Data-based generation of residential floorplans using neural networks

Louise Deprez, Ruben Verstraeten and Pieter Pauwels
University of Ghent, Belgium
louise.deprez@ugent.be, ruben.verstraeten@ugent.be, pipauwel.pauwels@ugent.be

Most generative design applications used in architectural design are developed with rule-based approaches, based on rules collected from expert knowledge and experience. In other domains, machine learning and, more in particular, neural networks have proven their usefulness and added value in replacing these hard-coded rules or improving applications when combining these two strategies. Since the space allocation problem still remains an open research question and common generative design techniques showed their limitations trying to solve this problem, new techniques need to be explored. In this paper, the application of neural networks to solve the space allocation problem for residential floor plans is tested. This research aims to expose the advantages as well as the difficulties of using neural networks by reviewing existing neural network architectures from different domains and by applying and testing them in this new context using a dataset of residential floor plans.

Background

During the last decades, generative design (GD), a technique used for the automatic generation of design proposals, is more and more used when designing architectural projects. These types of design proposals can vary from for example the building envelope of the Beijing national stadium using genetic algorithms [9] to the volume generation of the prairie houses of Frank Lloyd Wright using shape grammars [26], both automatically generated by using a GD algorithm. More recently, with the improvement

Design Computing and Cognition DCC'22. J.S. Gero (ed), pp. 359-378. © Springer Nature 2022

of computing power and the emergence of more data, artificial intelligence (AI) is on the rise in many industries, even so in engineering and art. Architectural design has a long record in exploring the possibilities of AI within its domain. An old, but still open research question concerns the space allocation problem [28]. This research objective covers the generation of floor plans when an architectural program and a fixed building boundary, two requirements that are hard to unite, are given. An automation of this process will help the architect to quickly discover the potential of a parcel when used for a specific building program.

From around 1970 up until now, the automation of the space allocation problem started without the input constraint of a building boundary, to a rectangular constraint, later on going to orthogonal boundaries and finally also irregular boundaries. The possibility to use boundary constraints as an input for the generative program is important in for example renovation projects or when considering the context, such as neighboring buildings. The input constraints concerning the architectural program became also more complicated during this time span. First, almost no constraints were given, but later, room definition, room dimensions and adjacencies were also considered. However, when the complexity of the building boundary arose, room requirements were kept simple and vice-versa [28]. This resulted in tools dealing with complex interior layouts where the building boundary is the direct result of the spatial organization or tools where a fixed boundary is filled with rooms that could fit into this building, mostly leading to room adjacencies that do not correspond with the input requirements. A lot of research has been done on this design problem using common GD techniques and few good solutions were found merging the complexity of the building boundary and the room requirements [28]. Therefore, the aim of this research is to test the applicability of a popular AI technique, namely neural networks (NNs), for solving the space allocation problem in its full complexity.

First, the paper will document some disadvantages of common GD applications used for space allocation, so these can serve as points of attention when evaluating the different types of NNs. Secondly, related work will be discussed. Thereafter, pixel-based strategies are reviewed, after which the same is done for graph-based approaches. The most promising types of pixel- and graph-based NNs will be tested using a database of residential floor plans. Finally, the main advantages and difficulties of using NNs for solving the space allocation problem will be discussed.

Disadvantages of common GD applications

In most GD applications, when used for the automation of space allocation, input is processed directly into a final result. This full automation results in a first common problem, namely the lack of control of the architect over the design process. To resolve this, the design process can be broken down into several consecutive steps. This allows the architect to interfere between each step or to only automate certain steps in the process. The idea is based on a *grey boxing* method instead of *black boxing*, which allows the user to intervene along the way instead of inputting information upfront and getting a finished design at the end of the process [6]. Within this paper, the process will be broken down into two steps. First, a building boundary is generated on a blank page. This step becomes unnecessary when the architect wants to use the boundary conditions of a project as an input. Secondly, the interior spaces are filled into this boundary. Each step in this g*rey boxing* method will be supported by a separate NN to execute a specific task. It is possible that different types of NNs will be needed for these different steps.

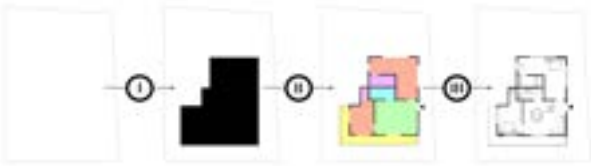


Fig. 1 Chaillou [6] used the *grey boxing* method to generate floor plans.

Most algorithms used in the field of architectural design are based on one or more of the following five GD techniques: shape grammars, L-systems, cellular automata, genetic algorithms and swarm intelligence [30]. All of them are rule-based techniques, which reveals a second common problem in the application of GD techniques concerning the time-consuming task of the development of new rules by experts, in this case architects. Because the space allocation problem situates itself in the initial stage of the design process, during the exploration phase when decision-making still needs to start, the development of new rules can be even harder and an automation of this process would help the architect in this phase. Furthermore, since each algorithm is developed to perform a specific task in a specific project, rules supporting the algorithm can seldom be used in another context. This means that for each project, new rules need to be defined and since the rules are to some extent hard-coded, a lot of reprogramming or even starting from scratch is needed. To avoid the need of developing new rules for each new GD algorithm, one could use generic grammars [4], potentially allowing greater flexibility than shape grammars, or one could use NNs to replace

