Exploring Domain Adaptation for Floor Plan Vectorization

September 2024

J.L. Hofland

Delft University of Technology

TU Delft

Exploring Domain Adaptation for Floor Plan Vectorization

by

J.L. Hofland

Student Name	Student Number
Jeroen Hofland	4678141

To obtain the degree of *MSc Computer Science* at the faculty of Electrical Engineering, Mathematics & Computer Science (EEMCS), part of Delft University of Technology (TU Delft), to be defended publicly on Thursday September 12, 2024 at 14:00 AM.

Project Duration:	Dec, 2023 - Sept, 2024	
Faculty:	EEMCS	TU Delft
Thesis Committee:	Dr. S. Khademi	TU Delft, Daily supervisor
	Dr. ir. J. van Gemert	TU Delft, Thesis Advisor
	Ir. C.C.J. van Engelenburg	TU Delft, Daily co-supervisor
	Dr. ir. L. Cavalcante Siebert	TU Delft, External member
	Ir. R. Jongerius	Accenture, Company supervisor

Cover: 3D floor plan (coloured) of an apartment by Tallbox under CC BY-SA 4.0 (Modified)

TU Delft accenture

Preface

This thesis, *Exploring Domain Adaptation for Floor plan Vectorization* was conducted between December 2023 and September 2024 at the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) at TU Delft. This work was conducted as part of the Computer Vision Lab of the EEMCS faculty, and the AiDAPT AI lab at the architecture faculty.

I want to express my gratitude to my daily co-supervisor, Ir. Casper van Engelenburg's consistent support and feedback were key in shaping this work. I also appreciate the guidance provided by Dr. Seyran Khademi, whose domain insights and kind mentorship were invaluable throughout our meetings. My thanks also go to Dr. ir. Jan van Gemert for his sharp feedback during our evaluation meetings, which helped refine the direction of this research. I am also grateful to Ir. Ricardo Jongerius from Accenture for his strategic advice and for being a sparring partner in resolving challenges. I also want to thank Accenture the Netherlands for making it possible to do this as a thesis internship.

At last, I would like to acknowledge my thesis graduation committee, consisting of Dr. S. Khademi, Dr. ir. J. van Gemert, Ir. C.C.J. van Engelenburg, Dr. ir. L. Cavalcante Siebert, and Ir. R. Jongerius.

J.L. Hofland
Delft, September 2024

Contents

1

Introduction

The research is presented as a scientific article in Chapter 2. This is followed by a background section in Chapter 3 that explains the technical concepts and terms used. We begin that background with an overview of machine learning, then discuss domain adaptation, and conclude with floor plan vectorization.

2

Scientific Article

Exploring Domain Adaptation for Floor Plan Vectorization

Jeroen Hofland
Delft University of Technology
AiDAPT Lab

Abstract

This paper explores the challenges of converting architectural floor plans from raster to vector images. Unlike previous studies, our research focuses on domain adaptation to address stylistic and technical variations across different floor plan datasets. We develop and test our vectorization method on the CubiCasa5K benchmark, which includes 3 different floor plan styles. Our analysis reveals differences in input features across the CubiCasa5K styles, indicating the potential for domain adaptation research, mostly in room segmentation. However, we also find multiple indications that labelling in the CubiCasa5K dataset is ambiguous and inconsistent. Furthermore, styles with more training data do not always perform better, highlighting the complexity differences between floor plan styles. Our baseline shows a 0.7% gap for rooms yet a 0.6% improvement for objects, likely caused by the smaller feature gaps and inconsistent labelling. To address the adaptation gap, we add a Multi-Kernel Maximum Mean Discrepancy (MK-MMD) loss to the CubiCasa5K model to minimize feature distribution differences between domains. While our MK-MMD implementation shows potential for reducing the adaptation gap, persistence issues and mixed results across classes make it difficult to draw clear conclusions. Our findings also show the role of balancing spatial context in the MK-MMD calculation. These insights lay a foundation for future domain adaptation research in floor plan vectorization.

1. Introduction

Architectural floor plans are crucial for defining the layout, distribution, and function of indoor spaces, which is essential for designing, analyzing, and remodelling buildings [24]. These plans are typically created using software that generates vector-based graphics, allowing for precise and geometry-based drawings of shapes, connections, and room functions [12, 42]. However, for easier viewing or storage, these vector graphics are often converted into raster images, causing a loss of metadata like layers and object information [29, 42, 45]. The loss of metadata is important because it makes it harder to create accurate indoor databases, which are needed for services like buildings management [39], indoor mapping [32, 38], and data modelling [19, 37], making tasks such as room segmentation, object detection, and 2D/3D reconstruction more difficult [21]. In older projects completed before the widespread use of Computer-Aided Design (CAD) tools, floor plans only exist as hand-drawn

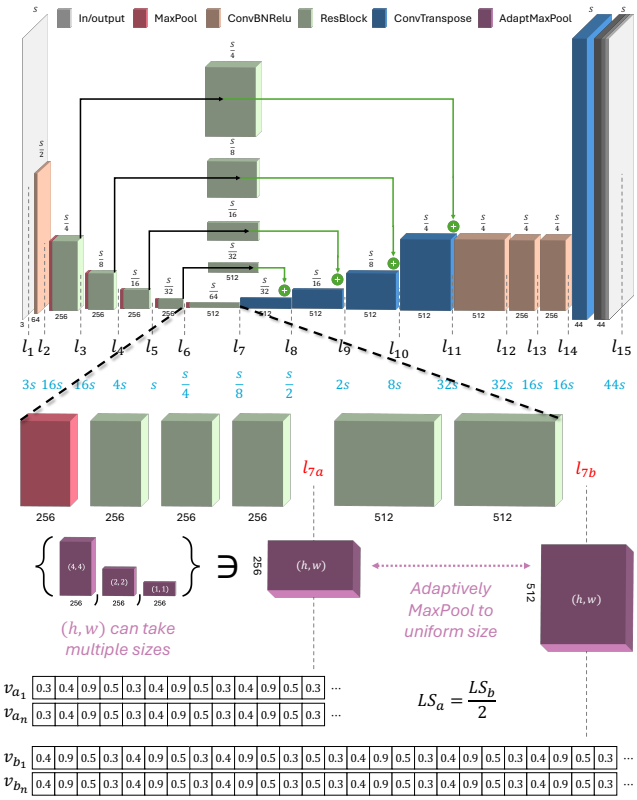


Figure 1. The model architecture and adaptation pipeline. We extract the latent vector for domain adaptation after contraction between layers 6 and 7. Each layer has a corresponding size (in blue) relating to the original input size. We can split layer 7 in a with 256 and b with 512 channels. These are forwarded through an adaptive max pool to obtain the latent vector.

images that were later scanned into digital formats [29]. Furthermore, floor plans typically include complex, interconnected elements like walls, windows, furniture, and text, making it difficult to automatically analyze and retrieve information from them [24].

One approach to mitigating the challenges of working with rasterized floor plans is vectorization. Vector graphics represent images using mathematical equations that define basic geometric shapes, such as points, lines, curves, and polygons. Unlike raster graphics, which are composed of pixels, vector graphics are resolution-independent, meaning they can be scaled without losing quality. Vectorization is the process of converting pixel-based images into these scalable vector formats by identifying and defining the geometric shapes and lines within the image [17, 35]. The vectorization process is key to restoring the precision, details, and scalability of floor plans, making them valuable in digital applications.

Automatically interpreting these rasterized floor plans presents several challenges. First, there is no universal standard among architectural companies, leading to differences in symbols, colors, and line styles [21, 28]. Second, the drawings themselves can be complex and unclear, which makes interpretation difficult [21]. Additionally, floor plans must follow certain geometric and topological rules, such as ensuring doors are placed within walls and walls define the boundaries of rooms. These rules add further complexity to the analysis, especially since the layout of rooms can vary widely between different buildings [24].

To address the challenge of interpreting diverse rasterized floor plans, our research introduces unsupervised domain adaptation for floor plan vectorization. A data domain refers to a specific type or style of data, like different styles of floor plans, each with its unique features, symbols, or layout patterns [9]. Unlike traditional approaches that rely on annotated datasets [27, 43], unsupervised domain adaptation involves developing models that can adapt to new, unannotated data domains. This approach is vital as a large number of variations makes it challenging for models to perform effectively in new contexts [44]. Our contributions are summarized as:

1. **Dataset analysis:** We analyze the CubiCasa5K dataset, focusing on the visual diversity and complexity across its three styles. While the label distribution is consistent, we find ambiguous labelling practices.

2. **Adaptation gap:** We identify an adaptation gap in input feature distribution and Intersection over Union (IoU), indicating that CubiCasa5K is suitable for domain adaptation research. However, the dataset and model show unusual behaviour in certain classes, likely due to labelling biases.

3. **Domain adaptation implementation:** We integrate Multi-Kernel Maximum Mean Discrepancy (MK-MMD) with the CubiCasa5K model, making this the first study to apply MMD to floor plan vectorization. Our results show potential for MK-MMD, especially with a balanced spatial context, but inconsistent IoU limits definitive conclusions.

2. Related Works

2.1. Floor plan vectorization

Before deep learning, researchers were already working on automating the segmentation of floor plans using traditional image processing techniques. A semi-automated method for segmenting rooms in building floor plans was developed using a proximity metric to identify bounded regions in scanned images [34]. Other methods focused on detecting shapes like lines, arcs, and small loops in layouts to identify rooms, walls, and doors [1, 8].

As deep learning gained traction, the focus shifted to using neural networks for automatic floor plan recognition. A neural network based on a Human Pose estimation architecture [2] was introduced to detect junction points and per-pixel classification maps in floor plan images, with modifications made to ResNet-152 [16, 24]. The model employs an encoder-decoder architecture similar to U-Net [31], ideal for tasks requiring precise localization and contextual understanding such as floor plan vectorization. These predicted junctions were then connected using a mathematical technique called integer programming to identify walls and vectorize the pixel classifications. However, this method was limited to floor plans with rectangular rooms and uniform wall thickness.

Building on this, a Fully Convolutional Network (FCN) was employed to segment wall pixels, while the Faster R-CNN framework was used to detect objects such as doors, kitchens, and bathtubs [7]. They also leveraged the Google Vision API for detecting and recognizing text in floor plans. Another approach used FCNs for segmenting floor plan images without accounting for the spatial relationships between different classes, treating each pixel independently [40]. CubiCasa5K [18] advanced the field by using a model similar to that of Raster-to-Vector [24], with the addition of a multi-task uncertainty loss [20]. The approach of CubiCasa5K still achieves state-of-the-art performance in floor plan image vectorization, making it the choice for our research.

2.2. Domain adaptation

Traditional machine learning assumes that training and test data share the same distribution, but this is often violated in real-world scenarios, leading to domain shift. A data domain is a specific distribution of data with unique fea-

tures and patterns. Collecting similar data is challenging, and publicly available datasets may not match the test data, causing performance issues [9]. Domain adaptation addresses the train-test gap by training a model on a source domain and adapting it to perform well on a different but related target domain, minimizing the differences between their distributions. Visual domain adaptation has evolved significantly over the years, with early efforts focusing on statistical alignment methods and more recent advancements integrating deep learning techniques. In our research, we focus on unsupervised domain adaptation, where the input feature distribution $P(X_s)$ and labelling distribution $P(Y_s)$ are known from the source domain, while only the samples $P(X_t)$ from the target domain are available [10].

Central to the research efforts in the unsupervised domain adaptation setting has been the use of Maximum Mean Discrepancy (MMD) for aligning distributions between the source and target domains, a nonparametric measure to compare distributions [14]. It compares the means of two distributions in a reproducing kernel Hilbert space (RKHS) where the mean embeddings of different domain distributions can be explicitly matched [15]. Their work demonstrated how MMD in combination with an appropriate kernel could be used to measure the distance between two distributions in RKHS, providing a powerful tool for statistical analysis.

Building on the MMD definition, Joint Distribution Adaptation (JDA) was introduced to align both marginal and conditional distributions [25]. JDA uses class-wise MMD to align both overall feature distributions and individual class distributions, addressing a key challenge in domain adaptation for semantic segmentation.

The integration of MMD into neural network frameworks marked a significant leap in domain adaptation techniques. MMD was embedded within deep learning models, demonstrating that these frameworks could enhance feature transferability by learning more robust and discriminative features across domains [11, 36]. Subsequent works expanded on these approaches, applying MMD to various tasks in deep learning [3, 4, 41]. The MMD was extended by introducing a multi-kernel variant (MK-MMD) to address kernel sensitivity issues, capturing complex data distributions more effectively and improving domain adaptation performance in deep networks [26]. Later research showed that aligning feature distributions in the latent space improves domain adaptation by making the learned representations more robust [3].

Motivated by the success of these methods, our approach leverages MK-MMD within the latent space of CubiCasa5K for unsupervised domain adaptation in floor plan vectorization tasks. By researching the CubiCasa5K dataset, its domain-specific properties, and integrating MK-MMD loss, we aim to explore if and how we can research and ultimately improve adaptation to unseen data distributions.

3. Preliminaries

In our preliminaries section, we explain the key concepts and methods that we use in our research. These ideas come from well-known studies in the field, as referenced, but are adjusted to make the concepts clearer and more relevant to our specific setting. We start by describing the problem setting and the CubiCasa5K model [18], followed by an explanation of the loss functions used. Finally, we introduce the concept of Maximum Mean Discrepancy (MMD) [14] and its extension, Multi-Kernel MMD [26].

3.1. Problem Setting

Our research explores unsupervised domain adaptation for floor plan vectorization, aiming to convert a rasterized image x of size $H \times W \times C$, where H is height, W is width, and C represents color channels, into a structured SVG file y. The SVG file captures the geometric and semantic details of the floor plan, such as room layouts, icons, and junctions [17].

The goal is to train a model using labelled data from a source domain (X_S, Y_S) and ensure it generalizes to a target domain X_T, where labels are not available. The objective is to minimize the prediction error on the source domain while also making the model's features transferable to the target domain [10]. For our task, we minimize a vectorization loss $\mathcal{L}_V$ on the source domain:

$$\min_{\mathbf{W}} \mathcal{L}_V(f_{\mathbf{W}}(X_S), Y_S) \tag{1}$$

where $\mathbf{W}$ represents the model parameters, $f_{\mathbf{W}}$ is the model function parameterized by $\mathbf{W}$, X_S is the source domain input data, and Y_S is the corresponding labels. Simultaneously, the discrepancy $\mathcal{D}$ between the source and target domains is reduced:

$$\min_{\mathbf{W}} \mathcal{D}(f_{\mathbf{W}}(X_S), f_{\mathbf{W}}(X_T)) \tag{2}$$

where X_T is the target domain input data. The overall objective is then expressed as:

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} [\mathcal{L}_V + \lambda \mathcal{D}] \tag{3}$$

where $\mathbf{W}^*$ is the optimized set of model parameters, and λ is a weighting factor that balances the importance of source accuracy $\mathcal{L}_V$ and domain alignment $\mathcal{D}$, ensuring that the model performs well on both the source and target domains.

3.2. The CubiCasa5K model

The model introduced by Kalervo et al. [18] and visualized in Figure 1, takes an input image x with dimensions $H \times W \times 3$ (RGB) and outputs a tensor y with dimensions

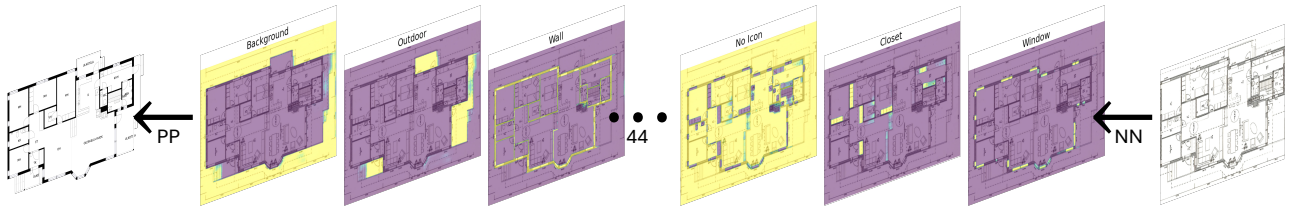


Figure 2. The process of CubiCasa5K takes as input an image and uses a neural network (NN), displayed in Figure 1, to get 44 probability maps of 12 room, 11 icon, and 21 junctions types displayed in Figure 3b. After prediction, post-processing (PP) is applied to get the class with the maximal probability after which the pixel-wise predictions are vectorized using the junctions and integer programming. The probability maps are scaled between 0 (purple) and 1 (yellow).

$H \times W \times (P_r + P_i + P_j)$. Here, P_r represents the number of channels for room probability maps, P_i represents the number of channels for icon probability maps, and P_j represents the number of channels for junction probability maps. These different probability maps are visualized in Figure 2. The transformation from the input x to the output y is carried out by a function $f_{\mathbf{W}}(x)$, which is parameterized by approximately 17 million weights $\mathbf{W}$.

Once the probability maps are predicted, post-processing using integer programming vectorises the pixel-wise predictions. The integer programming uses junction heatmaps and segmentation masks to generate vector representations, ensuring the final output adheres to the geometric constraints of architectural drawings. The overall process can be seen in Figure 2.

