Technische Universität München

TUM Data Innovation Lab

Sparkmap

Authors: Marsil Zakour, Sebastian Schlegel, Vladimir Yugay
Mentors: Lucas Spreiter, Oleh Melnyk, Stefan Reuther
Project Lead: Dr. Ricardo Acevedo Cabra
Supervisor: Prof. Dr. Massimo Fornasier

Submission Date: 07-02-2020

Abstract

We are given an empty floor plan blueprint image where each room has text annotation and want to place furniture icons inside it. The images have different texture, text annotations have different fonts and sizes, there is a random number of rooms and not all the rooms are strictly separated by the walls. Therefore, deterministic approaches are not applicable to this problem. In this work, we focus on generating the dataset simulating real world data, training neural networks for extracting and classifying rooms polygons and placing icons inside the rooms using context information of the floor plan.

Contents

1 Introduction and Objectives

In recent years AI has been successfully applied to multiple disciplines of engineering, improving workflows drastically and leading to better output. The Munich based Start-up "Unetiq" is a young company involved in this development, aiming to push the limits of engineering further using AI. This paper was done in close cooperation with Unetiq, working in the field of architectural engineering in specific. The overall goal of the company is to work towards an AI based system, being able to fulfil architectural tasks. This project is contributing to this goal by developing a solution for placing icons, like furniture or sanitary facilities, on an empty floor plan given as input. Empty floor plans are plans that only show walls, doors and windows. We describe our way towards this solution in this paper. For placing icons on a floor plan, it is necessary to know, of which room type each room is, e.g. if a room is a kitchen, a bath or a living room. We therefore look at related works at first, how the task of floor plan image processing has been handled in the literature so far. Next we describe the dataset, which we started with and how we adapted it to our needs. In the last chapter we explain the solution we developed before we close by presenting our results and drawing our conclusions.

2 Related work

Computer-based processing of floor plan images has been a well-researched topic in Pattern Recognition. Past approaches mainly rely on a sequence of low-level image processing heuristics, like extracting textual data, detecting lines and overcoming gaps through edge-linking or polygonal approximation [1] [8]. To increase the performance of processing floor plan images significantly, recent works have successfully applied deep convolutional neural networks (CNNs). In this chapter we will explore the works of [8, Liu et al] as well as [4, Kalervo et al] and will present their approaches as well as the regarding performance measurements.

2.1 Raster-to-Vector: Revisiting Floorplan Transformation

2.1.1 Approach

Liu et al. are applying a multi-step approach for automatically parsing rasterized floor plan images and recovering their lost geometric and semantic information. Figure 1 shows this pipeline.

In a first step, the input raster images are processed in a CNN. This network is a modified ResNet-152 where the last deconvolutional layer was dropped. Instead three deconvolutional layers in parallel were appended, each responsible for another output: For extracting geometric information, Liu et al. are using so called junction points. Junction points are vertices for polygons and endpoints for lines. Geometric information is concerning walls, icons (e.g. bathtub, sink, etc.) – both are extracted as polygons – and openings, which are extracted as lines. Openings are windows and doors and are only distinguished by their position being inside or outside the outer floor plan polygon. If an opening is inside the building it is considered to be a door, if it is outside, it is considered to be a window.

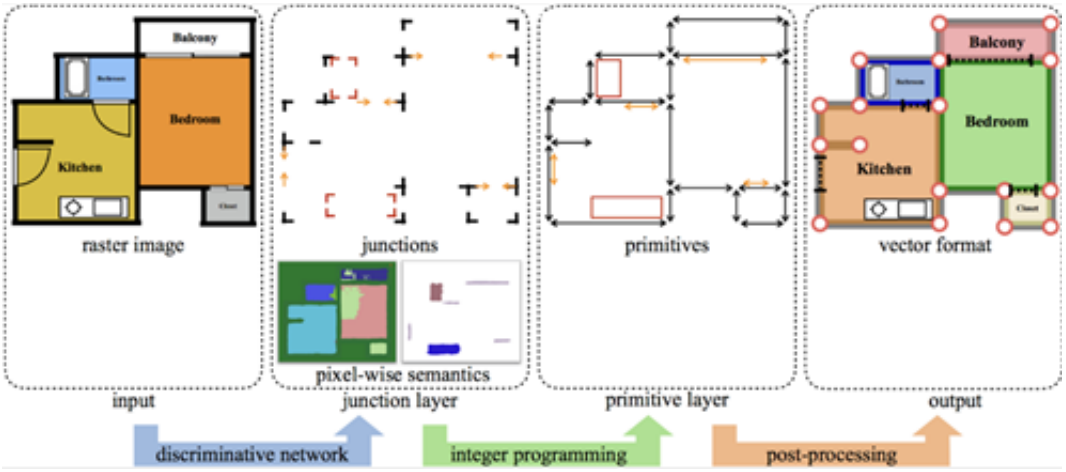


Figure 1: multi-step approach by [8, Liu et al.]

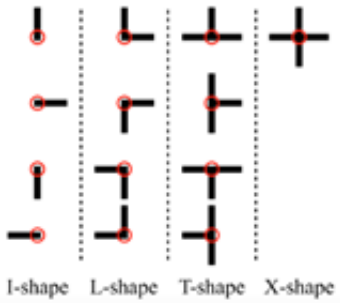


Figure 2: Showing all possible wall-junction points used by [8, Liu et al.]

Liu et al. are distinguishing 21 different junction points. Taking all possible orientations into account, there are four junction points describing opening-lines, four junction points for icons (icons are considered to be of rectangular shape) and 13 junction points for walls. (See Figure. 2) One of the three above mentioned deconvolutional layers is giving 21 heatmaps as output, one regressed for every junction type. Hereby pixel wise sigmoid cross entropy loss is used. It is important to note that this approach is not able to deal with any geometry not being horizontal or vertical. The other two deconvolutional layers are responsible for the semantic information. Each is giving a per-pixel classification as output, one for the room types and one for icon types. The icon classification is hereby also containing predictions for openings. Both classifications are using pixel wise SoftMax cross entropy loss.

At the beginning of the next step lines between two corresponding junction points, the so-called primitives, are generated. To filter out spurious candidates and to validate the generated primitives, integer programming (IP) is then applied. Therefore, Liu et al. are using five constraints. For example, an opening must be on a wall, meaning within a distance of ten pixels. Another constraint is contributing to rooms being of a consistent room type, describing that any connected pair of wall primitives must have the same room type on the inner side.