rule-development with deep learning. In this research the latter is chosen. A NN is a data-trained method which can, as the word says, learn rules from real-world data and extract statistical information that can be used in the generation of new artificial data points, i.e. designs. The advantage of NNs compared to rule-based systems is that, besides the detection of intentional rules, NNs could also detect rules that are not consciously used by the architect [2]. Since data is used to learn a specific task, the execution of a task is dependent on the data. Other data may lead to the execution of another task. This means that one sample of code can theoretically train on different datasets, learning different tasks without the need of rewriting the whole algorithm [17], saving valuable time for the architect. For example, the code used in this research for generating residential floor plans could be used for the generation of another typology of floor plans without the need of hard-coding new rules. The model only has to be trained and tuned again within another dataset with the right typology of floor plans.

The dataset

When implementing a data-trained method like NNs a suitable dataset is needed, in this case a dataset of residential floor plans. The quality of the final application will also be influenced by the quality of the training data. Moreover, the format of the data is important to control the type of information the network will learn. As an example, just showing the model building footprints will yield a model able to create typical footprints. There do not exist many, easily accessible, large datasets, that are formatted into a machine readable way and are at the same time based on real-world architectural designs. One available dataset is the *RPlan* dataset. It is a dataset of more than 80,000 residential floorplans manually collected by Wu et al. [33] from real-world residential buildings in the real estate market in Asia. Each data point represents a 255x255 px image existing of four channels: the building boundary, the interior room types (e.g. living room, bedroom), their room identities (e.g. id 1 and id 2 to distinguish two rooms of type bedroom) and the interior-exterior mask (Fig. 2). The channels are compatible with the *grey boxing* method mentioned before. This dataset will be used in the experiments when testing different types of NNs, by which statistical rules to generate new designs can be extracted from the dataset.

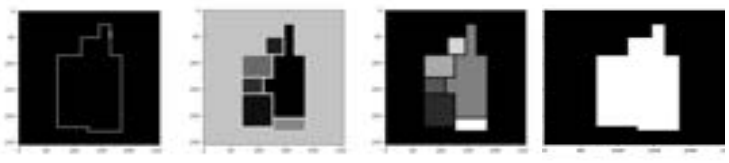


Fig 2. The four channels of one data point in the RPlan dataset [33].

Related work

Space allocation and building massing are old and popular research topics resolved many times, most of the time leading to an algorithm searching a restricted design space. In 2003, Hansmeyer [15] developed a flexible system that generates modular building forms adaptable to the environment using L-systems. In L-systems, design components are symbolized as strings and string rewriting mechanisms are applied according to a set of production rules, resulting in a string that can be translated back to a graphical representation. Kou et al. [19] used swarm intelligence in 2013 to find optimal evacuation routes for the Wuhan Sports Center. In 2016, Govaert [13] designed a building massing tool with hard-coded rules concerning sunlight, view and access based on cellular automata. Koning and Eizenberg designed a detailed shape grammar that makes it possible to generate all prairie houses designed by Frank Lloyd Wright and many more within his style [26]. These four examples show the possibilities of regular GD techniques, but despite the promising results, many rule-based systems still need to be hard-coded separately for each specific design problem.

Strobbe [31] showed the possibilities of machine learning (ML) by automatically classifying floor plans belonging to the style of the Malagueira houses designed by the architect Alvaro Siza Viera or to any other style by using a one-class support vector machine (one-class SVM) trained on adjacency graphs of floor plans. When using rule-based systems, these rules would need to be hard-coded for each style separately instead of being extracted automatically. Related to this, As, Pal and Basu [2] used GANs, a special type of NNs, to learn the main clusters of rooms in floor plans. Again, adjacency graphs are used to train the NN.

These deep learning algorithms need large datasets based on real-world data. Sharma et al. [27] did not only develop DANIEL, a deep learning application that captures semantic features of floorplans to use in floor plan retrieval, but they also created their own publicly available benchmark dataset, called ROBIN. Liu et al. [20] specifically recognized the need for converting rasterized floorplan images into vector-graphics representations to use in for example data analysis.

More recent research specifically tries to solve the space allocation problem. In 2020, Chaillou et al. [7] used Bayesian modeling, a statistical method, to generate adjacency graphs. However, they did not actually generate floor plans. In 2010, Merrell et al. [23] generated a floor plan given a list of rooms and their types, sizes and adjacencies using a Bayesian network. However, the external appearance of the building was a result of this layout. Nauata et al. [24] did the same thing only using a GAN, again

the building footprint was not used as an input parameter, but was a result of the internal layout. Chaillou [6] on the other hand, generated the interior layout when given a fixed building footprint using GANs. Liu et al. [21] generated the functional zoning and architectural layout of a campus when given a campus boundary and the surrounding roads, using *Pix2Pix*, a method based on GANs. Both algorithms of Chaillou and Liu cannot take a desired number of rooms nor their type and size as input restrictions. It seems that the restriction of a fixed building boundary cannot be united with the restriction of a specific number of rooms and their properties. It therefore stays an open research topic.