The problem involves multiple tasks, such as predicting room layouts, icons, and junctions, each requiring different loss functions. For room segmentation and icon detection, the cross-entropy loss is used which is defined as:

$$\mathcal{L}_{\text{CE}} = -y \cdot \log(\text{softmax}(f_W(x))) \tag{4}$$

where y is the ground truth label, and $f_W(x)$ is the predicted probability distribution after applying the softmax function. For junction detection, mean squared error (MSE) loss is used, which measures the squared difference between the predicted and actual values:

$$\mathcal{L}_{\text{MSE}} = \|y - f_W(x)\|^2 \tag{5}$$

where y is the ground truth value, and $f_W(x)$ is the prediction. To handle these tasks together, they use a multi-task loss approach. For segmentation tasks, the loss function is:

$$L_S = -\sum_{k \in \{\text{rooms, icons}\}} \frac{1}{\sigma_k} y_k \cdot \log(\text{softmax}(f_{W_k}(x))) \tag{6}$$

where y_k is the ground truth segmentation map, $f_{W_k}(x)$ is the predicted map for the k-th task, and σ_k is a learned task-specific weighting factor. No regularizer is needed here, as the weighting remains positive throughout the training. For tasks that involve heatmap predictions, the junctions, the loss function is:

$$L_H = \sum_i \left[\frac{1}{2\sigma_i^2} \|y_i - f_{W_i}(x)\|^2 + \log(1 + \sigma_i) \right] \tag{7}$$

where y_i is the ground truth, $f_{W_i}(x)$ is the prediction, and σ_i is a learned weighting factor. The term $\log(1 + \sigma_i)$, in contrast to the segmentation losses, is added as a regularizer. The total loss for the model combines both the segmentation and heatmap losses:

$$\mathcal{L} = L_S + L_H \tag{8}$$

3.3. Maximum Mean Discrepancy

The Maximum Mean Discrepancy (MMD) is a nonparametric metric used to compare two distributions by measuring the distance between their means in a Reproducing Kernel Hilbert Space (RKHS) [14]. Given two datasets, $A = \{x^1, x^2, \ldots, x^{n_A}\}$ from one data distribution and $B = \{x^1, x^2, \ldots, x^{n_B}\}$ from another data distribution, the MMD is defined as:

$$\text{MMD}^2(X_A, X_B) = \left\| \frac{1}{n_A} \sum_{i=1}^{n_A} \phi(x_A^i) - \frac{1}{n_B} \sum_{j=1}^{n_B} \phi(x_B^j) \right\|_{\mathcal{H}}^2 \tag{9}$$

where $\phi(\cdot)$ is a feature mapping function to the RKHS $\mathcal{H}$. RKHS is a space where a kernel function defines an inner product, enabling the computation of similarities in a high-dimensional feature space and capturing complex relationships between distributions. Expanding on equation 9, we can define the MMD for domain A and domain B distributions as:

$$\text{MMD}^2(X_A, X_B) = \mathcal{S}_{aa} + \mathcal{S}_{bb} - 2\mathcal{S}_{ab} \tag{10}$$

Here, $\mathcal{S}_{aa}$ represents the similarity within domain A data, $\mathcal{S}_{bb}$ represents the similarity within domain B data, and $\mathcal{S}_{ab}$ represents the similarity between the two distributions. By considering the variance within these distributions, the

method ensures that the model adjusts only for the differences between them. The similarity metric $\mathcal{S}_{ab}$ is further defined as:

$$\mathcal{S}_{ab} = \frac{1}{n_a n_b} \sum_{i=1}^{n_a} \sum_{j=1}^{n_b} k(x_a^i, x_b^j) \tag{11}$$

where n_a and n_b are the number of samples in sets a and b and x_a^i and x_b^j represent individual samples from these sets. The function $k(x_a^i, x_b^j)$ is a kernel function that computes the similarity between sample x_a^i from set a and sample x_b^j from set b. The most frequent kernel used for comparison is the Radial Basis Function (RBF) but other kernels are also used [5]. To handle the sensitivity of using a single kernel, equation 9 is extended to Multi-Kernel MMD (MK-MMD) [26] which is defined as:

$$\text{MK-MMD}^2(X_A, X_B) = \frac{1}{n_k} \sum_{j=1}^{n_k} \text{MMD}_k^2(X_A, X_B) \tag{12}$$

where n_k is the number of kernels used, and $\text{MMD}_k^2(X_A, X_B)$ is the MMD computed with the k-th kernel. The averaging across multiple kernels allows us to capture discrepancies between distributions at various scales, providing a robust measure of domain discrepancy.

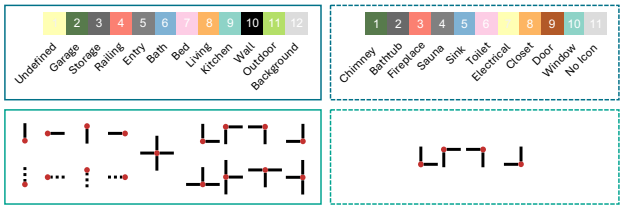
4. Data analysis

Floor plan datasets are essential for advancing the field of automated floor plan vectorization. Several datasets have been introduced, each contributing to the field. For example, the R3D dataset [23] comprises 215 floor plans, featuring 1,312 rooms, 6,628 walls, 1,923 doors, and 1,268 windows. Another significant contribution is the R2V dataset [24], which offers 870 annotated floor plan images with 11 room types and 8 icons. Among these, the CubiCasa5K dataset [18] stands out due to its scale and comprehensiveness, containing 5,000 samples that cover over 80 object categories.

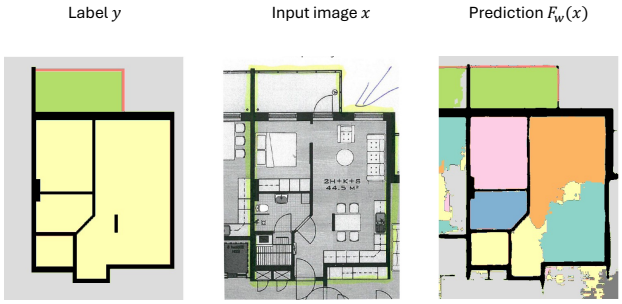
Despite its widespread use, CubiCasa5K has not been extensively analyzed in terms of its quality and domain properties, particularly in the context of domain adaptation. Such an analysis is vital as the dataset's quality and domain-specific characteristics directly impact the effectiveness of models trained on it.

As the first key contribution of our research, we provide a detailed analysis of the CubiCasa5K dataset to assess its suitability for domain adaptation research. CubiCasa5K samples are distributed across three distinct domains: high-quality architectural (3,732 samples), high-quality (992 samples), and colorful (276 samples), with examples shown in Figure 3a. These domains vary not only in their visual appearance but also in the complexity and

(a) The domains: high quality architectural, high quality, and colorful with their amount of samples. More samples can be seen in appendix A. Both the quantity and style are different per domain.



(b) The class types: rooms (**left**) and icons (**right**) with their classes and junctions. The dashed junctions represent openings used for doors and windows.



(c) Example of a problematic dataset instance: The image depicts a single room that combines elements from a kitchen, living room, bathroom, and bedroom, yet it is ambiguously labelled as undefined. Additionally, the areas outside the room are incorrectly labelled as background, illustrating ambiguous cutoffs. Additional examples of such labeling issues are provided in Appendix A.

Figure 3. The CubiCasa5K dataset.

consistency of annotations. The dataset was annotated by trained human annotators using a specialized CAD tool, and each annotated image underwent a two-stage quality assurance process to ensure accuracy and consistency.

Our analysis focuses on identifying variations in input features, e.g., pixel intensity, and output features, e.g., class distributions listed in Figure 3b. We also assess the labelling quality across different domains and data splits. By combining our examination of these factors with observations from our real-world problem, we define the source and target domains relevant to real-world applications. We further calculate metrics such as class- and split-specific pixel divergence and class density gaps to quantify the differences

between domains. Each analysis starts with qualitative visual inspections, followed by quantitative experiments to validate our observations. Finally, we interpret our findings in the context of setting up effective domain adaptation strategies for floor plan vectorization.

4.1. Pixel Distribution

Looking at the three domains present in the dataset, we observe distinct visual differences between the images, illustrated in Figure 3a. To quantify these differences, we compare the Red, Green, and Blue (RGB) input channels across each domain and the by Kalervo et al. [18] predetermined data split. We estimate histograms for pixel intensities in each channel, determining the probability distribution of pixel values within each domain [13]. Using these pixel density distributions, we calculate intra-domain, within the same domain, and inter-domain, between different domains, similarities using the Kullback-Leibler (KL) divergence.

The KL divergence between two probability distributions $P_{i,A}$ and $P_{i,B}$ for a given channel i (Red, Green, or Blue) in domains A and B, is defined for discrete values as:

$$\mathrm{KL}(P_{i,A} \| P_{i,B}) = \sum_{x} P_{i,A}(x) \log \frac{P_{i,A}(x)}{P_{i,B}(x)} \tag{13}$$

Here, $P_i(x)$ represents the probability of observing a specific pixel intensity x in channel i for domains $A : P_{i,A}(x)$ and $B : P_{i,B}(x)$. Given the non-symmetric nature of KL divergence, where $\mathrm{KL}(P_{i,A} \| P_{i,B}) \neq \mathrm{KL}(P_{i,B} \| P_{i,A})$, we calculate the symmetric KL divergence by averaging the two directions:

$$\mathrm{KL}_{\mathrm{sym}}(P_{i,A}, P_{i,B}) = \frac{1}{2} \left[\mathrm{KL}(P_{i,A} \| P_{i,B}) + \mathrm{KL}(P_{i,B} \| P_{i,A}) \right] \tag{14}$$

For the intra-domain divergence, we calculate the KL divergence between each pair of color channels (Red, Green, and Blue) within the same domain D. The intra-domain divergence is then averaged across these pairs using the formula:

$$\mathrm{KL}_{\mathrm{intra}}(D) = \frac{1}{3} \sum_{\substack{i,j \in \{R,G,B\} \\ i \neq j}} \mathrm{KL}(P_{i,D} \| P_{j,D}) \tag{15}$$

Here, $P_{i,D}(x)$ represents the probability distribution of the pixel intensities for channel i (Red, Green, or Blue) in domain D. For the inter-domain divergence, we compare the same color channels across different domains. Suppose we are comparing domains A and B. The inter-domain KL divergence is computed by averaging the symmetric KL divergences for the Red, Green, and Blue channels as:

$$\mathrm{KL}_{\mathrm{inter}}(A, B) = \frac{1}{3} \sum_{i \in \{R,G,B\}} \mathrm{KL}_{\mathrm{sym}}(P_{i,A}, P_{i,B}) \tag{16}$$

	A	H	C
A	-	0.05	0.24
H	0.05	-	0.27
C	0.39	0.40	-

Table 1. Average Kullback Leibner (KL) divergence for the input feature distribution from row to column ($A \rightarrow C = 0.24$), i.e. we compute the divergence per channel and take the average over the channels. The headers represent the (A) High Quality Architectural, (H) High Quality, and (C) Colorful datasets. The table shows that the largest gap is from and to the Colorful (C) domain.

which calculates the KL divergence in both directions for each color channel - Red, Green, and Blue - between the two domains and then takes the average. Our combinatory approach accounts for both the visual similarities within each domain and the differences between them. The results of these calculations are visualized in Figure 4. We also present the KL divergence between specific source and target domain combinations in Table 1.

The analysis shows that the divergence within the colorful domain is significantly larger, both within (intra) a split and between splits (inter). The divergence follows from the broad range of colors used in the domain, as observed during the qualitative analysis. Another key observation is the varying uniformity across the different splits. Such variation could potentially cause problems for domain adaptation, as the distribution learned during training may differ from that of the testing set.

While not explored in detail, we suspect that the single-color peaks in the high-quality architectural and high-quality domains are due to colored marker annotations, indicating interest in the circled areas, also visible in Figure 3c. An interesting, and possibly related, but not further investigated phenomenon can be seen at an all-channel rounded peak in the high-quality training data which is not only unique in its shape but is also missing from the validation and testing splits.

4.2. Label Distribution

Next, the label distribution was examined to understand the dominance of certain classes. Qualitative inspection indicated that some classes were more frequent than others, prompting a detailed quantitative analysis. To compare the classes we plot the density of the pixels in Figure 5 as a boxplot, with variance across data splits, per domain.

The analysis shows that for most classes both the variance across splits and across domains is small which is perfect if we want to isolate the effect of distributional input feature shift. However, there are noticeable differences between the distributions of some prominent classes. The shift indicates that the domains possibly have either differ-

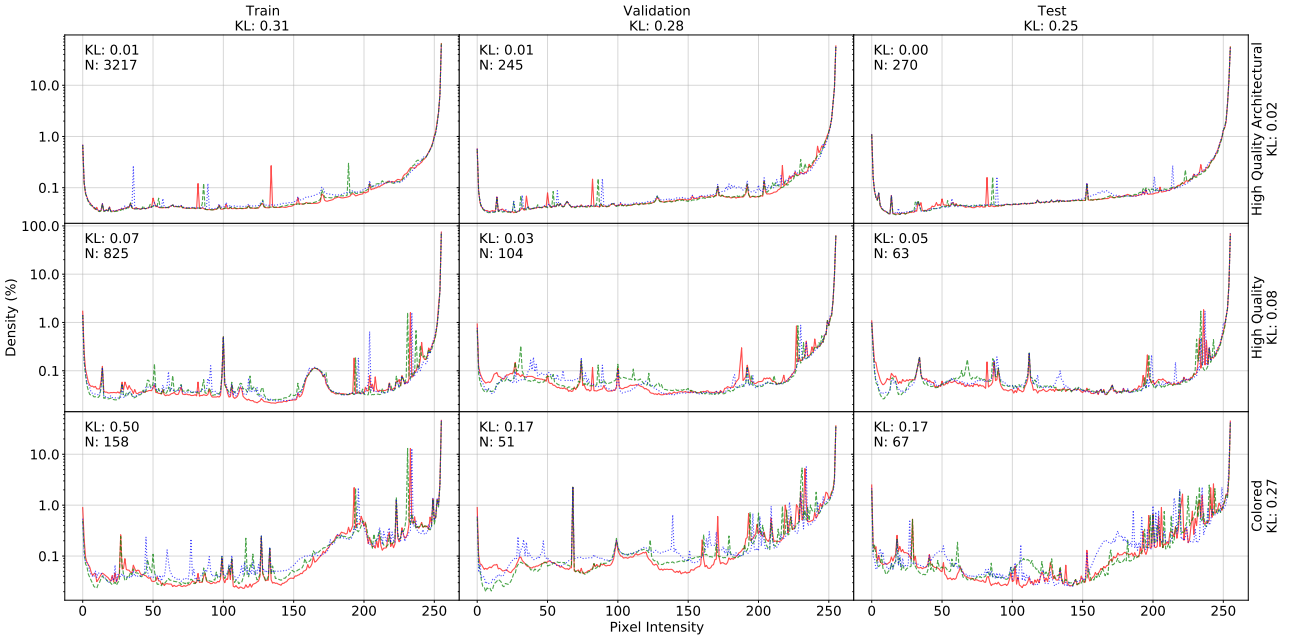


Figure 4. Input features (pixels intensity 0-255) distribution per channel (colors RGB), domain (rows) and data split (columns). The intra (within) Kullback-Leiber (KL) divergence as well as the number of samples (N) are shown for each plot. Each row and column also displays the inter-KL divergence. The plot shows large differences in pixel distribution, intra (within), and inter-row (between) divergences.

ent types of homes or include a specific labelling bias which should be considered when reasoning about domain adaptation settings.

4.3. Label Quality

Prompted by the observation that the domain with the largest quantity of labels performed the worst as well as the possible labelling bias, we assess the quality of the labelling to determine its correctness and clarity. The qualitative inspection showed instances of ambiguous labelling and unclear structural cutoffs. To quantify these issues, we analyzed two scenarios. First, how often the model predicts extra room masks where the label indicates the background class for structural cutoffs. Second, how frequently the label was undefined while the model predicted a specific room mask for ambiguous class labelling. Both issues are illustrated in the sample shown in Figure 3c.

To evaluate the structural integrity of the predicted room layouts, we introduce a binary mask M_s that identifies structural errors. M_s assigns a value of 1 to pixels where the model predicts a non-background class, but the ground truth indicates a background class. The mask is mathematically expressed as:

$$M_s(i,j) = \begin{cases} 1 & \text{if } L(i,j) = 0 \text{ and } P_r(i,j) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

where $L(i,j)$ represents the ground truth label at pixel (i,j), and $P_r(i,j)$ represents the predicted room label at pixel (i,j). Once M_s is defined, we calculate the relative structural error percentage S_{rel} as the ratio of structural error pixels to the total number of background pixels in the ground truth:

$$S_{\text{rel}} = \frac{\sum_{i,j} M_s(i,j)}{\sum_{i,j} \delta(L(i,j),0)} \tag{18}$$

Here, $\delta(L(i,j),0)$ is an indicator function that equals 1 if $L(i,j) = 0$ and 0 otherwise. Additionally, we compute the absolute structural error percentage S_{abs}, which is the ratio of structural error pixels to the total number of pixels in the image:

$$S_{\text{abs}} = \frac{\sum_{i,j} M_s(i,j)}{W \times H} \tag{19}$$

where $W \times H$ is the total number of pixels in the image. The structural error is considered significant if both S_{rel} and S_{abs} exceed their respective thresholds T_s and T_s'. For undefined regions, we define another binary mask M_u to identify undefined errors. The mask assigns a value of 1 to pixels where the model predicts a class different from the undefined class while the ground truth is labelled as undefined:

$$M_u(i,j) = \begin{cases} 1 & \text{if } P_r(i,j) \neq \ell_u \wedge 0 \text{ and } L(i,j) = \ell_u \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

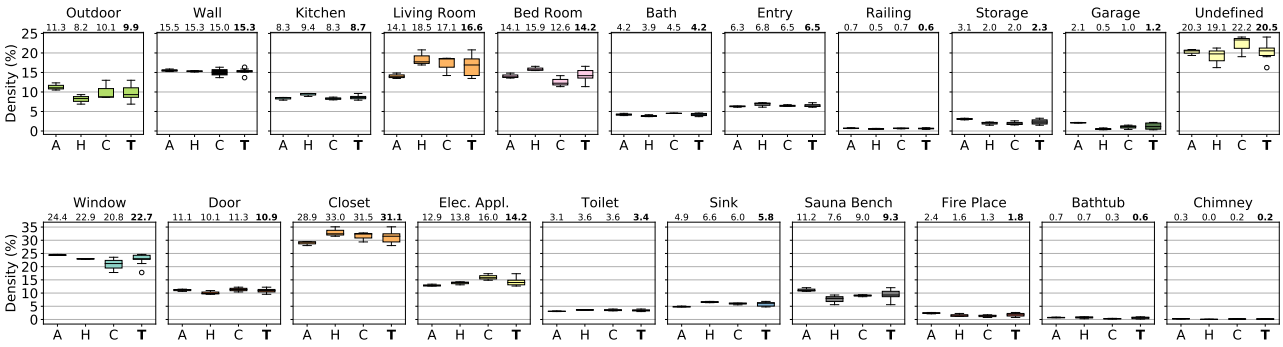


Figure 5. Class density distribution per domain with **top:** rooms and **bottom:** icons. The box is plotted based on the densities of the training, validation, and testing split. Here the measurements represent the variance for (A) High Quality Architectural, (H) High Quality, (C) Colored, and (T) Total data. The number above each class's boxplot is the mean density of the class in the respective data. The plots show that, although some small differences exist, the class densities are within the same order.