Method	Wall Junction		Opening		Icon		Room	
	acc	recall	acc	recall	acc	recall	acc	recall
Liu et al.	70.7	95.1	67.9	91.4	22.3	77.4	80.9	78.5
Liu et al. + IP	94.7	91.7	91.9	90.2	84.0	74.6	84.5	88.4

Figure 3: Accuracy and recall for [8, Liu et al.]

In a third step the output is post-processed. So far, room types were only associated with walls, one room type for each side. This was done according to the room type of the closest pixels on the respective side of the wall. At this step, Liu et al. are looking for closed wall polygons and are checking, if all corresponding walls have the same room type. If this is not the case, vertical or horizontal lines are drawn to split the room into subregions, each representing another room type.

2.1.2 Performance

Note that the model but not the dataset used by Liu et al. was available for us. Because of input restrictions we could not apply their model on available data and were therefore forced to use the performance numbers stated by themselves.

Figure 3 is showing the accuracy as well as the recall for junctions, openings, icons and rooms.

It stands out, that without applying the integer programming, the recall is significantly higher for junctions, openings and icons. This is indicating a much higher number of false positives than of false negatives. Through applying the integer programming, most of the spuriously detected geometries get filtered out and the accuracy for the respective elements is significantly increasing. Because every pixel in the image belongs to a room, there is not such a class imbalance for the rooms as for the other listed elements. Therefore, the above-mentioned effect of the integer programming is not as important for rooms. All in all, Liu et al. were introducing a promising approach to the field of automatic processing floor plan images using CNN's. For geometries, for junctions and openings respectively, they were achieving an accuracy above 90%. For icons and rooms, the accuracy was about 84%. Their network is able to detect nine different room types and seven different icon types. The approach of dealing with highly imbalanced classes through aiming for a high recall for the network output and increasing the accuracy through some further processing is also promising.

Method	Wall Junction		Opening		Icon		Room	
	acc	recall	acc	recall	acc	recall	acc	recall
Kalervo et al.	82.4	92.0	82.3	93.3	34.6	88.3	90.0	87.6
Kalervo et al. + FP	95.0	89.7	94.5	92.9	93.6	87.3	92.2	90.2

Figure 4: accuracy and recall for [4, Kalervo et al]

2.2 CubiCasa5k: A Dataset and an Improved Multi-Task Model for Floorplan Image Analysis

2.2.1 Approach

As the title of [4, Kalervo et al.] suggests, their contribution to research is twofold. On one hand, they are presenting and publishing a new dataset, containing 5000 annotated floor plan images. This will be the subject of the chapter "dataset". On the other hand, Kalervo et al. were working on the approach of Liu et al. adjusting their CNN and increasing their performance.

The main contribution of [4, Kalervo et al.] lies in applying automatic weighting to the CNN introduced by Liu et al. While the latter were adjusting the weights of their multi-task CNN by hand, for Kalervo et al. the relative weighting between the multi-task losses was learned automatically, applying concepts from [1]. As well as [8, Liu et al.], Kalervo et al. are also only able to deal with geometries being either horizontal or vertical.

[4, Kalervo et al.] also chose another approach for further processing the network output. Instead of applying integer programming on predefined constraints, they were using a heuristic to combine geometric with semantic information. For rooms, the floor plan was split up in a grid of multiple rectangular cells. Each cell was formed by a triplet of junction points spanning a rectangular area not containing any other junctions. The resulting cells were labelled applying pixel-wise maximum voting based on the segmentation map. Two cells were then merged if they share the same label and if no fully separating wall was in between them.

2.2.2 Performance

To compare their performance with [8, Liu et al.] [4, Kalervo et al.] were using the same dataset for evaluating their approach. Figure 4 is showing the corresponding accuracy and recall as stated by Kalervo et al. themselves, once with applying further processing (FP) and once without Compared to Liu et al. the accuracy of the network output of Kalervo et al. without applying any further processing is significantly higher. After applying further processing steps, Kalervo et al. are also showing better performance, especially

for the accuracy of icons and rooms. Note that on this dataset Kalervo et al. are showing the same overall pattern having a higher recall for the network output and increasing the accuracy then through further processing it.
All in all, Kalervo et al. were pushing the approach of Liu et al. even further. Because of higher performance and the availability of the corresponding dataset, we decided to use the approach of Kalervo et al. as basis for our work.
The code of both presented works was available for us through public Github projects [3][7]. Both projects are written in the programming language Python and are using frameworks from Pytorch.

3 Dataset

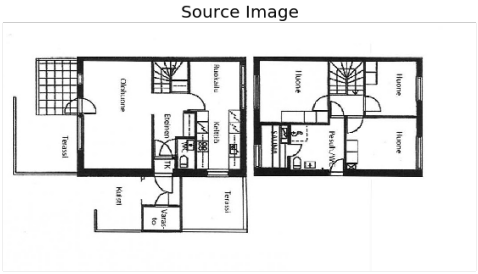
3.1 Problems with existing dataset

CubiCasa5k is known as the largest labelled dataset for floor plans. However, it doesn't align with the original goal of our project: place furniture icons inside the floor plan image.
There are two main problems with CubiCasa5k, first - is that a big part of the pretrained model provided by CubiCasa team heavily relies on text annotations on the input images (kitchen, bathroom, etc.) written in Finnish language. The second - is that there are no labels for the training of our own models for detecting room types. Moreover, the intermediate labelled data for inner segmentation and hand labelled svg images are not available publicly. Only things available are trained models for converting jpg images to svg, outputs of the network and the input images themselves.
As mentioned in the previous section, CubiCasa model comprises two parts: Segmentation network (which split an input image into separate colored regions where each color meant a certain room type) and optimization part, where room joints were classified and heuristics to improve visual results applied. However, it was shown experimentally that the segmentation network heavily relies on text and will not recognize images without Finnish language annotations.

