Research Internship - Product Development in AI (Image Recognition/Machine Learning)

Prepared by: Sophia Quesada

Date: 28th May, 2023

1. Introduction

This chapter outlines the purpose of the internship and what I was assigned to do during it.

1.1 Purpose

The task assigned to me by PorthouseDean involved designing and developing a user interface which would recognise and highlight any differences in two architectural floor plans: the original floor plan and the proposed floor plan. The program required to be coded in Python. Further, this internship involved the use of computer vision techniques to pre-process and analyse image data, such as feature extraction and object detection.

2. Internship Report

This chapter describes the methodology I took to execute the overall research task and justifies the design choices I made.

2.1 Image Difference

2.1.1 Image Preparation

While looking through the floor plans given to me, I noticed the existence of text in the floor plans. This had to be removed to reduce any noise in the image difference. The method I used to remove the text using OCR (Optical Character Recognition). In order to erase the text, there were three main steps:

1. Using `keras-ocr`, a machine learning Python package, identify text in the image and obtain the bounding box coordinates of each text.
2. For each bounding box, apply a mask to notify the algorithm which part of the image we should inpaint.
3. Apply an inpainting algorithm to inpaint the masked areas, resulting in a text-free image, using cv2.

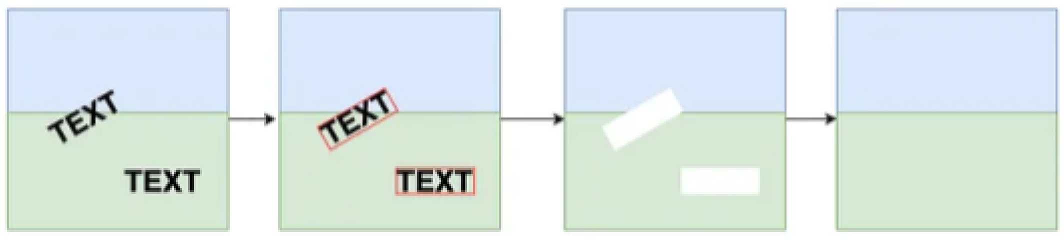

Figure 2.1: Text removal process

Keras-ocr provides pre-trained OCR models which is what was used for this project for the text removal. The final results after all of the image preparation and pre-processing of the data can be seen below.

The blurring of the removed text can be very visible which does affect the highlighted differences in the images. Despite this, I chose not to prioritise fixing this as it was established that all changes detected in my script would be double-checked by a human and a professional.

