Image Recognition Challenge for Rooms (from Microsoft)
— Final Report —

Tomasz Bartkowiak, Suampa Ketpreechasawat, Nattapat Chaimanowong,
Danlin Peng, Lin Li, Yini Fang
{tb2816, sk3217, nc17, dp2315, ll4117, yf3016}@doc.ic.ac.uk

Supervisor: Dr Anandha Gopalan
Course: CO530, Imperial College London

29th May, 2018

Abstract

The project is a collaboration with Microsoft aimed to create a smart accommodation platform utilizing various machine learning algorithms, with main emphasis put on the Faster RCNN object detection model. The results of our work include the implementation and deployment of a front-end web application and the back-end server for our smart accommodation platform, ***Smart Acc***. Our product provides unique functionalities (such as *Find me a similar room*) that are not yet available in well-known existing accommodation platforms (for example *Rightmove*). To support the products features, we trained and optimized a Faster-RCNN model to detect objects in room images and a Random Forests model to assign a room type to each room image. The models achieved the accuracy of 50.6% and 98.4%, respectively, on our custom dataset. The product can be found under the address *https://accommodation-platform.azurewebsites.net/*

Contents

1 Introduction

1.1 Overview of the Project

Motivation Visual impression is undoubtedly the most important factor influencing people's decision concerning the choice of accommodation. An accommodation platform should aim to organize visual information so that it is simple and intuitive for the user to browse through. The user may also be interested in features that are available in a room. From the perspective of the accommodation provider, the process of updating such information can be tedious and time-consuming. A *smart* platform should be able to understand and adapt the search results to the user's needs, as well as to automate and simplify the accommodation providers' actions. Development of a product to address these issues provides a good opportunity for the application of machine learning algorithms, specifically classification and object detection models.

Problem formulation The task was to, given two baseline proof-of-concept (POC) projects[1], improve the existing algorithm for object detection (and gather all necessary data if needed), create a rooms' image classifier and create and deploy a POC of a product supporting the aforementioned features.

The product Our product, ***Smart Acc*** web application (web app), targets the room owners who have a room for rent, and all the people willing to find an accommodation in London. The main functionality of the application is to auto-classify room types and identify all the room features. The automated feature tagging provides the flexibility for the room searching and reduces the time consumed for home owner in manually filling the room description. Another features of the web app, ***Find me a Similar Room***, allows users to search for the rooms containing features similar to the image they uploaded.

1.2 Background on Object Detection

Object detection is an important topic in computer vision, which involves detecting instances of objects of particular classes in an image. The goal of object detection is to identify classes of objects and their position in an image.

Today most detection systems work by by running a classifier on the specific region of convolutional features. The regions, defined by their position and scale, are so-called RoIs (Region of Interest), which represent the potential areas, where the objects that are to be detected, can possibly exist. Undoubtedly, the performance of detection systems relies very much on the accuracy and speed of finding RoIs.

Early research, e.g. DPM (Deformable Parts Model)[1], utilized the sliding window approach, where the entire image was divided evenly into lots of smaller regions. More recent studies, like R-CNN[2], SPP-net[3] and Fast-RCNN[4], use a region proposal method, Selective Search, which greedily merges adjacent regions based on the low-level features like color similarity, texture similarity and size similarity. The state-of-the-art detection systems, like Faster-RCNN, compute region proposals via a deep convolutional neural network named RPN (Region Proposal Network)[5] which generates RoIs based on the convolutional feature map.

More specifically, Faster-RCNN consists of a RPN running on top of convolutional features (shared by the region-based detector - Fast-RCNN) to generate region boxes where each box is associated with an *objectness* score - whether an object is present at a particular location on the feature map or not. So the output of the RPN is a bunch of boxes (proposals) that will be later on examined. The Fast-RCNN detector is then run on these region proposals to predict a specific class. After classification, post-processing techniques like NMS (Non-maximum Suppression) or *bounding box regression* are applied in order to eliminate duplicated detections and refine the coordinates of bounding boxes.

1. Baseline project web app: https://github.com/karolzak/CNTK-Python-Web-Service-on-Azure
Baseline project Faster-RCNN: https://github.com/karolzak/CNTK-Hotel-pictures-classificator

2 Specification

This section lists all the requirements (and the corresponding task achievement status) that we had defined at the beginning of our project. The details concerning meeting the requirements and our achievements can be found in Section 7.2. This section also focuses on the stakeholders involved in the project.

2.1 Minimum Requirements

The minimum requirement for the project was the deployment of an accommodation listing platform which utilizes CNN to organize images based on the room types and allows the user to search and filter accommodations based on common metrics such as location and price range. To achieve that, the following (sub) requirements were stated at the beginning of the project.

1. Implementation and deployment of a web application that supports all the functionalities in the specification. (Requirement met)
2. Deployment of RESTful API containing the CNN room classification model and database as an Azure Web App. (Requirement met)
3. Creation and training of the model in CNTK[2]. Gathering any data needed to train the model. (Requirement met)
4. The model should perform the classification task with reasonable accuracy. (Requirement met)

2.2 Advanced Requirements

The extensions are listed below in order of priority (highest to lowest) and are also shown with the achievement status. Note that *Revised* status means that the requirement was altered a little bit (details can be found in Section 2.3).

1. Improve the model accuracy by utilizing some more advanced CNN architecture such as VGG-16[6] and ResNet[7]. (Requirement partially met)
2. Additional features of the web application. Allow users to sign in and save accommodations into their shortlist. Suggest similar accommodation based on uploaded images. (Requirement met)
3. Additional deployment platform. Improve the user experience by adapting screen size automatically. (Requirement met)
4. Improve model evaluation performance by using alternative methods of object detection. (Requirement partially met & revised)
5. Comparison of accuracy between different models. (Requirement met & revised)

2.3 Revised Requirements

All of our minimum requirements have been met, but some of the advanced requirements needed to be revised mainly based on feasibility, experiment results and the feedback from Microsoft staff and users. The details of revisions are shown as below.

1. The requirement *Improve model evaluation time by using alternative methods of object detection* has been changed to *Improve model evaluation performance by using alternative methods of object detection*.

   We thought the improvement of accuracy was more important than the improvement of detection time (concerning the use cases of our product), and therefore we changed our research direction from improving efficiency to improving effectiveness.

2. Microsoft Cognitive Toolkit (CNTK): https://www.microsoft.com/en-us/cognitive-toolkit/

2. Another revised requirement was *Comparing the accuracy against the models which go from input images to (room) category directly without any separate identification of objects.*

   We changed that requirement to *Comparison of accuracy between different models* since the original requirement was not essential, i.e. tightly related to the performance of our web product, and required altering the majority of the detection model architecture.

2.4 Stakeholders

The stakeholders involved in the project include:

1. **The user**, which can be anyone who is looking for an accommodation or has an accommodation for rent.

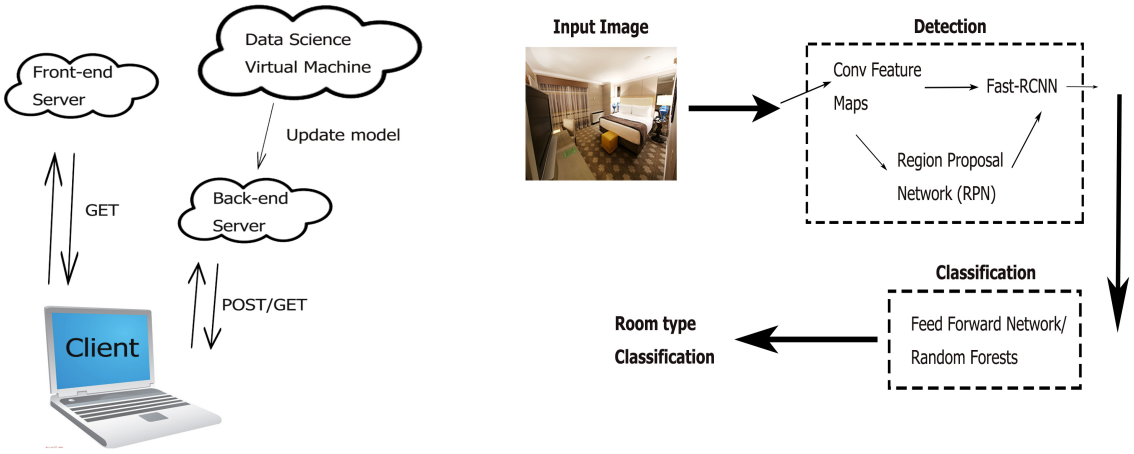
2. **Microsoft**. The project aims to provide Microsoft with the feedback on the usage of CNTK in implementing CNN. This will allow Microsoft to further improve their tool.

3. **The open source community**. From the experience and insights gained during the project, some contribution has been made to the open source community (GitHub), especially relating to the topics of CNN and CNTK.

3 Design

3.1 High-level Design Overview

The design of the system involves four distinct modules: website (also referred to as a web app), back-end server, object detector and a room classifier. The website provides a basic interface between the user and the server and supports services like registering/logging in or room searching, where requests are being handled by the server and sent back to the client. The specific functionalities like detecting objects in a particular image and classifying the type of a room based on the detections, are implemented *behind the scenes*, on the server side, by utilizing the outputs of the object detector module and the room classifier module (which essentially are the model files created during the training process). Figure 1 shows the high-level view on the whole (general) architecture and the room classification architecture.



(a) General view on architecture

(b) Room classification architecture

Figure 1: High-level view on system architecture

In the following chapters we will discuss further the components of that design.

3.2 Website

User interface The user interface was designed to support the functionalities required by the project specification. These features include:

1. Allowing the user to search for accommodations based on location and to filter the results to show only images of selected room types.
2. Allowing the user to filter accommodations based on the price and features available in the room.
3. Allowing accommodation providers to upload new accommodations to the platform.
4. Supporting user authentication to manage uploaded accommodations or to allow user to save favourite accommodations for later access.

To make the website clear and simple to use, it was designed in a way that allows each page to be responsible for only one specific functionality. The *home page* is where the user can start the search. The user can choose to search by location or by uploading a picture to the website in order to find accommodations that are similar to their picture. The *result page* shows the results from the search that the user performed. Results can be filtered using a simple to use *filter menu* that is presented on the page. The *accommodation details page* shows all the relevant information on the accommodation that the user chooses to view. The *profile page* shows the accommodation that the user has added to their favorite, and any accommodations that the user has uploaded. The *upload page* allows a user to upload the details of the accommodation. Lastly, the *sign up page* is designed for the user to create an account on the platform.

The user interface was also designed to adapt across different screen or browser window sizes. By aiming to design an adaptable UI from the beginning, we were able to create a unified interface which works well for any screen sizes without having to rewrite the code for each separately.

In order to make the website user-friendly, we tried to incorporate commonly used UI elements into our website. This included a *search bar*, *navigation bar* and *scrolling carousel*. Additionally, we also tried to ensure that the styling is consistent across the website, which we achieved by utilizing an external CSS library *Semantic UI*. The main reason for this choice is that the naming convention in *Semantic UI* is much more readable compared to the more popular one *Bootstrap*. Additionally, *Semantic UI* can be easily customized and has an official support for *React*.

Design pattern In the actual implementation of the design, we broke down each page into modular, reusable components. These modular components can be made more specialized as required. This allowed the code to be reused effectively. The second concept that was used heavily in the design was the *component's states*. Here each component is allowed have several states which determine how they are rendered on the screen. User interaction can cause the states of each component to change, causing the appearance to change as a result. As an aside, if we view the application in an *Model-View-Controller* setting, React by itself provides just the view layer. Although there exists tools that act as the controller layer (i.e. *Flux* and *Redux*), we decided not use them in the project, since the project is simple enough to be built purely with *React*.

3.3 Back-end Server

Database design In order to store the information from the users, accommodation features, and the results from object detection and room classification model, the relational database was implemented. Regarding to Figure 2, the ER diagram consists of 4 entities: *user*, *accommodation*, *room*, and *room object*. Once the users signed up for the website, their identification and authentication records are kept in the user table. The membership allows the users to post their accommodation's details as well as relevant images, which are subsequently stored in the accommodation table. The results obtained from object detection and room classification are recorded in the room object, and the room table respectively. Apart from aforementioned accommodation records, the saved list table was also designed such that the users are able to keep the lists of their favorite accommodation.

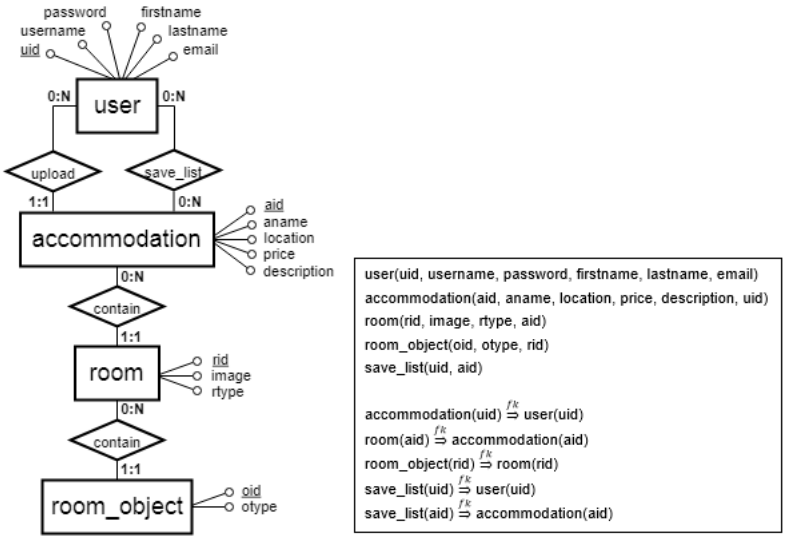


Figure 2: ER diagram for accommodation

Module design The internal structure of a web server includes a *SQLite* database, an evaluation module, and a *HTTP* module.

1. Database implementation: The SQLite is used in the web server as an internal database. Each table in SQL database represents each entity (as visualized in the previous section). For the room images, we have occupied separate blob storage in Azure to keep all image files provided by the users, and held the URLs to those images in the database.

2. Evaluation module: The evaluation module pertains to the implementation of a trained machine learning model, which could be divided further into 3 submodules, namely *object detection*, *room recognition*, and *Find me a similar room* utility.

   (a) *Object detection*: The trained CNN model is loaded and used for object detection. After the users upload the images, the image is fed into the model, and the lists of the objects existing in the image are obtained subsequently as an output (in JSON).