3.2 Label Generation

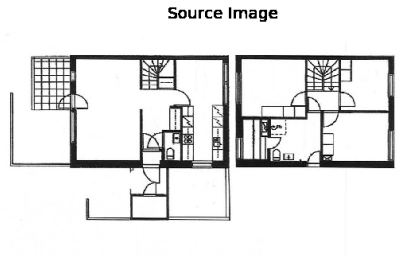
To classify a room, it was planned to use semantic segmentation. Semantic segmentation of an image means to classify each pixel separately and to color it correspondingly to the room type. However, semantic segmentation pre-trained network from CubiCasa is not applicable, since it relies on Finnish language annotations. Moreover, in order to train a completely new segmentation network on the data we expect to see in the future, segmentation labels are necessary. Labels were obtained by reverse engineering of the original work:

- the svg images and their inner representations in the CubiCasa codebase were parsed
- number of rooms types was reduced from 65 to 8 according to business requirements
- the corresponding color map was generated for each of the room types



(a) source image with text

(b) prediction with text



(c) source image without text

(d) prediction without text

Figure 5: CubiCasa segmentation on image with and without text

```
2019-11-08 15:16:43,358 - eval - INFO - IoU & Acc
2019-11-08 15:16:43,358 - eval - INFO - Background & 88.9 & 95.6 \\ \hline
2019-11-08 15:16:43,358 - eval - INFO - Outdoor & 56.2 & 74.0 \\ \hline
2019-11-08 15:16:43,358 - eval - INFO - Wall & 65.5 & 72.9 \\ \hline
2019-11-08 15:16:43,358 - eval - INFO - Kitchen & 63.9 & 85.5 \\ \hline
2019-11-08 15:16:43,358 - eval - INFO - Living Room & 73.1 & 84.8 \\ \hline
2019-11-08 15:16:43,359 - eval - INFO - Bedroom & 76.2 & 93.5 \\ \hline
2019-11-08 15:16:43,359 - eval - INFO - Bath & 56.8 & 63.7 \\ \hline
2019-11-08 15:16:43,359 - eval - INFO - Hallway & 57.6 & 75.0 \\ \hline
2019-11-08 15:16:43,359 - eval - INFO - Railing & 13.3 & 13.9 \\ \hline
2019-11-08 15:16:43,359 - eval - INFO - Storage & 53.4 & 57.2 \\ \hline
2019-11-08 15:16:43,359 - eval - INFO - Garage & 0.0 & 0.0 \\ \hline
2019-11-08 15:16:43,359 - eval - INFO - Other rooms & 49.7 & 64.4 \\ \hline
2019-11-08 15:16:43,359
```

```
228 2019-11-08 15:05:33,043 - eval - INFO - IoU & Acc
229 2019-11-08 15:05:33,043 - eval - INFO - Background & 80.3 & 95.8 \\ \hline
230 2019-11-08 15:05:33,043 - eval - INFO - Outdoor & 50.3 & 66.0 \\ \hline
231 2019-11-08 15:05:33,043 - eval - INFO - Wall & 64.0 & 72.0 \\ \hline
232 2019-11-08 15:05:33,043 - eval - INFO - Kitchen & 50.6 & 68.0 \\ \hline
233 2019-11-08 15:05:33,044 - eval - INFO - Living Room & 24.8 & 28.4 \\ \hline
234 2019-11-08 15:05:33,044 - eval - INFO - Bedroom & 37.0 & 45.0 \\ \hline
235 2019-11-08 15:05:33,044 - eval - INFO - Bath & 8.4 & 9.0 \\ \hline
236 2019-11-08 15:05:33,044 - eval - INFO - Hallway & 21.2 & 23.8 \\ \hline
237 2019-11-08 15:05:33,044 - eval - INFO - Railing & 12.1 & 12.9 \\ \hline
238 2019-11-08 15:05:33,044 - eval - INFO - Storage & 5.3 & 5.3 \\ \hline
239 2019-11-08 15:05:33,044 - eval - INFO - Garage & 0.0 & 0.0 \\ \hline
240 2019-11-08 15:05:33,044 - eval - INFO - Other rooms & 33.3 & 64.8 \\ \hline
241 2019-11-08 15:05:33,044
```

Figure 6: Evaluation of segmentation on image with (left) and without (right) text.

- images were plotted and the plots were saved
- images were reshaped proportionally with nearest neighbor interpolation to avoid mixtures of colors

3.3 Input Generation

Since we couldn't use input images in CubiCasa dataset because of Finnish annotations and we wanted to build a semantic segmentation network we decided to work with empty floor plans. An empty floor plan is defined as an image with only walls on it. In order to get it, the same approach as for label generation was used but with only black and white colors.
After running several experiments, it turned out that the network is not able to recognize room types on empty floor plans. Even for a professional this task is still quite challenging. In order to give a chance to the segmentation network, we enhanced the dataset with additional information. The input image would contain a floor plan with room annotations in English language.
However, to simulate real world environment several enhancements were made:

- Each input image has a different texture to simulate different types of paper. The background color is covered with black of intensity from uniform distribution in 0 - 0.5 range
- Each text annotation on the image has a random font size, weight and family
- Each text annotation is rotated by uniformly chosen angle from 0 to 11 degrees

In addition, the door opening icons were also generated for each of the door openings. A quarter circle of door width radius was attached under different angles to all the doors openings in the input image.
Furthermore, the input images were resized using Lanzcos interpolation technique, unlike the labels. This was done in order to keep the text annotations in the best possible quality.

Finally, the dataset comprises 1881 images with 1281 images in the training set, 300 in the validation and 300 in the test set.

4 Feature Vector Approach:

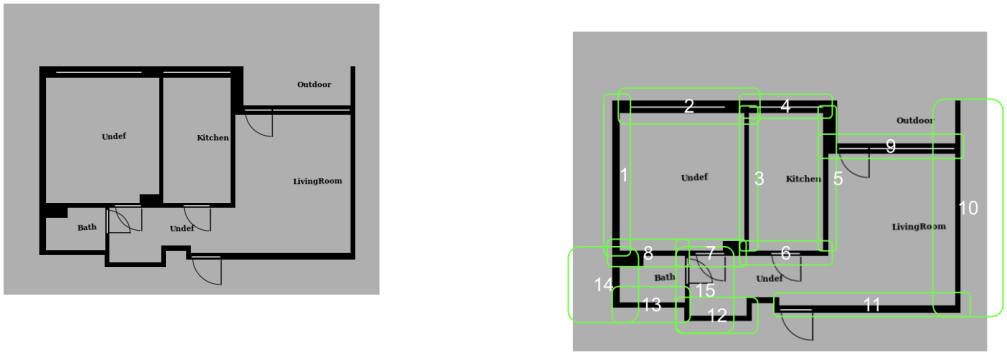
To find the room proposals and types we splitted our research directions into two different approaches at the same time. The first one aimed to create feature descriptors out of the geometrical elements in the floorplan diagram then using these features to classify room type using a neural network. The second one was an end-to-end approach which feeds the raw image pixels to the neural network. In this section we discuss the feature vector approach since we stopped working on it later and decided to go with the end-to-end approach (which is explained in the solution section).
The main idea behind this approach is to make use of the feature points extracted by the CubiCasa network. Since our quantitative tests 6 have shown that elements like walls

Figure 7: Example of the input and target images

predicted by the network are not dependent on the existence of text or text language.
We make use of the first 21 layer output of CubiCasa network which are predicting the junction points for walls and icons, and then these junction points are post-processed using some heuristics to create the rooms and icons polygons.
We build a Graph $G(V, E)$ where each node $v \in V$ either a room or wall, and we add an edge $e \in E$ connecting a room node and a wall node if the wall is part of the room and an edge between any two wall nodes if their polygons are intersecting or touching each other (Figires 8, 9 are an example building such a graph). For each wall we create a feature descriptor consisting of features like:

- width
- height
- aspect ratio
- number of connected rooms
- number of connected walls
- closest room distance
- furthest room distance
- has and opening/does not have an opening
- horizontal/vertical wall

To build the walls feature descriptor we extract the wall polygons which are the output of the post processing step in cubi casa pipeline. To create room nodes we assume that each room is a closed polygon - given that doors and windows are defined as openings inside the walls, so they do not cause a discontinuity of the walls. We detect cycles of wall nodes in the graph such that the cycle walls form a valid closed polygon.