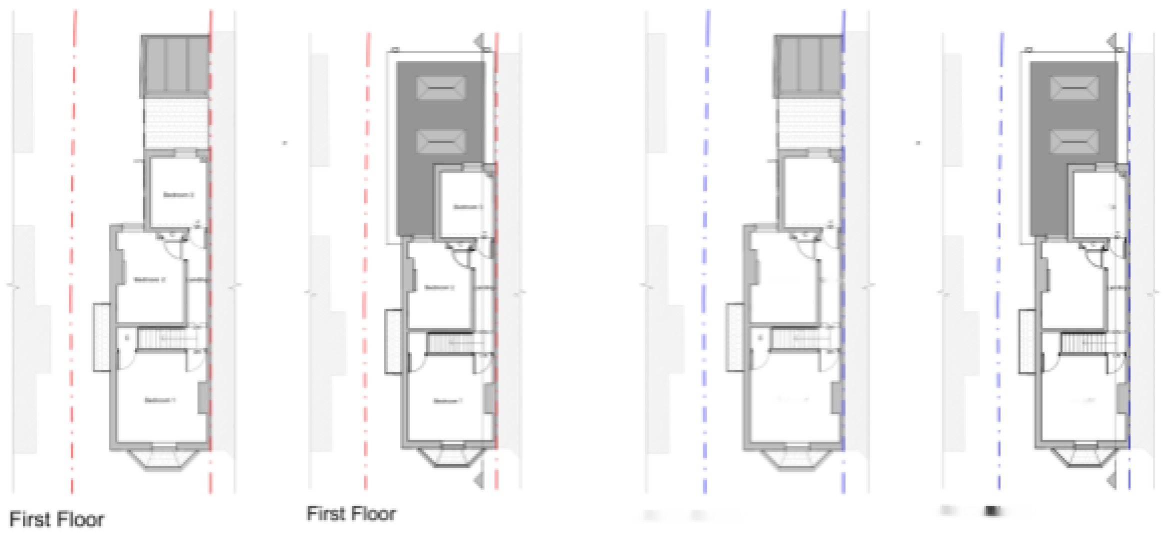

Figure 2.2: Before and after text removal

Before conducting any image difference computations, a few things have to be considered to prevent noise and optimise the result. The two images, before comparison, have to have the same of the following:

- Shape and dimension
- Alignment
- Exposure (Brightness)

Furthermore, both images have to have little to no noise as that would affect any calculations.

To ensure all of these requirements are met, I align both images by feature extraction using the ORB algorithm and then feature matching. The result of this are two images that are aligned like below in Figure 2.3.

It is also seen that I chose to use Structural Similarity Index (SSIM) instead of simpler metrics like Mean Square Error (MSE) or Peak signal-to-noise ratio (PSNR). This is because SSIM takes into account the three points above. MSE and PSNR estimate absolute errors rather than structural information [1].

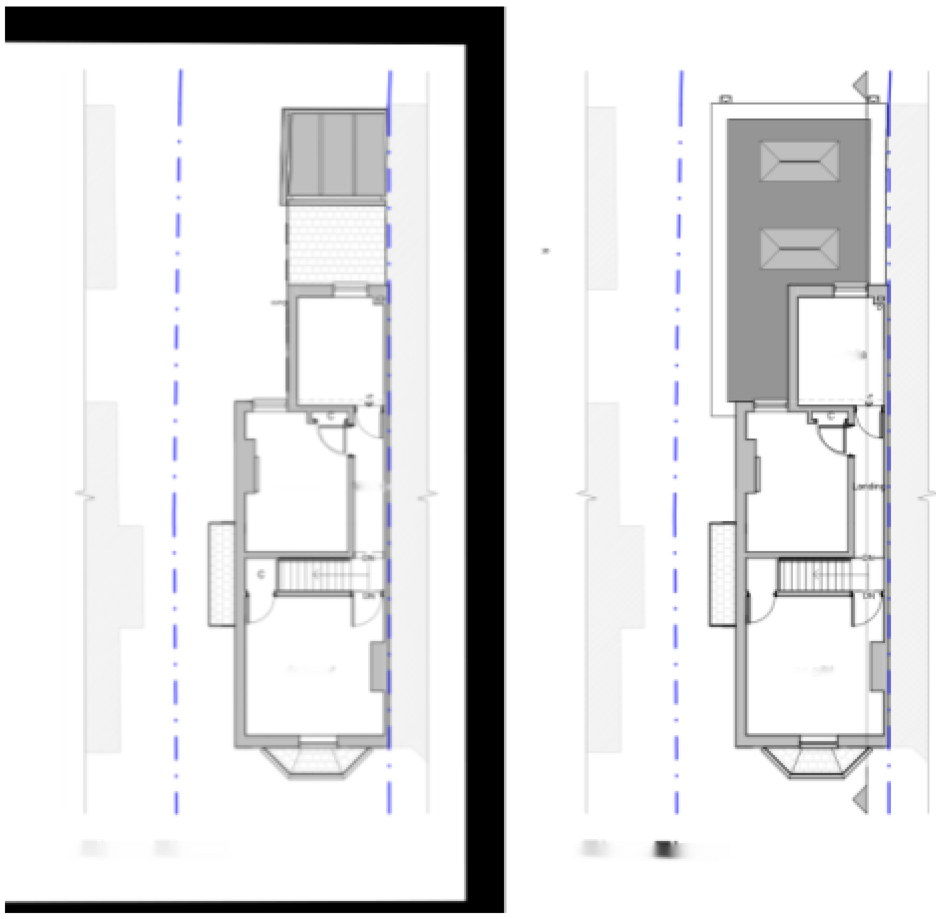

Figure 2.3: The aligned comparison images

The black bars on the sides are the result of cropping and aligning the two images. This bar would be detected as a difference between the two images but I left this in as I cannot change the colour from black and all image differences will be checked manually either way so I decided not to focus on fixing this.

