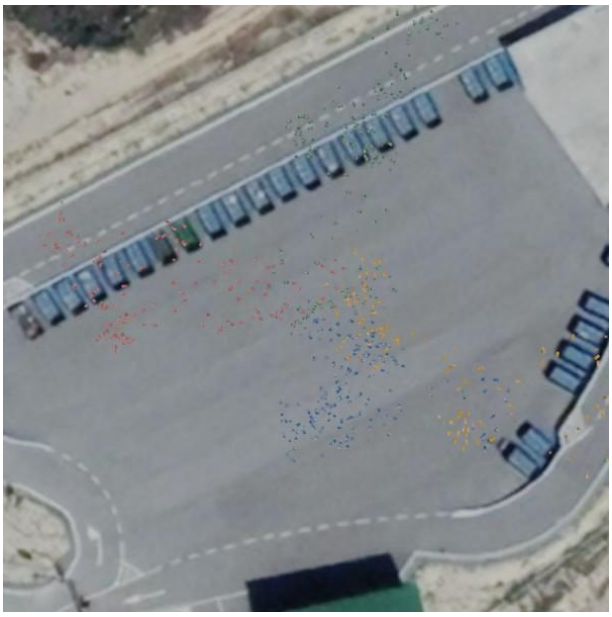


Figure 3.1 - 600 pixels by 600 pixels image.

```
{'cardboard': 0, 'metal': 1, 'plastic': 2, 'wood': 3}
[[0.21260688 0.29554123 0.27224338 0.21960849]
```

Fig 3.2 - Output of the softmax layers.

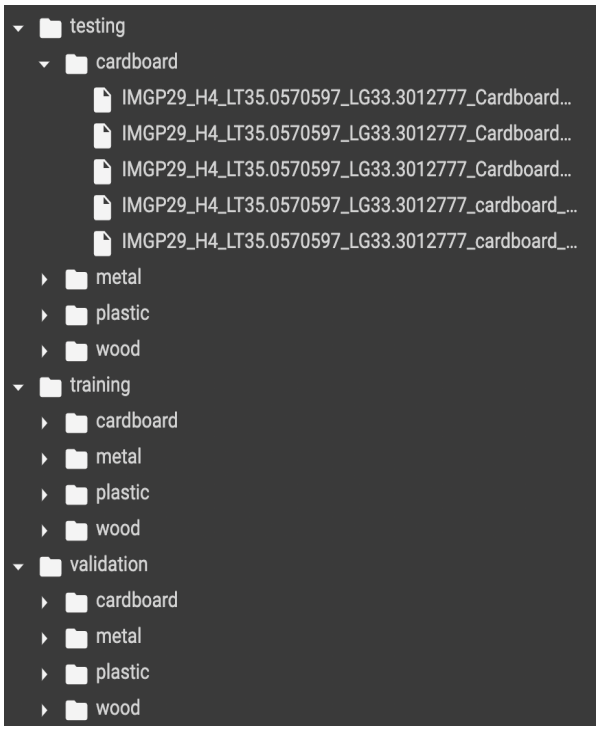


Figure 3.3 - Data Directory structure.

Figure 4.0

```
router.post("/workers", async (req, res) => {
  const { username } = req.body;
  const muni = await municipalities.findOne({ username: username });
  if (muni) {
    return res.status(200).send({
      Respworkers: muni.Respworkers,
    });
  } else {
    res.json("I cant log");
  }
});
```

Figure 4.1

```
router.delete("/delworker", async (req, res) => {
  const { username, workerName } = req.body;
  const muni = await municipalities.findOne({ username: username });

  if (muni) {
    muni.Respworkers.pull(workerName);
    await muni.save();
    res.redirect(200,'/workers/workers')
  } else {
    return res.status(400).send({
      message: "Delete Failed",
    });
  }
});

router.post("/addworker", async (req, res) => {
  const { username, workerName } = req.body;
  const muni = await municipalities.findOne({ username: username });
  const check = await municipalities.findOne({Respworkers:workerName

  if (muni && !check) {
    muni.Respworkers.push(workerName);
    await muni.save();
    return res.redirect(200,"/workers/workers");
  }else if(check) { // check if worker already exists
    return res.status(401).send({
      message: "This worker already exists"
    });
  } else {
    return res.status(400).send({
      message: "No collection for this district"
    });
  }
});
```