(a) Floor plan diagram. (b) Wall nodes highlighted with green.

Figure 8: Room walls highlighted

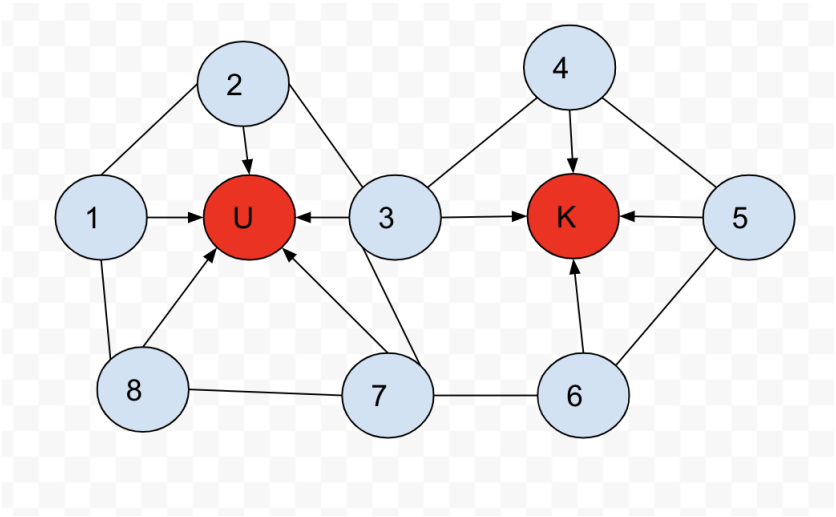


Figure 9: Graph representation for Kitchen and Undef room in Figure 8

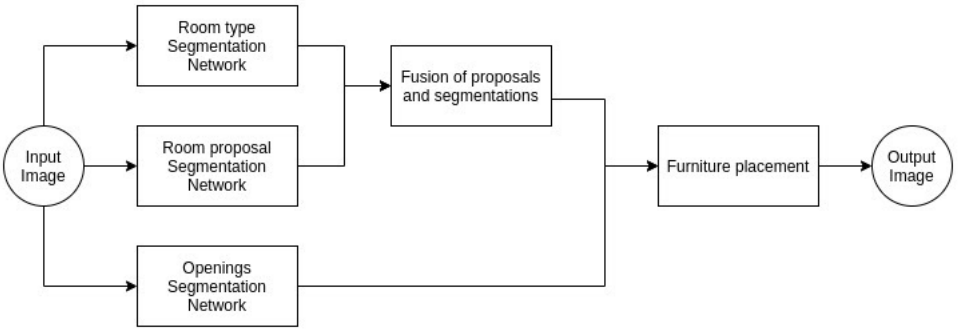


Figure 10: Pipeline structure

The feature vectors of the walls could be aggregated or concatenated into a room descriptor and the learning could be done using a fully connected neural network, or another option would be to use Graph Convolutional Neural Networks [5]. An intuition behind this choice is that the room is a polygon and we can learn some features like polygon area and polygon perimeter from its edges (room walls) which will help us determine the room type. And a more generalized descriptor could be learned using GCNNs[5] for each room node. However, due to the difficulty of detecting rooms using cycles only, since some rooms may form open polygons, and the high risk of implementing GCNNs[5] within the time limit of the project, we decided to stop working on this approach after the graph building step.

5 Solution

The problem statement is to place icons on the floor plan images with subscripted rooms. The problem itself can be splitted in two parts: first to detect the room polygons and their types, as well as the openings and their types on the image and second to utilize this information to place icons in each of the polygons.

The pipeline itself consists of several stages. First, a neural network processes an input image and produces a segmentation map of the floor plan which is the same floor plan where each room is filled with a color inherent to its type. Simultaneously, another network generates rooms' regions proposals out of the input image. Room proposals for one room are a set of filled polygons which describe the location and position of this particular room. After that rooms proposals and rooms type segmentations are fused to get classified polygons of each of the rooms. Further, another neural network detects and classifies openings in the input image. On the final step, the icon placing algorithm takes the classified polygons and openings and places icons on the floor plan based on this information.

Figure 11: Example of the input and target images

5.1 Segmentation Network

Since the network which was trained in CubiCasa paper is not suitable for segmenting newly generated input (Section 3.3), it was necessary to develop an accurate segmentation network.
The goal of semantic segmentation is to label each pixel of an image with a corresponding class. Each class is represented by a certain color. For instance, on Fig 11 dog has a yellow color, chair - blue and everything else is red.

There are many architectures of neural networks which allow doing good segmentation, however, most of them are trained on real-word images. Our dataset mostly consists of polygons and text inside them. The shapes are much more simple, but the information with which the network is able to learn is scarce. For example, the shape of a bathroom on a floor plan is rectangular in most cases, but the shape of a dog, like on Figure 11, can be diverse. However, in order to classify a dog, a network can take into account eyes, ears, tails and other important features, while for our case only text subscription and boundaries of an object are available.

The basic approach to do the segmentation was to do transfer learning with upconvolutions. Transfer learning is the process, where a pretrained network is used to get a feature vector or a representation of an input image. After that, the representation of the input image is upsampled or upconvoluted to the size of the original image. Finally, each pixel is classified. The advantage of transfer learning is simplicity and no need to train a network from scratch.
During the first iteration of segmentation, input images did not contain any text subscripts. Various architectures and techniques were applied to get the segmentation working: encoder-decoder architecture with upsampling, uplconvolutions, sliding convolutions [11], and U-Net [9] architecture. All of the approaches were tested with the following encoder networks: AlexNet [6], VGG16[10], VGG19, ResNet18[2], ResNet50 as encoders

with pre-trained and default parameters. Nevertheless, the results were not satisfactory. All the networks failed to interpret features of the image. Only black and white rectangles do not contain necessary information for the network to learn. Even for a human, segmenting an empty floor plan blueprint would be a very difficult task with no correct answer.