2.1.2 Algorithms

At the start of the internship, I researched multiple non-machine learning methods and algorithms to detect differences between two images. The main methods that I attempted are as follows:

1. Iterate through each pixel and if the two pixels are not the same, highlight.
2. Call an external function to calculate the difference between the two images.
3. Compute the SSIM value of the two images, threshold the difference image, then find contours to obtain the regions of the two input images that differ.
4. Similar to method 3, except the drawing and highlighting of the differences are more specific and compact.

All three techniques are pixel-based comparisons. I decided to focus on methods that did not involve machine learning first because traditional computer vision approaches massively reduce computation times and CPU or GPU load. Further, non-machine learning methods work best when images are near-aligned, which is what we are dealing with in this project.

Thus, I concluded that keeping to non-machine learning methods would not come at a disadvantage.

All of the above three techniques produced different results but the best was the fourth option (in the code, this is the method `find_diff_m2`), given the inputted images do not have a lot of noise and the majority of pixels match and align together. Due to the fact that these methods are pixel-based, they are very sensitive to noise. Both images have to have the same image quality or resolution, any pixelation disrupts the results massively. This is also the same for any textured areas in the images unless both textures are exactly the same in both pictures.

An example of the results can be seen in Figure 2.4 using method 4. SSIM usually needs the reference image to be of perfect quality [2].