Methods

ML is a broad field of research within AI. ML algorithms build a model, trained on training data, in order to make predictions or decisions without being explicitly programmed to do so. NNs are the backbone of deep learning algorithms that are a subfield of ML. A NN is a layered system of neurons passing messages from one layer to the next and the "deep" in deep learning refers to the depth of layers in a NN. By learning the probability distribution of a set of training data, a data-trained generative model is able to generate new valid data points that fit the probability distribution of the model [22]. This generative power is exactly what is needed in the scope of this research. Because NNs are powerful in the classification of images or the generation of new images, pixel-based NNs are considered before looking into graph-based methods.

Pixel-based methods

Image classification using a neural network (NN)

NNs have become extremely powerful in analyzing and even generating pictures. The most basic NN is a classifier or a prediction network. To explain the working of a simple NN, one can think of a network that gets a picture of a digit as an input and gives the digit represented on the picture as an output. The network is able to classify the picture because it was trained on a training dataset where each data point contains a picture and the class/digit, i.e. ground truth, it belongs to. The dataset mentioned here, with the pictures of digits, is called the *MNIST* dataset and can be seen as the *"Hello world!"* of NNs. At first, the untrained network predicts a random class when given an input image, the output is evaluated on the basis of a

loss function which compares the predicted digit to the ground truth. This type of learning is called supervised learning.

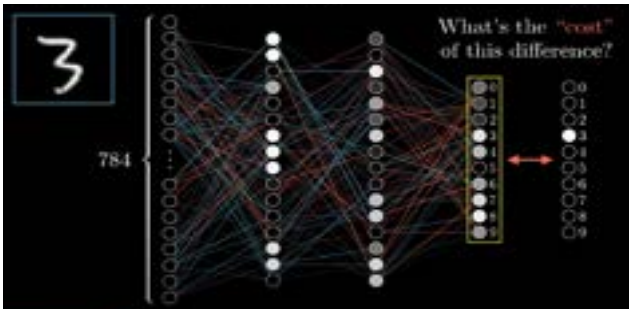


Fig 3. A NN trying to predict the correct class of the image at the beginning of training, thus resulting in a bad prediction [1].

To visualize this abstract concept of NNs, you could think of it as a set of layers, each with some neurons holding a value between zero (black) and one (white) (Fig. 3). The neurons of different layers are fully connected, i.e. each neuron of one layer is connected to each neuron of the next layer, and these connections each have a learnable weight or strength. The input layer receives the 28x28 px image of the digit, thus 784 pixels represented as neurons, communicates it to the first hidden layer and so on, until the last hidden layer communicates it to the output layer, holding ten neurons representing the ten digits, which tells us the prediction of the digit. The prediction can be recognized as the neuron/digit in the output layer with the highest value, i.e. the strongest signal.

The above NN could also be used to predict an architectural style when given an image of a floor plan, which was the research objective of Strobbe in 2016 [31]. Although this could be a possible solution, Strobbe used another ML technique, namely a one-class SVM. Since our interest is not simply the classification of images, but rather the generation of new images that would fit into the original dataset, a more complicated NN architecture is needed. Therefore, no experiments are done using this simple classification network.

Image generation using a generative adversarial neural network (GAN)

One very common network for generating pictures, that has furthermore proven its quality, is the GAN. Where a data point first had two pieces of information, namely the image and the ground truth, a data point now only has one piece of information, namely the image. To follow along, one could

think of a database full of images of building boundaries, such as the fourth channel of the *RPlan* dataset (Fig. 2), and our goal is to generate a new, synthetic image of a building boundary. The previous network learned to generate the ground truth given the input image by comparing them and calculating the cost. But now, when generating a synthetic image, there is no right or wrong, there is no clear output that can be compared to a true value to calculate the cost. And without a clear loss function to calculate the cost, the network cannot receive feedback to improve its performances.

So, another feedback system needs to make sure the learning process can happen. For this, two NNs are linked to each other: a generator and a discriminator. The generator learns to generate a synthetic image that fits the probability distribution of the original dataset, while the discriminator learns to separate the false, synthetic images from the true images present in the original dataset. The feedback from the discriminator helps the generator to generate better samples, while the discriminator gets more skilled in separating the real images from the synthetic samples. Increasing the error rate of the discriminator, i.e. fooling the discriminator, is the learning driver of the generator, comparable to what the loss function of a standard NN is. The generator and the discriminator are both separate NNs.

The discriminator is a convolutional neural network (CNN), a simple classifier with a ground truth, which gets a 28x28 px image as input and learns to classify the images into a "Real" or a "Fake" category. The generator is a deconvolutional neural network (DNN) which gets 784 input neurons filled with random noise and outputs a new synthetic image of a face as 784 output neurons, which correspond to a new 28x28 px image. The loss of the generator is solely the predicted label of the discriminator, i.e. the prediction of a "Real" image is telling the generator it is doing a good job in tricking the discriminator.

A CNN is not a fully connected network like a basic NN, but uses a tensor/filter to reduce the number of learnable parameters/weights. To clarify this, each connection between two neurons has a learnable strength. By dropping some of these (less significant) connections, the computational cost drops significantly, especially for images that tend to have a large number of input neurons. Interesting is that these filters can pick up on certain patterns, like horizontal lines or corners, that are in most cases very important to recognize what is shown on a picture. The generator is a DNN, which is similar to a CNN but runs in reverse.

Based on the promising results, this paper will test this method twice using the *RPlan* dataset [33], once to generate the building footprint, corresponding to the first step of our *grey boxing* method, and a second time to generate the interior layout. Only one channel of the data points is used

to keep the input layer a reasonable size and to support the *grey boxing* method.

