Region Of Interest Enlargement for Effective Resolution Enhancement of Object Detection Models for Flood Risk Assessment through Street View Imagery

BSc Thesis

Adrian Kepguep (5324645)
Philip N. Engelenberg (5003040)
Yannick Serrien (5235790)

July 11, 2023

Abstract

This thesis presents a comprehensive analysis and implementation of a program designed to detect the number of steps leading up to the front door of a house using a Google Street View Image. The purpose of counting the steps is to have a proxy of the ground floor elevation by multiplying with an average step height. After determining the street level's elevation from sea level, it becomes possible to assess the vulnerability of houses to flooding. Additionally, the system developed in this thesis provides a method for enhancing the effective resolution of computer vision models to be able to detect details with more accuracy, this is done through Region of Interest detection and enlargement.

The design process consists of four main stages: Acquiring and labelling data consisting of street view images which contain houses, staircases, and steps. Developing an initial model to gauge the ability of the current technology available. Developing algorithms to detect objects of interest in the images. Use a top-level to combine these algorithms and crop out any information that is not of interest.

In the structure of the final product, a top-level, that allowed a model to select a region of interest for step-detection and then cropped the image to this region of interest, was used. This approach allowed for an effective enhancement in resolution as the model is allowed to only focus on the 'useful' information. The detection in these models was done by YOLOv8x-algorithm that was transfer learned on the custom dataset.

The final product had a precision of 98.7% in detecting steps, an area under the curve (AUC) of 98% in the PR-curve for steps and a deviation no larger than 1 step.

Preface

The authors of this report would like to thank Justin Dauwels, Kasmalkar Indraneel Gireendra, Dennis Wagenaar, and Martin Schumacher for their support. Lastly, the authors would like to show their appreciation to the management team of the BAP project and the entire BAP team for making this challenging but wonderful project possible.

Group E.1 Team members
Delft, June 2023

Contents

Chapter 1

Introduction

This chapter presents an explanation of the problem at hand and a state-of-the-art analysis of existing solutions for this problem. It outlines the project's scope, objectives, and the needs it addresses. The chapter also discusses the limitations, factors for alternative approaches, and measurable quantities for performance monitoring. Lastly, a brief thesis synopsis is provided.

1.1 Problem Definition

According to modern estimates, floods around the world account for up to 40% of all economic losses caused by all-natural disasters [1]. Between the years 2000 and 2020, 44.9% of the total floods around the world occurred in South and East Asia and the North-Eastern part of Africa. Unfortunately, the majority of the nations in this so-called 'Belt and Road'-region are developing nations, meaning that they lack material reserves and emergency relief capacities for disaster response and have limited disaster resilience. As a result, when a calamity strikes, the countries in this region often sustain great losses.

Disaster losses in the 'Belt and Road'-region are more than twice as high as the global average, according to the Emergency Disaster Database [2]. The fatality rate of catastrophe victims in this region is, likewise, significantly greater than the global average, and in some countries in South and Southeast Asia, it is even ten times higher [3]. Worse still, flooding in this region is predicted to become more frequent and intense in the future as a result of global climate change [4].

Due to the limited resources that the countries in the 'Belt and Road'-region have, a system needs to be put into place that allows these countries to optimally manage their resources in order to minimize the amount of damage caused by floods. A flood susceptibility map that is based on street level elevation is a step in the right direction and the data for this is available to the project proposers Kasmalkar Indraneel Gireendra and Dennis Wagenaar. A more accurate flood susceptibility map would show the elevation at which economic or life-threatening risk rises, which is at the house's ground-floor level.

The problem at hand is, thus, to have a system for determining the height of a house's ground floor. To address this problem, this thesis proposes an inexpensive automated system capable of detecting the steps in front of houses to estimate ground floor elevation by multiplying with an average step-height. The steps will be detected from Google Street View (GSV) imagery as this is widely available in the 'Belt and Road'-region [5].

Currently, the best-known automated approach for determining ground floor elevation also uses GSV imagery, but it relies on bounding boxes of detected houses in the image and camera-house distance calculations [6, 7]. Camera-house distance measurements are often inaccurate and don't work well when a house is at an angle to the camera, in this case the geometry of the bounding box will change compared to the shape of the house and this causes more inaccuracy in the estimated elevation.

This thesis presents an alternative solution, it seeks to train computer-vision models that can identify and quantify the number of steps, this can then be multiplied by an average step height to get an approximate ground-floor elevation. By implementing this solution, it will be possible to inexpensively streamline the improvement of flood susceptibility maps and enable more effective flood vulnerability assessments.

The successful development of an automated step-counting system will not only save time and effort but also provide valuable insights into house ground floor elevations and potential flood vulnerabilities. It will contribute to the field of computer vision and assist in the development of efficient disaster management strategies.

1.2 State-Of-The-Art Analysis

This chapter will discuss machine learning (ML) and computer vision, a subbranch of machine learning specifically made for identifying objects in images and videos. Transfer learning and Convolution Neural Networks (CNN) will also be explained.

1.2.1 Machine Learning

Machine learning (ML) is a field that focuses on the development of algorithms and models that enable computers to learn from data and make predictions or decisions without explicit programming. ML-algorithms utilize statistical techniques to identify patterns and relationships in large data sets, continuously adjusting their internal parameters to fit to the training data and to improve performance.

The training of ML algorithms can be categorized into four major types: supervised, unsupervised, semi-supervised, and reinforcement learning [8, 9].

- Supervised learning is a type of ML where algorithms learn from labelled examples to make predictions or classify new, unseen data.
- Unsupervised learning involves finding patterns and structures in unlabeled data.
- Semi-supervised learning data are not stored in a relational database like the structured data mentioned above, but it does have certain organizational properties that make them easier to analyze.
- Reinforcement learning enables algorithms to learn optimal behaviours or strategies through interaction with an environment.

In this project, supervised learning will be used together with a labelled dataset, more about this in chapter 3. Supervised learning is used as the model should behave as the labelled input data and, thus, the objects of interest are labelled and fed to the model in training.

1.2.2 Computer Vision

Computer vision, an area within artificial intelligence and computer science, and thus an area within machine learning, aims to enable computers to understand and interpret visual information from digital images or videos. It utilizes algorithms and models to extract meaningful information from visual data, emulating and enhancing human vision capabilities. More applications of machine learning in computer vision are discussed in [9].

In the field of machine learning, various models and frameworks serve different purposes. Application Programming Interfaces (APIs) such as TensorFlow [10], PyTorch, and scikit-learn provide pre-built functions and tools for developing machine learning models.

YOLO (You Only Look Once) is an object detection algorithm known for real-time detection and localization of objects within images or video frames [11, 12]. It accomplishes this by dividing the input image into a grid and predicting bounding boxes, class probabilities, and confidence scores. TensorFlow Lite (tf_lite) is a lightweight version of TensorFlow designed for resource-constrained devices, allowing optimized performance and reduced memory footprint when deploying machine learning models [13, 14].

1.2.3 Transfer Learning

Training a model from scratch requires vast amounts of time and some very strong computational power, [15] mentions the heavy burden of training a deep learning model - which these models are. On the other hand, transfer-learning a pre-trained model requires much less time and computations [15].

Transfer learning aims at improving the performance of target learners on target domains by transferring the knowledge contained in different but related source domains [16]. It emerges as a promising approach, leveraging existing models trained on related tasks and adapting them to new data sets.

The dataset that was worked with, described in chapter 3, was not very large. The computational resources, that were available (a Google Colab server), were limited and time was too. These limitations and the size of the dataset led to the decision to use a pre-trained model and transfer learn this. More about this process will be in chapter 5.

1.2.4 Convolution Neural Networks

Convolutional Neural Networks (CNNs) are deep learning models that are organized in interconnected layers, with convolutional, pooling, and fully connected layers and these leverage convolutional layers to automatically learn hierarchical representations from grid-like data, such as images [17]. Convolutional layers perform convolutional operations, these are operations where each layer multiplies the elements of an array of features (kernel, usually 3x3 or 5x5) and an input of array numbers (tensor). For each kernel in a convolutional, it can detect a local feature and give a feature map [18]. The convolutional layer is fed into a pooling layer that reduces the dimensions of the input layers and lowers the number of parameters in a layer, it merges features that are alike. Fully connected layers are used to connect every local input from a previous layer [18].

Convolutional Neural Networks are designed to learn simpler patterns at low depths and more complicated patterns with deeper layers. Because of this, they excel in capturing intricate connections and features within data, particularly in tasks involving image analysis and object recognition [18].

While CNNs are not universally required for all machine learning applications, they are widely preferred due to their ability to enhance the power and effectiveness of machine learning algorithms. They use multiple hidden layers, uniform weights across multiple nodes in layers (weight-sharing), and each node only gets an input from nodes that are nearby so that each output is only related to certain parts of the input (local connectivity). Doing this avoids very large parameters [18].

CNNs significantly contribute to the strength of machine learning models, making them more powerful and enabling more accurate identification of objects. They have been highly effective in computer vision tasks, including image recognition and object detection, driving advancements in the field.

1.3 Thesis Synopsis

This thesis addresses the problem of automated step counting in front of houses to enhance flood vulnerability assessments and will, in doing so, present a technique that will allow computer vision models to work with a higher effective resolution. The absence of existing automated solutions and specific data sets for step detection within staircases creates a research opportunity.

The research employs transfer learning techniques to develop a specialized model capable of accurately counting the steps. Leveraging available data sets for houses, doors, and staircases, the project aims to train models which detect and quantify the number of steps in front of houses. This automated system streamlines the step-counting process and improves accuracy, contributing to more effective flood vulnerability assessments.

The thesis begins with a state-of-the-art analysis, evaluating existing methodologies and exploring transfer learning approaches. Building upon this analysis, the research formulates a problem description, highlighting the significance and implications of automated step counting. Afterwards, the employed dataset is discussed, this is then followed by an explanation of the metrics that were used to evaluate model performance. Then the design process is presented with finally an evaluation of the system's performance and the conclusions drawn from this thesis along with suggestions for further work.

In addressing the identified problem, the thesis contributes to the field of computer vision, offering a practical solution to detect detailed objects using lower resolution models. The research findings and implementation of an automated step-counting system provide valuable insights into house ground floor elevations and enhance disaster management strategies.

Chapter 2

Program Of Requirements

This chapter focuses on the requirements of the system developed for the automation of step counting in front of houses. The requirements are categorized into two main types: mandatory requirements and trade-off requirements. These requirements serve as the foundation for designing and developing an effective and efficient solution.

2.1 Mandatory Requirements (MR)

The mandatory requirements delineate the fundamental functionalities and attributes that must be integrated into the system for it to be able to fulfill its intended purpose. They are non-negotiable and serve as the foundation for the system's functionality.

The post-doctorates Kasmalkar Indraneel Gireendra and Dennis Wagenaar proposed this project along with their conceptualisation, constraints and future application of the final product of the project. They will be utilising the final product after the project is finished and, as such, their specifications are seen as the mandatory requirements for this project.

The program, that was requested by the aforementioned post-doctorates, must be able to be fed an image and then take no more than one minute to analyse this image. The program must then output the number of steps leading up to the front door of the most centered house in the picture. If the main house has multiple doors the step count corresponding to the lowest staircase must be given. The final product can deviate, at most, one step of from the correct answer. Additionally, the final product should have a precision (True Positive Rate) of 90%. This True Positive Rate (TPR, see equation 4.2) of 90% means that out of a hundred detected steps, ninety must be correct

The final product will be deployed by the post-doctorates so the program must be documented with a focus on an effortless transfer of usage to future communities expanding upon the final product and future users of the program.

The manufacturing requirements mandate that the program must be coded with the open-source coding language Python and only open-source functions must be used or new functions written during the project.

2.2 Trade-Off Requirements (TOR)

In addition to the mandatory requirements, trade-off requirements are considered in the system design process. Trade-off requirements encompass additional functionalities, features, or qualities that enhance the overall performance, usability, and user experience of the system. These requirements are not mandatory for the basic functionality of the system but add value and improve its effectiveness.

A few trade-off requirements can be noted, These were mainly based on improvements over the mandatory requirements. The first of which is processing individual images significantly faster than the required one minute.

In addition, the program could show an in-image result of the number of steps it found if so desired by the user. If the average height of a step in the area of the user is given the program can give the ground floor elevation in a metric height measurement instead of a proxy measurement as the number of steps. These trade-off requirements target to make the experience of using the program more applicable.

Finally, the system aims to find as many of the steps from the picture as possible. To quantify this, the recall is used. Recall (see equation 4.3) is the fraction of correctly detected steps over the total number of steps. The aim is to have a recall of at least 90% so that it can be expected that only in large staircases a step will sometimes be missed and that the amount of steps missed in smaller staircases is insignificant.

Chapter 3

Data Set

Training a machine learning algorithm requires training data for it to learn from and fit its parameters to (section 1.2.1). A validation set is then used to make sure the model doesn't overfit in training ([19]) and a test-set is then used to evaluate the algorithm's final performance. This chapter describes the data set that was used in the creation and evaluation of the program that is the subject of this thesis.

3.1 Initial Data Set

The post-doctorates that requested this program to be created provided a data set of 1029 Google Street View images depicting front views of houses. The pictures from various regions around the world. These regions were the following:

- Aizuwakamatsu, Japan
- Amsterdam, Netherlands
- Detroit, USA
- Miami, USA
- Padang, Indonesia
- San Francisco, USA
- Santa Maria, Brazil
- Venice, Italy
- Yogyakarta, Indonesia

As these images are from different regions around the world, the different styles of houses are well included in the data set. The images in this data set were labelled with rectangular bounding boxes for three classes. These classes were house, door, and step.

Of the 1029 pictures in this initial data set 829 were used for the training set of the initial model (see section 5.1), 100 for the validation set and 100 for the test set. This roughly follows an $80-10-10$ data split which is described in [20].

3.2 Final Data Set

When analyzing the initial data, it was found that certain images contained mislabelling or no labelling for objects of interest. This would mean that a house, a door or a step may not be labelled or labelled as a different object. In addition, (seemingly) identical objects of interest were not all labelled within a single image or across multiple images. This label noise is detrimental to model training as it essentially confuses the model ([21]). To resolve the label noise, the incorrect and incomplete labelling was corrected manually. This was done using the labelling tools provided by Roboflow. The test metrics that have been displayed in chapter 5 and 6 were made with test data from which label noise was removed to ensure the accuracy of these results.

Additionally, new data was added to the data set. This new data consisted of images of houses, staircases with steps, and doors. The entire data set was also labelled for staircases (where a staircase could consist of only one step) and the training and validation set was then enlarged from 929 to $3.3k$. This larger data set was used to train the final model and was again split as $8:1$ for training-validation, just like it was for the initial data set.

The images in the test set was not changed after creating the final data set, unlike the training and validation sets, as this split was agreed upon with another subgroup and it would allow for one-to-one comparison of the performances of the models.

Chapter 4

Employed Metrics

The machine learning models, developed in this thesis, were trained to be competent in identifying certain objects and accurately defining its position with a bounding box. These two parts can be classified as a multi-class case (classifying instances into three or more classes) and an object localization case (finding the location of instances in an image denoted with a bounding box).

Useful performance metrics for multi-class case labelling are confusion matrices and precision-recall curves [22, 23]. Effective performance metrics for object localization case labelling are the intersection over union (IoU), mean square error (MSE), root MSE (RMSE), and normalized RMSE (NRMSE) ([24]). A performance metric which encompasses both cases is the mean average precision (mAP) [23]. The code used to compute all these metrics can be found in appendix D.5.

4.1 Multi-Class Labelling Performance

4.1.1 Confusion Matrices

Confusion matrices are primarily used in binary classification to visualise whether the model did or did not classify an object and to show in both cases if this was correct (see figure A.1). Multi-class cases can be assessed using a binary classification by judging the classes individually.

A confusion matrix categorizes the prediction results into four groups: true positives, false positives, true negatives and false negatives. For example, the model can label an object as 'house' (which is a positive), if this is correct the prediction is a true positive. However, if the object is not a house but a gas station, then the model is wrong and the prediction is a false positive.

For the same class 'house' the model's prediction can be to not label the object (a negative). If the object is actually a house the model is wrong for not labelling it a house, this is a false negative. If the model was correct to not label the object a house, the prediction is a true negative.

For image recognition problems a true negative would mean that an object that does not exist is not detected. As this could be said for every point in the picture that does not contain anything, this number could be extremely large and it is not fully possible to define all locations where an object is not present. As can be seen in the confusion matrices of appendix A, true negatives are set to 0. The ideal case is for all labels to be found correctly which translates to 100% true positives and 0% for all other entries.

A Receiver Operating Characteristic-curve (ROC-curve) is often used to display the behaviour of the confusion matrices across various confidence levels. It could not be used in this case as the ROC-curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR). The latter is calculated as shown in equation 4.1:

$$FPR = \frac{False\ Positives}{False\ Positives + True\ Negatives} \tag{4.1}$$

As mentioned before, the true negatives are set as 0 which leads to the FPR being reduced to $FPR = \frac{FP}{FP}$ which is always equal to 1. This would result in a vertical line for an ROC-curve at $FPR = 1$. This would not give a clear representation of model performance.

4.1.2 Precision and Recall (P&R)

Precision and recall quantify how many correct predictions have been made. The 'precision' of the model represents how often a prediction is correct when the model labels an object. This translates to the fraction of all correct detections over the number of total detections made (see equation 4.2). The precision is often also referred to as the True Positive Rate (TPR). The 'recall' measures how often the model labelled an object of interest if an object of interest is present in the image. This translates to the fraction of correct guesses over the total number of annotated labels present in the images which the model could have potentially found (see equation 4.3).

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{4.2}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{4.3}$$

The precision and recall can be graphed unto a curve called the precision-recall (PR) curve. This PR-curve is created by computing the precision and recall for several confidence threshold values. For each prediction, the model expresses how confident it is in this prediction. Introducing a confidence threshold means ignoring all the predictions made with a confidence lower than the threshold. Increasing the confidence threshold leads to the removal of low-certainty predictions, which tend to be false positives, as derived from [25], where it is mentioned that confidence thresholds are used to filter out false positives. To be able to judge what threshold value is most optimal the precision and recall can be computed for several threshold values.

For the precision-recall curve, one would aim for the $(1, 1)$-coordinate which corresponds to 100% precision and 100% recall and thus a large area under the PR-curve. This Area Under the Curve is referred to as the AUC. A large AUC signals a model with great precision and recall, the ideal case is an AUC of 1.

4.2 Object localization performance

4.2.1 Intersection over Union (IoU)

The Intersection over Union (IoU) metric, is a metric that gives a measure of how well the bounding box of the detection label of the model overlaps with the annotated label. The IoU is defined as the fraction of the area of overlap between the detection label and the correct label, over the union of the area. Figure 4.1 is a visualisation of this.

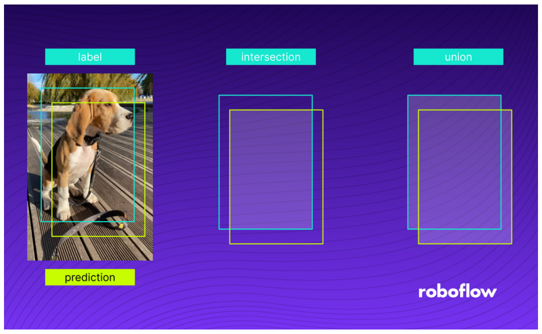


Figure 4.1: Visualisation of the intersection over union (IoU) of the prediction label and the correct label. Image taken from [26].

The IoU can be described mathematically, suppose the detected label has the coordinates $x_{1,det}$, $x_{2,det}$, $y_{1,det}$ and $y_{2,det}$ for the coordinates of the top-left and bottom-right corner and the correct label has the coordinates $x_{1,cor}$, $x_{2,cor}$, $y_{1,cor}$ and $y_{2,cor}$ these eight coordinates describe the positions of the top left and bottom right corners of two rectangles. From these coordinates the rectangle describing the intersection can be found with equations 4.4, 4.5, 4.6, 4.7.

$$x_{1,intersection} = max(x_{1,det}, x_{1,cor}) \tag{4.4}$$

$$x_{2,intersection} = min(x_{2,det}, x_{2,cor}) \tag{4.5}$$

$$y_{1,intersection} = max(y_{1,det}, y_{1,cor}) \tag{4.6}$$

$$y_{2,intersection} = min(y_{2,det}, y_{2,cor}) \tag{4.7}$$

The area of the intersection is the difference between $x_{2,intersection}$ and $x_{1,intersection}$ multiplied by the difference between $y_{2,intersection}$ and $y_{1,intersection}$. If the detection rectangle and the rectangle from the correct annotations do not overlap, one or both of these differences will be negative. In this case, the area of the intersection should be zero instead of a negative value. To ensure this is the case equation 4.8 is used to compute the area of intersection.

$$Area\ of\ intersection = max(0, x_{2,intersection} - x_{1,intersection}) \cdot max(0, y_{2,intersection} - y1_{intersection}) \tag{4.8}$$

The union is then computed as:

$$Area\ of\ union = Area_1 + Area_2 - Area\ of\ Intersection \tag{4.9}$$

So the IoU is calculated as the area of the intersection over the area of the union:

$$IoU = \frac{Area\ of\ intersection}{Area\ of\ union} \tag{4.10}$$

The IoU was used to remove poor detection results. All predictions made by the model must have an IoU higher than the IoU threshold, if this is not the case the detection is considered to be false. The optimal value for the IoU is debatable but for this project, it was set to 0.5 because this is a commonly chosen value ([23]).