   (b) *Room recognition*: The pre-trained Feed Forward Neural Network (FFNN)/Random Forests (RF) model receives object lists from object detection as an input, and returns the most likely room type as an end result.

   (c) *Find me a similar room*: This module is developed, aiming to demonstrate another possible usage of object detection functionality. Instead of feeding the object lists into room recognition module, we provide them as an input for matching function, and obtain the list of the accommodations that contain similar objects.

3. HTTP module: This module is a mean for an interaction between the front-end and the back-end web service via HTTP protocol, which allows the front-end service to gain the access to the internal database. With regard to user privacy, identification and authentication of users have also been implemented in this module.

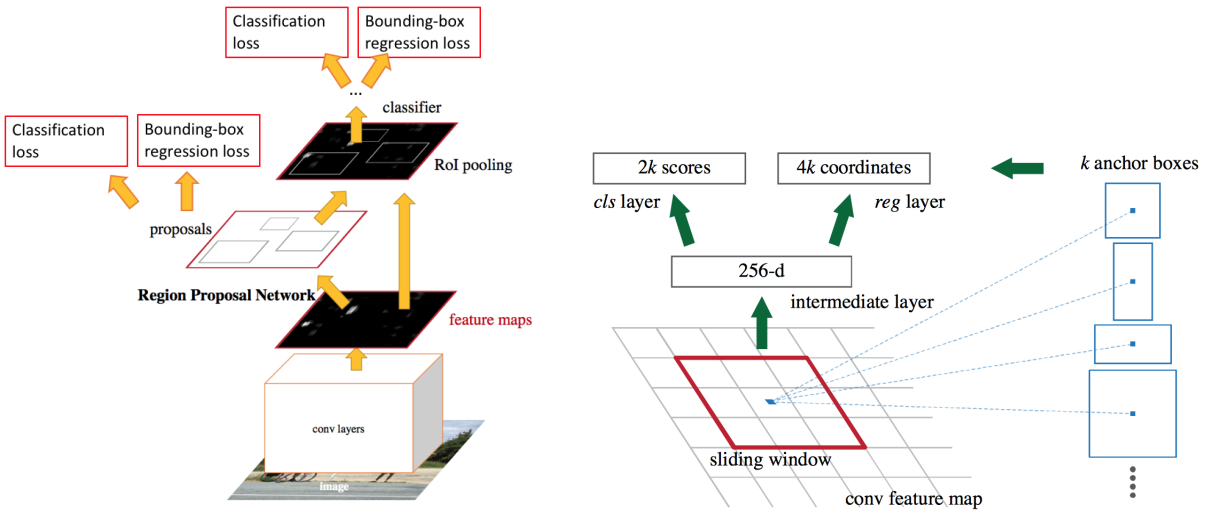
3.4 Object Detector

As given in the baseline project, this module utilizes the state-of-the-art object detection model (Faster-RCNN) to locate objects and predict their classes.

Network architecture The Faster-RCNN consists of three components: feature extractor, Region Proposal Network (RPN) and a Region-based classifier - Fast-RCNN. The feature extractor works on extracting convolutional features from the raw image input, which is usually derived from other successful deep CNNs like AlexNet[8] and VGG-16[6]. The feature map, output from feature extractor, is shared by RPN and the Fast-RCNN as parts of their inputs.

RPN's job is to slide a small convolutional layer over the feature map followed by two sibling 1×1 convolutional layers for the class score and bounding box regression respectively. At each sliding window, the center of window, denoted as the anchor, is associated with several scales and aspect ratios, which represent the potential boxes for detection regarding the corresponding window. Then the features bounded by each anchor box are fed into a box-regression layer and a box-classification layer respectively to refine the coordinates and predict the *objectness*, whether there is an object or just a background.

After the region proposal network, the convolutional feature map from the feature extractor and the region of interests from the RPN are both fed into a RoI pooling layer[4] in order to retrieve completed features belonging to a particular RoI and unify the sizes of all RoIs for further classification. Then, similarly to the two small sibling layers in the RPN, the Fast-RCNN classifier takes RoIs with convolutional features as input, and outputs a more specific class prediction (which class the object belongs to), and a further refined coordinate.



(a) Detection procedure[5] (b) Illustration of anchor system[5]

Figure 3: The Architecture of Faster-RCNN

Loss functions For the object detection problem, the objective is to minimize a multi-task loss function. Specifically, there are four loss terms in total (Equation 1), two of which are associated with RPN and the other two - with Fast-RCNN, i.e. each loss function is made up of a localization loss and a classification loss.

To train the RPN, a binary class label (indicating whether something is an object or not) is assigned to each anchor. Following the rules in Faster-RCNN[5], two kinds of anchors are considered as positive samples: (i) the anchor/anchors with the highest Intersection-over-Union(IoU) overlapping with a ground-truth box, or (ii) an anchor that has an IoU overlap higher than 0.7 with any ground-truth box. Besides, the anchors whose IoU ratio is lower than 0.3 for all ground-truth boxes, are assigned with negative labels, while all remaining samples, neither positive nor negative, are ignored, since they do not contribute to the objective. The equation of a multi-task loss of RPN is given below:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{1}$$

$$L_{reg}(t, t^*) = \sum_{i\in\{x,y,w,h\}} smooth_{L_1}(t_i - t_i^*) \tag{2}$$

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2, & \text{if}|x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{3}$$

where i is the index of the anchor in the mini-batch and p_i is the predicted probability of the anchor i being an object. The ground-truth label p_i^* is 1 if the anchor is positive and 0 if it is negative. t_i is a vector representing 4 parameterized coordinates (t_x, t_y, t_w, t_h), of the predicted bounding box, while t_i^* represents the coordinates of the ground-truth box associated with a positive anchor. The classification loss L_{cls} simply computes a log loss of two classes: object and not object.

Similarly to the RPN, the loss function of Fast-RCNN also involves two parts - classification and a bounding-box regression. Therefore, for a ground-truth class u, the loss function is as follows:

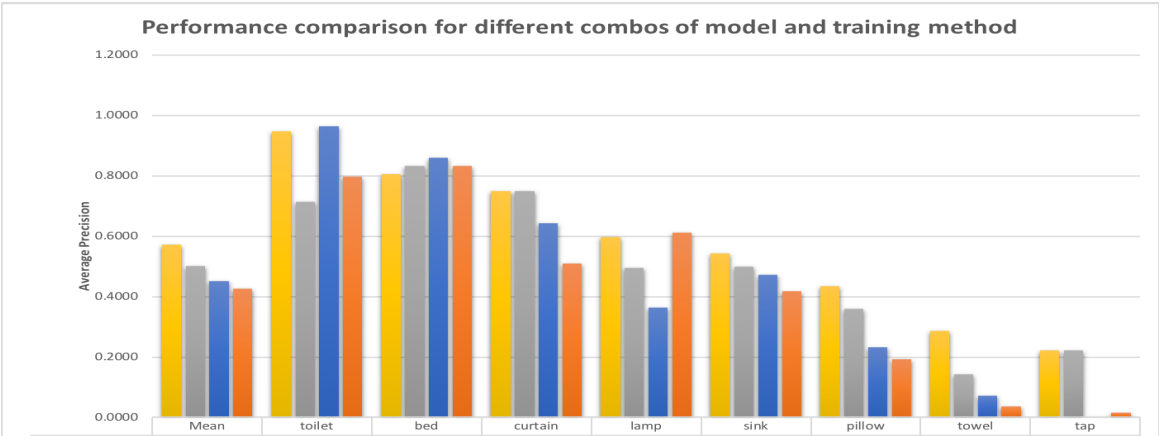
$$L(u) = L_{cls}(p_u) + \lambda[u \geq 1]L_{reg}(t_u, t_u^*) \tag{4}$$

in which, again L_{cls} is a log loss for the true class u. Moreover, p_u is the probability of the class u computed by a softmax classifier. $[u \geq 1]$ is the *Iverson bracket indicator* function, which evaluates to 1 when $u \geq 1$, i.e. the class u is not a background, and 0 otherwise. Terms λ and L_{reg} hold the same meaning (and form) as they do in Equation 1.

Training and optimization As shown before, there are two separated loss functions in total, RPN and Fast-RCNN, containing four loss terms - classification loss and regression loss per loss function - for the entire Faster-RCNN network. In order to train such complicated network, the original research[4] proposes two different ways, *end-to-end* (e2e) and *4-step alternating training* (4-step), to optimize the weights of layers. The 4-step alternating training procedure proposes that we (i) only train the RPN end-to-end, (ii) train a separated detection network Fast-RCNN end-to-end based on the region proposals generated by the RPN from step 1, (iii) initialize the RPN with the detection network trained in step 2, fix the shared convolutional layers and only retrain the layers unique to the RPN, and (iv) fine-tune the weights of unique layers to the Fast-RCNN with the shared convolutional layers and RPN fixed.

In contrast to the 4-step alternating method, the end-to-end training is much simpler and straightforward, and it is performed directly from raw input images to the final detection loss, with no intermediate partition and separate training of these neural networks. The ultimate objective is simply the sum of two separate loss functions from the RPN and Fast-RCNN without any extra balancing parameters or weights.

To make a decision between the two options, we simply ran an experiment on those two training methods with two available base models, AlexNet and VGG-16, on the dataset HotailorPOC2. During the experiment, we trained the model in four different combinations: AlexNet+e2e, AlexNet+4-step, VGG-16+e2e, VGG-16+4-step, and compared their results. As we can see in Figure 4, the combination of VGG-16 and end-to-end training performed best. Besides, training by 4-step alternating is much more expensive than end-to-end training concerning both memory (around 5GB for VGG-16+e2e and 17GB for VGG-16+4-step) and time (around 1 hour for VGG-16+e2e and 3 hour for VGG-16+4-step).



	Mean	toilet	bed	curtain	lamp	sink	pillow	towel	tap
VGG16_e2e	0.5732	0.9478	0.8056	0.7500	0.5975	0.5429	0.4339	0.2857	0.2222
VGG16_4stage	0.5022	0.7143	0.8333	0.7500	0.4944	0.5000	0.3603	0.1429	0.2222
AlexNet_4stage	0.4508	0.9643	0.8597	0.6438	0.3632	0.4714	0.2323	0.0714	0.0000
AlexNet_e2e	0.4267	0.7976	0.8333	0.5092	0.6111	0.4179	0.1930	0.0357	0.0159

Figure 4: Performance comparison for different combos of models and training methods

Hyper-parameters' selections Obviously, there must be lots of hyper-parameters to be considered and optimized for such a complicated neural network. But we can only fine-tune some of them given limited time and computational resources. In practice, we tended to follow our intuition and external knowledge (from existing research) to select potentially most significant hyper-parameters, and run the optimization procedure on them. The selected hyper-parameters to be optimized include the base model of feature extractor, the anchor scales and aspect ratios of region proposal network. Note that the remaining hyper-parameters may also contribute a lot to the performance of the model but we chose not to scrutinize them thoroughly due to, again, time and computational constraints.

The weights of all new layers, except from the ones from the pre-trained base model, are drawn from a zero-mean Gaussian distribution with standard deviation of 0.01, while the remaining layers, i.e. convolutional layers, are initialized by the base model and retrained afterwards. Besides, we adopted hyper-parameters' selection approach from the original research[5]. More specifically, we applied a learning rate of 0.001 for the first 10K samples, 0.0001 for the next 10K samples and 0.00001 for the

remaining ones, if any. Besides, the momentum was set to 0.9 and the weight decay is 0.0005.

Category	Name	Value	Source
Feature extractor	Base model	VGG-16	experiment
	Retrained layers	all	subjective
	Total stride	16	original research
RPN	Balancing parameter lambda in the loss function	10	original research
	Anchor scale	4, 8, 12	Bayesian Optimization
	Anchor aspect ratio	1:2, 2:3, 2:5	statistic
	Maximum number of RoIs	3000	original research
Fast-RCNN	Balancing parameter lambda in the loss function	10	original research
Post-process	NMS threshold of confidence	0.5	subjective
	NMS threshold of overlap	0.7	subjective
	Top-N: the amount of retained proposals for detection	500	Bayesian Optimization
Training	Learning rate schedule	0.001	baseline
	Momentum	0.9	original research
	Weight decay	0.0005	original research

Table 1: The values and sources for selected hyper-parameters

3.5 Room Classifier

The objects that have been recognized by the CNN include labels and objects positions on the image. This information is outputted in JSON format by the server using pre-trained CNN model. However, the task of the project also included recognizing the room type, not the objects and their locations alone. There was a need to create a classifier that, based on the list of labels/objects (their positions are not really important here), assigns a particular room type to that list. The high-level architecture can be seen below:

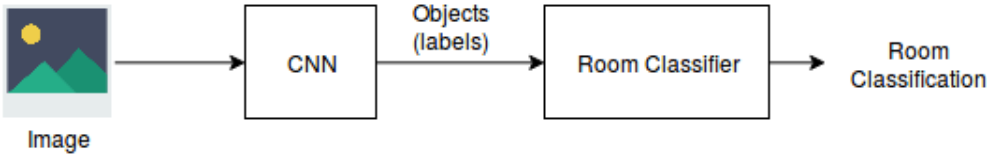


Figure 5: Room Classifier architecture

The image is fed to the object detector that outputs objects (labels), which are fed into some type of room classifier that outputs a room name, i.e. bedroom.

Model selection There are many candidates for a good classifier - Support Vector Machines, Neural Networks, Random Forests (with bagging or boosting applied) and others. We, however, decided to test just two models: Feed Forward Neural Networks (referred to as FFNNs) and Random Forests (RF). Reasons for our choice are given in Section 4.2.3. In practice, after optimizing the topology of the neural network, we constructed a simple Feed Forward Neural Network with two hidden layers and 10 neurons in each hidden layer (Appendix C). For the Random Forests model, we trained a set of decision trees based on the ID3 algorithm and obtained the final prediction by combining the predictions from all decision trees.

We trained both of two models on the same dataset and compared their performance. Based on the comparison and other factors, we finally decided to use the Random Forests model as the classifier. The details concerning the choice of the algorithm can found found in 4.2.3.

4 Implementation

4.1 Technical Solutions

The selected technical solutions (based on the aforementioned design choices) for each module are discussed below.

4.1.1 WebDev

Front-end website The front-end website was implemented in JavaScript using the library *React*. We used React 16.2.0 which is the latest version currently available. React application can be configured in many different ways but to save time, we used the official baseline configuration provided by create-react-app[3]. All the dependencies required by React itself are automatically handled by this tool. On top of React, the project makes use of several other libraries. The first of them is Axios[4], which is a library for making HTTP request to the server. The nature of React application is a single page. This means that when the user navigates to a different page in the application, the browser does not actually make a request for a new page to the front-end server. Instead, the application detects a change in the URL and updates the content to be rendered on the screen accordingly. React-Router[5] is a library we used to handle this task.