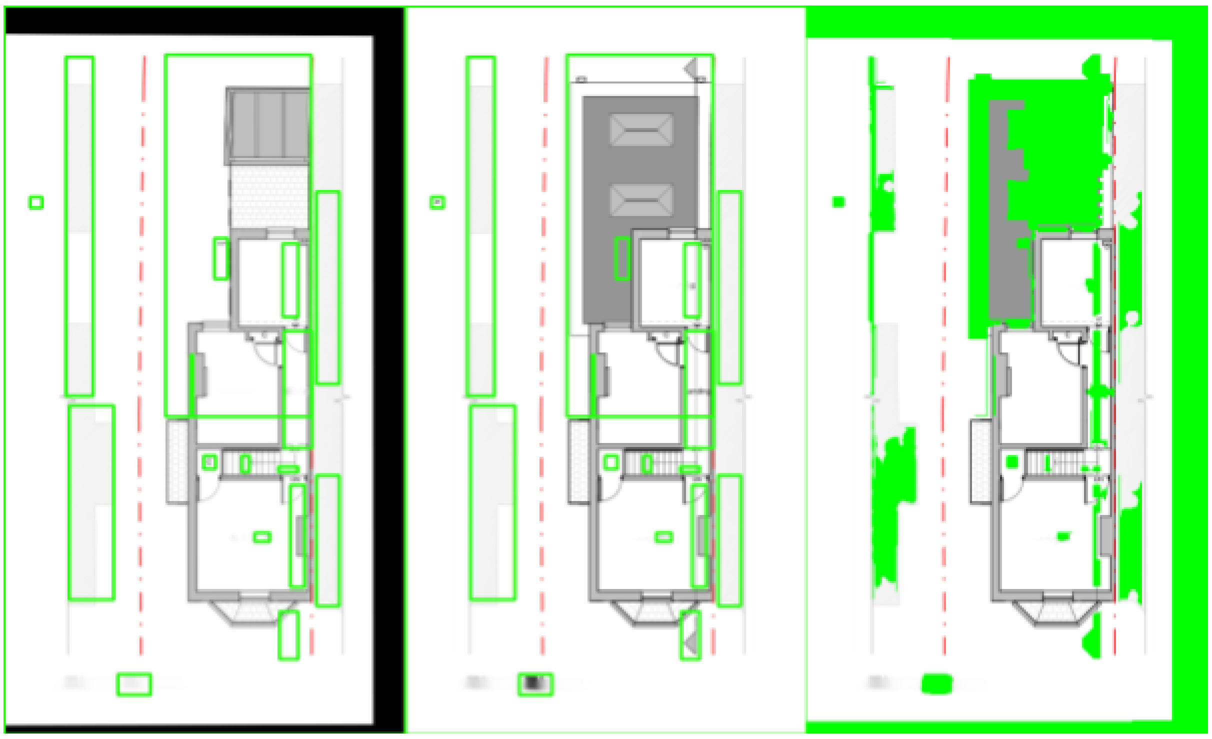

Figure 2.4: The results produced wherein the left is the original, the middle is the proposed, and the right is the highlighted differences.

An image with lots of textured noise, shown in Figure 2.5 will affect the results.

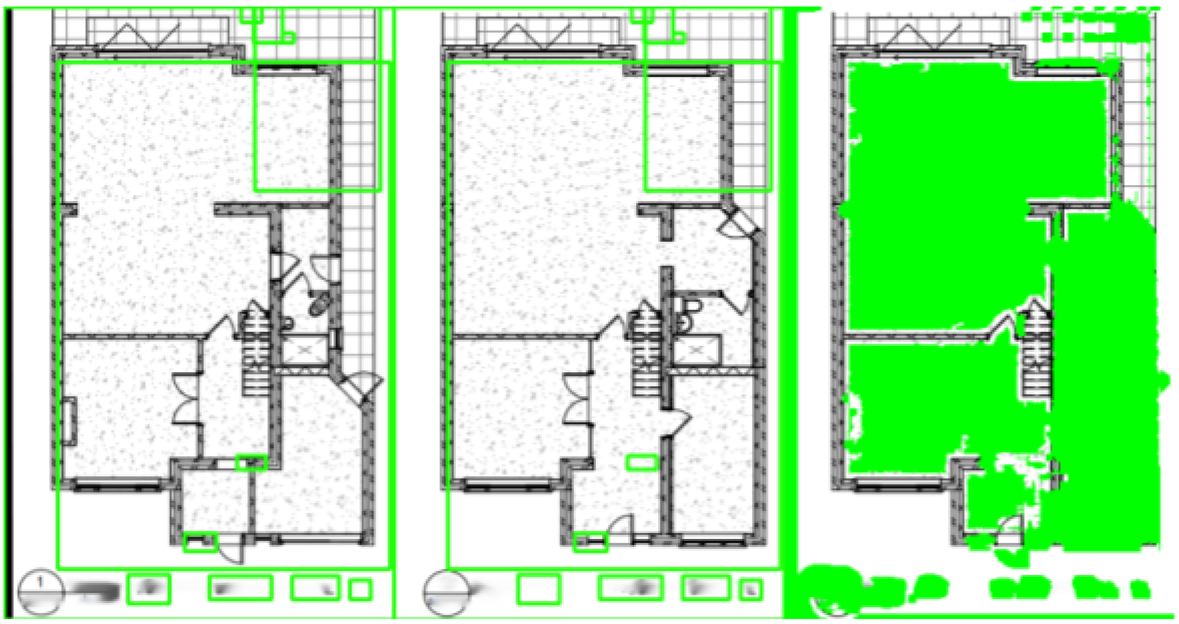

Figure 2.5: Example of the effects of textured, mismatched areas in the image

Thresholding could have been one way to reduce the number of changes detected (i.e. if the SSIM score is above a certain threshold, it will show as a change in the images). However, I found that this was not possible with any given Python modules.

