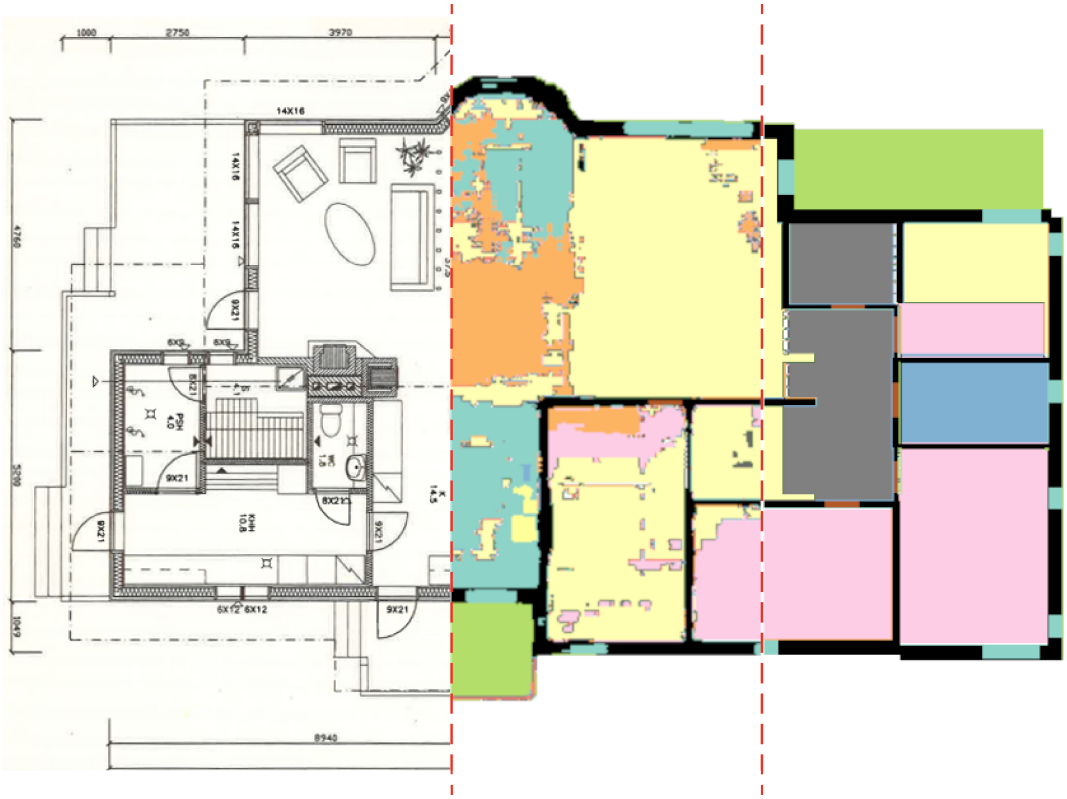
CHALMERS
UNIVERSITY OF TECHNOLOGY



Vectorization of architectural floor plans

PixMax – a semi-supervised approach to domain adaptation through pseudolabelling

Master's thesis in Complex Adaptive Systems

Alexander Radne
Erik Forsberg

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Master's thesis 2021

Vectorizaton of architectural floor plans

PixMax – a semi-supervised approach to domain adaptation through pseudolabelling

Alexander Radne
Erik Forsberg

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Computer Vision
Chalmers University of Technology
Gothenburg, Sweden 2021

Vectorization of architectural floor plans
PixMax – a semi-supervised approach to domain adaptation through pseudolabelling

Alexander Radne
Erik Forsberg

Supervisor and examiner: Fredrik Kahl, Department of Electrical Engineering

Master's Thesis 2021:NN
Department of Electrical Engineering
Division of Computer Vision
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of different stages of vectorization of a floor plan. Raster image from a scanned or rasterized architectural drawing (left), neural network's pixel-wise class segmentation map (middle) and polygonized vector graphics image (right).

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Vectorization of architectural floor plans
PixMax - a semi-supervised approach to domain adaptation through pseudolabelling
Alexander Radne, Erik Forsberg
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Machine Learning and Computer Vision techniques are rapidly improving computers' abilities of image comprehension. In recent years, these techniques have been applied to information parsing on floor plan bitmap images, thus addressing the problem of converting rasterized images to vector graphics. Current state of the art models have shown great results in predicting walls as well as room types and architectural drawing icons. However, these models require a large amount of annotated data, and since the cost of labelling can be quite high, the current available datasets are limited in terms of diversity of styles and regional-specific features. Therefore, there is an opportunity for algorithms that exploit unlabelled data to further improve these models. Semi-supervised learning is a set of algorithms commonly used to achieve this.

We propose and analyse three approaches utilising semi-supervised learning through self-training by letting a model trained on labelled data make predictions on unlabelled data. We then use a collection of the best of these predictions as a basis for creating pseudolabels for further training. In the first approach, we use a probability measure on the model output as a proxy for high quality predictions. Our second approach is to use a post-processing algorithm as a quality enhancement of the predictions on all unannotated images. Finally we propose and evaluate our proposed prediction quality measurement, PixMax. This method aims to give a proxy for how confident the network is on its predictions by measuring inter-consistency between several non-destructive augmentations of any input image. The created pseudolabels are then compared to evaluate whether the network is confident enough or not for the pseudolabels to be included in the continued training.

With PixMax we obtain results comparable with — and for recall better than — the fully supervised state-of-the-art model that we benchmark against. Our evaluations are carried out both on the labelled and unlabelled dataset used to train the models. As expected, the relative performance boost is most prominent on the unlabelled dataset where we reach a 69 % average recall. We show that the PixMax approach can be used for adapting a trained model to a new domain.

Keywords: semantic segmentation, object detection, semi-supervised learning, floor plan images, domain adaptation, self-training.

Acknowledgements

First we would like to thank our supervisor Fredrik Kahl for his support and guidance during the course of this project. He helped us to both on an academic and administrative level to find and access the right resources to develop the project in the desired way. Lars Hammarstrand helped us to get admitted to a compute project which allowed us to access GPU-resources. We would like to thank him as well as the team at C3SE for helping us with this. Also Anders Karlström was of great assistance to the project by taking of his time to read and sign the application for access to one of the datasets we used.

During this time of social distancing and isolation, taking time for some coffee and small talk is more important than ever. We would therefore like to send a special thanks to Erica Samuelsson and Sara Eidenvall for sharing the morning coffee break with us every day and for all the interesting discussions that this led to. We would also like to in particular thank Adnan Fazlinovic, Joel Ekelöf, Sofia Malmsten among many others who have been supportive during this process.

Finally we would like to thank our families and friends for all the support and patience shown during this time.

Alexander Radne
Erik Forsberg
Gothenburg, January 2021

"You have broken new ground for the Architecture and Engineering programme"
— Karl-Gunnar Olsson, former head of programme

"Really nice stuff!"
— Markus Häikiö, CTO, CubiCasa

"Sometimes you don’t see the full picture for all the pixels."
— Common saying

Contents

List of Figures

List of Tables

1
Introduction

In this chapter we will give a brief background and history to the topics that are being investigated in this thesis. We will start by describing the the possible benefits there could be in the industry in using the techniques that we propose and we will then continue to give a brief outline of our proposed method. In the last section of the chapter we present a chronology of work that has been done on related topics though the last decade and the papers that this project is based on.

1.1 Background

The building industry has undergone a lot of large technological changes during the last few decades. One of the most prominent examples of this is the way digital tools is now used to aid the complex coordination process of big projects that span over multiple disciplines and long time-scales.
Even though the industry is susceptible to digital development and progression in general, the evolution is slow due to the long project time scales and hence slow information reversal. The digitalisation has up until a few years ago mostly been focusing on how to streamline the design process. This has been done for instance by simplifying communication between disciplines with automated clash-checks and by moving from 2D-drawings to 3D-modelling software.
Today effectively all new production design and planning are heavily aided by digital tools such as CAD (Computer Aided Design) and BIM (Building Information Modelling) software. These software give a much improved way of coordinate the workflow between different disciplines and they make it easier and faster to change parts of the design in different stages of the design process compared to the traditional way of designing buildings with pen and paper. Moreover these modern software are in general both vector-based and support object oriented modelling in some form. This gives the user the ability to combine drawing and other kinds of information in an efficient way.
Vector-based drawings have the advantage of being easy to both modify and annotate compared to raster-based drawing formats. In combination with the metadata attachment capability of these drawings they are in many ways far superior compared to traditional raster-based images not only in the planning of a project, but throughout its whole life cycle when it comes to versatility and maintenance.
The problem that we want to address is that many of the drawings that are used to convey information to clients and customers are often stripped of this information when they get converted to a raster image format for distribution outside the soft-

ware where they were originally made. There are also a big fraction of drawings that were made before the adoption of these software and hence only ever have existed in raster-based image format or as a physical printed drawing. By converting these drawings (back) into vector format, many new possibilities for how they can be used will emerge.
Floor plans is the type of architectural drawing that most often is used to carry information about a building or an apartment to the general public and it is also one of the most common type of drawing to encounter in all sorts of projects. To have these drawings in an annotated, vector-based format would open doors for e.g. property owners, real estate agents and property management firms that wants to convey information about floor plan layout in a more intuitive way. This could be done by e.g. create a 3D representation of the property on their website. This is a much easier task to do with a vectorized floor plan as a basis rather than on a raster image due to the geometrical information being represented explicitly in a vector based image as opposed to a raster image. Another potential application is to extract information from old drawings to then be included in a reference database for architects, planners and engineers etc. to make informed decisions.

1.2 Proposal

Our proposal is to improve on current automated pipelines that exist for converting raster based floor plan images into a vectorized (mathematically represented) format with the use of machine learning.
To do this, we want to create a model that is able to able to distinguish between several of the most common ways that floor plans are represented by being able to detect a finite set of features such as walls, room spaces, doors and windows. After identification of these features the model should be able to create an accurate vector representation of the floor plan with geometries in the form of drawing "symbols" and metadata attached to these object symbols.
The output from the model should be a in a format that is easily read and convert to the most common and widely used CAD-software file formats. Once the result is converted to one vector-based format, it is quite easy to convert it to other formats since most modern CAD-software have methods for converting files from other common formats built in.
Recent work in the area by Kalervo et. al. and Liu et al. [1], [2] has been shown to give good and reliable results using Artificial Neural Networks (ANN). Despite reaching impressive results, the lack of large, annotated data sets are in these works pointed out as one of the greatest challenges in creating a model with even better generalisation capabilities. This project aims to work towards a solution to this problem by introducing a framework for using unannotated floor plan images to let the model learn to work on images with novel drawing styles.
This idea can also by extension be seen as a step in the direction of being able to create larger, annotated, custom datasets of floor plans that can be used for data analysis or to train ever more intricate models. In other words, a good and reliable model for parsing floor plan images might be used for annotating large datasets to be used for other applications.

1.3 Method outline

To address the lack of large quantities of annotated data of high quality, our proposed method is based on Kalervo et. al. [1] and Liu et. al. [2], with the difference that we instead use a semi-supervised approach allowing us to use a large data set of unannotated floor plan drawings to train the model further.
Our approach consists of letting the network model perform predictions on unnanotated data. These predictions will after carefully chosen refinements be used as pseudolabels for the model to be further trained on. In order to increase the performance of the network we test several techniques for improving the quality of the pseudolabel based on the network output.
In the first pseudolabelling method, we evaluate the potential correlation between the "confidence" (network output after softmax layer) and the average pixel-accuracy. Second, we measure the potential increase of quality after using a post-processing algorithm used in [1]. Finally, we develop a prediction quality measurement — PixMax — based on a batch of non-destructively augmentations to the same image. This measurement is used to select only pseudolabels of high quality for further training.

1.4 Scope and limitations

We limit the scope of the project to only architectural floor plans. In most projects, a variety of different floor plan drawings are used to convey different types of information to the construction workers. These plans can include installations and fixtures, electrical wiring and plumbing the materials being used and different phases of the building process. We have chosen to only look at architectural drawings since it is the type of drawing that is primarily used after the building is finished to display information about its architectural qualities.
To also limit the scope of the project we will only use single-level floor plans for our model. Following from the approach of [1], we limit the scope to use a set of 12 room classes, 11 icon types and 21 types of interest points as given in Table 3.1.

44 Output maps				
21 Interest points (Heatmaps)			23 Segmentation Maps	
Wall corners	Opening endpoints	Icon corners	Room class	Icon class
13	4	4	12	11

Table 1.1: The layers of information that our model extracts from a floor plan image.

Two datasets will be used in this project; one annotated dataset that is used to train the initial model and one unannotated datasets that will be used to for creating pseudolabels and evaluate the models performance after the extended training.

1.5 Research questions

- With what accuracy are we able to recover information from a raster image of a floor plan to a vectorized representation using our proposed method PixMax?
- How does our algorithm compare to state of the art results in the field?
- Could a semi-supervised algorithm be used to improve the results based on only a fraction of annotated data in the dataset?

1.6 Related work

The following is a brief summary of the field and the two papers that have inspired this thesis the most.

1.6.1 Vectorization of architectural floor plans

The problem of converting floor plan raster-images to a vector-based format has been explored extensively over the last few years [1]–[5]. The techniques used for the task has shifted from conventional algorithms such as patch-based segmentation [4], to the use of neural networks as bigger data sets have been released and the cost of computation has become cheaper, making the data-hungry networks a viable option [6]. Due to the intricacy of the task and the variety in the data, neural networks has been shown to yield good results compared to other algorithms. This can be accredited to their ability to find complex correlations in the data and to generalise[1], [2].
A commonly used method lately for image parsing is semantic segmentation where an image is split up into a set of pixels per class segmentation maps, each corresponding to a one of N predefined classes in the specific dataset [7]. Several datasets have been released and the and the results have been steadily improving as better techniques have evolved [8]. Although great results have been shown on a variety of different datasets, the main focus of the research has been on natural image segmentation since this has been an important problem to solve for several major industries such as the automobile industry where the goal of building self driving cars is a strong driving force.
Although not the main focus of the research in machine learning, a significant amount of work has been done on semantic segmentation for human-created images such as drawings. The idea of using segmentation maps as a way to automatically vectorize architectural floor plan drawing in some cases predates the use of neural network based methods. Heras et al. uses a statistical, grid-based method to segment walls, windows and doors from floor plans [9]. To train their model they used the very popular CVC-FP dataset [10]. This was at the time one of the biggest and most popular, publicly available floor plan datasets with annotated images. After this work it seems segmentation methods making use of neural networks has been increasingly popular. Dodge et al.[3] proposed in 2017 a method where Optical Character Recognition (OCR) from the Google vision API was combined with a fully convolutional network, the Faster R-CNN framework, to obtain a model that can both segment walls and also interpret semantic information in the input such

as measurements and room types written in the drawing. In their work they also introduced a new public floor plan dataset known as the R-FP dataset containing 500 high-resolution real-estate floor plan images.
In 2018 Yang et al. managed to segment walls and doors simultaneously [5] using U-Net+DCL - an alteration to the U-Net where the deconvolutional layers was replaced with a simplified version of pixel deconvolution layers. They managed to achieve an validation pixel accuracy of 97.5 respectively 99.5% for walls and doors, establishing that impressive results can be reached using convolutional neural networks on real-estate floor plans.

1.6.2 Raster-to-Vector & CubiCasa5k

The two works that this project is most heavily inspired by are Kalervo. et al. and Liu. et al. from [1], [2]. Liu. et al. proposed in their 2017 paper a learning-based method with multiple objectives. By transforming a rastor-image with the model into both heatmaps of low-level geometric and semantic information (a set of corner and end points) and also a semantic segmentation map of different room and icon types, they managed to extract multiple layers of information from a single image. In contrast to Dodge et al. [3] where the model architectures consisted of multiple network that learned separate tasks, instead of using a single model for all learning goals. This was done by implementing single fully convolutional network (FCN) with a multi-objective loss for the different output maps, and then combining the individual losses with a weighting for the prediction categories to a combined total loss that is used to back-propagate the network.
The advantage of this approach is that the geometric feature maps (corners, wall end points etc.) can be used in an intelligent post-processing schedule that aims to refine the rather coarse network output in terms of the room and wall segmentation maps. An conceptual illustration of this can be seen in Figure 1.1. For example, by knowing with high precision the four corner points of a rectangular room, the segmentation map of the pixels within that room can in some cases be improved by knowing that all pixels within that room most likely are of the same class. This fact have been researched and evaluated in chapter 3.2.2 to see if this can be used to further improve our model.
Their algorithm ultimately yielded around 90% precision recall for wall junctions, walls, drawing icons and rooms on the LIFULL HOME'S dataset [11], significantly outperforming most other methods trying to extract the same amount of information from a single drawing.
In 2019, Kalervo. et al. [1] continued on this work by using the same baseline network architecture, ResNet-152 [12] pretrained with ImageNet [13], but they extended the dataset used substantially by making use of their (CubiCasa) manual floor plan annotation pipeline to collect 5 000 high-quality, human-annotated data points from a set of 15 000. They also more than doubled the number of target room- and icon classes to get more reliable and exact predictions.
Making use of this data, they managed to outperform [2] in both recall and accuracy for all classes but one, all while just making use of one single model for predicting all the different feature maps with varying learning criteria.

This project intends to continuing to build on the framework that was created respectively refined by these two works.
e

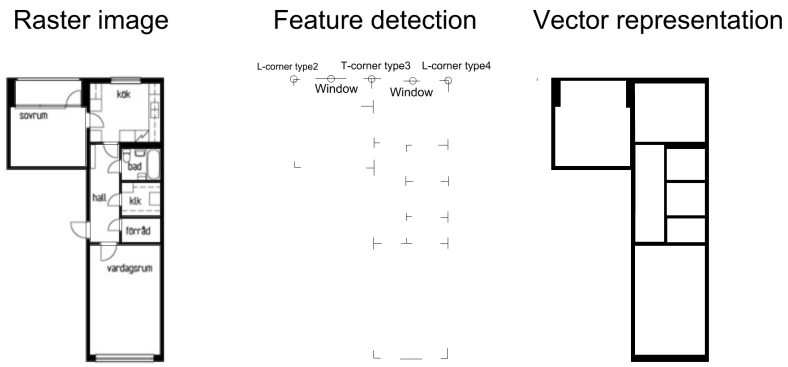


Figure 1.1: A concept representation of the method first introduced by Kalervo. et al [1] where a specific set of interest points are detected to aid the vectorization algorithm that is separate from the main network model.

2
Theory

This section will give a brief background of some to the techniques and key concepts used in machine learning in general and used in this project in particular. We will cover all necessary theory that is relevant to the project. It will however be assumed that the reader has a solid understanding of the field a priori. We will go into themes related to semi-supervised learning and computer vision through convolutional neural networks more thoroughly since these are the primary concepts for this work and it is vital to understand the techniques used properly. We will also touch upon concepts such as the residual neural networks and multi objective loss functions.