Figure 4.2

Figure 4.3

```
router.post("/confirmdump", async (req, res) => {
  const { long, lat } = req.body;
  const update = { status: "confirmed" };
  const dump = await Dumps.findOneAndUpdate({ long: long, lat: lat }, update);
  if (dump) {
    res.redirect(200, "Done");
  } else {
    res.redirect(500, "Status Change Failed");
  }
});
router.post("/suspectdump", async (req, res) => {
  const { long, lat } = req.body;
  const update = { status: "suspected" };
  const dump = await Dumps.findOneAndUpdate({ long: long, lat: lat }, update);
  if (dump) {
    res.redirect(200, "Done");
  } else {
    res.redirect(500, "Status Change Failed");
  }
});
```

```
router.get("/getdumps", async(req, res) => {
  const dump = await Dumps.find();
  if (dump) {
    return res.send(dump);
  } else {
    res.json("I cant log");
  }
});
router.delete("/deldumps", async (req, res) => {
  const { long, lat,imageurl } = req.body;
  const dump = await Dumps.findOneAndDelete({ long: long, lat: lat });
  cloudinary.v2.uploader.destroy(imageurl);
    if (dump) {
    res.redirect(200, "/getdumps");
  } else {
    res.json("Delete failed");
  }
});
```

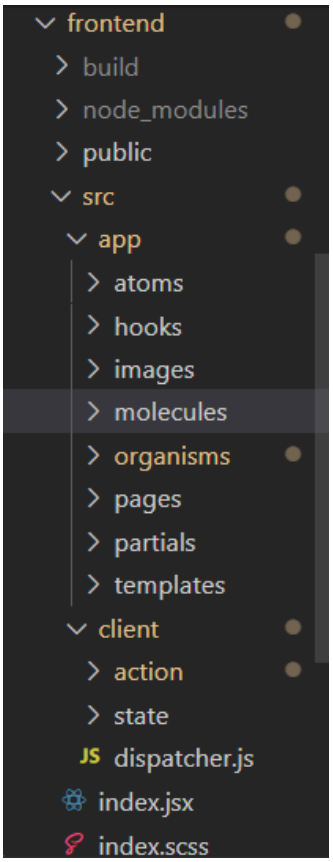


Figure 5.0 - Frontend directory structure following the principles of the Atomic Design Pattern.

Key	Value
mapbox.eventData.uuid:d21zcmlyYW5nYW4=	[redacted]
district_name	Famagusta District
district_id	Famagusta
theme	dark-mode
dumping_mapper_user_id	FamMun
dumpingmapper_access_token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtdW5pY2lwYWxpdGlI...

Figure 5.1 - Example of the data obtained from a successful authentication.

```
filteredDumpList = dumpList.flatMap(
    (dump) => {
        if(selectedDumpType){
            if(selectedDistrict.value !== 'All'
                && dump.districtId === selectedDistrict.value
                && selectedDumpTypeValues.includes(dump.type[0])){
                return [dump];
            }
            else if(selectedDistrict.value === 'All'
                    && selectedDumpTypeValues.includes(dump.type[0])){
                return [dump];
            } else{
                return [];
            }
        } else {
            if(selectedDistrict.value !== 'All'
                && dump.districtId === selectedDistrict.value){
                return [dump];
            }
            else if(selectedDistrict.value === 'All'){
                return [dump];
            } else{
                return [];
            }
        }
```

Figure 5.2 - District and dumping type filtering algorithm.

Appendix B - Annotation Distribution

First Batch

Responsible Person	**Assigned Locations to Annotate**	**Assigned Locations to Verify**	**Annotation Progress**
Wishal	1-40	41-64, 161-181	**Completed**
Tashfeen	41-80	81-104, 182-202	**Completed**
Illya	81-120	121-144, 203-223	**Completed**
Ege	121-160	1-24, 224-247	**Completed**
Job	161-249	25-40, 65-80, 105-120, 145-160	**Completed**

Second Batch

Responsible Person	**Assigned Locations to Annotate**	**Assigned Locations to Verify**	**Annotation Progress**
Wishal	327-352	249-259, 463-499	**Not started**
Tashfeen	301-326	327-337, 426-462	**Started, not finished**

Illya	249-274	275-285, 353-389	**Not started**
Ege	275-300	301-311, 390-425	**Completed**
Job	353-499	260-274, 286-300, 312-326, 338-352	**Not started**

References

1. https://medium.com/@janelle.wg/atomic-design-pattern-how-to-structure-your-react-application-2bb4d9ca5f97
2. https://formik.org/
3. https://docs.mapbox.com/mapbox-gl-js/api/
4. https://react-chartjs-2.js.org/
5. https://react-select.com
6. https://github.com/axios/axios