2.1.3 User Interface

The user interface I created was built using PyQt6, a Python library that provides extensive tools to develop UIs. While there are several similar libraries, I chose to use PyQt6 because of the experience I have with it over other alternatives. The final user interface can be found in the script, `MainWindow.py` and looks as seen in Figure 2.6.

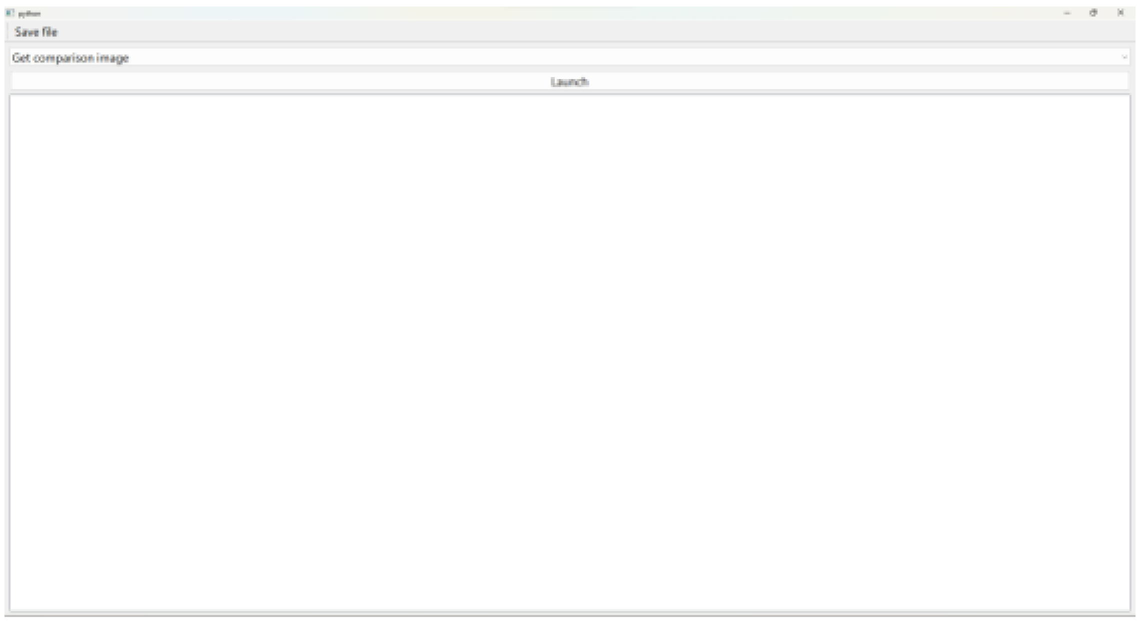

Figure 2.6: The user interface created for this project

I chose to build a simple interface and focus on the functionality rather than the appearance. This was to make navigating the UI easier and more straightforward. Users must select the option “Get comparison image” which will launch a file dialog box to select a file. This file must contain two images side by side wherein the left is the “before” image and the right is the “after” image. Results can take a while to load. Users can use the mouse scroll wheel to navigate the large box to zoom in and out of the images produced.

Users can also save the resulting image locally using the navigation bar at the top and clicking the tab that says “Save file”. This further allows users to choose which image file type to save it as.

The UI can be launched through the executable file, `Difference Detector.exe`. Otherwise, it can also be launched through the `MainWindow.py` file by running the command, “`python -u “path\to\file”`” or “`python MainWindow.py`” if the current directory is the directory where the Python script is located. The file `ImageDifference.py` is the script that contains all of the functions for highlighting the image difference.

2.2 PDF Processing

After confirming that PorthouseDean would prefer a PDF input into the UI rather than an image input, my focus then turned to processing PDF files.

My first course of action was to check if I could directly extract images from the PDF as the floor plans could originally be in an image format embedded into the PDF. To check this, I wrote my own script, `PDF2Images.py`. I have tried other, existing scripts on Github such as PDFFigures 2.0 [3] and PDF Image Extraction [4] but both have the same results as my script. Testing all scripts and evaluating their output shows that the PDFs that get sent to PorthouseDean have embedded images that are not necessarily the floorplans. The output of all scripts show that any images in the PDFs consist of divided sections of drawings of the houses which is not the data that is needed.