2.1 Deep learning

Deep learning is a subgenre of machine learning that deals with algorithms based on Artificial Neural Networks (ANN) and representation learning. Deep learning is substantiated by the notion that there exists some non-linear, often complex function G that can map points x from a high-dimensional input domain $\mathcal{H}$ to target points y in a defined target domain $\mathcal{I}$:

$$G : \mathcal{H} \rightarrow \mathcal{I}. \tag{2.1}$$

The assumption we usually make is that for certain kinds of problems we only need to know a small fraction of all point-mappings from input to target space to be able to predict a much bigger fraction with a good precision.
Just based on our trivial assumption we cannot induce any boundary on the complexity of G. To make this model framework useful we need to be able to approximate it with a reasonably good parameterised approximation G_θ [1]. So we want G_θ to mimic the mapping of G for x in the domain we are concerned with. For a parametrization θ of dimension m, the objective can be stated as follows:

$$\text{find } \theta \in \mathbb{R}^m \text{ s.t. } \forall x \in \mathcal{H} : \; G_\theta(x) \approx G(x) \tag{2.2}$$

Since we in general do not know what G does for the majority of all $x \in \mathcal{H}$, we cannot explicitly use equation 2.2 to find θ.
But if we have a random sample of known mappings of size B, we can view $\{\mathbf{x}, \mathbf{y}\} = \{\{x_1, \ldots, x_B\}, \{y_1, \ldots, y_B\}\}$ as a random variable and use this approximate the real probability distribution of x, $P(x)$, by an empirical approximation $P_{emp}(x)$. We can

[1] θ typically represents the weights and biases in an artificial neural network.

now try to find the set of parameters $\hat{\theta}$ that when applied to a fixed G minimises the expected loss over this simplified probability distribution $P_{emp}(x)$:

$$G_\theta := G_{\hat{\theta}} \text{ with } \hat{\theta} := \underset{\theta}{\operatorname{argmin}} \int \mathcal{L}(G_\theta(\mathbf{x}), y) dP_{emp}(\mathbf{x}). \tag{2.3}$$

Here $\mathcal{L}$ is a loss function that measures the difference between the prediction and the target. The exact composition of this function will be further discussed in Section 2.4.1. For practical purposes we would also like to require θ to not be too big since we want to be able to conduct calculations with it in reasonable timescales. Generally speaking, it is clear that it is not always the case that such a function exists, but it has been shown empirically that for certain kinds of problems it seems to often be the case. Fortunately these problems often coincides with the problems that have applications in many areas and that is the reason why deep learning and in particular deep artificial neural networks have become so popular in the recent years. They give us a framework for finding well-behaved parametrizations of seemingly arbitrary mappings.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is a type of artificial neural network that is commonly used in computer vision applications. Their high performance in image analysis partly comes from their shared-weights architecture and translation invariant characteristics.

A convolutional network is defined as any artificial neural network that uses one or more convolutional layers in its architecture. A convolution can be understood as a filter that is being slid over portions of the previous layer to calculate the next. This gives the model a way of perceiving neighbourhoods in the input vector and it is therefore useful when there are thought to be large structures in the data that is linked to the closeness of its building blocks. One of the most classical examples of this is shapes and objects in an image. The filter is called a *kernel* and it can be distributed in one or more dimensions. For image analysis of colour images, 3-dimensional kernels are most commonly used since this corresponds to the two spacial dimensions of the image plus and the "channel dimension" where the red, green and blue values are separately.

A kernel is a small matrix of weights. The placement of the individual weights in the kernel can be arranged such that it is tuned to detect a certain kind of low-level feature such as lines, edges or dots in the data tensor. This is done by element-wise multiplication between values in a region of the preceding layer and the kernel as can be seen in Figure 2.1. By combining multiple kernels with different feature detecting abilities into a kernel tensor the model can assimilate the distribution of such features in the data and can hence be analysed with a maintained geometrical interpretation as opposed to a fully connected layers where all geometrical integrity is lost.

Fully convolutional networks are networks that doesn't contain any fully connected layers but completely relies on convolutions throughout the propagation. Since a fully connected layer is equivalent to using a kernel of size 1×1, or 1 pixel in the image processing setting, the geometric interpretation of a FCN with larger kernels

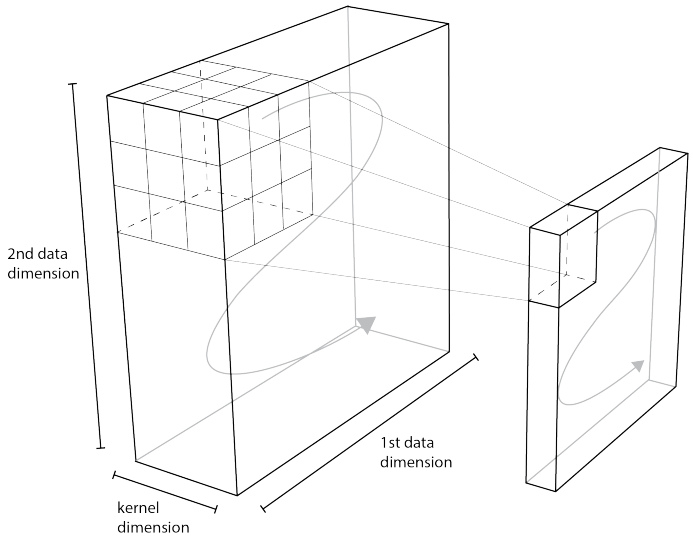


Figure 2.1: The concept of a convolutional layer. In this particular example we have data in 2 dimensions and a third kernel dimension. The items in the data tensor gets element-wise multiplied with a kernel tensor and summed to form the consequent layer in the network.

is that it exclusively considers regions in it's propagation and never values of unique neurons.

2.1.2 Optimisation and vanishing gradients

In Section 2.2 we described that we want to improve our candidate G_θ over time with at backpropagation algorithm B that is dependent on the current parameter state and the chosen loss function. There are however many ways we can define B to do this. The most straightforward approach would be to look for the direction in which to tweak the the parameters of G_θ to get the biggest local decrease in the loss function. We can then take a step of size η, known as the learning rate, in that direction. This is what is called i.e. the gradient descent method and one update step can be written as

$$\theta^{t+1} \leftarrow \theta^t - \eta \nabla_\theta \mathcal{L}(\theta^t | \mathbf{x}, \mathbf{y}) \tag{2.4}$$

for some loss function $\mathcal{L}$ our input-target data set $(\mathbf{x}, \mathbf{y})$ of size N. However, there are a few problems with this method. Since the gradient is calculated for every single data point for each step, it can be very slow if N is large, especially if θ is large as well.
Another unwanted feature of this method is that it is *greedy* in the sense that it will always choose the direction that is locally thought to be the most efficient step at any time. The problem with this is that the algorithm can get stuck in a local optimum without having any chance of getting out of it to find the global optimum. A popular way to solve these problems is to use a stochastic gradient descent method, first described in a paper by Robbins et. al. [14]. In our setting, one step can be described as

$$\theta^{t+1} \leftarrow \theta^t - \eta \nabla_\theta \frac{1}{K} \sum_i^K \mathcal{L}(\theta^t | x_i, y_i) \tag{2.5}$$

where $\{x_1, \ldots, x_k\} \subset_R \mathbf{x}$ is a random subset of $\mathbf{x}$ of size k with corresponding targets $\{y_1, \ldots, y_k\} \subset_R \mathbf{y}$. In the original paper k was set to 1, but in the general case the batch size can be set to any number $1 \leq k < N$ to reduce the probability of an unrepresentative samples while maintaining a big overhead compared to deterministic gradient descent. Since $\{x_1, \ldots x_k\}$ is a random variable, it introduces the possibility to occasionally move in locally non-optimal directions that can be globally beneficial which makes it less prone to getting stuck in local optima.
Both these presented algorithms have a fixed learning rate η that does not change thought the training. There has been many approaches to making the optimisation more effective with dynamic learning rates. Some of these methods include AdaGrad which works well with sparse gradients [15], and MSRProp with good performance in on-line non-stationary[16].
In 2015, Diederik Kingma and Jimmy Ba proposed an algorithm they called ADAM, short for adaptive moment estimation, that combines the benefits of the AdaGrad and RMSProp[17]. It does this by introducing a momentum in the training schedule by using a decaying moving average and squared average of the gradient. One step in the ADAM algorithm can be described as executing the following steps:
We first calculate the moving decaying averages of previous gradient m and squared gradient v respectively

$$\begin{aligned} m &\leftarrow \beta_1 m + (1 - \beta_1) \nabla_\theta \mathcal{L}(G_\theta | \mathbf{x}, \mathbf{y}) \\ v &\leftarrow \beta_2 v + (1 - \beta_2) \nabla_\theta \mathcal{L}(G_\theta | \mathbf{x}, \mathbf{y})^2 \end{aligned} \tag{2.6}$$

These become estimates of the 1^{st} and 2^{nd} moment of the gradient of the objective function $\mathcal{L}$. Since these are initialised to 0, they are negatively biased. To counteract this we calculate unbiased versions of these variables the following way

$$\begin{aligned} t &\leftarrow t + 1, \\ \hat{m} &= \frac{m}{1 - \beta_1^t}, \\ \hat{v} &= \frac{v}{1 - \beta_2^t}. \end{aligned} \tag{2.7}$$

We also update our time parameter since we have the time-dependent terms β_1^t and β_2^t that we want to be progressively smaller as the effect of the initialisation wears of. Finally we update our parameters of the network using $\hat{m}$ and $\hat{v}$ using the following equation

$$\theta^{t+1} \leftarrow \theta^t - \eta \frac{1}{\sqrt{\hat{v}} + \epsilon} \hat{m} \tag{2.8}$$

In ADAM we introduce three new hyperparameters β_1, β_2 and ϵ. The β-terms corresponds to the relative exponential decay rate of the 1^{st} and 2^{nd} moment of the gradient respectively and ϵ is just a small number that prevent us from a getting a zero-term in the the denominator of equation 2.8.

It was for some time hypothesised that it was possible to create more powerful convolutional networks just by stacking more layers because of the recent breakthroughs in image classification [18] and object detection [19]. However, it was also recognised that deeper neural networks are often more difficult to train and it was in 2016 shown that if you just keep adding more layer to a network, it eventually get worse, not better[12]. This is in part because adding more parameters will make a network more prone to overfitting if the dataset is small [19], but another big contributor to the problem is what has been called the *The fundamental deep learning problem*.
The vanishing gradient problem was first formally identified by S. Hochreiter in 1991. Ten years later an additional paper in English by Hochreiter et. al. was published that elaborates further on topic with more extensive surveys and [20], [21].
The vanishing gradient problem inherits from the way deep neural networks are traditionally trained. Through backpropagation, the weights of each layer is updated based on the gradient of the previous layers activation function [22]. The core of the problem comes form that the activation function is chosen to squeeze any input to a much narrower range, e.g. (0,1) for the commonly used sigmoid function. As the derivatives progresses thorough the network, the it becomes a chain of derivatives that each depend on the previous. For the weights of the first hidden layer, the update formula becomes

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial V_n} \frac{\partial V_n}{\partial V_{n-1}} \cdots \frac{\partial V_1}{\partial W_1}. \tag{2.9}$$

Where V_k and W_k are the outputs and weights of the k^{th} layer. Now, since each layer uses an activation function we are going to get the derivative of the activation function as the outer derivative for each layer. In the k^{th} layer we get

$$\frac{V_k}{\partial V_{k-1}} = \frac{\partial \phi(z_k)}{\partial z_k} W_k \tag{2.10}$$

For some activation function ϕ where $z_k = V_{k-1} \times W_k$ [23]. If we choose $\phi(\mathbf{x}) = Sigmoid(\mathbf{x})$, we have that the therms containing $\partial\phi$ always have an amplitude in the interval $(0, 1/4]$. The standard approach to weight initialisation in a typical neural network is $W \sim \mathcal{N}(0, 1)$. Hence, the weights in a neural network will also usually be between -1 and 1. As we multiply more and more of these terms together it is easy to see that the gradients quickly grow small and hence they are barley affected by the backpropagation. This also explains why the problem is especially prominent in networks with many hidden layers.
On the other hand, if an activation function with a large derivative in the relevant interval, these can also accumulate and instead cause exploding gradient problems. Exploding gradients results in exponentially large updates to the network weights which is likely to cause a very unstable network.

2.1.3 Residual networks

There are a few ways to deal with the problem of vanishing gradients. The trivial solution is to just make the networks shallower. However, this solution has some drawbacks since it has been shown that the depth of the network is often of great

importance to its performance as stated earlier [18], [19], [24]. Moreover, the activation at different depths of a deep network has been shown to sometimes have useful interpretation by encoding a hierarchy of different features sizes.
The early layers can be thought of as representing low level features such as lines and dots while layers closer to the output are capable of capturing high-lever features such as shapes or objects[25].
A better solution to the problem was proposed by He et. al. with the introduction of ResNet in their 2016 paper [12]. What they suggested was to add *skip connections* to the network avoid the problem. A skip connection is a connection that jumps over a certain number of layers, a so called ResBlock, and then connect back to the network.
The argument for using deeper networks is that since a deep network G_n define a more powerful function class than the shallow counterpart G_k, it should in some sense have a better potential to mimic the true mapping G that we want to find. However this might not be the case because it assumes that G_k is nested within G_n s.t. G_n can do everything that G_k can do an more [26]. This concept is illustrated in Figure 2.2.

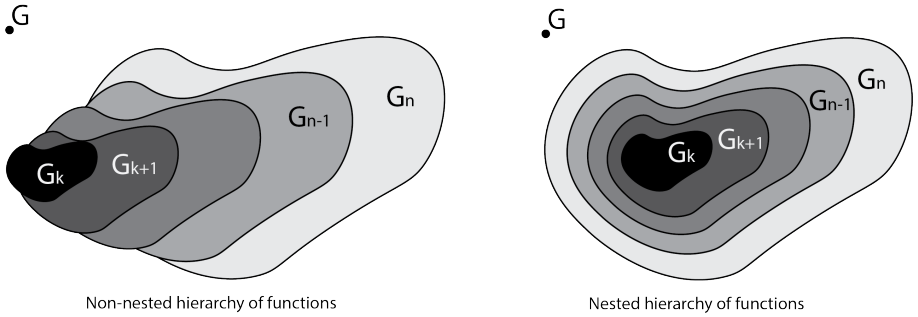


Figure 2.2: For nested function classes, using a bigger function class means that we can get closer to the true function G, but this is not necessarily the case for non-nested function classes.

The reasoning behind the proposal by He. et. al. is that in theory the function class of deeper model should completely inclose that of a shallower, since it can just mimic the shallower model by using identity mappings, but that in practice this is not always the case because the identity function is not a trivial function to learn. Hence the deeper model can struggle to make as good predictions as the shallow one just because it needs a lot of training data juts to lean what layers that should have an identity mapping.
By realising this it was deducted that we can help the model to by explicitly reformulate the layers as residual functions with reference to the layer inputs. This is done by the observation that we can split any function $f(x)$ into a sum of the identity function $I(x) := x$ and a residual function $r(x) := f(x) - x$. We can then propagate the residual function through a number of layers in the network and then add the identity back to it, as can be seen in Figure 2.3. This makes it easy for the model to "skip" a layer by just pushing all the weights to zero, which has empirically been shown to be much easier than to find the identity mapping. We are basically giving the network a shortcut that makes it possible to combine the power of a

deeper network with the agility of a shallower network.

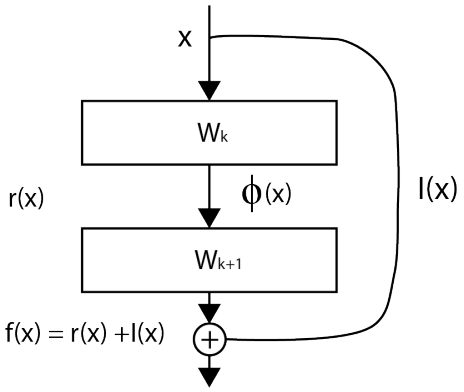


Figure 2.3: The structure of the ResBlock. The function f is split into a residual and an identity function. Only the residual function is propagated through the network to later be added back together with the identity function.

2.2 Supervised learning

The framework that deep learning presents to us for finding a suitable candidate for G_θ is the training of an artificial neural network. The principle is to initiate a model with the architecture of a layered network with many free parameters that can be tuned to make it imitate G. This is usually done through a process known as forward- and back propagation where a set of data points with known mappings, $\{\mathbf{x}, \mathbf{y}\} = \{\{x_1, y_1\}, \ldots, \{x_B, y_B\}\}^T \in \mathcal{H} \times \mathcal{I}$ s.t $\forall i \leq B : x_i \overset{G}{\mapsto} y_i$, are presented to the model and its parameters are updated to reduce the prediction error in each time step:

$$\theta^{t+1} \leftarrow \theta^t + B(\theta^t, \mathcal{L}(\mathbf{x}, \mathbf{y})) \tag{2.11}$$

Here B is a backpropagation algorithm that updates the the model in a way that is likely to reduce its prediction error, e.g. by using stochastic gradient descent. As earlier established, it is required to have some kind of metric for how good the models current prediction is. This distance function, represented by $\mathcal{L}$ in equation 2.11 is in deep learning known as the energy function or the loss function.

2.2.1 Bias-variance tradeoff

One of the biggest dilemmas in supervised learning is what is known as the Bias-variance tradeoff problem. The issue comes from the fact that we only use a small subset of all possible examples to fit a model that we want to generalise well for all data in the distribution. [27]. This leads to an inevitable tradeoff between two different sources of errors:

- **The model bias**
  Measures the average difference between the model prediction and the target. A model with high bias cares little about the training data it is presented with

and tries to oversimplify the problem. Therefore models with high bias are often described being *underfitted*.

- **The model variance**
  Measures how much the model predictions move around its mean on average. A model with high variance pays a lot of attention to the specific training data it is presented with but does not generalise well outside this specific sample. A model with high bias is often described as *overfitted* to the training data.

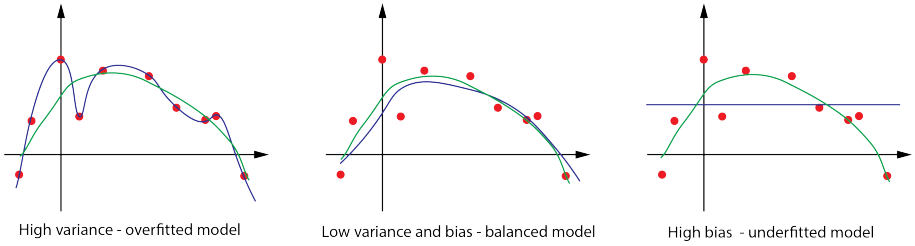


Figure 2.4: The concepts of underfitting and overfitting a model to the data. The model in the middle has a good balance between capturing the main features of the data but is at the same time stable to noise and therefore better approximates the true function (green).

