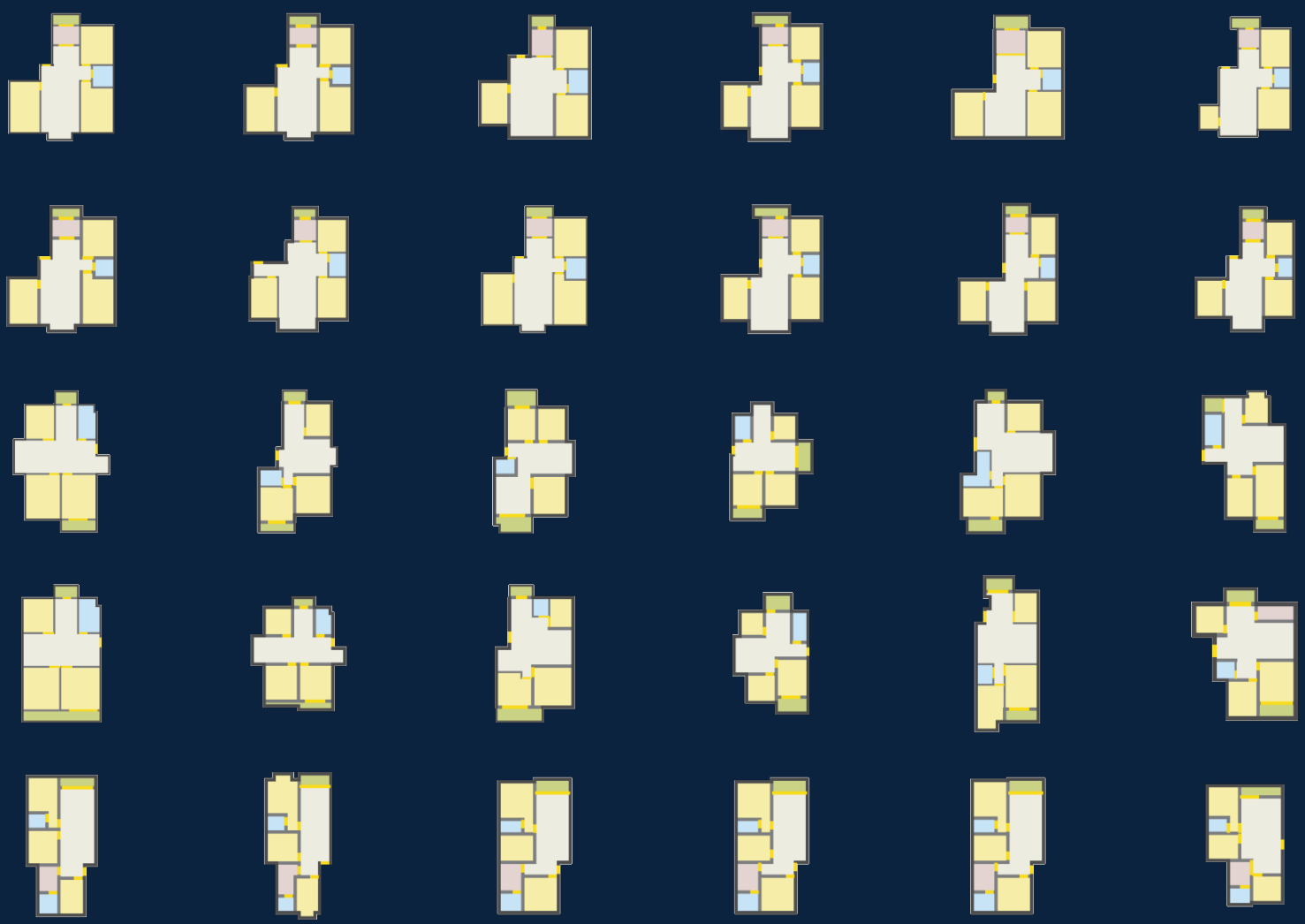
Exploring Training Pair-Generation Strategies for Deep Metric Learning for Floor Plan Retrieval

Emanuel Kuhn



Delft University of Technology

TU Delft

Exploring Training Pair-Generation Strategies for Deep Metric Learning for Floor Plan Retrieval

by

Emanuel Kuhn

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday February 21, 2024 at 10:00 AM.

Student number:	4727843	
Project duration:	September 22, 2022 – February 21, 2024	
Thesis committee:	Dr. S. Khademi,	TU Delft, Daily supervisor
	Dr. ir. J. van Gemert,	TU Delft, Advisor
	Ir. C.C.J. van Engelenburg,	TU Delft, Daily co-supervisor
	Dr. C. Oertel,	TU Delft, External committee member

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This report is the result of my thesis on *Exploring Training Pair-Generation Strategies for Deep Metric Learning for Floor Plan Retrieval*. In essence, I tried to develop a strategy for training a model to search for relevant floor plan images.

This work was conducted as part of the Computer Vision Lab of the EEMCS faculty, and the AiDAPT AI lab at the architecture faculty. As a computer science student with no background in architecture, it was fascinating to see how AI is applied in a creative field.

I am grateful for the guidance and support provided by my supervisors. I would like to thank my thesis supervisor, Dr. Jan van Gemert, for his insights and feedback during the evaluation meetings. Equally, I extend appreciation to my daily supervisor, Dr Seyran Khademi, for her valuable feedback and guidance during the process. My thanks also go out to my daily-co supervisor, Casper van Engelenburg, for the many productive discussions and valuable feedback during our weekly meetings. Additionally, I am thankful to the other members of the AiDAPT lab for the engaging coffee talk paper discussions.

Many thanks to my family and friends for their support and motivation while I was writing this thesis.

Finally, I would like to acknowledge my thesis graduation committee, consisting of Dr. ir. J. van Gemert, Dr. S. Khademi, and Dr. C. Oertel.

Emanuel Kuhn
Delft, February 2024

Contents

1 Introduction

In architecture, floor plans are diagrams that show a top-down view of the layout and relationships of rooms in a building. Floor plans are used by architects as a step in the design process to plan the use of space in a building. Figure 1.1 shows an example of a real world floor plan, as well as a simplified representation, where rooms are color-coded by their type.

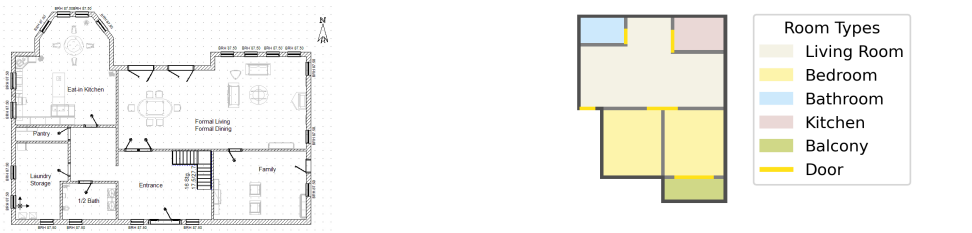
Floor plan retrieval. It would be beneficial for architects to be able to search for similar floor plans in a database. Similarly, floor plan retrieval could be useful for real estate websites, to recommend properties with similar floor plans to prospective buyers.

Existing retrieval techniques for finding similar images by example work well for natural photos. However, image retrieval techniques that work well for photos, are not well suited for retrieving similar floor plan diagrams images.

For photos, texture, color, and the presence of objects are features that an image retrieval model can learn to find similar images. Unlike natural photos, the information in a floor plan diagram is largely contained in the shape of rooms, and the connections between rooms, rather than in their visual texture or color.

Learning embedding vectors. This thesis explores a new way to find similar floor plans. We define a floor plan as a simplified drawing that shows a buildings' layout from above, like the example on the right in the figure below. The goal is to develop a method that takes as input one of these diagrams, and finds others that are similar in layout and structure.

To do this, we will try to learn *embedding vectors*. This means that every floor plan will be converted into a set of numbers, that can efficiently be compared by a computer. This set of numbers don't have a direct meaning or interpretation, but should contain important information about the floor plans layout and structure.



(a) Real world floor plan taken from wikipedia.

(b) Simplified floor plan example from the RPLAN dataset.

Figure 1.1: Examples of floor plan diagrams.

The process of learning embedding vectors is done using a method called *deep representation learning*. This is a type of machine learning, in which a model learns what makes a floor plan unique by showing many examples.

We explore two new ways to train a representation learning model to learn embedding vectors. These methods involve creating *training pairs*, pairs of floor plans that the model is trained on. By comparing the images in these pairs, the model learns to recognize and match similar floor plans.

Data augmentation. For natural images, representation learning techniques use *data augmentation* to learn embedding vectors that capture information in the image. Data augmentation techniques transform an image to randomize its appearance, while preserving essential features. Commonly used data augmentations include randomly cropping an image, changing the colors, and randomly rotating the image. The goal of using data augmentations, is to teach the model that the embedding vectors of two randomly augmented versions of the same image should be similar.

However, for floor plans, relying on standard image augmentations alone leads to poor retrieval results. The reason is that image augmentations alone may not introduce enough variation in the images, as floor plans can be similar without having exactly the same room shapes.

To address this, we propose GeomPerturb, an augmentation strategy specifically for floor plans, that involves randomly shifting the walls. The walls are moved in a manner that ensures that the connections between rooms remain intact. GeomPerturb in combination with standard image augmentations, introduces more variation between randomly augmented views of the same floor plan. These randomly augmented views are used as training pairs for the model, to learn embeddings that are invariant to small changes in the shape of the rooms.

Training pairs based on graph similarity. Floor plans can be represented by *graphs*, in addition to images. In the graph representation of a floor plan, each room (node) is connected to others by doors (edges). Graph representations of floor plans can be used to select pairs of floor plans from the training dataset which have similar connectivity between rooms. These training pairs are then used to train the same representation learning model, instead of using the GeomPerturb data augmentation strategy.

Report structure

The main body of work is presented as a scientific article in chapter 2. After the scientific article, the report includes technical background sections explaining concepts used in the scientific article.

The background chapter starts in section 3.1 with explanations on the deep learning concepts and methods used for learning embedding vectors from images. Then, section 3.2 explains how embedding vectors can be used to build a retrieval system. Finally, section 3.3 explains the concept of graphs, and goes into graph similarity heuristics, which are used for generating training pairs based on the similarity between graph representations of floor plans.

2

Scientific article

Exploring Training Pair Strategies for CNN-Based Metric Learning for Floor Plan Retrieval

Emanuel Kuhn
TU Delft

Abstract

Existing content-based image retrieval models work well for natural photos, but not for images of architectural floor plans. Previous work on floor plan retrieval has focused on graph-based methods, rather than image-based floor plans. Training a CNN-based representation learning framework on segmented floor plan images with standard image augmentations does not result in semantically meaningful retrievals. This work shows that a CNN-based representation learning model can learn features for retrieving floor plans that have similar graphs given the right training signal. Two methods were investigated here: GeomPerturb, a data augmentation that perturbs the underlying geometry of a floor plan, and a weakly supervised method with labels based on the graph edit distance between a pair of floor plans. The results show that while GeomPerturb learns representations that are correlated with the floor plan graph, training with GED labels leads to better retrievals both in terms of the floor plan graph and with respect to room shapes.

1. Introduction

In architectural design, floor plans are top-down drawings of a physical space, such as a building. Floor plans show how rooms and spaces are laid out and connected. It would be beneficial for designers to be able to search for similar floor plans in a database. In addition, floor plan retrieval could be useful on real estate websites to recommend properties that have similar floor plans to prospective buyers. For these use cases, it is advantageous for retrieval to be fast, i.e., ideally takes less than a second per query.

The useful features of a floor plan for retrieval are the shape of the rooms, as well as the connections between the rooms. Nevertheless, floor plan similarity is hard to define, as it is inherently multifaceted. A pair of floor plans may be considered similar by having similar room/overall shapes, similar relative positions of rooms, similar room types, or similar connections between rooms, but also other harder to define aspects. It is unclear how these aspects should be weighted, or what it even means for the shape to be similar between floor plans.

Floor plan diagrams are usually in the form of images. The structure in the floor plan, i.e., the connections between the rooms, can also be represented as a graph. In this paper, floor plans are considered to be images, where each room type is represented by a unique color.

The approach we take is to apply a deep metric learning model, SimSiam [4], to floor plan images to learn an embedding vector for each floor plan. These embedding vectors can then be used to efficiently retrieve similar floor plans [16]. SimSiam is trained on pairs of images, with each pair formed by randomly augmenting a training sample. We find that training a SimSiam model with standard image augmentations [3] does not lead to relevant retrievals. We believe this is due to a feature mismatch between natural photos and floor plan diagram images. Whereas natural photos have a lot of semantic information in the texture, the floor plan images are devoid of texture and instead, the information is in the shape of rooms and adjacencies between rooms. The image augmentations that are used in SimSiam were previously developed for learning useful features of natural photos, and are not tailored for retrieving similar diagrams.

To address the poor performance of training SimSiam with standard image augmentations, we explore two alternatives. First, we devise a handcrafted data augmentation, GeomPerturb, which randomly modifies the underlying geometry and draws the perturbed room shapes to a new floor plan image. This method keeps with the spirit of SimSiam that training pairs are formed as two randomly augmented views of the same training sample. Second, we select pairs of similar floor plans from the dataset based on the *graph edit distance* (GED) between them, and use these as training pairs instead of solely relying on data augmentation.

Our contributions are summarized as:

- CNNs can learn features of floor plans that correlate

with the similarity of floor plan graphs.

- Perturbing room geometries can be used as data augmentation (GeomPerturb).
- We propose a method to train with supervised GED similarity labels without the need to compute GED for a large number of pairs.

2. Related work

2.1. Content-based retrieval

Content-based retrieval (CBR) is the task of retrieving items based on their content, rather than by associated keywords or tags [37]. One of the querying approaches is retrieval by example [37]. For floor plans, this means searching for similar floor plans given an example floor plan, instead of searching by metadata such as the number of rooms. A fast CBR algorithm consists of two stages: characterizing the content by a low dimensional descriptor that can be used as an index, and efficiently searching the index. Faiss [16] is a library that allows for efficient similarity search in a vector space. Thus, one viable approach for CBR of floor plan images is to first learn vector embeddings of floor plans, and efficiently retrieve them using the vectors as index.