It is important to note that JavaScript has no support for object-oriented programming (e.g. classes) unlike languages such as Java and C++. Instead, we utilized powerful abstraction mechanism React provided to help organize our code. These concepts are *Component* and *Composition*. A React Component can be thought of as a function which takes in arbitrary input and returns an element to be rendered on the screen. Using Component, the UI can be split up into independent and reusable pieces which can be programmed in isolation. Complex user interface elements can be easily created by combining simpler Components. This is called a *Composition* in React terminology. Composition also allows Components to be made more specialized as required. Another concept in React, which is used heavily in our design, is Component's state. In React, each Component can have several states which determine how the Component is rendered on the screen. An interactive website which responds to user's actions can be created by allowing the users to modify these states during runtime. When a given state is updated, React automatically handles any re-rendering that is needed as a result.

Back-end server The back-end server was deployed on the Azure Web App. The modules of the web server were programmed in Python using *SQLAlchemy* and *Flask-RESTful API* to establish the communication between the internal database, the web server, and the front-end services. The *JWT* library was also utilized to provide user identification and authentication service. As a part of the *Flask-RESTful API* implementation, we separated the coding of HTTP module into 2 sections, the *Model* and the *Resource*. By inheriting the *Model class* provided by *RESTful API*, we were able to treat the information existing in SQLite database in object-oriented programming manner: the table as a class, the header of the columns as the attributes, and the tuple as the instance of the class. By this abstraction, we could directly obtain the data in the SQLite in form of class object, and record a newly created class object in the database file. For the resource section, the *Resource class* in *Flask-RESTful API* allowed us to simply treat each entity as the end-point for the HTTP protocol. In this project, we provided 4 protocols for each entity, composed of GET, POST, PUT, and DELETE. In order to maintain the integrity of the database, we designed a different access level for each protocol. Only the clients with username have the privilege to use the protocols pertaining to database modification (POST, PUT, and DELETE).

4.1.2 Object Detector

Development environment Due to the requirements from Microsoft, the detection model was trained and evaluated on the Azure DLVM (Deep Learning Virtual Machine) for both CPU and GPU

3. Create-react-app: https://github.com/facebook/create-react-app
4. Axios: https://www.npmjs.com/package/axios
5. React-Router: https://github.com/ReactTraining/react-router/tree/master/packages/react-router

versions. The platform utilized one Nvidia GPU K80 and six virtual CPUs with 56 GB memory to perform the training and evaluation. The code was developed using Python 3.5 (although the newest version is 3.6), which was caused by the compatibility issues with an external *cython* module.

Dependencies The work of the object detector depended heavily on the baseline project provided by Microsoft, in which the skeleton of a Faster-RCNN model had already been developed. Apart from that, the feature extractor of the Faster-RCNN model utilized the technique of Transfer Learning[9] to clone the suitable convolutional layers from some existing models, AlexNet[6][8], VGG-16[7][6] and ResNet[8][7], pretrained on the ImageNet dataset. Finally, the development of a detection system is based on a set of open source machine learning toolkits, i.e. CNTK, Bayesian Optimization[9], and more general packages like *NumPy*, *Matplotlib* and *Pillow*. The detailed list including all the dependent external resources with their specific versions can be found in the project directory.

Datasets In practice, two datasets have been used in the project. The *HotailorPOC2* dataset (provided together with the baseline project) has only 113 images and 8 object categories of two types of rooms - *bathroom* and *bedroom*, and was used during the early stage of the project, while our custom dataset, is a developed version of *HotailorPOC2* composed of much more images and classes. The custom dataset contains 516 images and 19 object categories of five types of rooms - *bathroom*, *bedroom*, *living room*, *kitchen* and *office*.

Object category	**Quantity of tagged objects**		**Percentage**
	Training set	**Test set**	
Pillow	289	71	11.07%
Picture	305	52	10.98%
Screen	290	43	10.24%
Sofa	233	42	8.46%
Lamp	214	49	8.09%
Chair	181	42	6.86%
Desk	171	35	6.34%
Tap	158	40	6.09%
Oven	139	30	5.20%
Sink	116	31	4.52%
Table	117	15	4.06%
Curtain	108	10	3.63%
Bed	90	17	3.29%
Towel	80	12	2.83%
Laptop	65	17	2.52%
Toilet	43	11	1.66%
Bathtub	40	13	1.63%
Ventilator	36	8	1.35%
TV	32	6	1.17%
Total	**3251**		**100.00%**

Table 2: The distribution of categories in the custom dataset

4.1.3 Room Classifier

The room classifiers (FFNNs and RFs) have been developed using standard data analysis tools (*NumPy*, *SciPy*) and *CNTK* in Python 3.5. To ensure the accuracy of model, we trained all decision

6. AlexNet model: https://www.cntk.ai/Models/AlexNet/AlexNet.model
7. VGG-16 model: https://www.cntk.ai/Models/Caffe_Converted/VGG16_ImageNet_Caffe.model
8. ResNet model: https://www.cntk.ai/Models/ResNet/ResNet_18.model
9. Bayesian Optimization: https://github.com/fmfn/BayesianOptimization

trees and the neural network based on our custom dataset.

Amount of trees The number of trees for each class is an important hyper-parameter for a Random Forests algorithm. Therefore, we ran a simple experiment to explore the relationship between the classification rate and the number of trees for each class. Surprisingly, it was observed that there was no obvious correlation between the classification rate and the number of trees. Additionally, the accuracy did not differ much among the number of trees. Therefore, to make the model simple and efficient, we decided to train 30 decision trees for each class, since that number resulted in a good accuracy and repeatability (for lower values of trees the performance fluctuated slightly from test to test). Thus the total number of trees is 30 * 5 (the number of room types) = 150.

Random Forests Random Forests is an ensemble of Decision Trees. Among many algorithms implementing a decision tree, we have decided to use the ID3 Algorithm. This algorithm involves four procedures: firstly, it calculates the entropy of every attribute; then, the data set is split into subsets using the attribute whose entropy (after splitting) is minimum; next, a node containing the "best" attribute is created. Finally, the algorithm recurses on the remaining attributes. Therefore, for a single example to be classified (a binary array containing 1s and 0s indicating all the objects detected in a picture), the final classification is a number that is from 0 to 4 representing the room type. For the Random Forests model, instead of having one decision tree, randomly selected samples from the training set are taken a set of different trees is created based on that set. Therefore, for a single example to be classified, the output given by the trees is in a form of array. The final classification is determined based on the mode of the entries of that prediction array. Compared to simple Decision Trees, Random Forests algorithm is a good choice since it reduces the problem of over-fitting.

4.2 Problems and Solutions

The following sections highlight important obstacles we encountered while developing our project and discuss the solutions we adopted to tackle them.

4.2.1 WebDev

Adaptable UI We had some issues with adapting the UI across different screen sizes which caused the development of the front-end website slower than expected. To deal with this issue, we used the CSS media size query function which allowed us to specify the styling to be applied for each range of screen sizes. In some parts, an external CSS library that we used had already implemented the solution to the problem for us.

Browser compatibility Different browsers' versions (and makes) can provide support (or not) for certain CSS and JavaScript functionalities (e.g. old browser does not support latest JavaScript syntax found in new ECMAScript 6 standard). We believed that dealing with this issue would be outside the scope of our current work, which was meant to be a POC rather than the actual production-ready website. As a result, we decided to focus our deployment to the latest version of Chrome. Our website has not been tested on other browsers.

Routing on Azure There was an issue with routing on the front-end website when it was deployed on Microsoft Azure. The problem was that the website made with React is basically a single page website where routing is done internally. The basic setup of Azure does not recognize this and tries to find a new page (which does not exist) when the user tries to navigate to a new page in the website. To fix this problem, we had to modify the default configuration file to let Azure allow the website to handle the routing by itself. This appears to be a common problem so solutions on how to modify the configuration could be found online.

Cross-Origin Resource Sharing In general, JavaScript is under the same-origin policy scheme, which limits the client's requests only to the hosts within the same domain. This policy led to the restriction in data transfer between our front-end and back-end. To relax that constraint, a Cross-Origin Resource Sharing (CORS) technique was applied. The additional origin header was generated in request form, and the server responds with Access-Control-Allow-Origin header. If the request origin matches the response's header, the web browser permits client's request.

Multithreading for long-running process The response time is considered as an important aspect of website interaction. Our problem was that POST protocol involved long-running processes such as execution of object detection and room classification model on the uploaded images. Combined with the fact that the HTTP protocol is a synchronous process, the client was blocked until this time-consuming process completed, such that the reply could be generated and sent back to client. This, however, led to high response time. To troubleshoot this issue, multithreading was implemented. By using the *threading* library in Python, the thread was created and assigned to perform image evaluation. As a result, the reply could be generated and sent back to the client shortly.

4.2.2 Object Detector

Hyper-parameters searching Training procedure in machine learning is usually very expensive, especially for such complicated models as Faster-RCNN with 136,797,594 parameters that need to be updated in each back-propagation step when using VGG-16 as a base model. These huge computation tasks costed us around 6 hours of training the model based on VGG-16 with 20 epochs on our custom dataset. As shown in Section 3.4, there are lots of hyper-parameters to be optimized. It is therefore impossible to fine-tune all of them if using traditional searching methods like random or grid search.

To speed up the hyper-parameter searching, we substituted expensive random search with Bayesian Optimization. The Bayesian Optimization works by predicting a better evaluation point (what parameters to choose for the next training), whereas the random search proposes next point to be evaluated randomly. In practice, we simply utilized the standard proxy Gaussian Process with Gaussian covariance function as a kernel. While searching, we introduced an acquisition function *Expected Improvement* (Equation 8) that trades off exploration (seeking places with high variance), and exploitation (seeking places with low mean) during optimization. The coarse ranges for hyper-parameters were, however, initially estimated based on other researches[5].

Base model adaptation The starting point for the whole object detection procedure, a feature extractor, does heavily rely on the pre-trained model, a.k.a. base model, to extract feature maps from the raw input images. The problem was how to adapt the existing CNNs to our Faster-RCNN model and fit the configurations and remaining layers of the other parts, i.e. RPN and Fast-RCNN. In order to solve this problem, we simply sliced the base model and retained the convolutional layers, which fitted the input dimensions of the Faster-RCNN model.

Essentially, the feature extractor utilizes the convolutional layers of base models to produce the convolutional feature map and part of fully connected layers to connect the RoI pooling layer to final two sibling layers. To make base models fit, firstly the last two dimensions, H and W, of convolutional feature map should be compatible with the Fast-RCNN's first fully connected layer, e.g. $H = W = 7$ for VGG-16. Secondly, the last fully connected layer and softmax are replaced (in the base models) by the two sibling layers introduced in the Section 3.4. The visualization of the architecture is illustrated in the Figure 6. Besides, based on the current settings from baseline project and the experiments' results from the original research[5, 7, 10], we fixed the total stride of all convolutional layers (up to last convolutional layer), at 16.

To implement an adaption that meets all the above conditions, we split convolutional layers for AlexNet from input layer $features$ to convolutional layer $conv5.y$ and for VGG-16 from input layer $data$ to convolutional layer $relu5\_3$. Then for both AlexNet and VGG-16, a RoI max pooling layer is appended to the tail of last split convolutional layer but with different input dimensions concerning two different base models, AlexNet($256 \times 6 \times 6$) and VGG-16($512 \times 7 \times 7$). Finally, the part of fully connected layers from base models, AlexNet(from $pool3$ to $h2\_d$) and VGG-16(from $pool5$ to $drop7$),

is used to connect the RoI pooling layer to the final two sibling layers, which produce the predicted class scores and coordinates of detections. Besides, the effects of using different base models has been already compared in Section 3.4.

Finally, as we mentioned before, we tested three different popular CNNs: AlexNet, VGG-16 and ResNet, as base models. But only AlexNet and VGG-16 performed well, while the ResNet failed to fit the current configurations of the rest networks of Faster-RCNN. Even though we modified the settings of the adjacent RoI pooling layer and RPN, it performed poorly on feature extraction, resulting in very low accuracy. The details related to this failure will be discussed later in Section 7.3.

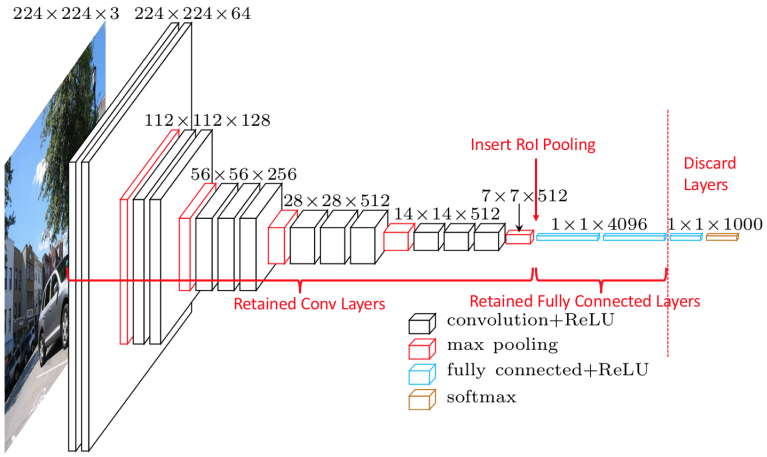


Figure 6: Illustration of adaption for VGG-16 (source: *https://www.cs.toronto.edu/ frossard/post/vgg16/*)

Post-processing NMS improvement Non-maximum suppression(NMS) is a very common and important part of the object detection pipeline. It can effectively reduce the duplicated detections by suppressing those boxes with a significant overlap, with a predefined threshold of *0.7*. But the original version of NMS leads to a miss when two real objects actually overlap, which certainly introduces a false negative error, i.e. failure to detect a real object.

