UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA CIVIL

CONCEPTUAL STRUCTURAL DESIGN OF SHEAR WALL BUILDINGS LAYOUT BASED ON ARTIFICIAL NEURAL NETWORKS

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN INGENIERÍA ESTRUCTURAL, SÍSMICA Y GEOTÉCNICA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL

PABLO NICOLÁS PIZARRO RIFFO

PROFESOR GUÍA:
LEONARDO MASSONE SÁNCHEZ

MIEMBROS DE LA COMISIÓN:
FABIÁN ROJAS BARRALES
RAFAEL OMAR RUIZ
NANCY HITSCHFELD KAHLER

Este trabajo ha sido parcialmente financiado por:
Agencia Nacional de Investigación y Desarrollo (ANID)
Beca para estudios de magíster / MAGÍSTER BECAS CHILE/2020 - 22200500

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL GRADO DE MAGISTER EN INGENIERÍA
ESTRUCTURAL, SÍSMICA Y GEOTÉCNICA
Y AL TÍTULO DE INGENIERO CIVIL
POR: PABLO NICOLÁS PIZARRO RIFFO
FECHA: JULIO DE 2021
PROF. GUÍA: LEONARDO MASSONE SÁNCHEZ

CONCEPTUAL STRUCTURAL DESIGN OF SHEAR WALL BUILDINGS LAYOUT BASED ON ARTIFICIAL NEURAL NETWORKS

In the structural design of shear wall buildings, the initial process requires the interaction between the architecture and engineering teams to define the appropriate distribution of walls, a stage typically carried out through a trial-and-error procedure, without any consideration of previous similar projects. For the engineering analysis, first, the wall thickness and length definition, their location, and in some cases, the presence of new wall sections, are required to fulfill not only architectural requirements but also engineering needs such as building deformation limits, base shear, among others. For this reason, the present work develops a structural design platform based on artificial neural networks (ANN) to design the shear wall layout for reinforced concrete wall buildings.

In the first place, the study includes surveying the architectural and engineering plans for a total of 165 buildings constructed in Chile; the generated database has the geometric and topological definition of the walls and slabs. As a second stage, an ANN model was trained for the regression of the wall segments' thickness and length, making use of a feature vector that models the variation between the architectural and engineering plans for a set of conditions such as the thickness, connectivity (vertical and horizontal), area, wall density, the distance between elements, wall angles, foundation soil type, among other engineering parameters, archiving remarkable results regarding the coefficient of determination (R^2) of 0.995 and 0.994 for the predicted wall thickness and length, respectively. However, a regressive model of this nature does not incorporate spatial detail or contextual information of each wall's perimeter; also, the prediction of other parameters such as the wall translation has a poor performance. For this reason, as a third stage, a framework based on convolutional neural network (CNN) models was proposed to generate the final engineering floor plan by combining two independent floor plan predictions, considering the architectural data as input. The first plan prediction is assembled using two regressive models that predict the wall engineering values of the thickness, length, wall translation on both axes from the architectural plan, and the floor bounding box width and aspect ratio. The second is assembled using a model that generates a likely image for each wall's engineering floor plan. Finally, both are combined to lead the final engineering floor plan, which allows predicting the wall segments design parameters and proposes new structural elements not present in architecture, making the methodology an excellent candidate to accelerate the building layout's early conceptual design.

RESUMEN DE LA MEMORIA PARA OPTAR
AL GRADO DE MAGISTER EN INGENIERÍA
ESTRUCTURAL, SÍSMICA Y GEOTÉCNICA
Y AL TÍTULO DE INGENIERO CIVIL
POR: PABLO NICOLÁS PIZARRO RIFFO
FECHA: JULIO DE 2021
PROF. GUÍA: LEONARDO MASSONE SÁNCHEZ

DISEÑO ESTRUCTURAL CONCEPTUAL DE LA CONFIGURACIÓN DE MUROS DE CORTE EN EDIFICIOS BASADO EN REDES NEURONALES ARTIFICIALES

En el diseño estructural de edificios de muros de corte, el proceso inicial requiere la interacción entre los equipos de arquitectura e ingeniería para definir la configuración adecuada de los muros, una etapa que suele llevarse a cabo mediante un procedimiento de prueba y error, sin tener en cuenta proyectos similares anteriores. Para el análisis de ingeniería, en primer lugar, se requiere la definición del espesor y la longitud de los muros, su ubicación, y en algunos casos la presencia de nuevas secciones de muros para cumplir no sólo con los requisitos arquitectónicos, sino también con las necesidades de ingeniería, como los límites de deformación, el corte basal, entre otros. Por esta razón, el presente trabajo desarrolla un framework basado en redes neuronales artificiales (ANN) para diseñar la configuración estructural en planta de muros en edificios de hormigón armado.

El estudio incluye, como primera etapa, el levantamiento de los planos de arquitectura e ingeniería de un total de 165 edificios construidos en Chile; la base de datos generada cuenta con la definición geométrica y topológica de los muros y losas. Como segunda etapa, se entrenó un modelo ANN para la regresión de los espesores y longitudes de los segmentos de muros, haciendo uso de un vector de características que modela la variación entre los planos de arquitectura e ingeniería para un conjunto de condiciones como el espesor, la conectividad (vertical y horizontal), el área, la densidad de muros, la distancia entre elementos, los ángulos, el tipo de suelo de fundación, entre otros parámetros de ingeniería, logrando notables resultados en cuanto al coeficiente de determinación (R^2) de 0.995 y 0.994 para la predicción del espesor y largo. Sin embargo, un modelo regresivo de esta naturaleza no incorpora el detalle espacial ni la información contextual del perímetro de cada muro; además, la predicción de otros parámetros como la traslación del muro no presenta un adecuado desempeño. Por ello, como tercera etapa, se propuso un framework basado en modelos de redes neuronales convolucionales (CNN) para generar el plano final de ingeniería combinando dos predicciones de plano independientes, considerando los datos arquitectónicos como input. El primer plano de planta se realiza mediante dos modelos regresivos que predicen los valores de ingeniería del grosor, la longitud, la traslación de los muros en ambos ejes a partir de la planta arquitectónica, y la anchura y relación de aspecto del cuadro delimitador del piso. La segunda predicción se realiza mediante un modelo que genera una imagen probable del plano de ingeniería para cada muro. Finalmente, ambas se combinan para obtener el plano final de la planta de ingeniería, lo que permite predecir los parámetros de diseño de los segmentos de muro y proponer nuevos elementos estructurales no presentes en arquitectura, lo que convierte a la metodología en un excelente candidato para acelerar el diseño conceptual de la configuración inicial de los muros de un edificio.

To my parents,
who always supported me and said
that nothing was impossible.

Acknowledgments

This work would not have been possible without the support of my family, my parents Rossana and Patricio, my sister Francisca, and Milú, who accompanied me during the good and bad times. Thank you for your trust, your teachings, patience, and commitment; with you, my time at the university, my master's degree, and my following challenges are and will be much more bearable.

Thank Bárbara, who accompanied me throughout this process, listened to my doubts, fears, complaints, and always knew what to tell me at the right time; your affection is evident in this work. I cannot ignore my childhood friends Felipe, Exequiel, Constanza, Catalina, and those I met at the University of Chile, particularly Cony, Nathalie, Ignacio, Mauricio, Gabriel, José, Sebastián, JR, CL, and many others. The beers, pizzas, trips, parties, laughs, and advice we have shared have meant a lot to me. I also thank from the bottom of my heart all my teachers from my childhood to university, whose hard work has brought me to where I am now. I would also like to express my sincere appreciation and thanks to my research advisor Professor Leonardo Massone for his guidance, time, and support throughout the process, and to Professor Nancy Hitschfeld for her support during so many years, who even encouraged me to pursue a master's degree in computer science. Finally, I would like to send a warm greeting to my grandfather Camilo, whose departure has meant a significant change in me, without you being able to see it.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1. Motivation

The wall layout selection in reinforced concrete shear wall systems, commonly used to sustain earthquake actions given their significant lateral stiffness and strength, is an extensive interaction process between the architectural and structural engineering groups. In some countries (e.g., United States), the structural configuration usually consists of a few rectangular walls [1], while in places such as Chile, Japan, or Europe, it is common to find wall systems with a non-uniform distribution composed of several smaller cross-section elements. In Chile, the residential buildings' configuration is characterized by many walls with complex cross-sections [2–4] (Figure 1.1), usually established due to architectural reasons. Thus, the interaction between architecture and structural engineering teams is first focused on wall thickness and length definition, their location, and in some cases, the presence of new walls to fulfill not only architectural requirements but also engineering needs such as building deformation limits, base shear, among others.