4.2.2 MSE, RMSE, and NRMSE

The mean square error (MSE) illustrates how far away the bounding box of the prediction of the model is from the bounding box of the correctly annotated label. This distance is calculated between the two centres of the bounding boxes in a Pythagorean fashion. This distance is referred to as the error which will subsequently be squared. Finally, the average of all correct detection results will be determined by taking the mean value (see equation 4.11).

$$MSE = \frac{\sum(x_{centre,det} - x_{centre,cor})^2 + (y_{centre,det} - y_{centre,cor})^2}{number\ of\ correct\ detections} \tag{4.11}$$

The x and y coordinates of the centre of the detection and correction labels are found with the equations 4.12, 4.13, 4.14, and 4.15.

$$x_{centre,det} = x_{1,det} + \frac{x_{2,det} - x_{2,det}}{2} \tag{4.12}$$

$$y_{centre,det} = y_{1,det} + \frac{y_{2,det} - y_{1,det}}{2} \tag{4.13}$$

$$x_{centre,cor} = x_{1,cor} + \frac{x_{2,cor} - x_{1,cor}}{2} \tag{4.14}$$

$$y_{centre,cor} = y_{1,cor} + \frac{y_{2,cor} - y_{1,cor}}{2} \tag{4.15}$$

Derived from the MSE, is the root mean square error (RMSE) which is the square root of the mean square error. The RMSE is in the same units as the x and y coordinates. What is meant by this, is that the MSE gives a result in the unit of $[pixels^2]$, the RMSE is in $[pixels]$. The equation for this is shown in equation 4.16.

$$RMSE = \sqrt{MSE} \tag{4.16}$$

The normalized root mean square error (NRMSE) removes the unit from RMSE. This is done by first dividing the x_{centre} and y_{centre} coordinates by the total amount of horizontal and vertical pixels respectively. Ideally, the MSE, RMSE, and NRMSE should approach zero, indicating perfectly located bounding boxes.

4.3 Mean Average Precision (mAP)

The last performance metric mentioned is the mean average precision (mAP) which embodies both multi-class classification and object localization.

The mAP combines the precision-recall curve and the IoU thresholds. The mean average precision is calculated by finding the average precision for multiple IoU thresholds, followed by taking the mean value of these average precisions. First, one takes a detection class like 'house' and draws a number of precision-recall curves, each linked to a different IoU threshold value. For each precision-recall curve, the discrete area under the curve is taken. Which is calculated as the weighted mean of precision values at discrete recall intervals, with the discrete increase in recall used as the weight, similar to a definite integral. This is done for each precision-recall curve which corresponds with a specific IoU threshold. This discrete area under the curve is the average precision, and the mean of the average precision for all precision-recall curves is the mAP. As the discrete area under the curve (AUC) should ideally be 1 for a PR-curve (as mentioned in 4.1.2, the ideal mAP is 100%. A more extensive explanation of the calculation of the mAP can be read in [23].

Chapter 5

Design Process

This chapter describes the steps taken and decisions made to create an automated system to accurately count the number of steps leading up to the front door of a house. It explains how the project was tackled by developing an initial design to assess the capabilities of the current computer vision technology and see what has to be optimized to get a better performance. The chapter then goes on to describe how the initial design was then built and tested. Subsequently, it analyzes the performance of this initial model and the introduces a new technique based on the shortcomings of the initial model. This new technique requires a separate model to be generated for each object (house, door, staircase, and step) in the image, the chapter describes 3 types of models that are used for this and finally combines all using a top-level.

5.1 Initial Design

For the reasons mentioned in 1.2.3, the decision was made to transfer learn a well-documented, pre-trained model. This was done using the TensorFlow Object Detection API ([10]) as this allows for a plethora of models to be easily transfer learned. Using this API allows one to finetune several different models, the model that was chosen here was the 'SSD MobilieNet V2 FPNLite'. This model was chosen because of computational limits present at the time and because of its performance on the MS COCO data set ([27]) as shown in [14].

The 'SSD MobileNet V2 FPNLite'-algorithm uses a 320x320 input resolution, this requires the input image to be resized to this input resolution. This resizing involves the merging and averaging of pixel values.

The idea for this initial model is to feed the model an image from the initial test set, described in section 3.1. The model should then give the user annotations for all objects of interest (house, door, and step) that it detects. These annotations are given as five numbers:

1. Confidence score of prediction (a number between 0 and 1)
2. x-coordinate of top left corner of predicted bounding box (x_{min})
3. y-coordinate of top left corner of predicted bounding box (y_{min})
4. x-coordinate of bottom right corner of predicted bounding box (x_{max})
5. y-coordinate of bottom right corner of predicted bounding box (y_{max})

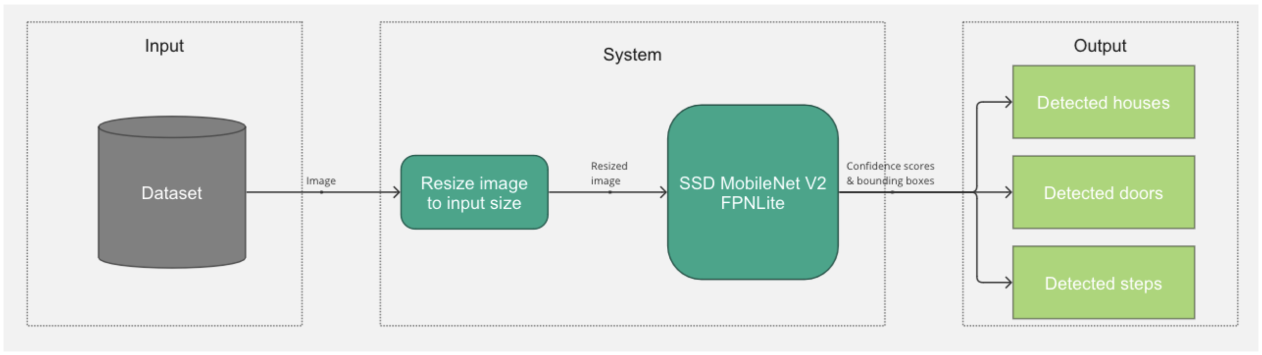


Figure 5.1: Schematic representing the data flow in the initial design

The behaviour observed from this initial model can then be used to determine what type of post- and prepossessing can increase the performance. A schematic of the functionality of the initial system is shown in figure 5.1.

5.1.1 Training

The first stage in the transfer learning process of a model using the TensorFlow Object Deterction API is to transform the given training data, which contained png-format images along with XML-format annotations, into TFRecords. TFRecords are a simple format for storing binary data and it is used by TensorFlow for training [28].

After creating the TFRecords, the training configuration had to be set up. In this configuration, the model type, base pipeline file and pre-trained checkpoint were specified. This pre-trained checkpoint is the point up to which the model was pre-trained and from which it will now be fine-tuned to be able to detect the labels specified in the Google Street View data set. The pipeline file is used to set the configuration of the training of the model. The path to the TFRecords is stored in this, as well as the path to the label map, which contains a list of the labels in the images, the classes, as well as the number of classes. The batch size of the training is specified here, the number of steps used in the training, and the learning rate is also specified here.

The batch size was set at 16 as this was the largest that did not exceed the limits of the GPU of the Google Colab server that was used for training. The batch size is the number of images used per training step, a batch size of 16 indicates that 16 images are used on every training step and that it takes 52 training steps for the model to use every image in the 829-image data set at least once. A large number of training steps was used for training, $100k$ steps. This was done as it was more than enough and, thus, always allowed for early termination of the training process when the loss on the validation set stopped decreasing but on the training set didn't as this would indicate that the model was overfitting on the training data and no longer increasing performance on the (out-of-sample) validation set [19].

The model was then trained and converted to TensorFlow Lite as this is lighter and faster for the computer to deploy and easier to implement [13]. The code that was utilized to complete the above-described process is displayed in appendix D.2.

5.1.2 Testing

After training, the machine learning model was tested for its effectiveness. For this, the images from the initial test-set (section 3.1) were fed to the model in the form of a matrix of BGR-values for every pixel (BGR is Blue, Green, and Red). The model would, in turn, analyse the image and provide its own annotations, hereafter called predictions. The predictions of the model were first checked by hand to ensure no malfunctions occurred. Some notable behaviour of the model came to light, these are shown in figures B.1, B.2, B.4 and B.3.

Initial Model Prediction Results

Figure B.1 displays five images from the predictions made by the model. These images are important to note as the output format of the model aligns with the vision set up by the program of requirements. The images show that the model has at least the ability to recognise the prominent objects in the image belonging to the main house visible.

Besides favourable cases, several problem cases are present within the prediction results. The problem cases present themselves as bounding boxes, that have labels which do not correspond with the object within this bounding box. In addition, some bounding boxes are drawn outside the region of interest without a clear characteristic which conveys why.

Figure B.2 displays six images singled out from the predictions of the model. Several prediction instances label windows as doors. Substantially more windows on the ground floor with architectural style similar to the door are labelled as doors as opposed to windows on higher elevations which are rarely mistaken as doors.

Figure B.3 has four images chosen from the predictions which present miss-classification of steps. In figure B.3a bounding boxes labelled as steps are in the sky which is outside the area of interest, which is the main house in the image. Furthermore, the two steps leading to the front door were not detected. In both figures B.3b and B.3c, dark-coloured brick walls are confused for steps. Similarly in figure B.3d, dark-coloured street tiles are detected as steps.

At last, figure B.4 depicts two images of houses photographed at an angle. For these instances, the model is capable of depicting a house, albeit with a bounding box almost surrounding the entire image. Any other classes of interest (which are doors and steps) are not detected.

Performance Analysis

With the predictions that are provided by the system, a performance analysis according to the metrics in chapter 4 can be executed. This performance analysis is done using an IoU-threshold of 0.5 to determine if a prediction is correct, as was described in section 4.2.1

Multi-class labelling performance The confusion matrices of the performance for the initial model are shown in figure A.1 for the objects house, door, and step. The initial model has a high tendency to label more objects in the images than there are present, as can be seen in the significant percentage of false positives. Furthermore, an increase in the confidence threshold has a positive effect on the class house but not on the classes door and step, visible by the change in the percentage of true positives. The performance for houses is much better than for steps but it is still not sufficient with a maximum share of correct predictions of 53.3%.

The precision and recall curves for the initial model are shown in figure 5.2. As can be seen in the graphs, the curves tend to stay quite close to the $(0, 0)$-coordinate with an AUC of 0.38, 0.14 and 0.03 for houses doors and steps respectively. The ideal case is an AUC of 1, as was mentioned in section 4.1.2.

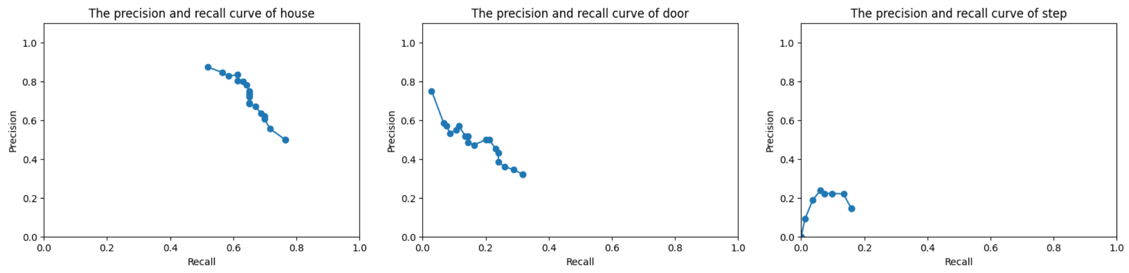


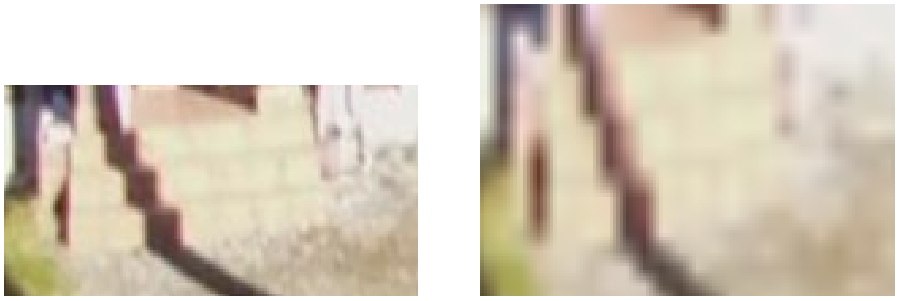
Figure 5.2: The precision and recall curve for the three classes house, door and step.

Object Localization Performance For the initial model, the mean square error and the root mean square error were 29.40 and 5.42 respectively. These values are on a 320x320-image so the RMSE normalized to the image size equals 1.69%. Ideally, the MSE, RMSE, and NRMSE should approach zero, indicating perfectly located bounding boxes (mentioned in section 4.2.2).

mAP For the initial system, the mAP was found to be 31.25% for houses, 5.56% for doors and 0.8% for steps. The ideal value is an mAP of 100% as was mentioned in 4.3.

Performance Shortcomings The initial model had a tendency to label too many objects in an image, mislabelled some objects, and had trouble identifying smaller objects. The roots of the observed problems, were found to be label noise and a low resolution used by the model. The label noise problem was addressed in section 3.2.

The model that was trained used an input-resolution of 320x320 pixels but the actual street view-images were of much larger resolution. When these street view-images are compressed to the size that the model uses, crucial details are lost. Steps are quite small objects in images and when the image is reduced in resolution to fit the 320x320 model input, their features can get lost. The loss of detail is visualized in figure 5.3. Due to the 'black box'-nature of the model, it is not known what unique identifiers/features are used to identify an object but it is apparent that detailed objects are either not detected or detected with very low confidence.



(a) (b)

Figure 5.3: Difference in level of detail between an image that isn't compressed (a) and one that is (b).

5.2 Final Design

As mentioned before, lower performance for detailed objects is, in part, due to the low input resolution employed by the model. This low resolution required the input images to be compressed to a resolution of 320x320, down from orders of 700x1800. Compressing the image forces the content that is displayed by multiple pixels to be contained in only one pixel. As objects like steps often don't comprise many pixels in an image, the compression can often remove most information needed to identify (individual) steps and make it difficult for even a human observer to do this. When analyzing the results in 5.1.2, it was apparent that the performance of the model in identifying houses was better than for steps. This was attributed to the fact that houses were large objects in the images and, thus, not much information was lost when the image was compressed. This observation led to the strategy of using multiple models in a single pipeline, the development of this system is the subject of this section.

Each model in this multi-model system was trained to identify specific elements such as houses, doors, staircases, and steps. The higher levels of this pipeline are used to identify regions of interest and the rest of the information in the image is then cut out. Cropping an image to 320x320 after cutting out some of the useless information preserves more useful information as it compresses this information less. It should thus be easier for the model to identify details. This approach is similar to the Selective Region Enlargement Network of [29]. The approach can also be justified as the only house of interest is the one in the center, so others can be removed. Furthermore, the assumption can be made that doors should only be inside the house, staircases can only be under the door, and steps can only be within a staircase. A schematic representation of this system is shown in figure 5.4.

5.2.1 Model Architectures

To implement the new model structure mentioned above, three different model-architectures were used. Two were generated using the TensorFlow API, these algorithms were 'SSD MobileNet v2 FPNLite', using a 320x320 resolution, and 'Faster R-CNN Inception ResNet V2', which uses a 640x640 resolution. The third model was the 'YOLOv8x' which also uses a 640x640 resolution and is trained using the ultralytics YOLO library.

Each model was only trained on images that contained a label for the object of interest, this was done both to avoid unnecessarily long training time, as more steps are required for a larger data set to go over

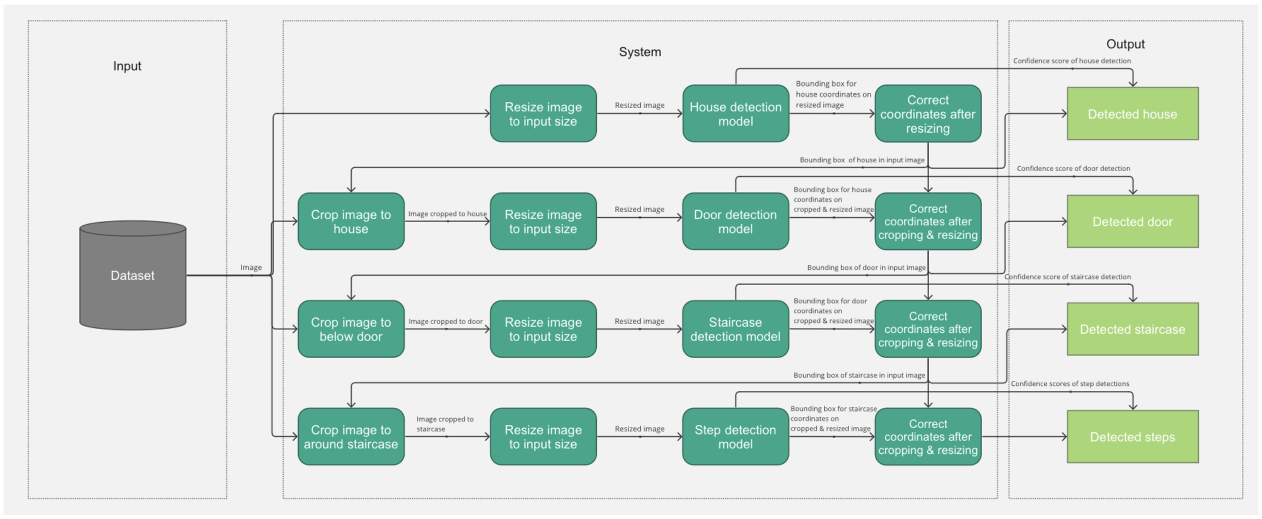

Input
Dataset
Image
System
Resize image to input size
Resized image
House detection model
Bounding box for house coordinates on resized image
Correct coordinates after resizing
Bounding box of house in input image
Crop image to house
Image cropped to house
Resize image to input size
Resized image
Door detection model
Bounding box for house coordinates on cropped & resized image
Correct coordinates after cropping & resizing
Bounding box of door in input image
Crop image to below door
Image cropped to door
Resize image to input size
Resized image
Staircase detection model
Bounding box for door coordinates on cropped & resized image
Correct coordinates after cropping & resizing
Bounding box of staircase in input image
Crop image to around staircase
Image cropped to staircase
Resize image to input size
Resized image
Step detection model
Bounding box for staircase coordinates on cropped & resized image
Correct coordinates after cropping & resizing
Output
Confidence score of house detection
Detected house
Confidence score of door detection
Detected door
Confidence score of staircase detection
Detected staircase
Confidence scores of step detections
Detected steps

Figure 5.4: Schematic representing the data flow in the concept of the final approach

all images and to avoid images that miss labels as they were overseen. This choice was based on [30]. To implement this, the data set was split using the code outlined in appendix D.1. The selected images and their corresponding labels were organized and stored separately, ensuring their availability for training each individual model. Additionally, the images will be cropped to a region of interest after each stage of the model when the system is deployed. As such, the images were cropped in a similar way during the training of the model to ensure that the model is trained on comparable data. The code to do this is shown in appendix D.3.

SSD MobileNet V2 FPNLite

The 'SSD MobileNet v2 FPNLite'-algorithm was also used in the initial design. The model was chosen for the new design as the performance for houses was reasonable for the purpose within the pipeline. Thus, there was a possibility of an improved performance now that the region of interest-selection can be used and a newer dataset, described in 3.2, can be used. The performance of this model architecture trained on cropped images from the updated data set is discussed here.