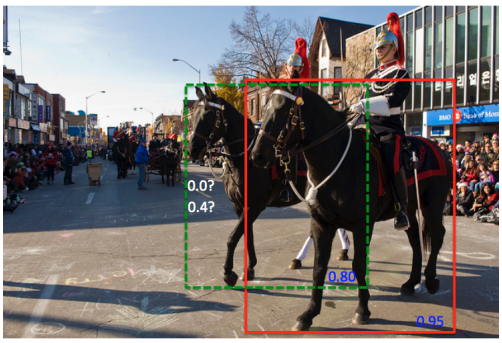


Figure 7: The Overlap between two real existing objects[11]

Following the idea of Soft-NMS[11], we replaced the original simple removing with a gradual decay on the detection score when processing the non-maximum detection boxes. The score decay simply takes a linear form as Equation 5. Besides, a relatively higher confidence threshold, at least 0.5, must be applied during non-maximum suppression to avoid a huge waste of time on meaninglessly processing those unconfident detections.

$$s_i = \begin{cases} s_i, & IoU < Threshold \\ s_i(1 - IoU), & IoU \geq Threshold \end{cases} \tag{5}$$

Custom dataset preparation In order to improve the ability of generalization and enlarge the range of detectable objects for the model, we collected more images of more types of rooms and tagged more categories of objects on those images. The statistical description of the new custom dataset has been given in Section 4.1.2. However, preparing a new dataset for object detection is quite tricky, since a bad dataset with many noises inside might confuse the model and make the detection task harder.

After suffering several failed attempts of creating a custom dataset, we have stated some important caviats, which need to be taken into consideration while developing a custom dataset. Firstly, the minimal size of the object should be proportional to the total stride, 16 pixels for the Faster-RCNN, of detection model. Because the stride operation leads to a miss of information from the low-level feature map where the most information of small-size objects are retained. Secondly, the features, e.g. shape, occlusion and viewpoint, of the objects in the different images but belonging to the same class should be consistent, or at least provide enough samples for different distinct cases. Finally, the balance between the amount of objects in the training set and test set should be kept at all times. For the object detection problem, each image should contain different amount of different kinds of objects, though it should be kept in mind that the imbalanced training set might lead to a loss in accuracy due to an insufficient amount of samples for some particular class.

4.2.3 Room Classifier

As mentioned before, there are several choices of classifiers and we have decided to try Neural Networks and Random Forests, since they have been extensively tested in the machine learning community and have been known for good performance on a test dataset and for a robustness against overfitting (provided that the architecture is well designed).

However, we then struggled to choose between the two to be implemented in the actual product. Performance of both models is almost the same, model weight is slightly bigger for Random Forests and the training time is also quite better for the Neural Networks. On the other hand, decision trees within the Forest can be easily visualized (visualization of five trees representing different room types is shown below) and the Forest can be adjusted to bigger datasets, whereas the Network might require time-consuming hyper-parameter optimization. We therefore decided to implement the Random Forests algorithm for room classification in our product.

		Neural Network	Random Forests
Average classification rate		98.9%	98.4%
Precision	Bathroom	94.9%	96.6%
	Bedroom	99.2%	100.0%
	Kitchen	100.0%	100.0%
	Living room	98.4%	97.1%
	Office	100.0%	100.0%
Recall	Bathroom	98.2%	100.0%
	Bedroom	98.5%	97.1%
	Kitchen	97.5%	95.0%
	Living room	100.0%	100.0%
	Office	100.0%	100.0%
F1 Score	Bathroom	96.6%	98.2%
	Bedroom	98.9%	98.5%
	Kitchen	98.7%	97.4%
	Living room	99.2%	98.5%
	Office	100.0%	100.0%

Table 3: Performance comparison for the FFNN and Random Forests

5 Methodology

In order to develop a high-quality product within given time, we adopted a set of modern software engineering techniques, efficiently used available technology and finally decided upon a systematic technical solution on which we built our product.

5.1 Software Engineering

5.1.1 Development Strategy

Agile development In order to systematically track work progress and convey a sufficient communication between team members, *Scrum* was selected as a practice for Agile Development. *Sprint* planning meeting was held biweekly to prepare the lists of tasks on upcoming sprint. Regular scrum meetings were also conducted for individuals to briefly update their progress within their sub-team. At the same time, the team held an official meeting once a week, aiming for all members to report any difficulties encountered and task achievements with respect to product backlogs. Additionally, the session provided the opportunity for brainstorming in which team members had an open discussion on each particular issue. Towards the end of the project, we also incorporated one of the elements of *Extreme Programming* (XP) - pair programming - within both the machine learning and the web development team. This approach helped us to fix the integration issues between the front-end and the back-end easily as well as allowed us to quickly design the API to support a new feature of finding a similar room.

Prototyping and feedback We utilized paper sketches and computer prototypes for the front-end website. Prototypes allowed us to explore possible design choices and early feedback gained from the prototypes (from other people using our application) helped us to tailor the product more towards the users needs as well as make it more user-friendly. Throughout the project we tried to gain feedback after making any changes to the design. By getting it regularly, we avoided going off the track.

Test-driven To ensure that the software we developed would meet the required specification and quality, we utilized a test-driven development approach. This approach means that test cases are written for as many required functionalities as possible before any actual coding is done. Back-end server could be an example of that. Test cases were written for all the required request endpoints to ensure that all the functionalities requirements have been met before the product can be deployed.

Modular programming We used a modular programming practice to improve the maintainability of the project. Each machine learning model has been developed as an independent module which can be exported to be used in any product that requires them. The front-end website and the back-end server are also completely independent. Both are deployed on a separate server and communicate only through HTTP requests. This means that if there was a need to alter the functionality of either of them, it could easily be done without affecting another component (as long as the API remains unchanged).

5.1.2 Communication and Collaboration

As mentioned in the section on Agile Development, we maintained a regular communication both face-to-face and through online communication tools. Apart from that we used *Slack* software to maintain communication outside the meetings. Slack was chosen because it is a powerful communication tool that is easy to use and allows for the creation of multiple chat groups in the same team, which made it easier for us to organize our communication. The ability to share and organize files in Slack is another advantage. It was is also used to communicate with our external supervisors from Microsoft. Having a single platform to handle all the communication helped to simplify our workflow.

5.2 Testing

The testing system was designed separately for each module: Website, Back-end Server, Object detector and Classifier. The majority of the testing methods involved unit testing, which checks the functionality and robustness of every low-level components like functions. Some other system tests have also been performed on the website and server to assess the performance of our product.

5.2.1 Test Suites

Front-end website Two kinds of unit tests were performed for the front-end website:

1. Testing that the each page is render correctly
2. Testing that user interaction results in a correct behavior

Both kinds of tests have been performed through the library *Jest*. *Jest* is a Facebook's official testing tool for React. The library *Enzyme* and *Sinon* are also used to assist in some specific cases which is not easily implemented in *Jest* alone (for example capturing an input parameter to a function when a button is pressed).

We used a technique called *snapshot testing* to test that each page gets rendered correctly. Snapshot test involves rendering a component and then comparing the output with a stored value. On the first run, the snapshot test simply creates and stores the snapshots of the component. This test reports any crash that occurs during the rendering process along with any differences between the rendered outputs and stored outputs (if any exists). This ensures that the UI is not unintentionally changed as the works are done for other parts of the code.

The user interaction test involves simulating an interaction and then comparing the state of a component (with the expected result) after the interaction. Various kinds of interactions are supported, for example clicking a button. This test ensures that the user interaction produces an intended result.

Back-end server We focused on testing the evaluation module associated with each HTTP protocol, the responses' body, and the user authentication. The unit test was performed through *Postman* application, and *unittest* library provided in Python. Firstly, we used *Postman* to gain the quick preview of the responses in which we could immediately verify if the format comes out as designed. Subsequently, we utilized the *unittest* to create test cases for each HTTP request on both valid and invalid input, and compared the outcomes with predetermined responses and status codes. To ensure the consistency of the databases, a dummy database file was constructed and recovered to its original state every time before testing.

For the GET method, we performed the functional testing on the database query. We initially tested the base end-point in which the query portion was left blank in the URL. Then, a filter function was tested by specifying query on individual attributes and multiple attributes (such as location and price). Apart from the normal input, we also performed the test cases with the incorrect input type.

For the POST, PUT, and DELETE method, the test suite aimed to verify if the information in the databases and the relevant images in the cloud storage was modified as intended, as well as to ensure that the corrupted data is always rejected from the databases. In similar manner to testing the GET method, unit test was performed with the correct request form, followed by the cases with the false input and missing input. Concurrently, the user authentication service was tested in a way to ensure that only registered clients had access to those functionalities.

Object detector The detection system can be divided into two parts: the main model, and auxiliary functions, with respect to unit testing. The auxiliary functions modularize some blocks of repeated code as separated helper functions to improve reusability, which makes it possible to test them separately. In practice, we only tested the modules modified by us, i.e. rpn module, nms module and map module from CNTK, and auxiliary functions, e.g. mini batch generator, data loader and image plotter. All remaining external modules are tested before by Microsoft.

To assess the code quality of the main model part, i.e. training and evaluation, we used a very common testing technique so-called mock test for the machine learning model. We split a small sub-dataset, trained a model on it and evaluated the detection results of the trained model. By comparing if the accuracy reaches the threshold, we determine if the model is well trained and can perform a reasonable detection job.

Room classifier In order to test correctness of the code, we used the *unittest* library to check the behavior of the functions. The scope of testing can be divided into 3 parts: the Random Forests training module (e.g. input formatting, information gain calculation, and attributes selection), the performance measure functions, and the interface for room classification (such as *pickle file* loading method). The test was conducted in relatively small dataset such that we could check the correctness of those auxiliary functions. We also put emphasis on testing the boundary conditions, e.g. the case in which the entropy is close to 0 (at the threshold), since it might strongly affect the structure of the trees.

5.2.2 Testing Results

The coverage rates for four modules are 89% (Website), 100% (Back-end), 85.08% (Detector) and 97% (Classifier). The details of testing results can be found in the Appendix B and are analyzed below.

Front-end website The test covers 15 source files. The files *AccommodationDetails.js* and *SimilarRoomFinder.js* have relatively low coverage compared to other files. This is due to heavy usage of asynchronous callbacks which are hard to test with the testing tool of our choice. However, extensive testing proved those modules to behave as expected.

Back-end server The test covered 17 python source files with 394 statements. In contrast to other modules, each HTTP protocol had clear-cut boundary for its usage. Therefore, all necessary test cases could be clearly identified for back-end, which led to achieving a perfect code coverage.

Object detector There are 7 source files including 838 statements under the testing, while 713 statements of them have been covered by the unit test. Most of uncovered codes come from the external modules and are not used in this program. For example, the map module includes unused evaluation metric (*VOC 2007 metric*). Besides, some codes are very hard to be covered since they exist in complicated conditional blocks, e.g. exception for unusual image extension, which are hard to be triggered.

Room classifier 261 statements of 268 statements are covered by unit test. Only a small portion of statements involving random selection have not covered.

5.3 Version Control System

Git was selected to help us manage the whole development life cycle - merging of different programmers' work and the management of different versions of the code. Due to the support from the Department of Computing, we used GitLab as our version control service.

6 Group Work

With regard to the work efficiency and time constraints of the project, the team members have been allocated into 2 sub-teams - *Machine Learning* (ML) Team and *Web Development* Team, working in parallel.

Web-development team focused on the front-end development that involved a graphical user interface design for a web service, and a back-end development that emphasized on the databases and the deployment of the machine learning model on the web service. ML team focused on the extension and the improvement of the baseline model with aim to improve its robustness and accuracy. The detailed development plan can be found in Appendix A.

6.1 WebDev Team

The tasks completed by the WebDev team are listed as follows:

Tasks	**Members**
'Web service on Azure' project (Github) exploration	Nattapat & Suampa
Web services' creation on Azure	Nattapat & Suampa
Functionalities' modification (from the previous project)	Nattapat & Suampa
GUI development	Nattapat
Development of a back-end web service	Suampa
Adaptation of different screen-size	Nattapat
Integration of the model and the web app	Suampa
Development of user authentication	Nattapat & Suampa
Testing/Improvement	Nattapat & Suampa

Table 4: Web Development team

6.2 Machine Learning Team

The tasks completed by the Machine Learning team are listed as follows:

Tasks	Members
Data (pictures of different types of room) collection	Tomasz & Yini & Danlin & Nattapat & Suampa
Data preprocessing	Danlin
Data Science VM setup (Azure)	Lin & Tomasz & Yini & Danlin
Troubleshooting/System compatibility check	Lin & Tomasz
Baseline project (Hotel room classification) exploration	Lin & Tomasz & Yini & Danlin
Baseline project bug fixing and issue tackling	Lin
Model training with CNTK	Lin & Tomasz & Yini & Danlin
Room type classification using Random Forests	Yini
Room type classification using Neural Network	Tomasz
Baseline model capabilities extension (adding new classes/objects)	Lin & Tomasz
Testing different training methods	Lin
Bayesian Optimization and Hyper-parameters' adjustment	Lin & Yini & Danlin

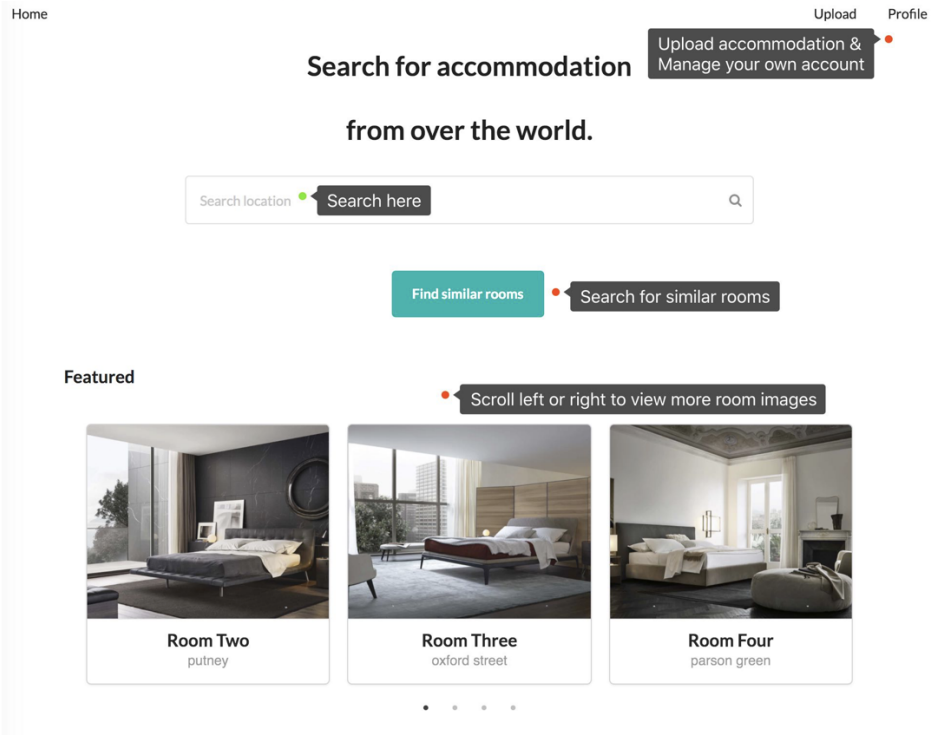
Table 5: Machine Learning team

7 Final Product

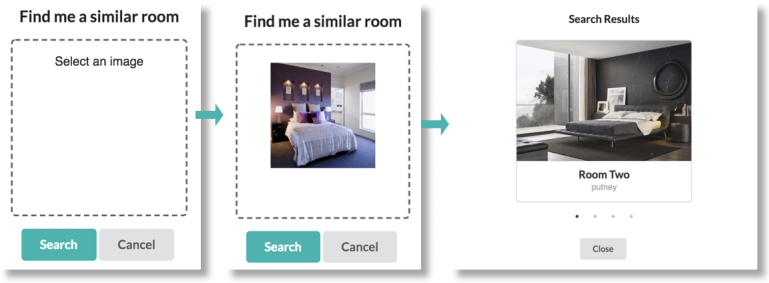
7.1 Product Demo

The website consists of 5 main sections: *Home page*, *Search result page*, *Detail page*, *New accommodation upload page* and *account register/login & authority management*. The application allows the user to search for specific accommodations based on some properties such as location, room type, price range and supported features etc.