The tradeoff is not only a conceptual construct to easier define model behaviour; it can be shown that the expected test loss of any model can be described in terms of its variance and bias errors in the following way [28]:

$$\mathrm{E}_{x\in\mathcal{H}}\left[(G_\theta(x)-y)^2\right] = \mathrm{Bias}_{x\in\mathcal{H}}[G_\theta(x)]^2 + \mathrm{Var}_{x\in\mathcal{H}}[G_\theta(x)] + \epsilon^2 \tag{2.12}$$

Where $\mathrm{E}_{x\in\mathcal{H}}\left[(G_\theta(x)-y)^2\right]$ is the expected test Mean Squared Error (MSE). This refers to the value we would approach if we estimated G_θ based on a large number of training sets from the distribution $\mathcal{H}$ and averaged the squared distances from the model predictions to the target of iid samples x, also in $\mathcal{H}$. Since the variance term is always non-negative and the bias term is squared, it is easy to see that $\mathrm{E}_{x\in\mathcal{H}}\left[(G_\theta(x)-y)^2\right] \geq \epsilon^2$, where ϵ is the irreducible error in the data. This is the so-called unexplained variance, also called the noise. Equation 2.12 also implies that it is impossible to escape this trade-off. A model with zero variance will inevitably have unbounded bias and vice versa [29].
When trying to find the parameters of a model that minimises some objective function we only use a small subset of the possible samples that could be in the distribution. If the model is trained until convergence we are therefore at big risk of lowering the bias term too much at expense of the variance. This effect is especially prominent for highly non-linear models with a large number of parameters such as ANNs [30]. To find a reasonable balance between the two sources of errors, we use a separate partition of the dataset — the test set — independent of the training set to determine when the model is starting to become overfitted and to terminate the training at this point.

2.3 Semi-supervised learning

Semi-Supervised Learning (SSL) is a framework for machine learning where we use both labelled and unlabelled data to train a model. The primary assumption in SSL that is used to justify the technique is that for a small amount of labelled data together with a bigger amount of unlabelled data can be used to create a stronger model than the two data sets each on their own. This has also empirically been shown to be the case for many important problems [31]. For instance it has been shown that classification models perform better than the models trained only on labelled data and that joint training — where both labelled and unlabelled data is used simultaneously — is one of the most successful iterative approaches to semi-supervised learning [32].
Thanks to their high performance-to-cost ratio, semi-supervised learning models has risen in popularity over the last years and many frameworks that uses a combination of labelled and unlabelled data has been developed [31]. But how can we know for what problems we can hope for semi-supervised models to work? Or more precisely: If we compare an algorithm that only uses labelled data to one that has access to both labelled and unlabelled data, when is it reasonable to think that the combined model can make a more accurate prediction?
In general one could say that there are gains to make if the knowledge on $P(\hat{\mathbf{x}})$ that one can make through the unlabelled data $\hat{\mathbf{x}}$, is useful in the inference of $P(\mathbf{x}|\hat{\mathbf{x}})$. For this to be the case some assumptions on the correlation between the labelled- and unlabelled data distributions needs to be fulfilled. All semi-supervised learning models make use of at least one of the following statements[33]:

- **The semi-supervised smoothness assumption**
  If two points $x_1 \in \hat{\mathbf{x}}, x_2 \in \mathbf{x}$ in a high-density region are close, then so should their labels y_1, y_2 be. This is to say that the true mapping G is at least as smooth in areas where we have many observations as in regions where we have few or none. This implies that if a path of high density links two points their outputs are likely to be close, but if a low-density region separates them then their outputs can very well be quite different.
- **The cluster assumption**
  The data tends to come in discrete clusters, and data within one cluster is likely to have similar labels. If this is the case then the unlabelled data points might help us to find the cluster boundaries more accurately. In the idealised case we just need one labelled point to tell us the flavour of the cluster and we can then map out its outline by introducing more unlabelled points. Note that this assumption does not say that points from multiple clusters can't have similar labels.
- **The Manifold Assumption**
  The data points $x \in \mathbb{R}^n, \hat{x} \in \mathbb{R}^n$ lies roughly on a manifold $M \in \mathbb{R}^k$ where $k << n$.
  This is useful because of what is known as *the curse of dimensionality*; the fact that the volume grows exponentially with the number of dimensions of our data and thus exponentially more data is required to have the same sample density in a higher-dimensional space. However, if we can find a manifold of

a lower dimension that accurately portrays the structure of the data we can operate in this subspace and partly avoid the problem.

For it to be reasonable to make any of these assumptions, we need to know that the probability distribution of our unlabelled data $P(\hat{\mathbf{x}})$ is the marginal distribution of that of our labelled data $P(\mathbf{x})$. This means that $\hat{\mathbf{x}}$ and $\mathbf{x}$ must come from the same underlying distribution. This is not always possible to guarantee in practice, but even if it does not hold there are things that can be done if $P(\mathbf{x})$ and $P(\hat{\mathbf{x}})$ share some similarities. For instance we can use unsupervised domain adaptation where a model is trained on labelled data from a different distribution than the one it will later be applied on [34].

2.3.1 Pseudolabelling

One of the the most obvious, and therefore also earliest, ways of implementing semi-supervised learning is through so called pseudolabelling or self-training.

One of the most basic implementations of a pseudolabelling framework is a wrapping of the supervised learning algorithm. First we train our model on only labelled data, but for each epoch of the training we label a fraction of the unlabelled data points with the current model state and use these as training examples from that point on. When all unlabelled points have gotten a label the we continue to train the model until convergence to reach our final model state[33]. The high-level idea of the framework was described already in the 60's [35], [36] but has been much refined and packaged since then.

Another way to perform self-training is to first train the model until convergence on the labelled data and then use this model to label all unlabelled examples at once [37]. The training is then continued with a certain fraction pseudolabelled data until convergence is once again reached.

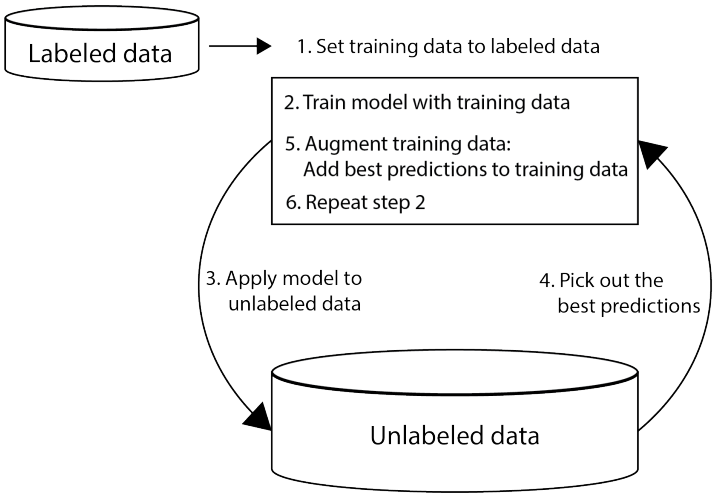


Figure 2.5: The self-training scheme described in [31].

2.4 Loss functions

In deep learning the loss function has two main purposes:

- To give a measure for how well a model is currently preforming.
- To give a prediction for what direction to nudge its parameters in to most likely increase its performance.

Since we want the model to learn something from our labelled data, the loss function is in general a distance function that measures how close the models prediction is to the ground truth e.g. the data label. Depending in the type of information the network is trying to learn, different types of loss functions might be more or less suitable.

2.4.1 Loss functions and probability transformations

For semantic segmentation, a common choice of loss function is the cross entropy loss that gives a measure for the difference between the models probability prediction for each class and the true class, summed over all pixels. Cross entropy loss uses a probabilistic scheme to determine the distance between the true and predicted label for a data point. Since $z := G_\theta(x)$ not necessarily is a per-pixel probability distribution over all classes, we might have to a format the output to something that can be interpreted as such, e.g. by using the Softmax function,

$$\begin{aligned} \text{Softmax}(x_i) &:= \frac{e^{z_i}}{\sum_{j \in \mathcal{C}} e^{z_j}} \\ q(x) &:= \begin{bmatrix} \text{Softmax}(x_1) \\ ... \\ \text{Softmax}(x_C) \end{bmatrix}, \end{aligned} \tag{2.13}$$

where $\mathcal{C} := \{1, \ldots, C\}$ is the sequence between 1 and the number of classes, denoted by C. Using this notion of q, the cross entropy loss can be written as

$$\mathcal{L}_{CE}(x, y) = -\sum_{p \in P} \sum_{i \in \mathcal{C}} y_{i,p} \log q(x)_{i,p} \tag{2.14}$$

where P is the pixels in an image, $y_{.p}$ is the one-hot [2] representation of the true class of pixel p and $q(x)_{i,p}$ is the models probability prediction that $x_p \in \mathcal{C}_i$.

As mentioned, a convenient property of the Softmax transformation is the class probabilities will sum to 1 for each pixel in the image and can hence be interpreted as a probability distribution. The transformation can however suffer from some numerical performance issues and is therefore with advantage combined with a logarithmic transformation, as in the case with cross entropy loss. In cases where no logarithmic transformation is performed, the sigmoid function might be a more suitable choice of transformation. The sigmoid function has a similar inferred interpretation as the Softmax, with the main difference being that it treats the probability of each class as independent from other classes [38],

[2] A C-vector with all zeros but a single 1 in the position of the true class.

$$Sigmoid(x_i) = \frac{1}{1 + e^{-z_i}}. \tag{2.15}$$

Still with $z = G_\theta(x)$ defined as the raw model output.

When we consider the task of object detection, a common loss measure is the intersection over union

$$\mathcal{L}_{IOU} = \frac{|A \cap B|}{|A \cup B|} \tag{2.16}$$

for a given prediction box A and the true bounding box B [39]. This base formula can be extended in many clever ways to account for multiple predictions and classes. However, for it to be stable it requires the objects to be detected to have dimensions big enough that it is a reasonable to look at it as a continuous function. If we instead want to find points of interest in an image, a regression loss function such as mean mean squared error

$$\mathcal{L}_{MSE}(x, y) = \frac{1}{N} \sum_{n=1}^{N} \sum_{i \in \mathcal{C}} \left(z_{i,n} - y_{i,n}\right)^2 \tag{2.17}$$

is usually a better choice. Here $z_{i,n}$ is the location of the models prediction of the n^{th} occurrence of a point of the i^{th} class. A more in-depth description of the concept of different loss functions will follow in the next section.

2.4.2 Multi objective loss and relative loss weighting

In some settings we want a model that can make different kinds of predictions on a single data point. The conventional way to do this is to train multiple separate models to perform parts of the full task, but it can also be done by utilising a so called multi-task model. Baxer et. al. showed in the early 2000s that this approach can increase efficiency and learning accuracy for each task [40]. Simply speaking, the way this is thought to work is that inductive knowledge transfer between complimentary tasks can improve the generalisation capabilities of a model and therefore result in more stable and reliable results.
However, this approach comes with a cost. It requires the models total loss to be treated as a sum of multiple individual losses corresponding to the different learning objectives. This raises the question of how the different terms of the loss functions should be weighted against each other. The performance of each task is in one sense arbitrary since the objectives of the model can have different scales and units, but it is often desirable to have a model that at least prioritise improving on all tasks equally.
Tuning such a hyper parameter manually can be a tedious and time-consuming task and it has to be done over again for each model that we want to train. Kendall et. al. [41] showed that the relative weighting of parameters can be considered an implicit learning goal of the model and can therefore be learned automatically thought the training.

The proposed method is based on looking at the homoscedastic uncertainty for each task of the model. Homoscedastic uncertainty can be defined as the intrinsic uncertainty of the model i.e. the part that is not dependent on how well-trained the model is, but rather the insufficiency of the data that the model has been presented with. For regression, if we assume identical observation noise for each data point x in the we can write

$$\mathbf{y} \sim \mathcal{N}\left(G_\theta(\mathbf{x}), \sigma^2 \mathbb{I}\right), \tag{2.18}$$

where $\mathbf{y}$ is the batch model output, $\mathbb{I}$ the identity matrix and σ the noise scalar of the model. From this we can see that we use the assumption that the model predictions have the same variance and no co-variance. For classification we instead have under the same assumption that

$$\mathbf{y} \sim \text{Softmax}\left(\frac{1}{\sigma^2} G_\theta(\mathbf{x})\right). \tag{2.19}$$

Using this we can calculate the joint probability distribution of multiple outputs by

$$p\left(\mathbf{y}_1, \ldots, \mathbf{y}_n \mid G_\theta(\mathbf{x})\right) = p\left(\mathbf{y}_1 \mid G_\theta(\mathbf{x})\right) \ldots p\left(\mathbf{y}_n \mid G_\theta(\mathbf{x})\right). \tag{2.20}$$

This means that we can then use maximum likelihood inference for the terms on the right hand side. For regression, respectively Softmax, what we finally arrive at is a joint objective function:

$$\min_{\theta, \sigma_1, \sigma_2} \mathcal{L} = \frac{1}{2\sigma_1^2} \min_\theta \mathcal{L}_1 + \frac{1}{\sigma_2^2} \min_\theta \mathcal{L}_2 + \log \sigma_1 + \log \sigma_2. \tag{2.21}$$

The first term of this equation

$$\mathcal{L}_1 = \left\| \mathbf{y}_1 - G_\theta(\mathbf{x}) \right\|^2, \tag{2.22}$$

is for the model regression labels $\mathbf{y_1}$ and the second term

$$\mathcal{L}_2 = -\mathbf{y}_2 \log \text{Softmax}\left(G_\theta(\mathbf{x})\right) \tag{2.23}$$

is for the cross entropy loss of the classification outputs with the and classification labels $\mathbf{y_2}$.

We can now optimise $\mathcal{L}$ w.r.t all the model parameters θ, σ_1 and σ_2. This can be seen as the combined loss function giving the model a way of learning the relative weights of the losses for each output. If the value of e.g. σ_2 is small, it will increase the contribution of $\mathcal{L}_2$, whereas a big value will decrease its contribution. The equation is regulated by the last two terms in the equation that penalises large values for σ. More details and the entire derivation of equation 2.21 can be found in the paper by Kandall et. al [41].

2.5 Consistency regulation

2.5.1 Vicinal Risk Minimisation

The task of labelling big sets sets is very labour expensive, especially when it comes to rich information extraction such as bounding-box annotation for objects or even pixel-wise segmentation. Introducing augmentations to increase both the amount and the diversity of the data has been shown efficient [42], [43] and is today seen as a standard procedure in all of machine learning.
Data augmentation has traditionally been viewed in theoretical statistics as a method of Vicinal Risk Minimisation (VRM) [44]. The reasoning can be understood by first defining a the learning problem as a search of a θ that minimises the expected loss. We can write this as a *risk function*

$$R(G_\theta) = \int \mathcal{L}(G_\theta(\mathbf{x}), \mathbf{y}) dP(\mathbf{x}, \mathbf{y}) \tag{2.24}$$

where our objective is to find $\hat{\theta} := \underset{\theta}{\operatorname{argmin}} R(G_\theta)$ and $P(\mathbf{x}, \mathbf{y})$ is the probability density function over all possible source-target pairs. The problem here is that we cannot know what the true distribution $P(\mathbf{x}, \mathbf{y})$ is since we do not have all that data in our dataset. But given a data set $\{\mathbf{x}, \mathbf{y}\} = \{\{x_1, \ldots, x_B\}, \{y_1, \ldots, y_B\}\}$ we can still estimate an empirical risk function[3] since

$$R_{emp}(G_\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(G_\theta(x_i), y_i) \propto \int \mathcal{L}(G_\theta(x), y) \delta_{x_i}(x) dx \tag{2.25}$$

where the delta function

$$\delta_{x_i} = \begin{cases} 1 & \text{if } x_i \text{ is in } \mathbf{x} \\ 0 & \text{otherwise.} \end{cases} \tag{2.26}$$

The VRM framework is built on the assumption that we can preform some random modifications to the data and still retain the overall structure and its semantic information to a high degree. This means that we can include modified samples to out training data set with similar (or even identical) labels as the sample that they are derived from to get a better approximation for $P(x, y)$. This is done by exchanging δ_{x_i} by some estimate of the density in the vicinity of x_i, say $P_{x_i}(x)$ to get the vicinal risk function

$$R_{vic}(G_\theta) = \int \mathcal{L}(G_\theta(x), y) P_{x_i}(x) dx \tag{2.27}$$

By using data augmentations the models performance is expected to increase by making it less sensitive to overfitting [45].

[3] Note that we only have the delta function explicitly dependent on x. This is because $\{x_i, y_i\}$ is ordered in source-target pairs and hence $\forall i : \delta_{y_i} = \delta_{y_i}$, so adding the y-dependent part would not change the value of Equation 2.27.

2.5.2 Geometric transformation consistency regularisation

The VRM framework is in its simplicity very effective for e.g. image classification as the class for an image is still the same after an augmentation and its original label therefore can be used. However, for the segmentation task, the unchanged-label assumption cannot be made with the same confidence. Mustafa et. al. presented in their 2020 paper a consistency regularisation scheme that partly solves this problem by applying reversible transformations to both source and targets of the training data [46]. By doing this, we can get more than one training example with a perfect label for each of the data points in the training set. The loss function for this training scheme can be written as a sum of a supervised and an unsupervised term

$$\mathcal{L} = \mathcal{L}_s(x, y) + \lambda \left(\mathcal{L}_{us}(u) + \mathcal{L}_{us}(x)\right) \tag{2.28}$$

where the two loss terms are defined as

$$\begin{aligned} \mathcal{L}_s(\mathbf{x}, \mathbf{y}) &= \frac{1}{B} \sum_{i=1}^{B} \left\| G_\theta\left(x_i\right) - y_i \right\|_2^2, \\ \mathcal{L}_{us}(\hat{\mathbf{x}}) &= \frac{1}{rB} \sum_{i=1}^{rB} \left(\frac{1}{M} \sum_{m=1}^{M} \left\| T_m\left(G_\theta\left(\hat{x}_i\right)\right) - G_\theta\left(T_m\left(\hat{x}_i\right)\right) \right\|_2^2 \right), \end{aligned} \tag{2.29}$$

with B the supervised batch size, r the ratio of unsupervised to supervised samples in a training batch, $\{T_1, \ldots, T_M\}$ the set of transformations applied to each data point in the unsupervised training set and λ a the supervised to unsupervised weighting parameter.

As can be seen in Equation 2.29, the unsupervised part of the loss function penalises the model for not making consistent predictions for transformed variations of the same image. Mustafa et. al. [46] showed that this technique indeed can be used to improve model performance.

3
Method

This section describes our preliminary experiments that lead to our chosen training scheme that we call PixMax. We also describe the data sources and the data splits that were used in the training as well as the model, our proposed model training scheme and the metrics we have used to evaluate our models.

3.1 Data

This project investigates how a large amount of unannotated data can be used together with a smaller amount of annotated data through the semi-supervised training framework to create a better model than what could be achieved with annotated data alone. Previous work has been done where performance increase has been investigated based on how the fraction of unlabelled examples are introduced to train the model. It has been shown that significantly better results can be reached by adding a large fraction of unlabelled examples, especially in the sparse setting [47], [48].
Since no really large dataset with both annotated and unannotated floor plan images exists currently, we have chosen to compose our data from two different sources.
The datasets both contain floor floor plans but they both have their similarities and their differences. The assumption that all data comes from the same distribution cannot necessarily be made. In the following sections we will explain where the data comes from and the discrepancies in the data are being dealt with.