The MSE, RMSE, and NRMSE that were observed for this newer model are different for all classes, this is summarized in table 5.1 shown below. As can be seen, these metrics are also slightly improved compared to the previous iteration of this model that was deployed in the initial design.

Table 5.1: The MSE, RMSE, and NRMSE for the 'SSD MobileNet v2 FPNLite'-algorithm on predicting bounding boxes for four classes: house, door, staircase, and steps.

Model	MSE	RMSE	NRMSE
House	18.22	4.27	1.33%
Door	25.91	5.09	1.59%
Staircase	26.86	5.18	1.62%
Steps	31.52	5.61	1.75%
Average of all	27.82	5.27	1.65%

It is worth noting that the average MSE, RMSE, and NRMSE are not equal to the arithmetic mean of the MSE and RMSE for all four labels as the average is the average for all detected instances. The values presented here are on a 320x320-image so the NRMSE equals 1.65%.

The confusion matrices have been computed on images that are cropped in the manner they would be in the actual model and at three thresholds: 25%, 50%, and 75% (figure A.2). When compared to figure A.1, it can be seen that the newer iteration of this model architecture performs better than the first one. This is apparent from the increased share of true positives and likely due to the new dataset (section 3.2) and, for smaller objects, the region of interest enlargement.

Faster R-CNN Inception ResNet V2

Faster R-CNN is a network that aims to detect objects in images faster by using region proposal networks (RPNs) instead of region proposal algorithms. RPNs are fully convolutional networks (explained in 1.2.4) that predict object bounds and objectness scores at each position, and share full-image convolutional features with the detection network. This enables nearly cost-free region proposals, which are used by Faster R-CNN for detection. Faster R-CNN and RPN improve the detection accuracy, speed, and efficiency of the network [31]. The 'Inception ResNet V2'-algorithm is an improved version of the 'Inception V3'-algorithm that uses residual connections to improve training speed and accuracy [32]. The 'Faster R-CNN Inception ResNet V2'-algorithm is a variant of Faster R-CNN that uses the Inception ResNet V2 as its backbone feature extractor.

This model was chosen because of its higher resolution of input images, which is 640x640, and because of its impressive performance as described in papers such as [33, 34]. The 'Faster R-CNN Inception Resnet V2'-algorithm was trained on the new data set that was described in section 3.2 and, as it is also deployed using the TensorFlow API, it was trained following the same code used to train the 'SSD MobileNet V2 FPNLite'-algorithm. This training code is described in section 5.1.1.

The MSE, RMSE, and NRMSE observed for this model for all classes are summarized in table 5.2 shown below. Comparing the contents of this table to table 5.1, one can see a significant improvement in these metrics, hinting at better performance by this 'Faster R-CNN Inception Resnet V2'-algorithm. The values in table 5.2 are on a 640x640-image so the NRMSE equals 0.69%.

Table 5.2: The MSE, RMSE, and NRMSE for the 'Faster R-CNN Inception ResNet V2'-algorithm on predicting bounding boxes for five classes: house, door, staircase, and steps.

Model	MSE	RMSE	NRMSE
House	11.47	3.39	0.53%
Door	15.36	3.92	0.61%
Staircase	18.94	4.35	0.68%
Steps	21.63	4.65	0.73%
Average of all	19.76	4.45	0.69%

The better performance of this model compared to the newer version of 'SSD MobileNet v2 FPNLite'-algorithm from 5.2.1 can also be seen in the confusion matrix that is shown in figure A.3, where the share of true positives is larger than in A.2.

YOLOv8x

YOLOv8 is a state-of-the-art model for object detection, image classification, and instance segmentation in computer vision. It is a very new model that was released on January 10th, 2023 by Ultralytics. YOLOv8 has a DarkNet-53 backbone network that has a deeper convolutional neural network, allowing for better feature detection. YOLOv8 also has a larger feature map and Feature Pyramid networks that use different scales of feature maps to capture more complex relationships between differrent features, both result in a higher accuracy [35]. It also uses augmentation to expose the model to different variations of the images during training. YOLOv8 can be used for various computer vision tasks, such as object detection, classification, and segmentation, on custom or public data sets [11, 36].

YOLOv8 was chosen for its impressive accuracy and its ease of use. The YOLOv8 algorithm also allows one to use a 640x640-resolution which is useful for the preservation of details. YOLOv8x is the largest version of YOLOv8 with 68.2M parameters, its inference time is the highest out of all YOLOv8 models but its accuracy is also the highest ([37]). The MSE, RMSE, and NRMSE are summarized in table 5.3 shown below. Comparing the contents of this table to table 5.1 and table 5.2, one can see that this model is the best-performing model in terms of object localization performance.

Table 5.3: The MSE, RMSE, and NRMSE for the 'YOLOv8x'-architecture on predicting bounding boxes for five classes: house, door, staircase, and steps.

Model	MSE	RMSE	NRMSE
House	9.38	3.06	0.48%
Door	12.91	3.59	0.56%
Staircase	12.95	3.60	0.56%
Steps	14.63	3.82	0.60%
Average of all	13.02	3.61	0.56%

The 'YOLOv8x'-algorithm performs even better than the others presented before as can be seen in the confusion matrix that is shown in figure A.4, where the share of true positives is the highest out of all cases up to now.

5.2.2 Building the Top-Level

As could be seen in section 5.2.1, the models that were generated using YOLOv8x show the best performance for all labels. With much higher precision, reflected by the larger share of true positives, and a much lower MSE, RMSE, and NRMSE. As such, the YOLOv8x-models were the ones that are deployed by the final model.

The models employed in this project are as follows: model 1 - house, model 2 - door, model 3 - staircase, and model 4 - steps. These models play a crucial role in accurately detecting and counting steps within a house. The models need to be combined in order to deploy them in a way that allows them to work as intended and to ensure the best possible performance. The code that combines the models will henceforth be referred to as 'the top-level', it is an updated version of what is shown in figure 5.4, it partially resembles the code that is used to crop the images in training as shown in appendix D.3. The system is shown in figure 5.10

Identifying the House

The first model that is deployed is the model that finds the house. This model may detect multiple houses but as the center house is the only one of interest, this is the only one that is selected. The selection of the middle house is done by averaging the x-coordinates of the bounding boxes and finding the one that has the lowest deviation from the center.

The house is the first region of interest so once it has been identified, all information around it is removed. A margin of 10% of the image width is maintained around the sides of the bounding box as a staircase can be partially off to the side of a house and cropping the image this way can ensure that it is not cropped out. If the model doesn't detect a house, the image is not cropped at all and is continued to the next stage as it is.

An illustration of the result of this process is shown below in figure 5.5. The green line represents the boundary around which the image will be cropped whilst the red line is the bounding box that was detected for the house.

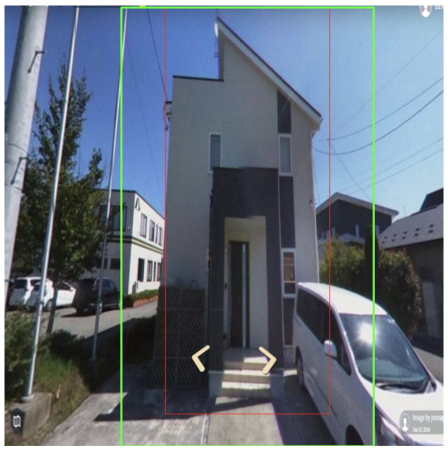


Figure 5.5: Bounding box (red) and cropping boundary (green) around house after model 1.

5.2.3 Identifying the Doors

Once a house has been found and all the information in the image outside this region of interest has been removed, doors within the house can be detected. The rest of the models will be deployed on the results of the images cropped to every individual door in the image.

After a door is detected by the model, the image is cropped only in the y-direction. The same argument as giving a margin around the house applies to the reason for only cropping around the door in the y-direction, a staircase could go sideways so cropping in the x-direction could remove this staircase. For a similar reason, images are cropped to the height of the middle of the bounding box around the door so as to not remove useful information from the image. This height is found by averaging the y-coordinates of the bounding box. If no doors are found, the image is not cropped and the output image is the same as the input image. The result of this cropping is visualized below in figure 5.6

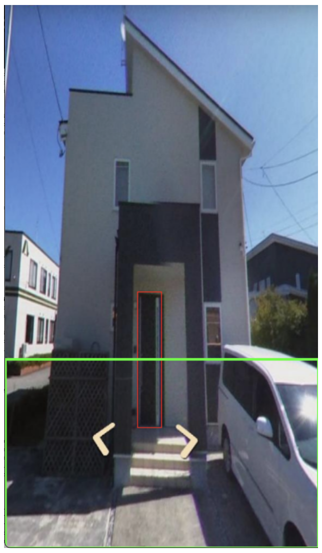


Figure 5.6: Bounding box (red) and cropping boundary (green) around the door after model 2.

Identifying staircase

Detecting a staircase is the final stage before detecting the steps that are on this staircase. For every image cropped around a door, only one staircase is selected, which is the shortest one. It would be possible that the image contains multiple staircases of varying sizes leading up to the same house. The end result only needs to contain the lowest amount of steps leading up to a door, thus the shortest staircase must be selected. The shortest staircase is selected as the one with the smallest difference between the minimum and maximum y-coordinate of the bounding box around the staircase.

Once a staircase has been detected, the image is cropped in both the x-direction and the y-direction. To prevent any details, that could be used to identify the staircase, from being cropped out of the region of interest, the top-level ensures that there is a 10% margin around the staircase. This 10% refers to the width of the image after the crop around the house as this is the only stage, before the staircase, where the x-coordinates are cropped and the height of the image after the crop around the door. If no staircase is found, both the bounding box and cropping boundaries are defined as the borders of the input image. The result of this stage looks as shown below in figure 5.7.

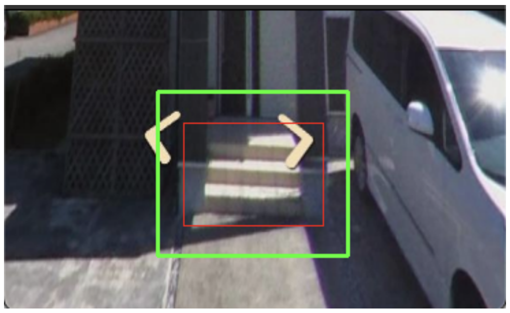


Figure 5.7: Bounding box (red) and cropping boundary (green) around the door after model 3.

Finding the steps

Once the staircase has been identified, it is certain that the steps must be within this staircase. At this stage the model has three different crops of the image, the crop around the region of interest of the house, the door, and the staircase. These three different crops allow the models to use as much context as possible. Just taking the last crop of the staircase was found to not be enough as the model sometimes missed steps.

The IoU threshold for detected bounding boxes of steps with one another is set to 35%, this ensures that a step is not detected multiple times. This is different from the IoU threshold value established in 4.2.1 as the application differs. Before it was about determining whether a detection was correct by comparing the detected bounding box with the labelled bounding box. The detection result with the most amount of steps detected within the staircase is selected as the one with the correct result. It is assumed that the most steps within the staircase would mean that all steps on the staircase are detected and, thus, that the least are missed. This process is demonstrated in figure 5.8.

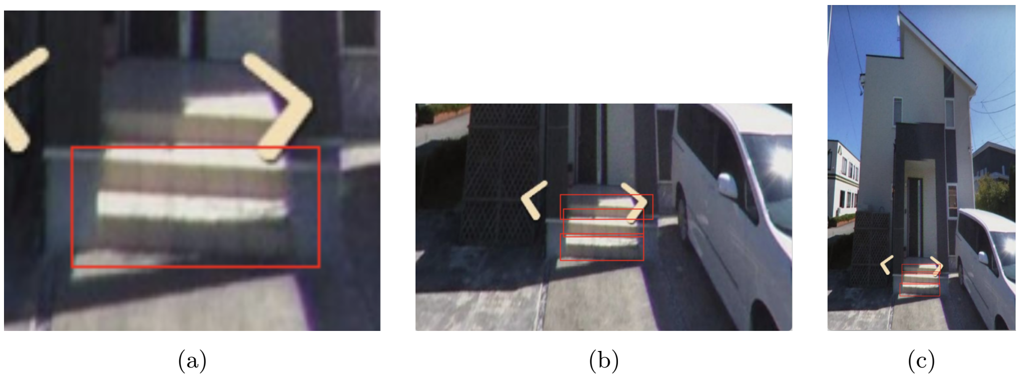
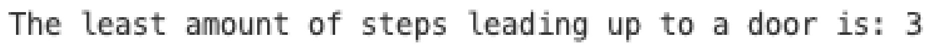


Figure 5.8: The stair detection results from the three different step-detection models. The staircase-crop (a) allowed for the detection of only one step. Step-detection using the door-crop (b) or house-crop (c) detected all 3 and these are then used as the final result.

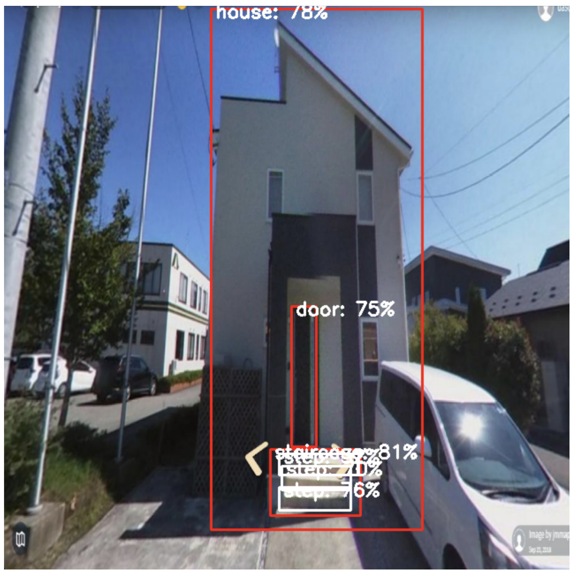
Creating Output

Now that the amount of steps have been determined for each door, the output can be formatted. Only the door with the least amount of steps underneath it is of interest as such the top-level looks at all steps and determines which one has the least amount of steps. This number of steps is printed in the terminal. Furthermore, the model keeps track of the amount of material that is cut out of the image and reconstructs the position of the bounding boxes in the original image from the ones that have been detected in the cropped images. The annotated images are displayed to the user if so desired, these annotations contain the bounding box, the label, and the confidence the model has in the detection. These annotations are also written to a .txt-file that has the same name as the input image. Illustrations of this output is shown below in figure 5.9.

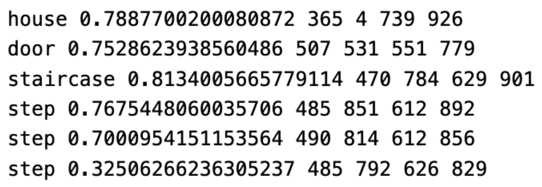


```
The least amount of steps leading up to a door is: 3
```

(a)



(b)



```
house 0.7887700200080872 365 4 739 926
door 0.7528623938560486 507 531 551 779
staircase 0.8134005665779114 470 784 629 901
step 0.7675448060035706 485 851 612 892
step 0.7000954151153564 490 814 612 856
step 0.32506266236305237 485 792 626 829
```

(c)

Figure 5.9: The output of the top-level which includes three things: the least amount of steps leading up to a door printed in the terminal (a), a visualisation of the detections on the image (b), and a list of the label, confidence, and coordinates of each detection made (c).

The final system is schematically shown in figure 5.10.

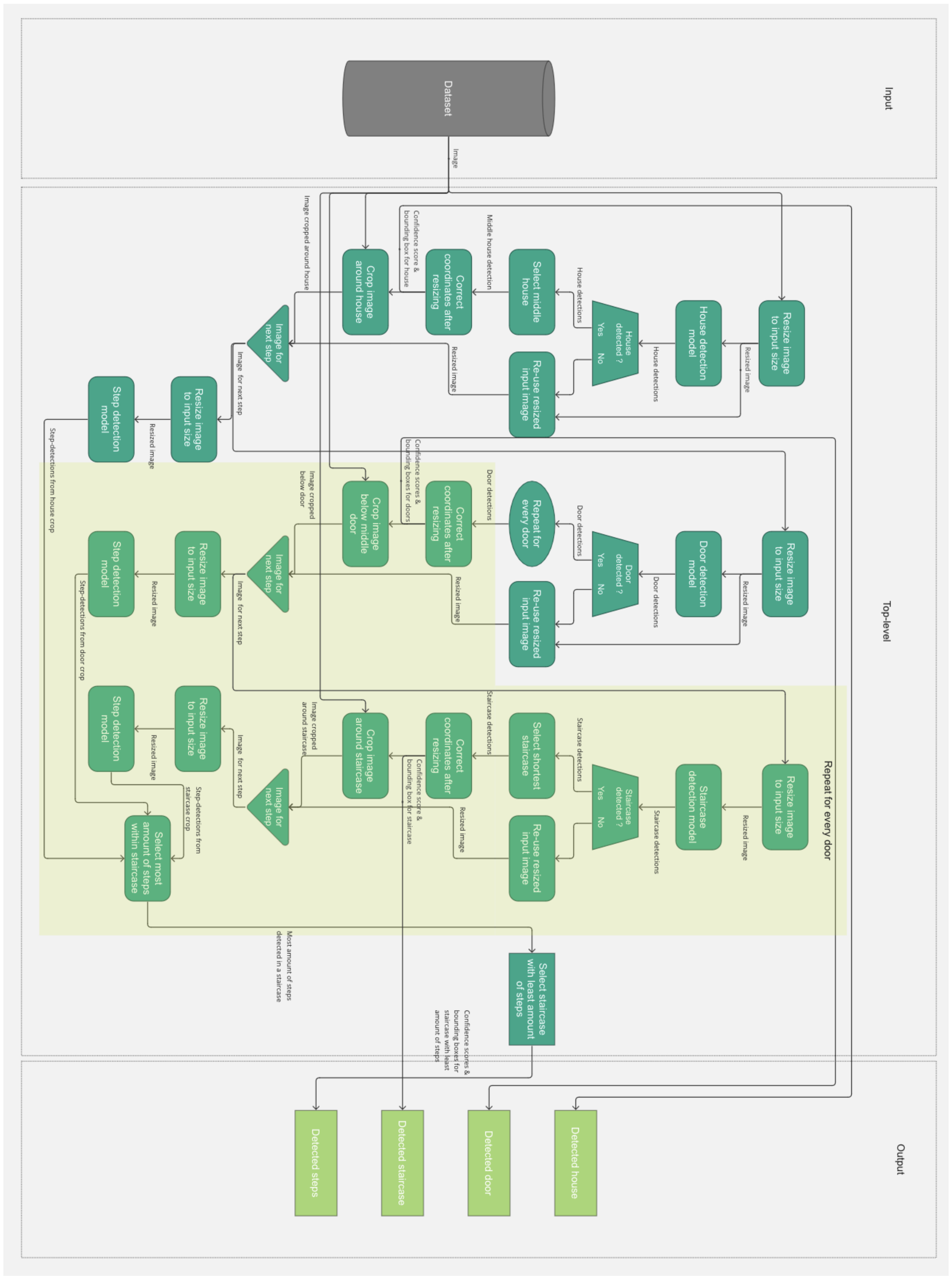

Input
Dataset
Image
Top-level
Resize image to input size
Resized image
House detection model
House detections
House detected ?
Yes
No
House detections
Select middle house
Middle house detection
Correct coordinates after resizing
Confidence score & bounding box for house
Crop image around house
Image cropped around house
Re-use resized input image
Resized image
Image for next step
Image for next step
Resize image to input size
Resized image
Step detection model
Step-detections from house crop
Resize image to input size
Door detection model
Door detections
Door detected ?
Door detections
Repeat for every door
Door detections
Correct coordinates after resizing
Confidence scores & bounding boxes for doors
Crop image below middle door
Image cropped below door
Step-detections from door crop
Repeat for every door
Staircase detection model
Staircase detections
Staircase detected ?
Select shortest staircase
Staircase detections
Crop image around staircase
Image cropped around staircase
Confidence score & bounding box for staircase
Step-detections from staircase crop
Select most amount of steps within staircase
Most amount of steps detected in a staircase
Select staircase with least amount of steps
Confidence scores & bounding boxes for staircase with least amount of steps
Output
Detected house
Detected door
Detected staircase
Detected steps

Figure 5.10: Schematic representing the data flow in the final approach

Chapter 6

Results

The performance of the final design will be judged in the same way as the various methods used to analyse the performance of the initial design (section 5.1.2), with the metrics from chapter 4. The performance metrics were: Intersection over Union (IoU), mean square error (MSE) root MSE (RMSE), and normalized RMSE (NRMSE) for object localization. The metrics confusion matrices and precision-recall curves are utilised for multi-class classification. The mean average precision (mAP) incorporates both. Lastly, the effectiveness of the final design on the project's intended application will be judged with a confusion matrix.