2.2. Floor plan retrieval

Previous floor plan retrieval methods can be divided into two categories, learned and non-learning-based approaches. Here, the learned approach refers to models with parameters that have to be optimized on training data. The approaches that learn a model can further be subdivided into those that operate directly on a floor plan image [29,35], and those that work with a graph representation of floor plans [15, 25, 26].

Image-based models. CNN-based classification models have been adapted for floor plan retrieval, trained on predicting floor plan subcategories based on room presence or manually labeled floor plan "shape" [29, 35]. For instance, [35] use feature vectors extracted from the VGG-16 backbone, trained on the floor plan subcategory classification task, for nearest neighbor search. In our study, we diverge from the classification-based approach by instead applying a distance metric learning loss, potentially allowing for more fine-grained embeddings instead of learning to map floor plans of the same sub-category to the same embedding.

Other works tackle floor plan analysis by segmenting rooms and objects such as walls and doors [17, 21, 41]. [41] extracts a rule-based transformation of a graph from the segmented image, which is subsequently used for retrieval by using the maximum common subgraph (MCS) as a heuristic for the similarity between floor plan graphs.

GNN-based models. To train models for measuring the structural similarity of user interface layouts, [22] proposed training a graph neural network (GNN) on graphs attributed with features derived from bounding boxes of layout elements. The GNN is trained on triplets with similarity labels based on Intersection over Union (IoU) between segmented images. Later [15, 26] applied this approach to floor plans, but used graph matching networks [20] to overcome the lack of structural information in IoU labels. The use of graph matching networks improved retrieval performance but is too slow for fast retrieval in a large database [31]. In our study, we reverse the approach taken in [15, 26] of using a model with a structural bias and similarity labels based on IoU. Instead, we use a CNN-based model, without a prior towards graph structure, and, in one of our experiments, train with similarity labels based on a graph distance metric.

Another graph-based method ignores the room shapes and locations completely [25], and instead, a GNN [1] is trained to approximate the graph edit distance between floor plan graphs. We similarly use GED for selecting training pairs of similar floor plans, but instead of predicting GED for a pair of images, we learn embedding vectors that should be similar for floor plans with low GED.

Heuristic approaches. Instead of learning a similarity metric, heuristic-based approaches are also used for computing similarity between floor plans. [41] use the maximum common subgraph (MCS) as a metric to compare floor plan graphs. Graph edit distance (GED), which assigns a cost to change one graph into another, is likewise used as a measure of dissimilarity in [33, 38]. A variant of GED for trees is used in [19]. SSIG [38] use a combination IoU and GED as similarity metric. Most heuristic approaches need time-consuming pairwise computation during retrieval. In this paper, we make use of the GED, IoU, and SSIG heuristic metrics for generating training examples but do retrieval based on learned embedding vectors.

Retrieval speed. Not all the methods for retrieval take the same amount of time to return a result. Methods that use time-consuming pair-wise computation are orders of magnitudes slower at retrieval compared to those that use fast pair-wise computation on compact representations. Table 1 shows reported retrieval times for the methods that included a reported retrieval time. Note that for methods that map each floor plan to an embedding vector [22, 35], fast algorithms [16] exist for nearest neighbor (k-NN) search in such a vector space. Table 1 shows that lookup in a vector space is faster than any of the reported pairwise computations used by other methods. A method that learns vector representations of floor plans is thus desirable for enabling fast retrieval.

Method	Reported retrieval time [1]
LayoutGMN [26]	55 min
FP recommendation GNN [25]	6.6 min
SSIG [38]	2-9s
Vector Space Retrieval (Faiss) [16]	30ms [2]

[1] Extrapolated to a database of 100k candidate floor plans based on reported pairwise compute times.
[2] Benchmark ours for (randomly generated) 1024 dim vectors with exhaustive k-NN search, on a single CPU.

Table 1. Comparison of retrieval compute time for various methods. This table presents the time required for retrieving a single query from a dataset of 100k candidate floor plans, based on the times reported in respective papers.

2.3. Deep metric learning

The goal of metric learning is to learn a mapping from data samples to an embedding space, such that similar data samples are close together in the embedding space and dissimilar samples are further apart. Use cases for learning embedding vectors include face [28] verification, speaker recognition [6], and tasks with an informational retrieval aspect [23, 27]. Embeddings can be extracted from models trained on a classification task [34], or trained with an embedding loss function based on pairs [11] or triplets [28, 39] of similar and dissimilar examples.

Siamese networks. Siamese networks [2] employ weight sharing between two branches to map two inputs to the same embedding space. Siamese networks are a natural tool for comparing entities [4] and are often used in conjunction with metric learning losses.

Metric learning losses. The contrastive loss [5] functions on a pair of data samples which is either labeled as positive, i.e., similar, or negative. The triplet loss [28] instead works on triplets of an anchor, a positive and negative sample, and pushes the distances between anchor and positive to be smaller than between anchor and negative. For effective training, it is important to select positive and negative examples that are currently hard for the model [28]. An alternative to negative hard-mining, is using all other examples in a mini-batch as negative examples in combination with a large batch size [3, 32].

Contrastive learning for SSL. Self-supervised learning (SSL) is a paradigm for learning from unlabeled data. In [3, 13, 42], contrastive learning is used in conjunction with data augmentation for self supervision. Two randomly augmented views of the same image form a positive pair, and randomly augmented views of different images can be used as a negative examples.

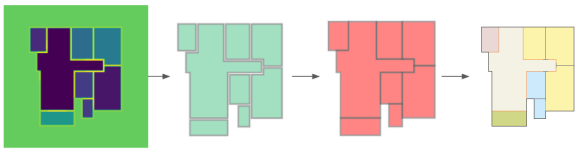


Figure 1. Illustration of the preprocessing steps for RPLAN images to be used with GeomPertub. From left to right: RPLAN categorical mask image, extracted shapes, enlarged shapes, rendered RGB image.

Non-contrastive learning. Other variants of SSL representation learning methods have found that the contrastive component of the loss, i.e., contrasting with negative examples, is not always necessary to learn useful embeddings [4, 10]. Although it is still not entirely well understood why non-contrastive learning works [36, 43, 44], we choose to use the SimSiam [4] representation learning framework for its simple architecture and robustness to training with smaller batch sizes.

3. Method

3.1. Dataset

We use the RPLAN [40] dataset for our experiments. RPLAN was originally proposed for studying floor plan generation, and is a densely annotated dataset of floor plans from real residential buildings [40]. It consists of around 80k segmented floor plan images, where pixel values in the image correspond to a room type or structural element that is present at that location. See the left most image in Fig. 1 for an example.

Preprocessing. For the GeomPerturb augmentation proposed in Sec. 3.3, the segmented floor plan images need to be preprocessed to a specific vector geometry format. Figure 1 shows an overview of the preprocessing steps. First, the room masks are turned into polygon shapes using the `features.shapes` method of the `rasterio` [8] python library. Then, each room is enlarged such that neighboring rooms exactly touch, i.e., making the walls between the rooms have zero thickness. Around 46k of the 80k RPLAN floor plans were successfully preprocessed in this manner. Finally, the preprocessed room geometries can be drawn to a new image. Note that the appearance of the drawn image, such as the thickness of the walls, can be controlled when drawing the geometries.

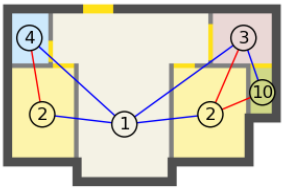


Figure 2. Figure of the *floor plan graph* overlaid on a floor plan. The nodes are labeled by room type. The edges are colored blue if there is a door between the rooms, and red if they only share a wall. Doors are colored yellow.

Graph representation. Floor plans can also be represented as graphs. We define the *floor plan graph* by construction: for each room, create a node labeled with a *room category* attribute. Now connect each pair of room nodes with an edge if they have a wall in common. Label the edge with a *door* attribute that indicates if the rooms are also connected by a door. See Fig. 2 for a visualization of the floor plan graph overlaid on a floor plan. The code for preprocessing RPLAN samples will be made available.

3.2. Siamese representation learning framework

SimSiam. In all our experiments, we use the SimSiam framework [4] with a ResNet-18 [14] backbone. The Resnet backbone consists of CNN layers with skip connections, and an MLP head that outputs a feature vector. During training, the SimSiam framework takes two randomly augmented views x_1, and x_2 from the same image x. The objective of the model is to map these two views to similar representation vectors. The SimSiam model [4] consists of two components, an encoder network and a predictor head. First, the two views are processed by the encoder network f, consisting of the ResNet backbone and a 3 layer projection MLP head, that maps the input image to an N(=512) dimensional vector. Then the predictor head h transforms the output of one of the two views. The predictor head is a 2 layer MLP with a hidden dimension of 128, and the same output dimension N, forming a bottleneck structure. After applying the encoder and transforming one of the two views using the predictor, there are two vectors, $p_1 = h(f(x_2))$ and $z_2 = f(x_1)$. The loss function minimizes the negative cosine similarity between p_1 and z_2:

$$\mathcal{D}(p_1, z_2) = -\frac{p_1 \cdot z_2}{|p_1|_2 |z_2|_2}$$

where $\|\cdot\|_2$ is the Euclidean norm. The authors [4] found that using a stop-gradient (`stopgrad`) operation on the z branch of the loss is an essential part of training the SimSiam model. In PyTorch, stop-gradient is implemented by calling the `z.detach()` method. The final loss is also symmetrized following [10], thus yielding the following loss function:

$$\mathcal{L} = \frac{1}{2}\mathcal{D}(p_1, \texttt{stopgrad}(z_2)) + \frac{1}{2}\mathcal{D}(p_2, \texttt{stopgrad}(z_1))$$

Image-space augmentations. SimCLR [3] studied the effect of combinations of image augmentations, and found that combinations of strong image augmentations lead to better performance. The augmentations used in SimSiam are based on previous works and are RandomResizedCrop with scale $(0.2, 1.0)$, RandomHorizontalFlip, ColorJitter, random grayscale and blurring.

Image augmentations & floor plans. The set of augmentations used in previous work specially makes sense for natural images, and less so for segmented floor plan images. For example, commonly used RandomResizedCrop parameters are set such that for a pair of views, one view can be a crop of the other view. For floor plans of single apartments, such as RPLAN, this is undesirable as a crop of just a few rooms is not indicative of the floor plan as a whole. For natural images, it makes sense to only use horizontal flips, as upside-down symmetry does not occur often in nature. For floor plans, however, horizontal and vertical flips both make equal sense, and 90-degree rotations additionally preserve the layout. Color augmentation was found to make a large impact on generalization on natural images [3] because otherwise, two random crops of an image will have similar color distributions. For segmented images, however, color augmentations do not make sense as color carries the semantic meaning much more compared to natural images.

The image augmentations used in this paper are: random horizontal and vertical flips, randomly applied 90 degree rotations, and random resized crops with scale $(0.8, 1.0)$ and ratio $(3/4, 4/3)$. These augmentations are used in addition to the proposed GeomPerturb augmentation (Sec. 3.3) or weakly supervised approach (Sec. 3.4). In addition, it is shown in Sec. 4.1 that both image space augmentations and either GeomPerturb or weakly-supervised labels are needed to learn useful representations.

3.3. GeomPerturb

The standard method for training a SimSiam model is to generate pairs of two randomly augmented views of the same image. We used the set of image augmentations listed in Sec. 3.2 as a baseline for learning representation vectors with SimSiam. As shown in Sec. 4.1, these representations fail to correlate with how rooms are connected to each other in a floor plan. This makes it clear that image augmentations alone are not sufficient for learning semantically useful representations of floor plan layouts, and that alternative methods should be investigated.

3.3.1 Augmenting underlying geometry

Intuitively, the model should learn that floor plans that have similar floor plan graphs should be mapped to similar embedding vectors. This means that floor plans that have (slightly) different room shapes, but with the same connections between rooms, should be similar. This leads to the idea that instead of augmenting the image representation of the floor plan, the underlying geometry should be augmented to introduce variation in the shapes while keeping the same connections between rooms.

Room geometries. For the RPLAN dataset, the floor plans are stored as segmentation masks that can be preprocessed into polygon shapes. The proposed GeomPerturb augmentation makes use of that by first extracting the preprocessed shapes from the images, then perturbing the shapes, and finally rendering the perturbed room shapes back to an image format. For processing geometries, the `shapely` python library [9] was used.

GeomPerturb method. GeomPerturb starts by picking a random wall element in the floor plan, and translating it perpendicularly by a random amount, resulting in the proposal of a geometrically perturbed floor plan. A heuristic acceptance function either accepts or rejects the proposal, generating new proposals until one is accepted. Then this procedure of moving a random wall is repeated M times. Figure 3 shows an example of a sequence of random wall moves used to generate a GeomPerturbed floor plan. In Fig. 4 examples of accepted versus rejected proposals are shown.

The steps in GeomPerturb are explained in more detail in the following:

1. **Sample wall move.** The first step consists of randomly picking a wall element, and then moving it perpendicularly by a random amount.
   (a) $r \sim \{\text{set of rooms}\}$
   (b) $w \sim \{\text{set of walls in room } r\}$
   (c) $t \sim \mathrm{Uniform}(-20, 20)$
   (d) $r' \leftarrow$ room r with wall w translated by t
   (e) If r' overlaps with other rooms, remove the overlap from the other rooms.
   (f) Return new proposal floor plan
2. **Acceptance heuristic.** The heuristic acceptance function either accepts or rejects the wall move. The following constraints are checked by the acceptance heuristic:
   (a) All rooms should be valid polygons
   (b) The area of each new room should be at least half of the previous area
   (c) The walls should still be horizontal or vertical
   (d) The new floor plan should not have empty spaces
   (e) The ratio between the exterior and area should increase by at most 5%
   (f) Compute the room adjacency graph of the proposal, the new adjacency graph should be the same as the original adjacency graph

   these constraints make sure that the proposed floor plan keeps the same room connections as the original, and doesn't result in unrealistically complex room shapes. If the proposal is rejected, generate proposals until one is accepted.
3. Repeat steps 1 and 2 M times, where M is a hyperparameter. The result is the geometries of a new floor plan with randomly perturbed room shapes.