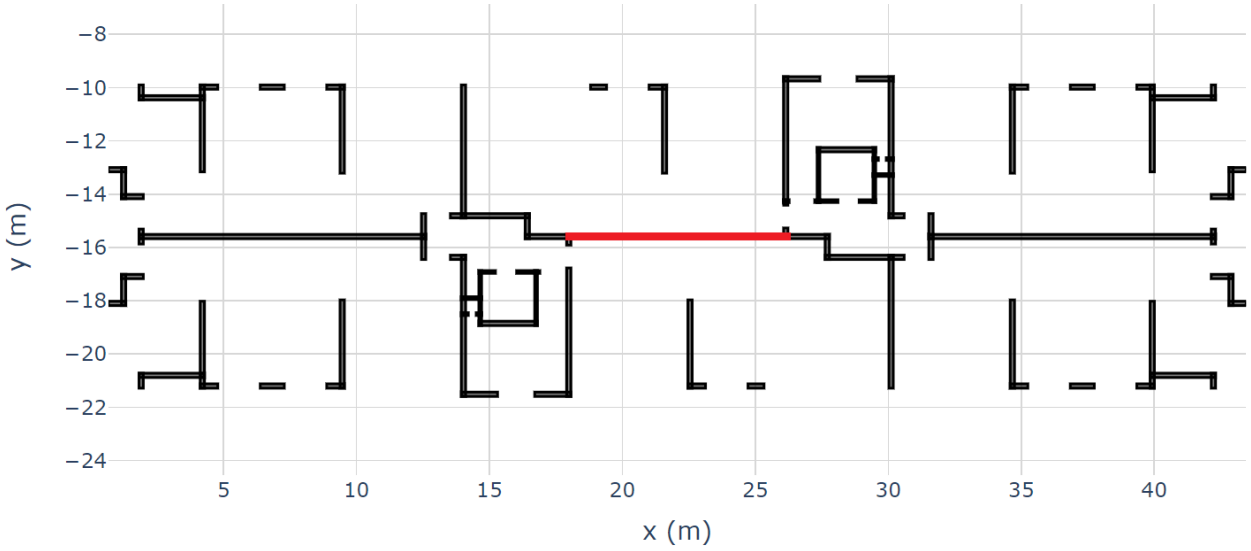


Figure 1.1: Example of a typical Chilean building wall layout.

A vast amount of data is generated from this traditional design process, usually, as architectural and engineering floor plans, which effectively convey the geometric and semantic

information of the structure [5]. This massive amount of data, coupled with the increasing development of technology, computing power, and artificial intelligence (AI), raises the question of whether it is possible to use it to speed up the design process, automatically obtaining the shear wall layout of the building, whose calculation resulted from learning the knowledge, experience, and regulations embedded in hundreds of previous floor plans. In other words: Is it possible to capture the DNA of structural systems based on existing designs, for example, the typical Chilean construction?

In order to answer this question, the present work proposes the study and development of an AI framework that allows to automatically calculate the length, thickness, displacement, and the proposal of new structural elements from the architectural floor plan information, with the ultimate goal of accelerating the design process of the wall's layout. The thesis hypothesizes that an algorithm of this nature allows achieving an adequate structural pre-design.

Within engineering, the principle of AI aims to learn from an input and output set of data to simulate the relationship, even if the interdependence is unknown or the physical phenomenon is difficult to interpret [6]. Machine learning (ML) is an area of study within AI that allows a computer to learn without being specifically programmed [7]; ML algorithms can be classified into two main categories: unsupervised learning (UL) and supervised learning algorithms (SL) [8, 9]. UL algorithms learn by identifying the relationships in the dataset features, such as separating the data into clusters with a certain similarity degree or applying dimensionality reduction techniques. On the other hand, SL algorithms require the separation of the data in training and validation datasets. Through the training dataset, SL develops an algorithm that uses pre-specified features (input) to predict labels (output); the validation dataset is later used to evaluate the model's effectiveness. Artificial neural networks (ANN) are among the most common SL algorithms, consisting of three or more interconnected neuron layers, typically referred to as the input layer, the hidden layer, and the output layer. Neurons receive the linearly weighted neuron output from the previous layer, apply a non-linear activation function, and subsequently feed the neurons connected in the following layer. The connections' weights are updated in the network training process to minimize the prediction error [10].

Convolutional neural networks (CNN) are another standard SL algorithm in the field of machine learning, widely applied in computer vision [11] due to their intrinsic relationship with the two-dimensional tensor processing, such as the pixel matrix of an image. CNNs have a topology composed of convolutional layers, non-linear processing units, and sampling layers. The convolutional layers apply a convolution operator on the input through a kernel matrix (also known as filters), transforming the data so that certain features become more dominant in the output; for example, a 3x3 kernel can sharpen an image by assigning a higher value to the center pixel. The kernel matrices, commonly used in image processing, can be manually defined to perform different tasks such as edge detection, blurring, or contrast change; however, those trained in a CNN model allow the extraction of more abstract non-trivial features. The convolutional layers' output is later assigned to a non-linear processing unit (activation function), which helps in the abstraction capacity while learning and provides non-linearity in the feature space, generating different activation patterns for different responses, facilitating the learning of semantic differences between the data. The activation function output is

usually followed by a sampling layer (subsampling or oversampling), summarizing the results and keeping the input invariant to geometric distortions. Within computer vision, CNNs have had a significant boom in detection, segmentation, classification, generation, and image recovery tasks [12].

The use of ML algorithms in structural engineering covers a wide variety of topics [13], such as structural health monitoring [14–17], structural design [18], element strength design [19], seismic engineering [20], building construction and life cycle [21, 22], among other areas. On the other hand, the usage of images and CNN models in civil engineering is common in fields such as the monitoring of displacement, strain, stress, and vibration of structures, dynamic characteristics identification, and cracks inspection and characterization [14, 23–25]. However, there is no application of ML-based algorithms in the wall layout design of buildings powered by a prior database. Zang and Muller [26] present a numerical procedure based on topology optimization through an evolutionary algorithm, in which architectural and structural considerations are taken into account for the objective function. Tafraout et al. [27] perform the structural design of RC wall-slab buildings through a genetic algorithm model that considers the wall's length, floor area, and torsion. Lou et al. [28] optimize the shear wall layout through a meta-heuristic algorithm and support vector machine (SVM) for minimizing the structural weight, constrained on the story drift and period ratio. Similarly, Talatahari and Rabiei [29] use a meta-heuristic algorithm for optimizing the shear wall layout, considering the structural costs, flexural, torsion, shear, and drift constraints into the objective function. However, these solutions do not track previously accepted solutions in the design, and in most cases, the objective function relies on complex hand-crafted metrics, which are hard to generalize for different structural conditions or realities.

Since ML models allow characterizing the complex relationship between the input and output, this thesis chooses neural networks to obtain the shear wall's layout pre-design, which learns the complex interaction between architectural and engineering plan information from a curated training dataset. As a first approach, a regressive ANN model was developed, named Sequential model, to predict the thickness and length of the rectangles that constitute a wall in the engineering plans (output) based on an architectural 30-feature input vector, archiving remarkable R^2-value results[1]. For this purpose, a new database was built considering 165 existing reinforced concrete residential Chilean buildings, which led to calculating the geometric and topological features of the wall segments, called the Regression Engineering Neural Estimator (RENE). Each project in the novel database considers three floor levels (basement, first floor, and typical floor) for the first version of the architectural plan and the latest version of the structural engineering plan.

Although the ANN model obtained an adequate representation of each wall's thickness and length, it cannot create new structural elements in the engineering plan. Also, the prediction of other variables than the rectangle geometry has poor performance because the proposed 30-feature vector (RENE), which accounts for each rectangle's geometric and topological properties, lacks an intrinsic spatial component of the surrounding walls. The proposition of new structural elements is an essential task within engineering design; in certain situations, the original architectural layout does not satisfy the structural requirements, or the incorpo-

[1] Coefficient of determination, a measure to evaluate the linear fit of the predicted value.

ration of new walls can improve the building's response or performance.

For the reasons mentioned above, a CNN-based framework was proposed to generate the final engineering floor plan by combining two independent plan predictions, which consider the architectural data as input, such as the numerical 30-feature vector (RENE), and the surrounding walls' image for each rectangle. The first plan prediction is assembled using two regressive models for the prediction of different design parameters of the wall rectangles, such as the thickness, the length, the translation on both main axes between the architectural and engineering plans, the floor offset between the geometric center and center of mass on both main axes, and finally the change of the width and aspect ratio of the engineering floor's bounding box. The CNN regressive models show a performance improvement compared to the Sequential model that does not consider the images as input. The second plan prediction is assembled using a model that generates the most likely engineering image given an architectural input for each rectangle (images and numerical features) to propose the existence of new walls, based on the analysis of similar cases found in the training process. In this contribution, the images processed by the convolutional layers allow the extraction of a greater quantity of low, medium, and high-level features (level of abstraction) [30] of the geometric and topological building wall distribution properties. Finally, both independent predictions can be combined to generate the engineering plan in the early process of the building wall layout's conceptual design. In that case, the structural design time is reduced by generating suggestions in the geometry, the position of each existing wall, and the appearance of new structural elements, propositions learned in the model training process using a database composed of hundreds of past validated projects.

The contribution of this thesis comprises a novel database of 165 Chilean projects, a feature calculation methodology for modeling the wall objects (RENE), an association mechanism for creating input/output pairs between the architectural and engineering plans, and finally, a framework for the structural design of shear wall buildings based on artificial neural networks (ANN, CNN). The thesis also contributes two published papers (2021) on the Engineering Structures Journal (Q1), "Structural design of reinforced concrete buildings based on deep neural networks" [31] and "Use of convolutional networks in the conceptual structural design of shear wall buildings layout" [32].

1.2. Problem statement

Given the rise of artificial intelligence, the development of technology, and the large amount of data that is generated in the traditional structural design process, the problem posed in the thesis is whether it is possible to develop a methodology that allows obtaining the conceptual design of the Chilean residential building's wall layout in an automated fashion, thus accelerating the structural pre-design process.

Since artificial neural network algorithms have had a great boom in the last decade, and allow modeling the regression of complex interactions between input and output domains through the training process, is that an ANN/CNN framework is proposed to predict the engineering parameters of the wall's segments length, thickness, displacement, and new struc-

tural elements, considering as input the information from an architectural floor plan; that is, the algorithms are trained to encode the DNA of the Chilean building design.