The fourth channel (Fig. 2), representing the inside mask, will be used as training data for the GAN in the first experiment when generating the building footprint. This means that the input neurons only take one of two values: one (white) for interior and zero (black) for exterior. Important to notice is that this dataset is highly normalized, by which the input data is highly controlled, to learn one small task, namely the masking of the interior. The GAN takes random noise as an input, so requirements concerning area cannot be inputted upfront, but need to be checked afterwards. This does not cause any problem since generating multiple images is a matter of milliseconds and an area check can be quickly done to filter out irrelevant results.

For the second experiment, the second channel of the *RPlan* data points (Fig. 2) representing the interior layout is used. The learning process of the NN will become more complicated because the pixels can take more values than strictly one and zero, since each type of room or wall is represented by a different pixel value, e.g. a value of two for the kitchen and three for the bathroom. Again, the GAN takes random noise as an input, so no building boundary will be inputted upfront. The GAN is expected to generate the building boundary along with the interior layout.

Interior layout generation using Pix2Pix

One major problem with the setup of the second experiment is the absence of a fixed building boundary as input for the network, a random interior layout was generated in a random building boundary. In some projects, the building boundary is a fixed requirement that cannot be changed to satisfy the interior layout. To constrain the GAN to a fixed building boundary, an image of it should serve as input. However, the architecture of a GAN should change since its input can only be random noise. Fortunately, Isola et al. [17] developed *Pix2Pix* which can be described as an image-to-image translation with a conditional GAN (cGAN).

Instead of using random noise as an input, *Pix2Pix* uses an input image and learns to map an output image to it. This means that this time, both the second and the fourth channel (Fig. 2) are used as respectively the ground truth and the input. One could think that solely using a basic NN could work since there is a ground truth, however, it would only learn deterministic outputs, meaning each building boundary only has one correct solution. In practice, one building boundary should result in different possible interior layouts. Because of this, *Pix2Pix* uses a GAN which makes it possible to implement noise and a less deterministic loss function and, by this, create

variation. The noise is not added as an additional input, like in a basic GAN, but will be added as dropout, i.e. by randomly dropping some of the neurons in the network [17].

Pix2Pix is previously used in several contexts for the mapping of labeled pictures or edges to images (Fig. 4). Within the third experiment, this paper will generate an interior layout starting from a building boundary, which corresponds to the second step in our *grey boxing* method.

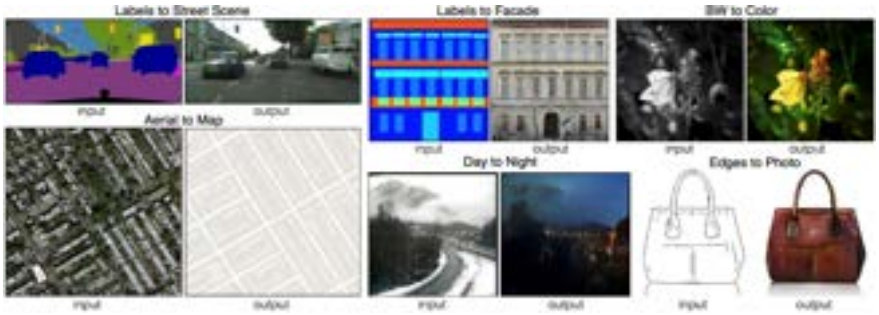


Fig 4. Several applications of Pix2Pix [17].

Graph-based methods

When approaching the research question with a pixel-based method, the input requirements are too limited. Inputting additional requirements such as room type, room number, areas and orientation are not possible when using one of the above NNs. Graphs can contain these requirements as node, edge and graph attributes, by which it is possible to add semantics to the graph, and they can be used as an input/output of other types of NNs [2]. This is why graph-based methods are considered.

An architectural plan can be represented by two different types of graphs or variations to them. In the first one, called semantic building footprint graphs (SBF) [12], rooms are represented by nodes and their adjacencies by edges. The second type is the one where the corners of the rooms are represented by the nodes and the lines between them by edges. In the context of this paper, the first type is used, because it is important for the NN to learn the adjacencies rather than the geometry of the rooms. As said earlier, what the NN is able to learn and generate is dependent on the data shown to it. Besides adding semantics to the data, using graphs has another advantage. Since each room is represented by one single node, the number of variables is strongly reduced. In the case of a NN, this means a smaller network with less neurons and consequently less weights and biases to learn, resulting in less computation time.

GANs are very powerful in the generation of real valued data, such as images, but they fail in generalizing to discrete objects, like graphs. The adaptation of GANs to support this new input format remained an open research question until 2018 because large repositories of graphs coming from the same distribution are not easily available. Since 2018, a few new approaches arose [5].

When representing an image as a graph, its nodes/pixels are fixed in space and they are always connected with their closest neighbors (Fig. 5) by which a filter can operate on it to find patterns in the picture. Because nodes in a graph are not necessarily ordered, do not have a fixed number of adjacent nodes and the graph does not have a fixed size, a filter cannot operate on this structure, therefore other methods for grasping the structure of the graph are needed. There are two possible ways to handle the variable size and structure of graphs. Common NNs, GANs included, need the input data to be fixed in size and structure, e.g. a fixed *n*m* px picture, by which the first approach is to represent the graph as a fixed matrix, e.g. adjacency matrices or random walks. A second approach is to use graph neural networks (GNN), which can handle data with a varying number of nodes and edges. Depending on the type of GNN, predictions on graphs can be made (like node attribute prediction, classification of graphs or link prediction) or even new graphs can be generated.

