Interactive LSTM-Based Design Support in a Sketching Tool for the Architectural Domain

Floor Plan Generation and Auto Completion Based on Recurrent Neural Networks

Johannes Bayer[1], Syed Saqib Bukhari[1] and Andreas Dengel[1]

[1]*German Research Center for Artificial Intelligence, Trippstadter Str. 122, Kaiserslautern, Germany*

{johannes.bayer, saqib.bukhari, andreas.dengel}@dfki.de,

Keywords: Archistant, Archistant WebUI, LSTM, Early Design Phases, Architectural Support

Abstract: While computerized tools for late design phases are well-established in the architectural domain, early design phases still lack widespread, automated solutions. During these phases, the actual concept of a building is developed in a creative process which is conducted manually nowadays. In this paper, we present a novel strategy that tackles the problem in a semi-automated way, where long short-term memories (LSTMs) are making suggestions for each design step based on the user's existing concept. A design step could be for example the creation of connections between rooms given a list of rooms or the creation of room layouts given a graph of connected rooms. This results in a tightly interleaved interaction between the user and the LSTMs. We propose two approaches for creating LSTMs with this behavior. In the first approach, one LSTM is trained for each design step. In the other approach, suggestions for all design steps are made by a single LSTM. We evaluate these approaches against each other by testing their performance on a set of floor plans. Finally, we present the integration of the best performing approach in an existing sketching software, resulting in an auto-completion for floor plans, similar to text auto-completion in modern office software.

1 INTRODUCTION

Similar to other engineering disciplines, architecture makes use of different iterative strategies for the generation of floor plans during early design phases in construction projects. One established approach in this area is the so-called *room schedule* (also referred to as architectural program), i.e. a high level description of the building (often given by the building contractor or customer). A room schedule can be either a list of rooms only (denoted by their room function like living, working, sleeping, etc.) or a graph of rooms (i.e. a set of rooms along with restrictions how rooms should be connected or at least placed adjacent to each other). Some room schedules already include restrictions regarding the sizes and shapes of individual rooms.

Given a room schedule, the architects task is to develop an actual floor plan. This is nowadays usually done in a manual and iterative manner. E.g. semi-transparent sketching paper is being written on with pencils. When putting on a new sheet of semi-transparent paper on top of the old one, the old sketch serves as a template for the next, refining design interaction. While a fair amount of this task is highly creative, many actions remain rather repetitive and monotone for the architect.

In this paper, we describe a semi-automated approach for drafting architectural sketches. In section 2, we outline the existing technologies and concepts on which our approach is based on. In section 3, we outline the presentation of floor plans for recurrent neural networks. In section 4 we describe our approach, i.e. how our models are trained and how floor plan drafts are extended and completed using our trained models. We also point out some of the problems we encountered during the design of our system and how the trade-offs we made attempt to solve them. After that, in section 5 we outline how we integrated our approach an existing sketching software. We evaluate our approach in section 6, where we present the results of an automatically conducted performance evaluation on a set of floor plans and provide examples of real-value outputs of our trained models and illustrate the use of the integrated system. We conclude our work in section 7 where we also give an outlook on our future work.

2 RELATED WORK

2.1 The Long-Short Term Memory

Long-Short Term Memories (Hochreiter and Schmidhuber, 1997), (Gers et al., 2000) are a class of recurrent artificial neural networks. During each time step, they are supplied with an input vector of arbitrary (but fixed) length, while they emit an output vector of a size equivalent to their amount of cells. Stacked with an MLP, they may also return a vector of also arbitrary (but fixed) size. As vector sequence processing units, LSTMs are capable of large variety of different tasks like prediction of future events as well as memorizing and transforming information. The components of their input vectors have to be normalized a certain the interval (often $[0,1]$). Likewise, their output values are limited to a certain interval (often $[0,1]$). The experiments described have been conducted using the OCRopus LSTM implementation (Breuel, 2008).