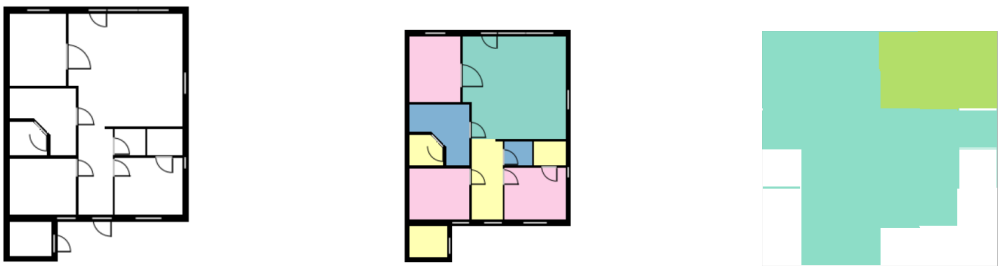


Figure 12: Example of bad segmentation: input, target, result

We decided to add more information to the training images in order to give some hints to the network. Particularly, we added a textual description of the room. For instance, a kitchen has "Kitchen" written on it. However, each of the floor plans has a different background structure, different font style and size and this makes it impossible to detect room types by some deterministic approach.

Adding text imposed additional problems. The original size of the images is 720 by 720 pixels. When there was no text information on them, we could easily resize them to 240 by 240 or to 320 by 320. With our previous shrink strategy, after resizing text information was almost lost and the network was unable to distinguish one text subscription from another. In order to overcome this, a special strategy needed to be applied to shrink the image. Out of linear, bilinear, quadratic, cubic, bicubic and lanzcos, lanzcos was chosen by comparing empirical results. To shrink the labels of the dataset, nearest neighbor strategy was used in order not to introduce new colors after shrinking.

Another strategy to overcome resizing images problem was to try fully convolutional networks and sliding convolutions. Fully convolutional networks utilize 1 by 1 convolution windows and thus can process images of any size. However, results were not satisfactory, because the convolution window was too small to also capture the region around the text to properly segment it. In the sliding convolution approach, a sliding convolution is applied to the original image to get a representation of size which is acceptable by transfer learning encoding network. While this approach was working, it was not definitely the best and was very expensive computationally.

The architecture which showed the best results was a U-net with ResNet18 as the encoder. The idea to use such a setup was based on several facts. First, U-net architecture uses previous information to navigate itself further. This is extremely important for us, since after seeing a region of the room with a piece of text, for example "Kit", the network should understand that meeting in "chen" in a nearby region it should color both of them in the same color. Moreover, ResNet as the encoder allows us to train a really

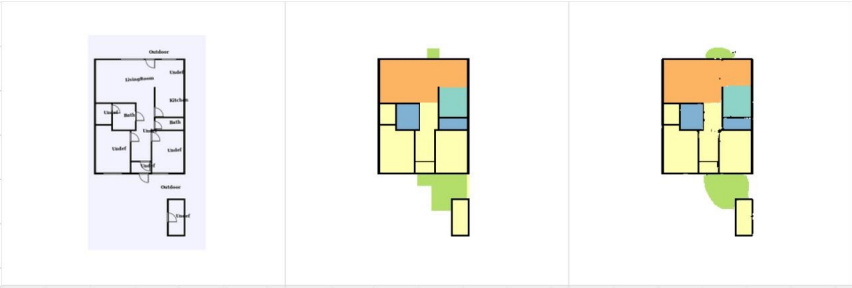


Figure 13: Example of the input and target images

deep model without vanishing gradient problem which we faced during training simple transfer learning models. In addition, the model contained 25 million parameters, which proved to be enough to learn simple shapes. Finally, we trained the encoding part of the network from scratch. This was done because ResNet18 was pre-trained on images from ImageNet[6] and our images are completely different from them.
Another problem faced during segmentation is the loss function. Cross entropy loss didn't work well, because classes were very unbalanced. There are always more undefined pixels than defined in the training set since we kept only 8 room types out of 65 and marked all others as undefined. Moreover, even using a weighted variant of cross entropy loss didn't lead to any significant improvements. After several experiments a dice loss was chosen, since it was designed for unbalanced classification and gave really good results.

5.2 Room Proposal Extraction

5.2.1 Problem Description

The objective is to extract a list of proposals each of them represents a room, each proposal could be a pixel mask stating which pixels in the image belong to the room, or polygon surrounding the room.
There are two challenges with this task. First one is the unknown number of rooms on each floor plan. Secondly, there are some rooms that are not separated by a physical wall, for example a living room and a kitchen could have no wall in between them.

5.2.2 Existing Solutions:

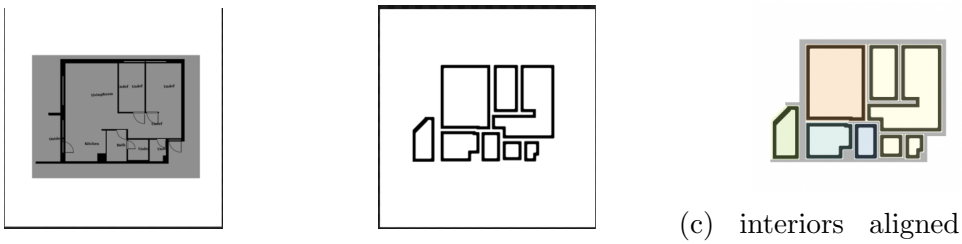
In the literature this problem is handled in different ways. In thier approach [8] which we discussed in section 2.1 use linear programming to find the room polygons, while authors in [4] divide the floor plan into a grid of rectangles and merge these rectangles using the room type segmentation information. In both cases, junction points and room type segmentation information is used for getting the room proposals.

5.2.3 Our Approach:

We tried multiple approaches to solve this problem. One of them was to assume that room is a cycle in the graph (which we discussed in section 4).

To do so we define the interior of the room to be a polygon touching the inner side of the room polygon. The reason behind choosing the interior and not the walls of the room is to have a set of disjoint interiors for each room (since walls may be shared by multiple rooms). This way we can have a segmentation network to solve this problem with two classes per pixel (interior, or none-interior). Figure 14 shows an example about such interior and its alignment with the room type segmentation.

We tried multiple models which are based on Unet segmentation network [9] architec-



(a) floorplan image (b) rooms interior (c) interiors aligned with room type segmentation

Figure 14: Room Interior Allignment

ture. Basically, we tested Resnet [2] 18 layer, and Resnet [2] 50 layer implementation , as a downsampling part of the segmentation and found that resnet 18 gives good enough results. For the upsampling we tried upsampling using interpolation (without learnable weights), and with convolution transpose (with learnable weights) and the added weights did not contribute so much to the result.