1.3. Research questions

From the proposed problem, the following questions arose:

- How is it possible to capture the structural information from a Chilean shear wall layout building so that a regressive algorithm can process it?
- How can the regressive input-output pair be assembled, which relates the architectural and engineering information from the floor plans?
- Which are the wall's engineering parameters that can be predicted from a regressive model?
- What is the network architecture of an ANN that allows obtaining an adequate result when predicting the wall's engineering parameters, considering time and computing resources constraints?
- Is it possible to capture more spatial detail using a CNN model, and how can the framework be configured to improve the engineering floor plan prediction?

1.4. Hypothesis

An ANN algorithm can achieve an adequate structural pre-design of the shear wall layout for Chilean residential buildings. Also, CNNs allow for improved predictions by obtaining a greater number of high and low-level features from the architectural floor plans.

1.5. Objectives

1.5.1. Main objectives

The main thesis objective is the development of a structural wall layout design framework driven by an artificial neural network model, which allows to predict the engineering design parameters and generate the final structural plan from architectural input data.

1.5.2. Specific objectives

- Construction of a building database that considers the geometric and topological properties regarding each floor plan, used for the machine learning models train and evaluation processes.
- Feature engineering, a process that aims to describe each wall segment's geometric and topological properties through a numerical vector, which the machine learning models later use to train and test the results.

- Evaluation, characterization, and regression of wall segments through conventional artificial intelligence ANN/CNN models. This process considers the research of the model's architecture, properties, and the data flow needed for archiving the best results in a finite amount of time and resources, aiming to predict the engineering design parameter with high accuracy.
- Assembling the engineering floor plan from the predicted wall segments parameters, including evaluating and implementing the algorithms, constraints, and heuristics to improve metrics such as the intersection over union (IoU).
- Results analysis and comparison between the predicted and real structural floor plan for all database projects.

1.6. Thesis structure

The thesis structure is composed of six chapters, whose description is shown below:

1. **Introduction**

   This chapter illustrates the thesis motivation, including a revision of the concepts, literature, and the overall summary of the work. The thesis research question, hypothesis, objectives, and scope are also described.

2. **Database construction**

   This chapter describes the development and details of the reinforced concrete residential Chilean buildings database.

3. **Feature engineering**

   The third chapter illustrates the design and calculation of the 30 features (RENE) used by the developed ML models and frameworks, including the data augmentation and the association mechanism for assembling the train, validation, and testing databases.

4. **ANN-based model**

   The fourth chapter describes the first proposed ANN model for predicting the thickness and length of each wall segment, considering the model formulation and the results in terms of R^2, correlation charts, and confusion matrices.

5. **CNN-based framework**

   The fifth chapter describes the CNN framework that generates the final engineering floor plan by combining two independent plan predictions, where each prediction, operation, and final assembly process is described. The results in terms of R^2, the correlation charts, histograms, confusion matrices, and intersection over union metrics are detailed.

6. **Conclusions**

   The last chapter summarizes the conclusions from the proposed database, features, and frameworks.

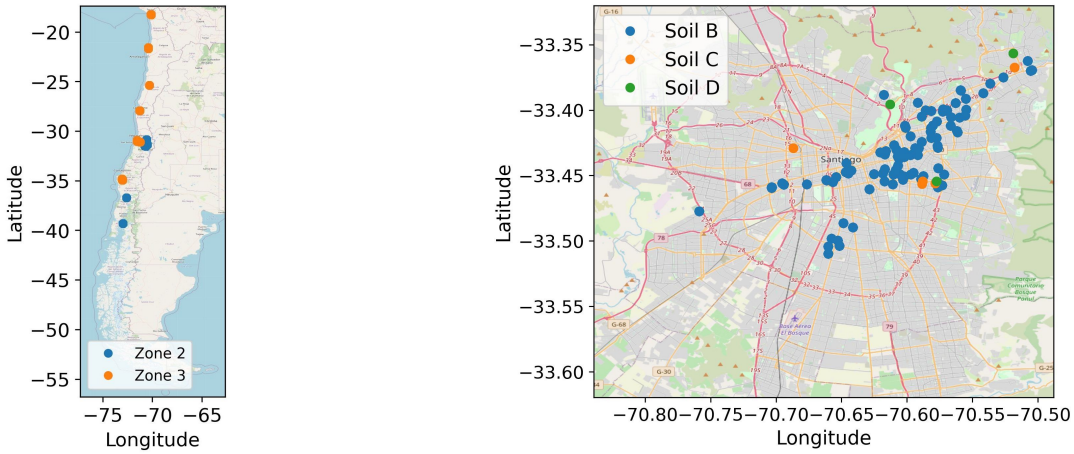
Chapter 2

Database construction

Database creation is a critical process while developing artificial intelligence-based models since they are used for both training and evaluation. Furthermore, the information contained in the data must account for features and properties of the domain under study, allowing the algorithms to learn and predict the fine-grained relationships embedded. Thus, this chapter covers the description and construction of the database, composed of 165 Chilean reinforced concrete residential buildings, which are discretized to store walls and slabs with a high level of detail.

2.1. Database description

The database comprises 165 projects of reinforced concrete residential buildings constructed in Chile, stored as digital plans for the basement, first floor, and typical floor, considering the first architectural version and the latest structural engineering plan version. The buildings are located in 17 cities throughout the country, as illustrated in Figure 2.1, organized by seismic zone and soil type [33]; however, 65% of them are located in the capital (Figure 2.1b), Santiago, followed by the northern coastal cities of Iquique and Antofagasta, with an 11.8% and 11.2% respectively (Figure 2.1a). The distribution of the project's cities location and soil type is detailed in Figure 2.2.

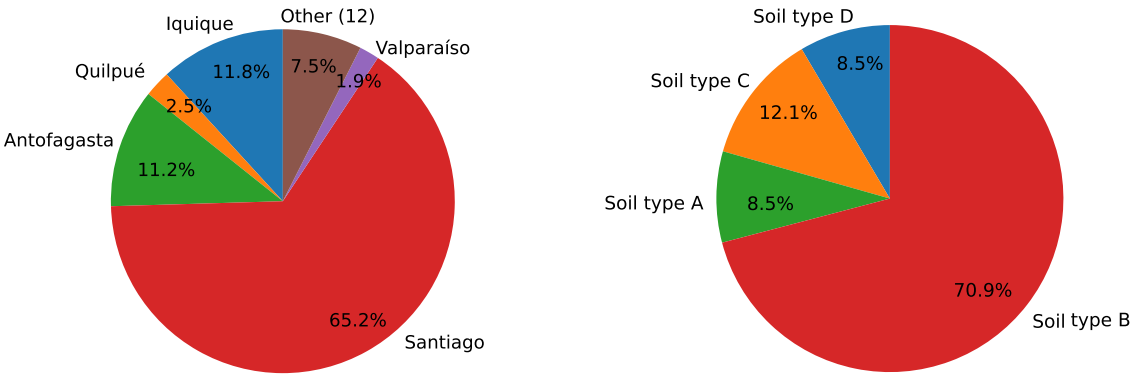


(a) Location in Chile by seismic zone

(b) Location in Santiago by soil type

Figure 2.1: Location of the projects, by seismic zone and soil type [33].

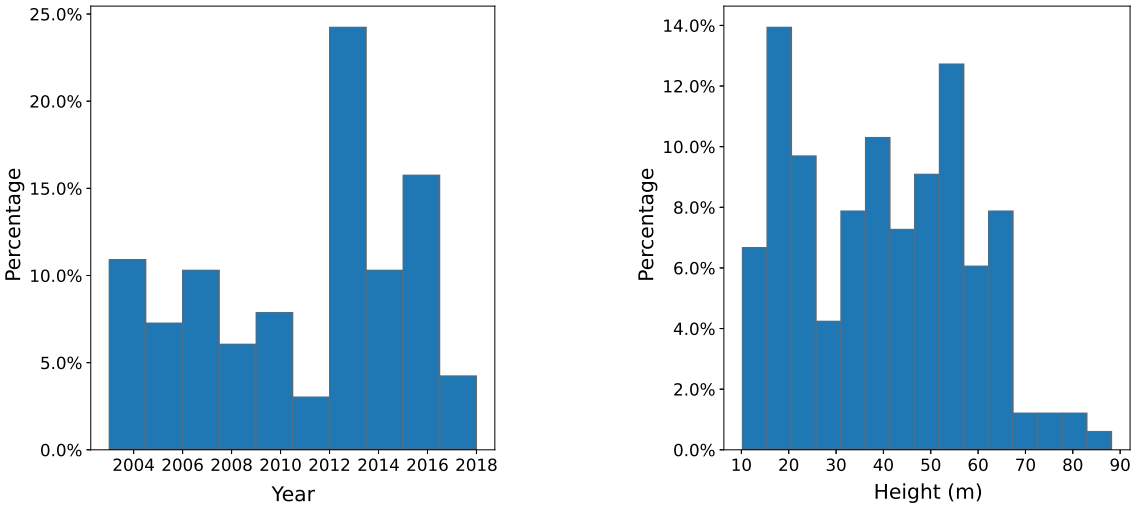
For each project, the polygon of the contour of the walls and slabs is obtained from the digital plans as an array of planar coordinates, in addition to the principal results of the design and structural analysis, such as fundamental periods in two main directions, seismic mass, foundation soil type, among other 51 parameters.