2.2 The Architectural Design Support Tool Archistant

Archistant (Sabri et al., 2017) is an experimental system for supporting architects during early design phases in the search of floor plans similar to an entered sketch. It consists of a web front-end, the Archistant WebUI, and a modular back-end, in which floor plans are processed between the individual modules via the AGraphML format.

2.2.1 Archistant WebUI

The Archistant WebUI (see Figure 1, also formerly known as Metis WebUI (Bayer et al., 2015)) provides functionality for sketching floor plans in an iterative way. The workflow of the Archistant WebUI employs the room schedule working method as it exists in architecture. Every aspect of a room may be specified as abstract or specific as intended by the user and the degree of abstractness may be altered by the user during his work. This continuous refinement allows for a top-down work process, in which a high-level building description is transformed into a specific floor plan. The Archistant WebUI comes as a web application and runs inside a JavaScript-supporting browser.

2.2.2 AGraphML

AGraphML (Langenhan, 2017) is Archistant's exchange format for floor plan concepts. AGraphML is a specification of GraphML (Brandes et al., 2013), hence the floor plans are described in a graph-based manner: each room is represented by a node in the graph. Hence, attributes of a room are implemented as node attributes. Likewise, graph edges model the connections between rooms (see Table 1).

Type	Description
Wall	Rooms share a uninterrupted wall only
Door	Rooms connected by door
Entrance	Rooms connected via a reinforced door
Passage	Rooms connected by a simple discontinuity in a wall

Table 1: Edge Types in AGraphML.

3 ENCODING FLOOR PLANS FOR RECURRENT NEURAL NETWORK PROCESSING

This sections outlines the representation of floor plans used for processing by (recurrent) neural networks. In the current status of our work, we restrict ourselves to a limited set of attributes, that are incorporated in our LSTM models and graph representations: Room functions, connections between rooms, room layouts (i.e. a polygon which is representing a rooms surrounding walls), and information whether or not natural light is available in a room or not.

A floor plan has to be described by a sequence of vectors, that are being processed one after another by an LSTM. There are several requirements to the floor plan representation in our scenario: Both sequence length and vector size should be as small as possible in order to minimize learning and productive execution time. The vectors should be easy to interpret, and the actual information should be separated by data-less so-called control vectors (their purpose will be explained later). Most important, the information flow in the vector sequence should mimic the actual workflow an architect may have when designing a floor plan. Consequently, abstract information should precede specific information, i.e. declaration of all rooms along with their room functions should be before the declaration of the actual room layouts.

3.1 Blocks

A complete floor plan description in our chosen representation consists of 3 different blocks:

1. Room Function Declarations.
2. Room Connections.

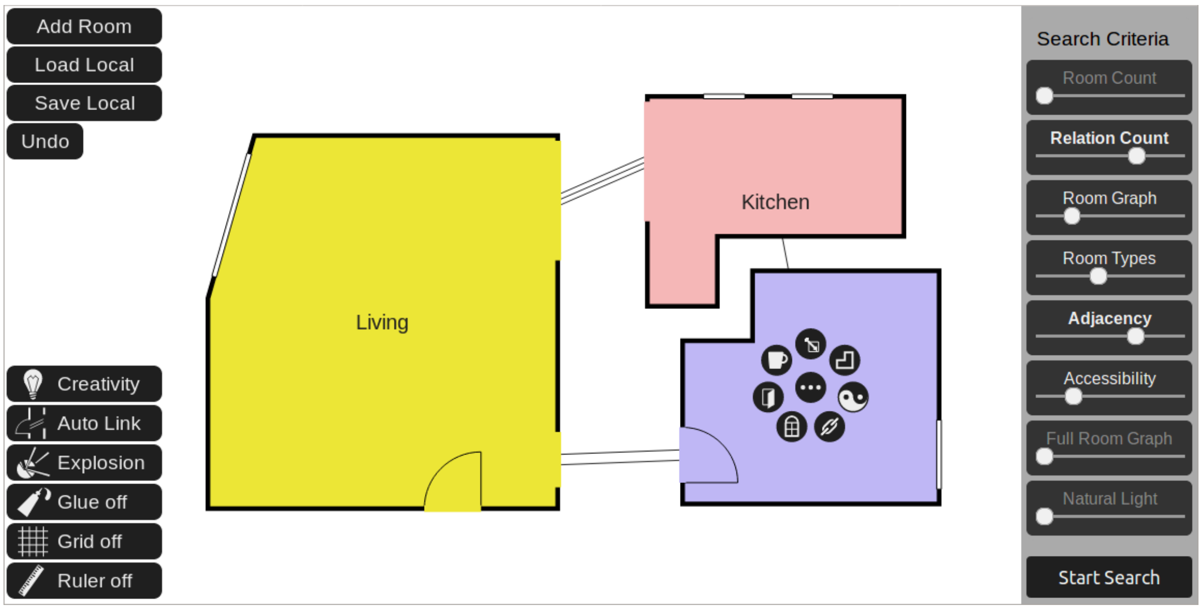

Figure 1: Screenshot of the Archistant WebUI.

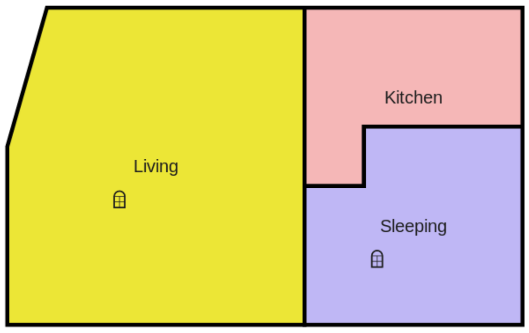

Figure 2: Rendered Image of a Sample Floor. Window Symbols indicate access to natural light. The detail level shown in this image equals the information represented in the feature vectors.

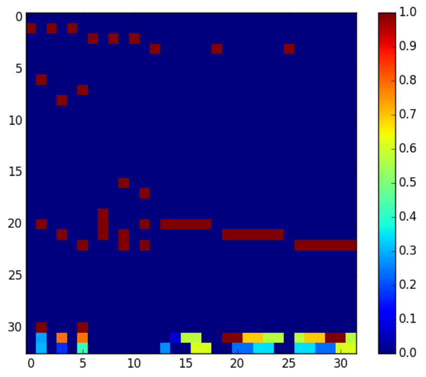

Figure 3: Feature Vector Sequence of the Same Sample Floor Encoding. Every feature vector occupies one column. The encoding consists of three blocks. The first block ranges from column 0 to 5 and defines the rooms with IDs (row 20-29), the room function, the room's center position (row 31 and 32), and whether or not a room has access to natural light (row 30). In Block 2 (column 6-12) connections between the rooms are defined. In block 3 (column 13-31) the polygons of the room surrounding walls are described.

3. Room Geometry Layouts.

Each block consists of a number of tags of the same kind, where each tag provides a piece of information. Each tag is represented by a number of vectors. An example for a rendered floor plan along with its representation as a sequence of vectors can be seen in Figure 2 and Figure 3 respectively.

3.2 The Feature Vector

A feature vector in the context of this paper can be considered as structured into several channels as follows (see Figure 4 for the relation between channels and actual feature vector components):

- The blank channel indicates that no information are present (used to indicate start and ending of floor plans or to signal the LSTM to become active)
- The control channel indicates that a new tag begins and what type the new tag is
- The room type channel (room types are called room functions in architecture)