The IoU was used to check the validity of a prediction, as such no quantifiable results are presented. The metrics presented in this chapter have been obtained by running the top-level for the 100 images in the test set, this took 6 minutes and 51.6 seconds which is an average time of 4.11 seconds to analyze an image.

6.1 Multi-class Labelling Performance

6.1.1 Confusion Matrices

As can be seen in the confusion matrix shown in figure A.5, the top-level model shows the same performance for houses but an improved performance for the detection of doors, staircases, and steps. What is especially interesting is the 95% of true positives for steps at a low certainty of 25%, this is a precision of 98.7%. As can be seen, the model performs best for lower thresholds, this is why the confidence thresholds of all models have been set at 25%. The region of interest selection along with the top-level looking for steps in multiple crops has been deemed responsible for this improvement in performance.

6.1.2 Precision and Recall

The performance metrics precision and recall are employed to analyse the correct predictions made by the final model, as explained in more detail in section 4.1.2. The precision and recall will be graphed into PR-curves 6.1.

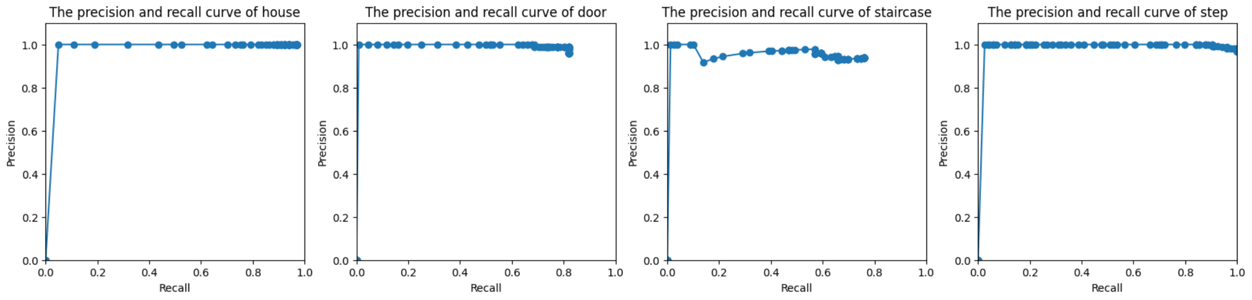


Figure 6.1: Four precision and recall curves for the four classes: house, door, staircase, and step respectively
.

As mentioned in section 4.1.2 the PR-curve of the final model should ideally approach the (1,1)-coordinate. A close-to-ideal result can be observed for the first and fourth curves. These curves represent the classes house and step respectively. The classes door and staircase fall short of reaching the ideal case PR-curve. The same holds for the AUC, the AUC is 0.95, 0.82, 0.72 and 0.98 for house, door, staircase, and step respectively compared to an ideal case of 1. This performance is, clearly, a great improvement over the initial model.

6.2 Object localization performance

6.2.1 MSE, RMSE, and NRMSE

The final model's proficiency in object localization will be analysed in conjunction with the multi-class labelling performance. The metrics MSE, RMSE, and NRMSE will be used to analyse the difference between the centre of the prediction bounding and the centre of the annotation bounding box as explained in 4.2.2. The MSE for all bounding boxes is 15.1, the RMSE is 3.89, and the NRMSE is 0.61%. This is a 48.6% decrease in MSE compared to the initial model, a 28.3% decrease in the RMSE, and 64.2% for NRMSE.

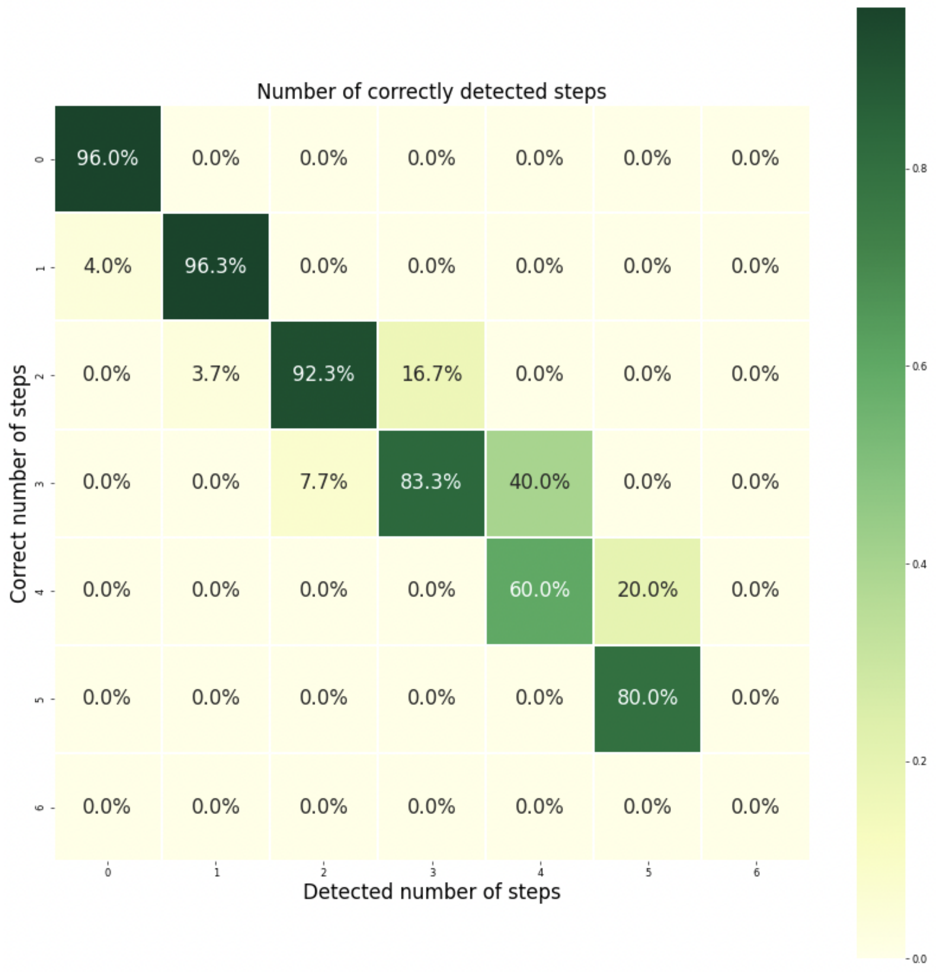
6.3 Mean Average Precision

Section 4.3 already emphasized the practicality of the mAP in the examination of both the multi-case classification and the object localization performance. The mAP result for the final model is 96.2% for house, 86.5% for door, 81.2% for staircase and 89.5% in the case of step. This is a 307.8%, 1555.6% and 11187.5% increase for house, door, and step respectively compared to the initial model.

6.4 Step Counting Performance

Lastly, the suitability of the final model on the application of the project will be analysed in a new type of confusion matrix. The confusion matrix is shown in figure 6.2. This matrix has the detected amount of steps on the x-axis and the correct amount on the y-axis. For every amount of detected steps, the confusion matrix shows the share of detections that belong to each of the correct number of steps. This means that of the images where the system predicted 0 steps, 96% of had 0 steps and 4% had 1 step. For the cases with 1 detected step, 96.3% really had 1 step and 3.7% actually had 2 steps, etc.

What can be seen in the figure is that the cells on the diagonal have the highest values, this means that most of the detected number of steps were correct. For the cases where the cell that is off the diagonal (the incorrect predictions) has a nonzero value, it is never more than 1 step off from the true value.



	0	1	2	3	4	5	6
0	96.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
1	4.0%	96.3%	0.0%	0.0%	0.0%	0.0%	0.0%
2	0.0%	3.7%	92.3%	16.7%	0.0%	0.0%	0.0%
3	0.0%	0.0%	7.7%	83.3%	40.0%	0.0%	0.0%
4	0.0%	0.0%	0.0%	0.0%	60.0%	20.0%	0.0%
5	0.0%	0.0%	0.0%	0.0%	0.0%	80.0%	0.0%
6	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Figure 6.2: Shares of detected steps compared to actual values

Chapter 7

Conclusion & Further work

7.1 Conclusion

The goal of this project was to develop an automated system that is capable of accurately detecting and counting the number of steps leading up to the front door of a house. Transfer learning on a computer vision model and effective resolution increase through region of interest enlargement have been implemented to achieve this goal.

The initial approach only used transfer learning on an 'SSD Mobilenet v2 FPNLite'-algorithm. This approach gave an idea of the capabilities and shortcomings of the architecture and illuminated what areas had to be worked on to increase the system's performance. Label noise and low image resolution were identified as the main causes for worsened performance by the system.

For a second approach, three models were tested. These three models were 'SSD Mobilenet v2 FPNLite', 'Faster R-CNN Inception ResNet V2', and 'YOLOv8x'. From these models, it was found that the 'YOLOv8x' algorithm performed best on the test set and, as such, it was used in the final design.

In the final design, label noise was resolved by ensuring that all labels were correct and the low resolution problem was resolved by using 'YOLOv8x', which has a higher resolution, as well as employing a technique whereby effective resolution is increased by identifying and focusing on regions of interest. This final approach showed a great improvement over training only a single model to identify steps from an entire image.

The final system is able to analyze an image in, on average, 4.11 seconds, which is well within the mandatory requirement of one minute. The model shows a precision (TPR) of 98.7% in detecting steps at a 25% confidence threshold, this is well above the required precision of 90%. The maximum deviation of the detected number of steps from the true number of steps leading up to a front door was found to be no more than 1 and this occurred in only a minor share of the test-set predictions. The final system can be deployed by only giving it the path to the image that has to be analyzed, this allows the user to easily loop through a folder of images and feed this to the system. The system is also able to display its outputs visually as bounding boxes that are annotated on the input image, as well as in a terminal output and a .txt-file output. The entirety of the system has been coded in Python and the necessary comments were added to increase readability.

7.2 Further work

The currently implemented system is already satisfying all predetermined requirements. However, there is room for improvement. Some are listed here:

- A model was already trained to identify garage doors but it was not yet implemented as time to the end of the project had run out. However, allowing the system to be able to detect garage doors would allow it to give a more detailed result of how much of the house is at risk of flooding as garage doors tend to be located at a lower height than front doors.
- A system to detect basements or a database of houses with a basement could also be used to improve the accuracy of how much of the house is at risk as a basement floods before the rest of the house does. A concrete idea for this has not been thought of but flooding of basements could be seen as a problem and thus some method to assert the risk of this should be devised.
- Post-processing logic could potentially be deployed to decrease the number of erroneous predictions and, thus, improve the performance and reliability of the model even more.
- If a ramp would be used to access a house, the model will miss this and return no elevation to the house despite there being some. A solution for this can be made in detecting a ramp and giving the height of a ramp by taking either an average value or the maximum height that is allowed before the ramp needs to flatten off. Further research would be required to solve this problem.

Bibliography

[1] F. Xia, X. Kang, S. WU, Q. Yang, X. Ma, P. Yang, and X. Li, "Research on dike breach risk of the hanging reach under different flood conditions in the lower yellow river," *Geographical Research*, pp. 220–239, 2008.

[2] D. Guha-Sapir, R. Below, and P. Hoyois. The cred/ofda international disaster database. [Online]. Available: https://www.emdat.be/

[3] Y. Ge, P. Cui, and X. Chen, "Strategy of the international cooperation with respect to disaster prevention and reduction in the belt and road areas," *Science & Technology Review*, pp. 29–34, 2020.

[4] S. Temmerman, P. Meire, T. J. Bouma, P. M. J. Herman, T. Ysebaert, and H. J. D. Vriend, "Ecosystem based coastal defence in the face of global change," *Nature*, pp. 79–83, 2013.

[5] Google. Discover when, where, and how we collect 360 imagery. [Online]. Available: https://www.google.com/streetview/how-it-works/

[6] F. Chen, A. Subedi, M. R. Jahanshahi, D. W. Johnson, and E. J. Delp, "Deep learning–based building attribute estimation from google street view images for flood risk assessment using feature fusion and task relation encoding," *Journal of Computing in Civil Engineering*, vol. 36, no. 6, 11 2022. [Online]. Available: https://doi.org/10.1061/(asce)cp.1943-5487.0001025

[7] Y.-H. Ho, C.-C. Lee, N. Diaz, S. Brody, and A. Mostafavi, "Elev-vision: Automated lowest floor elevation estimation from segmenting street view images," 06 2023.

[8] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *SN computer science*, vol. 2, no. 3, 3 2021. [Online]. Available: https://doi.org/10.1007/s42979-021-00592-x

[9] A. I. Khan and S. Al-Habsi, "Machine Learning in Computer Vision," *Procedia Computer Science*, vol. 167, pp. 1444–1451, 1 2020. [Online]. Available: https://doi.org/10.1016/j.procs.2020.03.355

[10] B. N. K. Sai and T. Sasikala, "Object detection and count of objects in image using tensor flow object detection api," pp. 542–546, 2019.

[11] J. Solawetz. What is yolov8? the ultimate guide. [Online]. Available: https://blog.roboflow.com/whats-new-in-yolov8/

[12] J. Terven and D. Cordova-Esparza, "A comprehensive review of yolo: From yolov1 and beyond," *ACM Computing Surveys*, 2023.

[13] G. Boesch. (2022) Tensorflow lite – real-time computer vision on edge devices. [Online]. Available: https://viso.ai/edge-ai/tensorflow-lite/

[14] E. Juras. (2022) Tensorflow lite object detection model performance comparison. [Online]. Available: https://www.ejtechconsultants.com/tensorflow-lite-object-detection-model-performance-comparison/

[15] M. F. Khan, M. U. Khan, M. A. Khan, and M. A. Khan, "Deep learning techniques: A comprehensive review," *Artificial Intelligence Review*, vol. 14, no. 3, 2021. [Online]. Available: https://doi.org/10.1007/s10462-021-10187-9

[16] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A Comprehensive Survey on Transfer Learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 1 2021. [Online]. Available: https://doi.org/10.1109/jproc.2020.3004555

[17] Ujjwalkarn. (2017, 5) An Intuitive Explanation of Convolutional Neural Networks. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[18] J. M. Vaz and S. Balaji, "Convolutional neural networks (cnns): concepts and applications in pharmacogenomics," *Molecular Diversity*, vol. 25, no. 3, pp. 1569–1584, 2021.

[19] S. Mohammadi, M. Ghatee, and H. Bejani, "Overfitting controlling methods in neural networks: a systematic review," *Artificial Intelligence Review*, vol. 54, no. 5, pp. 3577–3612, 2021.

[20] P. Baheti. Train test validation split: How to & best practices. [Online]. Available: https://www.v7labs.com/blog/train-validation-test-set

[21] M. Bernhardt, D. Castro, and R. e. a. Tanno, "Active label cleaning for improved dataset quality under resource constraints," *Nature Communications*, vol. 13, no. 1161, 2022.

[22] M. Grandini, "Metrics for multi-class classification: an overview," 8 2020. [Online]. Available: https://arxiv.org/abs/2008.05756

[23] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva, "A comparative analysis of object detection metrics with a companion open-source toolkit," *Electronics*, vol. 10, no. 3, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/3/279

[24] A. A. Jamali, "Evaluation and comparison of eight machine learning models in land use/land cover mapping using Landsat 8 OLI: a case study of the northern region of Iran," *SN Applied Sciences*, vol. 1, no. 11, 10 2019. [Online]. Available: https://doi.org/10.1007/s42452-019-1527-8

[25] S. Wenkel, K. Alhazmi, T. Liiv, S. Alrshoud, and M. Simon, "Confidence score: The forgotten dimension of object detection performance evaluation," *Sensors (Basel)*, vol. 21, no. 13, p. 4350, 2021.

[26] J. Solawetz, "Mean Average Precision (mAP) in Object Detection," *Roboflow Blog*, 11 2022. [Online]. Available: https://blog.roboflow.com/mean-average-precision//#the-map-formula-how-to-calculate-map.

[27] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and L. Zitnick, "Microsoft coco: Common objects in context," *European Conference on Computer Vision*, September 2014. [Online]. Available: https://www.microsoft.com/en-us/research/publication/microsoft-coco-common-objects-in-context/

[28] D. Oliveira. (2021) Creating tfrecords. [Online]. Available: https://www.tensorflow.org/tutorials/load_data/tfrecord

[29] J. Leng, Y. Liu, and X. Gao, "Selective region enlargement network for fast object detection in high resolution images," *Neurocomputing*, vol. 462, pp. 402–411, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231221011991

[30] E. Cole, O. M. Aodha, T. Lorieul, P. Perona, D. Morris, and N. Jojic, "Multi-label learning from single positive labels," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 933–942, 2021.

[31] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[32] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," p. 4278–4284, 2017.

[33] N. Hnoohom, P. Chotivatunyu, N. Maitrichit, V. Sornlertlamvanich, S. Mekruksavanich, and A. Jitpattanakul, "Weapon detection using faster r-cnn inception-v2 for a cctv surveillance system," pp. 400–405, 2021.

[34] V. V. Danilov, K. Y. Klyshnikov, O. M. Gerget, A. G. Kutikhin, V. I. Ganyukov, A. F. Frangi, and E. A. Ovcharenko, "Real-time coronary artery stenosis detection based on modern neural networks," *Scientific Reports*, vol. 11, no. 1, p. 7582, 2021.

[35] TowardsAI. (2023, 1) Yolov8 is here and it gets better! [Online]. Available: https://towardsai.net/p/l/yolov8-is-here-and-it-gets-better

[36] D. Reis, J. Kupec, J. Hong, and A. Daoudi, "Real-time flying object detection with yolov8," 05 2023.

[37] Ultralytics. (2023) Ultralytics yolov8 docs. [Online]. Available: https://docs.ultralytics.com/tasks/detect/

Appendices

Appendix A

Confusion matrices

A.1 Initial Design

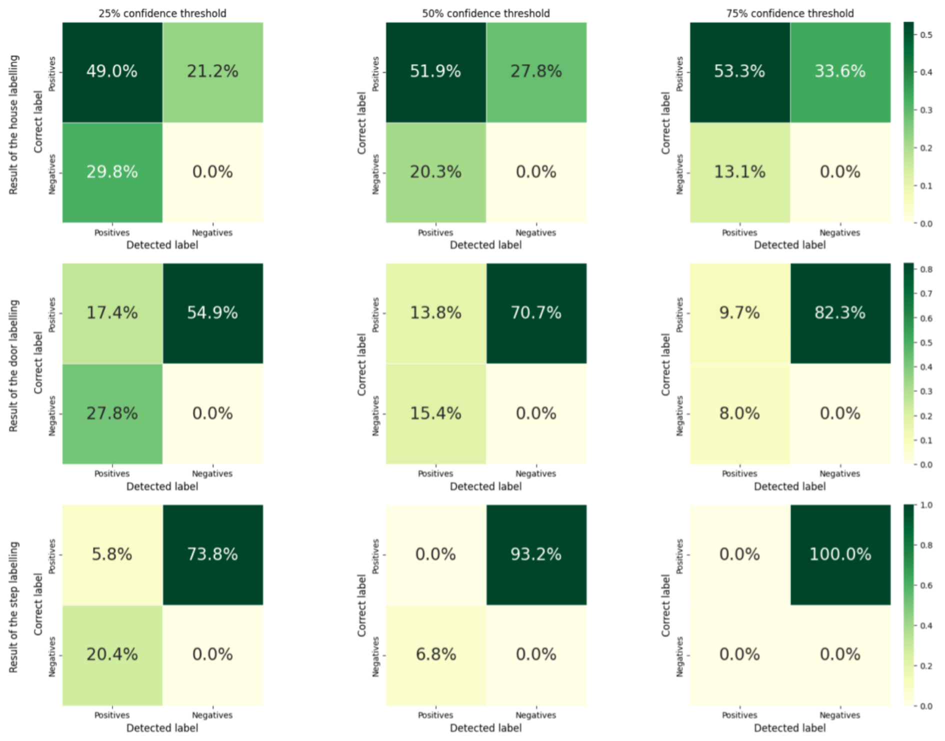


Figure A.1: Nine confusion matrices from the initial. Each column corresponds to confidence thresholds 25%, 50% and 75% and each row represents the classes house, door, staircase, and steps. The true positives are in the top-left corner, the false positives are in the bottom-left corner, the false negatives are in the top-right corner and the true negatives are in the bottom-right corner.