(a) Project's city location **(b)** Project's soil type

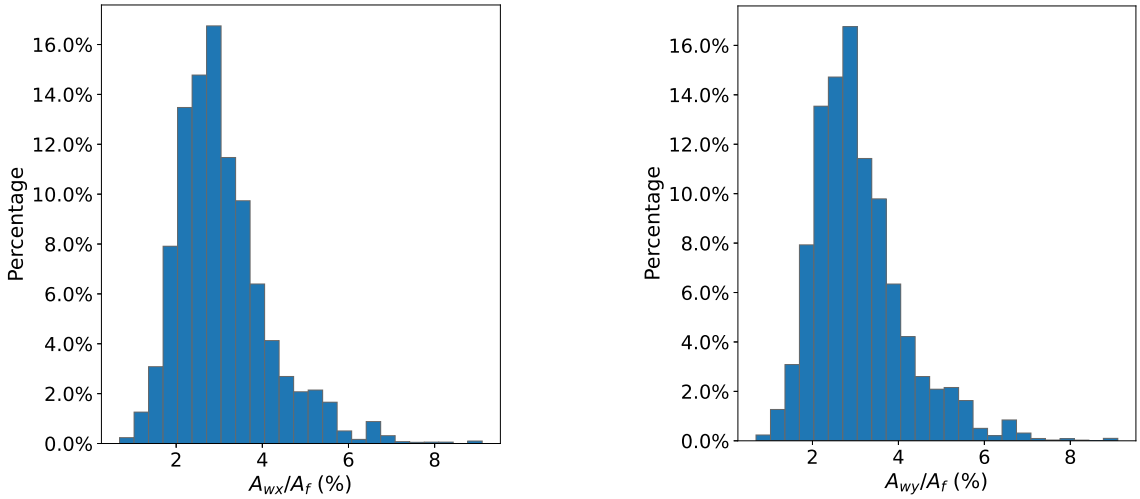
Figure 2.2: Distribution of project's location and soil type.

The projects were designed between 2004 and 2018 (Figure 2.3a) by 52 different architecture offices and calculated by the same structural engineering office. The buildings have between 5 to 35 floors, with an average floor height of 2.5 meters (m), and a maximum of 5 underground parking lots (basements), which yields a total height that ranges between 10 m and 88 m (Figure 2.3b). The average slabs' thickness is 16 centimeters (cm), and the average wall density corresponds to 3.1% in both directions (Figures 2.4a and 2.4b), typical values within Chilean residential buildings [34]. The fundamental periods on the x and y-axis correspond to 0.63 seconds (s) and 0.78 s, respectively (Figures 2.5a and 2.5b), where the x-axis corresponds to the larger side of each structure (Figure 1.1). It is important to note that the projects do not share any particular specifications; buildings with more than five floors were randomly selected. Finally, all material grades (concrete and steel) were practically the same, such that the material strength was not considered in the database.



(a) Construction year **(b)** Building height

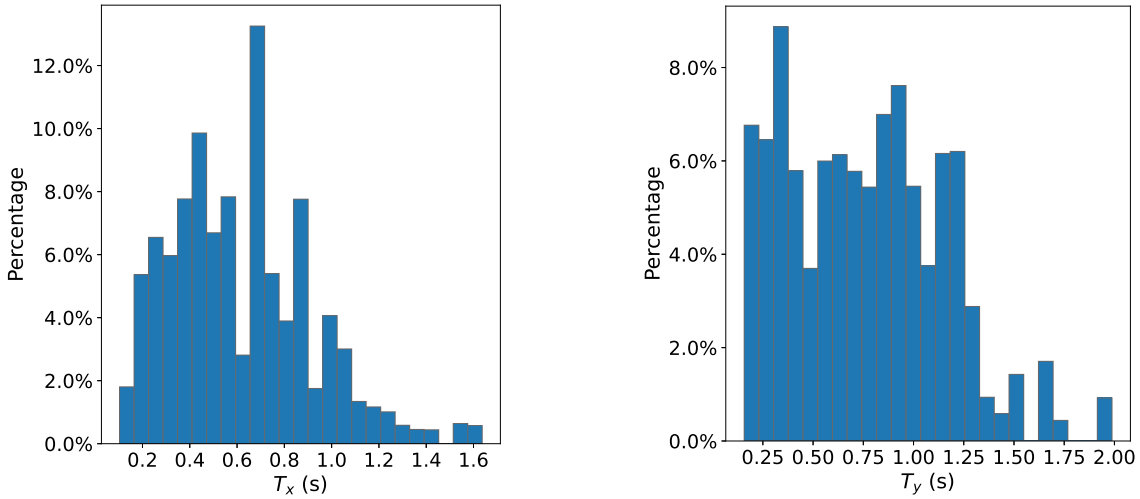
Figure 2.3: Project's building characteristics.



(a) Wall density on x-axis

(b) Wall density on y-axis

Figure 2.4: Project's wall density.



(a) Period on x-axis

(b) Period on y-axis

Figure 2.5: Building's fundamental period.

2.2. Data acquisition process

Given that the typical wall layout of residential buildings in Chile presents complex wall cross-sections [2–4] (Figure 1.1), the data analysis using complex polygons and images without contextualization or labeling is complicated and not extensible. Thus, in this work, it was decided to create the database considering a wall (complex-cross section) as a series of rectangles or wall segments connected by unique points on each floor, with an individual coordinate system per project. In this way, it is possible to assemble the structure's complete geometry, calculate properties at wall and floor levels, identify projections, or detect the elements' intersection between floors.

For the database construction, a web application was programmed, illustrated in Figure

2.6, which facilitates the manipulation of the plan images[2] and provides tools for the creation of projects, such as the importation of the wall and slab polygon coordinates from a text file, 2D and 3D visualization of the structure, and the edition of the building objects like walls, slabs, and floors using vector drawing tools. The application was made using PHP, HTML, and Javascript languages[3]. The database structure comprised a total of 9 different SQL tables and 161 fields.

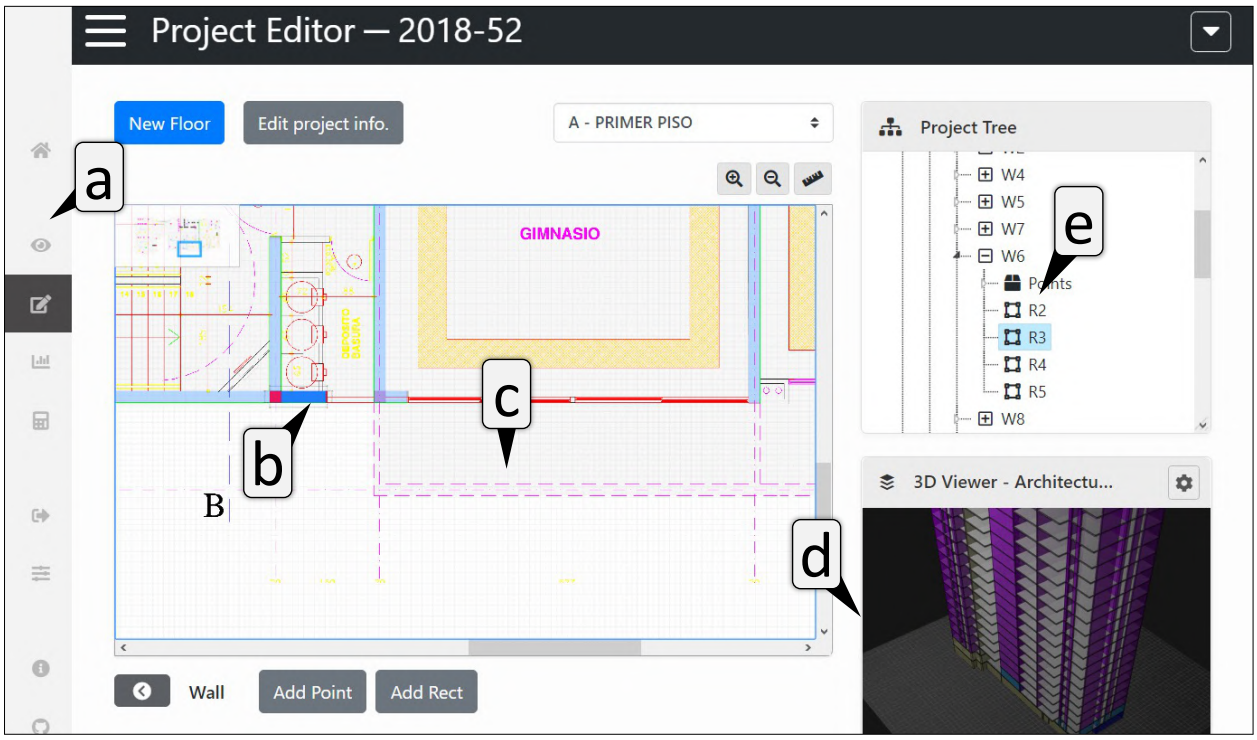


Figure 2.6: Implemented software (MLSTRUCTDB) – Main components: **(a)** Application menu to access the different modules, **(b)** Vector drawing interface being able to create the structural elements in a simple way using only an image of the plan, **(c)** Example of a wall discretization based on rectangles and points, **(d)** 3D viewer of each structure, and finally **(e)** Object structure of each project.