7.1.1 Home Page



(a) Home page



(b) Find similar rooms

Figure 8: Home page & Find similar rooms

The *Home page* serves as a concise introduction and can navigate users to core feature modules. The upper right corner of the page provides users with the entrance of personal account and uploading accommodation function. In the middle of the page, users are allowed to search for suitable accommodations based on their preference of location and search for similar rooms by using the 'find similar rooms' function. At the bottom of the page there are some hot accommodations and the users can scroll left or right to explore more of them.

7.1.2 Search Result Page

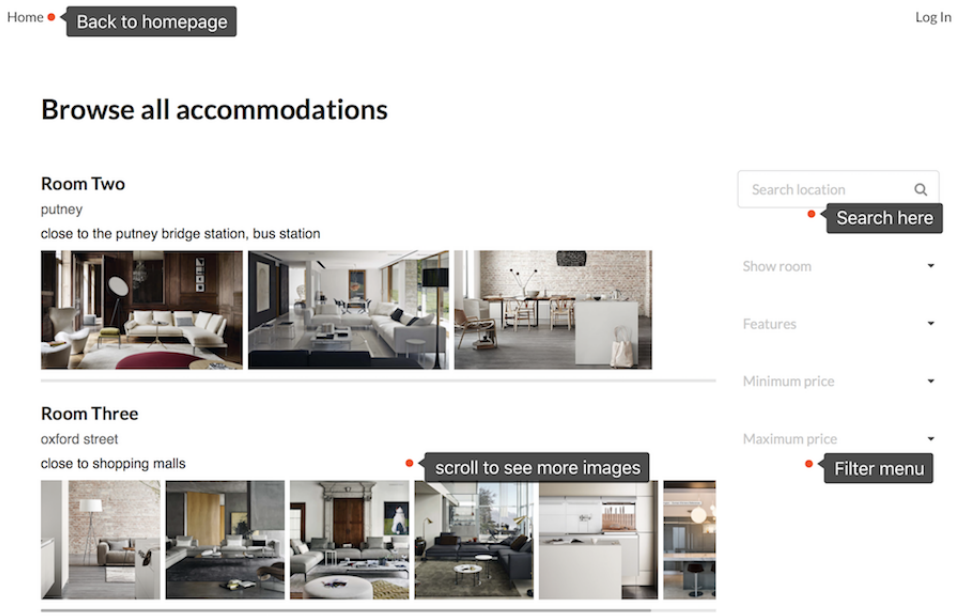


Figure 9: Full-size browser

The *Search result page* shows the list of all accommodations if users do not enter any keywords. Otherwise it shows the corresponding accommodation list according to the entered position keywords. Users can access the accommodation details by clicking its name. The right side of the page provides a search and a filtering function, where users can modify keywords and filter accommodations based on their preference (e.g. room type, supported features and price range).

7.1.3 Detail Page

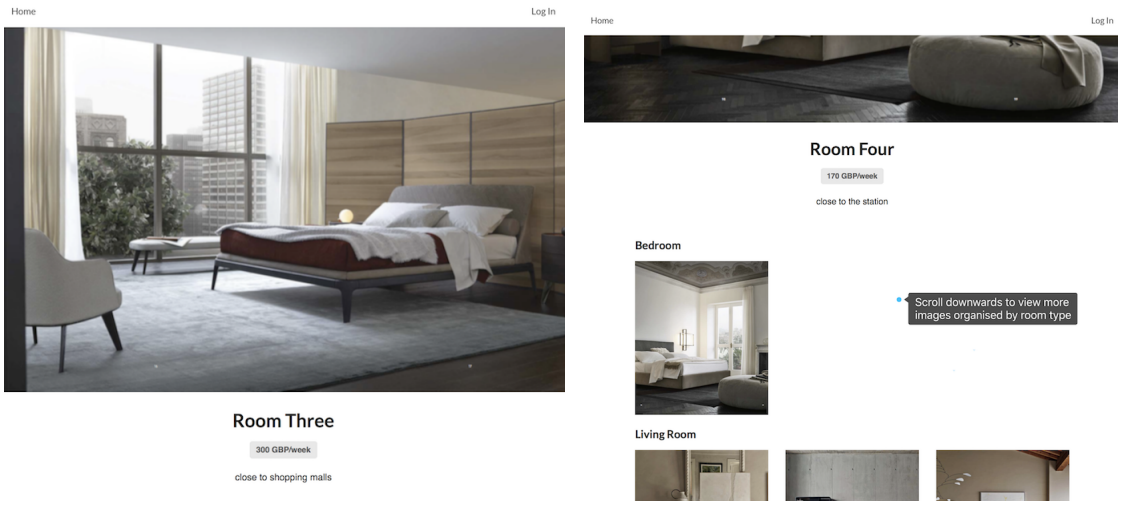
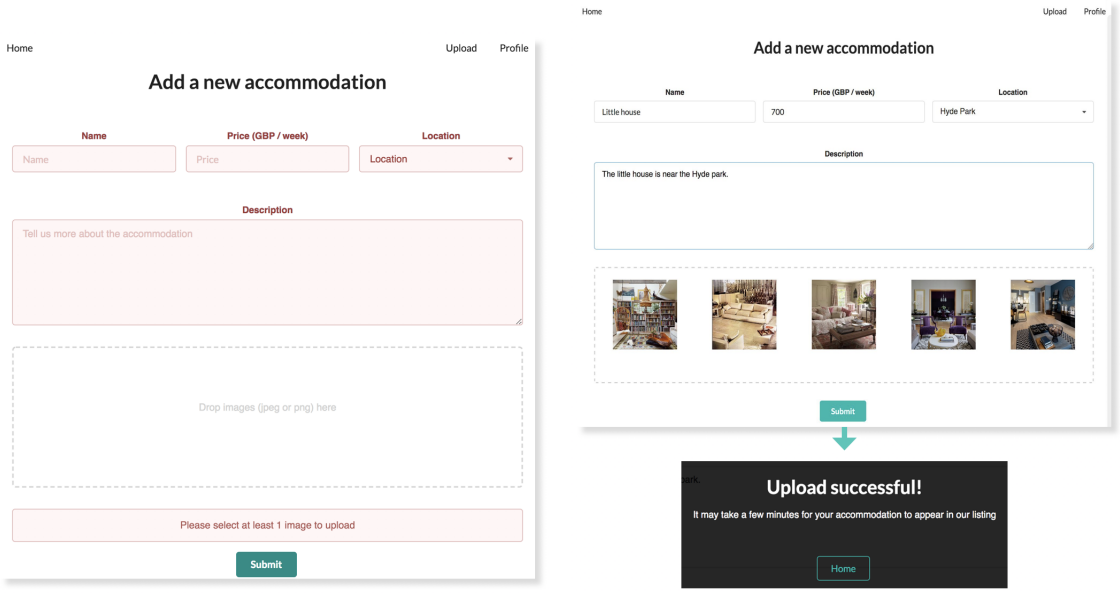


Figure 10: Detail page

The *Detail page* allows users to access more detailed information about the accommodation room and provides more room images organized by type.

7.1.4 New Accommodation Uploading



(a) Input validity check (b) Upload accommodation

Figure 11: Upload page

The *Add a new accommodation page* allows registered accommodation providers to post various room pictures and detailed descriptions on the website. This page provides a simple input validation function and accommodation holders can upload images by simply dragging the pictures from the computer to the particular area and click the *submit* button. A *successful* prompt will be shown right after the uploaded accommodation information has been sent to the back-end server. All the room images are classified by the Machine Learning server and once the classification is finished, the accommodation is published on the website and shown in the users profile page.

7.1.5 Account Register and User Authentication

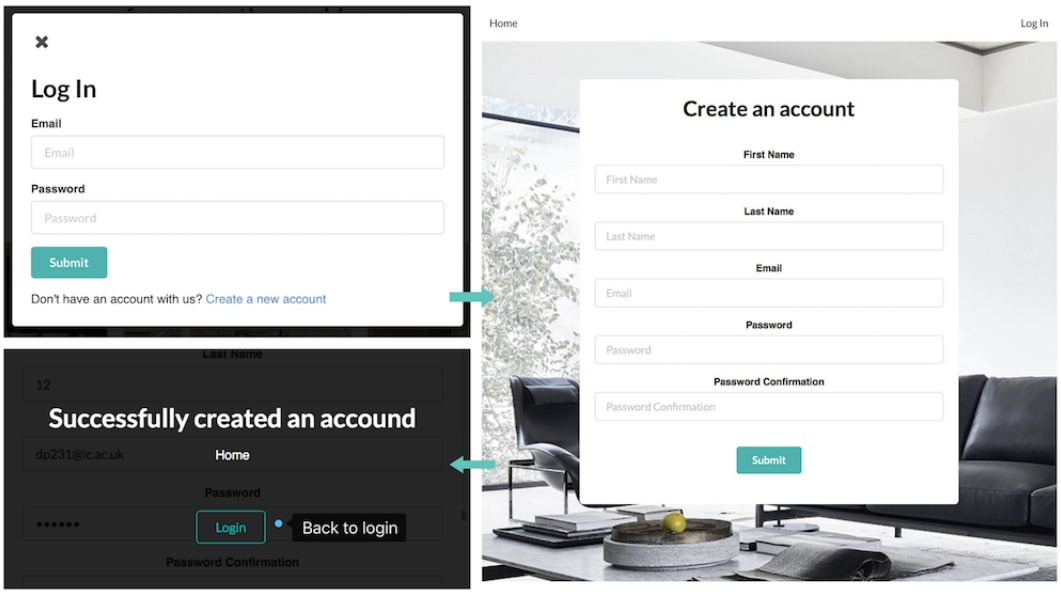


Figure 12: Login & Register

User can register an accounts to access full features for the website. Only registered users have the authority to upload new accommodations and view their own room on the *profile page*.

7.2 Achievements

As shown in Section 2, we have met all of the minimum requirements and almost all of the advanced requirements. The details of our achievements can be found below.

7.2.1 Web Product

1. *Implement and deploy front-end web application that supports all the functionalities in the specification.*

   The web application was implemented in JavaScript and React and is now deployed in Azure. The application can be accessed at *https://accommodation-platform.azurewebsites.net*. For detailed functionalities refer to the product demo section.

2. *Integrate the machine learning model and internal database into web server, and establish the associating end-points to serve HTTP protocol.*

   The end-points for GET, POST, PUT, DELETE protocol were created using Flask-RESTful API such that the front-end service was able to make a request for the resources in database, and the ML model execution. Apart from that, the multithreading technique was applied to minimize the response time of processes involving ML module.

3. *Implement additional features for the web application.*

   Two additional features has been implemented on top of the minimum requirement. First one is the authentication system which allows the user to create an account. The JWT library is used to provide this functionality. This feature protects the databases from being modified by anonymous users as well as allows users to customize their profile and creates lists of accommodation that they are interested in. Another feature is similar room search which allows users to find accommodations similar to a picture they provided.

7.2.2 Detection System

1. *Train the Faster-RCNN model and optimize it to perform a detection job with reasonable accuracy.*

   The model is well trained end-to-end and achieves accuracy of 57.32% on the dataset Hotailor-POC2 and 50.59% on our custom, larger and more compound, dataset.

2. *Optimize the hyper-parameter searching procedure by implementing Bayesian Optimization.*

   The efficiency of hyper-parameter optimization is improved due to the Bayesian Optimization, which makes it possible for us to search more potentially effective values of hyper-parameters given time and computational constraints.

3. *Reform the Non-maximum Suppression procedure of Faster-RCNN to improve the accuracy.*

   The algorithm of Non-maximum Suppression is improved to adapt to the situation that two or more real existing objects overlap together. In the original version of NMS, only one detection with highest score is retained and the remaining detections are ignored, whereas now the non-maximum detections are also retained but with decayed scores.

4. *Integrate the false-positive analysis into the evaluation component to provide a deeper analysis.*

   The false-positive analysis provides us with the details of false-positive errors, i.e. the detected object is not what/where it is, so that we can identify which problems responsible for the low accuracy and further improve the performance by adjusting the model.

5. *Create a new, larger custom dataset.*

   The new custom dataset contains 516 images in total with 418 images as the training set and the remaining 98 images as the test set. Moreover, the categories of objects are extended to 19 classes from 8 classes, while the number of rooms' types has also increased from 2 (bathroom and bedroom), to 5 with three new types: office, living room and kitchen.

7.2.3 Room Classifier

1. *Train and optimize the Random Forests model that has a satisfying accuracy*

   The optimized RF model achieves an accuracy of 98.4% for the room classification task given the detection results. Besides, it also has a good scalability concerning the changes of dataset.

7.2.4 Contributions to Stakeholders

Apart from achieving the requirements, we also contributed a lot to the second important stakeholder, i.e. open source community. We have made 9 contributions to the baseline project[10] on the Github up to the point of report submission. Besides, we also provided some useful feedbacks and suggestions to the supervisor from Microsoft related to the Azure and CNTK to help them improve their services.

7.3 Requirements not met

There are several advanced requirements that we failed to meet, which leaves a space for the further improvement of our product.