Here, ℓ_u represents the label for undefined regions in the ground truth. The relative undefined error percentage U_{rel} is then calculated as the ratio of undefined error pixels to the total number of non-background pixels in the prediction:

$$U_{\text{rel}} = \frac{\sum_{i,j} M_u(i,j)}{\sum_{i,j} \delta(P_r(i,j),0)} \tag{21}$$

where $\delta(P_r(i,j),0)$ is an indicator function that equals 1 if $P_r(i,j) \neq 0$ and 0 otherwise. Furthermore, the absolute undefined error percentage U_{abs} is calculated as the ratio of undefined error pixels to the total number of non-background pixels in the ground truth:

$$U_{\text{abs}} = \frac{\sum_{i,j} \delta(L(i,j),\ell_u)}{\sum_{i,j} \delta(L(i,j),0)} \tag{22}$$

where $\delta(L(i,j),\ell_u)$ is an indicator function that equals 1 if $L(i,j) = \ell_u$. The undefined error is considered significant if both U_{rel} and U_{abs} exceed their respective thresholds T_u and T'_u, chosen to balance sensitivity and filter out possible unclear cases.

We set thresholds in our evaluation to identify significant structural and undefined errors. For structural errors, the relative threshold $T_s = 0.3$ requires these errors to account for more than 30% of the background pixels. The absolute threshold $T'_s = 0.05$ ensures they cover more than 5% of the total image area, accommodating floor plans with minimal background. For undefined errors, the relative threshold $T_u = 0.3$ requires these errors to exceed 30% of non-background pixels in the prediction. The absolute threshold $T'_u = 0.5$ demands that undefined areas make up more than 50% of non-background pixels in the ground truth.

After an initial run, we manually inspect and exclude any images incorrectly labelled as ambiguous before rerunning the analysis to obtain the final results. During our inspection, we identified three major cases of potential ambiguous structural cutoffs. The first case involves situations where only part of the floor plan is labelled. The second case is where only the walls are annotated. The third case involves areas that are selectively labelled with a marker. While the third scenario is less problematic, as recognizing these areas could be part of the learning task, it is still worth discussing as it could inform future research directions. Although not explored in depth, we found that a large portion of the structural cutoffs are possibly being caused by the loading and pre-processing mechanism employed by Kalervo et al. [18].

While we recognize that our approach, the results reveal significant issues with ambiguous class labelling, as shown in Table 2. Examples include areas labelled as undefined that closely resemble kitchens and unclear structural boundaries where parts of the floor plan were left unlabeled as seen in Figure 3c. These ambiguities can cause errors in model predictions, leading to suboptimal training and conflicting patterns. Moreover, the extent of problematic labelling varies across domains and data splits, potentially impacting fair model evaluation.

4.4. Adaptation gap

Based on the data analysis, we define the source and target domains to reflect the problem setting where the input feature distributions differ, while the output class distributions remain, approximately, consistent. Specifically, the source domain includes high quality architectural and high-quality images, while the target domain comprises colorful images. The specific setting was chosen based on the larger divergence to the colorful domain and the small amount of annotated samples. Such a setting also fits in nicely with our objective to represent the realistic setting of adapting to older non-CAD floor plans.

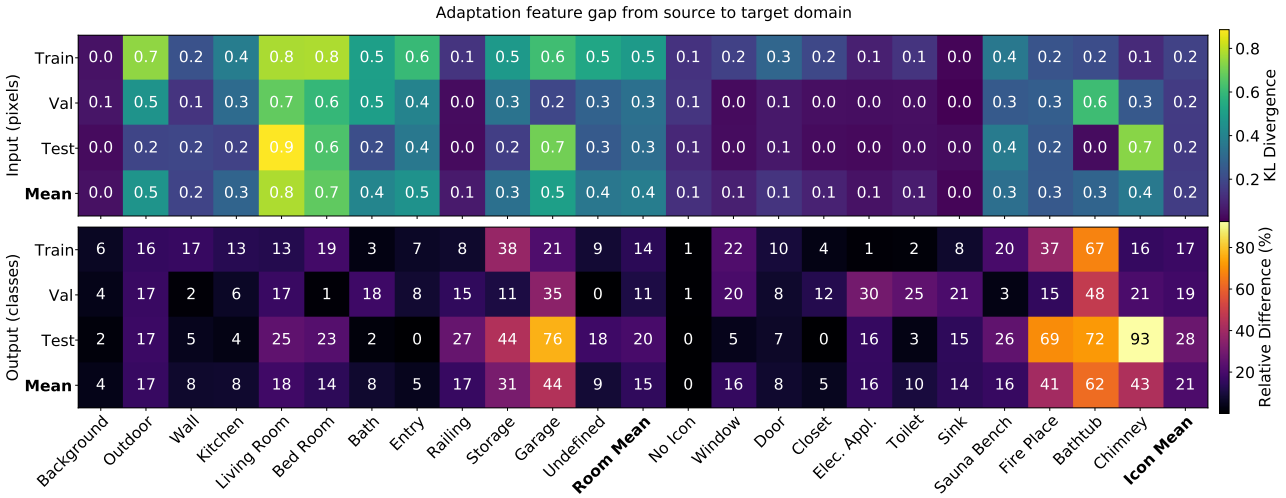


Figure 6. **Top:** Kullback-Leiber (KL) divergence and **bottom:** relative class density difference from source, containing both high quality and high quality architectural data, to target, the colorful data, domain. The table shows that the room classes have a larger divergence compared to the icons. Some classes that have a low frequency that show a larger divergence (i.e., Garage, Fire Place, Bathtub, Chimney) should be ignored as these are non-reliable.

	Structural			Undefined		
	Train	Val	Test	Train	Val	Test
A	8.0	6.5	8.5	3.6	4.9	3.3
H	1.6	0.0	3.2	2.1	1.0	1.6
C	1.3	0.0	4.5	3.8	2.0	7.5

Table 2. Percentage of errors per domain and data split for both structural cutoffs and ambiguous undefined labelling. The rows represent the (A) High Quality Architectural, (H) High Quality, and (C) colored data domains. The table shows a varying but significant amount of errors across domains and data splits which could result in contractionary learning and distributional shift.

In our analysis, we compare the data distributions between the source and target domains by focusing on pixel intensities and class distributions. For each class c, we estimate the probability density of pixel intensities $P_{c,s}(X_{ch})$ and $P_{c,t}(X_{ch})$ in each color channel ch (Red, Green, Blue) using histograms with 128 bins, covering 256 possible pixel values. Our approach thus allows for capturing small differences in pixel distributions while still being strict on larger gaps. We then calculated the Kullback-Leibler (KL) divergence between these distributions for each class and color channel, using the formula:

$$\mathrm{KL}(P_{c,s} \| P_{c,t}) = \sum_{i=1}^{128} P_{c,s}(i) \log \left(\frac{P_{c,s}(i)}{P_{c,t}(i)} \right) \tag{23}$$

Here, $P_c(i)$ represents the probability densities for class c and the i-th bin in the source $P_{c,s}(i)$ and target $P_{c,t}(i)$ domains. The KL divergence for each class c, averaged across all channels, is given by:

$$\mathrm{KL}^c = \frac{1}{3} \sum_{ch \in \{R,G,B\}} \mathrm{KL}(P_{c,s}(X_{ch}) \| P_{c,t}(X_{ch})) \tag{24}$$

The final KL^c value provides a single divergence measure for each class, reflecting the overall difference in pixel intensity distributions from the source to the target domain. Additionally, we computed the relative frequency of each class in both the source r_s^c and target r_t^c domain:

$$r_s^c = \frac{L_{s,c}}{L_{s,\text{total}}}, r_t^c = \frac{L_{t,c}}{L_{t,\text{total}}} \tag{25}$$

where $L_{s,c}$ and $L_{t,c}$ are the counts of pixels labelled as class c in the source and target domains, and $L_{s,\text{total}}$ and $L_{t,\text{total}}$ are the total number of labelled pixels in each domain. The relative percentage difference between these frequencies was calculated using:

$$\mathcal{F}^c = \frac{|r_s^c - r_t^c|}{\max(r_s^c, r_t^c) + \epsilon} \times 100 \tag{26}$$

where $|r_s^c - r_t^c|$ represents the absolute difference between the relative frequencies of class c in the source and target domains, and ϵ is a small positive value added to the denominator to prevent division by zero. Finally, we compiled the KL^c and $\mathcal{F}^c$ values into heatmaps, which visually represent the magnitude of these discrepancies which can be found in Figure 6. These heatmaps highlight areas where domain adaptation may be challenging due to differences in

data distribution, providing insights into potential sources of model IoU degradation. Note that these values should always be interpreted in conjunction with the number of samples available in the test set for each class, displayed in Figure 7. For infrequent classes like Garage, Fireplace, Bathtub, and Chimney, the probability density estimates are based on a limited number of instances, which may lead to estimates that do not fully represent the true distribution.

5. Method

Our method section outlines our approach to studying domain adaptation in scenarios where the feature distributions differ between the source and target domains ($P(X_s) \neq P(X_t)$), while the label distributions remain consistent ($P(Y_s) \approx P(Y_t)$). Consistent $P(Y)$ across domains and data splits, displayed in Figure 6, allows us to isolate the impact of distributional feature shift. We begin by replicating the CubiCasa5K model [18] to ensure our findings align with existing research. We then compare this model with different training, domain and data configurations.

Next, we explain how we implement the Maximum Mean Discrepancy (MMD) to align the feature distributions of the source and target domains. We then focus on extracting latent features for MMD, evaluating different network positions and pooling strategies. Following the extraction, we discuss the weighting of MMD loss relative to task-specific losses, investigating both constant and variable strategies. The section details our training procedures, including data pre-processing and optimization, and our evaluation methods.

5.1. Baselines

We first reproduce the CubiCasa5K model [18] using the same datasets as the original paper to ensure our results align with the reported findings. To improve training efficiency, we test low-precision matrix multiplication with two settings: "highest," which uses full float32 precision, and "medium," which uses bfloat16 precision to speed up computations. We compare these settings to evaluate the trade-offs between speed and accuracy. We also modify the optimization strategy to accelerate convergence. The original model reduces the learning rate after 20 epochs without improvement, lowering it by a factor of 0.1. We reduce the learning rate after 10 epochs by a factor of 0.5, allowing for faster experimentation. Additionally, we assess the model's IoU across different domains, which the original paper by Kalervo et al. [18] does not explore.

For baseline experiments, we use three configurations: Full Adaptation (Original), where we replicate the original data split with both source and target data; Full Adaptation (Corrected), where we remove some source data to keep the total sample size consistent while retaining all target data; and No Adaptation, where we train only on source data to evaluate how well the model generalizes to the target domain.

5.2. MMD Implementation

Formally, the source domain is defined by samples $X_s = \{x_s^1, x_s^2, \ldots, x_s^{n_s}\}$ with feature distribution $P(X_s)$ and labels $Y_s = \{y_s^1, y_s^2, \ldots, y_s^{n_s}\}$. The target domain is characterized by samples $X_t = \{x_t^1, x_t^2, \ldots, x_t^{n_t}\}$ with feature distribution $P(X_t)$ and labels $Y_t = \{y_t^1, y_t^2, \ldots, y_t^{n_t}\}$. To quantify the discrepancy between $P(X_s)$ and $P(X_t)$, we employ the Maximum Mean Discrepancy (MMD) as defined in equation 10 with a Radial Basis Function (RBF) [22] which is defined as:

$$k(x_a^i, x_b^j) = \exp\left(-\frac{\|x_a^i - x_b^j\|^2}{2\sigma^2}\right) \tag{27}$$

in which $\|x_a^i - x_b^j\|^2$ represents the squared Euclidean distance between the two data points x_a^i and x_b^j. The RBF kernel measures the similarity between these points by applying an exponential function to the negative squared Euclidean distance, scaled by $2\sigma^2$. The kernel gives higher similarity scores to closer points, with similarity decreasing exponentially as distance increases, while enabling the handling of non-linear relationships.

The parameter σ, known as the bandwidth, controls the width of the kernel function. A smaller σ results in a more localized kernel, focusing on close neighbours, while a larger σ broadens the kernel, allowing a wider range. In our implementation, we calculate the bandwidth by averaging the squared pairwise distances between all sample points in the dataset, which provides a measure of the typical distance scale in the data.

We extend MMD to its Multi-Kernel variant MK-MMD, as described in equation 12, and use a set of RBF kernels with varying bandwidths. We start with a base bandwidth and then create multiple kernels by scaling the base bandwidth. Specifically, we generate 5 kernels, each with a bandwidth scaled by a multiplicative factor of 2.0 from the base bandwidth. The scaling allows us to capture discrepancies between distributions at different scales. For each RBF kernel k, we compute the MMD squared between the source samples X_s and target samples X_t. To obtain the final Multi-Kernel MMD loss, we average the MMD squared values across all kernels.

5.3. Latent extraction

Using the definition for MK-MMD, we now need to determine where we extract the latent vector. The vector is typically taken from the latent space of a network, situated between the encoder and decoder of our network. CubiCasa5K's [18] U-shaped network can be seen in Figure 1 with its layers l_{1-15}. Below the layers, the total size of

the image of size s is shown. In the case of l_1, the size is s with 3 channels thus resulting latent size of $3s$. The down-sampling is done through convolutional and maxpool blocks and is displayed as s/n, where both the height h and width w are scaled by a factor of $1/n$.

The figure shows that the latent space of the network should be located between layers l_6 and l_7. Zooming in on l_7 we see that there are two options to extract the latent feature vector. We define these positions as l_{7a} with 256 and l_{7b} with 512 channels. In our research, we investigate the role of the latent extraction position on class IoU. For each channel configuration, we also evaluate various pooling strategies:

- 4×4 **Pooling (Base)**: retains the original latent feature map size of 4×4 without applying any further pooling, as the vector is already reduced to this size by layer l_7.

- 2×2 **Pooling**: reduces the size of the latent feature map from 4×4 to 2×2, effectively shrinking the spatial dimensions by a factor of four.

- 1×1 **Pooling**: reduces the latent feature map to a single point by applying pooling that downsizes the 4×4 map to 1×1, reducing the spatial dimensions by a factor of 16.

Our pooling analysis is required as we require a uniform feature size and the input size to the network is dynamic (i.e., we allow for varying image sizes). The pooling method avoids the need for learnable weights, which would require proper back-propagation and a uniform feature sizing somewhere in the network. Although theoretically, such a layer could provide more precise adaptation, it would drastically change the architecture and behaviour of our model which is both out of scope and not the goal of our research. After the pooling is applied we flatten the feature maps to gain a latent vector that can be used in the MMD calculation. The loss is then incorporated into the total loss as:

$$\mathcal{L} = L_S + L_H + \lambda \mathcal{L}_{\text{MMD}} \tag{28}$$

where L_S encompasses the loss for the room and icon segmentation tasks, L_H covers the loss for heatmap-based predictions for the junctions, and $\lambda \mathcal{L}_{\text{MMD}}$ is the MMD loss component. The weight λ is used to balance the contribution of MMD loss relative to the task-specific losses.

5.4. Adaptation weighting

The weights of task-specific losses are dynamically adjusted using the uncertainty parameters which makes it less trivial to find an optimal lambda. One solution would be to also add an uncertainty parameter for the domain adaptation weighting but this does allow us to understand the mechanics of the MMD alignment over training as the weighting is adjusted each epoch. As no research has been done on combining these losses we first investigate how the choice of lambda affects the adaptation. To isolate the weighting problem we focus on a latent vector extraction at l_{7b} in combination with 1×1 adaptive pooling block to get a 512-channel long latent vector that can be directly used in the MMD calculation.

In the first set of experiments, we apply constant weighting to the MMD loss. The weighting parameter λ is defined as $\lambda = \lambda_c$ where λ_c represents the constant weight assigned to the MMD loss. We determine λ_c based on initial observations where we estimate that about 10% and decrease λ_c to observe its effect.

Following the identification of the best performing λ_c value, a fine-grained search is performed within the range around the best lambda value. In the fine-grained search, λ is still defined as $\lambda = \lambda_c$, with λ_c chosen by linearly searching in the space around the best-performing lambda.

In the final set of experiments, variable weighting is applied to the MMD loss. The weighting parameter λ is defined as $\lambda = \lambda_c \cdot \lambda_v$ where λ_v is a variable component that scales with epoch progression. The variable component λ_v is defined as:

$$\lambda_v = \frac{2}{1 + e^{-d \cdot \frac{E_c}{E_{\max}}}} - 1 \tag{29}$$

where $d = 5$ determines the steepness of the sigmoid function, E_c represents the current epoch, and $E_{\max}$ represents the maximum number of epochs. The constant λ_c is chosen from a range based on the initial constant weighting experiments. The specific values for λ_c are determined through empirical loss analysis and iterative adjustments, reflecting different approaches to balance the MMD loss with task-specific losses.