The GeomPerturb method has the following hyperparameters: the number of wall moves M, the maximum translation amount, and the criteria of the acceptance heuristic. We chose the hyperparameters by visually inspecting the generated augmentations for feasibility.

GeomPerturb is compute-heavy. Thus, we precompute a fixed amount of augmentations, draw them to an image and store as an array of 50 images per floor plan. In the training loop, a precomputed augmentation is randomly sampled each time a floor plan is used.

We found that it works best to combine the GeomPerturb augmentation with image augmentations. This finding is inline with SimCLR's findings [3] that a combination of

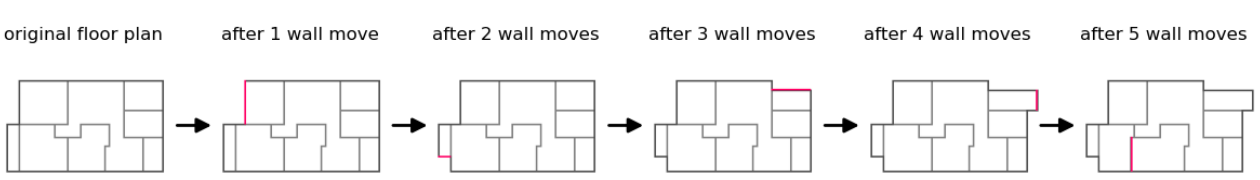


Figure 3. GeomPerturb example sampling trajectory. The wall that was moved is highlighted in red.

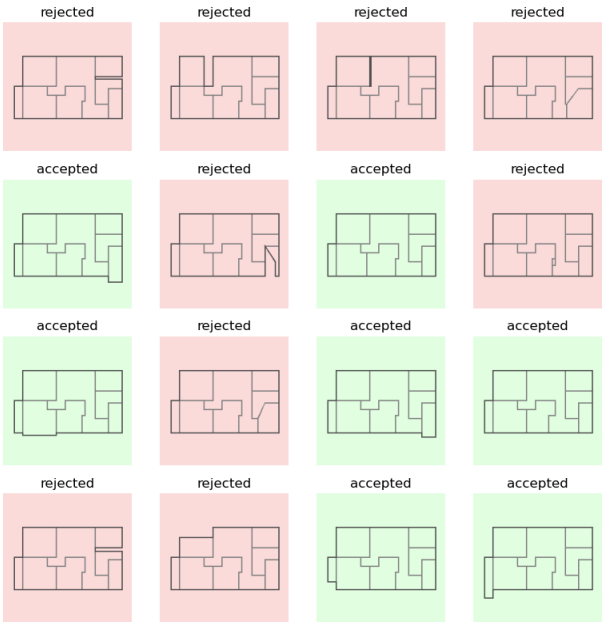


Figure 4. Visualization of accepted and rejected wall move proposals by the heuristic acceptance function. The accepted proposals generally look plausible, while the rejected proposals do not.

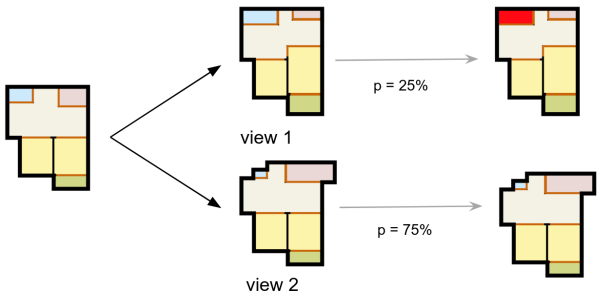


Figure 5. Illustration of generating a pair of GeomPerturb augmented views of a floor plan with masking. First, GeomPerturb is used to generate two augmented views. Then for each view, a randomly selected room is masked (in red) with a probability of 75%.

strong augmentations leads to better results. Let $G_{i,k}$ with $1 \leq k \leq 50$ be the k-th precomputed GeomPerturb augmented version of floor plan i. Then, during training, a pair of floor plan images x_1, x_2 can be sampled in the following way: $x_1 = t_1(G_{i,k_1})$ and $x_2 = t_2(G_{i,k_2})$ with random image space augmentations t_1 and t_2 , and k_1 and k_2 sampled from the indices of precomputed GeomPerturbed images of floor plan i.

3.3.2 Category masking

In addition to the geometric augmentation, we introduce a room masking strategy to improve the learned embeddings. While GeomPerturb keeps the floor plan graph identical, two floor plans should also be considered similar if not all the room types match, for instance if a living room is switched for a bedroom. Otherwise, the model would not learn that two floor plans with the same room shapes and connections, but with the function of a room swapped, should also be considered similar. The GeomPerturb augmentation is modified by randomly masking the room category of one of the rooms, hypothesizing that the model will learn that room categories don't always have to be identical for floor plans to be similar. The idea of masking a random room is inspired by the method of masking random patches in masked auto encoders [12, 18]. Figure 5 illustrates the process of first sampling two GeomPerturb augmented views, and then masking one room with a probability of 75%.

3.4. Weakly-supervised pairs

Instead of creating input pairs by generating two augmented versions of the same sample, it is also possible to train on pairs of samples from the dataset.

3.4.1 SimSiam with GED pairs

The GeomPerturb method resulted in representations that align quite well with the floor plan graph, and less with the IoU score. This led us to ask what would happen if we directly created the training pairs based on a graph edit distance (GED) metric, which measures the dissimilarity between graphs by the number of edits needed to change one into another. Previous work has used IoU pairs to train GNN embedding models [22, 26]. To the best of our knowledge, however, GED has not been used yet as a metric for obtaining training pairs in the context of a representation learning model.

GED pair generation To generate GED pairs, we propose an approximate filtering-verification scheme based on Weisfeiler-Lehman subgraph hashing. The proposed scheme does not find all similar graphs, but speeds up finding a subset of similar pairs. Note that as training data, it is not necessary to find all similar pairs but just a large enough amount.

WL graph hashing Weisfeiler-Lehman (WL) graph hashes [30] are a graph-kernel based method for computing hashes of graphs which have the following properties: the hashes are identical for isomorphic graphs, and hashes are likely to be different for non-isomorphic graphs. The graph hashes are computed in iteratively. In every iteration,

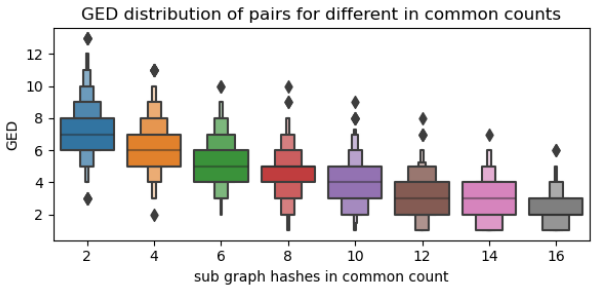


Figure 6. Plot showing the correlation between high *WL subgraph hashes in common count* and low graph edit distance (GED).

each node gets labeled based on its previous hash, as well as the hashes of adjacent nodes. The concatenation of the previous iteration's hash and the hashes of adjacent nodes is used to assign a new hash to each node. The final graph hash is computed by hashing a sorted list of the individual node hashes.

WL subgraph kernel The Weisfeiler-Lehman subgraph kernel is built on counting common intermediate subgraph hashes between two graphs. In the WL-kernel on two graphs as defined in [30], the WL-kernel is computed as the inner product of the two vectors that count how often each subgraph hash occurs in each graph. In this study, we use a slight variation, which counts how many subgraph hashes occur in both graphs, counting each subgraph only once.

Filtering based on WL graph kernel Using this WL subgraph kernel, subsets of pairs can be generated that are likely to have low graph edit distance. Figure 6 shows the correlation between pairs with high number of subgraphs in common and low GED. For the most likely subsets, the GED is computed for each pair. Training pairs can then be generated by thresholding by the computed GED. The benefit of this filtering scheme is that the number of GED computations is reduced substantially compared to enumerating all possible pairs, while still yielding enough pairs to be useful for training.

The result of Algorithm 1 is a list of candidate GED pairs with computed GED value. Training pairs can now be selected by filtering based on the computed GED. We trained both on pairs with GED ≤ 1, i.e. only include pairs that differ at most one edit operation. As well as on GED ≤ 2 pairs that differ by at most 2 graph edit operations.

3.4.2 SimSiam with IoU and SSIG pairs

In addition to generating training pairs by selecting based on graph edit distance, we also train with pairs selected based on IoU as well as SSIG. In [15,26] graph matching network

Algorithm 1 Procedure for generating GED pairs

- Compute dictionary of unique floor plan graphs by their WL hash
- For each floor plan graph, compute WL subgraph hashes
- For each subgraph hash, create a hash bucket of all the floor plan IDs that share the subgraph hash
- Create a dictionary mapping from pair to an in-common count
- Loop over all hash buckets, and increment the count dictionary item for each pair
- Now sort the pairs into subsets based on their in-common count
- For the subsets with the highest in-common counts, compute the GED for each pair
- Threshold the GEDs to generate a list of pairs based on GED
- Map unique floor plan graphs back to the original floor plans that share the same graph

models were trained on IoU. SSIG [38] was recently proposed, partly with the aim to be used for training floor plan retrieval models.

3.5. Evaluation

The learned embeddings will be evaluated in a retrieval setting. Previous work [26] used user studies in which Mechanical Turk workers were asked to indicate which retrievals they considered relevant given a query. Instead, we use retrievals generated for a set of $N = 100$ queries by pairwise computation of IoU, GED, and SSIG metrics as ground truth. Even though in practice pairwise computation of especially GED is infeasible for retrieval given a large dataset, it is feasible to generate ground truth retrievals for a limited set of queries within a reasonable amount of time.

Using IoU, GED, and SSIG retrievals to determine the relevancy of a retrieval gives insights into how well the retrievals from the models align with metrics that have previously been proposed to measure similarity between floor plans. IoU measures shape overlap, GED measures the distance between floor plan graphs, and SSIG [38] is a combined metric.

3.5.1 Mean Average Precision (MAP@R)

MAP@R [23] is chosen as the evaluation metric for its properties of being a single value that is more informative than other metrics [23, 37]. MAP@R is a variant of MAP where only the first R retrievals are taken into account for each query, with R set to the number of relevant items in the dataset for that query. To compute the MAP@R, first the AP@R is computed for each of the $N = 100$ randomly sampled queries, and then the MAP@R is the mean of the AP@R scores. If all relevant items are contained within the first R retrievals, the AP@R score is 1. Otherwise, the AP@R is equal to the average of the precision scores for each relevant retrieval.

For each query:

$$AP@R = \frac{1}{R}\sum_{i=1}^{R} P(i) \cdot \text{rel}(i) \tag{1}$$

where: $P(i)$ = the precision at i

$$\text{rel}(i) = \begin{cases} 1 & \text{if the ith retrieval is relevant} \\ 0 & \text{otherwise} \end{cases}$$

The MAP@R is the mean over AP@R for each query in the dataset:

$$\text{MAP@R} = \frac{\sum_{q=1}^{Q} \text{AP@R}(q)}{Q} \tag{2}$$

where: Q = the number of queries

3.5.2 Relevancy based on ground truth retrievals

The relevancy of a retrieval is based on ground truth retrievals generated from pairwise evaluation of the IoU, GED, and SSIG metrics. For each of the 100 evaluation queries, the floor plans in the database are ranked by pairwise computation of IoU, GED, and SSIG. For each metric, the top K highest ranked retrievals are labeled as ground truth relevant for that query. Setting the value of K makes an assumption on how many relevant retrievals exist in the dataset for each query, when ranked by these metrics. We compute the evaluation metrics with the top 5, 10, and 50 highest ranked ground truth retrievals labeled as relevant.

In some cases, especially for GED, items can have the same metric score. This happens for instance when many floor plans share the same graph, and thus have a GED of zero. When multiple floor plans have the same metric score, it is arbitrary which ones are labeled as relevant. To fix this, any floor plan for which the computed metric score equals that of the k-th retrieval is also labeled as relevant.

Relevancy labels Thus, formally, for a relevancy *metric* $\in \{\text{IoU}, \text{GED}, \text{SSIG}\}$ and a single query q, let L_k^* be the k-th ground truth retrieval ranked by *metric*. Then *metric*(q, L_K^*) is the relevancy metric score of the k-th ground truth retrieval.

Any retrieved floor plan that is at least as relevant as the k-th ground truth retrieval is considered relevant:

$$\text{rel}_{\textit{metric}}(L_i, q) = \textit{metric}(q, L_i) \geq \textit{metric}(q, L_K^*)^{[1]} \tag{3}$$

where: L_i = the ith retrieval from the model given query q

3.5.3 Floor plan similarity metrics

We define metrics in the following manner:

$$metric(i, j) : \mathcal{F} \times \mathcal{F} \to \mathbb{R}^+ \tag{4}$$

where $\mathcal{F}$ is the set of floor plan indices in the dataset. For each floor plan index $i \in \mathcal{F}$, g_i is the floor plan graph. $X_i \in \mathcal{C}^{H \times W}$ is the categorical image representation of that floor plan with $\mathcal{C} = \{c \in \mathbb{Z} \mid 0 \leq c \leq 9\}$ the set of room categories with 0 representing the background category.

GED Metric The graph edit distance (GED) between two floor plan graphs g_i and g_j is the minimum cost of changing floor plan graph g_i, into floor plan graph g_j. Each addition, removal, or change of nodes or edges incurs the same edit cost of 1:

$$\text{GED}(i, j) = \min_{(e_1, \ldots, e_k) \in \pi(g_i, g_j)} k \tag{5}$$

where $\pi(g_i, g_j)$ denotes the set of possible edit paths that transform g_i into g_j and each e_n is an edit operation that changes a vertex or edge in g_i.

IoU Metric Intersection over Union (IoU) is defined as the area where the room classes are equal, divided by the area of the union of the two floor plans. The background class, which has value 0, is not taken into account. Thus, IoU is defined as:

$$\text{IoU}(i, j) = \frac{\sum_k \mathbb{1}(X_{ik} = X_{jk} \neq 0)}{\sum_k \mathbb{1}(X_{ik} \neq 0 \vee X_{jj} \neq 0)} \tag{6}$$

where $\mathbb{1}(\cdot)$ is an indicator function, and the summation is over all pixels.