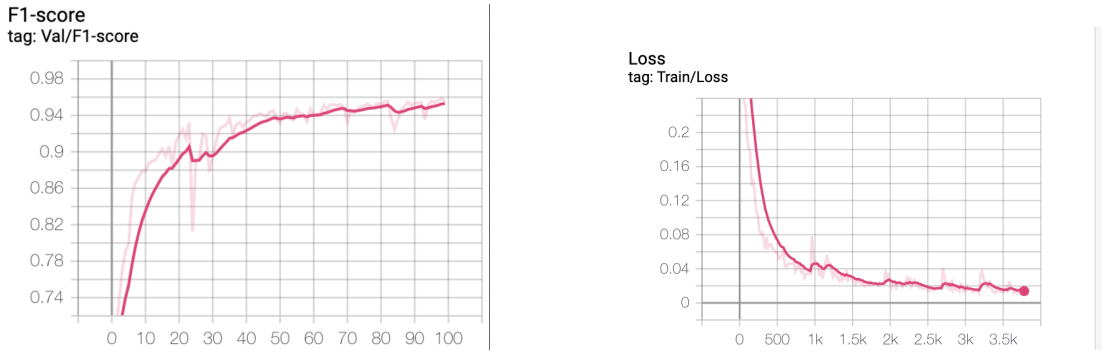
An important observation we found is that fully convolutional networks perform worse than networks with at least one fully connected layer. This could be because the fully convolutional networks do not maintain spatial coherence, in other words, a fully convolutional network does not distinguish between the left area or the right area of the image, causing a loss of a valuable information. For example outdoors are always on the edge of diagram.

Another problem is the unbalanced distribution of our classes since the none-interior pixels number will be always much higher than interior pixels. This makes the traditional cross entropy loss function a bad choice since predicting all pixels to be none-interior provide a very good loss score, and the network may not get out of this local minima. To solve this we tried two alternatives:

- Weighted cross entropy: which is similar to normal cross entropy but weights the classes, as a result we can control how important a misclassification in one class is, we found that using 0.95 weight to none-interior class and 0.05 to interior class results a better classification, and faster learning.
- F1 Loss function: which basically 1-(F1 score). And it works because F1 score is a valid accuracy measure even in unbalanced class distribution.

In the end we found that the weighted cross entropy provides equally good results and maximizes the F1 - score on the validation set. We choose it over F1 loss because it has a smoother derivative. Training is done for 100 epochs and the metric we track is F1-score

on validation set. The training loss and validation F1-score graphs are shown in Figure 15 and a qualitative results from validation set are shown in Figure 16



(a) Training Loss (b) Validation F1-score

Figure 15: Training Metrics

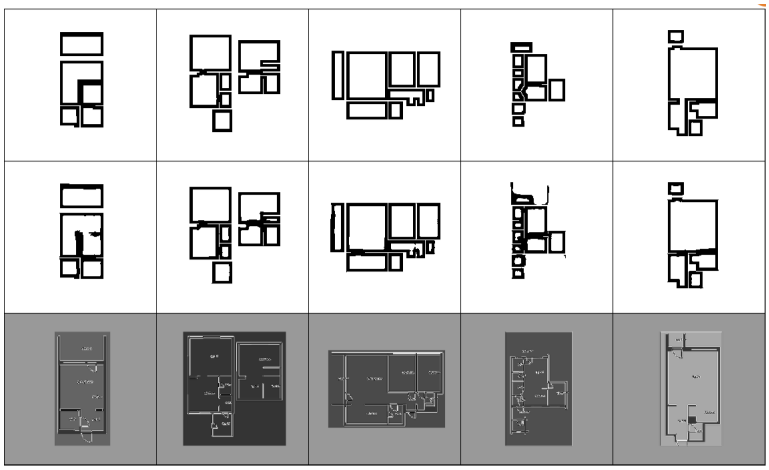


Figure 16: Room proposal network predictions on validation set (first row) ground truth, (second row) prediction, (third row) input

5.3 Predictions Fusion

The predictions of the room proposal and room types networks need to be merged into one where for each room proposal we know the type of the room. With the absence of a wall separating some types of rooms we cannot rely only on the room types segmentation for example having two adjacent undefined rooms with no wall in between will form one continuous yellow (color code for undefined room) component using room type segmentation only.

We find connected components on the proposal segmentation network and inside each component we do a majority voting to determine the room type. This is based on two intuitions which would hold for real life data and synthetic data:

- The predictions of the types-network will become uncertain close to the edges, especially if there are no actual walls separating the rooms or the text location is far away from the room walls
- The case of two adjacent rooms with the same type makes distinguishing them from types only impossible. In our synthetic data set this is happening because of two adjacent rooms of type undefined. In real life data -with more room types- this would happen additionally because of two adjacent bathrooms, or bedrooms for example.

5.3.1 Extracting Polygons out of the Proposal Masks

To approximate the fused mask into the closest polygon shape, we fit a decision tree into the proposals predictions, where we consider the spatial coordinates x,y as features that the decision tree should split the predictions by. As a result each leaf node in the decision tree will contain a rectangular bounding box which most of its pixels belong to one room type/proposal. We merge the rectangles of the same proposal to get a polygon that approximates the proposal pixels shape we have. Figures 17d, 17e are showing how such the proposals are enhanced to be more like polygons.
Note that the resulting polygons will be only axis aligned because splits are over X and Y axes, which is similar to Cubicasa [4] resulting polygons but utilizing a different approach.

5.3.2 Predictions Fusion Pipieline Steps

Our pipeline for fusing the results works as follows:

1. Align the room proposals, with the room types prediction.
2. For each proposal, do a majority voting on the room type for the pixels belonging the same proposal.
3. For the interior pixels assign them to the type/proposal of closest room type/proposal (using Manhattan distance which is sum of the absolute difference between the two pixels rows and columns)
4. To get polygon like shapes, run the Decision Tree fitting step on top of the proposals integer masks from the previous step.

The Figure 17 show the output during the Predictions Fusion process. Here the ensemble (step number 2) step takes as input the proposals and types, and after that an inpainting happens (step number 3). Finally, the polygons from the decision tree output (step number 4) are used to inforce the polygonal shape of the room predictions and the output final output is the refined proposals.