5.5. Training

The dataset is preprocessed by loading the SVG files into a house object that includes room, icon, and heatmap masks for each class which can be seen in Figure 3b. No additional preprocessing steps are applied beyond the initial loading. Similar to CubiCasa5K [18] and R2V Liu et al. [24], the network is first pre-trained on ImageNet [6, 33] and the MPII Human Pose Dataset [2]. For data augmentation, we follow the same approach as CubiCasa5K. The data loader uses a batch size of 20 over which the MMD is calculated during training. The data loader randomly matches images from the target dataset to the source dataset with the restriction that they should be of similar shape (i.e., max 10% difference) as they are used in a combined loader. During validation, we calculate the MMD loss, which requires a set of points, over the entire validation set as we use a batch size of 1.

For the first 100 epochs, training augmentations include 90-degree rotations, color jitter, and cropping or resizing

		Rooms												Icons									
A	$+$	Background	Outdoor	Wall	Kitchen	Living	Bed	Bath	Entry	Railing	Storage	Undefined	**Mean**	No Icon	Window	Door	Closet	Elec. Appl.	Toilet	Sink	Sauna Bench	**Mean**
F_{cc}	$-$	**83.3**	**54.0**	51.3	50.2	**59.3**	72.5	**59.9**	**55.0**	6.4	**46.2**	41.3	52.7	96.8	39.1	44.2	**65.4**	**64.4**	61.0	45.4	59.4	59.5
F_o	$-$	82.1	49.2	**60.9**	**54.4**	56.8	**75.8**	57.5	54.7	**11.0**	37.3	**47.9**	**53.4**	**97.1**	**54.8**	**50.8**	65.0	63.5	**64.2**	**52.3**	**74.2**	**65.2**
	$-$	82.1	49.2	60.9	54.4	**56.8**	**75.8**	57.5	54.7	11.0	37.3	47.9	53.4	97.1	54.8	50.8	65.0	63.5	**64.2**	52.3	**74.2**	65.2
F_o	P	**84.1**	**53.4**	**62.6**	**57.5**	56.6	74.5	**61.4**	**60.5**	**14.1**	**49.1**	**49.4**	**56.6**	**97.2**	**55.2**	**51.9**	65.7	**65.4**	62.2	**53.1**	72.0	**65.3**
	P, S	81.1	46.3	60.5	56.7	53.8	75.0	58.9	56.0	10.3	46.9	45.3	53.7	97.1	53.9	48.2	**66.2**	62.9	62.3	51.6	73.8	64.5
	$-$	**83.1**	**57.0**	**61.4**	54.5	54.3	74.5	58.8	**58.6**	11.8	45.6	**51.0**	**55.5**	**97.1**	**53.9**	50.4	65.2	**63.5**	**62.1**	52.0	72.7	**64.6**
F_c	P	81.7	51.2	60.6	54.6	**54.8**	**75.9**	60.8	55.7	**13.6**	**45.8**	48.2	54.8	97.0	50.5	**51.0**	65.1	60.8	62.1	52.7	**75.7**	64.4
	P, S	81.0	49.9	59.6	**55.1**	54.8	74.8	**61.2**	55.2	10.2	39.1	50.3	53.7	97.0	50.4	49.4	**66.0**	61.3	60.7	**53.7**	70.5	63.6
	$-$	81.8	50.2	60.4	**56.6**	54.7	75.4	57.2	**56.7**	8.9	**49.8**	**55.8**	**55.2**	**97.1**	54.4	47.5	65.2	**63.5**	61.9	**52.7**	**74.5**	**64.6**
N	P	**82.4**	**53.3**	**61.2**	56.3	54.2	**76.5**	**57.3**	53.1	**11.9**	39.6	52.1	54.4	97.1	**55.7**	**49.2**	64.8	62.8	60.7	50.3	72.5	64.1
	P, S	82.1	49.6	59.8	55.9	**55.3**	75.2	56.6	56.6	10.2	32.9	49.3	53.0	97.0	55.2	49.1	**65.4**	62.9	**62.3**	52.7	69.2	64.2

Table 3. Ablation study in Intersection over Union (IoU) for CubiCasa5K's [18] best model F_{cc} and our implementations: F_o Full Adaptation (original), F_c Full Adaptation (corrected), and N No Adaptation. The $-$ represents the setting of CubiCasa, we adjust these with a lower matrix precision P for speed and scheduler S that converges faster.

with zero padding to 256x256. After 100 epochs, we continue training with the best weights obtained, resetting the optimizer parameters and removing the resizing augmentation. We then train the network for an additional 400 epochs until convergence. We employed the Adam optimizer, setting the initial learning rate to 1×10^{-3}, with $\epsilon = 1 \times 10^{-8}$, and the beta parameters set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$. In contrast to CubiCasa5K [18], we implemented a learning rate scheduler that reduces the learning rate by a factor of 0.5 if no improvements are observed in the validation loss for 10 epochs.

5.6. Evaluation

For the evaluation, we also follow CubiCasa5K [18] but focus on using class Intersection over Union (IoU) as our primary metric because it gives a clear and balanced measure of how well the segments match [30]. While we do record other metrics, we exclude them from our discussion as they do not offer additional insights into class-specific domain adaptation performance. Given a predicted segmentation map P and a corresponding ground truth segmentation map G, IoU for a specific class c is defined as:

$$\text{IoU}(c) = \frac{|P \cap G|_c}{|P \cup G|_c} = \frac{TP_c}{TP_c + FP_c + FN_c} \tag{30}$$

where $|P \cap G|_c$ is the cardinality of the intersection set for class c, representing the number of pixels that are correctly predicted as belonging to class c (True Positives, TP_c). $|P \cup G|_c$ is the cardinality of the union set for class c, representing the total number of pixels that are either predicted as class c or are part of class c in the ground truth. The union includes True Positives TP_c, False Positives FP_c, and False Negatives FN_c. TP_c is the number of pixels correctly classified as class c. FP_c is the number of pixels incorrectly classified as class c when they actually belong to a different class. FN_c is the number of pixels that belong to class c but are not predicted as such. Given the class IoUs, we can now average them to obtain the mean Intersection over Union (mIoU):

$$\text{mIoU} = \frac{1}{C} \sum_{i=1}^{C} \text{IoU}(i) \tag{31}$$

where C is the total number of classes in the dataset. The metric provides a robust measure of the accuracy of the semantic segmentation by quantifying the overlap between predicted and ground truth segments. We leave out the other metrics, although we do measure them, in our research as they do not provide any additional insights into the adaptation capabilities. To ensure fair evaluation, we exclude classes with few instances, such as Fireplace, Bathtub, and Chimney as can be seen in Figure 7, to prevent sample-specific results skewing. Additionally, we apply test time augmentation by rotating input images at four angles (0°, 90°, 180°, and 270°) feeding all these images through the model and then taking the mean.

6. Results

6.1. Baselines

Ablation In this experiment we aim to research the influence of our training adjustments. The IoU of CubiCasa5K's

T	A	Rooms												Icons									
		Background	Outdoor	Wall	Kitchen	Living	Bed	Bath	Entry	Railing	Storage	Undefined	**Mean**	No Icon	Window	Door	Closet	Elec. Appl.	Toilet	Sink	Sauna Bench	**Mean**
S	F_c	85.7	66.5	78.5	62.2	58.1	75.3	61.3	53.5	29.4	39.7	49.5	60.0	98.0	72.4	62.6	70.0	73.0	70.4	60.4	85.5	74.0
	N	87.4	64.4	79.0	62.3	59.0	76.2	59.1	55.1	26.9	38.0	49.9	59.8	98.0	71.8	61.9	69.4	72.8	70.8	59.6	82.7	73.4
	Δ	1.7	-2.1	0.5	0.1	0.9	0.9	-2.2	1.6	-2.5	-1.7	0.4	-0.2	0.0	-0.6	-0.7	-0.6	-0.2	0.4	-0.8	-2.8	-0.6
V	F_c	81.0	49.9	59.6	55.1	54.8	74.8	61.2	55.2	10.2	39.1	50.3	53.7	97.0	50.4	49.4	66.0	61.3	60.7	53.7	70.5	63.6
	N	82.1	49.6	59.8	55.9	55.3	75.2	56.6	56.6	10.2	32.9	49.3	53.0	97.0	55.2	49.1	65.4	62.9	62.3	52.7	69.2	64.2
	Δ	1.1	-0.3	0.2	0.8	0.5	0.4	-4.6	1.4	0.0	-6.2	-1.0	-0.7	0.0	4.8	-0.3	-0.6	1.6	1.6	-1.0	-1.3	0.6

Table 4. Adaptation gap Δ in Intersection over Union (IoU) between the corrected full (F_c) and (N) no adaptation models. Here the column T represents the score type which is either (S) segmentation or (V) vectorization. The table shows there is an adaptation gap, yet there are multiple class instances that exhibit contractionary behaviour.

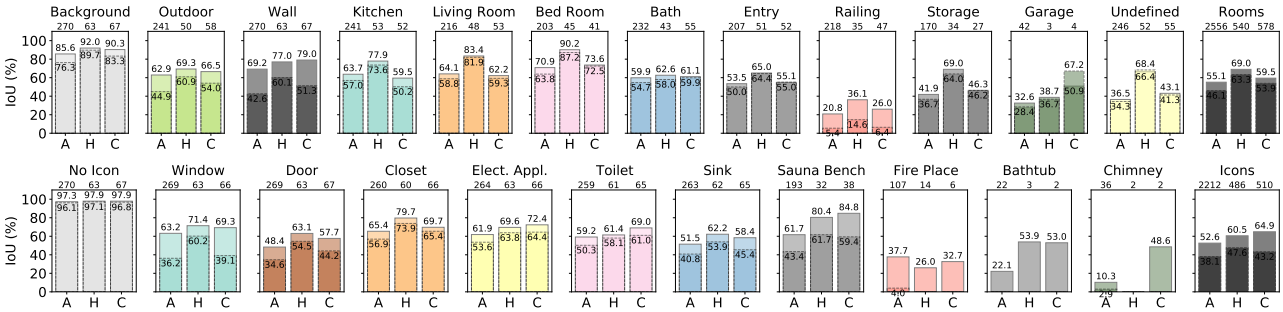


Figure 7. Domain Intersection over Union (IoU) with **top:** rooms and **bottom:** icons. Here the measurements represent the segmentation (line) and vectorization (dashed) intersection over union (IoU) for the (A) High Quality Architectural, (H) High Quality, and (C) Colored data. The number above each class's boxplot is the number of images in test data that contain the class. The difference in IoU shows that even with a larger dataset, High Quality Architectural (A) is harder to predict. It also shows that less frequent classes have no vectorization success. The last observation is that the vectorization of icons from the colorful (C) domain is much harder compared to the other domains.

model F_{cc} [18], listed in Table 3, shows a degraded IoU relative to our evaluation with the same settings and data. For all three of our own implementations, F_o Full Adaptation (original), F_c Full Adaptation (corrected), and N No Adaptation, we observe a general pattern. Specifically, there is a slightly lower Intersection over Union (IoU) when we introduce lower precision matrix multiplication and a faster-converging scheduler. However, there are exceptions to this pattern, such as in row $F_o, -$. The P, S rows of the corrected full adaptation as well as the no adaptation run are used to address the adaptation gap and function as a baseline during the experiments.

Domain performance In this experiment, we explore how the model performs for the different domains defined by CubiCasa5K [18]. Figure 7 shows that there is a varying IoU difference for both domains, classes, segmentation and vectorization. One noticeable observation is the fact that rooms from the colorful domain seem harder to segment compared to icons, in contrast with the other domains. Additionally, it shows that the vectorization process generally reduces scores and that this gap is significantly larger for icons and the colorful domain. It is apparently so hard to segment the least frequent icons - Fire Place, Bathtub, and Chimney - that almost no vectorization score is achieved over all three domains.

Adaptation gap Before researching the adaptation gap, we first need to determine if it exists and how significant it is. The results in Table 4 reveal that domain shift affects the IoU, with noticeable gaps in the "Bath," "Storage," and "Outdoor" room classes, as well as the "Sink" and "Sauna Bench" icon classes. Interestingly, some classes, particularly "Window," perform better without the colorful domain in the training data, likely due to labelling biases discussed in section 4.3. The IoU difference between rooms and icons may be related to the divergence shown in Figure 6. Additionally, there's a shift in IoU between segmentation and vectorization. The adaptation gap is negative for both in segmentation but widens for rooms and improves for icons

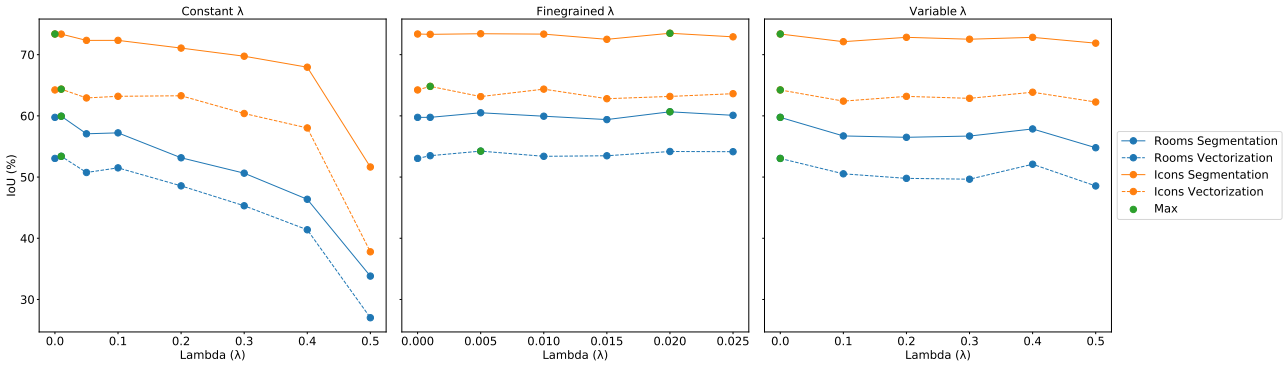


Figure 8. The results of the MMD weighting experiments with **left:** constant lambda **middle:** fine-grained lambda **right:** variable lambda and where $\lambda = 0$ means that no adaptation is done. The results show that a constant lambda ultimately collapses the intersection over Union (IoU), that the experiments suffer inconsistency problems due to batch sampling, and that a variable lambda shows more potential.

when vectorized, indicating junction heatmaps behave differently for rooms and icons.

6.2. MMD Weighting

Constant Weighting Given the adaptation gap, we aim to investigate the role of weighting, beginning with the simplest approach: constant weighting. In the constant weighting experiments displayed in Figure 8, the best performing λ value was 0.01, which showed a slight improvement over not using MMD. The general trend indicated that larger λ values resulted in a collapsing IoU, although class-specific exceptions exist as can be seen in Table 5a, resulting in a 20% drop in mIoU for $\lambda = 0.5$. The conclusion from the experiments is that increasing the constant λ to achieve domain alignment does not benefit adaptation in general. Interestingly, the two classes with the largest adaptation gaps per type, Bathroom and Sauna Bench, had the best IoU with a larger λ, suggesting they do benefit from alignment.

Constant Weighting (Fine-Grained) Using the optimal constant weighting we identified, we explore whether a better solution can be found in its immediate surroundings. In the fine-grained search, no clear pattern emerged for the best λ value as can be seen in Figure 8, indicating that results might be influenced by random batch sampling during training. The same phenomenon is also clearly observable in the class-specific IoU seen in Table 5a. The conclusion is that fixed weighting for the MMD implementation is not effective, and experiments, especially for smaller λ deltas, are sensitive to random batch sampling, making it difficult to draw definitive conclusions.

Variable Weighting Given the unsatisfactory results of our previous experiments, we now aim to determine if the model can benefit from variable weighting. In the variable weighting experiments, the IoU collapse is less prominent, and less affected by random batch sampling as seen in Figure 8. Classes with an adaptation gap performed better with larger λ values, while others did better with no alignment, as shown in Table 5a. The experiment shows that increasing domain alignment can improve the IoU, but random batch sampling combined with the uncertainty loss affected by λ leads to inconsistent results, making it hard to draw strong conclusions.

6.3. Latent extraction

Latent size Given the adaptation gap, we also want to investigate the role of latent size. The results in Figure 1 and Table 5b show that MMD extraction at 512 (l_{7b}) channels is preferred over 256 (l_{7a}) for all segmentation IoUs. Since results with smaller lambdas can vary, we focus on the larger lambda experiments for the rest of this analysis. In these, vectorization behaves differently from segmentation, generally benefiting from alignment at l_{7a} with 256 channels. We believe segmentation benefits from alignment at the end of the contraction phase, where fine details are ready for upsampling, while l_{7a} may be better for capturing global relationships, like those between junctions.

Pooling size After the latent size, we now want to assess how different pooling methods influence the model's behaviour. For pooling, 2x2 pooling was beneficial for icons for both segmentation and vectorization, highlighting the importance of structural preservation, even when the space is flattened to a vector. We also found that the speed gain from using smaller latent sizes was found to be negligible. An interesting future direction, found in a later stage of our research, could involve balancing the pooling strategies by using a weighted sum approach, possibly allowing for both global and local alignment.