[1] For the GED metric the $\geq$ sign is swapped for $\leq$.

Rotation and flip invariant IoU_{D4} The rotation of a floor plan is somewhat arbitrary, as it depends on which direction is north. In addition, in this evaluation, we choose to regard mirrored version of the same floor plan as similar because they are compositionally equivalent [2].

In RPLAN walls are all axis aligned, and thus are invariant to rotations and flips, only 90deg rotations and horizontal and vertical flips need to be taken into account. To construct an IoU metric that is invariant to rotations and flips, we take inspiration from [7] and define

$$\text{IoU}_{\text{D4}}(x_1, x_2) = \max_{t \in \mathcal{D}_4} \{\text{IoU}\left(t(x_1), x_2\right)\}, \tag{7}$$

as the maximum IoU over the 8 possible rotation and flip combinations of the $\mathcal{D}_4$ symmetry group.

SSIG Metric Recent work proposed Structural Similarity by IoU and GED (SSIG) [38] as a metric for measuring floor plan similarity. SSIG is a weighted combination of GED and IoU, and should capture floor plan similarity better than either GED or IoU alone [38]. The equation for SSIG is given by:

$$SSIG(i,j) = \frac{\text{IoU}(i,j) + (1 - \left(\frac{\text{GED}(i,j)}{|N_i| \cdot |N_j|}\right)^{\gamma}}{2}, \tag{8}$$

where $|N_i|$ depicts the number of rooms of floor plan i, and $\gamma = 0.4$ is a constant to balance the relative influence of IoU and GED.

When the SSIG metric is based on IoU_{D4} instead of on the untransformed IoU, it is written as SSIG_{D4}.

3.6. Baselines

SimSiam with image augmentations only We compare the GeomPerturb augmentations to training SimSiam with image augmentations only, as this is the original use case for SimSiam. The augmentations used in this baseline are the same as those used for GeomPerturb, and when training with weakly-supervised pairs, see Sec. 3.2.

Graph2vec. Graph2vec [24] is Doc2Vec like model that can learn representation vectors of graphs. In Doc2Vec, the δ dimensional vectors are learned for a vocabulary of words, and the aim is to maximize the similarity between a document and the words it contains. Similarly, Graph2Vec learns δ dimensional representations of a vocabulary of Weisfeiler-Lehman subgraphs of a certain depth that occur in the training data, and fit representations of graphs that maximize the similarity between the vector of a graph and the subgraphs it contains.

[2]The authors are aware that floor plans with mirrored or differently oriented layouts still can feel different to the building user.

This baseline uses a graph representation of floor plans, where the rooms are nodes and adjacent rooms are connected by edges. Graph2Vec was chosen as a baseline, because it predicts a 128-dimensional vector embedding for each graph, and thus, like the SimSiam models, can also be used for fast nearest neighbor search in the embedding space.

LayoutGMN. LayoutGMN has been used for predicting floor plan similarity, and can be used for floor plan retrieval. LayoutGMN predicts similarity on pairs of floor plan graph representations, employing a graph matching network to identify similar nodes across the graphs. To retrieve similar floor plans given a query, the similarity has to be predicted using the GMN model for each pair of query and candidate graph.

Brute force baselines. It is possible to obtain retrievals by pairwise computation of the IoU, GED, and SSIG metrics. While in practice these methods are often too slow for fast retrieval, especially computing the graph edit distance is computationally expensive, they can serve as a comparison for what the best retrievals available in the dataset are given the metric. These baselines are the same as those used to label which retrievals are relevant.

For the IoU brute-force baseline, the IoU is computed pairwise between each query and candidate floor plan image. For the GED brute-force baseline, the graph edit distance is computed pairwise for each query and candidate floor plan graph. For the SSIG brute-force baseline, the IoU and GED are computed pairwise for each query and candidate. Retrieval scores for the brute-force baselines are shown in Appendix A.

SSIG top 50 IoU Retrievals based on the SSIG metric, are based on first filtering by the top 50 floor plans with highest IoU, and then sorting these based on SSIG. This filtering-based SSIG method is claimed to be faster than LayoutGMN [38], which is why it is included in the comparison.

The difference between the SSIG top 50 IoU baseline, and the brute-force SSIG baseline, is that the graph edit distance does not need to be calculated for all pairs of query and candidate floor plan, but only for the candidates that have the highest IoU with respect to the query. Computing the IoU is reasonably fast, although still magnitudes slower compared to nearest neighbor search in an embedding space.

4. Results

4.1. GeomPerturb results

Table 2 shows that training with only image augmentations leads to retrievals that are not relevant when evalu-

ated against ground truth GED retrievals. With GeomPerturb, the MAP@R score increases from 1.8 to 10.6. Adding the masking strategy further improves the MAP@R score to 17.8. Thus, with a specially designed data augmentation for floor plans, it is possible to improve the retrieval results of a CNN based representation learning model.

Image augmentations should be used in combination with GeomPerturb. Training without image augmentations leads to a 4.1 MAP@R score for GED ground truth relevancy labels, compared to 10.6 when training with both GeomPerturb and image augmentations, or 17.8 with GeomPerturb + masking and image augmentations.

4.2. Weakly supervised pairs

Table 3 shows that training with GED pairs considerably improves the MAP@R retrieval performance on GED ground truth retrievals compared to GeomPerturb (Tab. 2). Training with GED pairs also performs better on the SSIG and SSIG_{D4} metrics, and performs similarly to training with GeomPerturb + masking on the IoU and IoU_{D4} metrics. Thus, training with GED pairs selected from the dataset performs better than training with the proposed GeomPerturb augmentation.

An interesting observation is that training with both GED pairs and GeomPerturb – with or without masking – leads to worse performance compared to training with just GED pairs. Thus, the GeomPerturb augmentation does not add useful variation to the training pairs on top of selecting by low graph edit distance.

GED training pairs threshold Table 3 shows that training with GED pairs based on a threshold of less than or equal to 2 performs better than GED pairs with a threshold less or equal to 1. This is somewhat surprising, as pairs with a GED of 2 are intuitively less similar and thus less relevant than those with a GED of 1. It suggests that training with a more varied set of pairs increases performance, instead of only using pairs with a GED of 0 or 1.

IoU rotation invariance Even though the models are trained with rotation and flip augmentations, the MAP@R of the SimSiam model trained on IoU pairs is higher on the IoU compared to on the IoU_{D4} metric. This indicates, that although the model is trained to be rotation invariant, it is biased towards floor plans with high IoU in the original orientation.

SSIG based training pairs Surprisingly, training with SSIG pairs gives worse performance on both the SSIG and SSIG_{D4} ground truth metrics, compared to training with GED pairs. This indicates that in order to retrieve floor plans that are relevant with respect to SSIG and SSIG_{D4},

	GED 10	IoU 10	IoU_{D4} 10	SSIG 10	SSIG_{D4} 10
Image aug only	1.8	3.5	2.6	3.4	1.9
GeomPerturb w/o img aug	**4.1**	**17.1**	**7.6**	**14.9**	**6.4**
GeomPerturb	**10.6**	**8.3**	**6.4**	**14.5**	**9.5**
GeomPerturb + masking	**17.8**	**5.2**	**5.5**	**14.2**	**13.2**

Table 2. MAP@R for SimSiam models trained with different data augmentation strategies. The top K=10 ground truth retrievals generated using GED, IoU, IoU_{D4}, SSIG, and SSIG_{D4} metrics are labeled as relevant. The table shows that training with GeomPerturb + masking significantly improves retrieval MAP@R for GED compared to training with image augmentations only.

	GED 10	IoU 10	IoU_{D4} 10	SSIG 10	SSIG_{D4} 10
GED pairs ≤ 1	**22.8**	**9.6**	**7.0**	21.2	**19.3**
GED pairs ≤ 2	**33.8**	8.0	6.5	**25.4**	**22.9**
GED pairs ≤ 1 + masking	22.0	8.3	6.5	20.0	16.9
GED ≤ 2 + GeomPerturb + mask	**31.5**	6.9	5.6	**24.2**	**20.7**
SSIG pairs	12.4	**14.2**	**10.5**	**22.3**	17.6
IoU pairs	3.3	**26.4**	**16.5**	15.2	8.7

Table 3. MAP@R for SimSiam models trained with weakly supervised pairs. The table shows that training with GED pairs improves performance on the GED metric compared to GeomPerturb. Training with GED pairs also leads to better performance on the SSIG metrics compared to training with SSIG pairs. Training with IoU pairs gives the highest performance on the IoU metric.

retrieving floor plans with low GED is more important than retrieving plans with high IoU. Additionally, an explanation for this could be that the SSIG training pairs are selected by first filtering on the top 50 highest IoU floor plans, and are thus less diverse than the GED training pairs.

4.3. Comparison to baseline methods

LayoutGMN The LayoutGMN model, while performing slightly better than SimSiam with GED pairs on the IoU metric (8.8 vs 8.0), scores significantly lower on both the GED metric (4.6 vs 33.8), and the SSIG metric (11.6 vs 25.4). Compared to Graph2Vec, LayoutGMN performs slightly lower on SSIG top 10, but slightly better on SSIG top-5 (see Tab. 6). A limitation to keep in mind is that our reproduction of the LayoutGMN might perform worse than in the original paper due to differences in training; alternatively, Appendix B shows a qualitative comparison to LayoutGMN based on retrievals presented in the LayoutGMN paper.

Graph2Vec Section 4.3 shows that on the GED metric, the Graph2Vec baseline outperforms SimSiam trained with GED pairs. However, the difference in MAP@R performance is not large (36.2 vs 33.8), showing that an image based model can learn embeddings that are similar in terms of performance for retrieving floor plans based on graph

	GED 10	IoU 10	IoU_{D4} 10	SSIG 10	SSIG_{D4} 10
SimSiam w/ GeomPerturb + masking	**17.8**	5.2	5.5	14.2	13.2
SimSiam trained on GED pairs	**33.8**	8.0	**6.5**	**25.4**	**22.9**
LayoutGMN	4.6	**8.8**	5.3	11.6	6.7
Graph2Vec	**36.2**	2.5	2.5	13.7	19.1
SSIG top 50 IoU	-	**40.3**	**14.1**	**67.6**	**23.3**
SSIG_{D4} top 50 IoU_{D4}	-	**13.1**	**37.8**	**17.9**	**61.4**

Table 4. Comparison of MAP@R scores of SimSiam models to baseline methods. The table shows that SimSiam trained on GED pairs performs better than LayoutGMN on almost all metrics. Additionally, SimSiam w/ GED pairs outperforms Graph2Vec on the SSIG and SSIG_{D4} metrics, while performing slightly worse on GED. Notably, while the SSIG filtered by top 50 IoU, leads significantly on the SSIG metric, SimSiam w/ GED pairs shows comparable performance on the SSIG_{D4} metric, which accounts for different orientations across floor plans.

similarity, compared to Graph2Vec, a general method for learning embedding vectors from graphs.

SimSiam trained on GED pairs outperforms Graph2Vec on the IoU and IoU_{D4} metrics, as well as on the SSIG and SSIG_{D4} metrics. This implies that the SimSiam model has an inherent bias towards visually similar floor plans, even though it is trained on floor plan pairs based solely on similarity of the floor plan graphs. This indicates that just as LayoutGMN is a graph based model trained on IoU pairs, it similarly makes sense to train a CNN based model on pairs derived from the floor plan graph.

SSIG top 50 IoU The SSIG top 50 IoU based retrievals perform well on the SSIG ground truth metric. However, the performance drops significantly on the SSIG_{D4} metric, which takes the highest IoU of flipped and rotated versions, instead of just the IoU in the original orientation. This highlights the importance of aligning the retrieval method with the evaluation method, particularly the relevance of flipped or rotated floor plans when using SSIG as a heuristic for floor plan similarity.

When there is a mismatch in considering rotations and flips – computing either the highest IoU from flipped and rotated version or only in the original orientation – the performance of SSIG/SSIG_{D4} top 50 IoU based methods becomes similar to that of the SimSiam model trained on GED pairs. This suggests that when it is uncertain if floor plan orientation is important to the user, the SimSiam trained on GED pairs model performs competitively.

It was not possible to calculate the MAP@R scores on the GED metric for the SSIG-based retrieval methods. Labeling candidate floor plans with the same or lower GED as the 10th best retrieval relevant, often leads to more than 50 relevant retrievals. This makes it impossible to compute MAP@R, as the number of relevant documents is higher than the number of retrievals.

4.4. Effect of data preprocessing steps for weakly supervised pairs

The GeomPerturb method needs specialized preprocessing of the floor plan geometry, see Fig. 1. The same preprocessing steps were applied for training with weakly supervised positive pairs as well, for a fair comparison. Table 5 shows that the MAP@R score with the default preprocessing steps is similar to the MAP@R score obtained without the geometry preprocessing steps. The run labeled "cat img" uses the categorical image from RPLAN directly, as visualized in the left most image of Fig. 1. The run labeled "rplanpy rgb", uses floor plans drawn with the same RGB colors as those used in the default preprocessing, but without altering the wall geometries. The rplanpy [40] library was used for drawing this "rplanpy rgb" version of the dataset.

This result implies that training with GED pairs is not dependent on specific preprocessing, making the method applicable to other floor plan datasets, for which both a floor plan graph, and floor plan images are available.

4.5. Qualitative evaluation

Figure 7 shows a comparison of the same two queries across different models and brute-force baselines. The retrievals are rotated to the orientation with the highest Intersection over Union (IoU) with respect to the query for easier visual inspection. The original orientation is indicated by the line and square overlaid on the image; when in the original orientation, the line is at the bottom and the square is at the top left.

The query in the left figure of Fig. 7 highlights the importance of taking flips and rotations into account. All the retrievals based on the brute-force SSIG_{D4} metric are flipped or rotated with respect to the query floor plan. The SimSiam with GED pairs model, which is trained on pairs formed by GED, and thus irrespective of room positions, provides the most relevant retrievals for this query.