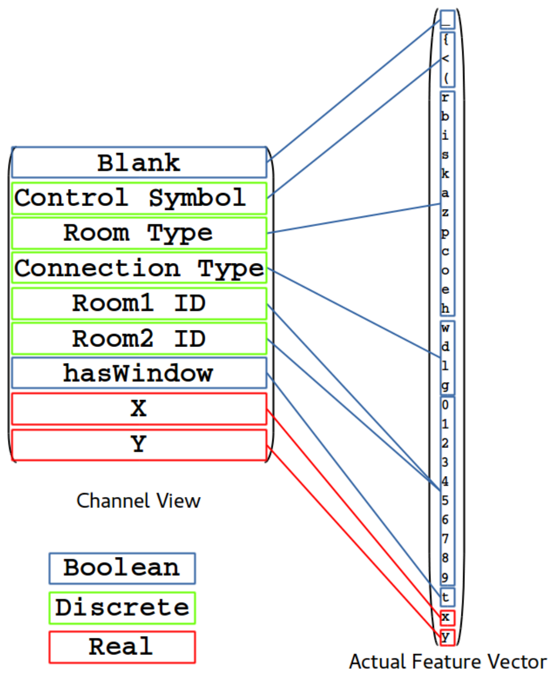

Figure 4: Structure of the Feature Vector.

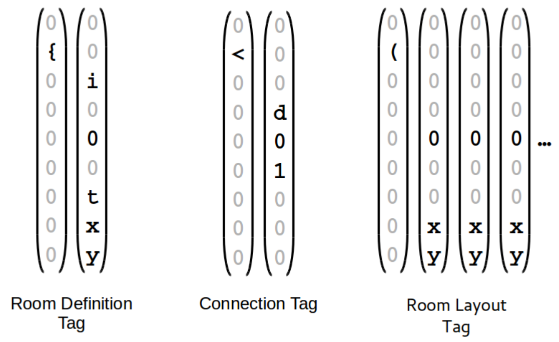

Figure 5: The different Tag Types in Feature Vectors Representation (channel view). Left: Definition of a Living (`i`) Room with ID (`1`) with a Window (`t`) with a center at position (**x**,**y**). Middle: Definition of a door connection (`d`) between rooms `0` and `1`. Right: Definition of the polygon layout of room `0`.

- The connection type channel
- The room ID channel is used to declare or reference an individual room
- A second ID channel is used for connection declaration
- has Window property
- X Ordinate of a Point
- Y Ordinate of a Point

The components, which encodes a point's position are real values between 0 and 1, all other components are boolean (0 for false, 1 for true). 1-Hot Encoding has originally been considered, but is inefficient (the final, tight encoding we have chosen is around 88 percent smaller than our experimental 1-Hot encoding). So in order to minimize the sequence length, each feature vector may contain several information. As an disadvantage, the finally chosen, tight encoding only allows for a fixed upper limit of rooms that has to be determined before training (we decided to allow for 10 rooms).

3.3 Tags

Tags are the atomic units a floor plan is consists of. At the same time, they can be considered actions carried out by an entity (like the user) to build up a floor plan. Every tag is represented by a set of successive feature vectors. All tags start with a so-called control vector solely indicating the tags type. Figure 5 illustrates how the different tag types are made up from feature vectors. Currently, there are three different types of tags:

3.3.1 Room Definition Tags

These tags define the very room by assigning it an ID as well as a room type, a flag indicating whether or not the room has a window, and the position of its center. This tag always occupies 2 feature vectors.

3.3.2 Connection Tags

Connection Tags declare the connection between rooms. They consist of the references between the two connection partners and the connection type. The connection types equals the ones from AGraphML. This tag always occupies 2 feature vectors.

3.3.3 Room Layout Tags

These tags define a polygon surrounding walls around a room. This tag occupies $p+1$ feature vectors, where p is the number of corners in the polygon.

3.4 Room and Connection Order

The order of rooms and connections underlies a trade-off: a sorted order of rooms and connections only allows for one representation of the same floor plan. By allowing for random order, the actual user behavior is better approximated and there are multiple representations of the same floor plan (many samples can be created from one floor plan). However, when considering connections to be given in a random fashion, an LSTM, that should predict them given a set of room definitions, is hard to train. Since a random order of room definitions and connections adds an unpredictable noise to the LSTM, we decided the order or rooms and connections as follows:

3.4.1 Room Order

The order in which the rooms are given is determined by the center position of the room within the floor plan. A room which center has a smaller X ordinate appears before before a room with a greater X center ordinate. In case of the centers of two rooms share the same X ordinate, the room with the smaller Y ordinate precedes the other room. The order of rooms is the same for block 1 and block 3.