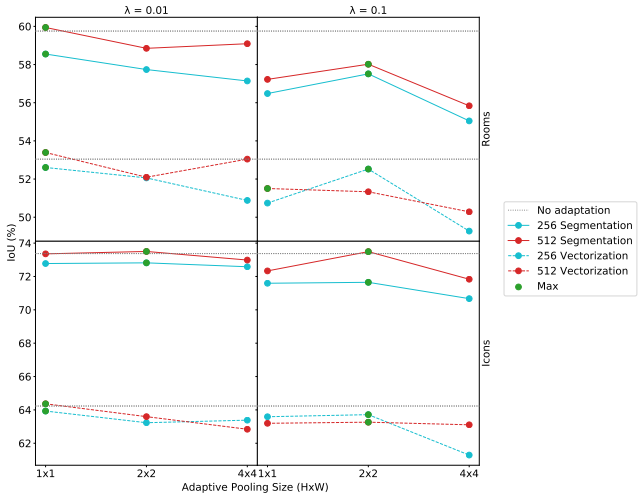


Figure 9. The results of the latent space experiments plotted in a grid for $\lambda \in 0.01, 0.1$ and the class types rooms and icons. In the experiments, the latent space is max pooled to the adaptive pooling size. It shows a balanced 2 by 2 pooling is beneficial for all $\lambda = 0.1$ cases.

7. Conclusion

Our paper identifies and analyzes key issues within the CubiCasa5K dataset, particularly concerning its labelling strategy, which requires further investigation for effective domain adaptation. Our study is the first to measure domain differences in the CubiCasa5K dataset both in terms of feature representation and model IoU, providing new insights into its domain-specific challenges and readiness for adaptation research. The findings from our domain alignment experiments indicate some benefits but are limited by the inconsistent data quality and variability observed during training. Although domain adaptation techniques for floor plan vectorization deserve further exploration, this must be preceded by rigorous dataset analysis. The inconsistencies in the CubiCasa5K dataset highlight the critical need for high-quality, standardized datasets to fully realize the potential of not just domain adaptation but also floor plan vectorization as a whole. Our findings underscore the critical importance for practitioners and researchers to ensure data quality before creating or benchmarking datasets.

References

[1] Christian Ah-Soon and Karl Tombre. Variations on the analysis of architectural drawings. In *Proceedings of the fourth international conference on document analysis and recognition*, pages 347–351. IEEE, 1997. 2

[2] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, pages 3686–3693, 2014. 2, 11

[3] Róger Bermúdez-Chacón, Pablo Márquez-Neila, Mathieu Salzmann, and Pascal Fua. A domain-adaptive two-stream u-net for electron microscopy image segmentation. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 400–404. IEEE, 2018. 3

[4] Ji Chang, Jing Li, Yu Kang, Wenjun Lv, Ting Xu, Zerui Li, Wei Xing Zheng, Hongwei Han, and Haining Liu. Unsupervised domain adaptation using maximum mean discrepancy optimization for lithology identification. *Geophysics*, 86(2): ID19–ID30, 2021. 3

[5] Chao Chen, Zhihang Fu, Zhihong Chen, Sheng Jin, Zhaowei Cheng, Xinyu Jin, and Xian-Sheng Hua. Homm: Higher-order moment matching for unsupervised domain adaptation. In *Proceedings of the AAAI conference on artificial intelligence*, pages 3422–3429, 2020. 5

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 11

[7] Samuel Dodge, Jiu Xu, and Björn Stenger. Parsing floor plan images. In *2017 Fifteenth IAPR international conference on machine vision applications (MVA)*, pages 358–361. IEEE, 2017. 2

[8] Philippe Dosch, Karl Tombre, Christian Ah-Soon, and Gérald Masini. A complete system for the analysis of architectural drawings. *International Journal on Document Analysis and Recognition*, 3(2):102–116, 2000. 2

[9] Abolfazl Farahani, Sahar Voghoei, Khaled Rasheed, and Hamid R Arabnia. A brief review of domain adaptation. *Advances in data science and information engineering: proceedings from ICDATA 2020 and IKE 2020*, pages 877–894, 2021. 2, 3

[10] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015. 3

[11] Muhammad Ghifary, W Bastiaan Kleijn, and Mengjie Zhang. Domain adaptive neural networks for object recognition. In *PRICAI 2014: Trends in Artificial Intelligence: 13th Pacific Rim International Conference on Artificial Intelligence, Gold Coast, QLD, Australia, December 1-5, 2014. Proceedings 13*, pages 898–904. Springer, 2014. 3

[12] Lucile Gimenez, Sylvain Robert, Frédéric Suard, and Khaldoun Zreik. Automatic reconstruction of 3d building models from scanned 2d floor plans. *Automation in Construction*, 63:48–56, 2016. 1

[13] Boqing Gong, Kristen Grauman, and Fei Sha. Learning kernels for unsupervised domain adaptation with applications to visual object recognition. *International Journal of Computer Vision*, 109(1):3–27, 2014. 6

[14] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. A kernel method for the two-sample-problem. *Advances in neural information processing systems*, 19, 2006. 3, 4

[15] Arthur Gretton, Dino Sejdinovic, Heiko Strathmann, Sivaraman Balakrishnan, Massimiliano Pontil, Kenji Fukumizu, and Bharath K Sriperumbudur. Optimal kernel choice for

large-scale two-sample tests. *Advances in neural information processing systems*, 25, 2012. 3

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[17] Xinyang Jiang, Lu Liu, Caihua Shan, Yifei Shen, Xuanyi Dong, and Dongsheng Li. Recognizing vector graphics without rasterization. *Advances in Neural Information Processing Systems*, 34:24569–24580, 2021. 2, 3

[18] Ahti Kalervo, Juha Ylioinas, Markus Häikiö, Antti Karhu, and Juho Kannala. Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis. In *Image Analysis: 21st Scandinavian Conference, SCIA 2019, Norrköping, Sweden, June 11–13, 2019, Proceedings 21*, pages 28–40. Springer, 2019. 2, 3, 5, 6, 8, 10, 11, 12, 13

[19] Hae-Kyong Kang and Ki-Joune Li. A standard indoor spatial data model—ogc indoorgml and implementation approaches. *ISPRS International Journal of Geo-Information*, 6(4):116, 2017. 1

[20] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018. 2

[21] Seongyong Kim, Seula Park, Hyunjung Kim, and Kiyun Yu. Deep floor plan analysis for complicated drawings based on style transfer. *Journal of Computing in Civil Engineering*, 35 (2):04020066, 2021. 1, 2

[22] Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *International conference on machine learning*, pages 1718–1727. PMLR, 2015. 10

[23] Chenxi Liu, Alexander G Schwing, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Rent3d: Floor-plan priors for monocular layout estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3413–3421, 2015. 5

[24] Chen Liu, Jiajun Wu, Pushmeet Kohli, and Yasutaka Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2195–2203, 2017. 1, 2, 5, 11

[25] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision*, pages 2200–2207, 2013. 3

[26] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015. 3, 5

[27] Xiaolei Lv, Shengchu Zhao, Xinyang Yu, and Binqiang Zhao. Residential floor plan recognition and reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16717–16726, 2021. 2

[28] Sébastien Macé, Hervé Locteau, Ernest Valveny, and Salvatore Tabbone. A system to detect rooms in architectural floor plan images. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 167–174, 2010. 2

[29] Jaehwa Park and Young-Bin Kwon. Main wall recognition of architectural drawings using dimension extension line. In *Graphics Recognition. Recent Advances and Perspectives: 5th International Workshop, GREC 2003, Barcelona, Spain, July 30-31, 2003, Revised Selected Papers 5*, pages 116–127. Springer, 2004. 1, 2

[30] Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In *International symposium on visual computing*, pages 234–244. Springer, 2016. 12

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015. 2

[32] Mohd Ezanee Rusli, Mohammad Ali, Norziana Jamil, and Marina Md Din. An improved indoor positioning algorithm based on rssi-trilateration technique for internet of things (iot). In *2016 international conference on computer and communication engineering (ICCCE)*, pages 72–77. IEEE, 2016. 1

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015. 11

[34] Kathy Ryall, Stuart Shieber, Joe Marks, and Murray Mazer. Semi-automatic delineation of regions in floor plans. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, pages 964–969. IEEE, 1995. 2

[35] Ruoxi Shi, Xinyang Jiang, Caihua Shan, Yansen Wang, and Dongsheng Li. Rendnet: Unified 2d/3d recognizer with latent space rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5408–5417, 2022. 2

[36] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014. 3

[37] Rebekka Volk, Julian Stengel, and Frank Schultmann. Building information modeling (bim) for existing buildings—literature review and future needs. *Automation in construction*, 38:109–127, 2014. 1

[38] Weilin Xu, Liu Liu, Sisi Zlatanova, Wouter Penard, and Qing Xiong. A pedestrian tracking algorithm using grid-based indoor model. *Automation in Construction*, 92:173–187, 2018. 1

[39] Mehmet Yalcinkaya and Vishal Singh. Building information modeling (bim) for facilities management–literature review and future needs. In *Product Lifecycle Management for a Global Market: 11th IFIP WG 5.1 International Conference, PLM 2014, Yokohama, Japan, July 7-9, 2014, Revised Selected Papers 11*, pages 1–10. Springer, 2014. 1

[40] Toshihiko Yamasaki, Jin Zhang, and Yuki Takada. Apartment structure estimation using fully convolutional networks

and graph model. In *Proceedings of the 2018 ACM Workshop on Multimedia for Real Estate Tech*, pages 1–6, 2018. 2

[41] Hongliang Yan, Zhetao Li, Qilong Wang, Peihua Li, Yong Xu, and Wangmeng Zuo. Weighted and class-specific maximum mean discrepancy for unsupervised domain adaptation. *IEEE Transactions on Multimedia*, 22(9):2420–2433, 2019. 3

[42] JongHyeon Yang, Hanme Jang, JiYeup Kim, and JungOk Kim. Semantic segmentation in architectural floor plans for detecting walls and doors. In *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–9. IEEE, 2018. 1

[43] Zhiliang Zeng, Xianzhi Li, Ying Kin Yu, and Chi-Wing Fu. Deep floor plan recognition using a multi-task network with room-boundary-guided attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9096–9104, 2019. 2

[44] Zhaohua Zheng, Jianfang Li, Lingjie Zhu, Honghua Li, Frank Petzold, and Ping Tan. Gat-cadnet: Graph attention network for panoptic symbol spotting in cad drawings. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11747–11756, 2022. 2

[45] Ruiyun Zhu, Jingcheng Shen, Xiangtian Deng, Marcus Walldén, and Fumihiko Ino. Training strategies for cnn-based models to parse complex floor plans. In *Proceedings of the 2020 9th International Conference on Software and Computer Applications*, pages 11–16, 2020. 1

A. Appendix

This appendix contains a table with the class-specific results and more examples of ambiguous class labelling.

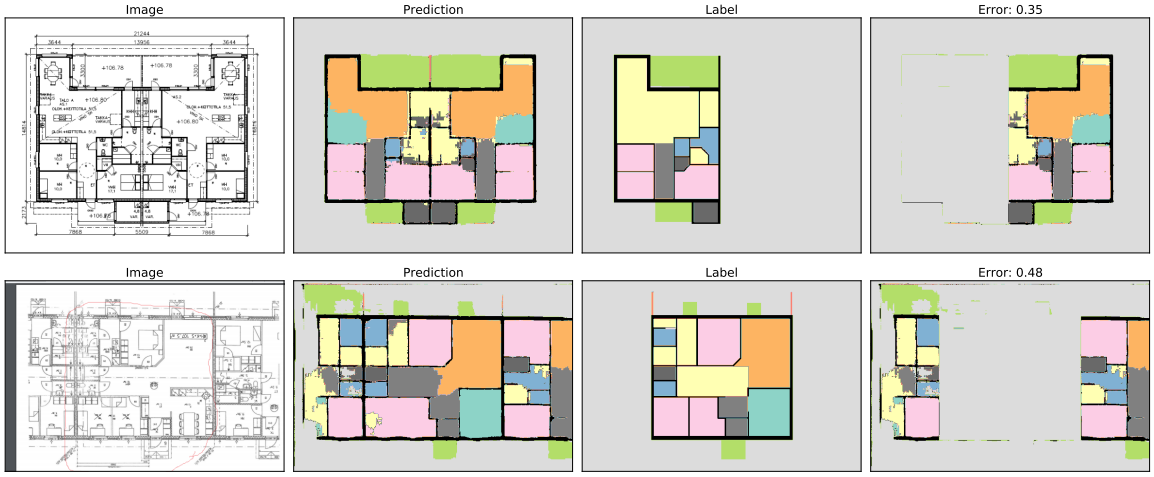
	λ	Rooms												Icons								
		Background (+)	Outdoor (-)	Wall (+)	Kitchen (+)	Living (+)	Bed (+)	Bath (-)	Entry (+)	Railing (-)	Storage (-)	Undefined (-)	**Mean (-)**	No Icon (+)	Window (+)	Door (-)	Closet (-)	Elec. Appl. (+)	Toilet (+)	Sink (-)	Sauna Bench (-)	**Mean (+)**
Constant	0.00	82.1	**49.6**	59.8	**55.9**	**55.3**	75.2	56.6	**56.6**	10.2	32.9	**49.3**	53.0	**97.1**	55.2	49.1	**65.4**	**62.9**	62.3	**52.7**	69.2	64.2
	0.01	**82.4**	48.1	**61.7**	54.3	54.2	**77.0**	58.9	54.0	**10.5**	**37.4**	48.8	**53.4**	97.0	**55.5**	**51.8**	64.2	60.2	**63.4**	52.1	70.7	**64.4**
	0.05	79.7	49.0	59.1	51.9	53.2	72.7	56.6	53.7	10.2	32.9	39.0	50.7	97.0	48.6	49.5	64.4	59.9	62.1	51.5	70.4	62.9
	0.10	80.2	46.8	61.4	50.0	51.8	71.9	**58.9**	56.0	9.1	34.9	45.4	51.5	97.0	55.5	48.0	63.4	60.5	61.8	50.4	69.0	63.2
	0.20	79.4	40.8	61.3	49.8	51.7	67.2	47.6	48.9	9.1	37.0	41.5	48.6	97.0	54.2	50.9	62.5	61.7	57.8	49.2	**72.9**	63.3
	0.30	75.5	38.9	56.9	45.5	47.9	64.5	50.7	45.0	8.1	29.7	35.6	45.3	96.7	45.6	48.4	61.8	56.5	56.1	49.9	67.9	60.4
	0.40	71.4	36.8	49.0	43.1	45.7	59.9	49.5	42.0	3.5	26.2	28.2	41.4	96.6	40.0	47.3	62.8	53.8	51.9	45.1	66.7	58.0
	0.50	53.5	18.7	28.1	29.5	25.5	45.4	37.2	20.0	0.9	19.6	18.7	27.0	95.3	23.3	28.7	38.3	40.2	32.8	27.4	16.3	37.8
Finetuned	0.000	82.1	49.6	59.8	**55.9**	55.3	75.2	56.6	56.6	10.2	32.9	49.3	53.0	97.1	55.2	49.1	**65.4**	62.9	62.3	**52.7**	69.2	64.2
	0.001	81.8	49.5	**62.0**	53.4	56.1	76.6	58.2	54.2	9.7	36.5	50.5	53.5	**97.1**	54.8	**51.8**	64.9	64.2	62.3	52.4	71.1	**64.8**
	0.005	80.9	47.9	60.1	55.0	53.7	73.8	**62.8**	56.1	12.0	**44.2**	50.1	**54.2**	97.0	50.8	49.7	62.9	**64.5**	62.0	49.0	69.4	63.2
	0.010	**82.4**	48.1	61.7	54.3	54.2	77.0	58.9	54.0	10.5	37.4	48.8	53.4	97.0	**55.5**	51.8	64.2	60.2	**63.4**	52.1	70.7	64.4
	0.015	82.0	47.0	59.7	55.8	**56.7**	**77.6**	57.5	54.1	10.7	33.8	**53.3**	53.5	97.0	50.6	47.4	64.0	59.4	60.2	50.6	**73.3**	62.8
	0.020	82.4	**53.4**	59.6	54.5	54.9	76.6	60.6	54.5	11.4	40.1	47.9	54.2	97.0	51.9	49.7	63.6	60.2	62.9	50.9	69.3	63.2
	0.025	82.0	51.4	60.3	53.9	54.5	76.0	59.2	**59.6**	**13.3**	37.6	47.8	54.1	97.0	51.4	48.5	63.7	63.3	63.1	51.0	71.0	63.6
Variable	0.0	82.1	49.6	59.8	**55.9**	**55.3**	**75.2**	56.6	**56.6**	10.2	32.9	**49.3**	**53.0**	97.1	**55.2**	49.1	**65.4**	62.9	62.3	52.7	69.2	**64.2**
	0.1	78.2	49.7	57.0	51.8	53.6	71.8	53.3	53.9	9.9	34.0	42.7	50.5	96.9	49.8	47.2	64.6	61.7	61.1	50.4	67.6	62.4
	0.2	80.2	49.0	60.2	49.6	46.1	71.5	54.8	53.1	11.3	34.2	37.7	49.8	97.0	52.2	**49.1**	64.3	61.2	60.3	**52.9**	68.5	63.2
	0.3	78.7	47.5	57.6	49.3	50.7	74.3	49.4	52.3	7.8	37.0	41.6	49.7	97.0	49.4	48.4	64.4	61.4	61.8	49.5	71.1	62.9
	0.4	**82.4**	**51.2**	**60.6**	50.7	53.9	72.9	**58.3**	52.8	**11.6**	**38.6**	39.9	52.1	**97.1**	54.1	47.4	64.9	**62.9**	**62.5**	52.4	69.6	63.9
	0.5	77.2	42.0	58.2	50.8	49.4	68.3	52.8	53.4	8.1	36.0	37.8	48.5	96.9	48.4	47.9	63.7	59.5	58.8	51.0	**71.9**	62.3

(a) Lambda experiment results in which the first column represents the lambda λ used in the experiments, for the variable λ this value is scaled by epochs of which the formula can be found in formula 5.4. Constant experiment shows collapse for larger λ, but promising smaller λ. The fine-tune experiment shows that the model is inconsistent for small λ changes. The variable experiment shows promising results as mostly classes with an adaptation gap improve with larger lambda.