5. Conclusion

The representation learning framework SimSiam was applied to floor plan retrieval in two novel ways: a data augmentation that perturbs the underlying geometry of the floor plan, and training based on pairs supervised by graph edit distance on the floor plan graph. In contrast to previous methods using graph-based GNN models and weakly supervised IoU pairs, this work demonstrates that training a CNN model with pairs supervised by graph edit distance is a feasible alternative. In addition, the results highlight that it is advantageous to be able to retrieve flipped or rotated versions of the same floor plan, as these often occur in the RPLAN dataset. A limitation of this work is that the floor plan images have a uniform style, which is not the

	GED			IoU			IoU_{D4}			SSIG			SSIG_{D4}		
	5	10	50	5	10	50	5	10	50	5	10	50	5	10	50
SimSiam trained on GED pairs	**36.8**	**33.8**	**31.0**	**10.3**	**8.0**	**5.6**	**8.4**	**6.5**	**4.9**	**25.9**	**25.4**	**19.3**	**25.1**	**22.9**	**22.6**
SimSiam GED pairs cat. img	32.4	30.1	28.7	10.2	8.3	5.4	7.7	6.1	4.5	23.5	23.3	16.9	21.4	20.3	19.0
SimSiam GED pairs rplanpy rgb	**38.6**	**34.9**	**33.1**	8.1	6.3	4.7	7.1	5.2	3.8	**24.6**	23.0	**18.4**	**22.4**	**21.1**	**19.8**

Table 5. The table shows that MAP@R is similar for different preprocessing methods when trained on GED pairs. With relevancy labels based on labeling the top 5, 10, and 50th ground truth retrievals generated by GED, IoU, IoU_{D4}, SSIG, and SSIG_{D4} metrics relevant.

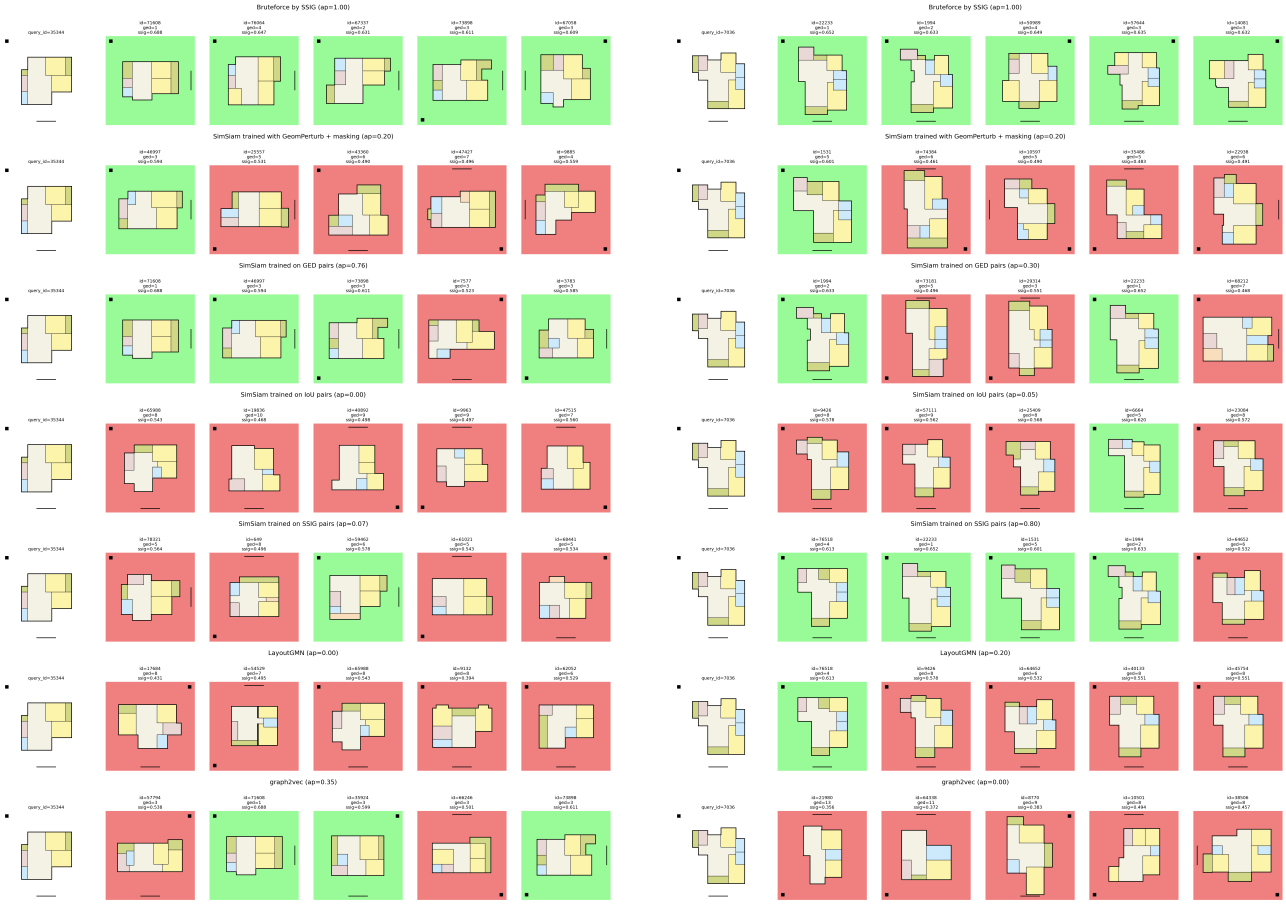


Figure 7. Retrieval comparisons for two randomly sampled query floor plans. The average precision (ap) is calculated based on SSIG_{D4} as relevance metric, with an adaptive threshold based on the SSIG_{D4} value of the top 50th brute-forced SSIG_{D4} retrieval. Retrievals are highlighted as relevant (green) or not relevant (red). In the original orientation, the dot is at the top left, and the line is at the bottom.

case in general for real world floor plans drawn by architects. Future work could investigate if it is feasible to train a model on floor plan images collected in the wild with GED as a learning signal, in order to retrieve structurally similar floor plans end-to-end. This work shows that CNN models trained with graph edit distance can be competitive on the floor plan retrieval task. This exploratory research opens up a new avenue to approach end-to-end floor plan retrieval based on CNN models as an alternative to having to parse a graph representation at inference time.

References

[1] Yunsheng Bai, Haoyang Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2018. 2

[2] Jane Bromley, James W. Bentz, Léon Bottou, Isabelle M

Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. *Int. J. Pattern Recognit. Artif. Intell.*, 7:669–688, 1993. 3

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. 1, 3, 4, 5

[4] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15745–15753, 2020. 1, 3, 4

[5] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1:539–546 vol. 1, 2005. 3

[6] Joon Son Chung, Jaesung Huh, Seongkyu Mun, Minjae Lee, Hee-Soo Heo, Soyeon Choe, Chiheon Ham, Sung-Ye Jung, Bong-Jin Lee, and Icksang Han. In defence of metric learning for speaker recognition. In *Interspeech*, 2020. 3

[7] Taco S. Cohen and Max Welling. Group equivariant convolutional networks, 2016. 9

[8] Sean Gillies et al. Rasterio: geospatial raster i/o for Python programmers, 2013–. 3

[9] Sean Gillies, Casper van der Wel, Joris Van den Bossche, Mike W. Taves, Joshua Arnott, Brendan C. Ward, and others. Shapely, Oct. 2023. 5

[10] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised Learning. 3, 4

[11] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2:1735–1742, 2006. 3

[12] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners, 2021. 6

[13] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9726–9735, 2019. 3

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 4

[15] Jiongchao Jin, Zhou Xue, and Biao Leng. Shrag: Semantic hierarchical graph for floorplan representation. *2022 International Conference on 3D Vision (3DV)*, pages 271–279, 2022. 2, 7

[16] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. 7(3):535–547. 1, 2, 3

[17] 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 *Scandinavian Conference on Image Analysis*, pages 28–40. Springer. 2

[18] Xiangwen Kong and Xiangyu Zhang. Understanding masked image modeling via learning occlusion invariant feature. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6241–6251, June 2023. 6

[19] Philip K Lee and Björn Stenger. Shape-based floor plan retrieval using parse tree matching. In *2021 17th International Conference on Machine Vision and Applications (MVA)*, pages 1–5. IEEE. 2

[20] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects, 2019. 2

[21] 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. 2

[22] Dipu Manandhar, Dan Ruta, and John Collomosse. Learning structural similarity of user interface layouts using graph networks. In *ECCV*. 2, 6

[23] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check, 2020. 3, 8

[24] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs, 2017. 9

[25] Hyejin Park, Hyegyo Suh, Jaeil Kim, and Seungyeon Choo. Floor plan recommendation system using graph neural network with spatial relationship dataset. *Journal of Building Engineering*, 2023. 2, 3

[26] Akshay Gadi Patil, Manyi Li, Matthew Fisher, Manolis Savva, and Hao Zhang. LayoutGMN: Neural Graph Matching for Structural Layout Similarity. 2, 3, 6, 7, 16

[27] Filip Radenovic, Giorgos Tolias, and Ondřej Chum. Fine-tuning cnn image retrieval with no human annotation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41:1655–1668, 2017. 3

[28] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 3

[29] Divya Sharma, Nitin Gupta, Chiranjoy Chattopadhyay, and Sameep Mehta. Daniel: A deep architecture for automatic analysis and retrieval of building floor plans. *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, 01:420–425, 2017. 2

[30] Nino Shervashidze and Karsten Borgwardt. Fast subtree kernels on graphs. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. 6, 7

[31] Chia-Ying Shih and Chi-Han Peng. Floor plan exploration framework based on similarity distances. *ArXiv*, abs/2211.07331, 2022. 2

[32] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. 3

[33] Kihoon Son and Kyung Hoon Hyun. A framework for multivariate data based floor plan retrieval and generation. 2

[34] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation from predicting 10,000 classes. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1891–1898, 2014. 3

[35] Yuki Takada, Naoto Inoue, T. Yamasaki, and Kiyoharu Aizawa. Similar floor plan retrieval featuring multi-task learning of layout type classification and room presence prediction. *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6, 2018. 2

[36] Yuandong Tian, Xinlei Chen, and Surya Ganguli. Understanding self-supervised learning dynamics without contrastive pairs, 2021. 3

[37] Vipin Tyagi. Content-based image retrieval. *Springer Nature*, 2017. 2, 8

[38] Casper van Engelenburg, Seyran Khademi, and Jan van Gemert. Ssig: A visually-guided graph edit distance for floor plan similarity, 2023. 2, 3, 7, 9, 15

[39] Kilian Q Weinberger, John Blitzer, and Lawrence Saul. Distance metric learning for large margin nearest neighbor classification. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press. 3

[40] Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. Data-driven interior plan generation for residential buildings. *ACM Trans. Graph.*, 38(6), nov 2019. 3, 11

[41] Mantaro Yamada, Xueting Wang, and T. Yamasaki. Graph structure extraction from floor plan images and its application to similar property retrieval. *2021 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–5, 2021. 2

[42] Mang Ye, Xu Zhang, PongChi Yuen, and Shih-Fu Chang. Unsupervised embedding learning via invariant and spreading instance feature. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6203–6212, 2019. 3

[43] Chaoning Zhang, Kang Zhang, Chenshuang Zhang, Trung Xuan Pham, Chang Dong Yoo, and In So Kweon. How does simsiam avoid collapse without negative samples? a unified understanding with self-supervised contrastive learning. *ArXiv*, abs/2203.16262, 2022. 3

[44] Zhijian Zhuo, Yifei Wang, Jinwen Ma, and Yisen Wang. Towards a unified theoretical understanding of non-contrastive learning via rank differential mechanism. *ArXiv*, abs/2303.02387, 2023. 3

A. Additional results

Table 6 shows additional results, including brute-force baselines, which retrieve based on pairwise computation of the proposed evaluation metrics. The brute-force baselines show how optimal retrieval, for one of the evaluation metrics, scores on the other evaluation metrics.

In addition, Tab. 6 includes MAP@R scores for relevancy labels based on marking the top 5, 10, and 50th ground truth retrievals relevant. The top 5 and top 10 are results are most relevant for evaluating the retrieval quality, as top 50 is only relevant if users of the system are interested in looking at 50 retrievals per query.

Brute-force baselines Table 6 shows that for each brute-force baseline, the MAP@R score on the same ground truth metric is 100. This is expected, as all ground truth relevant retrievals, are returned by each brute-force baseline.

Effect of rotation and flip invariance Comparing the results of retrieving by IoU evaluated on $\mathrm{IoU}_{\mathrm{D4}}$ shows that the MAP@R score drops to around 30, implying that when rotated and flipped versions of a floor plan are also considered to be relevant, retrieving based on IoU leaves performance on the table, by not including flipped or rotated versions.

The other way round, retrieving based on $\mathrm{IoU}_{\mathrm{D4}}$, and evaluating on IoU, only labeling floor plans with high IoU in the original orientation as relevant, leads to an even worse MAP@R score of around 15. Thus, it is important to know whether the user is interested in only floor plans with similar orientations, or also in flipped or rotated versions.

SSIG baseline The "SSIG top 50 IoU" method proposed in [38] filters candidate floor plans on IoU, and only computes the GED and SSIG for the top 50 IoU candidates. Its high MAP@R scores of around 60, show that this seems to be a reasonable compared to full pairwise computation of SSIG. However, the large drop in MAP@R to around 20, when evaluated on rotation and flip invariant $\mathrm{SSIG}_{\mathrm{D4}}$, highlights that the reliance on IoU can be a limitation when the query floor plan has a different orientation.

On RPLAN, the small set of rotations and flips from the $\mathcal{D}_4$ symmetry group achieves invariance to flips and rotations, due to the axis aligned nature of RPLAN floor plans. While SimSiam could be adapted to arbitrary rotations through data augmentation at training time, extending SSIG to handle more arbitrarily rotated floor plans would be challenging without greatly increasing the computational cost. Specifically, computing IoU for many different orientations, instead of just four 90 degree rotated versions, would significantly slow down the filtering stage.