A.2 Final Design

A.2.1 SSD Mobilenet v2 FPNLite

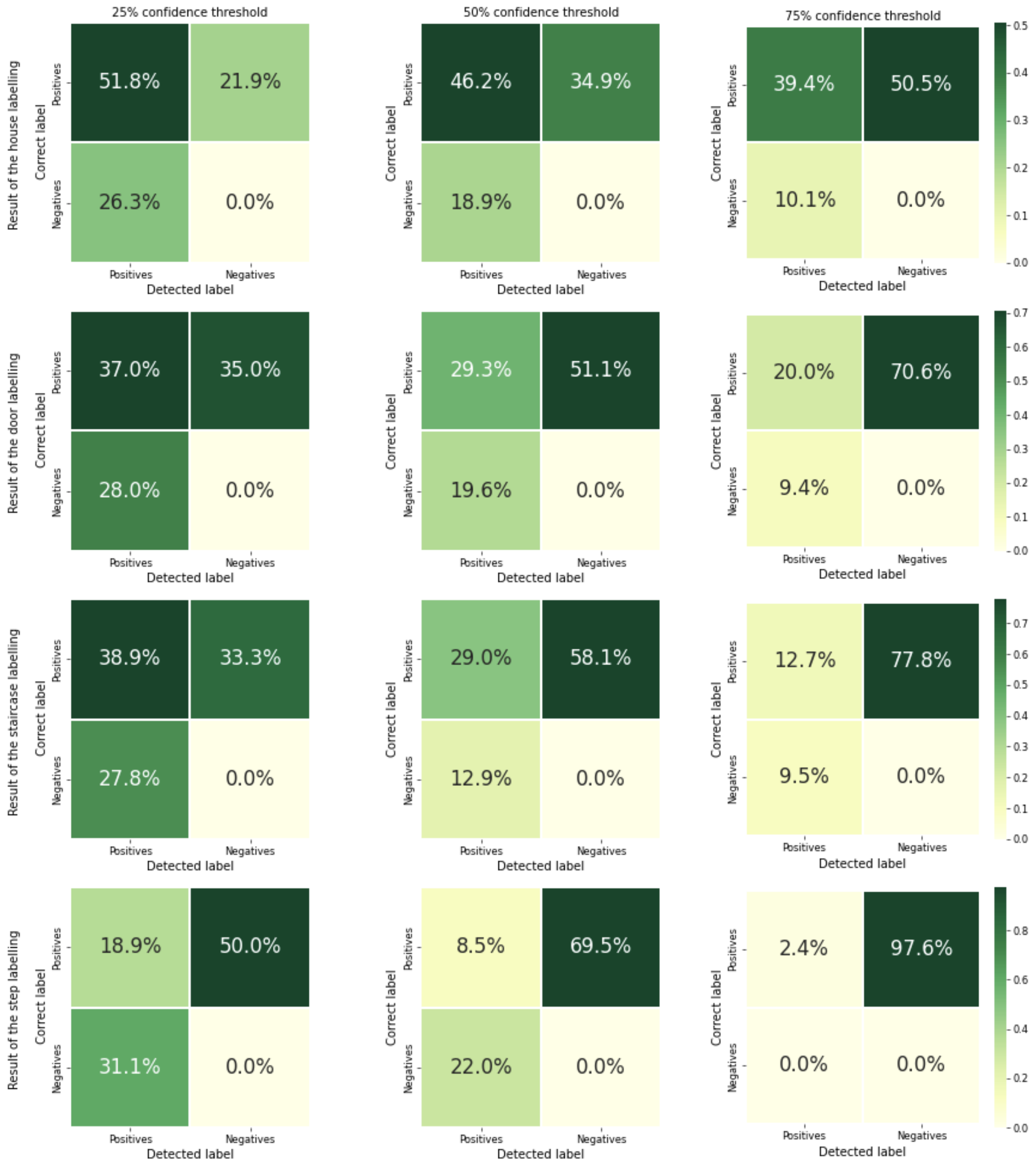


Figure A.2: The confusion matrices for the 'SSD MobileNet v2 FPNLite'-architecture on predicting five classes: house, door, staircase, and steps.

A.2.2 Faster R-CNN Inception ResNet V2

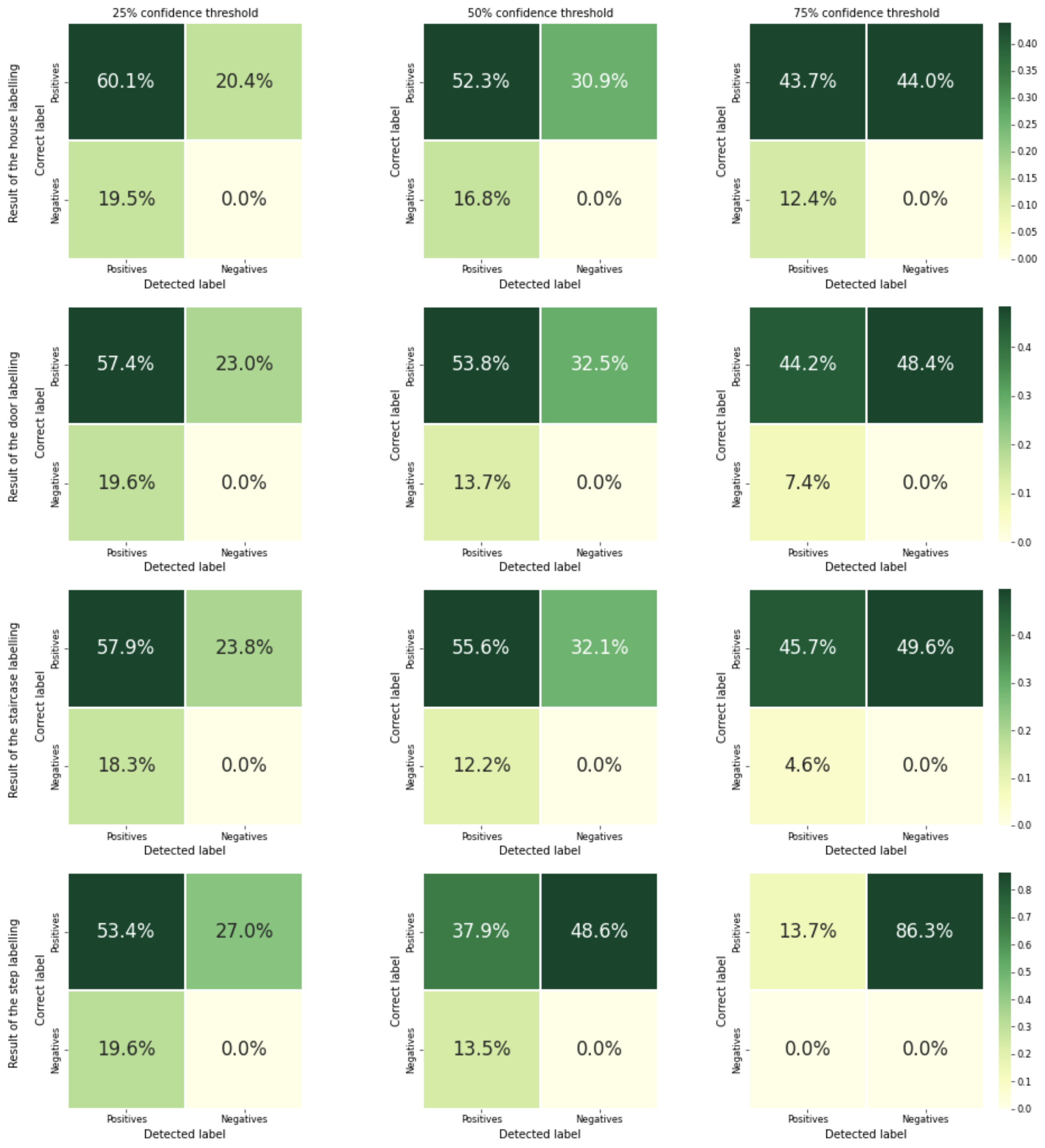


Figure A.3: The confusion matrices for the 'Faster R-CNN Inception ResNet V2'-algorithm on predicting five classes: house, door, staircase, and steps.

A.2.3 YOLOv8x

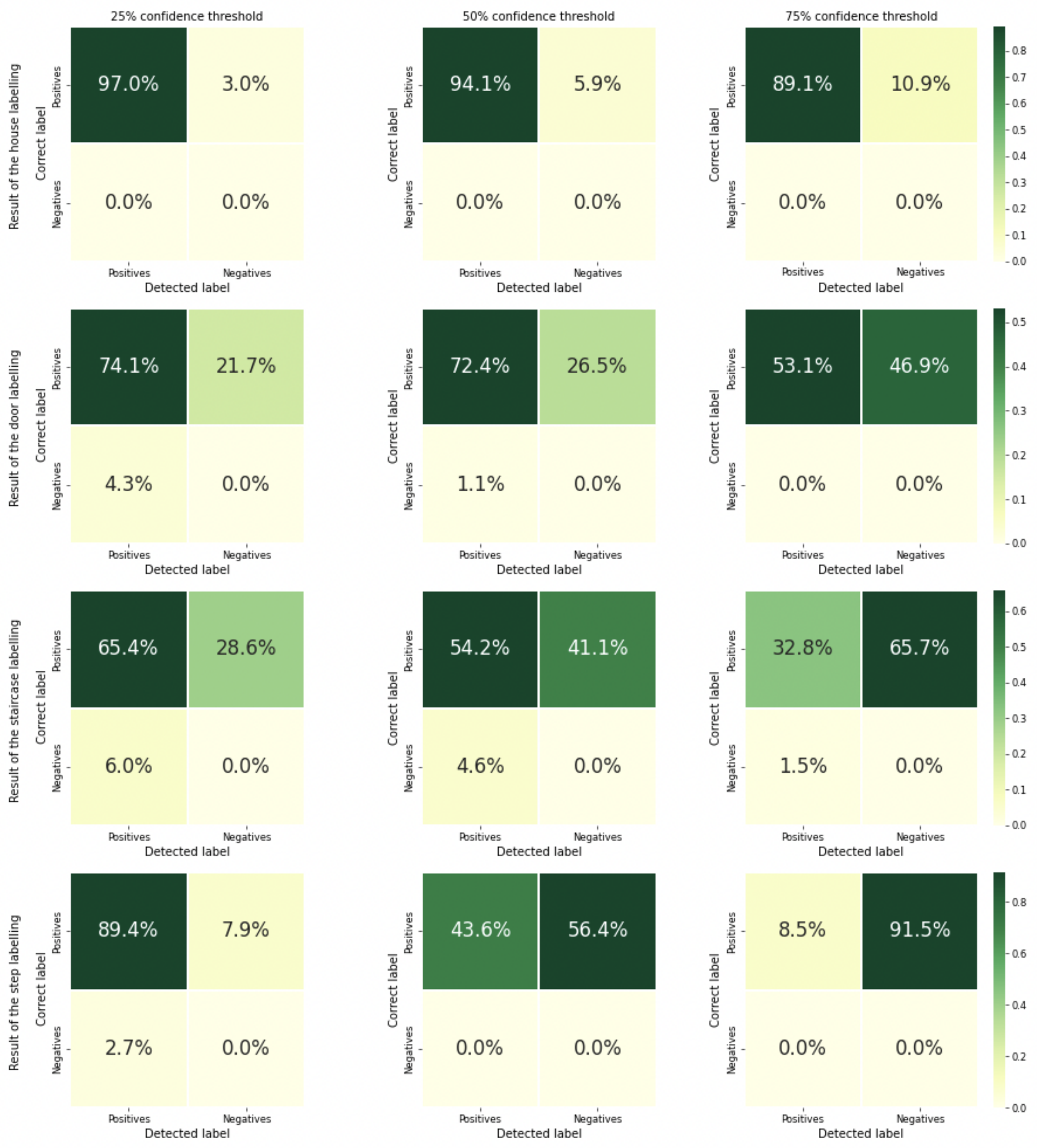


Figure A.4: The confusion matrices for the 'YOLOv8x'-algorithm on predicting five classes: house, door, staircase, and steps.

A.3 Top-level

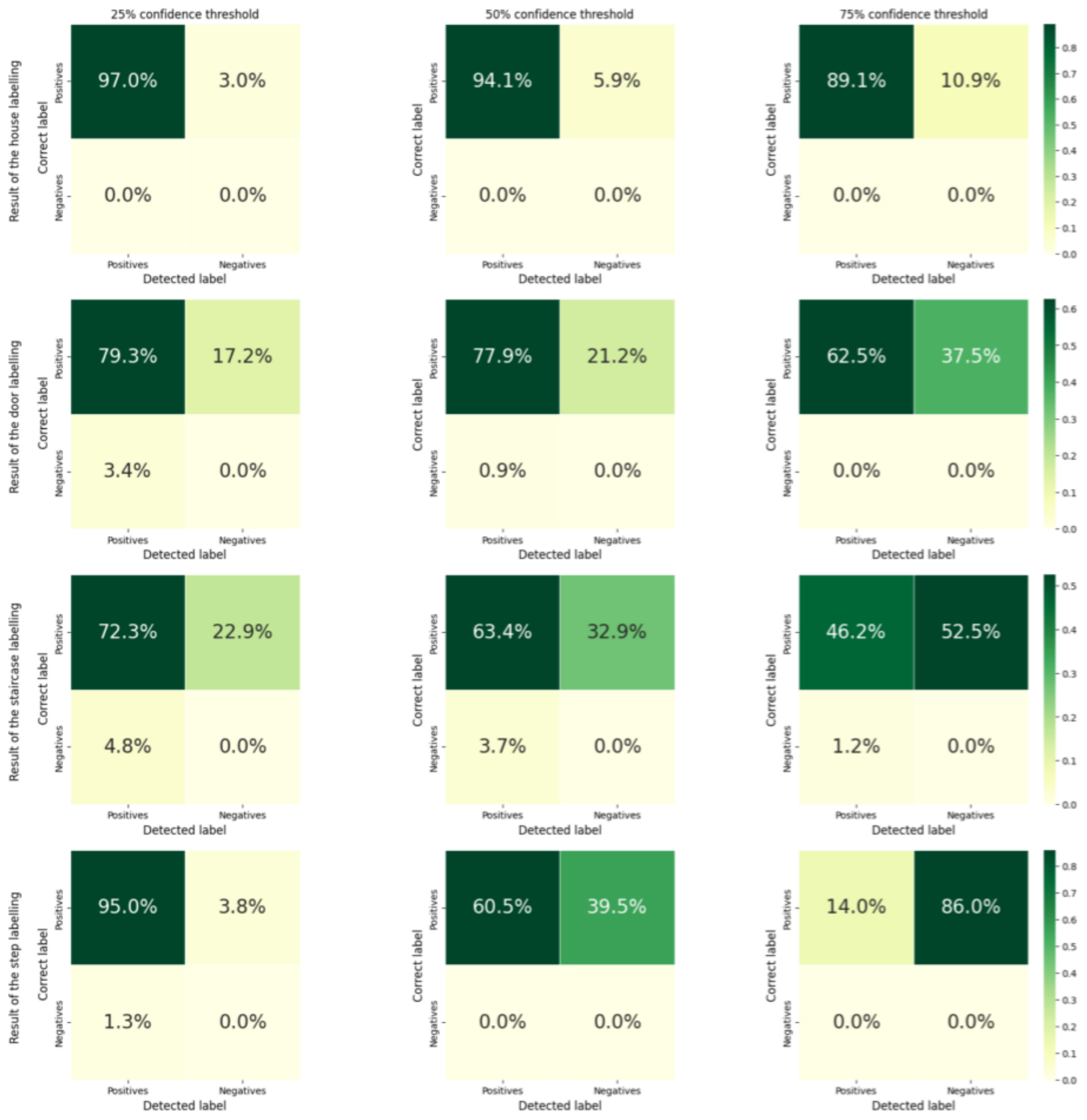


Figure A.5: Twelve confusion matrices from the final design are positioned in a three-by-four array format. Each column corresponds to confidence thresholds 25%, 50% and 75% and each row represents the classes house, door, staircase, and steps.

Appendix B

Initial model predictions

B.1 Good predictions

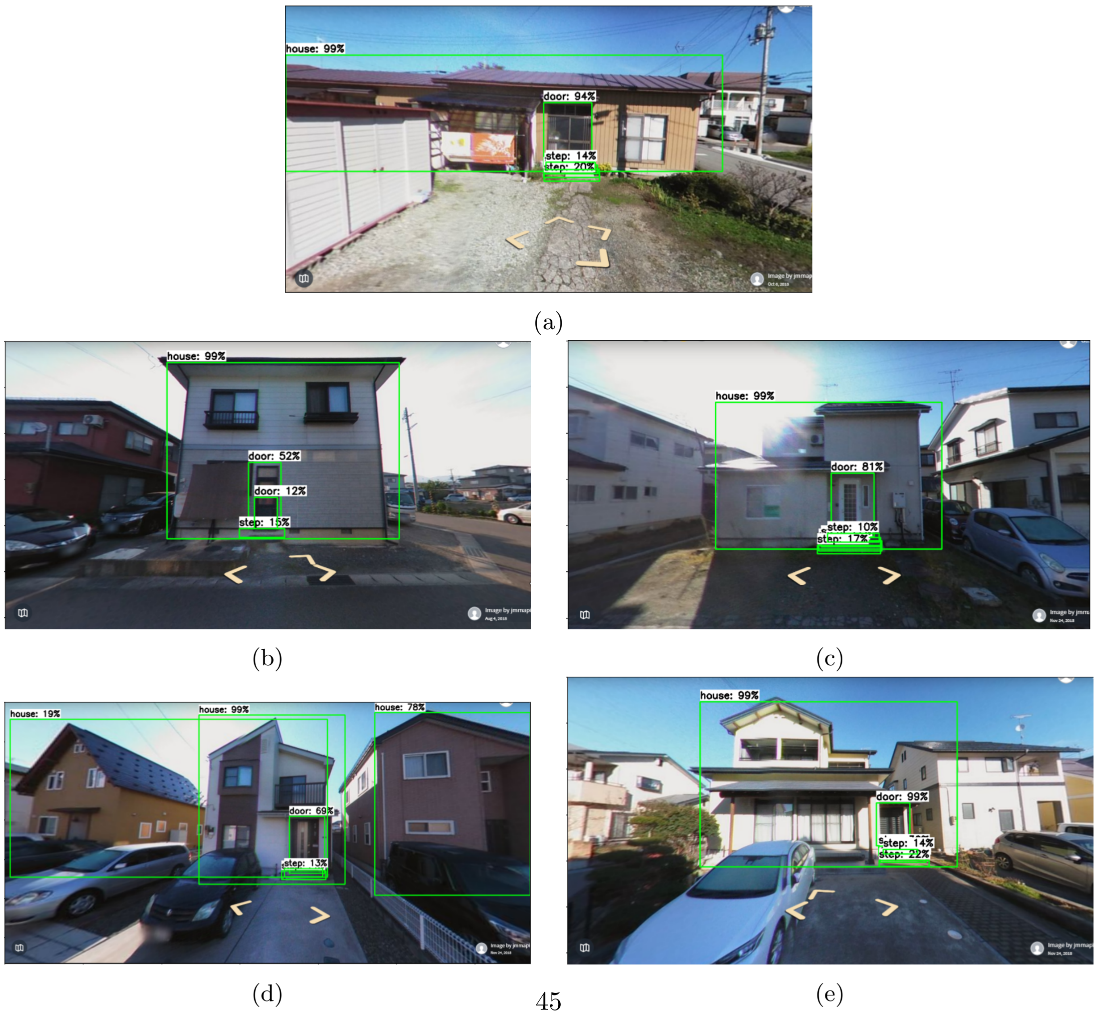
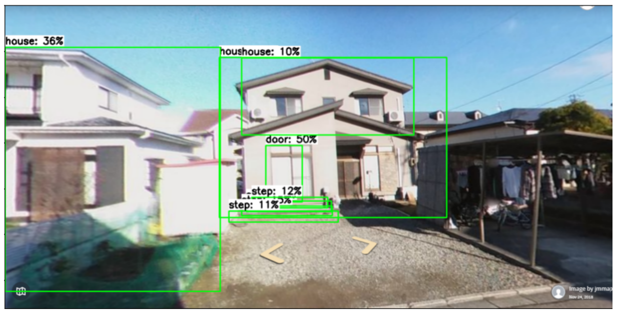
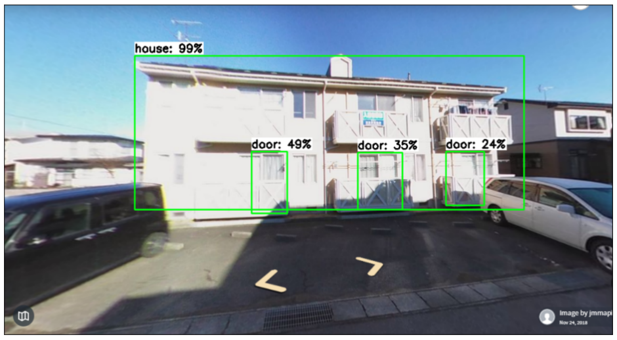


Figure B.1: Prediction results of the model. The predictions are denoted with green bounding boxes. Each bounding box has its corresponding label written directly above with a percentage representing the confidence level of the model.