7.3.1 Adapting ResNet Model

As introduced in the Section 4.2.2, we tried our best to implement the ResNet model but the adapted model performed poorly. Following the best practices[7, 10], we adopted the convolutional layers from input layer($features$) to convolutional layer $conv4\_x(z.x.x.p)$ of ResNet-101 as a feature extractor, and all layers from $con5\_x$ up to the end were analogous to the fully connected layers in VGG-16. However, after training the network on the HotailorPOC2 dataset, the detection system still performed poorly with around 12.3% mean average precision (mAP). There are several reasons for this bad performance:

Firstly, compared to other models like AlexNet and VGG-16, ResNet has much more layers, which makes it hard to converge during the training if the data is insufficient. Unfortunately, neither HotailorPOC2 (102 images) nor our custom dataset (570 images), is comparable in size with prevalent general-purpose object detection datasets, like MS COCO and VOC PASCAL, on which the ResNet model is originally trained. Secondly, after discussing with the members of CNTK open source community[11], we find that currently CNTK does not support the technique of RoI average pooling, which has an influence on the performance of ResNet-based Faster-RCNN model.

7.3.2 Further Improvement on Performance

Due to the lack of time and computational resources, the accuracy of object detection is obviously not very satisfying, since we only fine-tuned a small sub-set of all available hyper-parameters. Compared to the best accuracy (75.6%) achieved by the original research[4] on the dataset PASCAL VOC 2012 test set, there is still a considerable space for improvement by just optimizing hyper-parameters.

Apart from that, we also explored some variations of Faster-RCNN's detection pipelines but did not implement them due to a lack of time. Inspired by the result of research HyperNet[12], we planned, at the beginning, to implement some pipelines to extract the features from low-level convolutional feature maps, e.g. conv1 and conv3 from VGG-16, in order to improve the model's ability of detecting

10. karolzak/CNTK-Hotel-pictures-classificator: https://github.com/karolzak/CNTK-Hotel-pictures-classificator/issues
11. Discussions: https://github.com/Microsoft/CNTK/issues/2679 and https://github.com/Microsoft/CNTK/issues/2202

small-size objects and generating more precise localization of region proposals. Currently, the model does the job of localization and prediction only based on the features from the last convolutional layer of the feature extractor, which misses a low-level information from the first convolutional layers.

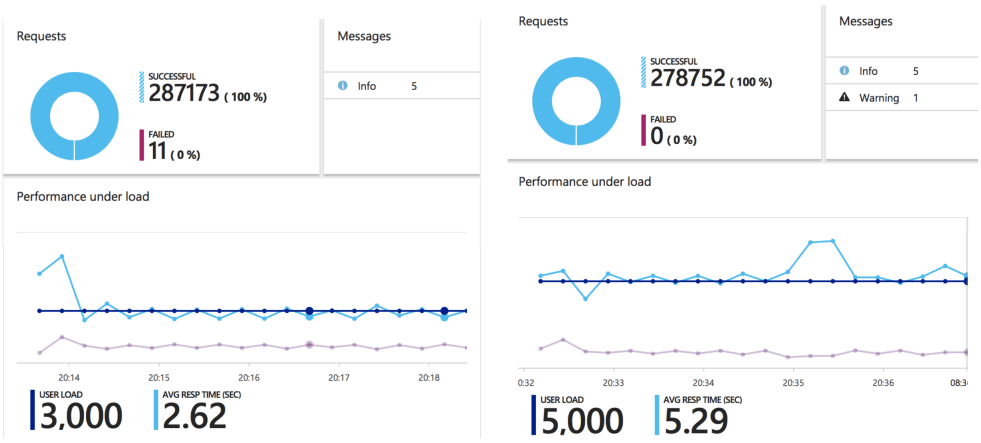
7.4 Evaluation

In this section, we assess the performance of our products and analyze the results to give a guidance for further development.

7.4.1 Website

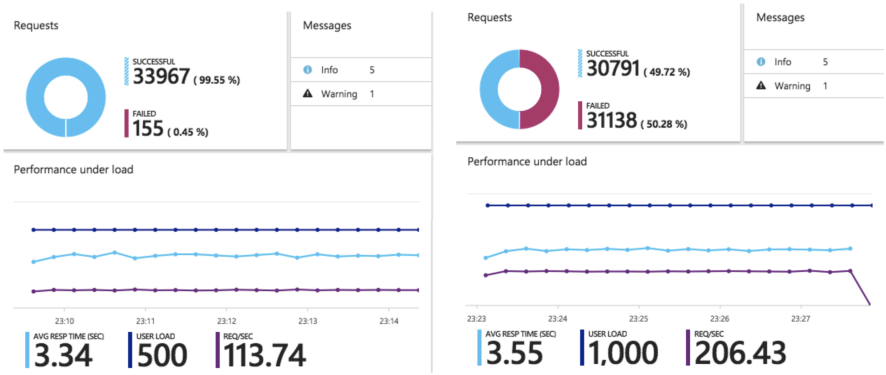
We performed some simple performance test on the front-end web application. The performance test was performed using Azure testing tool and user test was performed based on common usage scenario. The performance test simulated concurrent access by 3000 and 5000 users. The test shows that the front-end can handles these load with average response time of 2.62 and 5.29 seconds, respectively.

The similar performance test was conducted on the back-end server as well. The test showed that the web server was able to handle approximately 500 users simultaneously with the average response time of 3.34 seconds. From the result, the back-end server could handle fewer concurrent users due to high computational loads including SQL query and response formatting, and therefore, was a bottleneck of web application.



(a) Performance with 3000 concurrent users (b) Performance with 5000 concurrent users

Figure 13: Front-end performance



(a) Performance with 500 concurrent users (b) Performance with 1000 concurrent users

Figure 14: Back-end performance

7.4.2 Object Detector

Average precision metric The overall mean average precision rates on two datasets are both over 50%, whereas the performance on the HotailorPOC2 (57.32%) is sigtly better than that on the custom dataset (50.59%). This is caused by the fact that the custom dataset is far more complicated that HotailorPOC2 - there are, indeed, far more object categories.

However, the average precision rate distribution over 8 common categories of two datasets is very similar. Basically, the model performs worse job on detecting small-size objects like tap (22.22% and 37.96% respectively) and towel (28.57% and 16.67%), while it is able to achieve a very high accuracy on the large-size objects like bed (80.56% and 84.51%) and toilet (94.78% and 66.15%). The reason for this is that the small-size objects usually include relatively less features, compared to the large-size objects, and their missing information caused by the stride operation of convolutional layers takes a larger proportion with respect to the total information contained by them.

False-positive errors analysis To find the reasons for the unsatisfactory accuracy on some objects, we introduced the false-positive errors analysis[13]. As we can see in Figure 15, the small-size objects like tap and pillow are usually localized at wrong positions or are confused with the background. For some groups of objects like toilet, bathtub and sink, they usually share a very similar shape, color and surface texture so that the model is confused with the others when detecting one of them. For some objects like picture and table, their features are not consistent within the different images in the training set, which makes it hard for the model to build a unified criterion, e.g. what a picture looks like.

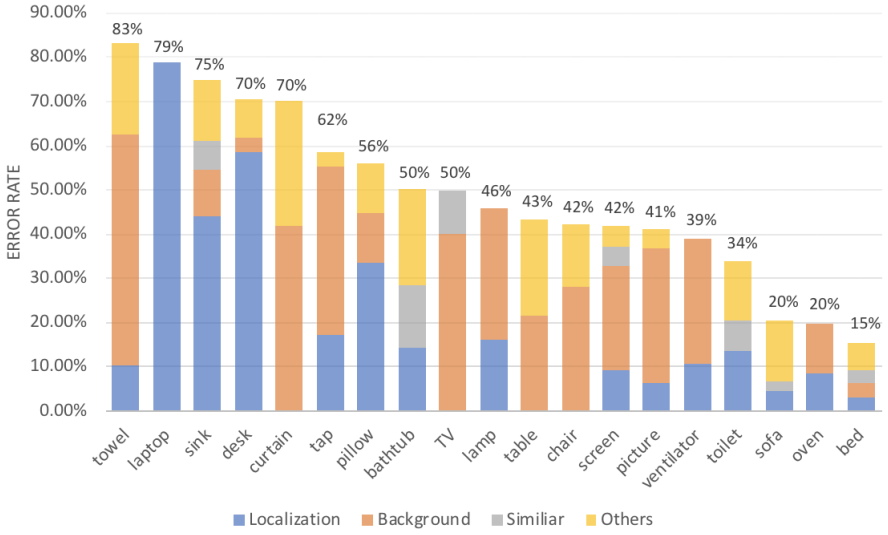


Figure 15: False-positive errors analysis

7.4.3 Room Classifier

The confusion matrix (CM) for the Random Forests classifier resulting from a 10-fold cross validation method are shown in Table 6. As we can see, the Random Forests classifier performs almost ideally. Note that the CM is normalized such that every entry in given row sums to one. This allows us to visualize what the probability distributions are for predictions given particular room type, i.e. kitchen is 98% time classified as kitchen and 2% of time as bathroom.

	Bathroom	Bedroom	Kitchen	Living Room	Office
Bathroom	1	0	0	0	0
Bedroom	0	0.97	0	0.03	0
Kitchen	0.02	0	0.98	0	0
Living Room	0	0	0	1	0
Office	0	0	0	0	1

Table 6: Confusion Matrix for the Random Forests classifier

The CM above indicates that a small number of kitchen pictures were classified as a bathroom, and a small number of bedroom were classified as living room. The possible reason is that they have some common objects, such as sink and table, and some of them are vital when deciding which room type the picture represents.

According to Figure 16, for all the room types except the living room, there is one object that uniquely decides upon the room type, which means the classifier can easily make a correct prediction based on the presence of such object. In other words, the detection accuracy of such key object significantly affects the accuracy of classifying corresponding room type.

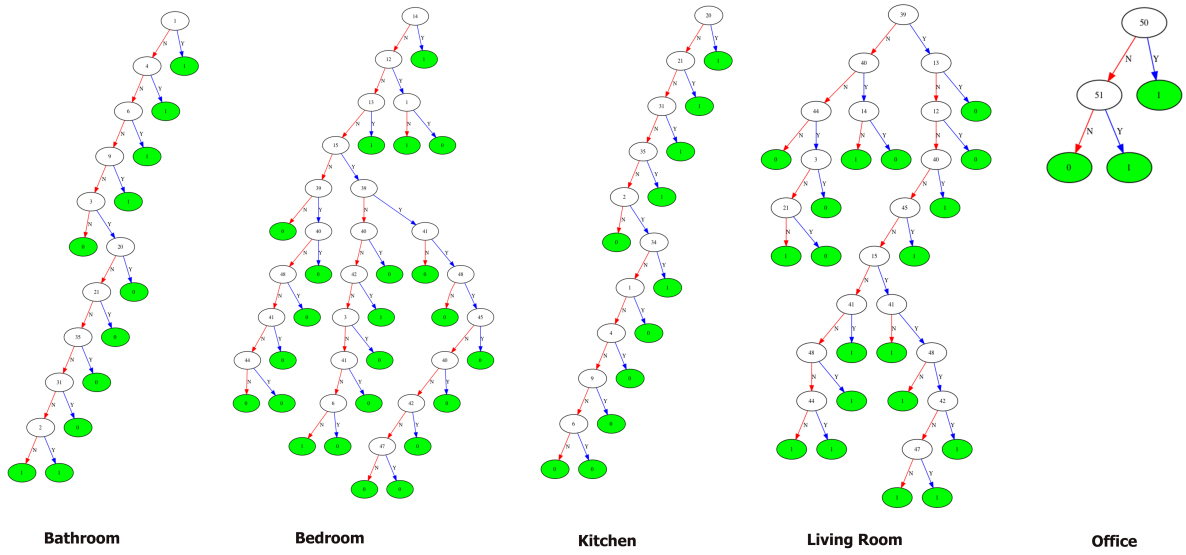


Figure 16: Tree visualization

7.5 Further Improvement

The next steps towards the product improvement would involve ameliorating the existing functionalities of our product, especially concerning the performance of response time and integrating more useful features based on the feedback from the stakeholders.

On the other hand, more improvement possibilities concerning the architecture of the Faster-RCNN should be scrutinized. The ideas include: (i) using the technique of so-called bottleneck (caching to accelerate the training speed by saving redundant convolutional operations) and (ii) utilizing the low-level convolutional features to improve the accuracy of the model concerning the detection of small-size objects.

References

[1] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, pages 1627–1645, 2010.

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1904–1916, 2015.

[4] Ross Girshick. Fast r-cnn. *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[5] 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*, page 1137, 2017.

[6] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *International Conference on Neural Information Processing Systems*, pages 1097–1105, 2012.

[9] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 2010.

[10] Shaoqing Ren, Kaiming He, Ross B. Girshick, Xiangyu Zhang, and Jian Sun. Object detection networks on convolutional feature maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1476–1481, 2016.

[11] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Soft-nms improving object detection with one line of code. In *IEEE International Conference on Computer Vision*, pages 5562–5570, 2017.

[12] T. Kong, A. Yao, Y. Chen, and F. Sun. Hypernet: Towards accurate region proposal generation and joint object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 845–853, 2016.

[13] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. Diagnosing error in object detectors. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part III*, ECCV'12, pages 340–353, 2012.