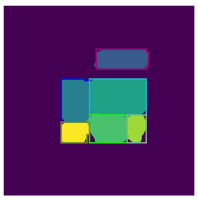
5.4 Pre-processing for placing icons - openings segmentation

In order to place furniture in a room wisely (in our case we can rephrase it as putting icons in the room polygons) knowing where a window or a door is, is extremely important. For example, nobody would place a sofa right in front of the door. Additionally, distance

(a) Input image

(b) Proposals Prediction

(c) Types Prediction



(d) Proposals Polygons

(e) Refined Proposals with Types

(f) Ensemble and Inpaint

Figure 17: Predictions Fusion Pipieline

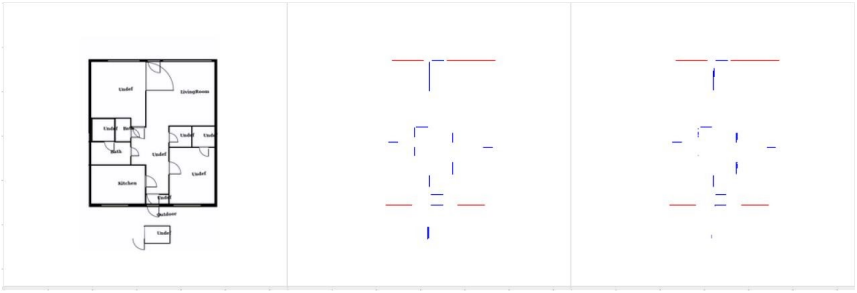


Figure 18: Example openings segmentation: input, target, result

from windows and doors was heavily used during placement of the furniture.
To get precise locations and distinguish doors from windows, we decided to use an approach which already showed very good results in segmenting rooms. The approach is to segment doors and windows in different colors. In the input data there are quarter circles near the doors and windows are marked with thin stripes. We assumed that this information is enough for the network to effectively distinguish and classify those openings.
To generate labels, for rooms and doors segmentation original images were plot and windows and doors openings were colored in red and blue correspondingly. However, there appeared color mixtures after plotting which resulted in lighter or darker options for red and blue, while for proper training a limited number of colors is needed. To overcome this, the maximum coordinate was used to replace color mixtures with exact colors. For example, [255, 55, 67] is converted to [255, 0, 0] since the first coordinate is maximum. In case white color, pixels were not changed.
UNet with ResNet18 as the backbone was used and retrained from scratch. Moreover, the choice of the loss is also quite similar to the one from rooms segmentation since we're still dealing with highly unbalanced classes.

5.5 Algorithm for placing icons

The task of placing icons on a floor plan is rule-based. For example, no icon should be placed in front of a door and most icons are close to a wall or even in a corner. Therefore, we decided to use a rule-based algorithm for placing icons instead of training a generative network for this task to get good results in a reasonable amount of time.

5.5.1 Extracting rooms from prediction

The input of the algorithm are two masks, both being a per-pixel representation of the floor plan image. One is giving the information, if a pixel belongs to a window or an opening. The other one is containing the information, if a pixel belongs to a specific room. Each pixel is carrying a number, either zero for being a wall or being outside the floor plan, or a unique number given for every specific room. In addition, a file is used for providing the information, of which type each room is.

To allow for independently working on a room and therefore for parallelisation, each

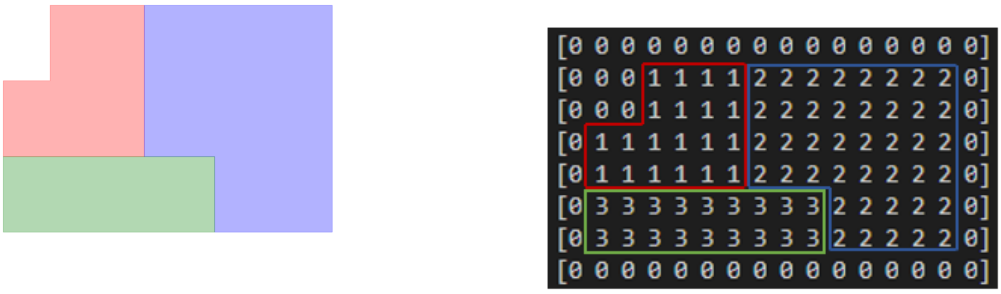


Figure 19: simplified floor plan image and corresponding mask representation

room is extracted from the input mask in a first step. This is done by iterating over all unique room numbers. If the value of a pixel is different from the respective room number, the value is evaluated to zero. For each room a new mask with adjusted size is saved.

5.5.2 Assigning information to pixels

Each pixel is carrying different values, together representing the information used for placing an icon. These values are shown in Figure 20 To achieve the same results for any given floor plan, the corresponding scale must be estimated. Unless the scale is explicitly stated, we are using the average door width of a floor plan as an estimator for scaling. Here we assume that most doors are of the same width.

A pixel being right next to an object (e.g. a wall) is assigned the corresponding maximum value. Each pixel being orthogonal to the orientation of the respective object is assigned the maximum value minus the smallest distance. The resulting value is either positive or zero. For doors the maximum value is two times the average door width. This means

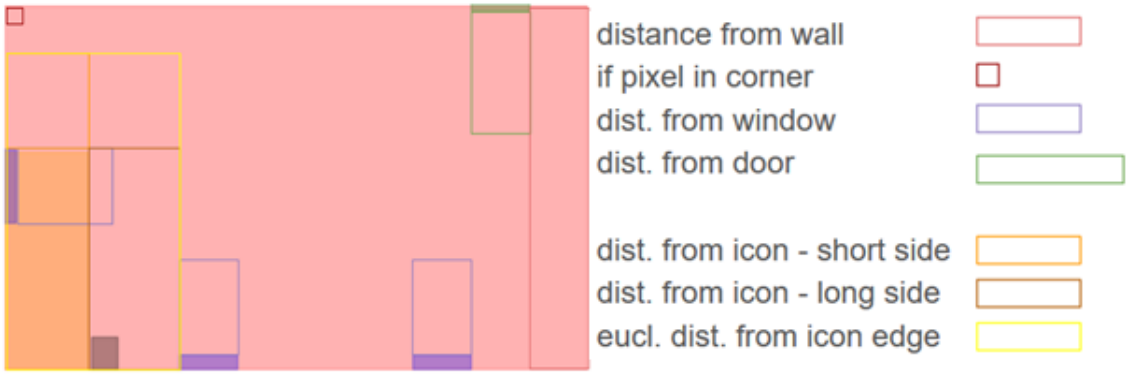


Figure 20: Information assigned to pixels

that all pixels being in the pathway of a door and less than two door widths away get punished. For all other elements the maximum value is one and a half times the average door width.
Through iterating over all walls, windows and doors the affected pixels get assigned the corresponding value, for all other pixels the value is zero.
If an icon gets placed, the respective pixels get assigned a number indicating the icon they belong to. All surrounding pixels get assigned a value describing their distance to the icon. Because all icons are assumed to be of a rectangular layout, we are distinguishing between the smallest distance of a certain pixel to either the long or the smaller side of an icon. Taking a bedside table as an example, it should be placed next to a wall and next to the small side of a bed. Furthermore, we are also using the Euclidean distance to the edge of an icon.
The result of this step is a value vector for each pixel, containing its distances to the mentioned objects.