			Rooms												Icons								
λ	C	S	Background (+)	Outdoor (-)	Wall (+)	Kitchen (+)	Living (+)	Bed (+)	Bath (-)	Entry (+)	Railing (-)	Storage (-)	Undefined (-)	**Mean (-)**	No Icon (+)	Window (+)	Door (-)	Closet (-)	Elec. Appl. (+)	Toilet (+)	Sink (-)	Sauna Bench (-)	**Mean (+)**
0.01	256	1	82.2	**52.1**	60.0	52.8	52.9	76.1	57.6	52.0	**11.7**	39.7	41.6	52.6	97.0	52.7	49.6	65.7	**62.3**	62.4	51.1	70.6	63.9
		2	82.1	48.7	59.3	54.3	51.8	72.1	**60.9**	54.0	10.9	33.5	45.2	52.1	97.0	49.8	48.2	64.9	60.8	63.4	51.4	70.4	63.2
		4	80.2	48.4	60.2	51.2	52.6	73.1	58.8	53.5	7.1	32.4	42.2	50.9	97.0	55.0	49.1	63.2	59.2	61.3	52.0	70.3	63.4
	512	1	82.4	48.1	**61.7**	54.3	**54.2**	**77.0**	58.9	54.0	10.5	37.4	48.8	**53.4**	**97.0**	**55.5**	**51.8**	64.2	60.2	**63.4**	52.1	**70.7**	**64.4**
		2	**83.4**	49.4	60.1	52.1	53.9	72.9	51.4	50.1	9.1	**40.2**	**50.5**	52.1	97.0	51.9	48.1	**65.8**	61.8	61.5	**53.7**	69.0	63.6
		4	81.1	50.0	58.8	**56.9**	53.6	75.5	58.5	**55.1**	9.4	35.2	49.4	53.0	96.9	50.9	49.7	62.0	61.1	61.7	52.3	67.9	62.8
0.1	256	1	79.0	46.2	60.9	50.3	51.8	73.7	53.6	54.3	**11.8**	**37.6**	38.9	50.7	**97.0**	54.5	48.4	64.3	62.5	60.5	**52.6**	68.8	63.6
		2	80.9	**50.8**	60.1	**54.6**	51.7	72.6	**60.4**	**58.3**	10.2	35.4	42.8	**52.5**	97.0	52.8	**49.8**	61.0	**64.0**	**63.0**	52.0	**70.1**	**63.7**
		4	77.9	46.7	57.9	49.9	53.6	70.3	53.0	51.3	7.7	32.2	41.4	49.3	96.9	47.7	48.1	63.1	57.5	58.2	49.5	69.2	61.3
	512	1	80.2	46.8	**61.4**	50.0	51.8	71.9	58.9	56.0	9.1	34.9	**45.4**	51.5	97.0	**55.5**	48.0	63.4	60.5	61.8	50.4	69.0	63.2
		2	78.8	50.1	58.5	49.1	52.4	**76.0**	59.1	51.2	11.3	35.5	42.7	51.3	97.0	51.6	47.6	62.4	63.9	60.2	52.1	**71.4**	63.3
		4	**81.5**	46.7	60.7	52.7	**55.5**	73.7	51.9	52.5	5.3	34.4	38.3	50.3	97.0	52.9	48.6	**65.4**	58.3	61.8	51.7	69.1	63.1

(b) Latent space experiment results in which the first columns represent the lambda λ, extraction channels C, and the sizing S of the adaptive max pooling size $S \times S$. We take as base the $\lambda = -.1$, reducing the chance of influence by random sampling, in these we observe that a balance pooling of 2 is beneficial for our classes.

Table 5. Results of experiments ordered by class type, either rooms or icons, with its respective classes. The numbers displayed are the vectorization intersection over union (IoU) scores with the mean of the class type (mIoU) in the last column. The minus or plus above each class indicates the adaptation gap, with + showing improvement and - indicating a gap.



(a) High quality architectural domain samples

(b) High quality domain samples

(c) Colorful domain samples

Figure 10. Instances of the domains showing the problems occurring in the dataset. The samples also contain the three types of structural cutoffs occurring in the dataset where either only part of the floor plan, only the walls, or only a selected area is labeled. Some also clearly exhibit the ambiguous undefined labeling.

3

Background

3.1. Machine Learning

3.1.1. Introduction

Machine learning (ML) is a branch of artificial intelligence (AI) that focuses on developing algorithms that allow computers to learn from data and make predictions. The field has grown significantly since its beginnings in the mid-20th century, thanks to advances in computational power, access to large datasets, and new algorithm development. Early ML models relied on rule-based systems, but the introduction of statistical methods in the 1980s and 1990s led to more advanced models. Recently, the rise of deep learning has transformed the field, leading to significant improvements in tasks like image and speech recognition, natural language processing, and autonomous driving.

A key element of machine learning is the use of features (x), which are specific, measurable characteristics of the data being analyzed. For example, in a dataset about house prices, features might include the size of the house, the number of bedrooms, and the location. Selecting and engineering relevant features are crucial steps in machine learning because they directly affect how well the model performs.

Labels (y) are the target values or outputs that the ML model aims to predict. In supervised learning, each data point has an associated label. For instance, in a house price prediction dataset, the label would be the actual price of the house. Labels provide the correct answers that the model uses during training.

To train a model, its performance must be measured using loss functions and evaluation metrics. The loss function measures how closely the model's predictions ($\hat{y}$) match the actual labels (y) during training. Common loss functions include mean squared error (MSE) for regression tasks and cross-entropy loss for classification tasks. Evaluation metrics like accuracy, precision, recall, and F1-score are used to assess the model's performance on new data and help choose the best model.

One of the simplest machine learning algorithms is linear classification, where the goal is to separate data into different classes using a straight-line decision boundary. This can be represented mathematically as $y = ax + b$, where a and b are parameters learned from the data. Examples include logistic regression and support vector machines (SVMs) with a linear kernel. While linear classifiers are simple and efficient, they assume a linear relationship between the features (x) and the target (y), which might not be true in complex real-world scenarios. For instance, in complex data like floor plan images, the data is often not linearly separable, leading to underfitting, where the model does not capture the underlying patterns, resulting in poor performance [5].

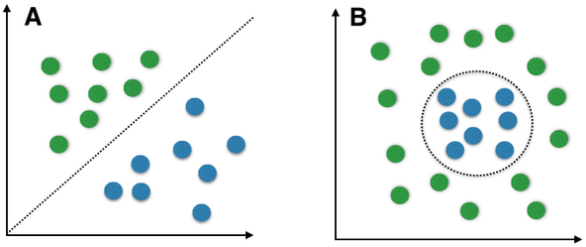


Figure 3.1: Example of A linear separable and B non-separable data

Building on the discussion of underfitting, it's important to address the common issues of overfitting and underfitting in machine learning. Overfitting occurs when a model learns the noise and specific details of the training data too well, which harms its ability to perform well on new data. This usually happens when the model is too complex for the amount of training data available. Underfitting, on the other hand, occurs when a model is too simple to capture the true patterns in the data, leading to poor performance on both the training data and new data. Balancing the model's complexity and ensuring enough training data are key to avoiding these problems. Techniques like cross-validation, regularization, and pruning can help reduce the risks of overfitting and underfitting.

3.1.2. Deep Learning

Deep learning is a specialized area within machine learning that uses neural networks with multiple layers to learn data representations at increasing levels of abstraction. A fundamental architecture in

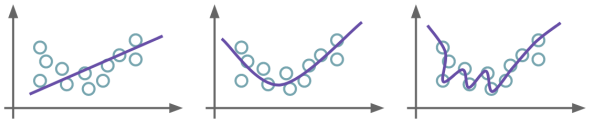


Figure 3.2: Example of **left:** underfitting, **middle:** appropriate fitting, and **right:** overfitting

deep learning is the Multi-Layer Perceptron (MLP) [16], which consists of several layers of interconnected neurons. Each neuron acts as a computational unit that processes input data. When a neuron receives an input vector $\mathbf{x}$, it computes a weighted sum, mathematically expressed as $z = \mathbf{w} \cdot \mathbf{x} + b$, where $\mathbf{w}$ is the weight vector and b is the bias term. The neuron's output is then transformed using a non-linear activation function $f(z)$, such as sigmoid, tanh, or leaky ReLU. Non-linear activation functions are essential because they enable MLPs to approximate complex functions and model intricate relationships in the data.

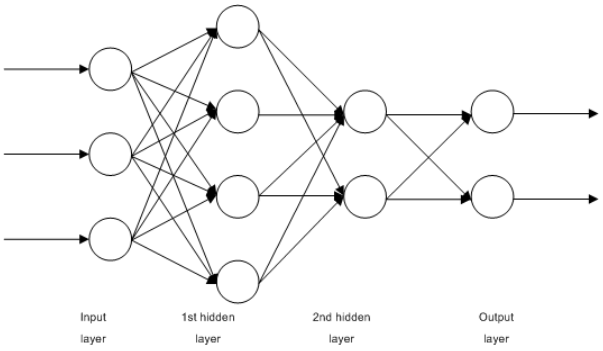


Figure 3.3: Example of a multi-layer perceptron

An MLP architecture consists of an input layer, one or more hidden layers, and an output layer. Each hidden layer contains multiple neurons, allowing the model to learn more complex features at each layer. The choice of activation function plays a crucial role in the MLP's performance. The sigmoid function produces values between 0 and 1, making it ideal for binary classification tasks. The tanh function outputs values from -1 to 1, which centres the data around zero and can help the model converge faster. The leaky ReLU activation function allows for a small, non-zero gradient when the input is negative, which helps prevent the vanishing gradient problem often encountered with traditional activation functions.

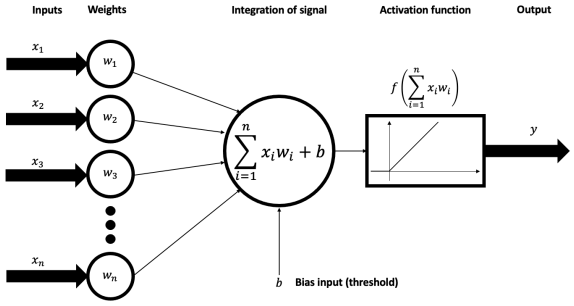


Figure 3.4: Example of a neuron with its inputs, weights, bias, summation, activation function, and output.

MLPs can be seen as an extension of linear regression, which models the relationship between input features $\mathbf{x}$ and a continuous output y using the equation $y = \mathbf{w} \cdot \mathbf{x} + b$. While linear regression works well for simple linear relationships, it often underfits when dealing with complex data that cannot be separated linearly. MLPs, with their multiple layers and non-linear activation functions, can model complex relationships. This allows MLPs to learn advanced features from input data, which is essential for tasks like image recognition.

Deep learning includes three main learning paradigms: supervised learning, unsupervised learning, and reinforcement learning.

- **Supervised learning** trains a model on labelled data, where each example has input features $\mathbf{x}$ and corresponding output labels y. The goal is to learn a function $f(\mathbf{x})$ that can accurately predict the output based on the input. This approach is commonly used in tasks like image classification, where the model learns to assign images to predefined categories.
- **Unsupervised learning** deals with unlabeled data. The model identifies patterns or structures within the data without being told the correct output. Techniques like clustering and dimensionality reduction are used in this paradigm. For instance, in image data, unsupervised learning can be applied to image segmentation, where the model identifies distinct regions in an image without predefined labels.
- **Reinforcement learning** trains an agent to make decisions by interacting with an environment. The agent learns to take actions that maximize a reward over time. This approach is commonly used in robotics, game-playing, and autonomous systems, where the agent learns optimal strategies through trial and error.

The training process of an MLP involves optimizing a loss function that measures the difference between the predicted output $\hat{y}$ and the actual output y. Gradient descent is a common optimization technique used to minimize this loss. The key idea of gradient descent is to calculate the gradient of the loss function with respect to each weight and then update the weights in a way that reduces the loss. The update rule is mathematically expressed as:

$$\theta := \theta - \eta \nabla L \tag{3.1}$$

where θ represents the model parameters (weights), η is the learning rate, and ∇L is the gradient of the loss function.

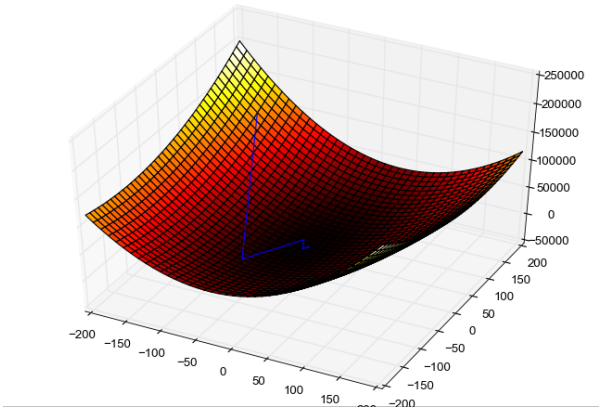


Figure 3.5: Gradient descent example in a 2D space, the height and color represent the loss.

Several variations of gradient descent exist, each with its advantages. One of the most popular optimization algorithms is Adam (Adaptive Moment Estimation). Adam combines the benefits of two other methods, AdaGrad and RMSProp, by calculating adaptive learning rates for each parameter based on the first and second moments of the gradients. Specifically, Adam maintains two moving averages: the first moment (the mean of the gradients) and the second moment (the uncentered variance of the gradients). The updates for the weights in Adam are given by:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.2}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3.3}$$

$$\theta := \theta - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t \tag{3.4}$$

Here, g_t is the gradient at time step t, m_t is the first-moment estimate, v_t is the second-moment estimate, β_1 and β_2 control how quickly the moving averages adjust, and ϵ is a small constant to prevent division by zero. This adaptive learning rate helps Adam converge more quickly and effectively, especially in situations with sparse gradients or noisy data.

In the training process, regularization techniques are used in deep learning to prevent overfitting, which happens when a model performs well on training data but fails to generalize to new, unseen data. Common regularization methods include L1 and L2 regularization, dropout, and early stopping. These techniques help improve the model's ability to generalize by limiting its complexity [15].

Alongside regularization, hyperparameter tuning is essential for optimizing model performance and ensuring generalization. Hyperparameter tuning involves selecting the best fixed values, like learning rate (η), batch size, and network architecture, that influence the training process. The goal is to find the combination that minimizes the loss function on a validation set, thereby improving the model's performance on new data. Techniques for hyperparameter tuning include grid search, random search, and Bayesian optimization, each aiming to efficiently explore the hyperparameter space to find the best configuration. Effective tuning is crucial for maximizing the performance of deep learning models.

Beyond MLPs, various deep learning architectures exist, including Convolutional Neural Networks (CNNs), which are particularly effective for image processing tasks, Recurrent Neural Networks (RNNs), which excel at handling sequence data like time series or natural language, and Generative Adversarial Networks (GANs), which are used for generating new data samples. Each architecture is designed to address specific types of problems, highlighting the versatility and power of deep learning [11].

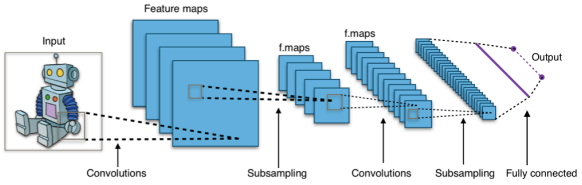


Figure 3.6: Example of convolutional neural network

Despite its success, deep learning faces several challenges and limitations. It requires large labelled datasets, which can be difficult and expensive to obtain. The models are computationally expensive, needing significant hardware resources for both training and inference. They can also overfit when trained on small datasets or using complex architectures. Additionally, deep learning models often lack interpretability, making it hard to understand their decisions. Training can encounter issues such as getting stuck in local minima or dealing with vanishing or exploding gradients, which complicate the optimization process.

3.1.3. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of deep learning model designed for processing data with a grid-like structure, such as images. They excel at automatically learning and extracting important features, making them particularly useful for visual tasks. The core component of a CNN is the convolutional layer, which applies a set of filters or kernels to the input data. Each filter is a small matrix that slides over the input image, producing a feature map that highlights local patterns like edges and textures. This convolution operation can be described mathematically as follows:

$$Z[i,j] = (X * W)[i,j] = \sum_m \sum_n X[i+m, j+n] W[m,n] \tag{3.5}$$

In this equation, $Z[i,j]$ represents the value of the feature map at position (i,j), X is the input image, and W is the filter. The indices m and n denote the rows and columns of the filter. As the filter moves across the image, it focuses on small regions, allowing the network to learn features that are crucial for understanding the image.

To illustrate with a 1D example, consider an input vector $X = [x_1, x_2, x_3, x_4]$ and a filter $W = [w_1, w_2]$.

The convolution operation would involve sliding the filter across the input vector to produce an output vector Z. Mathematically, this is represented as:

$$Z[1] = x_1 \cdot w_1 + x_2 \cdot w_2 \tag{3.6}$$

$$Z[2] = x_2 \cdot w_1 + x_3 \cdot w_2 \tag{3.7}$$

$$Z[3] = x_3 \cdot w_1 + x_4 \cdot w_2 \tag{3.8}$$

In this example, Z is the resulting feature map, highlighting patterns in the input vector. The 1D convolution operation works similarly to the 2D case but processes data in one dimension, which is often used for tasks like text or time series analysis.

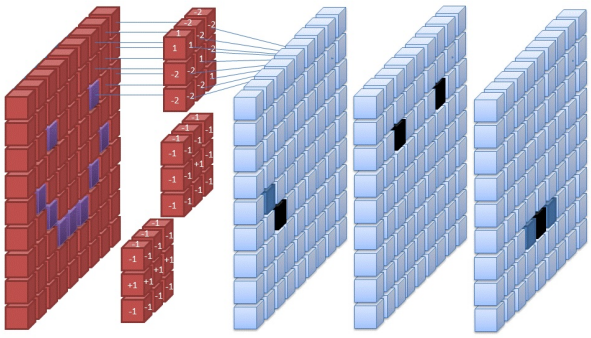


Figure 3.7: Example of a filter sliding over an input image, resulting in a new feature map.