3.4.2 Connection Order

The order of connections is determined by the order of the rooms. At this point, the connection graph is considered to be directed and that the source room IDs are always smaller than target room IDs. If two connections have different source room IDs, the one with the lower source room ID will come before the one with the higher source room ID. If two connections have the same source room ID, the connection with the lower target room ID will precede the connection with higher target room ID.

4 PROPOSED MECHANISM OF AUTOCOMPLETION OF FLOOR PLANS USING LSTM

In this section, we present our proposed algorithm for expanding and completing of floor plans. Here, we outline the modus operandi, with which we hand existing parts of floor plans the (LSTM) model and retrieve new parts.

4.1 LSTM Input and Output Sequences

The structure of the model's input vector is identical to the structure of its output vector and are both referred to as sequences of feature vectors here. In this paper, we examine two different approaches: The block generation sequencers and the vector prediction sequencers.

4.1.1 Block Generation Sequencers

Block generation sequencers follow a simple pattern: The first n blocks are given to the model's input. Simultaneously, the model's output is simply a series

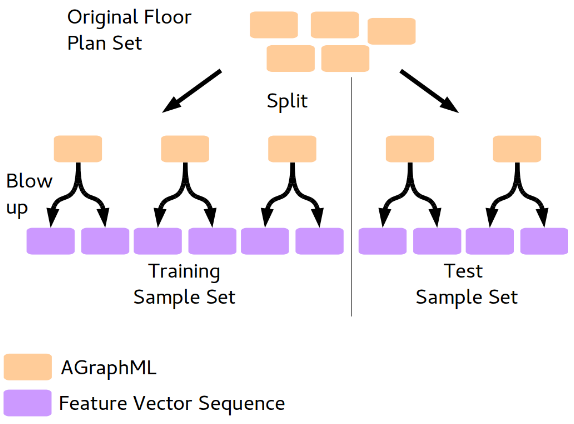

Figure 6: Sample Preparation.

of blank vectors (the blank component is 1, while all other components are 0). Afterwards, a sequence of blank vectors is used as input while the $n+1th$ block is emitted by the model (eventually finished by a blank vector). As a result, there has to be one model per block in order to allow for the support of the user during the entire work flow. Additionally, a model for supporting the first design step cannot be created.

4.1.2 Vector Prediction Sequencers

Vector prediction sequencers aim to predict the n-th vector of a sequence given the first $n-1$ sequence vectors. Vector prediction sequencers are trained by using the vector sequence of a floor plan as the input of the model and the same sequence (shifted by one position to the past) as output. The existing gaps at the begin of the input sequence (and at the end of the output sequence) are filled with blank vectors.

4.2 Preparation of Database

In order to make use of the limited set of floor plans available in AGraphML, some preprocessing is applied to generate a training sample set as well as a test set (see Figure 6). We omitted a validation set since the amount of floor plans he had available was very limited and we knew from previous experiments with a similar DB that overfitting is not a serious issue in our situation. First of all, the sample set is split into two disjoint subsets. Each floor plan is now converted into b different samples (we refer to this process as blow-up and to b as blow-up factor). A sample is derived from a floor plan by rotating all point of the floor plan (centers, corner points) by an random vector and then create the feature vector sequence as described so far. During this step, the center and corner points of the floor plan are also normalized to the

$[0,1]^2$ space. We want to point out, that because of the applied rotation, also the order of rooms and connections is different within the samples generated from one floor plan.

4.3 Extension of Floor Plans

The two different approaches need different strategies to generate new floor plan aspects, as outlined below:

4.3.1 Block Generation Sequencers

In this approach, a new block is generated by feeding a concatenation of previous blocks with a sequence of blank vectors into the model and reading the predicted block from the model's output. The blank vector sequence length must be bigger that the expected length of the predicted block. This is can be done by determining the upper limit of predicted block length in the training database.