3.1.1 Annotated data - The CubiCasa5k dataset

The main source of annotated data in this project is a novel dataset called CubiCasa5k. It was first introduced by Kalervo et. al. in their 2019 paper[1]. The dataset contains 5.000 labelled raster pictures fetched from image scans and they are divided into three categories based on their visual style, as can be seen in Figure 3.1. The annotation of the dataset is rich in the sense of precision and the amount of information contained in each label. Altogether, there are around 80 different object types and room labels are represented by polygons in contrast to other earlier datasets where often rectangles are used for simplicity [49].
The labels of the data has an intricate structure where there are three primary types of labels: junctions, rooms and icons. The junctions are pixel accurate locations for each interest point of types as can be seen in Figure 3.2. The rooms and icons are each represented by a pixel-wise segmentation maps of room and icon classes.

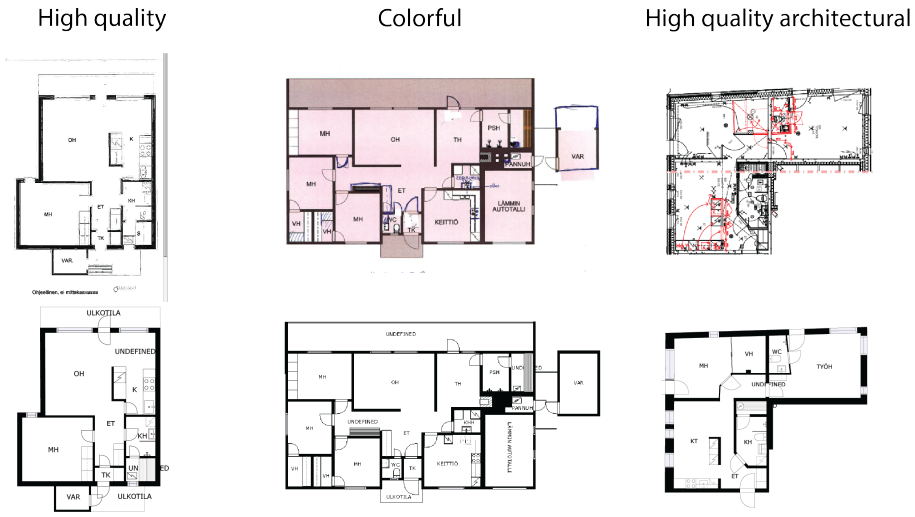


Figure 3.1: Examples of the visual style of the images of the three categories in the CubiCasa5k dataset with their respective labels above. *The images are scaled to fit the page format.*

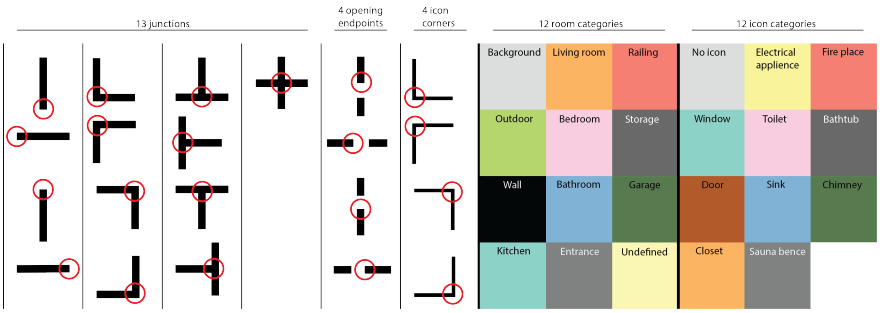


Figure 3.2: A visual representation of all the different annotation categories of the CubiCasa5k dataset. Junctions, openings and corners are lists of coordinates while rooms and icon categories are pixel-wise segmentation maps over the image.

44 Output maps				
21 Interest points (heatmaps)			23 Segmentation maps	
Wall corners	Opening endpoints	Icon corners	Room class	Icon class
13	4	4	12	11

Table 3.1: The structure of the output layers of the network.

3.1.2 Unannotated data - The Lifull Home's dataset

The LIFULL HOME's Dataset [11] is the biggest collection of floorplan image data available for research today. The dataset consists of about 5.31 million images in .jpg format. The images have a high variance in size, colour and quality since it has been collected over some period of time from multiple sources all over Japan. The

architectural qualities of the floor plans as well as many of the graphical representations used to convey information is quite different from the western style used in the CubiCasa5k dataset.

Figure 3.3: Examples of the visual style of the images of in the LIFULL HOME's dataset. *The images are scaled to fit the page format.*

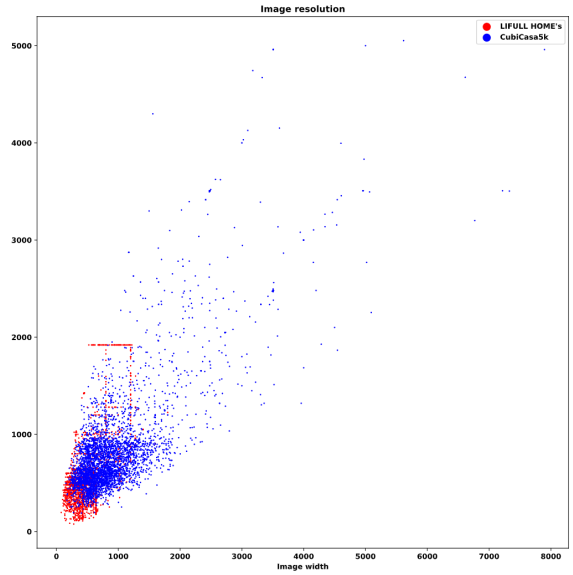


Figure 3.4: The resolution distributions of the different datasets used in the project. A simple random sample of 4000 image instances of each set was used.

The CubiCasa5k dataset is split into a training, a validation and a test set. The training set consists of 4200 data points while the validation and test sets hold 400 images each. Each of the slices are assigned equal proportions of images from each of the visual style categories to not introduce some unwanted bias, see Figure 3.1. For convenience we also chose to use this predefined partitioning.
For the LIFULL HOME'S dataset we had to slice it ourselves. We also had to manually annotate a small portion of it for testing purposes. How this was done is described in Section 3.3.3.
For the validation slice used to determine when to terminate training and to provide the ADAM optimisation algorithm with a time series of the model performance evolution during training we concluded that we had 3 options:

- Using pseudo labels for the validation set.
- Manually labelling a big enough portion of the LIFULL HOME'S dataset to be used as a validation set.
- Use the validation slice of the CubiCasa5k dataset for all the experiments.

We judged that using pseudolabels as annotation in the validation set would be too unreliable and labelling a big enough portion of the LIFULL HOME's dataset would be too time consuming and outside the scope of this project. We therefore decided to use the validation slice of the CubiCasa5k dataset for validation in all our preliminary and final experiments.

3.2 Pseudolabelling

Our proposed method utilises semi-supervised learning through self-learning by letting a model trained on labelled data make prediction on unlabelled data and use these predictions as a basis for creating pseudolabels for further training.
When run on an image, the model outputs its predictions in terms of sets of pixel-wise feature maps that each correspond to either rooms, icons or interest-points. In other words we get a measure for the model prediction on each pixel for every class, but we want the labels to be on the same format as our pre-labelled examples. This leads us to the non-trivial task of picking a way of creating psuedolabels from the model output.
Moreover, if we can't make any of the assumptions stated in Section 2.3, we at least need to do some kind of augmentation to the information of the model output when we create the pseudolabel for for there to be any reason to believe that the model should perform any better after the continued training than after just being trained on the labelled dataset. If $\mathbf{x}, \mathbf{y}$ is our labelled dataset and $\hat{\mathbf{x}}$ is our unlabelled data with corresponding pseudolabels $\hat{\mathbf{y}}$, we can express this as

$$\min_{\hat{\theta}} \mathcal{L}(G_{\hat{\theta}}(x), y | x \in \mathbf{x}, y \in \mathbf{y}) \stackrel{?}{=} \min_{\hat{\theta}} \mathcal{L}(G_{\hat{\theta}}(x), y | x \in \mathbf{x} \cup \hat{\mathbf{x}}, y \in \mathbf{y} \cup \hat{\mathbf{y}}), \tag{3.1}$$

where θ is the parameters of the model after being trained on only labelled data. This is simply to say that if our unlabelled data is not telling us something about the true distribution of $\mathbf{x}$ nothing new can by learnt by sole extrapolation of what is already known. The following sections will explain the tested approaches of finding a suitable way of creating enhanced pseudolabels better than the raw network output.

3.2.1 Statistical approach

The most obvious way of picking pseudolabels would be to use some probability measure on the model output to decide what information to keep. The idea would be that predictions with a high likelihood of being correct would be kept while those with a high prediction uncertainty would be discarded. This could for instance be done in the following way:

- For heatmap classes[1]: From all prediction instances, keep only those with a higher probability of being correct than a certain threshold.
- For the one-hot encoded[2] classes: For each pixel, pick the class that is most likely to be the true class.

[1] The corner classes.
[2] The the rooms and icon classes where each pixel must be one and only one of the classes.

The use of this approach can be motivated by the fact that if we select what information to include in our pseduolabels based the probability of the information to be correct, we can expect the information in our pseudolabels to be correct with that same probability on average. We can express this as

$$\mathbb{E}_{x\sim P_T(x)}[T(x)] = \int T(x) dP_T(x) \tag{3.2}$$

where $P_T(x)$ is the probability density function of any finite statistic $T(x)$ of the data x. In our case $T(x)$ would be the correctness of the model prediction w.r.t the label of. Here, $T(x)$ can be interpreted as what is known as a *conformal predictor* [50]. This means that we by utilising this metric could control the quality of our pseudolabels by requiring a higher or lower probability for the included information to be correct. For instance, we could require a confidence level $\alpha \in [0, 1)$ in all information we chose to include and get

$$\mathbb{E}[T(x)|P_T(x) > \alpha]\mathbb{E}[[\![P_T(x) > \alpha]\!]] \geq \frac{1}{\int_\alpha^1 dP_T(x)}\mathbb{E}[T(x)] \tag{3.3}$$

where $[\![\cdot]\!]$ is the generalised Kroneker delta function:

$$[\![P]\!] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

However, to use this approach we are required to have a good measure for how confident we can be that a prediction is correct. Since the model outputs for the one-hot classes is in form of scores for all classes on each pixel, it is a reasonable assumption that there could be a correlation between a score of a class and the probability that the prediction is correct for that class. We can also transform the model output to something with the same form as a discrete probability distribution by passing it through the Softmax function, described in Section 2.4.1.

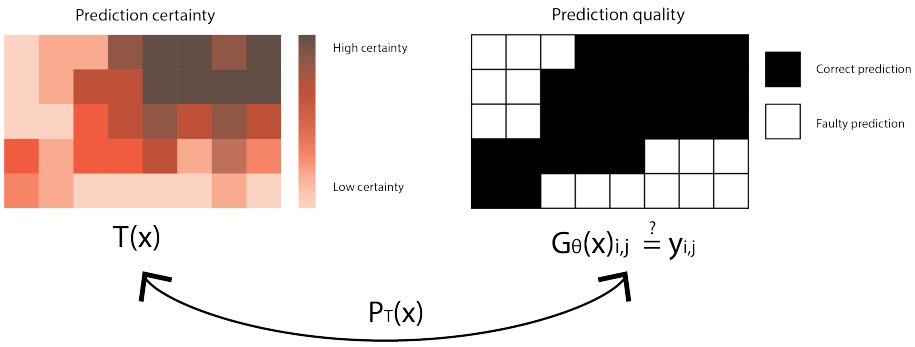


Figure 3.5: An example of what a correlation between the prediction certainty and correctness could look like. The pixels that the network is most sure about is to a high extent also the pixels that are classified correctly.

To confirm that Softmax of the model output itself is indeed a good proxy for $P_T(x)$, we had to run a few experiments. For the room and icon classification this would mean the pixels with high prediction certainty would be classified correctly more

often than those with low prediction certainty. To see if this was the case, we used the the *absolute loss function*

$$\mathcal{L}_{ABS}(x, y) = \frac{1}{P} \sum_{p \in P} |y_p - x_p|_1 \tag{3.5}$$

as our statistic T. This is the normalised pixel-wise distance function that returns the Manhattan distance between the network prediction x and the correct label y. Under the assumption of the described scenario, we would expect to see

$$\frac{\partial}{\partial \alpha} \mathbb{E}_{P_T(x)>\alpha}[\mathcal{L}_{ABS}(x, y)] \leq 0 \qquad \forall \alpha \in (0, 1) \tag{3.6}$$

.

This is to say that we expect the loss to be lower, and hence a higher quality of the information, when we are more conservative with what information to include w.r.t the certainty of the prediction. An easy way to check if this assumption holds is to use what is known as calibration plot [50], [51] where the correctness is plotted as a function of the model certainty.

The results of this experiments can be found in Section 4.1.1. It showed that the correlation was to weak to be useful for our purpose.

3.2.2 Post-processing technique

Another possible approach to the problem of creating a pseudolabel from the networks raw output is to run it through some kind of function that can enhance it's quality. If we call this function q, this would mean that we could expect to see

$$\mathcal{L}(G_\theta(x), y) \geq \mathcal{L}(q(G_\theta(x)), y) \tag{3.7}$$

for some arbitrary loss function $\mathcal{L}$ that in some sense measures the quality of the prediction. Although the existence of such a function q does not seem too unreasonable, we need to address the question of why this transformation is not implicitly learned by the network by a different parameterisation $\hat{\theta}$ to get $G_{\hat{\theta}}(x) = q(G_\theta(x))$ if it reduces the loss of the model.

One possible explanation for why a function with these properties could exist is if it has access to different information than the model itself that can be used to enhance its performance. In the model that are used for this project, we can do exactly this. The model output is naturally divided into three distinct categories of predictions; pixelwise room and icon segmentation maps and heatmaps of junctions. This means that we can potentially use the junction heatmaps to infer better room and icon segmentation. If we for convenience define the network output $z := G_\theta(x)$ so that z_h represents the heatmap channels and $z_{r,i}$ the room and icon channels, we can reformulate our criterion on a sufficient function q in equation 4.3 to be explicitly dependent on z_h

$$\mathcal{L}_{r,i}(z_{r,i}) \geq \mathcal{L}_{r,i}(q(z_{r,i}|z_h)) \tag{3.8}$$

where $\mathcal{L}_{r,i}$ is some loss function that only cares about the room and icon channels. With this reasoning we can conjecture that a function with the properties of q may

exist, but it is in no way a guarantee for it's existence and we have no general algorithm for finding it. In our setting where we are concerned with the parsing of floor plans, there are however a few empirical observations to be made that can be used to guide our search of a sufficient q. Some of these are the following:

- Rooms are made up of simple polygons with orthogonal corners.
- Icons are rectangles.
- Every region completely surrounded by walls and apertures contains only one room type.
- The region that is outside the outermost closed wall-loop is the background.

The hope is that we can use these observation in combination with the output from the heatmaps to create a prediction that is better than the raw network output in the sense that it gives a substantially lower loss for some loss function $\mathcal{L}$. Ideally this should also be the case for the cross entropy loss function $\mathcal{L}_{CE}$ since this is what we use to train the network.

Kalervo et. al [1] proposed in their work a novel post-processing algorithm with the structure of q that we we will call q_{cc}. It is a procedural algorithm with the aim to extract all elements of interest in the floor plan including walls, rooms openings and icons. It can be understood as 4 distinct steps that are being executed in sequence in the following way:

- **Inferring wall skeleton**
  The algorithm starts by connecting pairs of junctions based on their position, type and orientation. This means that if there are two junctions that are close to being vertically or horizontally aligned and they have a joining direction facing each other they will be connected by a line. The junctions are also batched together so that multiple neighbouring junction points of the same type gets mapped to just a single point to avoid crowding. The result from this part of the algorithm is a "skeleton" of possible wall centre lines.
- **Inferring walls**
  The wall skeleton is used together with the wall segmentation map to construct a final wall prediction. First the skeleton is pruned by removing lines that do not consist with the wall segmentation and the wall thickness is then decided based on the intensity profile of the wall segmentation map.
- **Inferring rooms**
  Next the processed room segmentations are calculated based on outcome from the prevous two steps. The algorithm searches for all junction triplets that spans a rectangle without any junctions inside it to create a grid of the interior of the floor plan. For each of the grid cells, a voting mechanism is being applied that samples from the pixel predictions of the room segmentation maps to decide what room type to assign for for the cell. Adjacent cells of the same class are being merged together if and only if there are no fully separating walls between them. The same mechanism is being used to find the icons but here the icon heatmaps and segmentations are being used.
- **Inferring apertures**
  The last step is to find the doors and windows by utilising the corresponding endpoint heatmaps. First all points that do not coincide with wall segments the processed wall segment map are abandoned. The remaining points are

then matched into window and door segments and the width of the aperture is chosen to be the same as the host wall where it is located.

Figure 3.6: Examples of what q_{cc} does with the segmentation maps from the room channel. The examples are randomly sampled from the test set of the CubiCasa5k dataset. The most visually prominent changes is that all segmentation have been translated into simple polygons.

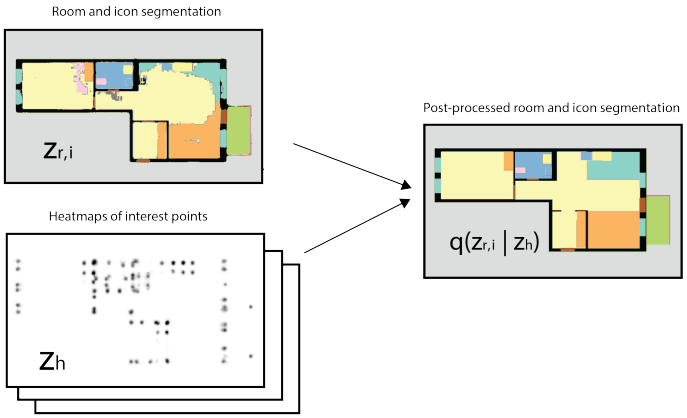


Figure 3.7: The concept of how the post-processing algorithm q_{cc} works. Given the predicted junction heatmaps and the room and icon segmentations, q_{cc} can "clean up" the segmentations e.g. by inferring a closed room between 4 suitable L-type corners.

As previously mentioned, we want the choice of q to give us better performance than the raw network output. To confirm that this can be expected with q_{cc} we measure the loss before and after applying q_{cc} to the network prediction. The results from these experiments can be found in Section 4.1.2.