The output size of the feature map generated by a convolutional layer depends on several factors, including the input size, filter size, stride, and padding. The stride determines how far the filter moves across the input at each step. A stride of 1 means the filter shifts by one pixel at a time, while a stride of 2 means it jumps two pixels. Larger strides generally produce smaller feature maps, which can reduce computational load while still capturing essential features.

Padding involves adding extra pixels around the input image to ensure that the filters can process edge pixels. There are two common types of padding:

1. **Same Padding**: This method adds zeros around the input image to keep the output size the same as the input size after the convolution. It ensures that the convolutional layer does not reduce the spatial dimensions of the feature map.
2. **Valid/No padding**: This method does not add any extra pixels to the input image. As a result, the output feature map is smaller than the input. This approach is useful when the goal is to focus the convolution operation solely on the actual data without any additional padding.
3. **Full Padding**: This method adds enough padding to the input image so that the output feature map becomes larger than the input. This type of padding allows the filter to slide over all possible positions, including those where the filter extends beyond the original image boundary. Full padding is less commonly used but can be useful in certain contexts where preserving more spatial information is important.

In PyTorch, these concepts are implemented through the ‘Conv2d‘ class, which lets you create a 2D convolutional layer with customizable parameters like the number of input channels, the number of output channels, kernel size, stride, and padding. This flexibility allows the model to adapt the convolutional layer to fit the specific needs of the input data.

After the convolutional layers, pooling layers are used to down-sample the feature maps, reducing their size while keeping the most important information. The two main types of pooling are max pooling and

average pooling. Max pooling selects the maximum value from a defined window of the feature map, helping the network focus on the strongest features. This operation can be expressed as:

$$P[i,j] = \max_{m,n} Z[i+m, j+n] \tag{3.9}$$

Here, $P[i,j]$ represents the value of the pooled feature map at position (i,j). For instance, consider a 1D feature map $Z = [3,5,2,8,6]$ and a max pooling window of size 2. The operation would yield a pooled output of $P = [5,8,8]$, selecting the highest value from each pair. Average pooling, on the other hand, calculates the average of the values within the pooling window, helping to smooth the feature maps and retain more general information. This operation can be expressed as:

$$P[i,j] = \frac{1}{k} \sum_{m,n} Z[i+m, j+n] \tag{3.10}$$

where k is the total number of elements in the pooling window. Using the same 1D example, with an average pooling window of size 2, the operation would yield $P = [4, 3.5, 7]$, where each value is the average of the two corresponding values in the feature map.

Adaptive pooling is a type of pooling that adjusts the pooling window size to produce a fixed output size, regardless of the input dimensions. This is especially useful for handling inputs of varying sizes. In PyTorch, adaptive pooling can be implemented through layers that allow you to specify the desired output size, making it easy to integrate into different architectures.

After the pooling layers, CNNs usually include one or more fully connected layers. In these layers, every neuron is connected to all neurons in the previous layer, enabling the model to learn complex combinations of features extracted from earlier layers. The output of a fully connected layer can be represented as:

$$\hat{y} = \sigma(W \cdot h + b) \tag{3.11}$$

Here, $\hat{y}$ is the predicted output, W is the weight matrix, h is the input vector from the previous layer, b is the bias vector, and σ is the activation function that adds non-linearity to the model. In practice, these fully connected layers are also implemented in PyTorch, providing an easy way to build complex models that leverage the features learned in previous layers.

The main advantage of CNNs is their ability to learn relevant features directly from raw input data, reducing the need for manual feature extraction. This capability is particularly valuable in image classification tasks, where CNNs have demonstrated excellent performance across various datasets, accurately categorizing images based on learned features. Their structure allows them to identify complex patterns from pixel data, making them suitable for a wide range of applications.

Beyond image classification, CNNs are widely used in object detection, with models like YOLO (You Only Look Once) and Faster R-CNN excelling at detecting and localizing multiple objects within an image. This is crucial in areas like autonomous driving and security, where fast and accurate processing is essential. CNNs are also applied in image segmentation, which involves classifying each pixel in an image.

3.2. Domain Adaptation

3.2.1. Introduction

Domain adaptation is a part of machine learning that deals with transferring knowledge from one domain (source) to another (target). This is important when the data distribution in the target domain differs from the source domain. Domain adaptation helps models trained on one dataset to perform well on a different but related dataset, which is crucial for many applications, including the analysis of floor plans.

In the context of floor plans, domain adaptation can improve the performance of models designed to interpret architectural layouts. For example, a model trained on modern, digitally-created floor plans (source domain) might not perform well on hand-drawn historical floor plans (target domain) due to differences in style, detail, and representation. By applying domain adaptation, the model can handle these variations without extensive retraining.

An example of domain shift in floor plans could be the difference in line thickness and clarity between CAD-generated floor plans and hand-drawn ones. CAD-generated floor plans usually have consistent, clean lines, while hand-drawn floor plans may have irregular, variable-thickness lines and annotations. This difference can cause a significant performance drop in models that are not adapted to handle such variations.

3.2.2. Domain adaptation settings

Unsupervised domain adaptation is used when there are no labelled examples in the target domain. The model relies solely on the labelled data from the source domain and the unlabeled data from the target domain to adapt. Formally, we have a source domain with known probabilities $P(X_s, Y_s)$ and a target domain with known probabilities $P(X_t)$. In the context of floor plans, this relates to adapting a model trained on labelled digital floor plans to work with a set of unlabeled hand-drawn floor plans. This method is useful in scenarios where acquiring labelled data for the target domain is costly or impractical.

Weakly supervised domain adaptation uses weak labels in the target domain. Weak labels provide limited information, such as the presence of categories in a sample but not precise details like class segmentation masks. In this case, we can represent the source domain as $P(X_s, Y_s)$ and the target domain as $P(X_t, Y_{\text{weak}})$, where Y_{weak} contains only category information without specific segmentation. For example, a model trained on digital floor plans with detailed annotations might adapt using hand-drawn floor plans where only the types of rooms (e.g., kitchen, bedroom) are known without their exact locations. This approach can help in scenarios where obtaining complete annotations is challenging, bridging the gap between fully labelled and unlabeled data.

Semi-supervised domain adaptation uses strong labels in the target domain, but only a small portion of the data is labelled. Formally, the source domain remains $P(X_s, Y_s)$, while the target domain consists of very limited samples with labels $P(X_t, Y_{\text{small}})$ and possibly others with only the samples $P(X_t)$. In floor plans, this could mean using many labelled digital floor plans, a few hand-drawn floor plans with detailed annotations, and possibly combining that with some hand-drawn floor plans with no annotations. This technique allows models to leverage a small amount of reliable data while still benefiting from a larger volume of unlabeled data, enhancing generalization to the target domain.

Supervised domain adaptation occurs when there is a substantial amount of labelled data available in both the source and target domains. This approach allows for the most accurate adaptation since the model can learn directly from the labelled data in the target domain. Formally, both the source and target domains can be expressed as $P(X_s, Y_s)$ and $P(X_t, Y_t)$, where both domains have known probabilities for their respective input-output pairs. For floor plans, this would involve training a model on a large set of labelled digital floor plans and a large set of labelled hand-drawn floor plans, ensuring high performance on both types. This method is optimal when sufficient labelled data exists in both domains.

3.2.3. Types of Domain Adaptation

Domain adaptation involves transferring knowledge from a source domain to a target domain with different data distributions. There are several types of domain adaptation [4]:

Closed Set Domain Adaptation: The source and target domains have the same classes but different

data distributions. The main challenge is aligning these distributions.

Open Set Domain Adaptation: The source and target domains share some classes, but the source domain has additional classes not present in the target domain. The goal is to adapt to the shared classes and ignore the source-only classes.

Partial Domain Adaptation: The target domain is a subset of the source domain, containing fewer classes. The focus is on adapting to the relevant classes in the target domain.

Universal Domain Adaptation: This type covers scenarios where the overlap between the source and target label sets is unknown. It aims to adapt across domains without prior knowledge of their relationship.

3.2.4. Techniques Used in Domain Adaptation

Different techniques are used to address the challenges of domain adaptation, categorized into shallow and deep methods [4]:

Instance-Based Adaptation: This technique re-weights instances from the source domain to align with the target domain. Methods like Kernel Mean Matching (KMM) and Kullback-Leibler Importance Estimation Procedure (KLIEP) are used to estimate and adjust the differences between the domains.

Feature-Based Adaptation: These methods find a common feature space where the distributions of the source and target domains are aligned. Techniques like Principal Component Analysis (PCA) and Transfer Component Analysis (TCA) are used to transform the original feature space.

Deep Domain Adaptation: Deep neural networks are used to learn domain-invariant features. Techniques include discrepancy-based methods, such as Maximum Mean Discrepancy (MMD), and adversarial-based methods like Domain-Adversarial Neural Networks (DANN), which aim to make the features from both domains indistinguishable.

3.2.5. Maximum Mean Discrepancy

We now focus on closet set unsupervised deep domain adaptation, which is the task we aim to research. Formally, let $\mathcal{D}_S = \{(\mathbf{x}_i^S, y_i^S)\}_{i=1}^{n_S}$ represent the labelled data from the source domain, where $\mathbf{x}_i^S \in \mathcal{X}_S$ and $y_i^S \in \mathcal{Y}_S$. Similarly, let $\mathcal{D}_T = \{\mathbf{x}_j^T\}_{j=1}^{n_T}$ represent the data from the target domain, where $\mathbf{x}_j^T \in \mathcal{X}_T$, but the labels y_j^T are unknown. The challenge of domain adaptation arises from the distributional shift between the source and target domains, typically expressed as $P_S(\mathbf{x}, y) \neq P_T(\mathbf{x}, y)$.

In unsupervised domain adaptation, we do not have access to labelled data in the target domain ($\mathcal{Y}_T$ is unknown). Our objective is to train a model that performs well on the target domain by leveraging the labelled source data and aligning the distributions between the source and target domains.

One approach to unsupervised domain adaptation is to align the feature distributions of the source and target domains within a shared feature space. Formally, let $\phi : \mathcal{X} \to H$ represent a feature mapping from the input space $\mathcal{X}$ to a shared feature space $\mathcal{H}$. The goal is to minimize the discrepancy between the distributions $P_S(\phi(\mathbf{x}^S))$ and $P_T(\phi(\mathbf{x}^T))$ in this feature space, ensuring that the model learns features that are invariant across domains.

One effective way to align features in machine learning is by using a Reproducing Kernel Hilbert Space (RKHS). In simple terms, RKHS is a special mathematical space where we can work with functions and compare them efficiently using something called a kernel function.

Imagine you have a function that predicts house prices based on features like size, location, and number of bedrooms. In RKHS, we can represent and compare such functions without directly dealing with the complexity of all these features.

The key idea in RKHS is the kernel function. A kernel function, $k(\mathbf{x}, \mathbf{x}')$, is a tool that lets us measure how similar two inputs $\mathbf{x}$ and $\mathbf{x}'$ are without explicitly mapping these inputs into a high-dimensional feature space. Formally, if we have a function f in RKHS, we can express it as an inner product $f(\mathbf{x}) = \langle f, \phi(\mathbf{x}) \rangle_{\mathcal{H}}$, where $\phi(\mathbf{x})$ is a feature mapping from the input space $\mathcal{X}$ to the RKHS $\mathcal{H}$. The kernel function corresponds to the inner product in this space, so $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}}$ [6].

Think of the kernel function as a quick way to compare two things (like two houses with different features) without listing out every detail. For example, if $\mathbf{x}$ and $\mathbf{x}'$ are two different houses, the kernel function $k(\mathbf{x}, \mathbf{x}')$ tells us how similar they are by calculating this inner product.

This property of RKHS, called the reproducing property, allows us to evaluate functions and compare inputs efficiently, making it a powerful tool in machine learning. It helps in aligning and comparing data from different sources, which is particularly important in tasks like domain adaptation, where we need to align features from different domains.

To measure the discrepancy between the source and target distributions in the RKHS, we use the Maximum Mean Discrepancy (MMD) [6]. The MMD is a distance metric between distributions P_S and P_T based on the difference between their mean embeddings in $\mathcal{H}$. Formally, the MMD between two distributions P_S and P_T is defined as:

$$\text{MMD}(\mathcal{D}_S, \mathcal{D}_T) = \left\| \frac{1}{n_S} \sum_{i=1}^{n_S} \phi(\mathbf{x}_i^S) - \frac{1}{n_T} \sum_{j=1}^{n_T} \phi(\mathbf{x}_j^T) \right\|_{\mathcal{H}} \tag{3.12}$$

Expanding this formula, the squared MMD can be expressed as:

$$\text{MMD}^2(\mathcal{D}_S, \mathcal{D}_T) = \frac{1}{n_S^2} \sum_{i=1}^{n_S} \sum_{i'=1}^{n_S} k(\mathbf{x}_i^S, \mathbf{x}_{i'}^S) + \frac{1}{n_T^2} \sum_{j=1}^{n_T} \sum_{j'=1}^{n_T} k(\mathbf{x}_j^T, \mathbf{x}_{j'}^T) - \frac{2}{n_S n_T} \sum_{i=1}^{n_S} \sum_{j=1}^{n_T} k(\mathbf{x}_i^S, \mathbf{x}_j^T) \tag{3.13}$$

Here, the first term captures the intra-domain discrepancy within the source domain, the second term captures the intra-domain discrepancy within the target domain, and the third term captures the inter-domain discrepancy between the source and target domains. The goal is to minimize the MMD, thereby reducing the discrepancy between the source and target distributions.

In the context of machine learning, this alignment is achieved by minimizing the MMD as a loss function, alongside the primary task loss. For instance, in deep learning models, the MMD loss is computed at a specific layer in the network, typically in the latent space. The model's parameters, including the network weights, are adjusted through backpropagation to minimize the combined loss

Formally, let $\mathcal{L}_{\text{task}}$ represent the task-specific loss (e.g., segmentation loss), and let $\mathcal{L}_{\text{MMD}}$ represent the MMD loss. The total loss is given by:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \lambda \mathcal{L}_{\text{MMD}} \tag{3.14}$$

where λ is a hyperparameter that controls the trade-off between task performance and domain alignment. By optimizing $\mathcal{L}_{\text{total}}$, the model learns features that are invariant across the source and target domains, ensuring that it performs well on the target domain.

Different kernel functions can be used for the MMD calculation. A commonly used kernel, also employed by our research, is the Radial Basis Function (RBF) kernel, defined as:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2} \right) \tag{3.15}$$

where σ is a bandwidth parameter that determines how quickly the kernel function decreases as the distance between $\mathbf{x}$ and $\mathbf{x}'$ increases. A small bandwidth σ causes the kernel to decay rapidly, meaning only data points that are very close will be considered similar, while distant points will have a kernel value close to zero. This makes the Maximum Mean Discrepancy (MMD) highly sensitive to local variations, which could overemphasize minor differences between the source and target distributions. On the other hand, a large σ leads to a slower decay, making the kernel less sensitive to small differences and more focused on the overall structure of the distributions.

The choice of bandwidth σ significantly impacts the MMD. A small σ might highlight fine-grained discrepancies but could also introduce noise by reacting to minor variations. Conversely, a large σ may smooth out these details, potentially overlooking important differences between the distributions. This sensitivity to σ underscores the importance of selecting an appropriate bandwidth. Alternatively, a Multi-Kernel Maximum Mean Discrepancy (MK-MMD) can be used [14], which combines multiple kernels with different bandwidths. This approach balances sensitivity and robustness, providing a more reliable measure of distributional alignment in domain adaptation tasks. Formally, MK-MMD is defined as:

$$\text{MK-MMD}(\mathcal{D}_S, \mathcal{D}_T) = \sum_{l=1}^{L} \beta_l \text{MMD}(\mathcal{D}_S, \mathcal{D}_T; k_l) \tag{3.16}$$

where β_l are weights assigned to each kernel k_l, and L is the number of kernels used. The combination of multiple kernels helps to capture discrepancies at different scales, improving the robustness of the domain adaptation process.

3.3. Floor plan vectorization

3.3.1. Introduction

Floor plan vectorization is the process of converting raster images of floor plans into vector-based representations. While raster images consist of a grid of pixels, vector representations use mathematical descriptions to define elements like walls, doors, and rooms. This allows for easier scaling, modification, and detailed analysis without losing quality or precision. This process is different from methods like those used in [12], where floor plans are reconstructed from 3D point cloud scans—a related but separate topic.

In this section, we will explore the key parts of floor plan vectorization. We will look at the data augmentation methods used to improve the robustness and generalization of models involved in the vectorization process. The structure of the CubiCasa model, a deep learning framework created specifically for this task, will be explained.

We will also discuss the floor plan representation, which organizes different elements of a floor plan into separate channels. The method of multi-task learning with uncertainty modelling, which is used to learn both segmentation and heatmap regression tasks, will be covered. Finally, we will explain the post-processing technique that combines the segmentation masks and junctions to produce a vectorized floor plan.

3.3.2. Data augmentations

In the data augmentation pipeline for our architectural floor plan analysis, several transformations were implemented to increase the variability of the training dataset. The primary purpose of data augmentation is to expand the training set artificially by applying transformations that simulate variations likely to occur in real-world data, thereby improving the model's ability to generalize.

A key transformation applied is rotation. Rotation augmentation ensures that the model can accurately interpret floor plans regardless of orientation. Specifically, images were rotated by angles of $0°$, $90°$, $180°$, and $270°$. This process preserves the spatial relationships within the image, ensuring that the model learns to recognize architectural features from multiple orientations. The corresponding labels, including segmentation masks and key points, were also rotated to maintain alignment with the image data. The use of rotation for data augmentation is well-established in computer vision as it allows the model to learn rotational invariance [20].