A project comprises a collection of floor plans in both architecture and engineering development. Each floor has an image, a scale in pixel-meter factor, an offset that indicates the displacement in planar coordinates relative to the lower floor, the height of the floor, and finally, its vertical position. A floor contains slabs and walls as objects; the slab in this context was stored as the complete polygon obtained from the digital drawing, without considering openings, and its thickness; the wall is a collection of points (*Point*) and rectangles (*Rect*). A point consists of each vertex's planar coordinates based on the floor coordinate system. A rectangle is an element that represents the discretization of the wall, uniquely defined by connecting two points with a certain thickness and insertion point (local direction). This model makes it possible to determine the structure's geometry with a high level of detail. In order to simplify the problem, the beams were not incorporated in the database, neither

[2] Saved as a PNG file with 9,000 pixels on the longest side for transparency support.
[3] The programmed web application comprises 76,000 lines of code, and the Python software (later discussed) yields 48,000 lines of code.

were included the reinforcing bars, and it was assumed that the walls' height is equal to the height between floors. Figure 2.7 illustrates a simplified schematic of the object system; all objects have a unique identifier used for linking and assembling the logical hierarchy.

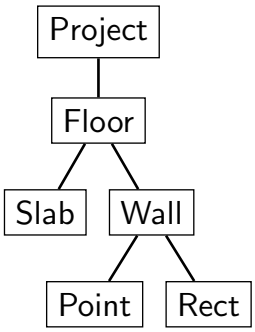
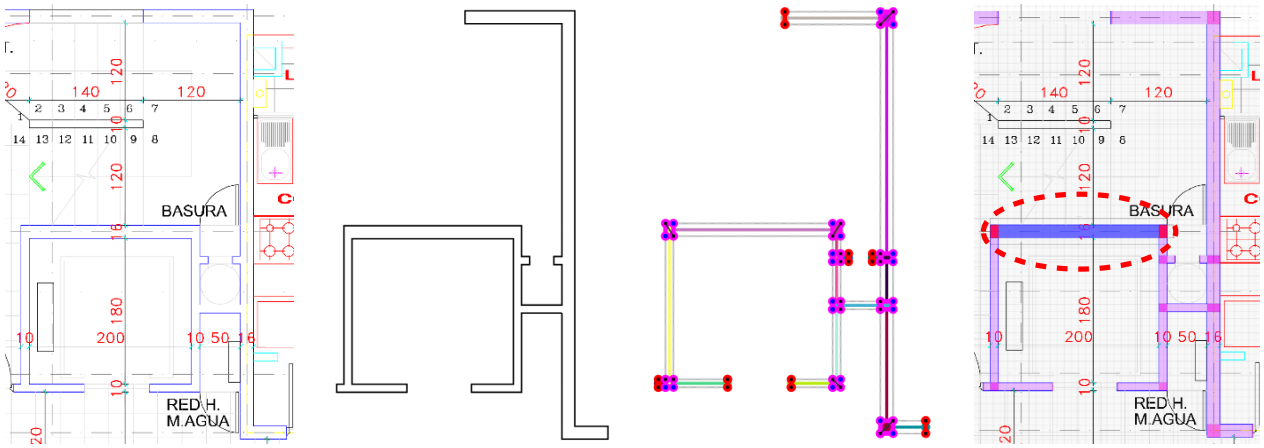


Figure 2.7: Organization of the structural discretization objects.

The walls' construction process consisted of four stages, represented in Figure 2.8: First, the architectural and engineering plans were collected, correcting the digital drawing layers (Figure 2.8a); then, the walls' contour polygon (Figure 2.8b) was created and saved into a text file per floor [35]. From these data, each polygon was processed, discretizing the complex cross-section figure into a series of rectangles connected by points (Figure 2.8c), applying various algorithms in the process, such as convex hull [36], vertex projection, outlier removal with statistical methods, and planar graph analysis for identifying the connections. For each wall, the implemented algorithm costs $O(n^3)$, where n is the number of vertices; despite the high cost, n is ten on average, thus, the process takes less than a tenth of a second for an entire floor. For automation purposes, it was considered that each corner of the wall must be orthogonal with a tolerance of 10°, that there cannot be intersections between edges, and that the thickness of each wall must be limited to a maximum of 0.5 m. Any other case is considered rejected and created manually with the web application editor using the plan image as a guide. Finally, the polygon is imported from the text file, processed, and located in the plan's correct position (Figure 2.8d).



(a) Original plan in digital format

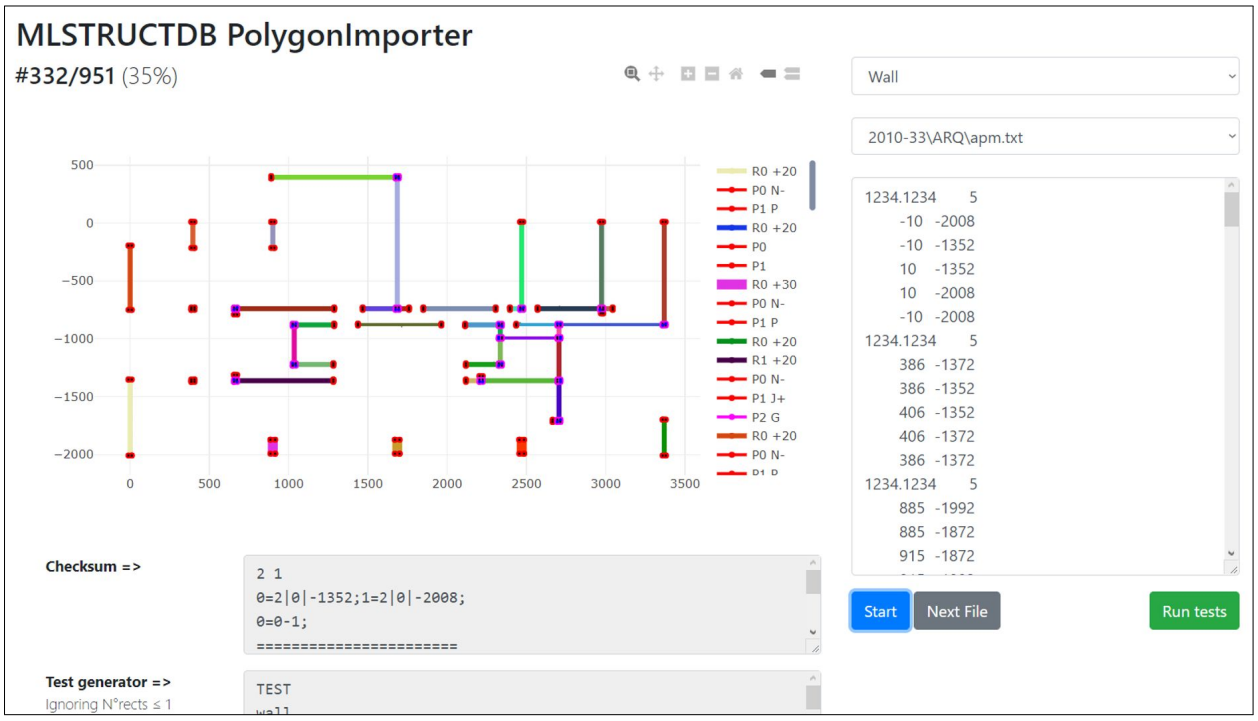
(b) Polygon of the wall contour

(c) Wall discretization in rectangles and points

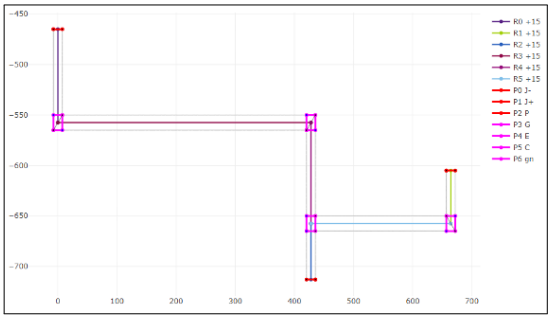
(d) Modeling of the wall (a rectangle is highlighted)

Figure 2.8: Example of the walls' discretization process.

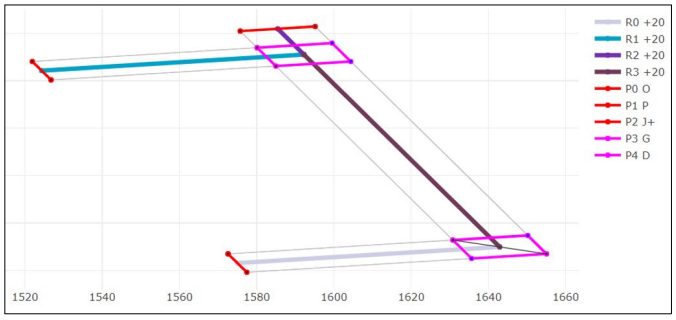
A database of 4000 previously verified polygons was used to validate the correct execution of the algorithm; for each polygon, the topology was verified using a custom visualization interface, illustrated in Figure 2.9. Due to the data's significant variability, a visual inspection was necessary for all projects' walls after processing, detecting a rejection percentage of less than 15%, which were manually corrected. Usually, the discretization error was associated with the architectural plan, as the layers were not well defined, the polygons were invalid, or in some cases, the thickness or angles between walls were too large.



(a) The software included tools for loading, visualizing, and testing the wall's discretization. A floor example is shown composed of 19 walls, 42 rectangles, and 61 points



(b) Example of a discretized wall



(c) The wall distribution contains objects with different angles, orientations, shapes, and thickness

Figure 2.9: Visualization of the geometry and topology for several wall examples using the web application.

Slabs were processed similarly, differing only in the discretization, where the element was saved as the complete perimeter polygon obtained from the digital drawings. Openings inside the slab area, such as the elevator shaft, were not considered; outliers were also removed, and in some cases, a manual correction had to be made. However, the relatively simple shape of the elements leads to fewer errors in the whole process.