3.2.3 PixMax pseudolabelling technique

Similar to the method proposed in Section 3.2.2, we might choose a function $q : \mathbb{R} \to \mathbb{R}^B$ operating on predictions of several different non-destructive rotations and flips of the input image. Then the batch $\mathbf{b} := G_\theta(q(\hat{x}))$ of B predictions can be oriented back and combined to form our pseudolabel $\hat{y} = \hat{q}(\mathbf{b})$ for the given image in the unlabelled dataset. We use all unique orientations of 90 deg rotations and flips,

which gives $B = 8$ combinations. Here, $\hat{q}(\mathbf{b})$ is set to the mode-function, pixel-wise selecting the most common value in the batch $\mathbf{b}$ of predictions.

Also, instead of generating pseudolabels for all images in the unannotaded dataset, we could try to find a scalar measurement for how accurate the prediction is. As a proxy of accuracy, we implement a function for calculating a scalar β for how confident the network is of the prediction on a image. Then we can further improve the quality of the pseudolabels by only using the labels that satisfy $\beta(x) > \tau$ for a threshold hyper-parameter τ. Equations 3.9 and 3.10 describe this (denoted CONF$(\cdot)$ in Algorithm 1). Here $c_{mc,j,i}$ is the value of the most common class at pixel (i, j) of batch $\mathbf{b}$, and $c_{b,j,i}$ the value at the b^{th} prediction. $[\![\cdot]\!]$ is the generalised Kroneker delta function (Equation 3.4),

$$\beta_{j,i} = \frac{1}{B} \sum_{b \in B} [\![c_{b,j,i} = c_{mc,j,i}]\!] \tag{3.9}$$

,

$$\bar{\beta} = \frac{1}{HW} \sum \beta_{j,i} \tag{3.10}$$

.
Initial results in 4.1.3 was promising and this method was chosen for the PixMax self-training.

3.3 PixMax self-training

Once we have found a way of generating pseudolabels of good quality we might use this in a self-training scheme to further improve a supervised model.

As can be seen in Figure 3.8, the PixMax model training scheme consists of three phases; a supervised learning phase, a pseudolabelling phase and a semi-supervised self-training phase. In the supervised training phase a model is trained until convergence in a purely supervised manner. We are then using this model to create pseudolabels for an unlabelled dataset with similar features as the original dataset. We do this by combining predictions for multiple light augmentations of the images into a single label as described in 3.2.3. If the label passes the β-threshold check, then it is accepted and will be used as a pseudolabel for the second phase of the model training. In the last phase the trained models weights and biases are copied but new hyperparameters are initialized. The model is once again trained until convergence with a combined dataset consisting of both the original labelled dataset and the accepted portion of the pseudolabelled dataset.

The following sections describe how we implement the network model and what augmentations we have applied to diversify the data as well as the details of how we evaluate the performance of the trained models.

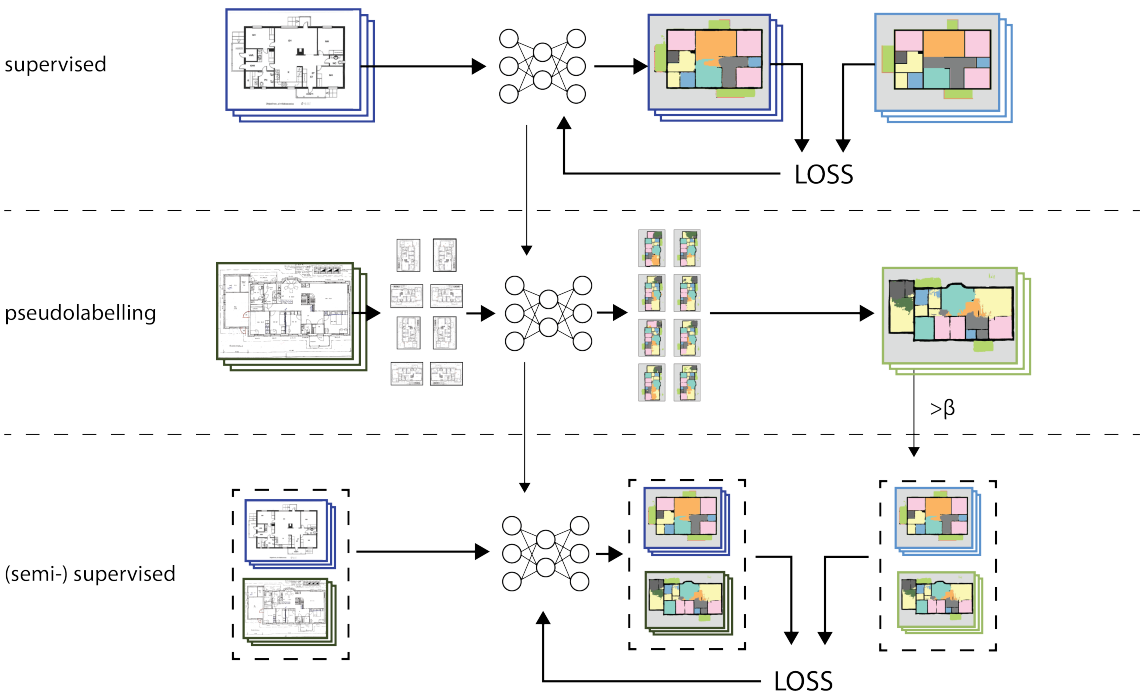


Figure 3.8: The PixMax model training scheme. Light blue: The labels for the labelled dataset. Dark blue: The images and predictions for the labelled dataset. Dark green: The images and model predictions for the images in the unlabelled dataset. Light green: The pseudolabels created by the model in the pseudolabelling phase.

3.3.1 Network model

The model used in [2] have achieved state of the art results and we have therefore chosen it as our model to be used. The model converts the floor plan image through two intermediate representation layers. The first step is the network inference step which outputs 44 maps of interest points and pixel-wise semantics. Second these are converted through integer programming (IP) to form a set of geometric primitives. Note that the sole purpose of the interest points is to infer points to be used to construct the geometric primitives. Finally a step of post processing is applied to form the vectorized output of geometries with class labels.
The architecture of the model is borrowed from the ResNet-152 [12] model with an altered output layer as shown in Table 3.1. Figure 3.9 shows the high-level structure of the model.

Algorithm 1: PixMax Pseudolabelling scheme

Input : Set of unlabelled images: $\hat{\mathbf{x}} = \{\hat{x}_i : i \in \{1, \ldots, M\}\}$.
Pre-trained model G_θ with parameters θ.
Parameters: Threshold τ, for β metric.
for $\hat{x}_i \in \hat{\mathbf{x}}$ **do**
  $\mathbf{b} \leftarrow G_\theta(q(\hat{x}_i))$ ▷ Batch of predictions
  $\hat{\mathbf{y}}_\mathbf{i} \leftarrow \hat{q}(\mathbf{b})$ ▷ Inverse augmentation
  $\hat{y}_{mcp} \leftarrow \text{MODE}(\mathbf{b})$ ▷ Most common prediction
  $\beta_i \leftarrow \text{CONF}(\hat{y}_{mcp}, \hat{\mathbf{y}}_\mathbf{i})$ ▷ Calculate confidence
Output : Images with pseudolabels: $\{\{\hat{x}_i, \hat{y}_i\} : i \leq M, \beta_i \geq \tau\}$.

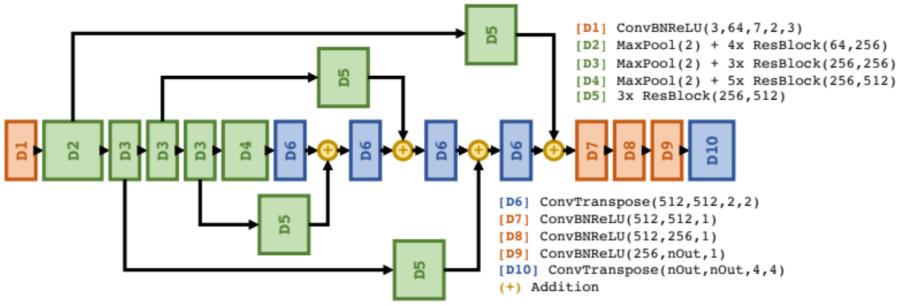


Figure 3.9: *A simplified illustration of the architecture of the model used. Image from [1].*

3.3.2 Data diversity augmentations

Computer vision tasks benefits greatly from having augmentations applied to the data to generate a more diverse dataset of which the model can better generalise from. Hence, we have used a few common augmentations to vary the data during the training. All images are either cropped or resized to a target size. This speeds up the training time of the network and also allows the network to learn features from different scales of floorplan drawings.

- **Resize with padding**. Resizing the image to the target size keeping the aspect ratio intact by padding the needed pixels with zeros (black pixels and no label).
- **Crop to size**. Cropping out a part of the image and label to get a smaller piece for the training.
- **Rotations**. Random 90 deg rotations of the image and label.
- **Color adjustments**. Weakly adjusting the brightness, contrast and saturation of the image.

3.3.3 Evaluation datasets

A model that has been trained on the CubiCasa5k[1] dataset in a exclusively supervised fashion will be used as our primary benchmark for our semi-supervised model. It is clearly reasonable to expect that this model will have a positive bias on it's performance on the CubiCasa5k dataset relative to our semi-supervised model. To be

able to get a reliable measure and fair comparison of the performance of our model we needed to be able to evaluate models on both the labelled and the unlabelled datasets.
For our unlabelled dataset, we therefore needed to create enough labels to be able to evaluate the models performance ourselves. For this task we used a online image-annotation pipeline called labelbox [3] that supports polygon segmentation of images. We selected a random subset of 50 images from the LIFULL HOME'S dataset and manually annotated them with the same set of room and icon classes as the CubiCasa5k dataset. To not introduce unnecessary bias to the results we tried to use the same style and conventions as possible when performing the annotation. All our annotations are publicly available here: github.com/xRadne/LH_annotations.

3.3.4 Evaluation metrics

We have chosen a set of performance metrics to evaluate our model in reconciliation with what's common practice for semantic segmentation [52]. Although our model predicts both room- and icon segmentation maps as well as heatmaps of interest points, we have chosen not to evaluate the model's performance on the heatmaps for several reasons. Annotating interest points is a very time-consuming task and using a really small test set would result in very wide confidence intervals that does not provide much information about the model's actual quality. Also, this information will be captured indirectly by the segmentation evaluations of the polygonised (post-processed) predictions since these are constructed using the interest points (3.2.2). Moreover, to convert the heatmaps to discreet point locations, a threshold value has to be chosen[4] — that makes the recall and accuracy valued somewhat arbitrary.
With C classes and $n_{i,j} = \sum p_{i,j}$ the number of pixels of class i predicted to be of class j, our chosen metrics can be described in the following way:

- Overall accuracy - *The ratio of correctly classified pixels.*

$$Overall\ Acc = \frac{\sum_i n_{i,i}}{\sum_i \sum_j n_{i,j}} \tag{3.11}$$

- Frequency weighted averaged accuracy - *The ratio of correctly classified pixels, weighted for the occurrence frequency of each class.*

$$FreqW\ Acc = \frac{1}{\sum_i \sum_j n_{i,j}} \frac{\sum_i n_{i,i} \sum_j n_{i,j}}{\sum_j n_{i,j} + \sum_j (n_{j,i} - n_{i,i})} \tag{3.12}$$

- Mean intersection over union - *The overlap of the predicted and true pixels, averaged over all classes.*

$$Mean\ IoU = \frac{1}{C} \frac{\sum_i n_{i,i}}{\sum_j n_{i,j} + \sum_j (n_{j,i} - n_{i,i})} \tag{3.13}$$

[3] The annotation tool used can be found here: https://labelbox.com/.

[4] For the evaluation to be fair we would need to create a metric based on the distance between each predicted point and the closest ground truth point of the same category. This would require a conversion of the heatmaps to a boolean map for each layer through the use of a threshold function.

These four parameters are picked to give fair metric for the model performance. Since some of the room classes are heavily under-represented compare to other classes in both the datasets used, it is a legitimate hypothesis that the model will not be able to recall these classes to the same extent as the others. For this reason, the *Frequency weighted average accuracy* is in a sense the metric that gives the most nuanced grade for the actual performance of the model. All models are evaluated both with and without Test Time Augmentations (TTA) for a complete result. The TTA is a set of light augmentations that is applied to each image in the testing set during model evaluation. The final model prediction is then defined as the pixel-wise most common class prediction. This technique reduces the statistical fluctuations and therefore gives a better prediction on average.

In addition to this we also evaluated the models *mean precision* and *mean recall* for rooms and icons. These metrics can be described as follows:

- Mean Precision - *The fraction of relevant instances among the retrieved instances.*
$$Mean\ Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{3.14}$$
- Mean Recall - *the fraction of relevant instances that were retrieved.*
$$Mean\ Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{3.15}$$

3.4 Implementation details

3.4.1 Hardware specifications

The code for this project was all written in python with pytorch v1.6.0 for the deep learning implementations. This framework was chosen since the earlier work that the project is based on uses this framework. It is also one of the currently most popular and widely supported implementations for building deep learning models. The code for this project can be found at .
We had access to gpu-clusters for running our code throughout the second part of the project. For model evaluations and small scale experiments we used a cluster with a Nvidia Tesla T4 GPU with 16GB RAM and for pseudolabel creation and model training we used gpu-clusters with the following specifications:

- Nvidia Tesla V100 SXM2 GPU with 32GB RAM
- 2 x 8-core Intel®Xeon®Gold 6244 CPU @ 3.60GHz (total 16 cores)
- 387GB SSD scratch disk

The training time naturally varied between different runs. The maximum training time of about 9 hours was for a run of 100 epochs with a combined dataset of about 14.000 data points consisting of both annotated and pseudo-annotated images.

3.4.2 Experimental setup for model training

Here we list our settings and hyperparametes used in training that have been consistent in all runs as well as the search ranges for parameters we have tried to optimise. Since optimizing for all parameter would result in a search space too large for the scope of this project, we have limited out search for the best model candidate to the subset of the parameters that we think are the most rewarding and interesting for our purpose.
For all experiments, the pretrained model from [1] have been used as the starting point but with new hyperparameters. The model is a ResNet-152 [12] with a modified output layer that has been pretrained on ImageNet [13] and MPII Human Pose dataset [53]. The ADAM optimiser as described in Section 2.1.2 was used with an initial learning rate of 1e−4 and a scheduled learning rate drop of $0.5^{\frac{k_{curr}}{k_{max}}}$ where k_{curr} and k_{max} are the indices of the current and last epoch respectively. The initial learning rate was picked according to what we found to be conventional and the learning rate drop schedule was picked based on empirical results from our preliminary experiments. We discovered early that the models performance rarely changed significantly after more than 80 epochs of continued training from the pretrained model state. For this reason we choose to train all models for no more than 100 epochs from the pretrained state. For all runs a — for the model previously unseen — subset of the CubiCasa5k [1] datset was used as the validation set. The reason for this was simply the lack of a big enough manually labelled set of LIFULL HOME's [11] images to be able to determine a reliable stopping criteria for the training.

The model we label *Ours* in Section 4.2 is the best performing model from our final set of runs. Since hypothesised that the β-value was an important hyperparameter we trained models with β-values ranging from 0.95 to 0.99 with intervals of 0.01, everything else identical. From our preliminary experiments we saw that both the reference model and the models that we tested were heavily biased to predicting false positives the *undefined* room class. To take this into account, we tried training all our models with a setting that would ignore the contribution of the undefined room class to the loss. It would then weight the loss to compensate for the proportion of the image removed. Figure 3.10 shows an how the overall accuracy varies between all our final models and similar plots for the other performance statistics can be found in Table 4.3, Appendix A.

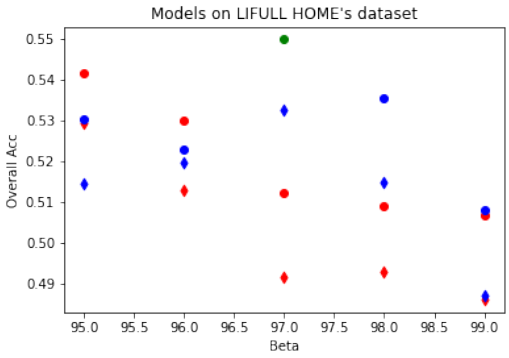


Figure 3.10: The overall accuracy on the LIFULL HOME'S test set for models trained with different β-thresholds for selecting what pseudolabels to use. Blue: Models trained using the *ignore index* setting described above. Red: Models trained without the ignore index setting. ●: Models evaluated with test-time augmentations. ♦: Models evaluated without test-time augmentations. ●: The best model that we found in our final experiments. Called *ours* in the following section.

In this set of runs we trained models with a combined data set of 4200 annotated images from the CubiCasa5k dataset [1] and different numbers of pseudo-annotated images from the LIFULL HOME'S dataset [11] based on the β-threshold.

The model labelled with *Best reproduced* is from the best performing model that we managed produce by following the training schedule in our reference project [1].

4
Results

In this section we will first present the results from our preliminary experiments with different techniques for creating pseudolabels for the unannotated dataset. For each technique we will also provide a short discussion about what is evaluated and how the results have been interpreted to facilitate the reading and motivate our decisions on how to proceed. In the last section of this chapter we will present the performance results for our model and comparisons with our benchmark model from Kalervo et. al.[1] that our project is based on, as well as our best reproduced model from that paper.

4.1 Pseudolabelling techniques

This section describes the results for the preliminary tests that were performed to inform the decisions on what technique to use for pseudolabelling. In Section 4.1.3 we present our findings that led to the choice of the technique used for the final results presented in Section 4.2.

4.1.1 Statistical approach

As discussed in Section 3.2.1, if we want to use only the regions of an image where the prediction is correct with a high probability, we need to have a proxy for this value with a reasonably high precision since the value itself is unknown to us. Here we present our findings for the correlation between our hypothesised best candidate, the maximum of the softmaxed model output, and the label value.
To perform this test we had to know the true label of the images. To not introduce any unwanted bias, the test set of the CubiCasa5k dataset was picked for this experiment since these images are previously unseen by the model and they have what can be considered to be true labels. We also used the best[1] model provided by [1] since this is the model we want to use for creating our pseudolabels. We also chose to only look at the room channels as we expect this to be representative for the model's performance. With $z_{room} := \{G_\theta(x)_i \mid i \; is \; a \; room \; channel\}$ only the room channels of the model output, our question can be expressed the following way:

$$Corr(\max_i(Softmax(z_{i \in room})), \mathcal{L}(z_{i \in room}, y)) \overset{?}{\neq} 0 \tag{4.1}$$

[1]The model with the lowest 'validation loss with variance' according to the paper.

Our first investigation was to look at the the distribution of the models *uncertainty score*. We define this as $US(G_\theta) := 1 - \frac{1}{|P|}\sum_{p\in P} \max_i(Softmax(z_{i\in room,p}))$ where p are the pixels in the image. Note that this is the sum of the sofmaxed output of all channel but the one with the highest value[2]. It is then clear to see that this can be interpreted as the models prediction of the probability that the pixel is of any other class than what it actually predicted it to be. Under the assumption stated in Section4.1, we expect to see a distribution that is roughly proportional to the distribution of correctly classified pixels when averaged over many images.