B. Qualitative comparison to LayoutGMN

In their paper, LayoutGMN includes retrieval examples with RPLAN IDs shown in the figure. These allow for a fair comparison irrespective of our reproduction of LayoutGMN performing as well as the original implementation. Figure 8 shows a comparison of LayoutGMN retrievals (left) to retrievals of a SimSiam model trained on GED ≤ 2 pairs (right). As the preprocessing pipeline shown in Fig. 1 does not work for all RPLAN images, the SimSiam model in this comparison is trained on the segmented RPLAN images without preprocessing.

	GED			IoU			IoU_{D4}			SSIG			SSIG_{D4}		
	5	10	50	5	10	50	5	10	50	5	10	50	5	10	50
Bruteforce by GED	*100.0*	*100.0*	*100.0*	*2.0*	*1.5*	*1.4*	*3.5*	*3.3*	*3.0*	*13.2*	*15.0*	*13.6*	*28.4*	*29.3*	*27.7*
Bruteforce by IoU_{D4}	*6.2*	*5.1*	*4.4*	*19.8*	*15.7*	*12.1*	*100.0*	*100.0*	*100.0*	*10.4*	*9.1*	*10.1*	*24.0*	*24.6*	*30.9*
Brutefoce by untransformed IoU	*4.3*	*3.0*	*2.4*	*100.0*	*100.0*	*100.0*	*30.0*	*28.2*	*27.0*	*27.3*	*28.3*	*37.0*	*12.3*	*11.8*	*12.2*
Bruteforce by SSIG_{D4}	*55.2*	*47.5*	*38.0*	*10.2*	*8.6*	*7.0*	*24.8*	*25.1*	*28.5*	*21.9*	*22.6*	*21.5*	*100.0*	*100.0*	*100.0*
Bruteforce by SSIG (untransformed IoU)	*29.3*	*25.6*	*20.3*	*27.3*	*27.1*	*33.6*	*13.2*	*11.7*	*13.3*	*100.0*	*100.0*	*100.0*	*31.8*	*34.4*	*34.1*
SimSiam trained with image augmentations only	2.7	1.8	1.7	4.3	3.5	2.6	3.3	2.6	1.9	3.4	3.4	2.9	2.7	1.9	2.0
SimSiam trained with GeomPerturb + masking	**19.2**	**17.8**	**17.9**	7.1	5.2	4.2	6.7	5.5	4.2	15.4	14.2	12.4	15.0	13.2	14.2
SimSiam trained on GED pairs	**36.8**	**33.8**	**31.0**	10.3	8.0	5.6	8.4	6.5	4.9	**25.9**	**25.4**	19.3	**25.1**	**22.9**	**22.6**
SimSiam trained on IoU pairs	4.4	3.3	3.4	**29.0**	**26.4**	**26.0**	**18.0**	**16.5**	**20.9**	15.0	15.2	**19.4**	8.7	8.7	12.7
SimSiam trained on SSIG pairs	13.3	12.4	15.2	**15.8**	**14.2**	**12.7**	11.4	10.5	12.0	**23.9**	**22.3**	**23.3**	16.9	17.6	**22.1**
LayoutGMN	6.1	4.6	4.3	10.7	8.8	10.2	6.3	5.3	4.3	14.1	11.6	13.6	7.1	6.7	6.1
Graph2Vec	**46.6**	**36.2**	**20.6**	3.5	2.5	1.5	3.5	2.5	2.2	12.0	13.7	10.6	19.2	19.1	16.2
SSIG top 50 IoU	-	-	-	**35.3**	**40.3**	**100.0**	**15.5**	**14.1**	**19.2**	**71.0**	**67.6**	**47.8**	**23.3**	**23.3**	19.1
SSIG_{D4} top 50 IoU_{D4}	-	-	-	13.2	13.1	11.8	**33.6**	**37.8**	**100.0**	18.8	17.9	12.1	**63.7**	**61.4**	**42.7**

Table 6. Mean average precision at R (MAP@R), where R is the amount of relevant retrievals for each query in the dataset, based on GED, IoU, IoU_{D4}, SSIG, and SSIG_{D4} ground truth relevance labels. Retrievals are labeled relevant if they are at least as relevant as the k-th ground truth retrieval per query, shown for $k \in \{5, 10, 50\}$. For 100 randomly sampled evaluation queries.

Figure 8. Comparison of LayoutGMN retrievals taken from fig. 15 of the LayoutGMN [26] paper (left) to retrievals of the SimSiam model trained on GED≤ 2 pairs (right). The SimSiam model is trained on the category channel of RPLAN without additional preprocessing ("SimSiam GED pairs cat. img" in Tab. 5).

3

Background

3.1. Deep Learning

Deep learning is a subfield of machine learning that uses neural networks to approximate functions. Traditional machine learning methods usually require manual feature engineering, that is, extracting features from raw data to use as input. This process can be time-consuming and may not capture all the relevant information. In contrast, deep learning methods can be trained end-to-end, with the neural network acting as a feature extractor. The 'deep' in deep learning refers to the multiple layers that deep neural networks have. This depth is essential to deep learning's success in training models in many domains, ranging from image and speech recognition to natural language processing.

Neural networks are general function approximators. That means that given some target function $y = f(x)$, a neural network can be trained to approximate $\hat{f}_\theta(x)$. The target function is usually not known, but given implicitly by a dataset of training examples $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.

A neural network consists of many individual neurons, which have learnable parameters that can be trained. Neurons are grouped together in layers, and layers are connected to each other to form a network.

3.1.1. Multi Layer Perceptron

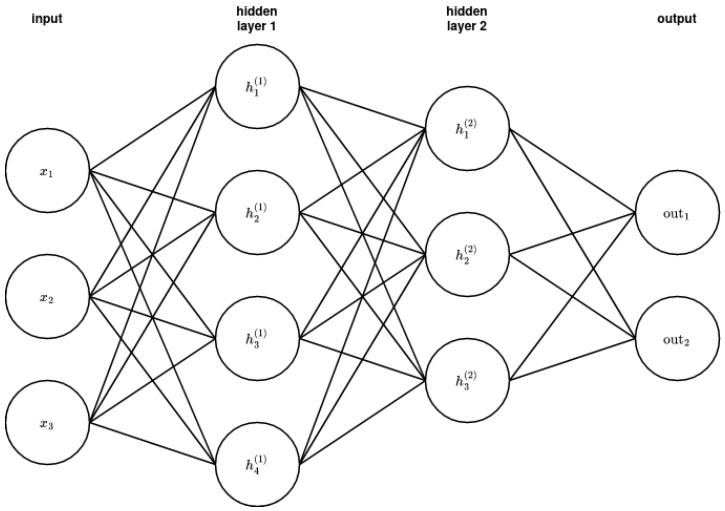


Figure 3.1: Illustration of an MLP with two hidden layers.

A Multi Layer Perceptron (MLP) [17] is composed of input and output layers, and one or more hidden layers. Each layer is composed of neurons. Neurons are functions that take the outputs of neurons of

the previous layer as input, and compute a new value based on a set of parameters. The output of the MLP is computed by the last layer.

Figure 3.1 shows an MLP with two hidden layers. The input layer has 3 neurons, the first hidden layer has 4 neurons, the second hidden layer has 3 neurons, and the output layer has 2 neurons. MLPs have fully connected layers, that means that each neuron takes as input the output of all the neurons in the previous layer.

The learnable parameters of a neuron consist of weights for each incoming connection and a bias term. Let's take the first neuron of the first hidden layer as an example. It has three incoming connections, one from each input neuron. This neuron thus has three weights and one bias. Its output is computed as follows:

$$h_1^1 = f(w_{1,1}^{(1)} \cdot x_1 + w_{1,2}^{(1)} \cdot x_2 + w_{1,3}^{(1)} \cdot x_3 + b_1^{(1)}). \tag{3.1}$$

Here $w_{1,i}^{(1)}$ is the weight for the i-th incoming neuron of the first neuron of the first hidden layer, and $b_1^{(1)}$ is the bias term for the first neuron of the first hidden layer. The function f is an activation function to introduce non-linearity, see section 3.1.1.

MLP layer as matrix multiplication Equation (3.1) shows how to compute the output of a single neuron in an MLP layer. MLP layers usually have many neurons. The notation for a weight, e.g. $w_{1,3}$, already hints that the weights of an MLP layer can be stored as a matrix. The first index identifies which neuron it is in the layer, and the second index is the incoming connection the weight should be applied to. Thus, the weights of a fully connected MLP layer are an $N \times M$ matrix where N is the number of neurons in the layer, and M is the number of neurons in the previous layer.

The output of an MLP layer can be computed as $f(Wx+b^1)$ where W is the NxM weight matrix, and x is the $M \times 1$ dimensional output vector of the previous layer. The result of the $W \cdot x$ matrix multiplication as an $N \times 1$ dimensional vector, i.e., one value for each of the N neurons in the layer. The bias term likewise is also an $N \times 1$ dimensional vector, namely one bias for each neuron, and can thus be added to the result of the weight matrix multiplication.

Activation functions

Without activation functions, an MLP would just be a weighted sum of weighted sums, i.e., a linear function. Thus, to learn more complex relations, a non-linear function needs to be introduced somewhere. This is the purpose of the activation function, f in equation (3.1), that gates the output of each neuron. There exist many possible activation functions. The most commonly used are ReLU, sigmoid, and tanh. Figure 3.2 illustrates the behavior of these functions.

3.1.2. Training a neural network

A randomly initialized neural network is very unlikely to approximate the target function well. Thus, the neural network should be trained on the dataset to learn to approximate the target function. In order

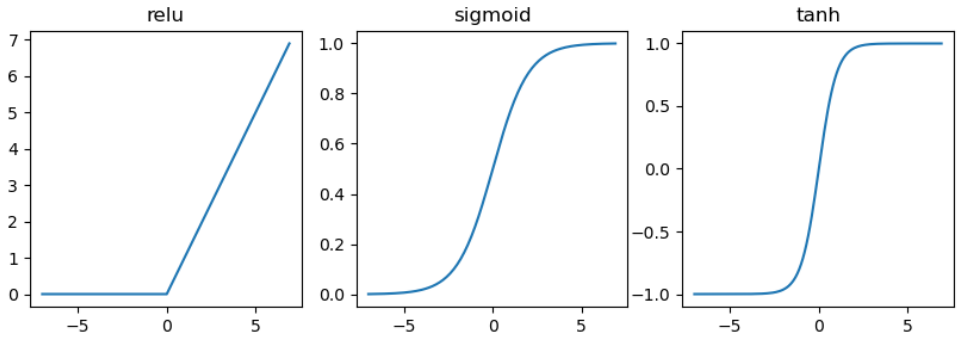


Figure 3.2: Illustration showing the behavior of the ReLU, sigmoid and tanh activation functions for input between -7 and 7.

to optimize the parameters of a neural network, a loss function should be defined. A loss function is a function of the model parameters and the training examples that measures how well the model approximates the target function. The goal of training is to find parameters that minimize the loss function.

Loss functions

For regression problems, where the output is a real valued number, mean squared error is a common loss function. For classification problems, where the output is one of multiple discrete classes, and the task is to predict the class of the input image, cross entropy is a common loss function.

The mean squared error loss,

$$MSE(y, \hat{y}) = \sum_{i=0}^{N} (y_i - \hat{y}_i)^2, \tag{3.2}$$

computes the mean L2 distance between the target values in the dataset, y_i, and the values predicted by the network, $\hat{y}_i$.

Optimization

Now that we have defined a neural network with trainable parameters, a loss function, and we have a dataset, we can optimize the neural network to learn to fit the data. The common method for optimizing a neural network is through gradient descent and backpropagation [17]. Gradient descent entails computing the derivative of each neuron's weights with respect to the loss function. Backpropagation is a technique to efficiently compute the gradient of the weights w.r.t. the loss, and update the weights.

The loss function is evaluated by computing the output of the network for some input features x for a pair of training data (x, y), and computing the loss, e.g., for MSE loss: error $= MSE(y, f_\theta(x))$. Here, y is the ground truth label for input x, and $f_\theta(x))$ is the prediction by the network.

Gradient descent For different weights, a neural network will give a different output. If you plotted the weights of the network versus the loss on the training data, you could imagine this as a hillscape of peaks and valleys. See figure 3.3 for an illustration for a model with only two weights, in practice neural networks can have millions of weights, making it infeasible to visualize. Here, valleys have a low loss and are desirable. The dimensionality of the weights of a neural network is large, and thus it would be inefficient to compute the loss value for many different weight configurations and choose the weights that yield the lowest loss value. Luckily, neural networks are differentiable, and thus the slope of points in the loss hillscape can be computed. Knowing the slope means that you know in which direction to step to go to a point with a lower loss value. The slope of the loss function is also known as the gradient, and stepping in the direction of a lower loss is called gradient descent.

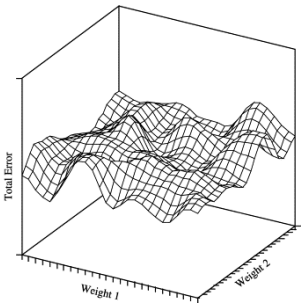


Figure 3.3: Visualization of the loss landscape for a network with two weight parameters. Taken from [23].

When optimizing the loss by making individual steps in the loss landscape, you can get stuck in a local minimum. This is an area where if you step in any direction, the loss will increase, but where the loss is not as low as possible. The amount that the weights are changed at each step, i.e., the size of the step, is called the learning rate. The learning rate should be adjusted such that the optimization process does not get stuck in small local minimums, but also not so large that it steps over valleys with a minimum loss. What is often done, is decreasing the learning rate during training, a process called learning rate decay.

Mini-Batch Stochastic Gradient Descent In the standard gradient descent method, the gradient is computed using the loss calculated over the entire dataset. Instead, weights can also be updated by computing the gradient using the loss of just a single randomly training example. Doing this iteratively for each training example in the dataset is called Stochastic Gradient Descent (SGD). It is stochastic, because the choice of a random example at each iteration introduces randomness into the process. SGD helps avoid local minima, and can lead to faster convergence for large datasets [23].

A variation of SGD that is commonly used in practice is mini-batch SGD. Mini-Batch SGD strikes a balance between the two by updating the weights based on the gradient computed on a small subset of the data. The subset is called a mini-batch. This mini-batch is randomly sampled, and the size of the mini-batch has an influence on how fast the optimization converges, as well as on how many computational resources are needed. Common choices for the mini-batch size are around 32–128 samples. It can be advantageous to use a smaller or a lot larger batch size, depending on the problem and model architecture.