Finally, the floors were adjusted to fit the vertical continuity of the wall objects. For such purpose, an offset was calculated, that is, a displacement on the x and y-axis of each floor that maximizes the area intersection between elements. The offset was obtained by performing an iterative grid search ranging from 3 m to 10^{-8} m, requiring a visual inspection to ensure the offset was adequate. On average, each project took about 6 hours to be entirely created, imported, and corrected using the web application (Figure 2.6). The correction took most of the time because the invalid discretized walls must be manually drawn over the plan images, considering exact dimension and position. Figure 2.10 illustrates the 3D visualization of a selection of ten structures, representing various configurations in shape and the number of floors. Finally, Table 2.1 details the number of total elements included in the database at different levels, such as the project, floors, wall points, wall rectangles, and vertices, used to compute the feature vector (Chapter 3).

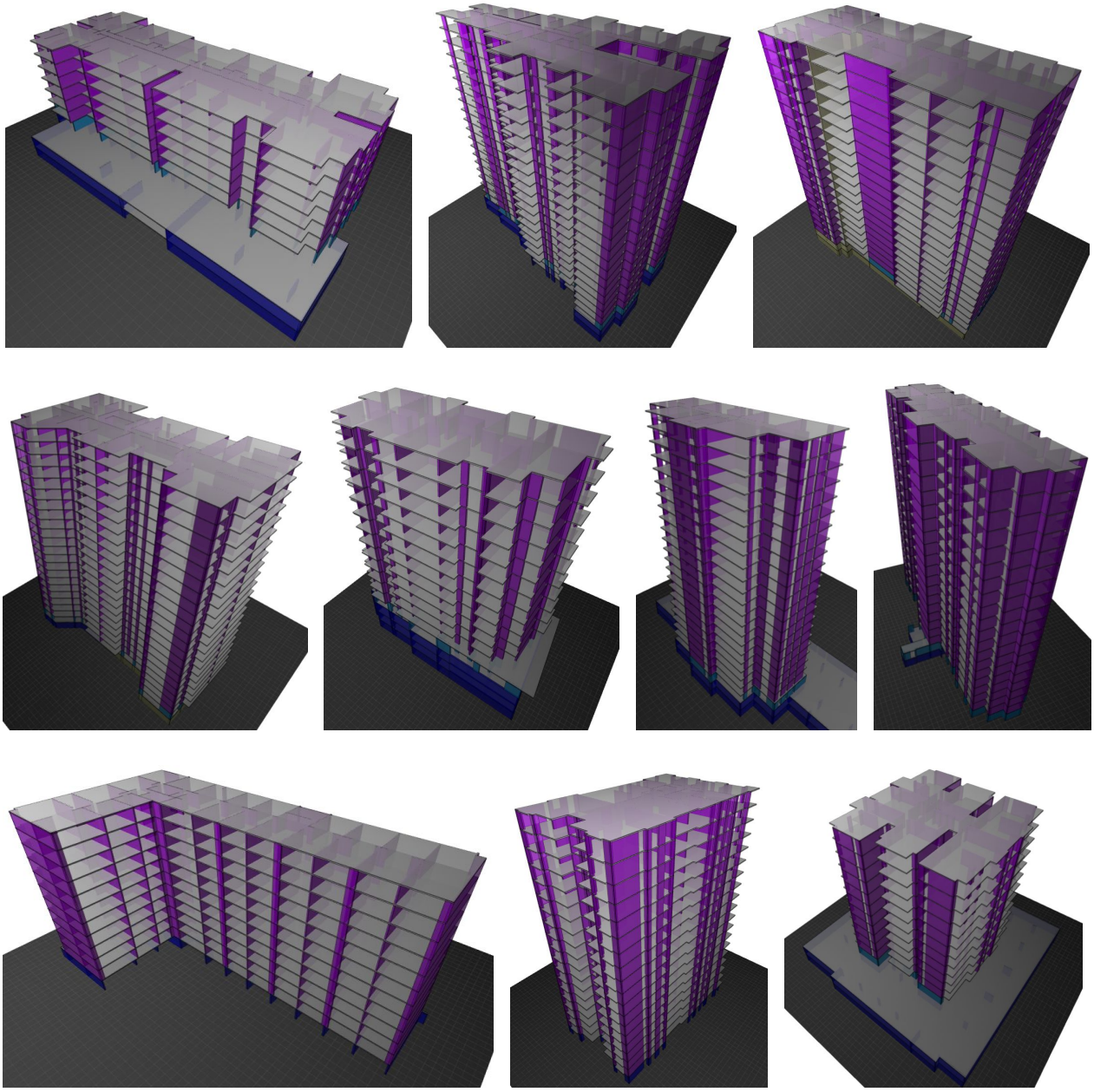


Figure 2.10: 3D visualization of a random project selection.

Table 2.1: Number of total elements in the database.

Object	Total
Projects	165
Floors	954
Slabs	954
Walls	28,527
Wall points	99,078
Wall rectangles	70,871
Vertices	287,088

2.3. Construction of the images for each rectangle

The surrounding walls' image was built for each rectangle, both for architecture and engineering plans, to incorporate greater geometric and topological features into the developed models. The Python libraries Matplotlib and OpenCV [37] were used for image generation. Each image has two classes of pixels: the background (white pixel) and the walls (black pixel), where 5.4% of pixels correspond to walls. The proposed image size is 64x64 pixels (px), where the source rectangle is centered and covers a region of 10x10 m, leading to a 5 m margin between the center of a rectangle and the perimeter of the image, a representative distance within Chilean structuring [34], resulting in a factor of 0.156 m/px (meter/pixel). In this way, walls with a thickness less than 7.5 cm disappear in the image, and the 20 cm thickness walls, typical within the structures present in the database, cover a total of 2 px. Figure 2.11 illustrates a 64x64 px image example from a source rectangle (highlighted in red) with a cropped region of 10x10 m.

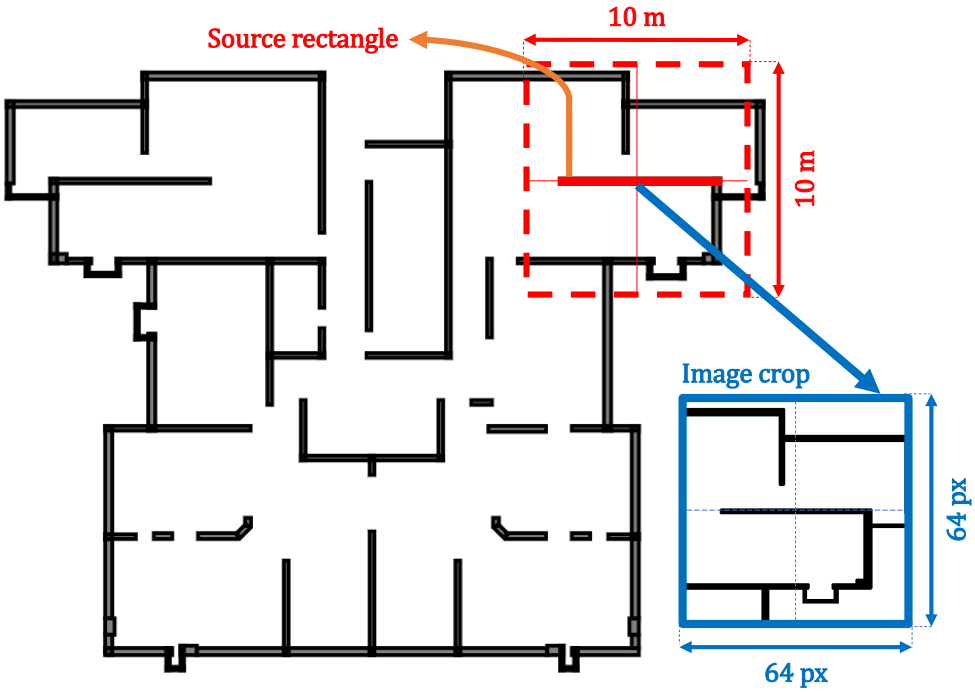
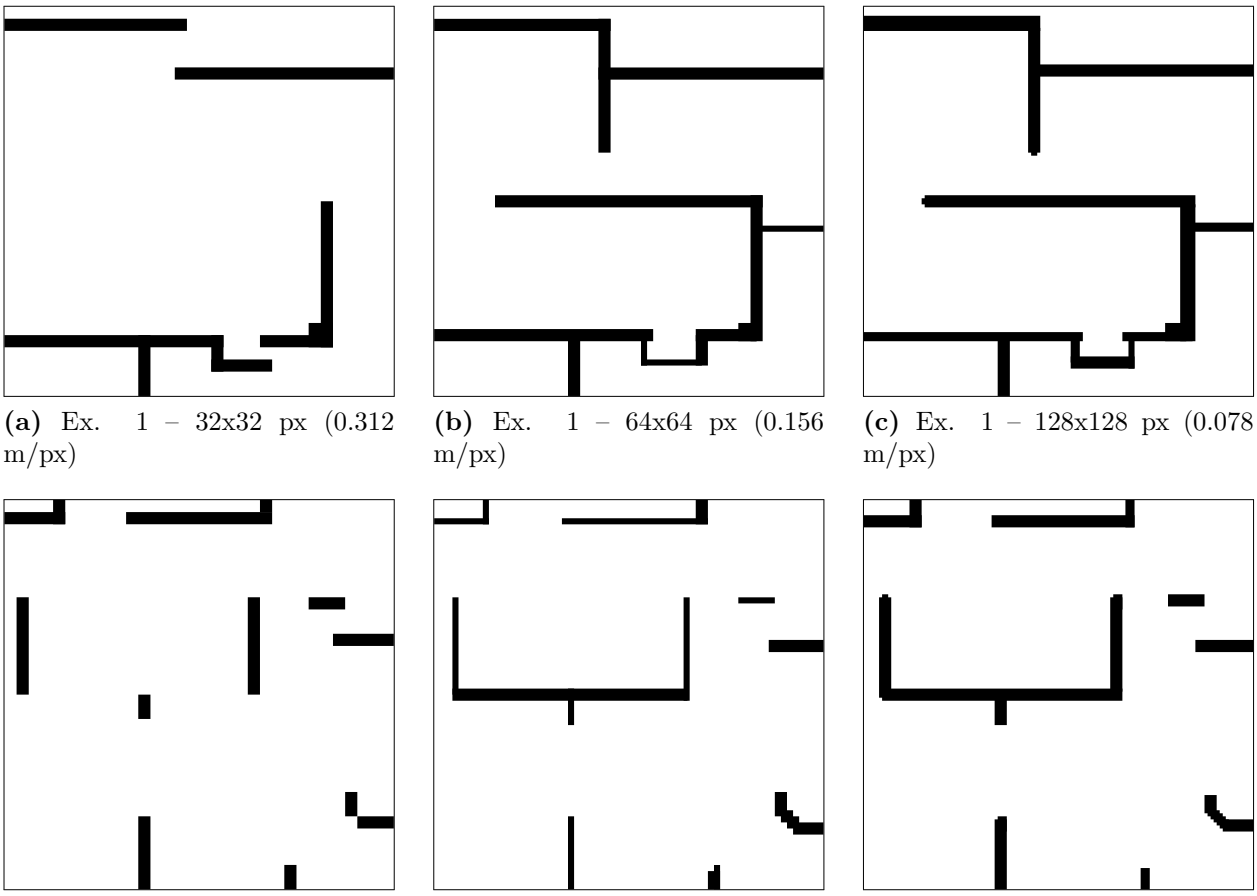