4.3.2 Vector Prediction Sequencers

Following a metaphor by Alex Graves, in which sequence predictors used for sequence generation are compared to a person dreaming (by iteratively treating their own output as if they are real (Graves, 2013)), this structures works like a dreaming person who gets inspiration from outside and who combines the information from outside with its flow of dreaming. Because of that, we refer to this technique as the *shallowDream* structure (see Figure 7).

Basically, this structure operates in two different phases. During the first phase, the existing floor plan (in this context also referred to as *concept*) is fed into the LSTM. During this phase of concept injection, all outputs of the LSTM are ignored. After the concept has been injected completely, the LSTM takes over both the generation of the structures output that also serves as its own input. This phase is also referred to as generation phase.

Using the shallowDream structure, we are able to implement multiple different functions by simply altering the concept type and the stop symbol. E.g. in order to predict room connections, the a concatenation of block 1 and the control vector of a block 2 tag (connection tag) is used as concept and the control vector of a room layout tag is used as stop symbol. The control vector at the end of a concept is used to instruct the LSTM to generate the favored tag type (and hence to start the new block).

Even after intensive training, the output produced by the LSTM only approximates the intended feature vectors. Consequently, these feature vectors have to be regenerated during the generation phase. For that purpose, three different strategies are proposed.

No Regeneration In this primitive approach, the current feature vector is reinserted without any modifications into the models input.

Vector-Based Regeneration This strategy solely utilizes knowledge about the feature vector's structure. Generally, all boolean components are recovered by mapping them to 1.0 or 0.0 based on which the component is closer to and the real-valued components remain unaltered.

Sequence-Based Regeneration In this approach, a state machine is keeping track of the current block and tag the sequence is in (thereby utilizing knowledge about the sequence structure). Based on that information a vector is regenerated by calculating the most likely, possible vector.

5 INTEGRATION OF THE PROPOSED MECHANISM INTO ARCHISTANT

In this work, we restrict ourselves on the two following functions:

- Room Connection Generation. Given a set of rooms (each room is described by a center position coordinate and room function) connections are generated between them, turning a set of rooms into a room graph.
- Room Layout Generation. Given a room Graph, layouts for each room a layout (i.e. a polygon describing its surrounding walls) is generated.

For the sake of simplicity, we added a single button to the WebUI only, which we labeled ”Creativity”. Based on the current state of the user's work, the different functions are selected automatically.

6 EXPERIMENTS

We trained LSTMs based on our two sequencing approaches. In all cases, we used a training database with 200 entries, a test database of 40 entries, a blowup factor of 30, 500 LSTM cells and a learning rate of 0.01.

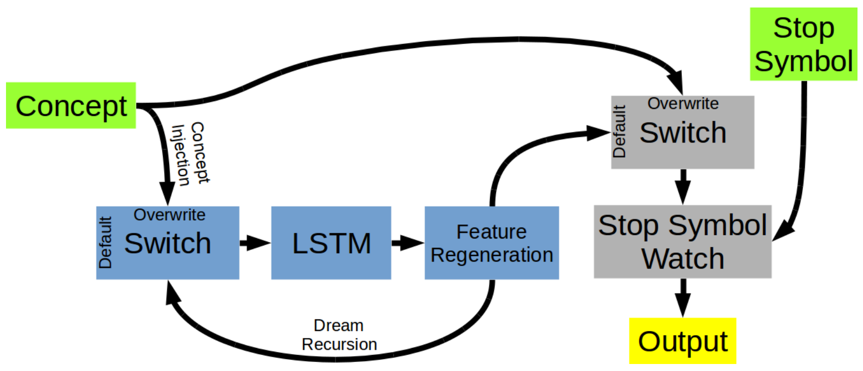

Figure 7: The ShallowDream structure. The inputs are marked green. Components of the LSTM recursion are marked blue.