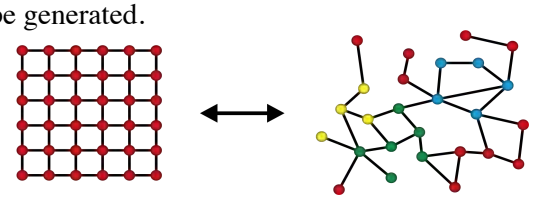


Fig 5. A graph is a data type without a fixed size or structure. An image can be represented by a fixed size matrix.

Adjacency matrix generation using a GAN

The following method is based on molecule generation with GANs, where atoms are represented by nodes and their bonds by edges [10]. This method shows the difficulties concerning the representation of a graph as an adjacency matrix, however, when we choose to continue using GANs, such a representation is needed. Most researches focus on molecular graphs, since these datasets are easily accessible, something which is missing in the domain of architecture. A graph, whether it is representing a molecule or a floor plan, is defined by its nodes and edges and these two can in turn have several properties. A fixed size adjacency matrix ($A \in \{0,1\}^{n\times n\times(b+1)}$) and

a node feature matrix ($X \in \{0,1\}^{n \times d}$) are able to hold this information [22] where the graph has a fixed number of nodes n, edge types b ($b+1$ to include non-edges) and node types d. When a node or edge has a certain type, the value 1 is inserted, otherwise 0. This is called one-hot encoding, which has proven to perform well as training data representation in prediction networks [12]. The GAN can, after learning, generate these type of matrices representing new floor plans.

When using adjacency matrices, some challenges occur. First, a graph with n nodes has to output at least n^2 values, mostly zeros. Secondly, this same graph can be represented by $n!$ different matrices depending on the node ordering [34]. Tavakoli et al. [32] learned the topology of social graphs by making 10,000 permutations of node orderings over only 4 social graphs. Imagine the computational power needed to represent a few more graphs. On top of that, the adjacency matrix is still a very restricted method where every graph has a fixed number of nodes. When more rooms are present in a new reference project, the whole model needs retraining on a new dataset with architectural plans containing an equal number of rooms. This means that the design space is strongly restricted beforehand. Because of its limitations, no further experiments are done using GANs to generate adjacency matrices.

Random walk generation using a GAN

Another possible solution is the transformation of the problem of generating graphs, to the generation of fixed-length random walks. Two major advantages are the possibility of inputting graphs with varying dimensions and avoiding the problems concerning node ordering. This method, called *NetGAN* [5], is developed for the generation of large graphs like social networks, where one graph, represented by a lot of random walks, serves as input data. The generator learns to generate walks, as a sequence of nodes, that are plausible in the real graph. After generating a set of walks, they are assembled in a count matrix and used to produce the adjacency matrix of the new graph. The resulting graph has a comparable size and connectivity as the input graph [5].

Since *NetGAN* is developed for large graphs and these are not the subject of this research, one can think of an extension of this algorithm to produce small graphs based on a set of random walks over a set of small input graphs. However, one major disadvantage remains, this model is developed for homogeneous graphs, i.e. networks without any edge and node types, so no further experiments are done using this exact method.

Variational auto-encoders (VAE)

All above mentioned methods are based on GANs, but in reality other generative structures exist. For the generation of graphs, three structures can be distinguished: GANs, VAEs and autoregressive models.

VAEs are comparable to GANs, because their key building blocks are the same: a DNN/decoder and a CNN/encoder. First, the encoder compresses a given input data point to a lower dimensional space after which the decoder reconstructs the low D representation back to a high D representation. Finally, each reconstruction can be compared to its original input data point and the loss is calculated. This means that a new artificial sample can be generated when decoding a random point in this low D space.

The graphs are in most cases still represented as adjacency matrices [14, 22], thus a lot of challenges occurring with GANs still remain in VAEs. Since VAEs still show too much limitations, again, no further experiments are done using this method.

Autoregressive models

GANs and VAEs are types of NNs that were originally developed for pixel-based approaches. Autoregressive models, on the other hand, are developed for more complicated data structures, such as graphs. *GraphAF* [29], for example, is especially developed to generate molecules represented by a graph structure.

In autoregressive models the adjacency matrix is generated by sequentially generating the adjacency vector of each node considering the previous state of the graph. Starting from an empty graph, each step a new node is added after which its edges are computed (Fig. 6). Additional architectural rules can be added, which can be used to check whether the addition of a certain node or edge is regulated. These rules could for example include the need of certain rooms or the avoidance of less desired adjacencies.

It is the only method that can generate graphs of varying size, with node and edge types (wall, door) and attributes (area, center coordinates) present. *GraphAF* is originally developed to generate molecules, but can be adjusted to generate interior layout graphs. Thus, in the fourth experiment an autoregressive model will be trained to generate adjacency matrices of interior plan layouts. In this paper, the *RPlan* dataset is automatically converted from an image dataset to a graph dataset, by first vectorizing the images and then creating their matrices.