Appendix

A. Gantt Chart

B. Testing & Coverage report

B.1 Website coverage report

```
Test Suites: 17 passed, 17 total
Tests:       82 passed, 82 total
Snapshots:   52 passed, 52 total
Time:        4.862s, estimated 5s
Ran all test suites.
```

File	% Stmts	% Branch	% Funcs	% Lines
All files	89.2	90.16	86.99	89.63
src	100	100	100	100
utilities.js	100	100	100	100
src/Components	88.42	87.5	87.18	88.04
Carousel.js	91.67	100	75	91.67
FilterMenu.js	100	100	100	100
Footer.js	100	100	100	100
Login.js	100	100	100	100
MenuBar.js	100	100	100	100
RequestDetailsForm.js	83.33	100	75	83.33
SimilarRoomFinder.js	70.97	72.22	70	70.97
src/Components/ProfileItem	82.61	78.57	80	82.61
ProfileItem.js	82.61	78.57	80	82.61
src/pages	89.5	91.47	87.01	90.32
AccommodationDetails.js	74.51	83.33	76.19	76
Home.js	88.89	83.33	90	88.89
NewAccom.js	98.18	100	91.67	98.18
NotFound.js	100	100	100	100
Profile.js	100	87.5	100	100
SearchResults.js	87.1	83.33	87.5	90
Signup.js	93.48	96.88	88.89	93.48

Figure 18: Front-end coverage report

Coverage report: 100%

Module ↓	statements	missing	excluded	coverage
blob.py	3	0	0	100%
create_app.py	13	0	0	100%
db.py	2	0	0	100%
find_me_similar_room.py	30	0	0	100%
model.py	6	0	0	100%
models\accommodation.py	29	0	0	100%
models\object.py	22	0	0	100%
models\room.py	25	0	0	100%
models\savedacc.py	17	0	0	100%
models\user.py	29	0	0	100%
path.py	1	0	0	100%
resources\accommodation.py	79	0	0	100%
resources\auxilary.py	47	0	0	100%
resources\room.py	30	0	0	100%
resources\savedacc.py	20	0	0	100%
resources\user.py	32	0	0	100%
security.py	9	0	0	100%
Total	**394**	**0**	**0**	**100%**

Figure 19: Back-end coverage report

B.2 Machine learning coverage report

Coverage report: 85%

Module ↑	statements	missing	excluded	coverage
/home/mcsml/CNTK-Hotel-pictures-classificator/Detection/utils/map/map_helpers.py	125	20	0	84%
/home/mcsml/CNTK-Hotel-pictures-classificator/Detection/utils/map/det_analyzer.py	45	2	0	96%
/home/mcsml/CNTK-Hotel-pictures-classificator/Detection/FasterRCNN/od_reader.py	162	38	0	77%
/home/mcsml/CNTK-Hotel-pictures-classificator/Detection/FasterRCNN/od_mb_source.py	32	6	0	81%
/home/mcsml/CNTK-Hotel-pictures-classificator/Detection/FasterRCNN/FasterRCNN.py	453	59	0	87%
/home/mcsml/CNTK-Hotel-pictures-classificator/Detection/FasterRCNN/cntk_helpers.py	21	0	0	100%
Total	**838**	**125**	**0**	**85%**

coverage.py v4.5.1, created at 2018-05-13 14:22

Figure 20: Faster-RCNN coverage report

Coverage for **decision_tree.py** : 97%

268 statements | 261 run | 7 missing | 0 excluded

Figure 21: Decision tree coverage report

Coverage report: 75%

Module ↑	*statements*	*missing*	*excluded*	*coverage*
FFNN_classificator_test.py	52	0	0	100%
FFNN_classificator.py	165	55	0	67%
Total	**217**	**55**	**0**	**75%**

coverage.py v4.5.1, created at 2018-05-15 11:32

Figure 22: Neural Network coverage report

Coverage report: 100%

Module	*statements* ↓	*missing*	*excluded*	*coverage*
/home/Group/DataPreprocessing/ETL.py	73	0	0	100%
/home/Group/DataPreprocessing/DataClean.py	193	0	0	100%
Total	**266**	**0**	**0**	**100%**

coverage.py v4.5.1, created at 2018-05-13 14:22

Figure 23: Data preprocessing coverage report

C. Feed Forward Neural Network

The first option of Room Classifier model is a simple feed forward neural network with two hidden layers created using Microsoft CNTK API. The schematic diagram of the network is presented below:

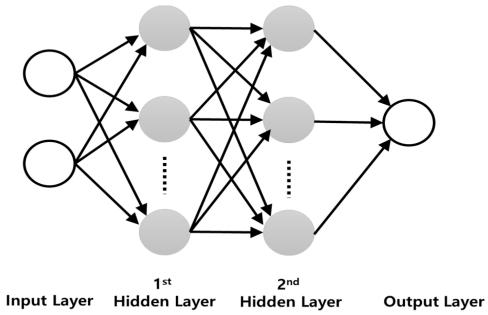


Figure 24: Feed Forward Neural Network Architecture[12]

The network is trained using training dataset, validated (using cross validation) and tested on unseen subset of the data. The proportions are: 80:10:10 for training, validation and test accordingly. Test set is subtracted first from the dataset and then K-fold cross validation is used on the remaining part to be able to evaluate the performance of the model and tune the hyper-parameters. After that, the performance on unseen test set is measured. The architecture and hyper-parameters values are summarized in the table below:

Number of hidden layers	2
Number of neurons in each hidden layer	10
Cost function	Cross entropy with Softmax
Activation	Sigmoid
Regularization	None
Learner	Gradient Descent
Learning rate	0.5
Minibatch size	25

Table 7: Summary of FFNN's Architecture and hyper-parameters

The input array, before being fed into the network is first shuffled to ensure that each subset of data is representative (random). The output layer outputs a vector of size 1x5, where each position corresponds to a particular room type. The predicted room for that vector v is $argmax(v)$. Since that particular classification task is not hard (as mentioned earlier, room types are strongly correlated with some particular objects), parameter tuning was not really necessary - satisfying accuracy (99%) was achieved immediately. However, in order to minimize the complexity of the network (to speed up testing time and minimize the weight of the model) we decided to use as simple model as possible. A satisfying accuracy was achieved with as little as ten neurons in the hidden layer. Finally, the performance on the test dataset was measured. The confusion matrix visualizing the performance of the classifier is can be seen in the *Evaluation* section.

D. Bayesian Optimization and Gaussian Process

The idea of Bayesian Optimization is to build a probabilistic proxy model, which is much cheaper to evaluate, for the original objective function, using the results of past experiments as training data. Then, the optimization result of cheap proxy function is used to determine where the next evaluation point should be for the true objective. The pseudo code of Bayesian Optimization's key steps is shown in Figure 25 (source: Marc Deisenroth, Data Analysis and Probabilistic Inference).

1: **Init:** Data set $\mathcal{D}_0 = \{\boldsymbol{X}_0, \boldsymbol{y}_0\}$
2: **for** iterations $t = 1, 2, \ldots$ **do**
3: Update GP using data $\mathcal{D}_{t-1}$
4: Select $\boldsymbol{x}_t = \arg\max_{\boldsymbol{x}} \alpha(\boldsymbol{x})$ by optimizing acquisition function
5: Query true objective g at $\boldsymbol{x}_t$
6: Augment data set $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(\boldsymbol{x}_t, y_t)\}$
7: **end for**
8: **Return** best input in data set: $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} y(\boldsymbol{x})$

Figure 25: Pseudo code of Bayesian Optimization

Gaussian Process is a generalization of a multivariate Gaussian distribution to infinitely many variables, which is specified by a mean function $m()$ and a covariance function (kernel) $k(,)$. When considering Gaussian Process regression as a Bayesian inference problem, the posterior distribution over objective functions is described in Equation 6. Then we place the GP as a prior knowledge and compute $p(f(\mathbf{X}_*)|\mathbf{X}, \mathbf{y})$(Equation 8) for training data $\mathbf{X}, \mathbf{y}$ and test inputs $\mathbf{X}_*$ by applying the properties of GP.

$$p(f|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y}|f, \mathbf{X})p(f)}{p(\mathbf{y}|\mathbf{X})} \tag{6}$$

$$p(f) = GP(m, k) \tag{7}$$

$$p(f_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*) = N(E[f_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*], V[f_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*]) \tag{8}$$

$$E[f_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*] = m(\mathbf{X}_*) + k(\mathbf{X}_*, \mathbf{X})(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}(\mathbf{y} - m(\mathbf{X})) \tag{9}$$

$$V[f_*|\mathbf{X}, \mathbf{y}, \mathbf{X}_*] = k(\mathbf{X}_*, \mathbf{X}_*) - k(\mathbf{X}_*, \mathbf{X})(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}k(\mathbf{X}, \mathbf{X}_*) \tag{10}$$

$$y = f(x) + \epsilon, \quad \epsilon \sim N(0, \sigma_n^2) \tag{11}$$

E. Object Detection Evaluation

E.1 Evaluation Metric

Due to the specifics of the object detection problem, the evaluation of the detection system can be divided into two targets: accurate location and accurate classification. To measure the accuracy of localization, a metric, Intersection-over-Union (IoU), is proposed and is computed by calculating the ratio of the area of overlap with respect to the area of union (Equation 12).

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{12}$$

PASCAL VOC metric To evaluate our detection model fairly, we adopted an evaluation system so-called VOC metric[13], which is being used in the PASCAL Visual Object Classes(VOC) challenge. Here, all the detections predicted as class A are taken as the positive detections of class A, but only the detections whose IoU is higher than the given threshold, usually 0.5, are counted as true-positive samples, while all other positive detections are considered as false-positive samples. Finally, the average precision rate of a particular class is calculated based on the amounts of true-positive and false-positive samples.

False-positive errors To find the reasons and further improve the performance based on the discovered insights, we introduce four categories of false-positive errors, localization error, confusion with similar objects, confusion with dissimilar objects and confusion with background, and run an corresponding diagnose for the detection results. Generally, localization error occurs when the predicted bounding box of the target object is misaligned, $0.1 < IoU < 0.5$, with its ground-truth box. The remaining false-positive detections overlapping, at least 0.1 IoU, with an object similar to the target object are considered as confusion with similar objects, while these overlapping with dissimilar objects are counted as confusion with dissimilar objects. Finally, all rest false-positive samples are categorized as confusion with background.

E.2 Detection Results

Figure 26: Bed room

13. PASCAL VOC challenge: http://host.robots.ox.ac.uk/pascal/VOC/

Figure 27: Living room

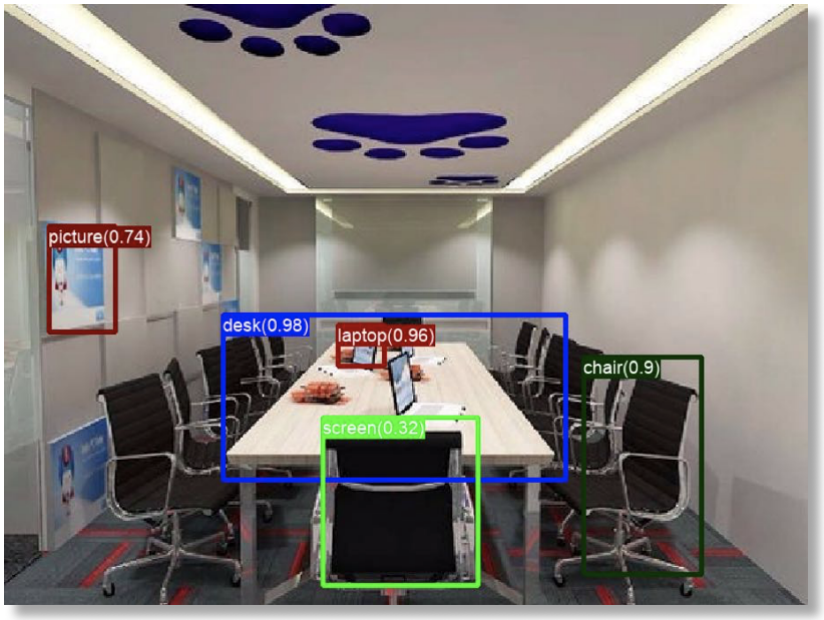


Figure 28: Office

Figure 29: Kitchen

Figure 30: Bath room

F. Final product demo

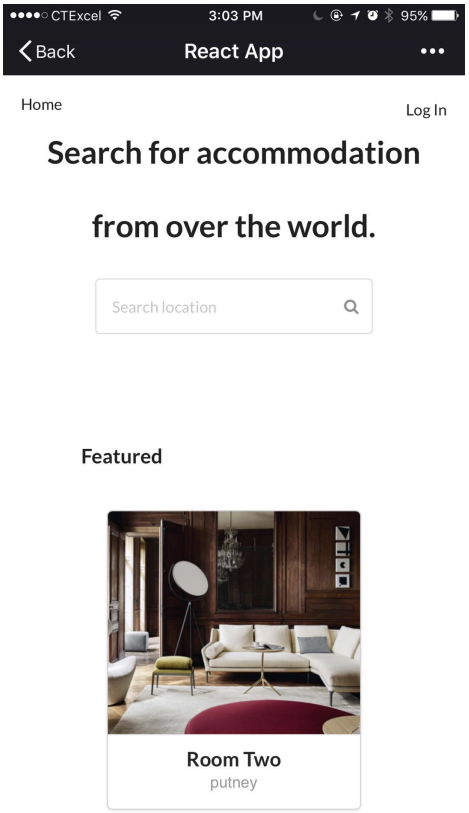


Figure 31: Mobile: Home page

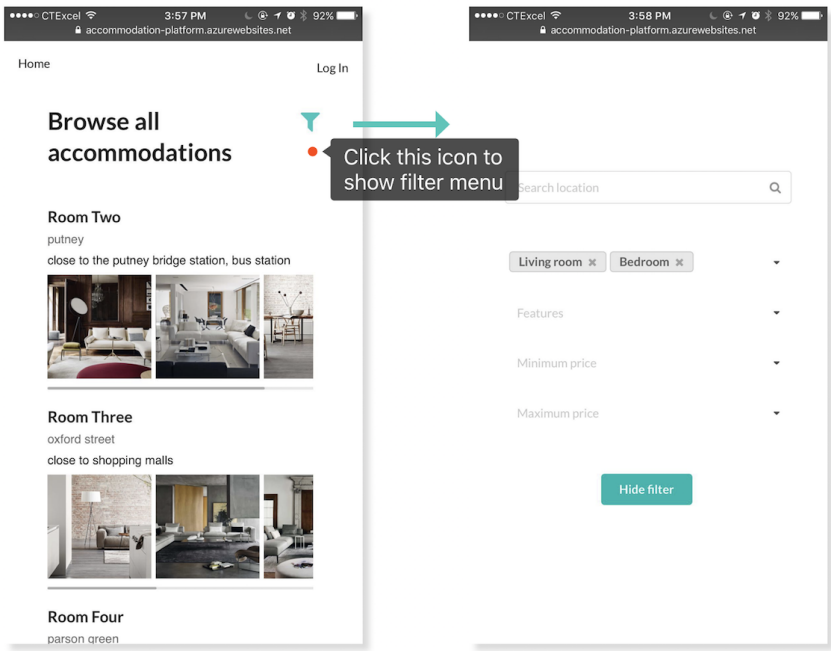
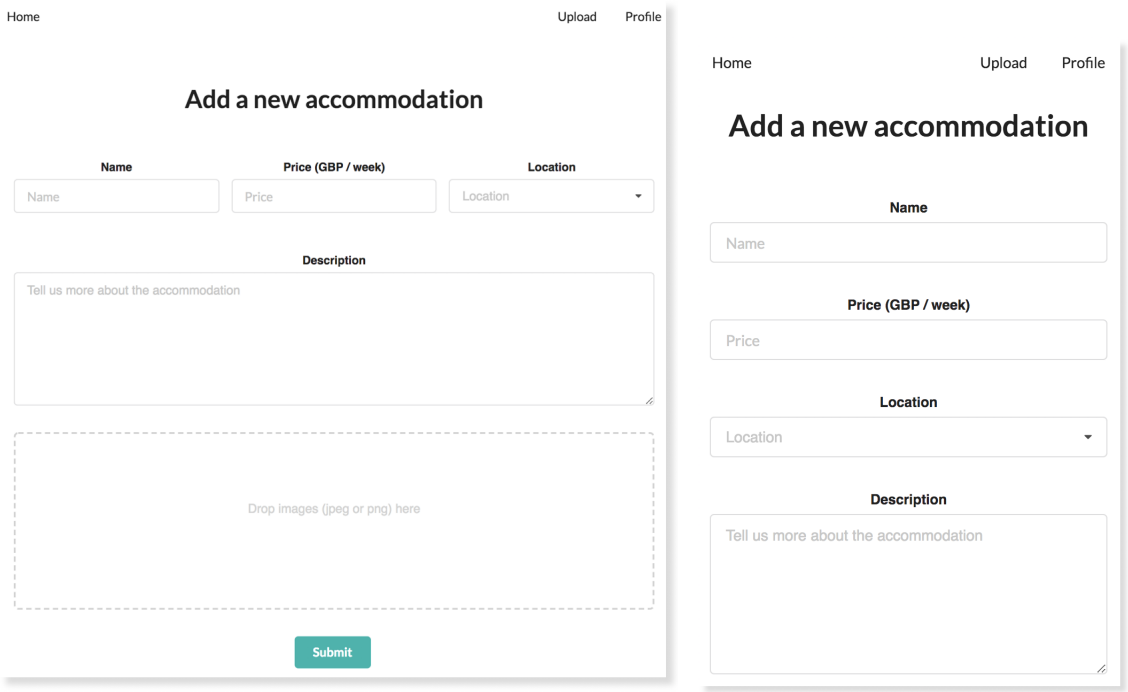