I also checked through the majority of the PDF files on Adobe Acrobat, as the software allows users to see the layers. The concept of layers and PDF files were brought up during a meeting and I thought it might be a good idea to utilise it if, potentially, those layers contained information on different materials or textures on the drawing. However, it was found that not all files had layers and more often than not, those layers were not useful for this project. Therefore, the idea of using PDF layers was discarded.

The other option would be to build a script that converts all pages of the PDF input into images and create a custom object detector to detect the existence of floor plans in a page.

2.2.1 Custom Object Detection

The first step in a machine learning pipeline is usually data preparation and pre-processing. Preparing the data for machine learning meant manually outlining any floor plans in the PDFs provided and labelling them into the “floorplan” class, which is the class of interest. The software “labelImg” is a commonly used tool for annotation and produces .XML files for machine learning. An example screenshot of the usage of this tool can be seen in Figure 2.7.

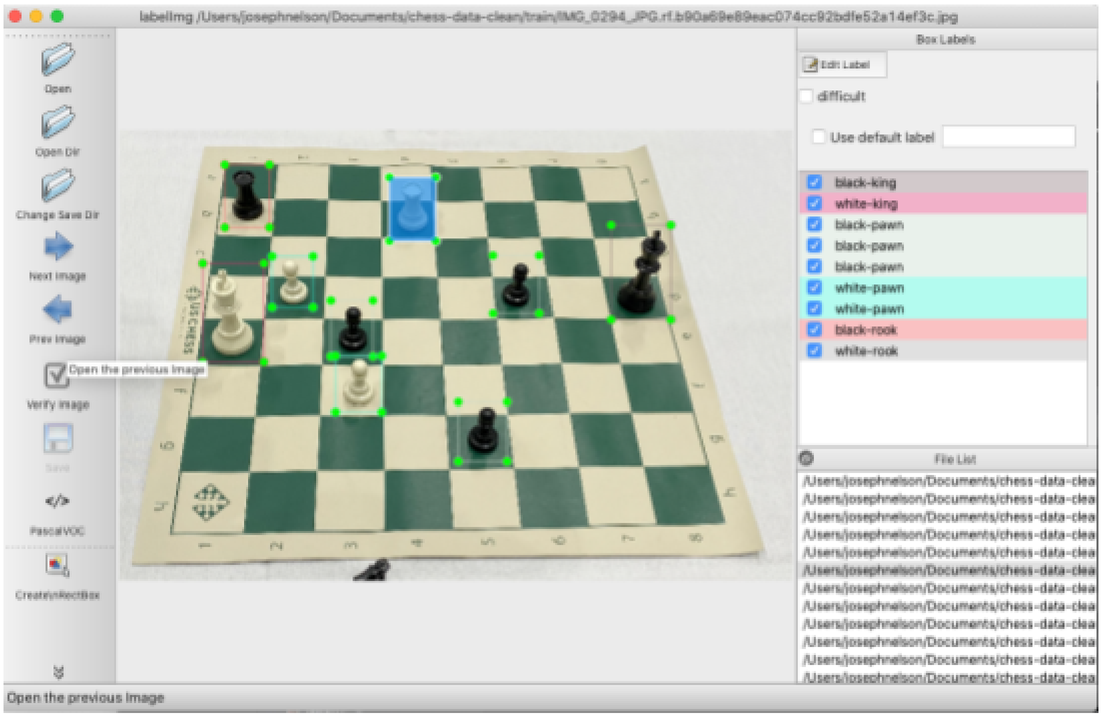

Figure 2.7: Using the annotation software, labelImg

The annotations for future use can easily be changed by clicking within the bounding boxes and manually renaming their labels.