Figure 2.11: Image crop generation surrounding a rectangle in a plan (architecture).

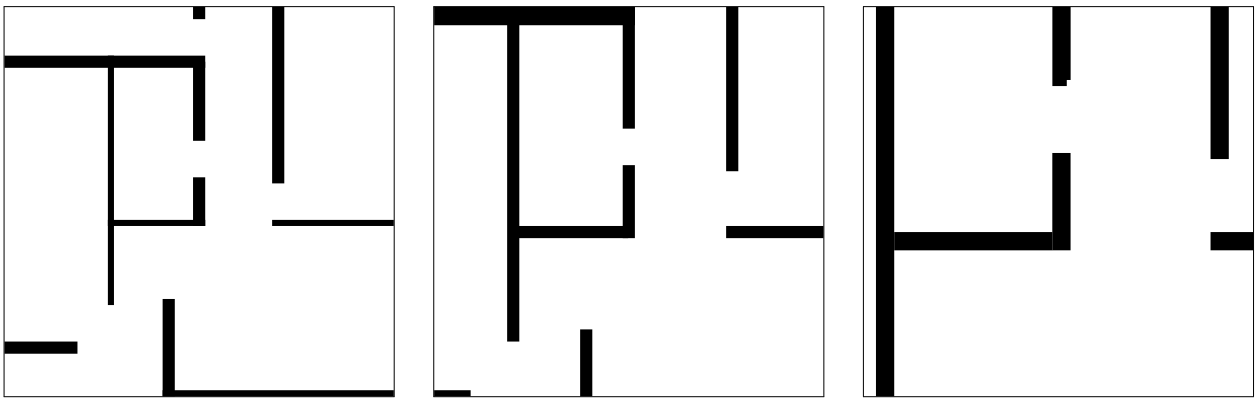
The size of the cropped region is related to the type of structure under analysis. If a limit of 10 cm per pixel is considered, a region up to 6.4x6.4 m (64x64 px image) can be selected without compromising the wall rectangle's definition. Figure 2.12 shows an example of images from two rectangles with different m/px factors. It is possible to observe that the 32x32 px images (Figures 2.12a and 2.12d) suffer from significant information loss associated with the walls of small thickness. In the case of 64x64 px images (Figures 2.12b and 2.12e), they are similar to the 128x128 px ones (Figures 2.12c and 2.12f) in terms of shape, with small differences in the elements' thickness, despite being an image four times smaller in size.

A loss of detail or context is observed as the crop region becomes smaller than 10x10 m, as illustrated in Figure 2.13. On the other hand, the size of the image is related to the convolutional layer's parameter number and the model's memory usage, such that the larger the image, the better m/px factor can be achieved, at the cost of higher computing time and resources while performing data processing, training, and evaluation. For these reasons, an image size of 64x64 px was chosen in the development of all models.



(a) Ex. 1 – 32x32 px (0.312 m/px)
(b) Ex. 1 – 64x64 px (0.156 m/px)
(c) Ex. 1 – 128x128 px (0.078 m/px)
(d) Ex. 2 – 32x32 px (0.312 m/px)
(e) Ex. 2 – 64x64 px (0.156 m/px)
(f) Ex. 2 – 128x128 px (0.078 m/px)

Figure 2.12: Resolution effect for two examples of images with sizes of 32, 64 and 128 px (pixels) with a 10x10 m region.



(a) 10x10 m margin (0.156 m/px) **(b)** 7.5x7.5 m margin (0.117 m/px) **(c)** 5x5 m margin (0.078 m/px)

Figure 2.13: Effect of the plan crop margin for a 64x64 px size image.

Chapter 3

Feature engineering

The design and calculation of features provide the necessary information to machine learning algorithms for train and evaluate the models, allowing them to learn the complex relationships present in the data and predict new ones.

In this chapter, first, the numerical features used as input and output of the models are detailed, which describe the discretized rectangular building objects in the geometric and topological domains. Next, the features were augmented and associated to construct the training pairs. This process is known as feature engineering and contributes the foundation for the ANN algorithms development, later explained in this work.

3.1. Design and calculation of features

Since the regression's central objective is at the rectangle level (wall segments), the feature calculation must account for geometric properties, such as the thickness, length, angle, area, and inertia. Topological properties that indicate the relationship of the element with the building's objects (Figure 2.7) are also considered, such as the distance between walls and the rectangle's projection on the lower or upper floors. Finally, properties other than geometry or topology, for example, the soil type and the seismic zone, are included. For the calculation, a Python API was programmed, which downloads a project from the database, assembles the building object hierarchy, and performs the algorithms. The implemented features, named the Regression Engineering Neural Estimator (RENE) are detailed in Table 3.1; the ones estimated in two orthogonal directions or relative to the upper or lower floor levels are represented with two values, yielding a total of 30 features.

From the geometric features, *RectThickness* belongs to the thickness of each rectangle, obtained directly from the database. *RectLength* calculates the length of the rectangle as the most significant distance of the convex hull [36] from the vertices of the contained points. *RectAngle* is the angle respect to the minor axis of the floor (y), normalized between 0 and π. *RectFloorMassCenterDistance* is the distance between the center of mass (MC) of the floor and the center of mass of the rectangle; for the center of mass, the rectangles' average area (A_i) on each main axis (x_i, y_i) was considered ($MC_x = \sum A_i \cdot x_i / \sum A_i$ and $MC_y = \sum A_i \cdot y_i / \sum A_i$). *RectWallMassCenterDistance* is the distance between the center of mass of the wall containing each rectangle and the center of mass of the floor. *SlabThickness* is the thickness of the floor's

slab to which each rectangle belongs. *FloorMassCenterDistance* is the distance between the geometric center and the center of mass of each floor. *FloorAreaNormalized* is the sum of the areas on each main axis of all walls belonging to the floor, normalized by the slab's area. *FloorBoundWidth* is the length of the floor's minimum bounding box. *FloorAspectRatio* is the aspect ratio of the length and width of the floor bounding box. *FloorHeight* is the height of the floor that contains the rectangle (total height of stories at either the basement, the first floor, or the typical floor levels). Finally, *FloorInertiaNormalized* is the sum of the decoupled moment of inertia I_{Nx} and I_{Ny} calculated according to Eqs. 3.1 and 3.2, as:

$$I_{Nx} = \sum \left(\frac{I_x + I_y}{2} + \frac{Ix - Iy}{2} \cdot \cos(2\alpha) \right) \cdot \frac{1}{I_{px}} \tag{3.1}$$

$$I_{Ny} = \sum \left(\frac{I_x + I_y}{2} - \frac{Ix - Iy}{2} \cdot \cos(2\alpha) \right) \cdot \frac{1}{I_{py}} \tag{3.2}$$

Where $I_x = \frac{Lt^3}{12}$, $I_y = \frac{L^3 t}{12}$, $I_{px} = \frac{BH^3}{12}$, $I_{py} = \frac{B^3 H}{12}$, B and H are the length and width of the bounding box; L, t, and α are the length, the thickness, and the angle of the rectangle.

Table 3.1: Description of the 30-feature input vector (RENE).