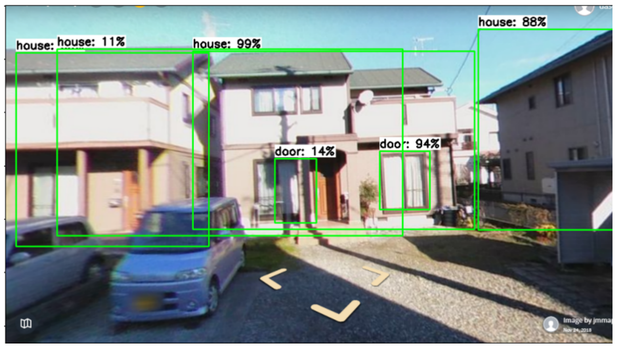
B.2 Problems with doors



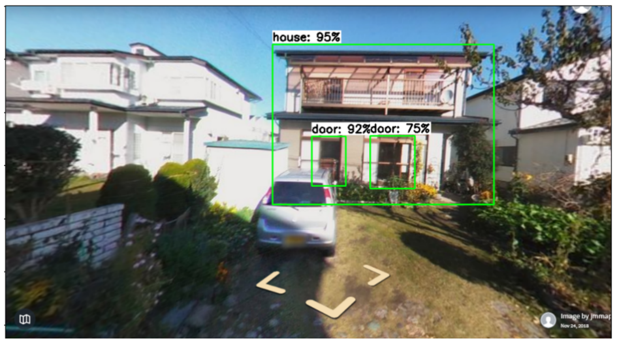
(a)



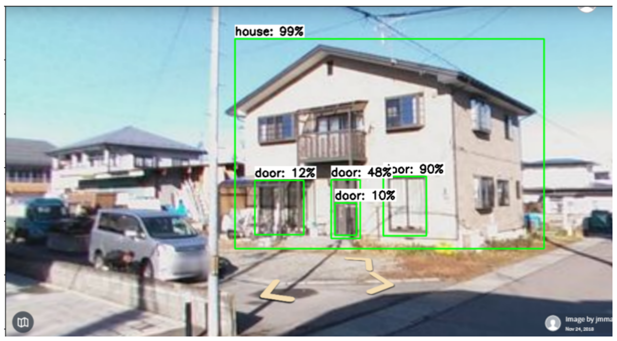
(b)



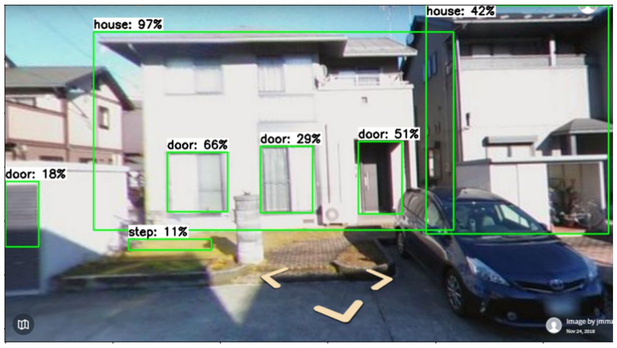
(c)



(d)



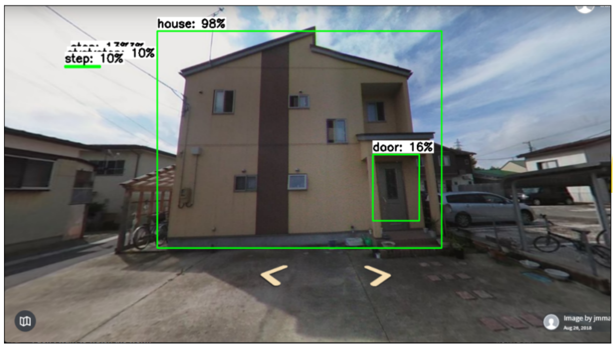
(e)



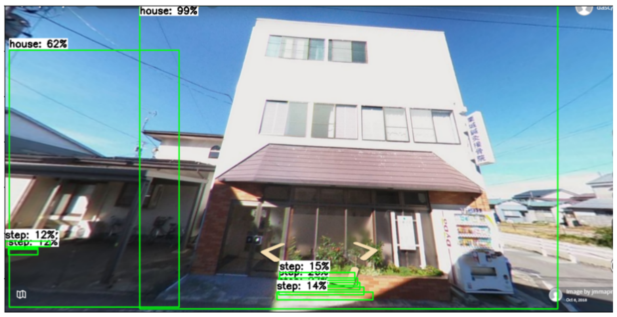
(f)

Figure B.2: Six images selected from the problem cases of the predictions of the model where objects were incorrectly labelled as doors. Each bounding box has its corresponding label written directly above with a percentage representing the confidence level of the model.

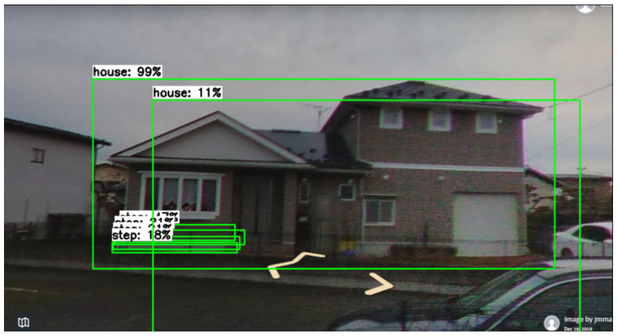
B.3 Problems With Steps



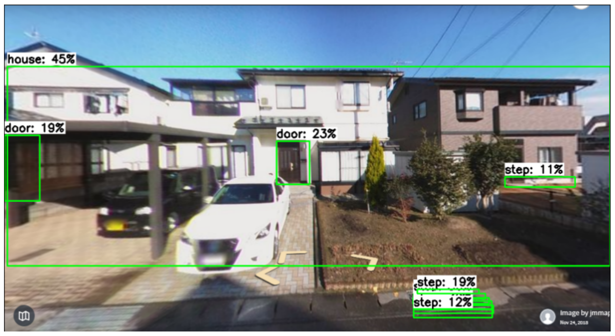
(a)



(b)



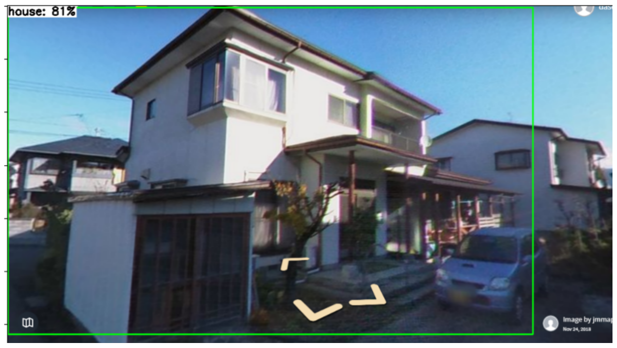
(c)



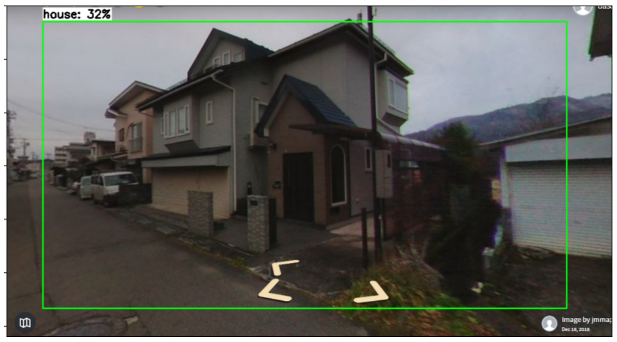
(d)

Figure B.3: Four selected images from the prediction results of the model which show mistaken predictions for steps. Each bounding box has its corresponding label written directly above with a percentage representing the confidence level of the model.

B.4 Problems With Houses



(a)



(b)

Figure B.4: Two selected images, which show a house photographed at an angled, taken from the prediction results of the model. Each bounding box has its corresponding label written directly above with a percentage representing the confidence level of the model.

Appendix C

Code for Top-level

C.1 Top-level

```python
import cv2
import os

from deploy_house_model import detect_house_y
from deploy_door_model import detect_door_y
from deploy_staircase_model import detect_staircase_y
from deploy_step_model import detect_steps_y
from deploy_garagedoor_model import detect_garagedoor_y

def full_model(image_path, image_output):

    model_path = '/Users/philipengelenberg/Library/CloudStorage/OneDrive-Personal/
    TUD/EE3L11-BAP/Group 1/Models/'

    image_bgr = cv2.imread(image_path)
    image_unstretched = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image_unstretched, (640,640))

    size = image_unstretched.shape

    imgH = size[0]
    imgW = size[1]

    house_model_path = model_path + 'fin_house/best.pt'
    door_model_path  = model_path + 'fin_door/best.pt'
    stair_model_path = model_path + 'fin_staircase/best.pt'
    step_model_path  = model_path + 'fin_step/best.pt'

    house_lbl = 0
    door_lbl  = 1
    stair_lbl = 2
    step_lbl  = 3

    house_conf = .25
    door_conf  = .25
    stair_conf = .25
    step_conf  = .25

    max_iou_step = 0.35

    least_num_steps = 1000
    best_staircase = []
    steps_best_staircase = []


    image_file_name = os.path.splitext(os.path.basename(image_path))[0]
    print(image_file_name)
    txt_output_file = '/Users/philipengelenberg/Library/CloudStorage/OneDrive-Personal/TUD/EE3L11-BAP/Group 1/txtoutputs/' + image_file_name + '.txt'
    with open(txt_output_file, 'w') as f:
        # Detect house and crop accordingly
        cropped_house, bb_house, cfacs_house = detect_house_y(house_model_path, image.copy(), min_conf = house_conf, req_label = house_lbl )

        # Make bounding box for house and write output to txt file
        cv2.rectangle(image, (bb_house[1], bb_house[2]), (bb_house[3], bb_house[4]), (255, 0, 0), 2)
        cv2.putText(image, ('%s: %d%%' % ('house', int(bb_house[0]*100))), (bb_house[1] + 5, bb_house[2] + 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
        f.write(f'house {bb_house[0]} {int(bb_house[1]*imgW/640)} {int(bb_house[2]*imgH/640)} {int(bb_house[3]*imgW/640)} {int(bb_house[4]*imgH/640)}\n')

        # Detect doors and crop accordingly
        bbs_doors, cropped_doors, cfacs_doors = detect_door_y(door_model_path, cropped_house.copy(), min_conf = door_conf, req_label = door_lbl)

        # Loop through all the doors and find staircase and steps for each
        for bb_door, cropped_door, cfacs_door in zip(bbs_doors, cropped_doors, cfacs_doors):
            bb_staircase, staircase_cropped, cfacs_stair = detect_staircase_y(stair_model_path, cropped_door.copy(), min_conf = stair_conf, req_label = stair_lbl)

            #  Convert cropped coordinates of door and staircase to the full image
            xmin_door  = cfacs_house[0] + bb_door[1]
            ymin_door  = cfacs_house[2] + bb_door[2]

            xmax_door  = cfacs_house[0] + bb_door[3]
            ymax_door  = cfacs_house[2] + bb_door[4]

            xmin_stair = cfacs_house[0] + (cfacs_door[0] + bb_staircase[1])
            ymin_stair = cfacs_house[2] + (cfacs_door[2] + bb_staircase[2])

            xmax_stair = cfacs_house[0] + (cfacs_door[0] + bb_staircase[3])
            ymax_stair = cfacs_house[2] + (cfacs_door[2] + bb_staircase[4])

            # write door bounding box to tct file
            f.write(f'door {bb_door[0]} {int(xmin_door*imgW/640)} {int(ymin_door*imgH/640)} {int(xmax_door*imgW/640)} {int(ymax_door*imgH/640)}\n')

            # put bounding box for door
            cv2.rectangle(image, (xmin_door, ymin_door), (xmax_door, ymax_door), (255, 0, 0), 2)
            cv2.putText(image, ('%s: %d%%' % ('door', int(bb_door[0]*100))), (xmin_door + 5, ymin_door + 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

            # Find steps from staircropped image (sc), doorcropped image (dc), and housecropped image (hc)
            sc_bbs_steps, sc_num_of_steps = detect_steps_y(step_model_path, staircase_cropped.copy(), min_conf = step_conf, req_label = step_lbl, overlap = max_iou_step)
            dc_bbs_steps, dc_num_of_steps = detect_steps_y(step_model_path, cropped_door.copy(), min_conf = step_conf, req_label = step_lbl, overlap = max_iou_step)
            hc_bbs_steps, hc_num_of_steps = detect_steps_y(step_model_path, cropped_house.copy(), min_conf = step_conf, req_label = step_lbl, overlap = max_iou_step)

            sc_step_count = 0
            dc_step_count = 0
            hc_step_count = 0

            # For each crop (sc, dc, and hc) see how many steps are within staircase
            if sc_num_of_steps > 0:
                for bb_step in sc_bbs_steps:
                    xmin_step = cfacs_house[0] + cfacs_door[0] + cfacs_stair[0] + bb_step[1]
                    ymin_step = cfacs_house[2] + cfacs_door[2] + cfacs_stair[2] + bb_step[2]

                    xmax_step = cfacs_house[0] + cfacs_door[0] + cfacs_stair[0] + bb_step[3]
                    ymax_step = cfacs_house[2] + cfacs_door[2] + cfacs_stair[2] + bb_step[4]

                    xmid_step = (xmin_step + xmax_step)/2
                    ymid_step = (ymin_step + ymax_step)/2

                    if xmid_step > xmin_stair and xmid_step < xmax_stair and ymid_step > ymin_stair and ymid_step < ymax_stair:
                        sc_step_count+=1
```

```python

        if dc_num_of_steps > 0:
            for bb_step in dc_bbs_steps:
                xmin_step = cfacs_house[0] + cfacs_door[0] + bb_step[1]
                ymin_step = cfacs_house[2] + cfacs_door[2] + bb_step[2]

                xmax_step = cfacs_house[0] + cfacs_door[0] + bb_step[3]
                ymax_step = cfacs_house[2] + cfacs_door[2] + bb_step[4]

                xmid_step = (xmin_step + xmax_step)/2
                ymid_step = (ymin_step + ymax_step)/2

                if xmid_step > xmin_stair and xmid_step < xmax_stair and ymid_step > ymin_stair and ymid_step < ymax_stair:
                    dc_step_count+=1

        if hc_num_of_steps > 0:
            for bb_step in hc_bbs_steps:
                xmin_step = cfacs_house[0] + bb_step[1]
                ymin_step = cfacs_house[2] + bb_step[2]

                xmax_step = cfacs_house[0] + bb_step[3]
                ymax_step = cfacs_house[2] + bb_step[4]

                xmid_step = (xmin_step + xmax_step)/2
                ymid_step = (ymin_step + ymax_step)/2

                if xmid_step > xmin_stair and xmid_step < xmax_stair and ymid_step > ymin_stair and ymid_step < ymax_stair:
                    hc_step_count+=1

        # Use the version that detects the most steps within the staircase, because of the IoU threshold, this is likely the one with that sees all the steps
        if hc_step_count > dc_step_count and hc_step_count > sc_step_count:
            step_set = hc_bbs_steps
            x_offset = cfacs_house[0]
            y_offset = cfacs_house[2]
        elif dc_step_count > sc_step_count:
            step_set = dc_bbs_steps
            x_offset = cfacs_house[0] + cfacs_door [0]
            y_offset = cfacs_house[2] + cfacs_door [2]
        else:
            step_set = sc_bbs_steps
            x_offset = cfacs_house[0] + cfacs_door [0] + cfacs_stair[0]
            y_offset = cfacs_house[2] + cfacs_door [2] + cfacs_stair[2]

        # Draw bounding box for used amount of steps
        steps = []
        step_count = 0
        for bb_step in step_set:
            xmin_step = x_offset + bb_step[1]
            ymin_step = y_offset + bb_step[2]

            xmax_step = x_offset + bb_step[3]
            ymax_step = y_offset + bb_step[4]

            xmid_step = (xmin_step + xmax_step)/2
            ymid_step = (ymin_step + ymax_step)/2

            # Plot steps within staircase and write output to txt-file
            if xmid_step > xmin_stair and xmid_step < xmax_stair and ymid_step > ymin_stair and ymid_step < ymax_stair:
                steps.append([bb_step[0], xmin_step, ymin_step, xmax_step, ymax_step])
                step_count+=1

        staircase = [bb_staircase[0],xmin_stair, ymin_stair, xmax_stair, ymax_stair]
        # See if staircase is shorter than all others detected upto now
        if step_count < least_num_steps:
            least_num_steps = step_count
            steps_best_staircase = steps
            best_staircase = staircase


    # Select and display shortest staircase
    total_steps = 0
    if best_staircase[0] > 0:
        xmin_stair = best_staircase[1]
        ymin_stair = best_staircase[2]
        xmax_stair = best_staircase[3]
        ymax_stair = best_staircase[4]

        # Put bounding box for staircase and write to txt-file
        cv2.rectangle(image, (xmin_stair, ymin_stair), (xmax_stair, ymax_stair), (255, 0, 0), 2)
        cv2.putText(image, ('%s: %d%%' % ('staircase', int(bb_staircase[0]*100))), (xmin_stair + 5, ymin_stair + 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
        f.write(f'staircase {best_staircase[0]} {int(xmin_stair*imgW/640)} {int(ymin_stair*imgH/640)} {int(xmax_stair*imgW/640)} {int(ymax_stair*imgH/640)}\n')

        for bb_step in steps_best_staircase:
            xmin_step = bb_step[1]
            ymin_step = bb_step[2]
            xmax_step = bb_step[3]
            ymax_step = bb_step[4]

            # Plot steps within staircase and write output to txt-file
            cv2.rectangle(image, (xmin_step, ymin_step), (xmax_step, ymax_step)
, (255, 255, 255), 2)
                cv2.putText(image, ('%s: %d%%' % ('step', int(bb_step[0]*100))), (xmin_step + 5, ymin_step + 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
                f.write(f'step {bb_step[0]} {int(xmin_step*imgW/640)} {int(ymin_step*imgH/640)} {int(xmax_step*imgW/640)} {int(ymax_step*imgH/640)}\n')
                total_steps+=1


    # Give output in terminal
    print(f'\n\n\n The least amount of steps leading up to a door is: {total_steps}\n\n\n')

    # Optional: Display annotated input image
    if image_output:
        cv2.imshow('Annotated output', cv2.cvtColor(image, cv2.COLOR_RGB2BGR))
        cv2.waitKey()
```

C.2 Deploying house-model

```
import cv2
import numpy as np
import tensorflow as tf
from ultralytics import YOLO


def detect_house_y(modelpath, image, min_conf=0.5, req_label = 0, verbose = 0):
    model =  YOLO(modelpath)
    results = model.predict(image)
    xmid_dev_best = .5
    size = image.shape

    imgH = size[0]
    imgW = size[1]

    xmin = 0
    xmin_crop = 0
    xmax = imgH
    xmax_crop = imgW
    ymin = 0
    ymin_crop = 0
    ymax = imgH
    ymax_crop = imgH
    detections = [0, xmin, ymin, xmax, ymax]

    image_cropped = image.copy()

    for result in results:
        for i in range(len(result.boxes.conf)):
```

```python
            label = int(result.boxes.cls[i])
            conf = float(result.boxes.conf[i])
            if ((conf > min_conf) and (conf <= 1.0)) and label == req_label:
                xmin = int(result.boxes.xyxy[i][0])
                ymin = int(result.boxes.xyxy[i][1])
                xmax = int(result.boxes.xyxy[i][2])
                ymax = int(result.boxes.xyxy[i][3])

                xmid_rel = (xmax-xmin)/imgW
                xmid_dev = abs(xmid_rel-.5)

                if xmid_dev < xmid_dev_best:
                    xmid_dev_best = xmid_dev
                    detections=[conf, xmin, ymin, xmax, ymax, xmid_dev]

                    ymin_crop = max(detections[2],0)
                    ymax_crop = imgH

                    xmin_crop = int(max(detections[1] - .1 * imgW, 0))
                    xmax_crop = int(min(detections[3] + .1 * imgH, imgH))

    image_cropped = image[ymin_crop:ymax_crop, xmin_crop:xmax_crop].copy()
    cv2.rectangle(image, (xmin, ymin), (xmax, ymax), (255, 0 ,0), 1)
    cv2.rectangle(image, (xmin_crop, ymin_crop), (xmax_crop, ymax_crop), (0,255,0),
     2)
    cv2.imshow('house', cv2.cvtColor(image.copy(), cv2.COLOR_RGB2BGR))
    cv2.waitKey()

    cropping_factors = [xmin_crop, (xmax_crop - xmin_crop)/imgW, ymin_crop, (
    ymax_crop - ymin_crop)/imgH]

    return image_cropped, detections, cropping_factors
```