Tensorflow has published an official tutorial [5] for custom object detection using Tensorflow v2, an up-to-date version of the library. I chose to follow the tutorial step-by-step as it was very comprehensive and detailed. However, I ran into a few errors with version dependencies on a variety of other Python libraries. After a few days of trying to fix this, I ran into another error with the `generate_tfrecord.py` script, specifically this line:

```
import tensorflow.compat.v1 as tf
```

The library `tensorflow.compat.v1` does not exist in Tensorflow's v2 so an alternative had to be found.

Despite there not being another official Tensorflow tutorial, there were a few other unofficial tutorials compatible with the updated version of the library. The resource I primarily used [6] had trouble generating .tfrecord files for .PNG files as it was only compatible with .JPG files and I made the decision to adapt the `generate_tfrecord.py` file to include .PNGs as I wanted to use .PNGs for later code. JPGs are lossy and so a lot of information is lost while PNGs preserve a lot more information. I found those image file types to be more effective when using the image difference algorithms which was the reasoning behind using PNGs.

While I tried to train a model, I encountered a few technical challenges along the way. For instance: version errors with different Python libraries, file type incompatibilities, and other errors related to Python libraries. These made it difficult for me as it meant that my time was spent debugging these errors, some of which I could not find the solution for. As a result, while a model has been trained, it is not yet optimised to its highest potential.

3. Conclusion

Reflecting on the project as a whole, this final chapter presents personal thoughts on my methodology and implementation, and any suggestions for future work.

3.1 Review of Aims

The task given to me was to develop a user interface which would call upon computer vision techniques to detect and highlight any differences between two architectural floor plans. As a result of this project, a UI was successfully built and machine learning techniques were explored.

3.2 Future Work

While I completed the task assigned to the best of my abilities and knowledge, there are definitely areas for improvement.

One approach to building upon this work would be to further improve the image difference detection by incorporating machine learning techniques. Looking into siamese networks, which are commonly used to learn a similarity metric between two photos, to detect those changes instead might be worth doing. Unfortunately, for this project, I did not have the expertise to delve into this approach. Considering vectorised images may also be an option. Vectorising the images would steer away from a pixel-based comparison and there is less likely to be noise caused by pixels. However, the conversion from a non-vector image file type to a vector image file type such as .SVG would be less valuable compared to the original vectorised floor plan which is not within the control of PorthouseDean.

An additional area that could be investigated is the use of Reinforcement Learning from Human Feedback (RLHF). While this is a more technical and advanced method of machine learning and optimisation, and is also not entirely necessary, the UI could incorporate a button to determine whether the results are satisfactory or not and the classification or detection model can use that human feedback to learn in real-time. Again, due to this being online machine learning, it would require a lot more resources than is necessary.

A third area of this project that needs to be developed is the completion of the PDF processing pipeline. This involves incorporating the floor plan object detection model into the user interface to detect floor plans in user-inputted PDF files. The model, particularly the parameters and choice of model, may need further refining in order to do this so the accuracy score produced is high enough. Additionally, the files given to PorthouseDean contain the “Original” and “Proposed” images on different pages on the PDF. This may be difficult to overcome but I have allowed for the .XML files of the sampled images and data to easily be changed for this. The only changes that need to be made would be the re-labelling of the training and testing images. The “floorplan” class would have to be removed and replaced by the “original” and “proposed” classes. It would then be a matter of training the model to detect both classes, which was the reasoning behind including text in the bounded boxes as the text could be used to differentiate between the two classes, and to match them up using a similarity metric potentially.

Finally, another way in which the code could be expanded upon is the detail of the detection algorithm. For example, meetings between PorthouseDean and I involved the detection of different materials being used in the floor plans. This is an area that was not delved into but could be investigated further. Again, this would be a problem appropriate for machine learning but personally, I would deem it too difficult as of now because the drawings given to the company are not very consistent and the model may not produce high accuracy scores. Despite that, it would be beneficial to explore but if results are not found to be satisfactory, the inconsistency may be the cause.