6.1 Quantitative Analysis

In order to compare the performances of our different approaches, we calculated the room connection generation on the room definitions of the test set of floor plans and compared it with these floor plans actual connections. As a metric, we divided the amount of wrong connections by the amount of actual connections (we consider a connection to be completely wrong (error 1.0), if the predicted connection does not actually exist in the ground truth or partly wrong (error 0.5) if the connection type in ground truth is different). The final evaluation value is the arithmetic average over all floor plans in the test set. The results are shown in Table 2. We want to emphasize that floor plan generation is a creative task and that the is not necessarily one solution to a given problem, i.e. the used error calculation metrics do only hint the actual performance of the approaches (an error of 0% appears unrealistic to accomplish given that also the floor plans in the database are only one way to solve the problem).

Approach	**Error**
Block Generation Sequencer	65.78%
Vector Prediction Sequencer	66.08%

Table 2: Performance comparison of the individual approaches for the connection generation task on the test set.

6.2 Qualitative Analysis

The performance of the shallowDream structure is shown exemplary in two scenarios. For the room connection generation, four rooms are given (fig. 9) as a concept. As illustrated in Figure 8, the regeneration strategy influences the output. Figure 10 depicts a rendered version of the output produced by the sequence based regeneration.

In order to illustrate the performance of the room layout generation, a different starting situation is used (see fig. 11), the result is depicted in fig. 12.

7 FUTURE WORK

We have shown the general viability of our approach (the output of the trained models resembles the intended syntax in a quality sufficient for our inferencing algorithm to produce results). Nevertheless, a lot of floor plan aspects are not yet covered in the existing models. The actual position of doors and windows are needs to be included into the models as well as support for multi-storey buildings. Apart from that, the performance of the existing models is still limited. At the moment, there is only one phase of concept injection followed by a generation phase. By allowing for multiple alternating phases of generation and concept injection, even more functions could be realized. Apart from that, better metrics have to be found to assess a models performance.

In order to both allow for better comparison to similar approaches and to improve the performance of our system, the presented approach can be applied to a standard database (de las Heras et al., 2015) and existing algorithms (Delalandre et al., 2007) can be used to increase the sample size of the sample database.

Apart from that, our approach can be used as a general template for machine learning of user behavior, given that the data structure manipulated by the user can be described as a graph. Consequently, a more general implementation of a graph-based machine learning framework can be build from our approach.

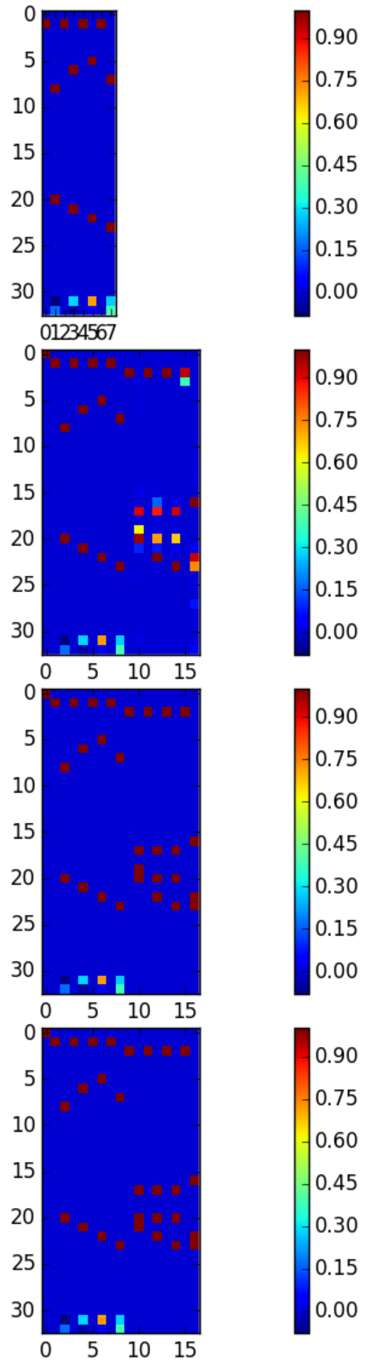
Figure 8: The regeneration technique in shallowDream influences the generated feature vector sequences. From Top to Bottom: 1. The concept. 2. No Regeneration. 3.Vector-Based Regeneration 4. Sequence-Based Regeneration

ACKNOWLEDGMENT

This work was partly funded by Deutsche Forschungs-Gemeinschaft.