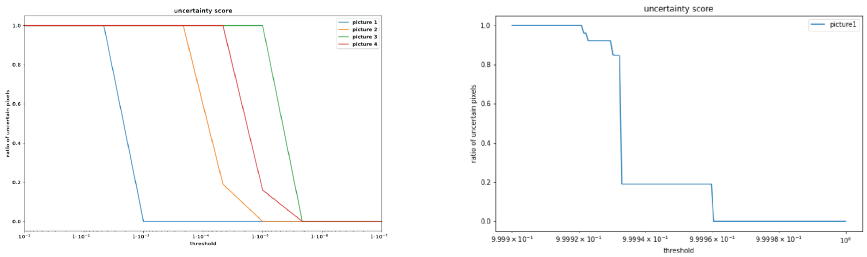


Figure 4.1: Left: The proportion of the pixels with $US(G_\theta) \geq x$ for 4 different images. Take note of the logarithmic scale on the x-axis. Right: A zoom-in on the graph of the first image with a higher resolution.

Figure 4.1 shows the accumulated proportion of pixels with a higher uncertainty than the value on the x-axis. This result can be interpreted in terms of what is known as a calibration plot [50]. The result tells us primarily two things:

- The model is much more certain about it's prediction than what it has reason to be. It's uncertainty score seems to often be $\leq 10^{-3}$ for every pixel in an image.
- The model seems to be roughly equally certain about all pixels in an image, but the average certainty between images often varies by multiple orders of magnitude.

These are two clear indications that the distribution pixel uncertainty does not seem to have a direct strong correlation with the distribution of $\mathcal{L}$ for a set of images. However there might be some transformation that could fix this issue.
To investigate this the next experiment was designed to see how big decrease in loss could be expected when ignoring different proportions of the pixels when calculating $\mathcal{L}_{abs}$. Since it was learnt from the first experiment that the average pixel uncertainty of images varies by multiple orders of magnitude, the pixels were sorted based on their uncertainty score independently for 100 images. We calculated the loss after removing different proportions of the pixels with the lowest predicted certainty and and weighted the resulting value based on the fraction removed.

[2]Since for $x^{(k)}$ *with* $i, j \leq k : Softmax(x_i) = 1 - \sum_{i\neq j} Softmax(x_j)$

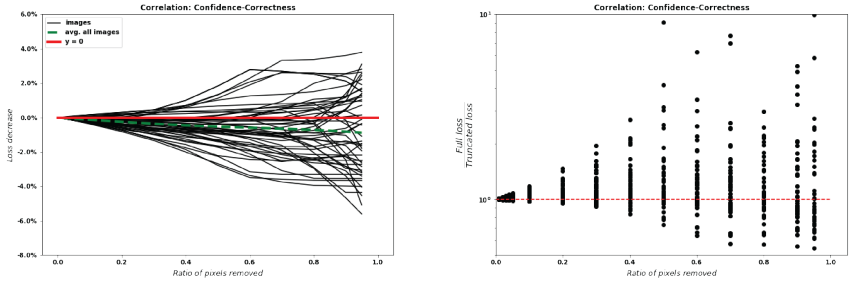


Figure 4.2: Left: The decrease in $\mathcal{L}_{abs}$ as a function of how big proportion of the pixels removed for 100 images. The green, dashed line shows the average over all images. The values are calculated at fixed intervals and interpolated in between. Right: The quotient of the loss of the whole image and the truncated image with respect to the fraction of pixels removed. Note the logarithmic y-axis. *All values are weighted to compensate for the fraction of the pixels removed and the size of the image.*

As the green line indicates in Figure 4.2, the average per-pixel loss seems to be lower for pixels with high certainty score. However, the average decrease in loss is only a couple of percent, even when only keeping a 10^{th} of the pixels and hence removing 90% of the training data.
Even though it seems like the assumption stated in Section4.1 holds, the correlation is not strong enough to be useful for being a good proxy for $P_T(x)$ as discussed in 3.2.1. Therefore this approach was abandoned.

4.1.2 Post-processing technique

Our second attempted approach was to use the integer programming algorithm as proposed by [1] in the way discussed in Section 3.2.2 to create pesudolabels from the network output. With the proposed post processing function q_{cc} and the network output $G_\theta(\hat{x}) := z$ the pseudolabel $\hat{y}$ would be calculated as

$$\begin{cases} \hat{y}_{r,i} = q_{cc}(z_{r,i}|z_h) \\ \hat{y}_h = z_h, \end{cases} \tag{4.2}$$

where the subscripts r,i and h each corresponds to the channels of the predicted rooms, icons and heatmaps. The motivation behind using q_{cc} as a post processing algorithm would be the hypothesis that this on more often than not brings the prediction closer to the "true" label of the image, and therefore gives a lower loss. To confirm that this is a feasible method we would need to confirm

$$\mathcal{L}_{r,i}(z_{r,i}) \overset{?}{>} \mathcal{L}_{r,i}(q(z_{r,i}|z_h)). \tag{4.3}$$

We ran the experiment for L_{ABS} and L_{CE}. As can be seen in Figure 4.3 and Figure 4.4, the loss actually increases on average after performing q_{cc}. This means that the proposed post-processing algorithm actually makes the model classify fewer pixels correctly than the raw output from the network.

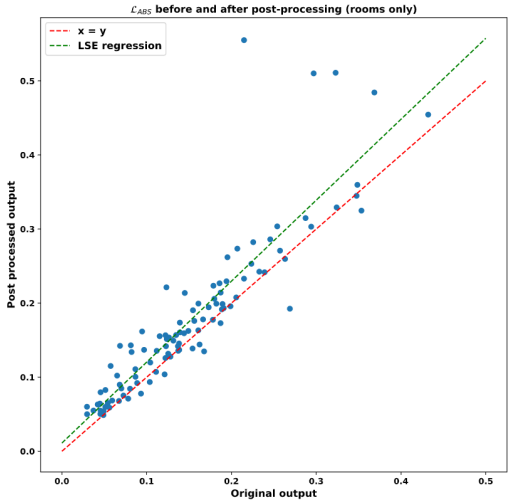


Figure 4.3: The correlation between $\mathcal{L}_{\mathcal{ABS}}$ before and after q_{cc} for 100 images.

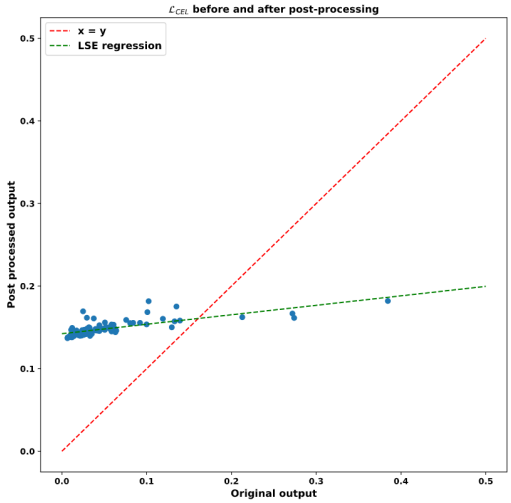


Figure 4.4: The correlation between $\mathcal{L}_{\mathcal{CE}}$ before and after q_{cc} for 100 images.

In Figure 4.3 we can see that the regression line is rather close to the identity line, indicating that the post-processing algorithm's negative effect on the loss is actually not too strong. It could also be augured that even though the loss is increasing, the post processing could still be a good idea since it may capture high-level features in the data to a greater extent as discussed in Section 3.2.2. However, we decided that the increase in average loss was a strong enough indication that the method was not going to give the desired results and therefore this approach was abandoned.

4.1.3 PixMax pseudolabelling technique and model training scheme

Our next suggested approach to creating reliable pseudolabels was to use averaging over weak augmentations of the data. As discussed in Section 3.2.3 we came up with the scalar measurement β that quantifies how large discrepancies there are between the different predictions for all the augmented versions of the same image. The higher the β value is for an image, the higher conformity in the set of predictions. Figure 4.5 shows the β-values for a large set of images in the LIFULL HOME's dataset and it resembles as expected a gamma distribution. It has a sample mean of $\bar{x} = 0.929$ and sample std $s = 0.0359$.

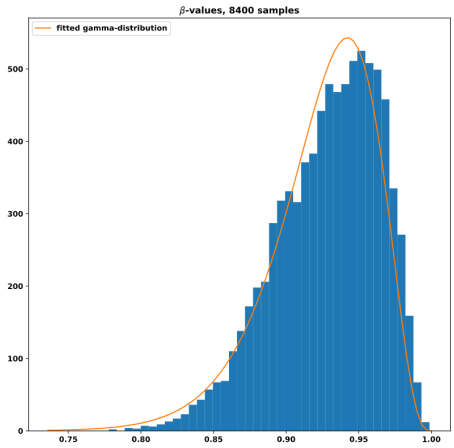


Figure 4.5: A histogram over the distribution of β for 8400 images from the LIFULL HOME'S dataset. The best sample fit for the gamma-distribution has the shape parameter $k = 4.22$ and the scale parameter $\theta = 55.69$.

This is the method for pseudolabeling that we used to train the model and the results in Section 4.2 hence comes from a model trained with this technique. For our final runs we used hyperparameters that are specified in 3.4.2 and for the β-value threshold we tried all values between 0.95-0.99 with a step of 0.1. This resulted in different number of psuedolabelled images as can be seen in Table 4.1.

β-threshold	no threshold	0.95	0.96	0.97	0.98	0.99
# of images	40000	9680	6295	3270	1156	130
% pseudolabelled images	90.5	69.7	60.0	43.7	21.6	3.0

Table 4.1: 1^{st} row: The β-thresholds used for model training in the PixMax training scheme. 2^{nd} row: The number of images with a β-value larger than the tested set of thresholds. 3^{rd} row: The percentage of pseudolabelled images used in the model training scheme (4200 labelled examples used for all runs.)

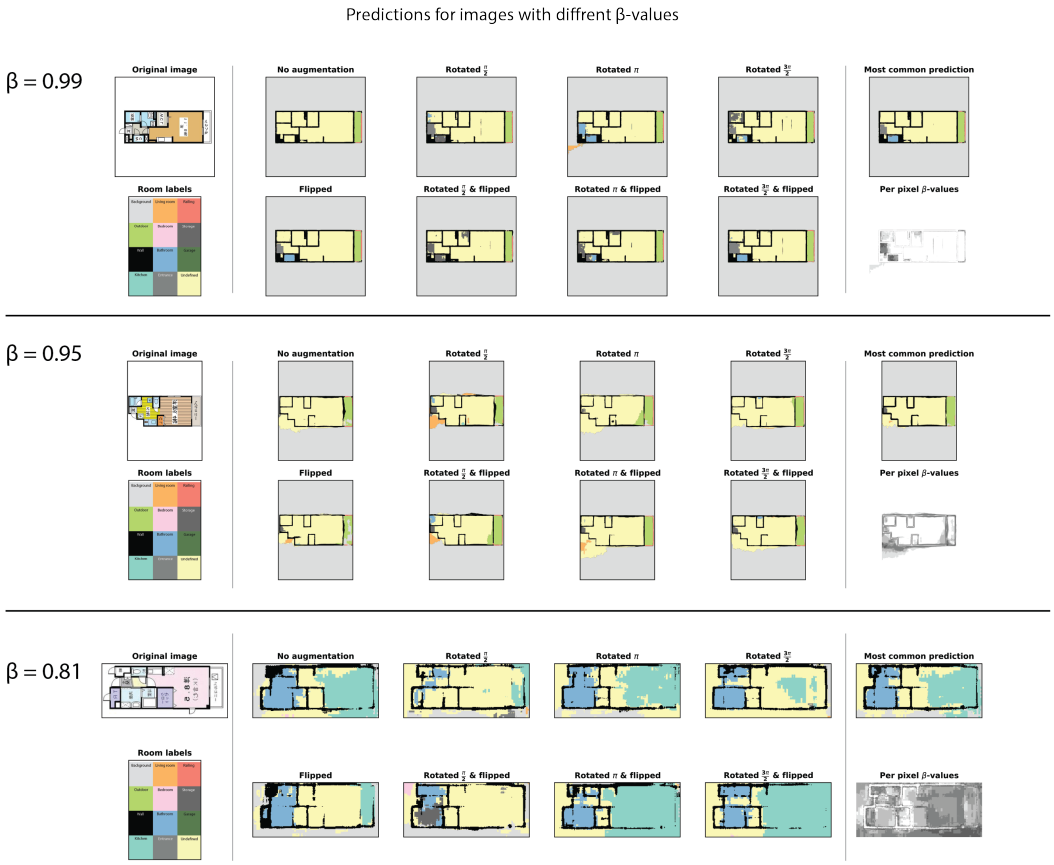


Figure 4.6: Examples of predictions images with different β-values. Column 1: The original image and the room colour legend. Column 2-5: The predictions for each of the augmentations. Column 6: The resulting pseudolabel (most common pixel prediction) and per-pixel $\beta_{i,j}$-value maps for the images.

Figure 4.6 gives us an indication of what the β-value for an image tells us. For images with a high β, the model is consistent in its predictions and hence the variation in the prediction maps is low. For a small β-value on the other hand, the model have a large spread in its predictions, which can be interpreted as a higher uncertainty in its predictions.
We trained several models (listed in Appendix C) with pseudolabels over the different β-thresholds. For the models coming out of the these runs, the one trained with a β-threshold of 0.97 gave us the overall best results and is therefore the one that we used to produce the results given in Section 4.2.

4.2 Results for PixMax model training scheme

Here we present a comparison the performance of our best model, called *Ours*, together with the performance of the current state of the art performance achieved by Kalervo et al.[1], called *CC* or *CubiCasa* and our best reproduction, *rep*. How these models were initiated, trained and selected is fully described in Section 3.4.2. All tables shows performance metrics in percent.

Table 4.2 is a comparison of per class-performance on the measures IoU and precision. The benchmark models and our best model have all been tested on our test set partition of LIFULL HOME. Some of the classes (sauna bench, fire place, chimney, garage) does not occur in our test set and can not be evaluated.

	IoU			Precision			Recall		
Class	CC	$CC_{rep.}$	Ours	CC	$CC_{rep.}$	Ours	CC	$CC_{rep.}$	Ours
No Icon	94.9	94.4	**95.1**	99.0	99.3	**99.6**	**95.9**	95.0	95.4
Window	**20.9**	14.1	18.3	**50.6**	32.4	40.8	**26.3**	20.0	25.0
Door	**33.6**	25.8	26.0	**34.9**	27.1	26.7	89.5	84.3	**90.3**
Closet	5.1	5.6	**8.2**	6.7	7.8	**9.2**	17.5	16.2	**43.4**
Electrical Appl	13.1	11.3	**13.7**	**17.3**	12.9	15.8	34.9	48.1	**50.5**
Toilet	**45.7**	9.8	34.2	**49.3**	11.7	38.1	**86.5**	38.6	76.8
Sink	**11.6**	7.9	7.9	**25.7**	11.0	12.4	17.3	**21.8**	17.7
Sauna Bench	-	-	-	-	-	-	-	-	-
Fire Place	-	-	-	-	-	-	-	-	-
Bathtub	10.2	0.0	**16.5**	10.2	0.0	**16.6**	**95.3**	0.0	93.9
Chimney	-	-	-	-	-	-	-	-	-
Background	**84.0**	75.5	70.6	92.1	**94.7**	94.3	**90.5**	78.8	73.8
Outdoor	**47.6**	46.7	47.1	**56.7**	52.8	55.5	74.7	**80.1**	75.6
Wall	**76.8**	74.7	72.4	**91.9**	87.9	87.4	82.3	**83.2**	80.7
Kitchen	**21.4**	5.5	16.9	**22.2**	5.6	17.0	85.8	86.8	**97.1**
Living Room	16.2	**30.6**	15.1	17.5	**33.8**	16.3	69.0	**76.1**	65.7
Bed Room	11.3	**40.4**	12.7	12.9	**52.7**	13.1	47.5	63.4	**83.4**
Bath	20.1	**21.6**	15.0	22.6	**25.9**	16.2	64.5	56.5	**66.8**
Entry	3.9	**9.5**	0.6	6.3	**17.7**	0.6	9.3	17.0	**42.4**
Railing	**20.0**	2.3	15.4	**20.2**	2.3	15.9	94.3	**95.4**	83.2
Storage	9.1	**14.2**	6.5	9.8	**23.3**	7.6	**54.1**	26.9	31.8
Garage	-	-	-	-	-	-	-	-	-
Undefined	5.5	5.6	**6.4**	**79.6**	49.2	77.8	5.5	5.9	**6.5**

Table 4.2: Per class-comparison between CubiCasa's (CC) model[1], our best reproduced model of CC and our best model achieved using PixMax. All models evaluated on our LIFULL HOME's test set. Note that classes with a (-) is not present in the test set and can not be evaluated.

In tables 4.3 and 4.4 we summarise the performance of the models on the CubiCasa5k test set as well as our annotated LIFULL HOME's test set. We compare the performances in terms of accuracy, mean recall and mean precision of icons as well as rooms.
Following the scheme of [1] we also apply a test time augmentation (TTA) in which the final predictions are based on the mean of four rotations of the input image. The image was rotated by 90 degrees each time. As can be seen this method seems to give better accuracy, recall and precision both with and without integer programming, IP (as described in chapter 3.2.2).
Note that both Table 4.4 and Figure 4.7 show the models results on the LIFULL

HOME's dataset.

CubiCasa5k dataset	Icon			Room		
Model and method	Acc	Recall	Precision	Acc	Recall	Precision
CubiCasa	97.4	75.7	61.3	81.2	71.0	68.6
CubiCasa + IP	97.0	84.9	49.2	78.9	70.0	63.9
CubiCasa (TTA)	**97.6**	83.2	**61.5**	**82.7**	**73.9**	**69.8**
CubiCasa (TTA) + IP	96.7	**86.2**	45.2	77.3	69.9	61.6
CubiCasa Rep. (TTA)	96.3	77.1	39.4	71.8	59.9	50.4
CubiCasa Rep. (TTA) + IP	94.1	65.2	11.5	42.1	36.5	10.5
Ours	96.9	70.7	56.5	77.9	66.7	62.1
Ours + IP	95.9	71.6	38.7	74.5	65.9	56.5
Ours (TTA)	97.2	82.5	55.3	80.0	72.1	63.4
Ours (TTA) + IP	96.0	82.1	37.8	74.2	69.4	55.3

Table 4.3: Performance comparison of models. CubiCasa's best model vs our best model trained on CubiCasa5k training data and unannotated LIFULL HOME's data, tested on the CubiCasa5k test set

LIFULL HOME's dataset	Icon			Room		
Model and method	Acc	Recall	Precision	Acc	Recall	Precision
CubiCasa	93.9	40.5	**34.7**	54.8	52.8	38.1
CubiCasa + IP	94.0	51.3	30.7	52.6	59.7	34.9
CubiCasa (TTA)	94.1	42.1	32.6	55.9	56.5	39.3
CubiCasa (TTA) + IP	94.0	52.2	28.2	54.0	63.7	35.7
CubiCasa Best Repr. (TTA)	93.7	40.5	22.5	**60.8**	55.8	**40.5**
CubiCasa Best Repr. (TTA) + IP	93.5	23.4	11.1	40.2	39.8	16.4
Ours	94.3	46.3	30.7	53.3	52.4	37.3
Ours + IP	94.0	57.6	21.5	48.4	60.3	31.5
Ours (TTA)	**94.5**	49.3	28.8	55.0	64.3	36.5
Ours (TTA) + IP	94.0	**68.6**	20.9	48.4	**68.5**	29.1

Table 4.4: Performance comparison of models. CubiCasa's best model vs our best model trained on CubiCasa5k training data and unannotated LIFULL HOME's data, tested on our annotated LIFULL HOME's test set

Type	Acc	FreqW Acc	Mean IoU	Recall	Precision
Icon	94.5	90.1	22.0	49.3	28.8
Icon_p	94.0	88.7	17.9	68.6	20.9
Room	55.0	42.2	25.3	64.3	36.5
Room_p	48.4	35.4	19.2	68.5	29.1

Table 4.5: Performance of our best model (with $\beta = 0.97$ and TTA) on the LIFULL HOME's test set. Subscript p stands for the post-processed (polygonized) predictions.