Regularization

In training a neural network, the weights can take on many configurations while still predicting the training data with the same accuracy. However, not all model configurations are able to generalize well to unseen data. Regularization entails methods to change the training process to result in a ”simpler” solution space. A ”simpler” solution space means constraining the space of possible weight configurations in some way. This can be explicit, such as by using weight decay, or implicit, such as drop out or early stopping.

Weight decay Weight decay is an explicit form of regularization that works by adding a term to the loss that penalizes the norm of the weights [14]:

$$L^{'}(\theta; y, \hat{y}) = L(\theta; y, \hat{y}) + \alpha\Omega(\theta), \tag{3.3}$$

here $\Omega(\theta)$ is the regularization term based on the norm of the weights, and α controls how much the regularization term effects the loss. Usually $\Omega(\theta)$ is either the L_1 or L_2 norm of the weights θ. The L_1 norm constrains the solution space to prefer sparse solutions [14], i.e., solutions where some weights are set to zero. On the other hand, training with L_2 regularization reduces the magnitude of the weights, resulting in a model that is less likely to overfit [14].

Dropout Dropout [10] randomly sets some activations of the network to zero during training, thus dropping out individual neurons. Each dropped out network can be seen as a subnetwork of the full model. Training with dropout can be seen as a form of model averaging [10], thus making the model less prone to overfitting.

Early stopping To train a model, usually the dataset will be split into three disjoint sets: training, validation and test set. The model is trained on the training set, and the validation set is used for tuning the training process. Finally, in the end, the performance of the model is tested on the test set once, to get an unbiased estimate of the performance on unseen data from the same distribution.

While training, the training loss usually goes down, and as long as the model is not overfitting on the training data, the loss on the validation set will also go down. With early stopping [14], the validation loss is checked every epoch, and training is stopped after the validation loss stops to improve after a certain amount of epochs. Early stopping can be seen as a regularization method, which works as a stronger regularizer when the stopping condition is reached after a small number of epochs without improving the validation loss [14].

Backpropagation

When computing the output of a neural network, the input is fed to the network, and the result of each layer is computed based on the output of the previous. This is called the forward pass. Backpropagation is a method that is used to efficiently compute the gradient for each neuron w.r.t. to the loss function. Backpropagation works by first computing and storing the output of each layer during the forward pass. Then the error is computed by evaluating the loss function using the output of the network, and the loss function. Now the derivative of the final layer’s weights w.r.t. the error is easy to compute. Computing the gradient of each layer is possible by iteratively computing it using the gradient of the next layer. Thus, this method is called backpropagation.

3.1.3. Convolutional Neural Networks

Training a neural network to classify images with high accuracy takes both a large dataset, and a network with many parameters. To reduce the amount of parameters, and speed up training, it is advantageous to incorporate prior knowledge into the model. For images, training image based models, the use of Convolutional Neural Networks (CNN), was popularized by AlexNet [12] who improved classification performance on a large scale image dataset using a fast GPU based implementation of CNNs. Instead of representing each pixel by an input neuron, CNNs learn convolution filters, which are applied to each pixel in an image. CNNs thus encapsulates the prior knowledge that information in images is often translation invariant, and that many image features are local.

An example of translation invariance is that to detect a cat, the position of the cat in the image does not matter. An example of the locality of image features is that to detect the presence of an edge at a pixel location in the image, only the surrounding pixels are needed to detect the edge. In contrast, a fully connected network, such as an MLP, would require the model to learn analogous pixel relationships from the data. Thus, an MLP would need more parameters, computation, and potentially a larger training dataset to achieve the same accuracy as a CNN based model.

CNN filters A convolution filter is a square kernel matrix, that is slid across the image. Figure 3.4 shows how such a kernel is applied pixel wise to the input. To compute the output of the convolution at a pixel location, the pixel values are element wise multiplied with the overlapping filter values, and summed together. The same filter is applied to all pixels in the image . The values of the convolution kernel values are learned, similar to neurons in an MLP.

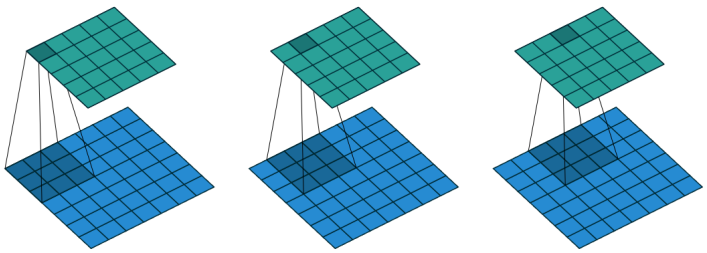


Figure 3.4: Illustration of a 3×3 convolution kernel (dark blue) sliding over a 7×7 input image (blue). The output is a 5×5 feature map (green). Illustration taken from [9].

3.1.4. Siamese representation learning

Output of a neural network as embedding vector

The output of a hidden layer of a neural network can be used as a feature extractor [2]. Take the output of an intermediate fully connected layer, and put it into a feature vector of the same length as the number of neurons in the layer. This results in a feature vector that can be used for downstream tasks such as image retrieval.

Triplet loss

Instead of using the activations of a network trained on image classification, a specialized distance metric learning loss function can be used. One of these is the triplet loss [18]. A use case for the triplet loss is face-verification. For face-verification the model is trained on triplet of anchor, positive and negative examples. The anchor and positive are faces of the same person, and the negative example is a face from a different person. The triplet loss training objective is now to learn that the distance between features vectors extracted from anchor and positive should be smaller than the distance between anchor and negative. The negative examples needed to form a triplet given an anchor are generally sampled from the dataset, excluding the positive samples of the anchor. If the distance between the predicted representation vectors for the anchor and positive is already smaller than the distance between anchor and negative, the gradient of the loss is small, and the triplet was not informative for training the model. Thus, during training, it is important to choose negative examples that are currently hard for the model [18]. Choosing negative examples that are currently hard, is called negative hard-mining.

Contrastive losses

Instead of working on triplets of anchor, positive, and negative, the contrastive loss [6] works on image pairs that are labeled as similar or not similar. An extension of the contrastive loss, is to use all pairs of training examples in a mini-batch as part of the loss computation. In N-pairs loss [21], all non-positive examples in a mini-batch are used as negative example. Using all negative examples from a batch, in combination with a large batch size can be used as an alternative to negative hard-mining [21, 4].

Siamese networks

In Siamese networks [3], two branches of the network share identical weights during training. Siamese networks can be seen as a form of incorporating prior knowledge into a model. Whereas CNNs are a prior for extracting features from images, Siamese networks are a prior for comparing entities [5]. In a contrastive learning setting, the training examples consist of pairs which are labeled as similar or not similar. In this setting, each branch of the Siamese network computes a representation vector for each of the samples in a pair. The loss function then acts on the distance between the vectors, and if the pair is labeled as similar or not. Thus, Siamese networks incorporate the prior information that similarity is symmetric, i.e., to compute the distance between the representation vectors of a training pair, the order of the examples within a pair does not matter.

3.2. Retrieval

In this chapter, the basics of building a content-based image retrieval system, based on learning representation vectors for each document, will be explained. Throughout the text, the words image and document are used interchangeably to refer to items that are retrieved.

3.2.1. Building an index and querying

The goal of an information retrieval system is to return relevant results to a user, and do this efficiently. Information retrieval systems need two essential components to facilitate fast retrieval: building an index structure, and a method for efficiently searching the index [22].

Without an index, querying would require comparing the query to every document in the database. Thus, for the goal of enabling fast retrieval, building an index beforehand is essential. For text search, an inverted index can be build analogous to the index in a book: for every word in the documents, keep track of the documents that are relevant for that word. Then sort the words alphabetically to make efficient querying possible. Text can naturally be broken down into words, making it possible to construct an inverted index. Unlike text, images are represented by as high-dimensional pixel data, necessitating a different approach: low dimensional feature vectors that characterize each image.

Vector representations

For content-based image retrieval, where images are retrieved by example, the indexing step should compress and characterize each image by a low dimensional descriptor to enable efficient querying [22]. Traditionally, this was done by using hand-crafted feature extractors, such as a color histogram [7]. A recent trend is to use learned feature descriptions based on deep learning methods [8].

Learned feature descriptors can take the form of binary hash codes or real-valued vectors. These feature vectors are the output of a deep neural network and are of low dimensionality compared to the input image. An example of a neural network that outputs a real-valued feature vector computed from an image are the Siamese networks described in section 3.1.4.

Nearest neighbor search

With an index based on vector representations, the distance between two vectors is used as a measure of similarity of the images. The problem of retrieving K images, based on a query image, is thus to return the K-nearest neighbors of the query representation vectors. To define the distance between two vectors, a commonly used metric is the L2 distance [11].

Faiss [11] is a collection of algorithms for efficient nearest neighbor search of real-valued vectors. Exact nearest neighbor search can be done efficiently for a database of up to 1̃M vectors. For even larger databases, faiss includes approximate methods for k-nearest neighbors search.

A retrieval system build on nearest neighbor search of representation vectors, is a type of *ranked retrieval system*. In a ranked retrieval system, instead of retrieving a set of documents, the documents are sorted by how relevant they are, in this case by distance of their representation vector to the representation vector of the query.

3.2.2. Retrieval evaluation

The goal of developing a retrieval system is to provide relevant documents to the user. To achieve this goal, it is essential to measure the performance of the retrieval system. To evaluate an information retrieval system, you need [13]:

- A document collection
- A set of queries
- A set of relevance labels for each <query, document> pair

The document collection is the set of all candidate documents from which the system should retrieve relevant documents. The set of queries express an information need by the user, and the relevance labels express a binary judgement for if the retrieved document is relevant to the query.

Relevance labels The relevance label of a document given a query is used as ground truth judgement for the relevance of a retrieved document given the query. The relevance of a document depends on the information need of the user. The measured performance of a retrieval system can vary a lot between different queries. Thus, it is important that the set of queries is large enough to average the performance over different queries, a set of at least 50 queries has been found to be sufficient [13].

Evaluation metrics

Many metrics exist for evaluating retrieval systems. However, not all of them paint an accurate picture of how well a retrieval system performs [16].

Precision One of the simplest metrics, is computing the ratio between how many retrieved documents are relevant, and the amount of documents that were retrieved [13]:

$$Precision = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})}. \tag{3.4}$$

Note that the precision score is calculated for a single query.

Precision@K Precision can be calculated for a retrieval system that returns a set of retrievals. To this metric to a retrieval system that instead ranks all documents, the precision of first K retrievals can be calculated instead. The precision@K is thus defined as the ratio of relevant documents within the first K retrievals:

$$P@K = \frac{\#(\text{relevant items retrieved in top K})}{K}. \tag{3.5}$$

R-precision R-precision is a variant of P@K, where K is taken as the number of relevant documents. The relevant number of documents depends on the query. The advantage of this metric, is that a perfect retrieval system, that ranks all the relevant documents higher than any non-relevant document, achieves a P@R score of 1 [13]. In contrast, a perfect retrieval system evaluated on P@20 with a query that only has 5 ground truth relevant documents, would get a score of $\frac{5}{20} = 0.25$.

$$P@R = \frac{\#(\text{relevant items retrieved in top R})}{R}, \tag{3.6}$$

where R is the number of relevant items in the document collection for the query.

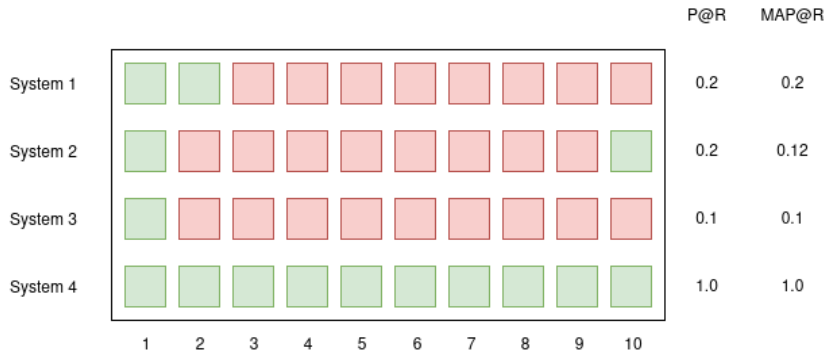


Figure 3.5: Figure showing precision@R and MAP@R scores for different retrieval systems. It is assumed that R, the amount of relevant items for the query, is equal to 10. Green is used for relevant retrievals, and red for non-relevant ones.

Mean average precision @R The precision metrics measures the system's ability to only retrieve relevant documents, without taking the order of the retrievals into account. Figure 3.5 shows an example where two systems, system 1 and system 2, have the same precision score for some query, even though system 1 is preferable. System 1 is preferable because it retrieves relevant results at higher spots compared to system 2.

A metric that incentivizes retrieving relevant results earlier, while still being a number that can be compared, is the average precision metric. Average precision is defined as [16]:

$$AP@R = \frac{1}{R}\sum_{i=1}^{R} P(i) \cdot rel(i), \tag{3.7}$$

where $P(i)$ is the prevision at i, and $rel(i)$ denotes if the i-th retrieval is relevant. Mean average precision at R (MAP@R), is the AP@R average over the query set (3.2.2).

Figure 3.5 shows that MAP@R gives a higher score for the system that returns relevant results earlier. It also shows that the MAP@R for a system that retrieves all relevant results for a given query is 1.0 (system 4).

3.3. Graph (dis)similarity

3.3.1. Graphs

A graph is a mathematical structure consisting of nodes and edges. Nodes are objects, and objects that are related to each other are connected by an edge. Edges model pairwise relations between the nodes. Thus, mathematically a graph $G = (V, E)$ is represented by nodes $v \in V$, and edges $e = (u, v) \in E$ with V the set of nodes, and E the set of edges.

Two nodes $u \in V$ and $v \in V$ in a graph are adjacent if there is an edge $(u, v) \in E$ between them. The set of neighbors $\mathcal{N}(v)$ of a node v is the set of all nodes that are adjacent to v.