C.3 Deploying door-model

```python
import cv2
import numpy as np
from ultralytics import YOLO

def detect_door_y(modelpath, image, min_conf=0.5, req_label=0, verbose = 0):
    model =  YOLO(modelpath)
    results = model.predict(image)

    detected_doors = []
    cropped_images = []

    size = image.shape

    imgH = size[0]
    imgW = size[1]

    xmin = 0
    xmin_crop = 0
    xmax = imgW
    xmax_crop = imgW
    ymin = 0
    ymin_crop = 0
    ymax = imgH
    ymax_crop = imgH
    detected_doors = []
    cropping_factors = []

    image_cropped = image.copy()

    for result in results:
        for i in range(len(result.boxes.conf)):
            label = int(result.boxes.cls[i])
            conf = float(result.boxes.conf[i])
            if ((conf > min_conf) and (conf <= 1.0)) and label == req_label:
                xmin = int(result.boxes.xyxy[i][0])
                ymin = int(result.boxes.xyxy[i][1])
                xmax = int(result.boxes.xyxy[i][2])
                ymax = int(result.boxes.xyxy[i][3])

                ymin_crop = int((ymin+ymax)/2)
                ymax_crop = imgH

                xmin_crop = 0
                xmax_crop = imgW

                image_cropped = image[ymin_crop:ymax_crop, xmin_crop:xmax_crop].
copy()
                detected_doors.append([conf, xmin, ymin, xmax, ymax])
                cropped_images.append(image_cropped)

                cv2.rectangle(image, (xmin, ymin), (xmax, ymax), (255, 0 ,0), 1)
                cv2.rectangle(image, (xmin_crop, ymin_crop), (xmax_crop, ymax_crop)
, (0,255,0), 2)
                cropping_factors.append([xmin_crop, (xmax_crop - xmin_crop)/imgW,
ymin_crop, (ymax_crop - ymin_crop)/imgH])

    cv2.imshow('doors', cv2.cvtColor(image.copy(), cv2.COLOR_RGB2BGR))
    cv2.waitKey(0)


    return detected_doors, cropped_images, cropping_factors
```

C.4 Deploying staircase-model

```python
import cv2
import numpy as np
from ultralytics import YOLO
def detect_staircase_y(modelpath, image, min_conf=0.5, req_label=0, verbose = 0):
    model =  YOLO(modelpath)
    results = model.predict(image)

    size = image.shape

    imgH = size[0]
    imgW = size[1]

    xmin = 0
    xmin_crop = 0
    xmax = imgW
    xmax_crop = imgW
    ymin = 0
    ymin_crop = 0
    ymax = imgH
    ymax_crop = imgH
    staircase_height_best = imgH

    detections = [0, xmin, ymin, xmax, ymax]

    for result in results:
        for i in range(len(result.boxes.conf)):
            label = int(result.boxes.cls[i])
            conf = float(result.boxes.conf[i])
            if ((conf > min_conf) and (conf <= 1.0)) and label == req_label:
                xmin = int(result.boxes.xyxy[i][0])
                ymin = int(result.boxes.xyxy[i][1])
                xmax = int(result.boxes.xyxy[i][2])
                ymax = int(result.boxes.xyxy[i][3])

                staircase_height = ymax-ymin

                if staircase_height < staircase_height_best:
                    staircase_height_best = staircase_height
                    detections=[conf, xmin, ymin, xmax, ymax]

    ymin_crop = int(max(detections[2] - .1 * imgH, 0))
    ymax_crop = int(min(detections[4] + .1 * imgH, imgH))

    xmin_crop = int(max(detections[1] - .05 * imgW, 0))
    xmax_crop = int(min(detections[3] + .05 * imgW, imgW))


    image_cropped = image[ymin_crop:ymax_crop, xmin_crop:xmax_crop].copy()

    cv2.rectangle(image, (detections[1], detections[2]), (detections[3], detections
    [4]), (255, 0 ,0), 1)
    cv2.rectangle(image, (xmin_crop, ymin_crop), (xmax_crop, ymax_crop), (0,255,0),
     2)
    cv2.imshow('staircase', cv2.cvtColor(image, cv2.COLOR_RGB2BGR))
    cv2.waitKey()

    cropping_factors = [xmin_crop, (xmax_crop - xmin_crop)/imgW, ymin_crop, (
    ymax_crop - ymin_crop)/imgH]
    return detections, image_cropped, cropping_factors
```

C.5 Deploying step-model

```
import cv2
import numpy as np
from ultralytics import YOLO

def detect_steps_y(modelpath, image, min_conf=0.5, req_label=0, overlap = 0.5,
    verbose = False):
    model =  YOLO(modelpath)
    results = model.predict(image, conf = min_conf, iou = overlap, show = False)

    detected_steps = []

    for result in results:
        for i in range(len(result.boxes.conf)):
            label = int(result.boxes.cls[i])
            conf = float(result.boxes.conf[i])
            # print(conf)
            if ((conf > min_conf) and (conf <= 1.0)) and label == req_label:
                xmin = int(result.boxes.xyxy[i][0])
                ymin = int(result.boxes.xyxy[i][1])
                xmax = int(result.boxes.xyxy[i][2])
                ymax = int(result.boxes.xyxy[i][3])

                detected_steps.append([conf, xmin, ymin, xmax, ymax])
                cv2.rectangle(image, (xmin, ymin), (xmax, ymax), (255, 0 ,0), 1)

    cv2.imshow('step', cv2.cvtColor(image.copy(), cv2.COLOR_RGB2BGR))
    cv2.waitKey()


    return detected_steps, len(detected_steps)
```

Appendix D

Other Code Used

D.1 Splitting Data

```python
import os
import shutil
import xml.etree.ElementTree as ET

# Set the source folder containing the images and XML files
source_folder = "Team1TrainingSet/train"

# Set the destination folders for each label
destination_folders = {
    "house": "splitting data/house",
    "door": "splitting data/door",
    "staircase": "splitting data/staircase",
    "step": "splitting data/step",
    "garagedoor": "splitting data/garagedoor"
}

# Iterate over all files in the source folder
for filename in os.listdir(source_folder):
    if filename.endswith(".xml"):
        # Parse the XML file
        xml_file = os.path.join(source_folder, filename)
        tree = ET.parse(xml_file)
        root = tree.getroot()

        # Extract the labels from the XML file
        labels = [elem.text for elem in root.findall(".//name")]

        # Set to store the unique labels found in the XML file
        unique_labels = set()

        # Check if any of the labels match the desired labels
        for label in labels:
```

```
        if label in destination_folders:
            unique_labels.add(label)

    # Copy the XML file and corresponding image file to the respective folders
    image_file = os.path.join(source_folder, filename.replace(".xml", ".jpg"))
    for label in unique_labels:
        if os.path.isfile(image_file):
            # Copy the XML file
            shutil.copy(xml_file, destination_folders[label])

            # Copy the image file
            shutil.copy(image_file, destination_folders[label])
```

D.2 Code for Development of Initial design

```
# Mount drive
from google.colab import drive
drive.mount('/content/drive')

# Copy required files into correct directory
!cp -r /content/drive/MyDrive/Data /content/images
```

```
# Clone the tensorflow models repository from GitHub
!git clone --depth 1 https://github.com/tensorflow/models
```

```
# Copy setup files into models/research folder
%%bash
cd models/research/
protoc object_detection/protos/*.proto --python_out=.
cp object_detection/packages/tf2/setup.py .
```

```
# Install the TensorFlow Object Detection API
!pip install /content/models/research/
```

```
# This create labelmap.txt file to store all classes the model will be trained on
cat <<EOF >> /content/labelmap.txt
house
door
staircase
step
EOF
```

```
# Download csv to tfrecord conversion scripts
! wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/master/util_scripts/create_tfrecord.py
```

```
import os
import glob
import pandas as pd
```

```python
import xml.etree.ElementTree as ET

# Convert information in .xml-files to information in .csv files which will be converted into TFRecords
def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for obj in root.findall('object'):
            bndbox = obj.find('bndbox')
            value = (root.find('filename').text,
                     obj.find('name').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     obj[0].text,
                     int(bndbox.find('xmin').text),
                     int(bndbox.find('ymin').text),
                     int(bndbox.find('xmax').text),
                     int(bndbox.find('ymax').text),
                    )
            xml_list.append(value)
    column_name = ['filename', 'class', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df

def create_csvs():
    for folder in ['train','validation']:
        image_path = os.path.join(os.getcwd(), ('images/' + folder))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv(('images/' + folder + '_labels.csv'), index=None)
        print('Successfully converted xml to csv.')
```

```python
# Create CSV data files and TFRecord files
create_csvs()
!python3 create_tfrecord.py --csv_input=images/train_labels.csv --labelmap=labelmap.txt --image_dir=images/train --output_path=train.tfrecord
!python3 create_tfrecord.py --csv_input=images/validation_labels.csv --labelmap=labelmap.txt --image_dir=images/validation --output_path=val.tfrecord
```

```python
train_record_fname = '/content/train.tfrecord'
val_record_fname = '/content/val.tfrecord'
label_map_pbtxt_fname = '/content/labelmap.pbtxt'
```

```python
# Select model and configuration
chosen_model = 'ssd-mobilenet-v2-fpnlite-320'

MODELS_CONFIG = {
    'ssd-mobilenet-v2-fpnlite-320': {
        'model_name': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8',
        'base_pipeline_file': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz',
    },
     'faster_rcnn_resnet101_v1_640': {
        'model_name': 'faster_rcnn_resnet101_v1_640x640_coco17_tpu-8',
        'base_pipeline_file': 'faster_rcnn_resnet101_v1_640x640_coco17_tpu-8.config',
        'pretrained_checkpoint': 'faster_rcnn_resnet101_v1_640x640_coco17_tpu-8.tar.gz',
    }
}

model_name = MODELS_CONFIG[chosen_model]['model_name']
pretrained_checkpoint = MODELS_CONFIG[chosen_model]['pretrained_checkpoint']
base_pipeline_file = MODELS_CONFIG[chosen_model]['base_pipeline_file']
```

```
    # Create "mymodel" folder for holding pre-trained weights and configuration files
    %mkdir /content/models/mymodel/
    %cd /content/models/mymodel/

    # Download pre-trained model weights
    import tarfile
    download_tar = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/' + pretrained_checkpoint
    !wget {download_tar}
    tar = tarfile.open(pretrained_checkpoint)
    tar.extractall()
    tar.close()

    # Download training configuration file for model
    download_config = 'https://raw.githubusercontent.com/tensorflow/models/master/research/object_detection/configs/tf2/' + base_pipeline_file
    !wget {download_config}
```

```
    # Set training parameters for the model
    num_steps = 100000
    batch_size = 16
```

```
    # Set file locations and get number of classes for config file
pipeline_fname = '/content/models/mymodel/' + base_pipeline_file
fine_tune_checkpoint = '/content/models/mymodel/' + model_name + '/checkpoint/ckpt-0'

def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())

num_classes = get_num_classes(label_map_pbtxt_fname)
```

```
    # Set the path to the custom config file and the directory to store training checkpoints in
    pipeline_file = '/content/models/mymodel/pipeline_file.config'
    model_dir = '/content/training/'
```

```
    # Load tensorboard to keep up with training progress
    %load_ext tensorboard
    %tensorboard --logdir '/content/training/train'
```

```
    # Train model
    !python /content/models/research/object_detection/model_main_tf2.py \
        --pipeline_config_path={pipeline_file} \
        --model_dir={model_dir} \
        --alsologtostderr \
        --num_train_steps={num_steps} \
        --sample_1_of_n_eval_examples=1
```

```
    # Convert model into TFLite model file
    import tensorflow as tf
    # Path to training directory (the conversion script automatically chooses the highest checkpoint file)
    last_model_path = '/content/training'

    !python /content/models/research/object_detection/export_tflite_graph_tf2.py \
        --trained_checkpoint_dir {last_model_path} \
        --output_directory {output_directory} \
        --pipeline_config_path {pipeline_file}


    converter = tf.lite.TFLiteConverter.from_saved_model('/content/custom_model_lite/saved_model')
    tflite_model = converter.convert()

    with open('/content/custom_model_lite/saved_model.tflite', 'wb') as f:
        f.write(tflite_model)
```

D.3 Code for Cropping Training Data

```
    %%bash
    mkdir /content/data/

    mkdir /content/data/train/
    mkdir /content/data/train/images/
    mkdir /content/data/train/labels/
    mkdir /content/data/validation/
    mkdir /content/data/validation/images/
    mkdir /content/data/validation/labels/

    mkdir /content/data/test/
    mkdir /content/data/test/images/
    mkdir /content/data/test/labels/
```

```python
    import os
import xml.etree.ElementTree as ET

def create_yolo_files(xml_file, output_dir, training_target, input_dir):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    file_name = os.path.splitext(os.path.basename(xml_file))[0]
    image_path = input_dir + '/' + file_name + '.jpg'
    image = cv.imread(image_path)

    if image is not None:
        if training_target == 'house':  # No cropping for houses
            image_written = 0
            txt_output_file = os.path.join(output_dir + '/labels' , file_name + '.txt')
            img_output_file = os.path.join(output_dir + '/images', file_name + '.png')
            with open(txt_output_file, 'w') as f:
                for obj in root.findall('object'):
                    size = root.find('size')
                    imgW = int(size.find('width').text)
                    imgH = int(size.find('height').text)

                    label = obj.find('name').text

                    if label == 'house':
                        bndbox = obj.find('bndbox')
                        xmin = int(bndbox.find('xmin').text)
                        ymin = int(bndbox.find('ymin').text)
                        xmax = int(bndbox.find('xmax').text)
                        ymax = int(bndbox.find('ymax').text)

                        training_image = cv.resize(image, (640, 640))

                        if image_written == 0:
                            cv.imwrite(img_output_file, training_image)
                            image_written = 1

                            f.write(f'0 {(xmin+xmax)/(2*imgW)} {(ymin+ymax)/(2*imgH)} {(xmax-xmin)/imgW} {(ymax-ymin)/imgH}')

        elif training_target == 'door': # First find house, crop to the house and then all doors with a bounding box within the house, make a new image for every house
            num_of_pics = 1
            for obj in root.findall('object'):
                size = root.find('size')
                imgW = int(size.find('width').text)
                imgH = int(size.find('height').text)
                label = obj.find('name').text

                if label == 'house':
                    image_written = 0
                    txt_output_file = os.path.join(output_dir + '/labels' , file_name + '___' + str(num_of_pics) + '.txt')
                    img_output_file = os.path.join(output_dir + '/images', file_name + '___' + str(num_of_pics) + '.png')

                    bndbox = obj.find('bndbox')

                    xminCrop = max(int(int(bndbox.find('xmin').text) - 0.1 * imgW), 0)
                    yminCrop = max(int(int(bndbox.find('ymin').text)), 0)
                    xmaxCrop = min(int(int(bndbox.find('xmax').text) + 0.1 * imgW), imgW)
                    ymaxCrop = imgH

                    training_image = cv.resize(image[yminCrop:ymaxCrop, xminCrop:xmaxCrop], (640, 640))

                    cropW = xmaxCrop - xminCrop
                    cropH = ymaxCrop - yminCrop

                    with open(txt_output_file, 'w') as f:
                        for obj2 in root.findall('object'):
                            label2 = obj2.find('name').text
                            if label2 == 'door':
                                bndbox2 = obj2.find('bndbox')
                                xmin = int(bndbox2.find('xmin').text)
                                ymin = int(bndbox2.find('ymin').text)
                                xmax = int(bndbox2.find('xmax').text)
                                ymax = int(bndbox2.find('ymax').text)

                                if xmin >= xminCrop and xmax <= xmaxCrop and ymin >= yminCrop and ymax <= ymaxCrop:
                                    f.write(f'0 {(((xmin + xmax)/2) - xminCrop)/cropW} {(((ymin + ymax)/2) - yminCrop)/cropH} {(xmax - xmin)/cropW} {(ymax - ymin)/cropH}\n')

                                    if image_written == 0:
                                        cv.imwrite(img_output_file, training_image)
                                        image_written = 1
```

```
                                        num_of_pics += 1

    elif training_target == 'staircase': # First find house, crop to the house and then all doors with a bounding box within a house, crop to all doors and the annotate all stairs that are within this image, make a new image for every door
        num_of_pics = 1
        for obj in root.findall('object'):
            size = root.find('size')
            imgW = int(size.find('width').text)
            imgH = int(size.find('height').text)
            label = obj.find('name').text

            if label == 'house':
                bndbox = obj.find('bndbox')
                xminCrop = max(int(int(bndbox.find('xmin').text) - 0.1 * imgW), 0)
                yminCrop = max(int(int(bndbox.find('ymin').text)), 0)
                xmaxCrop = min(int(int(bndbox.find('xmax').text) + 0.1 * imgW), imgW)
                ymaxCrop = imgH

                for obj2 in root.findall('object'):
                    label2 = obj2.find('name').text

                    if label2 == 'door':
                        bndbox2 = obj2.find('bndbox')
                        xmin = int(bndbox2.find('xmin').text)
                        ymin = int(bndbox2.find('ymin').text)
                        xmax = int(bndbox2.find('xmax').text)
                        ymax = int(bndbox2.find('ymax').text)

                        if xmin >= xminCrop and xmax <= xmaxCrop and ymin >= yminCrop and ymax <= ymaxCrop:
                            image_written = 0
                            txt_output_file = os.path.join(output_dir + '/labels' , file_name + '___' + str(num_of_pics) + '.txt')
                            img_output_file = os.path.join(output_dir + '/images', file_name + '___' + str(num_of_pics) + '.png')

                            yminCrop = int(int(bndbox2.find('ymin').text)/2 + int(bndbox2.find('ymax').text)/2)

                            training_image = cv.resize(image[yminCrop:ymaxCrop, xminCrop:xmaxCrop], (640, 640))
                            cropW = xmaxCrop - xminCrop
                            cropH = ymaxCrop - yminCrop

                            with open(txt_output_file, 'w') as f:
                                for obj3 in root.findall('object'):
                                    label3 = obj3.find('name').text
```

```python
                                    if label3 == 'staircase':
                                        bndbox3 = obj3.find('bndbox')
                                            xmin = int(bndbox3.find('xmin').text)
                                            ymin = int(bndbox3.find('ymin').text)
                                            xmax = int(bndbox3.find('xmax').text)
                                            ymax = int(bndbox3.find('ymax').text)

                                            if xmin >= xminCrop and xmax <= xmaxCrop and ymin >= yminCrop and ymax <= ymaxCrop:
                                                f.write(f'0 {(((xmin + xmax)/2) - xminCrop)/cropW} {(((ymin + ymax)/2) - yminCrop)/cropH} {(xmax - xmin)/cropW} {(ymax - ymin)/cropH}\n')

                                                    if image_written == 0:
                                                        cv.imwrite(img_output_file, training_image)
                                                        image_written = 1
                                                        num_of_pics += 1

    elif training_target == 'step': # First find staircase, crop to staircase and find all steps within staircase, make a new image for every staircase
        num_of_pics = 1
        for obj in root.findall('object'):
            size = root.find('size')
            imgW = int(size.find('width').text)
            imgH = int(size.find('height').text)
            label = obj.find('name').text
            if label == 'staircase':
                bndbox3 = obj.find('bndbox')
                xmin = int(bndbox.find('xmin').text)
                ymin = int(bndbox.find('ymin').text)
                xmax = int(bndbox.find('xmax').text)
                ymax = int(bndbox.find('ymax').text)

                if xmin >= xminCrop and xmax <= xmaxCrop and ymin >= yminCrop and ymax <= ymaxCrop:
                    image_written = 0
                    txt_output_file = os.path.join(output_dir + '/labels' , file_name + '___' + str(num_of_pics) + '.txt')
                    img_output_file = os.path.join(output_dir + '/images', file_name + '___' + str(num_of_pics) + '.png')

                    xminCrop = max(int(xmin - 0.05 * imgW), 0)
                    xmaxCrop = min(int(xmax + 0.05 * imgW), imgW)
                    yminCrop = max(int(ymin - 0.05 * imgH), 0)
                    ymaxCrop = min(int(ymax + 0.05 * imgH), imgH)

                    training_image = cv.resize(image[yminCrop:ymaxCrop, xminCrop:xmaxCrop], (640, 640))

                    cropW = xmaxCrop - xminCrop
                    cropH = ymaxCrop - yminCrop

                    cv.imwrite(img_output_file, training_image)

                    with open(txt_output_file, 'w') as f:
                    for obj in root.findall('object'):
                        label = obj.find('name').text
                        if label == 'step':
                        bndbox = obj.find('bndbox')
                            xmin = int(bndbox.find('xmin').text)
                            ymin = int(bndbox.find('ymin').text)
                            xmax = int(bndbox.find('xmax').text)
                            ymax = int(bndbox.find('ymax').text)

                            if xmin >= xminCrop and xmax <= xmaxCrop and ymin >= yminCrop and ymax <= ymaxCrop:
                                f.write(f'0 {(((xmin + xmax)/2) - xminCrop)/cropW} {(((ymin + ymax)/2) - yminCrop)/cropH} {(xmax - xmin)/cropW} {(ymax - ymin)/cropH}\n')

                                if image_written == 0:
                                    cv.imwrite(img_output_file, training_image)
                                    image_written = 1
                                    num_of_pics += 1

for set in ['train', 'validation', 'test']:
    source_directory = '/content/drive/MyDrive/SplittingData2/'+ model_target + '/' + set

    output_directory = '/content/data/' + set

    for filename in os.listdir(source_directory):
        if filename.endswith('.xml'):
            xml_file = os.path.join(source_directory, filename)
            create_yolo_files(xml_file, output_directory, model_target, source_directory)
    print (f'{set} done')
```