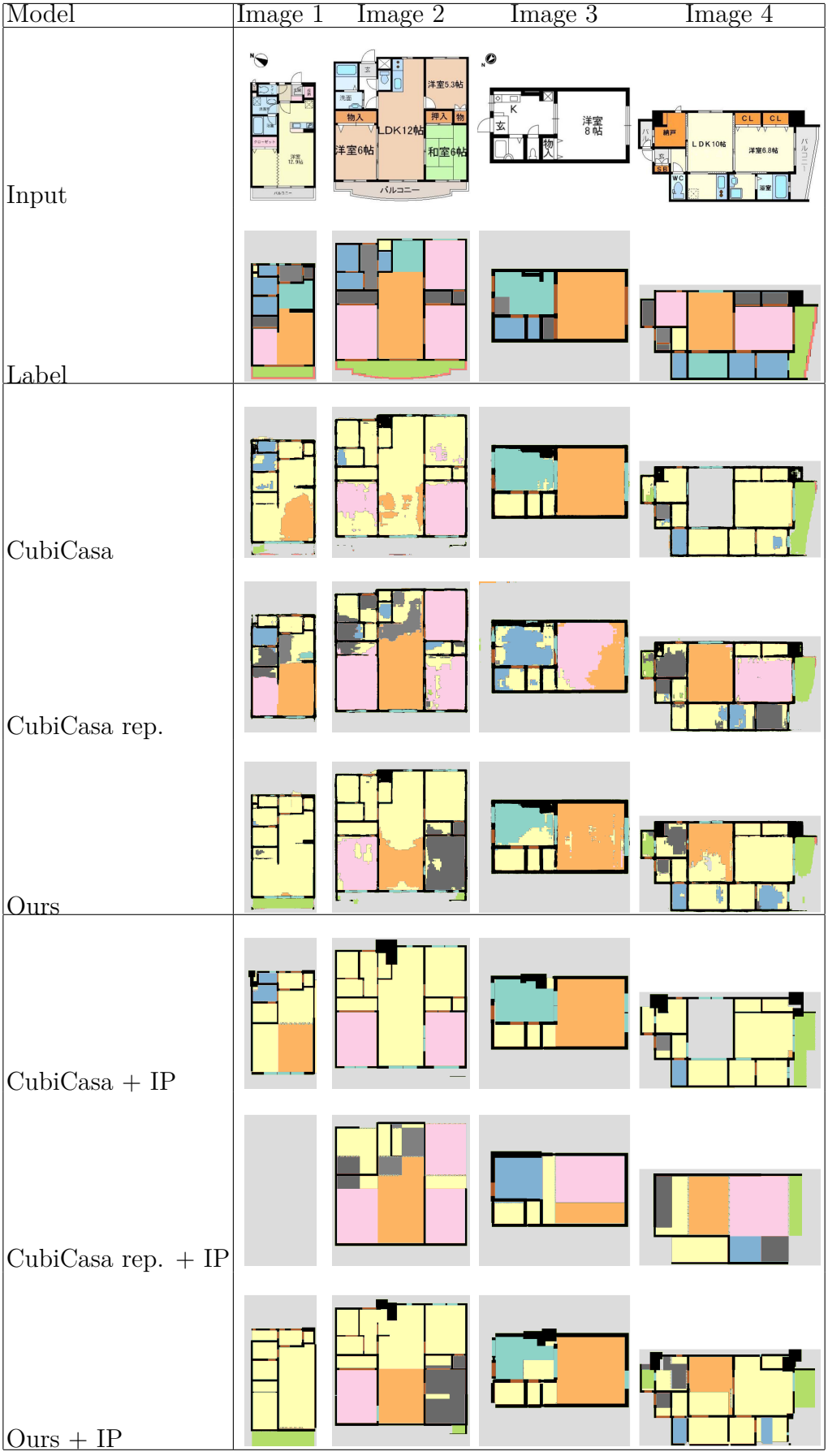


Figure 4.7: Comparison of results from different models. Evaluated on 4 images from the LIFULL HOME's dataset.

5
Discussion

In this section we discuss the results presented in the previous chapter and the method used in this project. We will do an in depth analysis of the similarities and differences between the results of our study and what we judge to be the most relevant related works. We will also discuss our method and try to to explicate the reason behind the behaviour and performance that we are seeing. The last section in this chapter is devoted to our suggestion for what future work could be done on the topic.

5.1 Discussion of results

The following is a discussion on both the results of the pseudolabelling techniques followed by a discussion of the results of the model using the full PixMax semi-supervised approach.

5.1.1 Pseudolabelling techniques

5.1.1.1 Statistical approach

We conjectured the model confidence to be a good proxy for the pixel-level model prediction quality, but our preliminary experiments showed it did not own this property. Alvarson et. al. establishes that there can be several underlying factors that causes the behaviour that we are seeing [50]. If the exchangeability assumption is violated, i.e. that the data that the model is trained on is to different from the test data, the behaviour could be explained. However, since the experiment was run on a simple random sample of images from the test slice of the CubiCasa5k dataset [1], we do not find this to be a good explanation although plausible. Statistical fluctuations may also play some role in what we are seeing since the experiment was only run on a set of images much smaller than the 4200 images that the model had been trained on.

Although outside the scope of this project, multiple ways of calibrating for model confidence have been developed that might be capable of resolving this issue. One is a workflow called *inductive conformal prediction*[50]. In this setting, we split the training set into two parts and use the bigger portion to train the model in an ordinary, supervised fashion while the other part is used as a confidence calibration set as can be seen in figure 5.1. By using an calibration set independent from the main training set, it is possible to force the model to have it's confidence as a lower

bound on the accuracy of the prediction.

Other ways to calibrate confidence include Histogram binning [54], Isotonic regression[55] and Baysian binning into quantiles[56]. These methods are however all outside the scope of this project. They are also not relevant for the result since we chose not to proceed with this method for pseudolabeling. However, for future works in the field, it could be suitable to investigate such methods further to allow for this sort of pseudolabeling technique.

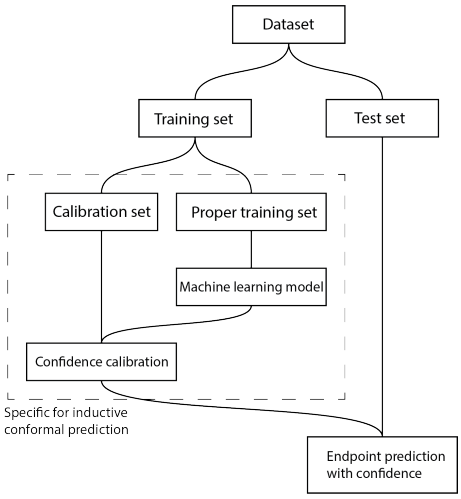


Figure 5.1: A conceptual model training training training scheme for inductive conformal prediction.

5.1.1.2 Post-processing technique

Likewise, the proposed post-processing technique that we investigated did not yield a reduced loss when applied on a prediction and the preliminary experiments with this method for pseudolabelling was therefore not continued. However, the fact that the post-processing algorithm proposed by Kalervo et. al. [1] that we investigated did not enhance the quality of the predictions in our case does not rule out the existence of such a mapping. From observing visualisations of the model's predictions e.g. in Figure 4.7, our conclusion is that it seems possible that there exists ways to enhance the prediction quality without including any extra information in addition to the models output layer.

5.1.1.3 PixMax pseudolabelling technique

The PixMax pseudolabelling technique for self-training shares many traits with combined consistency regularization/pseudolabelling schemes such as FixMatch[57]. The main difference between our proposed method and [57] is that they use the models output probability distribution to decide which predictions have a high confidence and hence should be kept for the continued model training. This is similar to what we have called *statistical approach* and discussed in 5.1.1.1. With PixMax on the other hand we use a batch of predictions and base the keeping criterion on

the variance within it. Even though this is an important difference, it could be argued that the outcome should be somewhat related since the same types of non-destructive augmentations are used.

We have formulated the keeping-criterion for PixMax as a threshold function for a batch-level metric in the data that we named β. Even if the β-value cannot unambiguously be said to give a measure for the prediction quality of an image, we conjecture it to be a reasonable assumption to make that it is at the very least highly correlated. However, there are also some problems with this threshold type method in it's proposed form as with all pseudolabelling techniques. From inspection of prediction on images with different β-vales we concluded that a relatively high β-threshold is preferable from a quality perspective. Since a high threshold results in a more conservative training scheme, this means that only a small part of the unannotated dataset will actually reach the semi-supervised model training stage. The fractions of images that remain ranges from around 25% down to under 1% of the dataset for the experiments that we ran. These can be seen in Table 4.1. This is a potential problem with the scheme even though it did not affect our results as we had an abundance of unannounced data for training. Another complication with a threshold-based technique is that another hyperparameter is introduced that might need to be optimised. Our approach to this was to run multiple, otherwise identical, model training schemes with different β-thresholds in the range we thought to be feasible for finding an optimal candidate as can be seen in Figure 4.6.

The proposed technique for pseudolabelling is heavily focused on picking images with a high invariance to global, geometrical transformations since the opposite (stochastic predictions) would mean that the network is not very confident. This is not necessarily a deficiency in the algorithm when it is applied to images of floor plans since all rotation and mirror symmetries of a floor plan image are equally suitable, but it is nevertheless a limiting factor in the grand scheme. An obvious extension on our work to other types of image domains would be to include other augmentation types that are adequate for the dataset in question e.g. colour jitter, shear, contrast and brightness amplification, translation and blurring.

5.1.2 PixMax model performance

Looking at tables 4.3 and 4.4, in general all models perform significantly better evaluated on the CubiCasa5k test set. This is somewhat expected since the models have all been trained initially using supervision on the training partition of this dataset. In 4.4, we find that while CubiCasa's model get a better precision (few false positives), our model gives a better recall (few false negatives). Depending on the use case this might be a preferred behaviour. An example would be a program that finds all doors and windows in drawings. Then it might be easier to remove the false positives at a later stage of the process rather than include the false negatives.

In 5.1.2 we found that we get a better prediction quality with a higher β-threshold. However it is not clear why we don't we get an improved performance using our

PixMax method.

In Table 4.2 we find that the best performing model varies a lot depending on what class and measure we are looking at. No model seems to significantly outperform any of the other.

Worth noting is that all evaluations on the LIFULL HOME's dataset have been made on a limited number: Only 50 samples. This is a potential cause for high statistical fluctuations that might obscure the actual performance of our approach.

5.2 Limiting factors

Due to finite time and resources, we have had to limit the the scope of some aspects of this project. We will here list and discuss some of the aspects that we judge to have had the largest impact on the evolution of the project and the final result.

5.2.1 Data sufficiency and utilisation

Model training schemes using semi-supervised learning through pseudolabelling in general uses a large amount of unannotated data compared to the amount of annotated data. In the FixMatch paper[57] they for example present an training scheme on the STL-10 [58] dataset where they use 5000 labelled images deluded with 100 000 unlabelled images resulting in a annotated-to-unannotated proportion of 1:20 [57]. However, it can also be seen in their ablation studies that it seems like the largest part of the performance boost comes when the ratio of annotated-to-unannotated data is around 1:4 which would suggest that there is only a small gain to be made from really big sets of unlabelled data in this setting.

In our case, we had access to just over 5 million unlabelled data points from the LIFULL HOME's dataset. Only 40 000 of these were used and from them only a even smaller part ever reached the semi-supervised portion of the model training due to the β-threshold keeping criterion. For our final runs the annotated-to-unannotated data ratio ranged from around 1:3 and down ans can be seen in Table 4.1. The main reason for the choice of datset size was limited storage on the GPU-cluster where the model training were performed. It is plausible that being able to utilise a bigger portion of the unlabelled data would have affected the result in a positive direction based on the results that FixMatch and others [47], [57].
As previously discussed, it is unclear whether or not the assumption can be made that the annotated and unannotated dataset contains images from the same distribution. We have therefore focused on trying to improve the models performance mainly on the unannotated dataset since this constitutes the novel part of the model training scheme that we bring to the table. The primary issue with this is that we did not have enough time resources to manually annotate a big enough portion of the annotated dataset to be able to use this for validation in a reliable way. We therefore chose to fall back on using only data from the validation slice of the annotated dataset for this task. We conjecture this to have a non-negligible negative

effect on the test performance of the final model since the optimisation is partly distorted as a consequence that the validation- and testing set have at best partly overlapping distributions.

5.2.2 Post-processing algorithm

The post-processing algorithm proposed by Kalervo et. al. [1] was as shown in Section 3.2.2 not suitable for creating pseudolabels since it on average yielded a higher loss. However it can nevertheless be a useful tool when applied on the raw model output for creating more visually appealing predictions as it captures the structure of some of the high-level features better. To present the results in a clear manner, we have also chosen to include the performance metrics for both the post-processed and the raw model prediction in the tables of Section 4.2. The post-processing algorithm has in addition to whats discussed in Section 5.1.1.2 also a few flaws from a performance viewpoint that has been somewhat limiting to the project. The algorithm in question uses a threshold function to map a per-pixel heatmap to a set of discrete interest points. In addition to introducing an extra hyperparameter in the from of this threshold, it also means that the number of resulting points of interest can vary greatly between images. This sometimes causes a performance issue since the algorithm seems to have an exponential time complexity w.r.t the number of remaining interest points. This has resulted in a very high variance for the processing time of different images with time scales ranging from a few seconds to more than an hour on the hardware presented in Section 3.4.1. Future work would therefore probably benefit heavily from finding a more efficient way of processing the model prediction.

5.3 Future works

During the course of the project, multiple topics were encountered that could have been investigated further. However, due to the limited time and scope of the project there was not always a possibility to do so. Here we will briefly present and discuss a selection of these themes with the motivation that they can be more thoroughly in the future.

The method that we developed can be interpreted as an hybrid between self-training and consistency regularisation with the main source of inspiration being FixMatch [57] when it comes to the model training scheme. An natural extension to whats proposed is therefore to instead adopt an pure consistency regularising training scheme to the same data. On the topic of global alterations to the proposed method, it could also be interesting to combine the method with another, more niche type of machine learning algorithm. A good candidate for this could e.g. be Object Character Recognition (OCR) to increase class certainty for rooms where the type is explicitly written in the proper location in the image.

When it comes to possible tweaks and possible improvements and extensions to the proposed technique, we would like to see a method for converting floor plan images

between drawing-scale and actual scale. Since there are no convention when it comes to sizes or scales of images, to do this would require utilising semantic understanding of the image and use this information to estimate an pixel-to-reality scale for the image. This could both be done as a pre-processing algorithm or be done by a different network model, alternatively be learnt implicitly by the main network model and used in a post-processing algorithm.

For all our runs had equal pixel-weighting for all classes in the loss function. This certainly a reasonable starting point since the type 1 and type 2 errors are essentially the same in our setting as all pixels gets assigned a class in the model's prediction. It is however not obvious that the model could not benefit from discriminate between classes in the loss weighting, depending on what performance statistic is evaluated. As can be seen in Figures B.1-B.4, Appendix B, the true class distribution is highly shifted towards certain classes compared to others, so it could be a reasonable strategy to compensate for this fact by introducing a asymmetric class weighting to e.g. get a higher frequency adjusted average accuracy. As with the other topics in this section we leave this as an open question question.

5.4 Contributions and implications

- New method for pseudolabeling developed: PixMatch
- General: More R&D in the semi-supervised problem domain will in the long term result in that more data can be used. Since computers can process more and more data, methods that can utilise big datasets get becomes more attractive since they are labour-efficient.
- Floor plan and architecture: The model is not radically better than the previously proposed method by [1], but the work can lead to insights in how this model was developed and inspire more people to continue the work.

6
Conclusion

The rapid improvement of computer vision techniques have opened up new possibilities in many areas in general and for parsing floor plans in particular. Labels for image segmentation are time-consuming to produce and the need for algorithms utilising unlabelled data is therefore high.
We have found that it is possible to adapt a network model to a new dataset and achieve good results without using any new labelled data for training. Our proposed method for producing pseudolabels, PixMax, was shown to give significant results considering the challenge of extracting information from previously unseen drawing styles. However, we found that the method had troubles classifying some information not present in the labelled training data such as regionally specific icons.
With PixMax our semi-supervised model achieve results comparable — and in some measures slightly better than — current state of the art supervised models.

Bibliography

[1] A. Kalervo, J. Ylioinas, M. Häikiö, A. Karhu, and J. Kannala, *Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis*, 2019. arXiv: 1904.01920 [cs.CV].

[2] C. Liu, J. Wu, P. Kohli, and Y. Furukawa, "Raster-to-vector: Revisiting floorplan transformation", pp. 2214–2222, Oct. 2017. DOI: 10.1109/ICCV.2017.241.

[3] S. Dodge, J. Xu, and B. Stenger, "Parsing floor plan images", pp. 358–361, May 2017. DOI: 10.23919/MVA.2017.7986875.

[4] L. de las Heras, J. Mas, G. Sánchez, and E. Valveny, "Wall patch-based segmentation in architectural floorplans", in *2011 International Conference on Document Analysis and Recognition*, 2011, pp. 1270–1274.

[5] J. Yang, H. Jang, J. Kim, and J. Kim, "Semantic segmentation in architectural floor plans for detecting walls and doors", *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 1–9, 2018.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, pp. 436–44, May 2015. DOI: 10.1038/nature14539.

[7] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[8] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, *A review on deep learning techniques applied to semantic segmentation*, 2017. arXiv: 1704.06857 [cs.CV].

[9] L.-P. Heras, S. Ahmed, M. Liwicki, E. Valveny, and G. Sánchez, "Statistical segmentation and structural recognition for floor plan interpretation: Notation invariant structural element recognition", *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 17, Sep. 2013. DOI: 10.1007/s10032-013-0215-2.