Labeled graphs Nodes and edges can have labels. For example: a graph representation of a floor plan can have rooms as nodes, with the nodes of adjacent rooms connected by an edge. The node label can then be the type of room, and the edge label can indicate if the rooms are connected by a door.

The structure of a graph carries information. In the floor plan graph example, the structure conveys which rooms are adjacent.

Sub graphs A sub graph G_{sub} is a subset of the nodes $V_{\text{sub}} \subset V$ and edges $E_{\text{sub}} \subset E$ of the graph. To be a valid sub graph, all nodes which are end points of an edge $(u, v) \in E_{\text{sub}}$ should be in the nodes set of the sub graph G_{sub}.

3.3.2. Graph representations of floor plans

There are multiple feasible graph representations of floor plans. The one used in this work is the following: rooms are represented by nodes. Edges are used to represent the pairwise adjacency relation between rooms. That is, two nodes are connected by an edge if the rooms they represent have at least one wall in common. The nodes have the room type as a label. The edges have as label either *door* or *wall*, to indicate if they are only adjacent, or if a door also connects them.

An alternative graph representation could be one where edges indicate a door connection. In this case, the nodes would be the same, but the edges would be a subset of only the edges with the door label.

For visualizing the floor plan graph on top of the floor plan image, nodes can have the x and y coordinates of a representative point in the room as additional label. The graph can then be visualized by drawing the nodes in their respective rooms.

Graph isomorphism

Informally, graph isomorphism is the problem of checking if the structure of two graphs is identical. Two graphs $G_1 = (V_1, E_1)$ and $G2 = (V_2, E_2)$ are isomorphic if there exists a one-to-one mapping from the nodes in G_1 to the nodes in G_2 $f : V_1 \to V_2$ such that u and v are adjacent in G_1 if and only if $f(u)$ and $f(v)$ are adjacent in G_2.

In this work, graph isomorphism for labeled graphs is defined, such that two labeled graphs G_1 and G_2 are isomorphic if there exists a mapping $f : V_1 \to V_2$ that demonstrates isomorphism without considering labels, and the label of $u \in V_1$ is equal to the label of $f(u) \in V_2$ for all $u \in V_1$. If the graph also has edge labels, then each edge $(u, v) \in E_1$ needs to have the same label as $(f(u), f(v)) \in E_2$ for all edges $(u, v) \in E_1$ for the graphs to be isomorphic.

3.3.3. Graph edit distance

Graph isomorphism only tests if two graphs have exactly identical structure. Even though two graphs might not be identical, they can still be more similar to each other compared to a third graph. Thus, it would be nice to have a measure of similarity between graphs.

Graph edit distance is a method for computing an “edit distance” of how many elementary edit operations are needed to transform one graph into another. Each of these edit operations occurs a cost, and the graph edit distance is the minimum cost of a sequence of edit operations that transforms one graph into the other. Graph edit distance is thus a measure of a measure of dissimilarity, with a lower distance indicating a higher similarity.

Edit operations The elemental edit operations either change a single node or edge. The node operations are: inserting a new node, removing a node, or substituting a node by altering the node label. The edge operations are: inserting an edge between two nodes, removing an edge, or substituting an edge by altering the edge label without changing the endpoints.

An edit path is a sequence of edit operation that transforms some graph G_1 into another graph G_2. Each of these edit operations incurs a certain cost. The cost of an edit path is the summed cost of each edit operation. Let $c(e)$ be the cost of edit operation e. The graph edit distance is then the cost of the minimum edit path:

$$\mathrm{GED}(G_1, G_2) = \min_{(e_1,\ldots,e_k)\in\pi(g_i,g_j)} \sum_{i=1}^{k} c(e_i)$$

The cost $c(e)$ can be different for each type of edit operation. If the cost of each type instead is equal to 1, the equation can be simplified to

$$\mathrm{GED}(G_1, G_2) = \min_{(e_1,\ldots,e_k)\in\pi(g_i,g_j)} k$$

which is just the length of the edit path $(e_1, ..., e_k)$.

3.3.4. Approximate graph similarity

Computing the graph edit distance between two graphs is computationally expensive. To efficiently find pairs of graphs with common sub structures, *graph kernels* can be used to approximate the similarity between graphs. The Weisfeiler-Lehman graph kernel is a fast subtree graph kernel that counts the number of common subtree patterns based on the Weisfeiler-Lehman isomorphism test [19].

Weisfeiler-Lehman isomophism test

The Weisfeiler-Lehman (WL) isomorphism test works by iteratively relabeling the nodes in a graph based on the labels of its neighboring nodes [19]. Initially, for labeled graphs, the node labels $l_0(v)$ are set to the initial labels of the graph.

WL iteration For iterations $i > 0$, first a multiset [1] label $\mathrm{M}_i(v)$ is assigned to each node consisting of $\{l_{i-1}(u) \mid u \in N(v)\}$. The labels in $M_i(v)$ are sorted and concatenated into a string $s_i(v)$. Set $s_i(v)$ to $s_i(v)$ by $l_{i-1}(v)$. Now map each string $s_i(v)$ into a new label $l_i(v)$ using an injective function $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(u))$ if and only if $s_i(u) = s_i(v)$. In practice f can be implemented using a hash function that makes collision where $f(s_i(v)) = f(s_i(u))$ even though $s_i(u) \neq s_i(v)$ unlikely.

Isomorphism test The WL iteration is repeated h times. The labels after the final iteration can be used to test if two graphs are non-isomorphic. If $\{l_h(v) \mid v \in G_1\} \neq \{l_h(u) \mid u \in G_2\}$, then graphs G_1 and G_2 are not isomorphic. Otherwise, the test is inconclusive, although it has shown to be a valid test for isomorphism for almost all graphs [20, 1].

Extension with edge labels In the standard WL-test iteration, the string $s_i(v)$ is the concatenation of the previous iteration's labels of neighboring vertices prefixed by the previous iteration's label of the current node. To include edge labels in the isomorphism test, the multiset label $\mathrm{M}_i(v)$ consisting of neighboring nodes labels can be changed to $\mathrm{M}_i(v) = \{(l_{i-1}(u), l'(u, v)) \mid u \in N(v)\}$ where $l'(u, v)$ is the label for the edge (u, v).

Isomorphic graph hash The Weisfeiler-Lehman graph hash is defined as the hash resulting of sorting the final node labels $\{l_h(v) \mid v \in V\}$, concatenating them into a string and applying a hash function. The resulting graph hash allows for efficiently finding subsets of isomorphic graphs using a hash map data structure.

Weisfeiler-Lehman graph kernel

The definition of the Weisfeiler-Lehman kernel on two graphs G_1 and G_2 is given by [19] as:

$$k_{\mathrm{WL}}^{(h)}(G_1, G_2) = |\{(s_i(u), s_i(v)) \mid f(s_i(u)) == f(s_i(v)), u \in V_1, v \in V_2, i \in [1, h]\}|, \tag{3.8}$$

where f is injective, and $\{f(s_i(v)) \mid v \in V_1 \cup V_2\}$ and $\{f(s_j(v)) \mid v \in V_1 \cup V_2\}$ for all $i \neq j$ are disjoint. Here, the notation $|\{\cdot\}|$ denotes the size of a multiset.

Thus, the WL kernel counts common subtree labels in two graphs, using the labels produced by the first h iterations of the WL-test.

WL-kernel as inner product of feature maps The WL-kernel can also be written as an inner product of feature map $\phi_{WL}(G)$s [15]. For each WL iteration i, $\phi_i(G)$ is a feature vector in $\mathbb{N}_0^{|\Sigma_i|}$ where each $\phi_i(G)_c$ counts the number of occurrences of the label $c \in \Sigma_i$. Then with the alphabet Σ_i known beforehand, and padded with zeroes for elements that do not occur in iteration i, $\phi_{WL}(G)$ is the concatenation of $\phi_i(G)$:

$$\phi_{WL}^{(h)}(G) = [\phi_0(G), ..., \phi_h(G)].$$

[1] A multiset is a set that allows elements to occur multiple times

Then the WL-kernel for h iterations becomes:

$$k_{\mathrm{WL}}^{(h)}(G_1, G_2) = \langle \phi_{WL}^{(h)}(G_1), \phi_{WL}^{(h)}(G_2) \rangle,$$

with $\langle \cdot, \cdot \rangle$ denoting the inner product.

Alternative based on presence An alternative to counting the number of common subtree pairs is counting how many subtree labels are in common, counting each common label only once. This can be implemented by counting each pair of $(s_i(u), s_i(v))$ in equation (3.8) only once if it occurs more than once in the multiset.

The same can be accomplished by taking the element wise $\max(\phi_{WL}^{(h)}(G), 1)$ to make each element $\phi_i(G)_c$ of the feature map denote the presence of label $c \in \Sigma_i$ instead of the occurrence count of label c.

3.3.5. Filtering-verification for finding similar graphs

Computing the graph edit distance for a pair of graphs is an expensive operation. Instead of computing it on all graphs, a filtering-verification approach, which consists of two main steps: filtering and verification, can be used to efficiently find pairs of similar graphs.

Filtering-verification works in two steps: filtering out candidates pairs that are too dissimilar to pass the verification step, and then verifying which of the remaining pairs are below some graph edit distance threshold. When the filtering-verification method finds all pairs would pass the verification step, it is called complete [24].

Depending on the application, it might not be necessary to find all the pairs that would pass the verification step. The rest of this section will go into an approximate filtering-verification method based on the Weisfeiler-Lehman graph kernel.

Filtering based on WL graph kernel

The output of the WL graph kernel is the count of common subtree patterns between two graphs. The filtering step consists of computing the WL kernel for each pair of graphs, and sorting the pairs decreasingly on the kernel output. For the verification step, group the graphs by the WL-kernel output. For each group in descending order, compute the graph edit distance for all graphs in the group, and add the pairs with a GED below some threshold to a list of generated pairs. Continue with the next group until some compute budget is reached, or enough pairs have been generated.

References

[1] Laszlo Babai and Ludik Kucera. “Canonical labelling of graphs in linear average time”. In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. 1979, pp. 39–46. DOI: 10.1109/SFCS.1979.8.

[2] Artem Babenko et al. *Neural Codes for Image Retrieval*. 2014. arXiv: 1404.1777 [cs.CV].

[3] Jane Bromley et al. “Signature Verification Using A ”Siamese” Time Delay Neural Network”. In: *Int. J. Pattern Recognit. Artif. Intell.* 7 (1993), pp. 669–688. URL: https://api.semanticscholar.org/CorpusID:16394033.

[4] Ting Chen et al. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. DOI: 10.48550/ARXIV.2002.05709. URL: https://arxiv.org/abs/2002.05709.

[5] Xinlei Chen and Kaiming He. “Exploring Simple Siamese Representation Learning”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 15745–15753. URL: https://api.semanticscholar.org/CorpusID:227118869.

[6] Sumit Chopra, Raia Hadsell, and Yann LeCun. “Learning a similarity metric discriminatively, with application to face verification”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)* 1 (2005), 539–546 vol. 1. URL: https://api.semanticscholar.org/CorpusID:5555257.

[7] Thomas Deselaers, Daniel Keysers, and Hermann Ney. “Features for image retrieval: An experimental comparison”. In: *Inf. Retr.* 11 (Apr. 2008), pp. 77–107. DOI: 10.1007/s10791-007-9039-3.

[8] Shiv Ram Dubey. “A Decade Survey of Content Based Image Retrieval Using Deep Learning”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 32.5 (2022), pp. 2687–2704. DOI: 10.1109/TCSVT.2021.3080920.

[9] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285 [stat.ML].

[10] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. arXiv: 1207.0580 [cs.NE].

[11] Jeff Johnson, Matthijs Douze, and Hervé Jégou. “Billion-scale similarity search with GPUs”. In: *IEEE Transactions on Big Data* 7.3 (2019), pp. 535–547.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012).

[13] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008. ISBN: 0521865719.

[14] Reza Moradi, Reza Berangi, and Behrouz Minaei. “A survey of regularization strategies for deep models”. In: *Artif. Intell. Rev.* 53.6 (Aug. 2020), pp. 3947–3986. ISSN: 0269-2821. DOI: 10.1007/s10462-019-09784-7. URL: https://doi.org/10.1007/s10462-019-09784-7.

[15] Christopher Morris et al. *Weisfeiler and Leman go Machine Learning: The Story so far*. 2023. arXiv: 2112.09992 [cs.LG].

[16] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. *A Metric Learning Reality Check*. 2020. arXiv: 2003.08505 [cs.CV].

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

[18] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A Unified Embedding for Face Recognition and Clustering”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. DOI: 10.1109/cvpr.2015.7298682. URL: https://doi.org/10.1109%2Fcvpr.2015.7298682.

[19] Nino Shervashidze and Karsten Borgwardt. “Fast subtree kernels on graphs”. In: *Advances in Neural Information Processing Systems*. Ed. by Y. Bengio et al. Vol. 22. Curran Associates, Inc., 2009. URL: https://proceedings.neurips.cc/paper_files/paper/2009/file/0a49e3c3a03ebde64f85c0bacd8a08e2-Paper.pdf.

[20] Nino Shervashidze et al. “Weisfeiler-Lehman Graph Kernels”. In: *J. Mach. Learn. Res.* 12.null (Nov. 2011), pp. 2539–2561. ISSN: 1532-4435.

[21] Kihyuk Sohn. “Improved Deep Metric Learning with Multi-class N-pair Loss Objective”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf.

[22] Vipin Tyagi. “Content-based image retrieval”. In: *Springer Nature* (2017).

[23] D.Randall Wilson and Tony R. Martinez. “The general inefficiency of batch training for gradient descent learning”. In: *Neural Networks* 16.10 (2003), pp. 1429–1451. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(03)00138-2. URL: https://www.sciencedirect.com/science/article/pii/S0893608003001382.

[24] Xiang Zhao et al. “A Partition-Based Approach to Structure Similarity Search”. In: *Proc. VLDB Endow.* 7.3 (Nov. 2013), pp. 169–180. ISSN: 2150-8097. DOI: 10.14778/2732232.2732236. URL: https://doi.org/10.14778/2732232.2732236.