5.5.3 Rules for placing icons

The rules for placing icons are expressed through a weighting vector for each icon. Each icon has at least four weights, describing its relation to walls, corners, windows and doors. Furthermore, icons can have weights describing their relation to other icons. Weights can either be positive (minimize distance), negative (maximize distance) or zero (ignore). So far, we can only maximize or minimize the distance of an icon to a certain object. For setting specific distances, e.g. a certain distance to another icon, our algorithm allows for defining constraints.
At the beginning, these weights were set according to common sense. Through a supervised-learning process, the results got evaluated and the weights manually adjusted. This could be further automated in the future.
To achieve certain constellations of icons, icon-dummies can be used to define more complex relations. Icon-dummies are icons only used for the placing process and are neither part of the output nor visualized. In a simple example they allow for a free corridor between the long side of a double bed and the nearest wall, where only certain icons like e.g. bedside tables can be placed.

The user is giving the icons he wants to place in a certain room as input to the algorithm. The order in which the algorithm is placing the icons can have a high influence on the result. Therefore, rules for the order of some icons are needed, e.g. in most cases a table should be placed before a chair. This is achieved through rules describing after which other icons a specific icon should be placed. The rest of the placing order is randomly for achieving maximal variability of the result.
Another characteristic of an icon is its size. This size is expressed in regard to half of the average door width.

5.5.4 Finding the optimal icon spot

In a first step, for each icon to be placed a value for each pixel in the corresponding room is calculated. This value is the dot product of the distance vector of the pixel - containing all information about walls, openings and other icons - and the weighting vector of the icon that has to be placed. In a second step, the optimal icon spot is determined using brute force search. Therefore, an area value, being the sum of all pixel values of the respective icon area, is calculated for every possible icon spot in the room. For achieving a higher variability of the results, a small random value is added to each area for randomly picking one of the spots having the maximal value. This is repeated for the icon being rotated by 90 degrees.
So far, the algorithm contains rules for ten different icons.

6 Results

To sum everything up, here we show some qualitative results, of our full pipeline shown in Figure 10. The Figure 21 showcase how an input image is the processed.

1. Read an input image

2. Apply the room types segmentation to get a mask with room types for each pixel

3. Apply the room proposals segmentation to get a mask with room interiors for each pixel

4. Apply the doors room openings segmentation to get a mask with openings pixels

5. Use the output in step (2) and step (3) to create a refined room proposals and types (more information about the refinement steps in section 5.3)

6. Use the output in step (4) and step (5) to place the icons (more information about icon placing rules in 5.5)

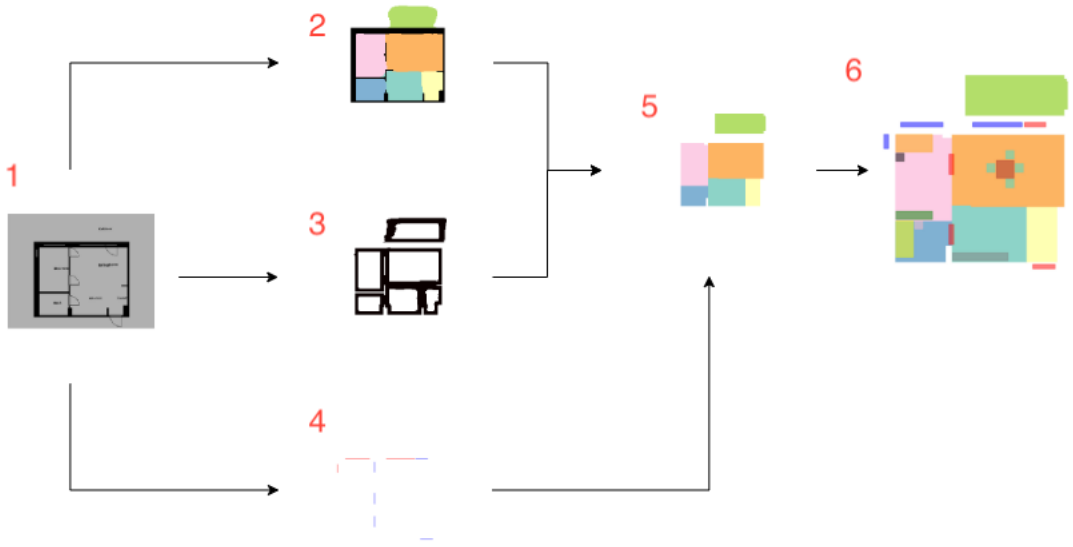


Figure 21: Pipeline Images, steps numbers explained in 6

7 Conclusion

In conclusion, all the initial goals of the project are achieved. We generated an unique dataset, created a pipeline consisting of room detection, room type classification, openings classification and placing icons algorithm
The generated dataset simulates real world data and contains 1881 images with image semantic segmentation annotations, segmentation masks for room types and for rooms openings. The algorithm for fusing segmentation masks and room proposals is totally new.
The networks for extraction and classification of the rooms show really good visual results. However, there are some limitations which can be improved in the future. First, the segmentation network can't tolerate text which is rotated by 90 degrees. Second, room proposals heavily depend on good segmentation and room proposal algorithm refinement can't perform well when the number of rooms is too large. Icon placement also faces problems when dealing with complex shapes of the rooms.
All those limitations may be overcomed by conducting more experiments with data augmentation for the networks and more rules and experiments with icon placing algorithm.

References

[1] S. Dodge, J. Xu, and B. Stenger. Parsing floor plan images. In *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, pages 358–361, May 2017.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[3] Ahti Kalervo, Juha Ylioinas, Markus Häikiö, Antti Karhu, and Juho Kannala. https://github.com/cubicasa/cubicasa5k.

[4] Ahti Kalervo, Juha Ylioinas, Markus Häikiö, Antti Karhu, and Juho Kannala. Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis, 2019.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[7] C. Liu, J. Wu, P. Kohli, and Y. Furukawa. https://github.com/art-programmer/floorplantransformation.

[8] C. Liu, J. Wu, P. Kohli, and Y. Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2214–2222, Oct 2017.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[11] Yi-Chao Wu, Fei Yin, Xu-Yao Zhang, Li Liu, and Cheng-Lin Liu. Scan: Sliding convolutional attention network for scene text recognition, 2018.