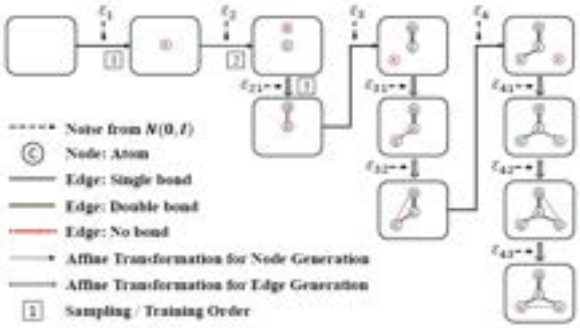


Fig 6. Autoregressive model for molecule generation [29].

From graph to plan

When the layout graph is generated together with its desired attributes, like room area or adjacency type, the graph needs to be mapped into a fixed building boundary. By breaking down this process into two consecutive steps, the architect has a lot more flexibility in deciding at which point to step into the process. The whole graph can for example be generated by the autoregressive model or the architect could create the graph all by himself/herself. To map the graph inside a building boundary, two researches pop out, giving an initial grasp of what a NN performing this task might look like.

The first research creates an interior layout in a fixed building boundary, however without using an input graph, but still using a pixel-based approach. This still remains relevant since a sequential process comparable to the autoregressive model is used. Wu et al. [33], who put together the *RPlan* dataset, trained a NN to first locate the center of the living room inside a given building boundary, then sequentially determine the next room type and location based on the current state of the plan and finally generate the interior walls. They trained an encoder-decoder network by randomly removing rooms from the dataset and trying to predict the center of these missing rooms [33] which could also be done with the autoregressive model by adding the center as an attribute to the graph.

Fig 7. Model architecture used by Wu et al. [33].

A second research, called *Graph2Plan*, tries to give the designer the option to add a graph additional to the boundary restriction. The NN serves as a search algorithm and searches for similar building boundaries and graphs in the dataset. Then, the layout graph from the similar boundary gets copied into the input boundary and finally the interior rooms are plotted. However, except from a well-trained search algorithm, the NN is not used to its full potential [16].

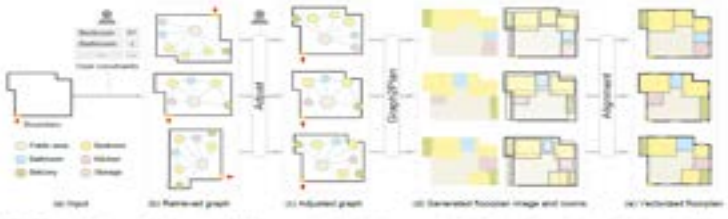


Fig 8. Model architecture used by Hu et al. [16].

Results

In these experiments, the learning capacity of a basic GAN is evaluated based on its number of data points, the sizes of the pictures and the number of epochs while learning. The results will show the influence of these three parameters.

Within the first experiment a GAN was trained on the fourth channel of the *RPlan* dataset. The generator uses convolutional layers to upsample the input and turn random noise into an image. *LeakyReLU* is used for the activation of each layer, except for the output layer that uses *tanh* as activation function. The discriminator uses *LeakyReLU* as activation for each layer and dropout to lower the computational cost. When using 1,000 data points for 100 epochs, the learning takes a long time and results in blurry pictures (Fig. 9a). When compressing each picture to a 56x56 px

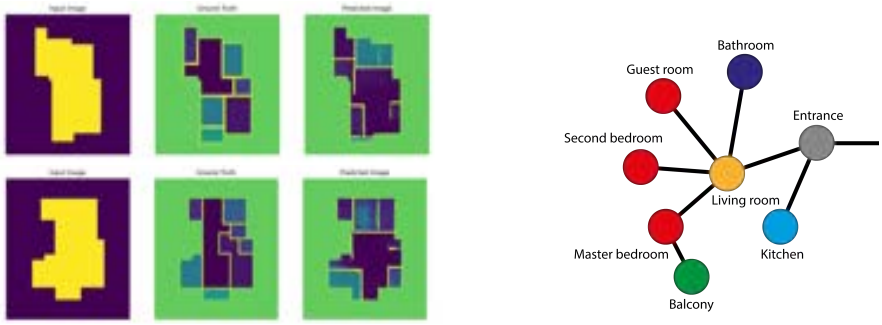
image, by which the number of trainable parameters is reduced significantly, the results become much better. The generated pictures are still blurry, but shapes are recognizable (Fig. 9b). When learning for a longer time, the results do not improve significantly, but when using the whole 80,000+ dataset, clear shapes can be recognized (Fig. 9c). However, note that a 80,000+ dataset is not common and may not be available in a lot of projects.

(a) (b) (c)

Fig 9. (a) Training 1000 data points for 100 epochs. (b) Training 1000 compressed data points for 250 epochs. (c) Training 80k+ compressed data points for 100 epochs.

Within the second experiment, the tests done in the first experiment are repeated, but instead of the fourth channel, the second channel of the *RPlan* dataset is used, representing the interior layout. Even when using the best model of the previous experiment, i.e. when using more than 80,000 data points, compressing the images and letting the network learn for a long time, the generated images are still too blurry to identify specific shapes, similar to the situation in Fig. 9a. Even when learning for a longer time or changing the architecture of the network, the results are not expected to be excellent within a reasonable computation time using a reasonable amount of data.