D.4 Code for Training YOLO

```
%%bash
pip install ultralytics
```

```
import ultralytics
import cv2 as cv
```

```
from ultralytics import YOLO
```

```
model_target = 'staircase'
```

```
# Mount drive
from google.colab import drive
drive.mount('/content/drive')

yamlpath = model_target + '.yaml'

# Copy required files into correct directory
!cp -r /content/drive/MyDrive/yolov8x.pt /content/yolov8x.pt

!cp -r /content/drive/MyDrive/full.yaml /content/yaml.yaml
```

```
model = YOLO('/content/yolov8x.pt')
model.train(data='/content/yaml.yaml',
        epochs=100,
        imgsz=640,)
```

```
%%bash
cp -r /content/runs/detect/train/weights /content/drive/MyDrive/models/
modeldata
```

D.5 Performance Analysis Code Documentation

D.5.1 IoU

```
import re
def iou(box1_det, box2_gt):
    box1_det=[int(s) for s in re.findall(r'\b\d+\b', box1_det)]
    box2_gt=[int(s) for s in re.findall(r'\b\d+\b', box2_gt)]
    #print('box1_det',box1_det,'box2_gt',box2_gt)
    # Calculate intersection area
    x1 = max(box1_det[2], box2_gt[0])
    y1 = max(box1_det[3], box2_gt[1])
    x2 = min(box1_det[4], box2_gt[2])
    y2 = min(box1_det[5], box2_gt[3])
    intersection = max(0, x2 - x1) * max(0, y2 - y1)

    # Calculate union area
    box1_area = (box1_det[5] - box1_det[3]) * (box1_det[4] - box1_det[2])
    box2_area = (box2_gt[3] - box2_gt[1]) * (box2_gt[2] - box2_gt[0])
    union = box1_area + box2_area - intersection

    return intersection/union
```

D.5.2 Squared error

```python
from math import sqrt
from statistics import mean
def squared_error(box1_det,box2_gt):
  box1_det=[int(s) for s in re.findall(r'\b\d+\b', box1_det)]
  box2_gt=[int(s) for s in re.findall(r'\b\d+\b', box2_gt)]

  centre_box1=[box1_det[3]+(box1_det[5] - box1_det[3])/2, box1_det[2]+(box1_det[4]
    - box1_det[2])/2]
  centre_box2=[box2_gt[1]+(box2_gt[3] - box2_gt[1])/2, box2_gt[2]+(box2_gt[0] -
    box2_gt[2])/2]

  se=sqrt(abs(centre_box1[0]-centre_box2[0])**2 + abs(centre_box1[1]-centre_box2
    [1])**2)

  return se
```

D.5.3 Compute metrics

```python
#savepath = '/content/mAP/input/detection-results/'
import numpy as np
import math

def compute_metrics(print_info, paths_order, savepath, ground_truth_folder,
    conf_threshold,IoU_threshold):
    #metrics=[house,door,step]
    TP=[0,0,0,0]
    FP=[0,0,0,0]
    TN=[0,0,0,0]
    FN=[0,0,0,0]
    precision=[0,0,0,0]
    recall=[0,0,0,0]
    nr_house_gt=0
    nr_door_gt=0
    nr_staircase_gt=0
    nr_step_gt=0
    nr_house_dect=0
    nr_door_dect=0
    nr_staircase_det=0

    se=[]
    mse=float('NaN')
    rmse=float('NaN')
    gt_labels=np.zeros((len(paths_order),4))
    dect_labels=np.zeros((len(paths_order),4))
    nr_steps_im_gt=[]
    nr_steps_im_dect=[]
    i=0
    #For each image
```

```
  for path in paths_order:

    #grab model detections
    nr_step_dect=0
    with open(path) as f:
      detection=f.readlines()

    #translate path to ground-truth path
    k=path.lstrip(savepath)
    z=ground_truth_folder +'/'+ k

    #grab ground truth labels
    nr_steps_im_gt1=0
    with open(z) as f:
      correction=f.readlines()
      #
      for item in correction:
        if item[:5]=='house':
          nr_house_gt+=1
        if item[:4]=='door':
          nr_door_gt+=1
        if item[:9]=='staircase':
          nr_staircase_gt+=1
        if item[:5]=='steps' or item[:4]=='step':
          nr_step_gt+=1
          nr_steps_im_gt1+=1
      nr_steps_im_gt.append(nr_steps_im_gt1)
      gt_labels[i]=[nr_house_gt, nr_door_gt, nr_staircase_gt, nr_step_gt]

    det_nr=0


    #start comparison
    #grab 1 detection
    len(detection)
    for box1 in detection:
      gt_nr=0
      FP_check=TP.copy()

      #extract the confidence of the detection
      box1_entries=[int(s) for s in re.findall(r'\b\d+\b', box1)]

      #if the confidence is 0 skip this detection
      if box1_entries[1] == 0:
        continue

      digits = int(math.log10(box1_entries[1]))+1
      confidence=box1_entries[1]/(10**digits)

      #check if confidence lower than the threshold, if so skip this detection
      if confidence < conf_threshold:
        continue
      #compare 1 detection with all ground truth labels

      for box2 in correction:
        IoU=iou(box1,box2)

        #check Intersection of Union is significant enough
        if IoU > IoU_threshold:
          image_name=path.lstrip(savepath) #grab image name for printing
          #if the labels between detection and ground-truth also match
          if box1[:5]==box2[:5]:
            #compute squared error to be used for mse and rmse
            se.append(squared_error(box1,box2))
            #increment the corresponding metric index
            if box1[:5]=='house':
              TP[0]+=1
            if box1[:4]=='door':
              TP[1]+=1
            if box1[:9]=='staircase':
              TP[2]+=1
            if box1[:5]=='steps' or box1[:4]=='step':
              TP[3]+=1
              nr_step_dect+=1

          elif print_info==True:
              print('labels do not match')
          #print information
          if print_info==True:
            print('image:', image_name)
            print('item1:',box1[:5], 'item2:',box2[:5])
            print('ground-truth object number:',gt_nr,'detection object number',
  det_nr, 'IoU:',IoU,'\n')

        gt_nr+=1

      #check whether TP have changed if not no match was found and it is a FP
      if FP_check==TP:
        if box1[:5]=='house':
          FP[0]+=1
        if box1[:4]=='door':
          FP[1]+=1
        if box1[:9]=='staircase':
          FP[2]+=1
        if box1[:5]=='steps' or box1[:4]=='step':
          FP[3]+=1
      det_nr+=1
    nr_steps_im_dect.append(nr_step_dect)
  #compute false negative. which is all ground truth labels not correctly found
  total_houses=np.sum(gt_labels[:,0])
  total_doors=np.sum(gt_labels[:,1])
  total_staircases=np.sum(gt_labels[:,2])
  total_steps=np.sum(gt_labels[:,3])
  FN=[max(0,total_houses-TP[0]),max(0,total_doors-TP[1]),max(0,total_staircases-
  TP[2]),max(0,total_steps-TP[3])]

  #compute rmse and mse if detections were made
  if len(se) != 0:
    mse=mean(se)
    rmse=sqrt(mse)

  #compute precision and recall
  if TP[0] != 0:
    recall[0]=TP[0]/(TP[0]+FN[0])
    precision[0]=TP[0]/(TP[0]+FP[0])
  else:
    recall[0]=0
    precision[0]=0
  if TP[1] != 0:
    recall[1]=TP[1]/(TP[1]+FN[1])
    precision[1]=TP[1]/(TP[1]+FP[1])
  else:
    recall[1]=0
    precision[1]=0
  if TP[2] != 0:
    recall[2]=TP[2]/(TP[2]+FN[2])
    precision[2]=TP[2]/(TP[2]+FP[2])
  else:
    recall[2]=0
    precision[2]=0
  if TP[3] != 0:
    recall[3]=TP[3]/(TP[3]+FN[3])
    precision[3]=TP[3]/(TP[3]+FP[3])
  else:
    recall[3]=0
    precision[3]=0


  return [TP,FP,TN,FN],nr_steps_im_dect,nr_steps_im_gt,mse,rmse,precision,recall
```

D.5.4 Plot confusion matrices

```
#.25
conf_threshold=0.25
IoU_threshold=0.1
[conf,steps_det,steps_gt,mse,rmse,precision,recall]=compute_metrics(print_info,
    paths_order, savepath, ground_truth_folder,conf_threshold,IoU_threshold)

# CM House .25
cm_house25 = np.array(([conf[0][0],conf[3][0]],[conf[1][0],conf[2][0]]))
cm_house25 = cm_house25 / np.sum(cm_house25)

```

```
# CM Door .25
cm_door25 = np.array(([conf[0][1],conf[3][1]],[conf[1][1],conf[2][1]]))
cm_door25 = cm_door25 / np.sum(cm_door25)

# CM Staircase .25
cm_strc25 = np.array(([conf[0][2],conf[3][2]],[conf[1][2],conf[2][2]]))
cm_strc25 = cm_strc25/ np.sum(cm_strc25)

# CM Step .25
cm_step25 = np.array(([conf[0][3],conf[3][3]],[conf[1][3],conf[2][3]]))
cm_step25 = cm_step25 / np.sum(cm_step25)

#.5
conf_threshold=0.5
IoU_threshold=0.1
[conf,steps_det,steps_gt,mse,rmse,precision,recall]=compute_metrics(print_info,
    paths_order, savepath, ground_truth_folder,conf_threshold,IoU_threshold)
# CM House .5
cm_house5 = np.array(([conf[0][0],conf[3][0]],[conf[1][0],conf[2][0]]))
cm_house5 = cm_house5 / np.sum(cm_house5)

# CM Door .5
cm_door5 = np.array(([conf[0][1],conf[3][1]],[conf[1][1],conf[2][1]]))
cm_door5 = cm_door5 / np.sum(cm_door5)

# CM Staircase .5
cm_strc5 = np.array(([conf[0][2],conf[3][2]],[conf[1][2],conf[2][2]]))
cm_strc5 = cm_strc5 / np.sum(cm_strc5)

# CM Step .5
cm_step5 = np.array(([conf[0][3],conf[3][3]],[conf[1][3],conf[2][3]]))
cm_step5 = cm_step5 / np.sum(cm_step5)

# .75
conf_threshold=0.75
IoU_threshold=0.1
[conf,steps_det,steps_gt,mse,rmse,precision,recall]=compute_metrics(print_info,
    paths_order, savepath, ground_truth_folder,conf_threshold,IoU_threshold)
# CM House .75
cm_house75 = np.array(([conf[0][0],conf[3][0]],[conf[1][0],conf[2][0]]))
cm_house75 = cm_house75 / np.sum(cm_house75)

# CM Door .75
cm_door75 = np.array(([conf[0][1],conf[3][1]],[conf[1][1],conf[2][1]]))
cm_door75 = cm_door75 / np.sum(cm_door75)

# CM Staircase .75
cm_strc75 = np.array(([conf[0][2],conf[3][2]],[conf[1][2],conf[2][2]]))
cm_strc75 = cm_strc75 / np.sum(cm_strc75)

# CM Step .75
```

```
cm_step75 = np.array(([conf[0][3],conf[3][3]],[conf[1][3],conf[2][3]]))
cm_step75 = cm_step75 / np.sum(cm_step75)






#plot three confusion matrices
fig, ax = plt.subplots(4,3, figsize=(20, 20), dpi=100)

#create chart in each subplot
sns.heatmap(cm_house25, annot=True, fmt=".1%", linewidths=.5, square = True, cmap =
     'YlGn', ax=ax[0][0], cbar=False, annot_kws = {'size':20} )
sns.heatmap(cm_door25, annot=True, fmt=".1%", linewidths=.5, square = True, cmap =
    'YlGn', ax=ax[1][0], cbar=False, annot_kws = {'size':20} )
# sns.heatmap(cm_gdoor25, annot=True, fmt=".1%", linewidths=.5, square = True, cmap
     = 'YlGn', ax=ax[2][0], cbar=False, annot_kws = {'size':20} )
sns.heatmap(cm_strc25, annot=True, fmt=".1%", linewidths=.5, square = True, cmap =
    'YlGn', ax=ax[2][0], cbar=False, annot_kws = {'size':20} )
sns.heatmap(cm_step25, annot=True, fmt=".1%", linewidths=.5, square = True, cmap =
    'YlGn', ax=ax[3][0], cbar=False, annot_kws = {'size':20} )

sns.heatmap(cm_house5, annot=True, fmt=".1%", linewidths=.5, square = True, cmap =
    'YlGn', ax=ax[0][1], cbar=False, annot_kws = {'size':20} )
sns.heatmap(cm_door5, annot=True, fmt=".1%", linewidths=.5, square = True, cmap = '
    YlGn', ax=ax[1][1], cbar=False, annot_kws = {'size':20} )
# sns.heatmap(cm_gdoor5, annot=True, fmt=".1%", linewidths=.5, square = True, cmap
    = 'YlGn', ax=ax[2][1], cbar=False, annot_kws = {'size':20} )
sns.heatmap(cm_strc5, annot=True, fmt=".1%", linewidths=.5, square = True, cmap = '
    YlGn', ax=ax[2][1], cbar=False, annot_kws = {'size':20} )
sns.heatmap(cm_step5, annot=True, fmt=".1%", linewidths=.5, square = True, cmap = '
    YlGn', ax=ax[3][1], cbar=False, annot_kws = {'size':20} )

sns.heatmap(cm_house75, annot=True, fmt=".1%", linewidths=.5, square = True, cmap =
     'YlGn', ax=ax[0][2], cbar=True, annot_kws = {'size':20} )
sns.heatmap(cm_door75, annot=True, fmt=".1%", linewidths=.5, square = True, cmap =
    'YlGn', ax=ax[1][2], cbar=True, annot_kws = {'size':20} )
# sns.heatmap(cm_gdoor75, annot=True, fmt=".1%", linewidths=.5, square = True, cmap
     = 'YlGn', ax=ax[2][2], cbar=True, annot_kws = {'size':20} )
sns.heatmap(cm_strc75, annot=True, fmt=".1%", linewidths=.5, square = True, cmap =
    'YlGn', ax=ax[2][2], cbar=True, annot_kws = {'size':20} )
sns.heatmap(cm_step75, annot=True, fmt=".1%", linewidths=.5, square = True, cmap =
    'YlGn', ax=ax[3][2], cbar=True, annot_kws = {'size':20} )


ax[0][0].set_title('25% confidence threshold')
ax[0][1].set_title('50% confidence threshold')
ax[0][2].set_title('75% confidence threshold')


```

```python
for axis in ax.flat:
    axis.set_xlabel('Detected label', fontsize = 12)
    axis.set_ylabel('Correct label', fontsize=12)
    axis.xaxis.set_ticklabels(['Positives','Negatives'], fontsize=10)
    axis.yaxis.set_ticklabels(['Positives','Negatives'], fontsize=10)

ax[0][0].set_ylabel('Result of the house labelling \n \n Correct label', fontsize
    =12)
ax[1][0].set_ylabel('Result of the door labelling \n \n Correct label', fontsize
    =12)
# ax[2][0].set_ylabel('Result of the garagedoor labelling \n \n Correct label',
    fontsize=12)
ax[2][0].set_ylabel('Result of the staircase labelling \n \n Correct label',
    fontsize=12)
ax[3][0].set_ylabel('Result of the step labelling \n \n Correct label', fontsize
    =12)

plt.show()
```

D.5.5 PR-curve

```python
from sklearn.metrics import auc

def PR_curve_and_AUC(lblpath,PR_pre,PR_rec):
    AUC=[]
    with open(lblpath, 'r') as f:
        labels = [line.strip() for line in f.readlines()]

    #plot precision-recall curve
    fig, axes = plt.subplots(ncols=len(labels), sharex=False,
          sharey=False, figsize=(20, 4))
    for i in range(len(labels)):
      PR_x=[]
      PR_y=[]
      for j in range(len(PR_pre)):
        PR_x.append(PR_rec[j][i])
        PR_y.append(PR_pre[j][i])

      #print(PR_x, PR_y)


      axes[i].set_title('The precision and recall curve of ' + labels[i])
      axes[i].plot(PR_x,PR_y, 'o-')
      axes[i].set(xlabel='Recall', ylabel='Precision')
      axes[i].set_ylim(0,1.1)
      axes[i].set_xlim(0,1)
      AUC.append(auc(PR_x,PR_y))
    plt.show()

    return AUC
```

D.5.6 AUC

```
AUC=PR_curve_and_AUC(lblpath,PR_pre,PR_rec)
print('The area under the curve is %.2f for houses, %.2f for doors, %.2f for
    staircases and %.2f for steps.' % (AUC[0], AUC[1], AUC[2], AUC[3]))
```

D.5.7 mAP

```
#Create the map
%%bash
git clone https://github.com/Cartucho/mAP /content/mAP
cd /content/mAP
rm input/detection-results/*
rm input/ground-truth/*
rm input/images-optional/*
wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-
    Detection-on-Android-and-Raspberry-Pi/master/util_scripts/
    calculate_map_cartucho.py

#Next, we'll copy the images and annotation data from the **test** folder to the
    appropriate folders inside the cloned repository. These will be used as the "
    ground truth data" that our model's detection results will be compared to.

!cp /content/ztest-set/* /content/mAP/input/images-optional # Copy images and xml
    files
!mv /content/mAP/input/images-optional/*.xml /content/mAP/input/ground-truth/  #
    Move xml files to the appropriate folder


#The calculator tool expects annotation data in a format that's different from the
    Pascal VOC .xml file format we're using. Fortunately, it provides an easy
    script, `convert_gt_xml.py`, for converting to the expected .txt format.

!python /content/mAP/scripts/extra/convert_gt_xml.py

### 1.5 Create Labelmap
Finally, we need to create a labelmap for our classes for the detector.

The code section below will create a "labelmap.txt" file that contains a list of
    classes. Replace the `class1`, `class2`, `class3` text with your own classes (
    for example, `penny`, `nickel`, `dime`, `quarter`), adding a new line for each
    class. Then, click play to execute the code.

### This creates a a "labelmap.txt" file with a list of classes the object
    detection model will detect.
%%bash

cat <<EOF >> /content/labelmap.txt
house
door
```

```
staircase
step
EOF

#
We'll use the mAP calculator tool at https://github.com/Cartucho/mAP to determine
    our model's mAP score. First, we need to clone the repository and remove its
    existing example data. We'll also download a script I wrote for interfacing
    with the calculator.

%cd /content/mAP
!python calculate_map_cartucho.py --labels=/content/labelmap.txt
```