Figure 32: Mobile: Search result page



(a) Web: upload accommodation (b) Mobile: upload accommodation

Figure 33: Upload page

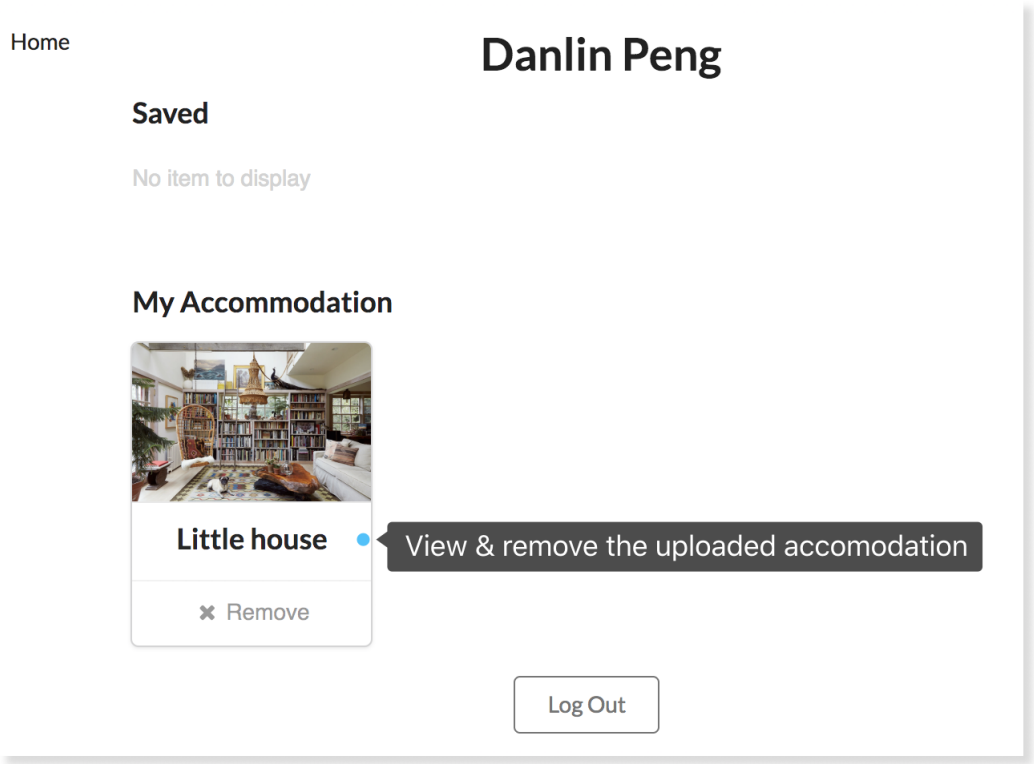


Figure 34: Web: Profile page

G. Meeting log

Date	**11/05/2018 11:00**	
Participants	Lin li, Yini Fang, Danlin Peng, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
	To do list	
No.	**Notes**	**Principals**
1	Update the current database: check tagging	All
2	Retrain model with new dataset	Lin
3	Website logo design	Yini
4	Report writing & revising: Tense, add table or pictures caption	All
5	Syntax check	Tomasz
6	Sharing the progress of the new website features implement	Nattapat & Suampa

Date	**17/04/2018 13:00**	
Participants	Lin li, Yini Fang, Danlin Peng, Suampa Ketpreechasawat, Nattapat Chaimanowong	
	To do list	
No.	**Notes**	**Principals**
1	Do data cleaning of the new dataset	Danlin
2	Draw the outline of the final report	Lin & tomasz
3	Hyper-parameter tunning	Lin & Yini & Danlin
4	New website feature	Lin
5	Design the front-end of the find me a similar room feature of the web application	Nattapat
6	Design the back-end of the find me a similar room feature of the web application	Suampa
	Work	
No.	**Notes**	**Principals**
1	Discuss about what hyper-parameter can be optimized and how to optimize.	ML group
2	Discuss about the implementation of the find me a similar room feature of the web application	Web group

Date	**10/04/2018 13:00**	
Participants	Lin li, Yini Fang, Danlin Peng, Suampa Ketpreechasawat, Nattapat Chaimanowong	
To do list		
No.	**Notes**	**Principals**
1	Utilize Bayesian optimization to tune hyper-parameter of the Faster-RCNN model	Yini
2	Tune anchor size of the Faster-RCNN model	Danlin
3	Debug	Lin
4	More precise and detailed evaluation of the model performance	Lin
5	Design the front-end of the find me a similar room feature of the web application	Nattapat
6	Design the back-end of the find me a similar room feature of the web application	Suampa
Work		
No.	**Notes**	**Principals**
1	Discuss about what hyper-parameter can be optimized and how to optimize.	ML group
2	Discuss about the implementation of the find me a similar oom feature of the web application	Web group

Date	**04/04/2018 15:00**	
Participants	Lin li, Yini Fang, Danlin Peng, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
To do list		
No.	**Notes**	**Principals**
1	Hyper-parameter optimize: learning the current hyper-parameter of the model	Danlin
2	Changing the base model: change the base model to ResNet	Yini
3	Debug: Try to improve accuracy rate of the class whose average precision is very low	Lin
4	Utilize more advanced model to speed up training and improve the accurate rate	Lin
5	Finish building the final classifier: Compare the performance of Neural Network classifier with Decision Trees classifier and choose a better one	Yini & Tomasz
6	Debug : fix the current problem of the website	Nattapat & Suampa
7	Prepare for the new web feature find me a similar room	Nattapat & Suampa
8	Set up overleaf document of the final report and then integrate the first and second report together	Danlin

Date	**29/03/2018 15:30**	
Participants	Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
To do list		
No.	**Notes**	**Principals**
1	Data collection and labeling (total 500 labeled images)	Yini & Suampa & Nattapat & Tomasz & Danlin
2	Take care of data organization ASAP	Danlin
3	Set up a report document on Overleaf	Danlin
4	Design a website logo	Yini
5	Decision Trees - architecture, model graph, tree visualization, accuracy metrics	Yini
6	Change, adjust object classes and update the object/features document	Tomasz
7	FFNN - finish, document graphs, architecture, confusion matrix, metric calculations, jupyter notebook	Tomasz
8	Change architecture from VGG to ResNet, compare accuracies, graphs	Lin
9	User authentication working on website, minor UI improvement, change header	Nattapat & Suampa
10	Find me a similar room features as POC	Nattapat & Suampa

Date	**09/03/2018**	
Participants	Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
To do list		
No.	**Notes**	**Principals**
1	Prepare for the jupyter notebook	ALL

Date	**27/02/2018 15:00**	
Participants	Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
To do list		
No.	**Notes**	**Principals**
1	Integrate the whole report together	Danlin
2	Finish the unit test	ALL
3	Prepare for the screen shot of the coverage report	ALL

4	Try to do a more detailed evaluation of the model performance (consider the UOI)	Danlin

Date	**23/02/2018 15:00**	
Participants	Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
To do list		
No.	**Notes**	**Principals**
1	Unit test for Decision Trees classifier	Yini
2	Unit test for Neural Network classifier	Tomasz
3	Evaluation of the model - Figure Generator	Tomasz
4	Unit test for Data preparation code	Danlin
5	Evaluation of the model: Confusion Matrix and related measures, e.g. precision/recall rates	Danlin
6	Unit test for Faster-RCNN model	Lin
7	Architecture drawings for the report	Lin
8	Try different base model: Train 6 networks with e2e/4stage - Alexnet/VGG16/VGG19 and prepare a 3x2 "efficiency/precision/evaluation matrix".	LIn
9	List possible hyper-parameters to improve network	Lin
10	Unit test for frontend	Nattapat
11	Unit tests for backend (do manual testing/find automated tools)	Suampa
12	Report on database structure, storing information etc.	Suampa
13	Architecture diagram (UI)	Yini & Suampa
14	Yini provides interface (explains how to use the decision tree) Nikki implements the classification on back-end.	Yini & Suampa

Date	**19/02/2018,13:30:00**	
Participants	Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
Discussion		
No.	**Notes**	**Principals**
1	Discuss about the architecture of Faster-RCNN	ALL
2	Discuss about testing implement	ALL
To do list		
No.	**Notes**	**Principals**

1	Discussion about how to deal with the non-binary vector in decision tree: Treat them as a binary vector, i.e the value of any attribute which is more than 2 is treated as 1; When calculating the cross entropy, add weights to it like a neural network	Danlin & Yini
2	Set the outline of the second report	ALL
3	Contact the MS to confirm if the program is tested or the available testing program can be provided.	Lin

Date	**13/02/2018 13:30**	
Participants	Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
To do list		
No.	**Notes**	**Principals**
1	Integrate ML model with web server	Suampa & Nattapat
2	Use tagging tool to expand current dataset	ALL
3	Try two possible classification methods: Neural Network & Decision Trees.	Tomasz & Yini & Danlin
4	Confirm the idea about classification model with Anandha	Tomasz
5	Optimizing the current objects detection model	Lin

Date	**08/02/2018 13:30**	
Participants	Anandha, Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
Discussion		
No.	**Notes**	**Principals**
1	Confirm the minimum requirement: detect the objects	ALL
2	Discussion about some possible extension: detect the space\Do the dimension detection	ALL
3	Discussion about feasibility: Time, how to do evaluation	ALL
4	Discussion about the possible application: 1. For disabled people: detecting space; 2.For blind people: use word to describe the picture and read it out; 3.Detecting special room: like the room with natural view	ALL

Date	**06/02/2018 13:30**	
Participants	Microsoft staff, Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
Discussion		
No.	**Notes**	**Principals**
1	Discuss about application of the machine learning model	ALL
2	Discuss about first report	ALL
3	Confirm implement details	ALL

Date	**02/02/2018 13:30**	
Participants	Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
Discussion		
No.	**Notes**	**Principals**
1	Discuss about the feasibility of this design	ALL
2	Discuss about the report for Microsoft: Use case, stakeholders, user experience	ALL
3	Data: Finding new data and applying to correct model	Lin & Tomsz & Danlin & Yini
4	Improve the accuracy and precision: Finding issues with current model, adjusting hyper-paramteres	Lin & Tomsz & Danlin & Yini
5	Front-end development	Nattapat
6	Back-end development	Suampa

Date	**29/01/2018 13:00**	
Participants	Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
Discussion		
No.	**Notes**	**Principals**
1	First report submission	ALL

Date	**23/01/2018 15:30**	
Participants	Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
Discussion		
No.	**Notes**	**Principals**
1	Separate team into two sub-team: Team a: Looking for efficiency improving solutions in current model; Team b: Enlarging database of pictures + retraining the model	ALL
To do list		
No.	**Notes**	**Principals**
1	Trying to run script on the VM with Python 3.5	Lin
2	Running a script ona separate VM (Python 3.5)	Tomsz
3	Read the code of the baseline model, do CNTK tutorials	Lin & Tomsz & Danlin
4	Provide detailed description of the web functionality: 1.Basic view of the GUI & functionaliy 2.Possibly allow the user for picture upload and give random results to mock functionality; 3.Design API	Suampa &Nattapat & Yini

Date	**19/01/2018 15:30**	
Participants	Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
No.	**Notes**	**Principals**
1	Attempt to run the script of baseline model	ALL
2	Setup the CNTK environment	ML group
3	Setup web application server	Web group
4	Confirm the outline of the first report	ALL
5	Draw Gantt diagram	Yini
6	Requirement identification & introduction	Nattapat
7	Feasibility & risk part	Tomasz
8	Development strategy	Suampa
9	Resource & boundaries & techniques & environment	Lin
10	Report integration	Danlin

Date	**12/01/2018 16:30**	
Participants	Anandha, Lin li, Danlin Peng, Yini Fang, Suampa Ketpreechasawat, Nattapat Chaimanowong, Tomasz Bartkowiak	
To do list		
No.	**Notes**	**Principals**
1	Confirm the meeting with Microsoft	Lin
2	Confirm the responsibility of two mentors	ALL
3	Confirm the frequency of meeting	ALL
4	Confirm the job assignments	ALL
5	Confirm the contact methods with microsoft	ALL
6	Confirm the requirements of project, specific implementation	ALL
7	Confirm whether we could public our project	ALL
8	Confirm the versions of CNTK and Python that we can use	ALL
9	Setup cloud drive	Tomsaz
10	Setup project management tool	Yini

Table 8: Meeting logs