[10] L. de las Heras, O. Terrades, S. Robles, and G. Sanchez, "Cvc-fp and sgt: A new database for structural floor plan analysis and its groundtruthing tool", *International Journal on Document Analysis and Recognition*, 2015.

[11] *Lifull home's high resolution floor plan image data*, image(jpeg).

[12] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[13] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[14] H. Robbins and S. Monro, "A stochastic approximation method", *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, Sep. 1951. DOI: 10.1214/aoms/1177729586. [Online]. Available: https://doi.org/10.1214/aoms/1177729586.

[15] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization", *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, Jul. 2011.

[16] Y. Dauphin, H. Vries, J. Chung, and Y. Bengio, "Rmsprop and equilibrated adaptive learning rates for non-convex optimization", *arXiv*, vol. 35, Feb. 2015.

[17] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[19] W. Ouyang, X. Wang, X. Zeng, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, C.-C. Loy, and X. Tang, "Deepid-net: Deformable deep convolutional neural networks for object detection", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[20] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen", *Diploma, Technische Universität München*, vol. 91, no. 1, 1991.

[21] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies", in *A Field Guide to Dynamical Recurrent Neural Networks*, S. C. Kremer and J. F. Kolen, Eds., IEEE Press, 2001.

[22] R. Rojas, *Neural networks : a systematic introduction*. Berlin New York: Springer-Verlag, 1996, ISBN: 3-540-60505-3.

[23] B. Mehlig, *Artificial neural networks*, 2019. arXiv: 1901.05639 [cs.LG].

[24] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[25] M. D. Zeiler and R. Fergus, *Visualizing and understanding convolutional networks*, 2013. arXiv: 1311.2901 [cs.CV].

[26] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. 2020, https://d2l.ai.

[27] U. von Luxburg and B. Schoelkopf, *Statistical learning theory: Models, concepts, and results*, 2008. arXiv: 0810.4752 [stat.ML].

[28] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013. [Online]. Available: https://faculty.marshall.usc.edu/gareth-james/ISL/.

[29] A. Derumigny and J. Schmidt-Hieber, *On lower bounds for the bias-variance trade-off*, 2020. arXiv: 2006.00278 [math.ST].

[30] A. Pasini, "Artificial neural networks for small dataset analysis", *Journal of Thoracic Disease*, vol. 7, no. 5, 2015, ISSN: 2077-6624. [Online]. Available: http://jtd.amegroups.com/article/view/4418.

[31] J. Bagherzadeh and H. Asil, "A review of various semi-supervised learning models with a deep learning and memory approach", *Iran Journal of Computer Science*, Dec. 2018. DOI: 10.1007/s42044-018-00027-6.

[32] K. Nigam, A. K. Mccallum, S. Thrun, and T. Mitchell, "Text classification from labeled and unlabeled documents using em", *Machine Learning*, vol. 39, no. 2, pp. 103–134, May 2000, ISSN: 1573-0565. DOI: 10.1023/A:1007692713085. [Online]. Available: https://doi.org/10.1023/A:1007692713085.

[33] O. Chapelle, *Semi-supervised learning*. Cambridge, Mass: MIT Press, 2006, ISBN: 978-0-262-03358-9.

[34] W. Li and S. Chen, *Unsupervised domain adaptation with progressive adaptation of subspaces*, 2020. arXiv: 2009.00520 [cs.LG].

[35] H. Scudder, "Probability of error of some adaptive pattern-recognition machines", *IEEE Transactions on Information Theory*, vol. 11, no. 3, pp. 363–371, 1965. DOI: 10.1109/TIT.1965.1053799.

[36] S. Fralick, "Learning to recognize patterns without a teacher", *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 57–64, 1967. DOI: 10.1109/TIT.1967.1053952.

[37] I. Triguero, S. García, and F. Herrera, "Self-labeled techniques for semi-supervised learning: Taxonomy, software and empirical study", *Knowledge and Information Systems*, vol. 42, Feb. 2015. DOI: 10.1007/s10115-013-0706-y.

[38] D.-H. Lee, "Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks", *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, Jul. 2013.

[39] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, *Generalized intersection over union: A metric and a loss for bounding box regression*, 2019. arXiv: 1902.09630 [cs.CV].

[40] J. Baxter, "A model of inductive bias learning", *Journal of Artificial Intelligence Research*, vol. 12, pp. 149–198, Mar. 2000, ISSN: 1076-9757. DOI: 10.1613/jair.731. [Online]. Available: http://dx.doi.org/10.1613/jair.731.

[41] A. Kendall, Y. Gal, and R. Cipolla, *Multi-task learning using uncertainty to weigh losses for scene geometry and semantics*, 2018. arXiv: 1705.07115 [cs.CV].

[42] H. Baird, *Structured Document Image Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, ISBN: 978-3-642-77281-8.

[43] P. Simard, D. Steinkraus, and J. Platt, "Best practices for convolutional neural networks applied to visual document analysis", in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, vol. 2, 2003, pp. 958–958.

[44] E. Harris, A. Marcu, M. Painter, M. Niranjan, A. Prügel-Bennett, and J. Hare, *Fmix: Enhancing mixed sample data augmentation*, 2020. arXiv: 2002.12047 [cs.LG].

[45] O. Chapelle, J. Weston, L. Bottou, and V. Vapnik, "Vicinal risk minimization", in *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds., MIT Press, 2001, pp. 416–422. [On-

line]. Available: http://papers.nips.cc/paper/1876-vicinal-risk-minimization.pdf.

[46] A. Mustafa and R. K. Mantiuk, *Transformation consistency regularization- a semi-supervised paradigm for image-to-image translation*, 2020. arXiv: 2007.07867 [cs.CV].

[47] K. Sohn, Z. Zhang, C.-L. Li, H. Zhang, C.-Y. Lee, and T. Pfister, "A simple semi-supervised learning framework for object detection", in *arXiv:2005.04757*, 2020.

[48] G. Papandreou, L.-C. Chen, K. P. Murphy, and A. L. Yuille, "Weakly- and semi-supervised learning of a deep convolutional network for semantic image segmentation", in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[49] D. Sharma, N. Gupta, C. Chattopadhyay, and S. Mehta, "Daniel: A deep architecture for automatic analysis and retrieval of building floor plans", in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 01, 2017, pp. 420–425. DOI: 10.1109/ICDAR.2017.76.

[50] J. Alvarsson, S. Arvidsson McShane, U. Norinder, and O. Spjuth, "Predicting with confidence: Using conformal prediction in drug discovery", *Journal of Pharmaceutical Sciences*, vol. 110, no. 1, pp. 42–49, 2021, ISSN: 0022-3549. DOI: https://doi.org/10.1016/j.xphs.2020.09.055. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002235492030589X.

[51] U. Norinder, L. Carlsson, S. Boyer, and M. Eklund, "Introducing conformal prediction in predictive modeling. a transparent and flexible alternative to applicability domain determination", *Journal of Chemical Information and Modeling*, vol. 54, no. 6, pp. 1596–1603, 2014, PMID: 24797111. DOI: 10.1021/ci5001168. eprint: https://doi.org/10.1021/ci5001168. [Online]. Available: https://doi.org/10.1021/ci5001168.

[52] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, 2015. arXiv: 1411.4038 [cs.CV].

[53] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2014.

[54] B. Zadrozny and C. Elkan, "Learning and making decisions when costs and probabilities are both unknown", *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Jul. 2001. DOI: 10.1145/502512.502540.

[55] ——, "Transforming classifier scores into accurate multiclass probability estimates", in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02, Edmonton, Alberta, Canada: Association for Computing Machinery, 2002, pp. 694–699, ISBN: 158113567X. DOI: 10.1145/775047.775151. [Online]. Available: https://doi.org/10.1145/775047.775151.

[56] M. P. Naeini, G. F. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using bayesian binning", ser. AAAI'15, Austin, Texas: AAAI Press, 2015, pp. 2901–2907, ISBN: 0262511290.

[57] K. Sohn, D. Berthelot, C.-L. Li, Z. Zhang, N. Carlini, E. D. Cubuk, A. Kurakin, H. Zhang, and C. Raffel, *Fixmatch: Simplifying semi-supervised learning with consistency and confidence*, 2020. arXiv: 2001.07685 [cs.LG].

[58] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning", in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., ser. Proceedings of Machine Learning Research, vol. 15, Fort Lauderdale, FL, USA: JMLR Workshop and Conference Proceedings, Nov. 2011, pp. 215–223. [Online]. Available: http://proceedings.mlr.press/v15/coates11a.html.

A

Results of all tested model hyperparameters

The following is a table of all tested models. Explanation of the naming scheme: The number following b is the percentage for the β-threshold value, "ii" means ignoring class *undefined* for the loss function, "+IP" means Integer Programming method was used for polygonizing (vectorizing) the output, TTA meaning Test-Time-Augmentation.
The chosen hyperparameters for the final training was choosen as the LH_b97_ii-LH-TTA model.

Lifull Home's dataset	Icon			Room		
Model and method	Acc	Recall	Precision	Acc	Recall	Precision
LH_b95-LH	93.9	46.8	30.8	52.9	50.1	35.9
LH_b95-LH + IP	94.0	55.9	22.0	46.4	53.4	29.5
LH_b96-LH	93.6	41.0	30.0	51.3	45.7	35.0
LH_b96-LH + IP	93.9	54.8	21.7	45.9	49.2	28.3
LH_b97-LH	94.2	42.4	31.3	49.2	47.0	32.1
LH_b97-LH + IP	94.1	54.3	24.3	43.7	51.5	26.1
LH_b98-LH	93.9	42.4	30.2	49.3	46.0	31.5
LH_b98-LH + IP	94.0	55.6	21.5	46.0	47.8	27.9
LH_b99-LH	93.8	42.7	30.2	48.6	44.8	31.5
LH_b99-LH + IP	94.0	49.1	25.0	45.8	42.0	28.4
LH_b95_ii-LH	93.6	44.0	32.1	51.4	51.4	33.2
LH_b95_ii-LH + IP	94.0	47.9	22.0	44.7	55.0	25.1
LH_b96_ii-LH	94.1	47.1	**32.1**	52.0	51.4	35.1
LH_b96_ii-LH + IP	93.9	54.2	22.4	46.9	55.7	29.8
LH_b97_ii-LH	94.3	46.3	30.7	53.3	52.4	37.3
LH_b97_ii-LH + IP	94.0	57.6	21.5	48.4	60.3	31.5
LH_b98_ii-LH	94.1	46.8	31.9	51.5	46.6	33.3
LH_b98_ii-LH + IP	94.0	51.1	21.9	47.1	50.7	27.9
LH_b99_ii-LH	93.6	44.2	28.2	48.7	49.6	31.0
LH_b99_ii-LH + IP	94.0	56.1	23.9	45.9	52.3	27.4
LH_b95-LH-TTA	94.1	47.4	27.4	54.2	60.3	35.9
LH_b95-LH-TTA + IP	94.0	59.9	20.5	45.3	58.1	26.4
LH_b96-LH-TTA	94.1	47.1	27.8	53.0	52.2	35.3
LH_b96-LH-TTA + IP	94.0	62.7	20.3	44.6	54.1	26.9
LH_b97-LH-TTA	94.4	50.3	29.2	51.2	59.9	33.4
LH_b97-LH-TTA + IP	94.1	66.0	22.2	45.3	59.9	26.6
LH_b98-LH-TTA	94.2	51.0	28.5	50.9	54.5	31.8
LH_b98-LH-TTA + IP	94.1	64.3	22.2	45.3	57.4	26.1
LH_b99-LH-TTA	94.1	49.8	28.1	50.7	51.4	32.8
LH_b99-LH-TTA + IP	94.2	49.2	25.4	47.0	53.9	28.4
LH_b95_ii-LH-TTA	94.2	46.7	30.6	53.0	62.7	34.1
LH_b95_ii-LH-TTA + IP	94.1	57.5	21.6	44.8	63.2	26.4
LH_b96_ii-LH-TTA	94.3	49.5	29.4	52.3	58.2	33.9
LH_b96_ii-LH-TTA + IP	94.1	59.4	22.7	45.9	53.9	27.5
LH_b97_ii-LH-TTA	**94.5**	49.3	28.8	**55.0**	64.3	**36.5**
LH_b97_ii-LH-TTA + IP	94.0	**68.6**	20.9	48.4	**68.5**	29.1
LH_b98_ii-LH-TTA	94.5	51.5	30.3	53.5	50.2	34.1
LH_b98_ii-LH-TTA + IP	94.0	61.3	21.3	45.0	56.3	26.4
LH_b99_ii-LH-TTA	94.2	53.2	26.6	50.8	57.7	32.0
LH_b99_ii-LH-TTA + IP	94.0	60.7	21.2	45.0	61.2	25.9

Table A.1: Full table of all evaluated models.

B

Class distribution

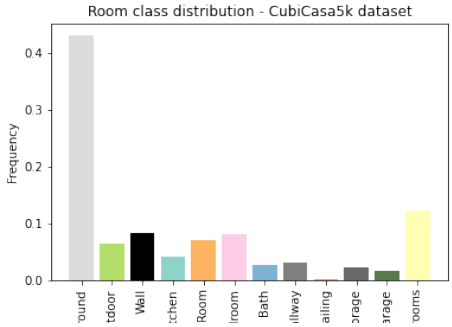


Figure B.1: The per-pixel distribution of the room classes in the CubiCasa5k dataset.

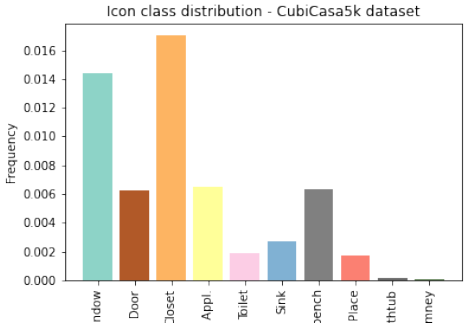


Figure B.2: The per-pixel distribution of the icon classes in the CubiCasa5k dataset.

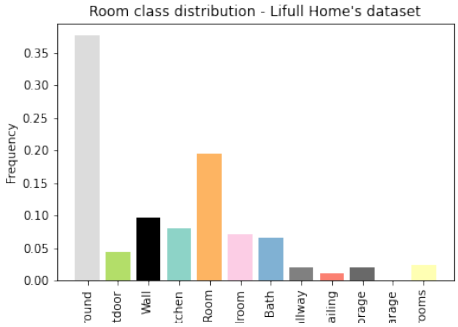


Figure B.3: The per-pixel distribution of the room classes in the LIFULL HOME's dataset.

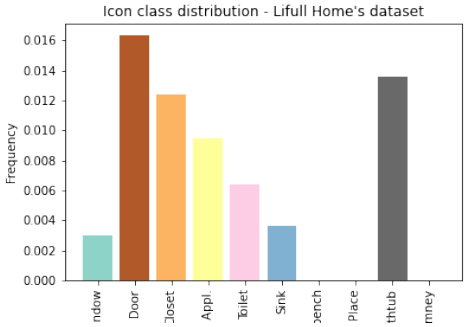


Figure B.4: The per-pixel distribution of the icon classes in the LIFULL HOME's dataset.

C

Visual comparison of models

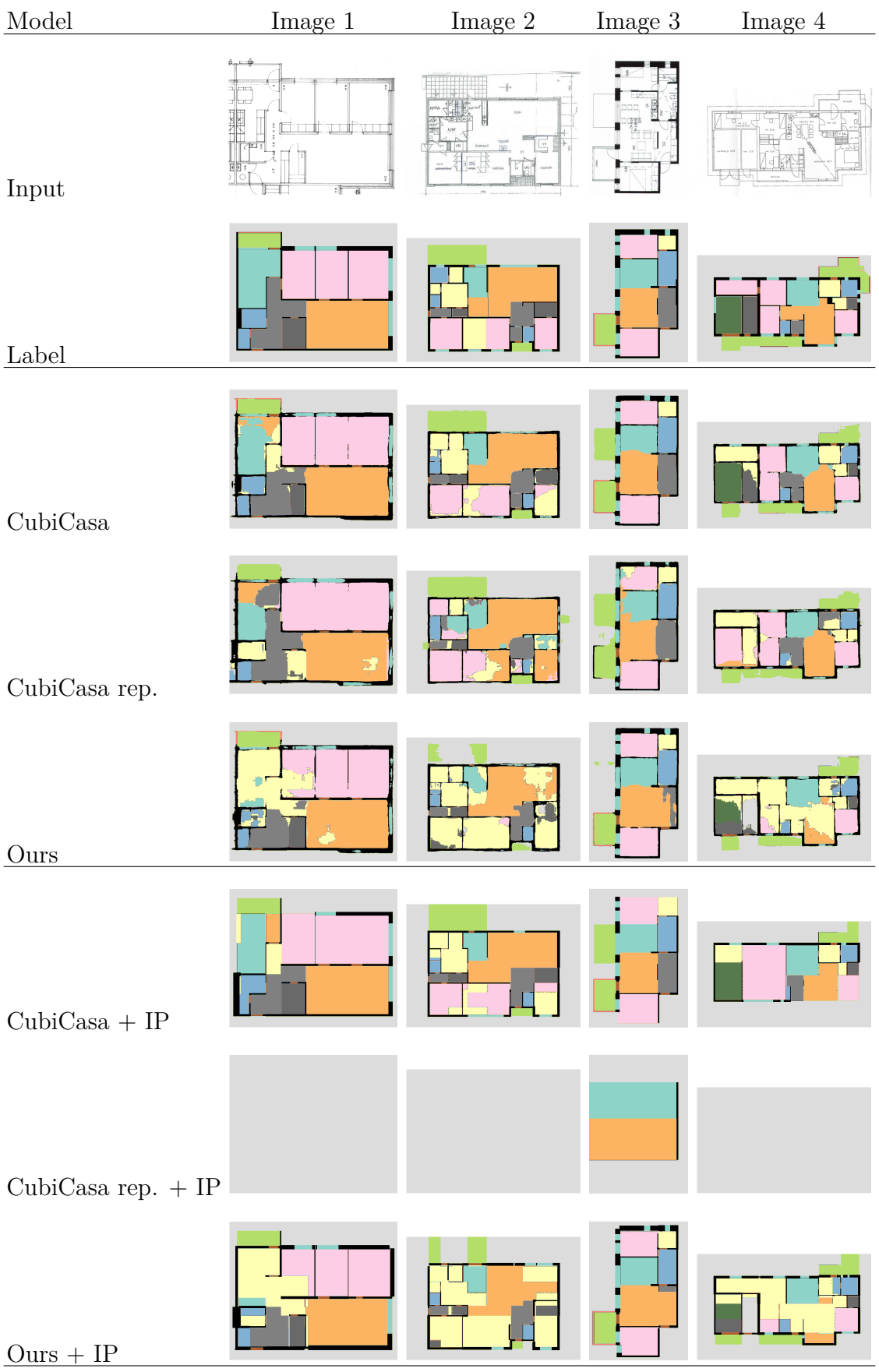


Figure C.1: Comparison of results from different models. Evaluated on 4 images from the CubiCasa5k dataset.