Within the third experiment, the interior layout is generated using *Pix2Pix*. Here, the fourth channel of the *RPlan* dataset is used as an input and the second channel as the ground truth. The encoder consists of convolutional layers activated with *LeakyReLU* and the decoder consists of transposed convolutional layers activated with *ReLU* and with dropout applied to them. Again, several problems occur when the algorithm is tested. While the model, trained with *Pix2Pix*, seems to result in acceptable results, one can see that the interior layout of the two data points as shown in Fig. 10a are almost identical. The model always places the interior walls at the same location regardless of the building boundary. Moreover, only minor stochasticity is observed in the output of the network when given the same input boundary, despite the dropout. This disadvantage was already observed by Isola et al. [17]. Still, if the network would give the desired results, only the input boundary is given as a demand, the desired interior rooms are not considered as an input.



(a) (b)

Fig 10. (a) In the third experiment Pix2Pix learned to map the interior layout of a residential unit to a building boundary. From left to right: the input image, the ground truth and the predicted image. (b) In the fourth experiment the autoregressive model learned to generate graphs.

When approaching the research question with a pixel-based method, an enormous amount of data and computation time is needed to avoid blurry pictures and even so, the input requirements are too limited.

Within the fourth and last experiment a graph-based method is used, namely an autoregressive model is trained using the adjacency matrices extracted from the *RPlan* dataset. Because of how the model is build, it is only trained during 3 epochs with a batch size of 15. The generated adjacency matrices have a novelty value of 1, which means no copies are made from the original dataset, and have a unique rate of 1, which means no identical data points exist in the set of newly generated graphs.

Conclusion

In this paper, the application of NNs to resolve the space allocation problem for residential floor plans was tested. This research aims to expose the advantages as well as the difficulties of using NNs by reviewing existing NN architectures from other domains and by applying and testing them in this new context using *RPlan*, a dataset of real-world residential floor plans.

First, pixel-based approaches were explored and tested. The experiments demonstrated that (c)GANs can generate acceptable pictures of floor plans when the model is trained for a rather long time on a large dataset. However, large, publicly available datasets of real-world floor plans are still rare. At the beginning of the paper, it was mentioned that the constraint of an irregular building boundary and the room constraints are hard to unite. *Pix2Pix* can handle highly irregular building boundaries, as long as the

training data also contains irregular building boundaries, but using room restrictions as an input to the model is not possible. Since images do not contain semantic information, pixel-based approaches are not promising considering room restrictions. Graphs, on the other hand, can hold this information and therefore graph-based approaches were explored as a following step.

Many graph-based approaches, like GANs and VAEs, are based on pixel-based approaches, resulting in new disadvantages concerning the representation of the graph. Autoregressive models are developed to handle complex data types, like graphs, and are able to generate new graphs of varying size with node and edge attributes. Since geometry is ignored in these graphs, except for maybe an area attribute or a center attribute, graph-to-plan methods need to be developed. Two researches give a grasp of what this might look like, but more research still has to be done on this topic.

To summarize, NNs can only perform well when trained on large, suitable datasets, which are rarely available. On top of this, the NNs tested within this research use predefined functions and are built in a way that needs highly structured data as an input. The experiments show that the tested NNs can be used to perform small tasks, but a larger program architecture would be needed to sequentially use networks to perform these small tasks. For now, the space allocation problem is still not completely solved, when taking into account both a fixed building boundary and room allocations, but this research shows the potential of using NNs for solving this problem. Also, this research, together with previous research, shows that, for now, NNs are more limited than rule-based methods. In other domains, the combination of NNs and rule-based systems have led to strong program architectures. Recently, together with the development of new graph databases, new researches concerning graph NNs keep popping up in other domains. Thus, the full potential of NNs are still to be discovered.

References

1. 3Blue1Brown. (2017, November 3). What is backpropagation really doing? | Chapter 3, Deep learning. Retrieved from YouTube: https://www.youtube.com/watch?v=Ilg3gGewQ5U
2. As, I., Pal, S., & Basu, P. (2018). Artificial intelligence in architecture: Generating cenceptual design via deep learning. International Journal of Architectural Computing: Vol. 16, pp. 306 - 327.
3. Arvin, S. A., & House, D. H. (2002). Modeling architectural design objectives in physically based space planning. Automation in Construction 11, 213 - 225.