Another transformation used is cropping and padding. Random cropping was applied to simulate scenarios where only parts of the floor plan are visible, forcing the model to interpret partial views of the data. After cropping, images were padded to restore them to a consistent size, ensuring uniformity across the dataset. The corresponding labels were adjusted to reflect their new positions within the cropped and padded images, maintaining the correct spatial relationships. Cropping and padding are standard techniques in image preprocessing that enhance the model's robustness to different viewpoints and scales [19].

Intensity transformations, including adjustments to brightness, contrast, and sharpness, were applied to account for variations in image quality and lighting conditions. These transformations simulate different conditions under which floor plans might be captured, ensuring the model is not overly sensitive to specific image quality or lighting scenarios. Instead, the model focuses on the structural and layout features of the floor plans. The importance of intensity transformations in enhancing model generalization has been widely recognized in recent literature, as they contribute to the robustness of models against varying conditions in input data [10].

3.3.3. Cubicasa Model Architecture

The network architecture employed for this study is derived from ResNet-152, which was initially introduced by He et al. [7] and pre-trained on the ImageNet dataset [3]. The ResNet-152 framework is well-established for its residual learning approach that facilitates the training of deep networks. For this study, the architecture was further refined by first training on ImageNet [18] and then on the MPII Human Pose Dataset [1] to adapt it to specific segmentation tasks.

A similar segmentation approach, leveraging an encoder-decoder architecture, was initially presented

by Ronneberger et al. [17], which laid the foundation for subsequent adaptations in segmentation models. In adapting the ResNet-152 architecture, modifications were made to suit the problem requirements. Specifically, the initial convolutional layer was adjusted to accommodate a change in input channels from 19 to 3. Additionally, the final two layers of the network were replaced to fit the required number of output channels necessary for generating two segmentation maps and 21 heatmaps. Due to these modifications, these layers were initialized randomly to ensure that the network parameters were suitable for the specific segmentation tasks addressed in this study.

The encoder path consists of convolutional layers and residual blocks that progressively downsample the input image, capturing hierarchical features. The decoder path upsamples these features to produce the final output for tasks such as segmentation or reconstruction.

The encoder begins with a convolutional layer that processes the input image, followed by batch normalization and a ReLU activation function. This initial layer outputs feature maps with reduced spatial dimensions. Subsequent layers include residual blocks, which apply convolutions, batch normalization, and ReLU activations. Residual blocks help the network learn residual functions, facilitating deeper network training and addressing the vanishing gradient problem.

The encoder uses max pooling operations to progressively downsample feature maps. Each residual block's output is fed into a max pooling layer, reducing spatial dimensions by a factor of 2. This downsampling captures high-level features and spatial hierarchies.

Residual blocks are crucial to the CubiCasa model. Introduced by He et al. (2016) in "Deep Residual Learning for Image Recognition" [7], residual blocks use skip connections to simplify the training of deep networks.

Let $\mathbf{x}$ represent the input to the residual block, and $\mathcal{F}(\mathbf{x})$ denote the function learned by the block's internal layers. The output of the residual block, $\mathbf{y}$, is given by:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}) + \mathbf{x}$$

This equation shows that the block learns the residual function $\mathcal{F}(\mathbf{x})$ and adds it to the original input $\mathbf{x}$. The goal is to learn the residual mapping rather than the complete transformation.

The residual block includes convolutional layers applied to the input $\mathbf{x}$. These layers consist of 1x1 convolutions for adjusting channel dimensions and 3x3 convolutions for spatial features. Each convolution is followed by batch normalization and ReLU activation functions. After these operations, the transformed feature maps are combined with the input through a skip connection.

The skip connection directly adds $\mathbf{x}$ to the output of the convolutional layers. If the dimensions of $\mathbf{x}$ differ from those of $\mathcal{F}(\mathbf{x})$, a 1x1 convolution adjusts the dimensions for compatibility. This adjustment ensures that the input and output dimensions match for the addition operation.

During the forward pass, $\mathbf{x}$ is processed through batch normalization and ReLU activation, followed by convolutional layers. The resulting feature maps are added to the original input $\mathbf{x}$, implementing the residual learning framework by He et al. [7].

The decoder path upscales the feature maps to the original image size using transposed convolutional layers. Each upsampled feature map is combined with a corresponding feature map from the encoder path through a skip connection. The 'upsample add' function aligns the upsampled feature maps with the lateral feature maps for precise reconstruction.

The final layers of the decoder include several convolutional layers that refine the output features. The network concludes with an upsampling layer that scales the output to the desired resolution. The final output is processed through a sigmoid activation function to generate segmentation maps.

3.3.4. Floor plan representation

The data in the floor plan is represented using different segmentation and junction regression masks. The segmentation maps consist of a wall and room map, which provides the probability that a pixel belongs to a wall or a specific room type, with the exact classes depending on the dataset used. The

second map is the icon map, which indicates the probability that a pixel represents a specific icon type or is empty, with the specifics depending on the dataset.

The junctions are categorized by their geometry and function. Wall junctions are where wall segments intersect and are classified by the number of wall segments meeting at a junction. These include I-shaped, L-shaped, T-shaped, and X-shaped, with a total of 13 different types considering multiple orientations. Opening junctions mark the endpoints of openings such as doors or windows. Icon junctions are located at the corners of axis-aligned bounding boxes representing icons, such as furniture or fixtures, with junctions at the top-left, top-right, bottom-left, and bottom-right corners.

3.3.5. Multi-task segmentation

Multi-task learning (MTL) is a machine learning paradigm where a single model is trained to perform multiple tasks simultaneously. This approach, introduced and explored by Caruana et al. in 1997 [2], relies on the principle that tasks often share commonalities that can be leveraged to improve learning efficiency and model generalization. Instead of training separate models for each task, MTL allows tasks to share representations, which can lead to better performance, especially in cases where data is limited for some tasks.

Mathematically, given a set of tasks $T = \{T_1, T_2, \ldots, T_k\}$, each with its own loss function $\mathcal{L}_{T_i}$, the overall objective in MTL is often formulated as a weighted sum of the individual task losses:

$$\mathcal{L}_{\text{MTL}} = \sum_{i=1}^{k} \alpha_i \mathcal{L}_{T_i} \tag{3.17}$$

Here, α_i represents the weight assigned to the loss of task T_i. A key challenge in MTL is determining the appropriate weights α_i so that the model can learn all tasks effectively without overfitting to any single one or neglecting others. If the weights are imbalanced, the model might prioritize certain tasks over others, leading to sub-optimal performance across the task set.

While MTL offers advantages such as improved generalization and more efficient data usage, it also faces challenges, particularly the risk of negative transfer. Negative transfer occurs when learning one task negatively affects the performance of another, typically because the tasks are not sufficiently related.

To address the problem of selecting appropriate task weights, [9] introduced a novel approach using task-specific uncertainty to dynamically weigh the losses in MTL. This approach is based on the idea that tasks with higher uncertainty should contribute less to the overall loss function, allowing the model to focus more on tasks that it can learn with greater confidence. The uncertainty-weighted loss function proposed by Kendall et al. is given by:

$$\mathcal{L}_{\text{MTL}} = \sum_{i=1}^{k} \frac{1}{2\sigma_i^2} \mathcal{L}_{T_i} + \log(\sigma_i) \tag{3.18}$$

In this equation, σ_i represents the uncertainty associated with task T_i. The first term, $\frac{1}{2\sigma_i^2}\mathcal{L}_{T_i}$, scales the loss of each task by the inverse of its uncertainty σ_i^2, ensuring that tasks with higher uncertainty, which likely have noisier or less reliable data, have a smaller influence on the overall loss. The second term, $\log(\sigma_i)$, serves as a regularizer to prevent the model from assigning excessively large uncertainties to any task, which would effectively reduce the task's importance to near zero.

Kendall et al. derived this weighting approach by treating the task-specific uncertainties as learned parameters that model the homoscedastic uncertainty. This is a type of uncertainty that remains constant across different inputs but varies between tasks. They started with the assumption that the task-specific losses follow a Gaussian distribution with variance σ_i^2. This leads to the formulation:

$$\mathcal{L}_{T_i} \sim \mathcal{N}(0, \sigma_i^2) \tag{3.19}$$

Given this assumption, the log-likelihood of the observed data can be maximized, which results in the loss function described earlier. The regularization term $\log(\sigma_i)$ follows from the maximum likelihood estimation process, ensuring that the learned uncertainties σ_i do not become unbounded.

This approach balances the contributions of different tasks during training, allowing the model to learn multiple tasks simultaneously even when the tasks differ in complexity or data quality. By dynamically adjusting the weights based on the uncertainty of each task, this method mitigates the risk of negative transfer and provides more robust learning.

For CubiCasa [8], the loss functions for different tasks are defined as follows. For segmentation tasks, cross-entropy loss is employed, while for heatmap predictions, mean squared error (MSE) loss is used. These loss functions can be refined by incorporating uncertainty to enhance learning efficiency and robustness.

The cross-entropy loss function quantifies the discrepancy between the predicted probability distribution and the ground truth labels. For a segmentation task, let y_k denote the ground truth label for category k, and let p_k represent the predicted probability for the same category. The cross-entropy loss is expressed as:

$$L_{\text{CE}} = -\sum_k y_k \cdot \log(p_k) \tag{3.20}$$

where $p_k = \text{softmax}(f_{W_k}(x))$ represents the predicted probability obtained through a softmax function applied to the output $f_{W_k}(x)$ of the model for category k.

The mean squared error loss is used for tasks involving continuous predictions, such as heatmap generation. Given the ground truth y_i and the predicted output $f_{W_i}(x)$ for the i-th task, the MSE loss is defined as:

$$L_{\text{MSE}} = \frac{1}{2}\|y_i - f_{W_i}(x)\|^2 \tag{3.21}$$

This loss measures the squared difference between the ground truth and the prediction, averaged over all instances. When incorporating uncertainty into these loss functions, the cross-entropy loss for segmentation tasks is modified as follows:

$$L_S = -\sum_{k \in \{\text{rooms, icons}\}} \frac{1}{\sigma_k} y_k \cdot \log(\text{softmax}(f_{W_k}(x))) \tag{3.22}$$

In this equation, σ_k represents the learned uncertainty parameter for each category k. The term $\frac{1}{\sigma_k}$ scales the cross-entropy loss, emphasizing categories with lower uncertainty. The regularization term is not included because the uncertainty parameter σ_k remains positive throughout training, which ensures that the scaling factor remains positive and does not require further adjustment. For heatmap prediction tasks, the mean squared error loss is refined to:

$$L_H = \sum_i \left[\frac{1}{2\sigma_i^2}\|y_i - f_{W_i}(x)\|^2 + \log(1+\sigma_i) \right] \tag{3.23}$$

Here, σ_i is the uncertainty parameter for the i-th task. The term $\frac{1}{2\sigma_i^2}\|y_i - f_{W_i}(x)\|^2$ scales the MSE loss by the inverse of the squared uncertainty, focusing more on tasks with lower uncertainty. The additional regularization term $\log(1+\sigma_i)$ prevents the uncertainty parameters from becoming excessively large, which ensures that their influence remains controlled and prevents them from reducing task contribution to (near) zero.

The total loss function for the CubiCasa model [8] integrates the losses from different tasks into a single objective function. It combines the segmentation loss for room and icon classification with the loss for

heatmap predictions to guide the model towards accurate multi-task learning. The total loss L_{total} is defined as:

$$L_{\text{total}} = L_{\text{rooms}} + L_{\text{icons}} + L_{\text{heatmaps}} \tag{3.24}$$

where L_{rooms} and L_{icons} represent the segmentation losses for rooms and icons, respectively, and L_{heatmaps} denotes the loss for heatmap predictions. Substituting the loss functions for segmentation and heatmaps, the total loss can be expressed as:

$$L_{\text{total}} = -\sum_{k \in \{\text{rooms, icons}\}} \frac{1}{\sigma_k} y_k \cdot \log(\text{softmax}(f_{W_k}(x))) + \sum_i \left[\frac{1}{2\sigma_i^2} \|y_i - f_{W_i}(x)\|^2 + \log(1+\sigma_i) \right] \tag{3.25}$$

Here, σ_k is the uncertainty parameter for each segmentation category k, and σ_i is the uncertainty parameter for each heatmap i. This total loss function combines the contributions of the segmentation and heatmap tasks, leveraging the uncertainty parameters to balance the model's performance across these different tasks.

3.3.6. Post-processing to vector-based representation

The vectorization process transforms the segmentation masks and junctions into a structured vector-based representation described by Kalervo et al. [8]. The method is based on Liu et al. [13] and begins by extracting key structural elements from the raster data, such as walls, rooms, icons, and openings, and then representing these elements in a geometrically consistent manner.

The initial step in the vectorization process involves detecting wall junctions from the heatmaps generated by a multi-task Convolutional Neural Network (CNN). Junctions are points where walls meet, and they define the structural layout of the floor plan. These junctions are identified by finding local maxima in the wall heatmaps. The junctions are then connected based on their geometric orientation. If two junctions are vertically or horizontally aligned within a small pixel tolerance and have joints facing each other, they are connected to form a potential wall segment. This step results in a skeletal representation of the floor plan, consisting of connected wall lines.

Once the wall lines are established, the process continues with refining these lines to form wall polygons, which represent the actual walls in the floor plan. The wall polygons are created by first pruning the wall skeleton based on the wall segmentation map. The width of each wall is inferred by sampling along the wall lines and examining the intensity profile of the wall segmentation map, allowing for the accurate tracing of wall boundaries. Each wall is then represented as a polygon, with its vertices defining the wall's boundary.

The next stage involves determining the locations and dimensions of rooms, which are inferred based on the previously identified wall junctions. Specifically, the process searches for triplets of junctions that span rectangular areas free of any internal junctions. These areas are then divided into a grid, with each cell representing a portion of the floor plan's interior. The cells are labelled based on a voting mechanism that considers the room segmentation map, and neighbouring cells are merged into larger room polygons if they share the same room label and are not separated by a wall. This results in a precise representation of the rooms within the floor plan.

Similarly, icons (representing objects like furniture or appliances) are restored using a method analogous to room extraction. Instead of using wall junction heatmaps, the process relies on icon corner heatmaps to detect and connect the icon corners, forming icon polygons. This approach ensures that icons are accurately placed within the vectorized floor plan.

The final step in the vectorization process involves the detection of openings, such as doors and windows. Openings are identified by connecting pairs of vertically or horizontally aligned endpoints, based on predictions from the opening heatmaps. The label of each opening is determined from the segmentation maps, and the width of the opening is set to match that of the corresponding wall polygon. Any opening endpoints that do not fall within the wall segmentation are discarded, ensuring that only valid openings are retained.

The result of this multi-step process is a set of vector-based polygons that represent all the structural elements of the floor plan, including walls, rooms, icons, and openings. Each element is encoded with information about its location, dimensions, and category, enabling the creation of a digital model from the original raster floor plan image.

References

[1] Mykhaylo Andriluka et al. “2d human pose estimation: New benchmark and state of the art analysis”. In: *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*. 2014, pp. 3686–3693.

[2] Rich Caruana. “Multitask learning”. In: *Machine learning* 28 (1997), pp. 41–75.

[3] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[4] Abolfazl Farahani et al. *A Brief Review of Domain Adaptation*. 2020. arXiv: 2010.03978 [cs.LG]. URL: https://arxiv.org/abs/2010.03978.

[5] Ian Goodfellow. *Deep Learning*. MIT Press, 2016.

[6] Arthur Gretton et al. “A kernel method for the two-sample-problem”. In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*. 2006, pp. 513–520.

[7] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 770–778. URL: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

[8] Ahti Kalervo et al. “Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis”. In: *Image Analysis: 21st Scandinavian Conference, SCIA 2019, Norrköping, Sweden, June 11–13, 2019, Proceedings 21*. Springer. 2019, pp. 28–40.

[9] Alex Kendall, Yarin Gal, and Roberto Cipolla. “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7482–7491.

[10] Nour Eldeen M Khalifa, Mohamed Loey, and Seyedali Mirjalili. “A comprehensive survey of recent trends in deep learning for digital images augmentation”. In: *Artificial Intelligence Review* 55.3 (2022), pp. 2351–2377. DOI: 10.1007/s10462-021-10066-4.

[11] Aditya Khamparia and Karan Mehtab Singh. “A systematic review on deep learning architectures and applications”. In: *Expert Systems* 36.3 (2019), e12400.

[12] Chen Liu, Jiaye Wu, and Yasutaka Furukawa. *FloorNet: A Unified Framework for Floorplan Reconstruction from 3D Scans*. 2018. arXiv: 1804.00090 [cs.CV]. URL: https://arxiv.org/abs/1804.00090.

[13] Chen Liu et al. “Raster-to-vector: Revisiting floorplan transformation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2195–2203.

[14] Mingsheng Long et al. “Learning transferable features with deep adaptation networks”. In: *Proc. ICML*. 2015, pp. 97–105.

[15] Reza Moradi, Reza Berangi, and Behrouz Minaei. “A survey of regularization strategies for deep models”. In: *Artificial Intelligence Review* 53.6 (2020), pp. 3947–3986.

[16] Marius-Constantin Popescu et al. “Multilayer perceptron and neural networks”. In: *WSEAS Transactions on Circuits and Systems* 8.7 (2009), pp. 579–588.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV]. URL: https://arxiv.org/abs/1505.04597.

[18] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115 (2015), pp. 211–252.

[19] Connor Shorten and Taghi M Khoshgoftaar. “A survey on image data augmentation for deep learning”. In: *Journal of big data* 6.1 (2019), pp. 1–48.

[20] Patrice Y. Simard et al. “Transformation Invariance in Pattern Recognition — Tangent Distance and Tangent Propagation”. In: *Neural Networks: Tricks of the Trade*. Ed. by Genevieve B. Orr and Klaus-Robert Müller. Vol. 1524. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1998, pp. 239–274. DOI: 10.1007/3-540-49430-8_13. URL: https://doi-org.tudelft.idm.oclc.org/10.1007/3-540-49430-8_13.