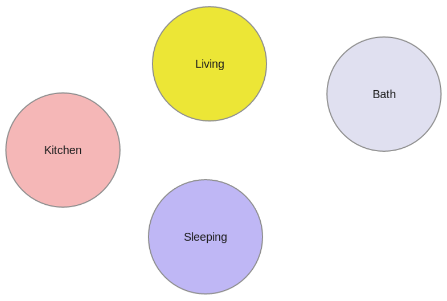

Figure 9: Input given to the shallowDream structure.

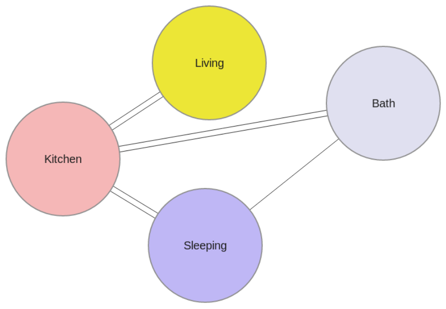

Figure 10: Output obtained from the shallowDream structure.

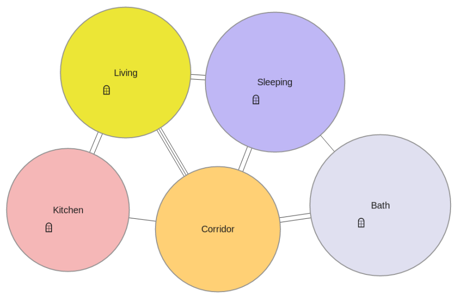

Figure 11: Input given to the shallowDream structure.

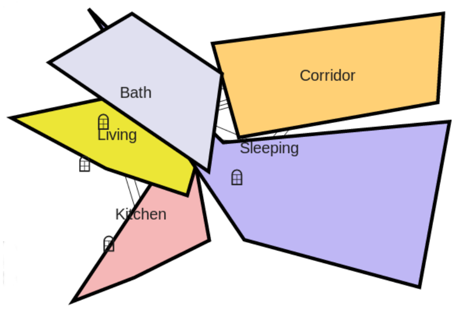

Figure 12: Output obtained from the shallowDream structure.

REFERENCES

Bayer, J., Bukhari, S. S., Langenhan, C., Liwicki, M., Althoff, K.-D., Petzold, F., and Dengel, A. (2015).

Migrating the classical pen-and-paper based conceptual sketching of architecture plans towards computer tools-prototype design and evaluation. In *International Workshop on Graphics Recognition*, pages 47–59. Springer.

Brandes, U., Eiglsperger, M., Lerner, J., and Pich, C. (2013). Graph markup language (graphml). *Handbook of graph drawing and visualization*, 20007:517–541.

Breuel, T. M. (2008). The ocropus open source ocr system. In *Electronic Imaging 2008*, pages 68150F–68150F. International Society for Optics and Photonics.

de las Heras, L.-P., Terrades, O. R., Robles, S., and Sánchez, G. (2015). Cvc-fp and sgt: a new database for structural floor plan analysis and its groundtruthing tool. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(1):15–30.

Delalandre, M., Pridmore, T., Valveny, E., Locteau, H., and Trupin, E. (2007). Building synthetic graphical documents for performance evaluation. In *International Workshop on Graphics Recognition*, pages 288–298. Springer.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Langenhan, C. (2017). *Datenmanagement in der Architektur*. Dissertation, Technische Universitt Mnchen, Mchen.

Sabri, Q. U., Bayer, J., Ayzenshtadt, V., Bukhari, S. S., Althoff, K.-D., and Dengel, A. (2017). Semantic pattern-based retrieval of architectural floor plans with case-based and graph-based searching techniques and their evaluation and visualization.