4. Beirão, J.N., Duarte, J.P. & Stouffs, R. Creating Specific Grammars with Generic Grammars: Towards Flexible Urban Design. Nexus Netw J 13, 73–111 (2011). https://doi.org/10.1007/s00004-011-0059-3
5. Bojchevski, A., Shchur, O., Zügner, D., & Günnemann, S. (2018). NetGAN: Generating Graphs via RandomWalks. Proceedings of the 35th International Conference on Machine Learning, (pp. 609 - 618). Stockholm.
6. Chaillou, S. (2019, Februari 24). AI & Architecture: An Experimental Perspective. Retrieved June 8, 2021, from towards data science: https://towardsdatascience.com/ai-architecture-f9d78c6958e0
7. Chaillou, S., Landes, J., Fure, H., & Dissen, H. (2020). Architecture as a Graph | A Computational Approach.
8. Chakure, A. (2020, January 10). Convolutional Neural Networks (CNN) in a Brief. Retrieved from Dev: https://dev.to/afrozchakure/cnn-in-a-brief-27gg
9. De Azambuja Varela, P. (2013, April). Genetic algorithms in architecture: history and relevance. 1ST eCAADe Regional International Workshop, pp. 133 - 142.
10. De Cao, N., & Kipf, T. (2018). MolGAN: An implicit generative model for small molecular graphs. ArXiv.
11. Deep Convolutional Generative Adversarial Network. (2021, June 17). Retrieved August 2021, from Tensorflow: https://www.tensorflow.org/tutorials/generative/dcgan
12. Eisenstadt, V., Arora, H., Ziegler, C., Bielski, J., Langenhan, C., Althoff, K., & Dengel, A. (2021). Comparative Evaluation of Tensor-based Data Representations for Deep Learning Methods in Architecture, Proceedings of the 39th eCAADe Conference – Vol. 1, University of Novi Sad, Novi Sad, Serbia, pp. 45-54.
13. Govaert, E., Verstraeten, R., Wyffels, F., Leenknegt, S., & Strobbe, T. (2016). Inzetbaarheid van cellulaire automaten in het architecturaal ontwerpproces. Ghent: Universiteit Gent.
14. Grover, A., Zweig, A., & Ermon, S. (2018). Graphite: Iterative Generative Modeling of Graphs. arXiv.
15. Hansmeyer, M. (2003). L-Systems in Architecture. Retrieved June 8, 2021, from michael-hansmeyer: http://www.michael-hansmeyer.com/l-systems
16. Hu, R., Huang, Z., Tang, Y., Van Kaick, O., Ahang, H., & Huang, H. (2020, July). Graph2Plan: learning floorplan generation from layout graphs. ACM Transactions on Graphics, 39(4), pp. 118:1 - 118:14.
17. Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-To-Image Translation With Conditional Adversarial Networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (pp. 1125 - 1134).
18. Koning H. and Eizenberg J. (1981). The language of the prairie: Frank Lloyd Wright's prairie houses. Environment and Planning B Planning and Design 8: 295–323.
19. Kou, J., Xiong, C., Fang, Z., Zong, X., & Chen, Z. (2013). Multiobjective Optimization of Evacuation Routes in Stadium Using Superposed Potential Field Network Based ACO. Computational Intelligence and Neuroscience.

20. Liu, C., Wu, J., Kohli, P., & Furukawa, Y. (2017). Raster-to-Vector: Revisiting Floorplan Transformation, 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2214-2222.
21. Liu, Y., Luo, Y., deng, Q., & Zhou, X. (2021). Exploration of Campus Layout Based on Generative Adversarial Network. Proceedings of the 2020 DigitalFUTURES, The 2nd International Conference on Computational Design and Robotic Fabrication (CDRF 2020), (pp. 169 - 178).
22. Ma, T., Chen, J., & Xiao, C. (2018). Constrained Generation of Semantically Valid Graphs via Regularizing Variational Autoencoders. 32nd Conference on Neural Information Processing Systems (NeurIPS 2018). Montréal, Canada.
23. Merrell, P., Schkufza, E., & Koltun, V. (2010, December). Computer-Generated Residential Building Layouts. ACM Transactions on Graphics, 29(6), pp. 1 - 12.
24. Nauata, N., Chang, K.-H., Cheng, C.-Y., Mori, G., & Furukawa, Y. (2020, March). House-GAN: Relational Generative Adversarial Networks for Graph-constrained House Layout Generation.
25. Ozdemir, S., & Ozdemir, Y. (2017). Prioritizing store plan alternatives produced with shape grammar using multi-criteria decision-making techniques. Environment and Planning B Urban Analytics and City Science, pp. 1 - 21.
26. Pupo, R., Pinheiro, E., Mendes, G., Kowaltowski, D., & Celani, G. (2007, November). A design teaching method using shape grammars. Graphica 2007.
27. Sharma, D., Gupta, N., Chattopadhyay, C., & Mehta,S. (2017), DANIEL: A Deep Architecture for Automatic Analysis and Retrieval of Building Floor Plans, 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), pp. 420-425.
28. Shekhawat, K., Upasani, N., Bisht, S., & Jain, R. N. (2021, April 12). A tool for computer-generated dimensioned floorplans based on given adjacencies. *Automation in construction*.
29. Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., & Tang, J. (2020). GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation. ArXiv.
30. Singh, V., & Gu, N. (2012, March). Towards an integrated generative design framework. Design Studies Vol 33, pp. 185 - 207.
31. Strobbe, T., wyffels, F., Verstraeten, R., De Meyer, R., & Van Campenhout, J. (2016). Automatic architectural style detection using one-class support vector machines and graph kernels. Automation in Construction, 69, 1 - 10.
32. Tavakoli, S., Hajibagheri, A., & Sukthankar, G. (2017). Learning Social Graph Topologies using Generative Adversarial Neural Networks. Conference: International Conference on Social Computing, Behavioral-Cultural Modeling & Prediction (Late Breaking).
33. Wu, W., Fu, X.-M., Tang, R., Wang, Y., Qi, Y.-H., & Liu, L. (2019, November). Data-driven Interior Plan Generation for Residential Buildings. ACM Transactions on Graphics (SIGGRAPH Asia), 38(6), Article 234.
34. You, J., Ying, R., Ren, X., Hamilton, W. L., & Jure, L. (2018). GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. International Conference on Machine Learning (ICML).