Feature	**Type**	**Description**	**No. (1)**
RectThickness	Geometry	Rectangle thickness	1
RectLength	Geometry	Rectangle length	1
RectAngle	Geometry	Rectangle angle (0–π) **(2)**	1
RectFloorMassCenterDistance	Geometry	Relative distance between MC of floor and rectangle **(2)**	2
RectWallMassCenterDistance	Geometry	Relative distance between MC of wall and floor **(2)**	2
SlabThickness	Geometry	Slab thickness	1
FloorMassCenterDistance	Geometry	Distance between MC and GC of each floor **(2)**	2
FloorAreaNormalized	Geometry	Sum of area in each principal axis divided by slab area **(2)**	2
FloorBoundWidth	Geometry	Floor bounding box width **(2)**	1
FloorInertiaNormalized	Geometry	Sum of uncoupled inertia of all walls normalized by floor inertia **(2)**	2
FloorAspectRatio	Geometry	Floor aspect ratio **(2)**	1
FloorHeight	Geometry	Floor height	1
RectTriangulationArea	Topology	Triangulation slab area of each rectangle	1
RectProjectionIntersection	Topology	Rectangle's area projection in lower and upper floors (%)	2
RectThicknessProjection	Topology	Rectangle's thickness variation in lower and upper floors (%)	2
RectAxisDisplacement	Topology	Distance between connected rectangle axis **(2)**	2
RectRelativeDistance	Topology	Normalized nearest distance to each rectangle in a different wall **(2)**	2
PointConnectivity	Topology	Point connectivity **(2)**	2
ProjectData	Other	Building seismic zone and soil type	2
		Total:	30

(1) Number of values per feature

(2) Value modified with data augmentation

Figure 3.1 illustrates a scheme of nine geometric features from a rectangle (hatched). The center of mass (MC) and geometric center (GC) of the rectangle, wall, and floor, is represented with a cross symbol.

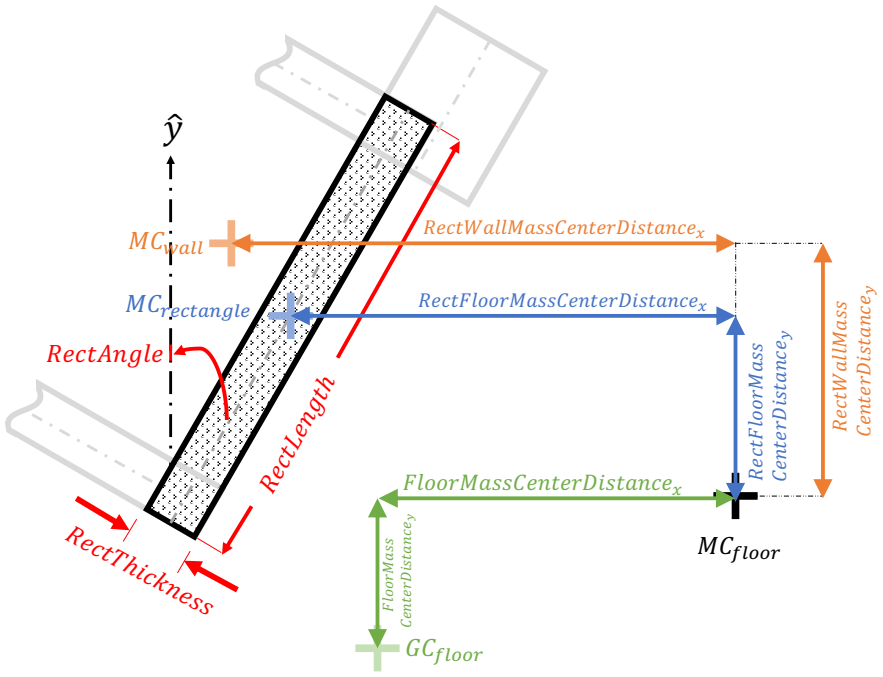


Figure 3.1: Scheme of a subset of nine geometric features from a given rectangle (hatched).

Regarding the topological properties, *RectTriangulationArea* corresponds to the sum of the triangle areas resulting from Delaunay's restricted triangulation [38, 39] that shares an edge with the rectangle, equivalent to a measure of the tributary area of the slab in such wall. *RectProjectionIntersection* is the percentage of the intersection of the rectangle area with the upper and lower floor rectangles, where 100% indicates that a rectangle has complete continuity and 0% indicates that the rectangle disappears; this parameter was calculated using the Shapely geometric manipulation library [40]. *RectThicknessProjection* is the thickness projection of each rectangle, to indicate the variation with respect to the lower and upper floor levels. *RectAxisDisplacement* is the distance between the strong axes of the rectangles connected by the two points on the same main axis. *RectRelativeDistance* corresponds to the distance of the closest rectangle in a different wall on the same and different axis. Finally, *PointConnectivity* indicates the number of connections that the two points of the rectangle have, ordered according to their position on the Cartesian axes. Additionally, the project's main characteristics are also included as a feature (*ProjectData*), such as seismic zone and soil type according to the Chilean seismic code [41].

Figure 3.2 illustrates a scheme of seven topologic features from a given rectangle (hatched); *RectAxisDisplacement-1* is not visible (zero value) as there is no other rectangle connecting the element in Point 1 (bottom left) on the same axis. The red squared hatch in Figure 3.2 represents a rectangle on an upper or lower floor that intersects the element. The blue triangular hatch represents the slab Delaunay triangulation area with the rectangle as an edge; also, an adjacent wall with an L-shaped cross-section (gray rectangles) is included.

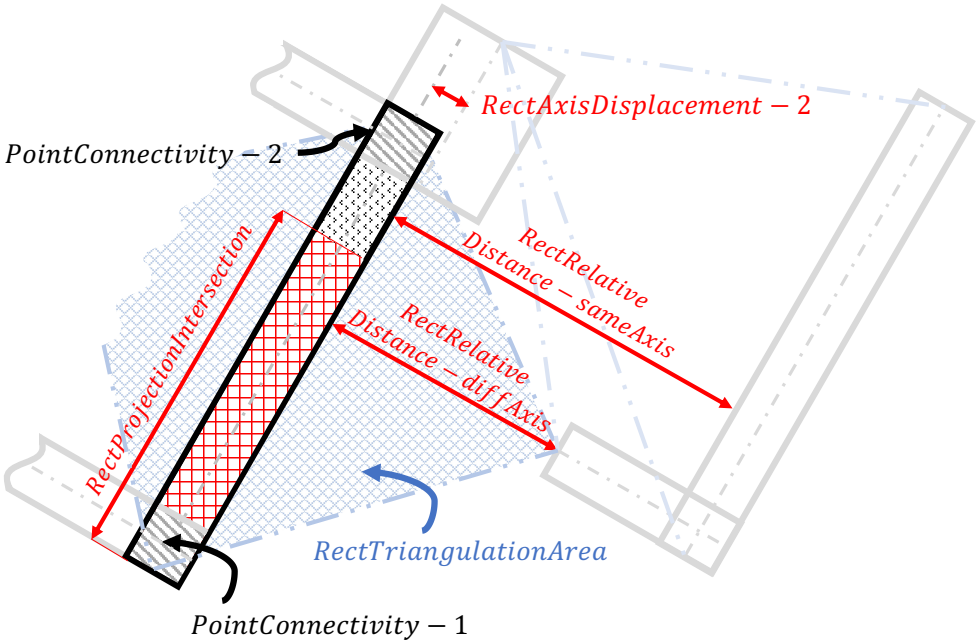


Figure 3.2: Scheme of a subset of seven topologic features.

It is important to note that if a building to be analyzed is different from the current work is always possible to withdraw or propose new features; for example, in buildings with different shapes, materials, design codes, locations, or more floor configurations. In this work, the three-floor configuration, such as the basement, first, and typical floor, is considered in *RectProjectionIntersection* feature.

3.2. Data augmentation

The projects' features were calculated considering variations in the building floors' angle and scale, simulating different structuring possibilities, leading to an augmented database, which helps the model training process, reducing overfitting [42]. Being s_x and s_y the scale factors of each project, and θ the rotation angle in the plan, eight different combinations were constructed, detailed in Table 3.2. In the case of the scale factors, it is only considered to mirror or not the wall configuration on each axis (-1 or 1), and the rotation angle considers four possible values (-90°, 0°, 90°, or 180°).

Table 3.2: Combinations used in data augmentation.

Combination	θ (°)	s_x	s_y	Dataset
1	0	1	−1	A, B, C
2	90	1	−1	B, C
3	−90	1	−1	B, C
4	180	1	−1	B, C
5	0	1	1	C
6	0	−1	−1	C
7	90	−1	−1	C
8	−90	−1	−1	C

Thus, three different feature datasets were created, where dataset A is the original case without alteration, containing a total of 36,916 rectangles; dataset B considers four combinations, with 147,651 rectangles, and finally dataset C consists of all eight combinations described in Table 3.2, with 295,302 rectangles. It was experimentally evidenced that the best results are obtained with dataset C. The angular and scale transformations introduced by the augmentation mechanism modify 63% of the features, marked in Table 3.1 as note (2). Figure 3.3 shows the correlation matrix where the features' linear independence is verified, with a mean correlation for all features of 0.043 ± 0.22, indicating little correlation for most cases. The highest correlation occurs in the thickness projection between floors with a 0.9 value (*RectThicknessProjection* vs. *RectProjectionIntersection*), followed by the correlation between wall areas on the x and y-axis with a 0.8 value (*FloorAreaNormalized-x* vs. *FloorAreaNormalized-y*), and the aspect ratio of the bounding box with the width (*FloorAspectRatio* vs. *FloorBoundWidth*), with a correlation of 0.7. Despite these single pairs of high correlation values, those features are used as they improve the model response. Figure 3.4 illustrates the histogram of all 30-feature values (RENE) of dataset C for architectural plans; some features take discrete values, such as the angle or the soil type.

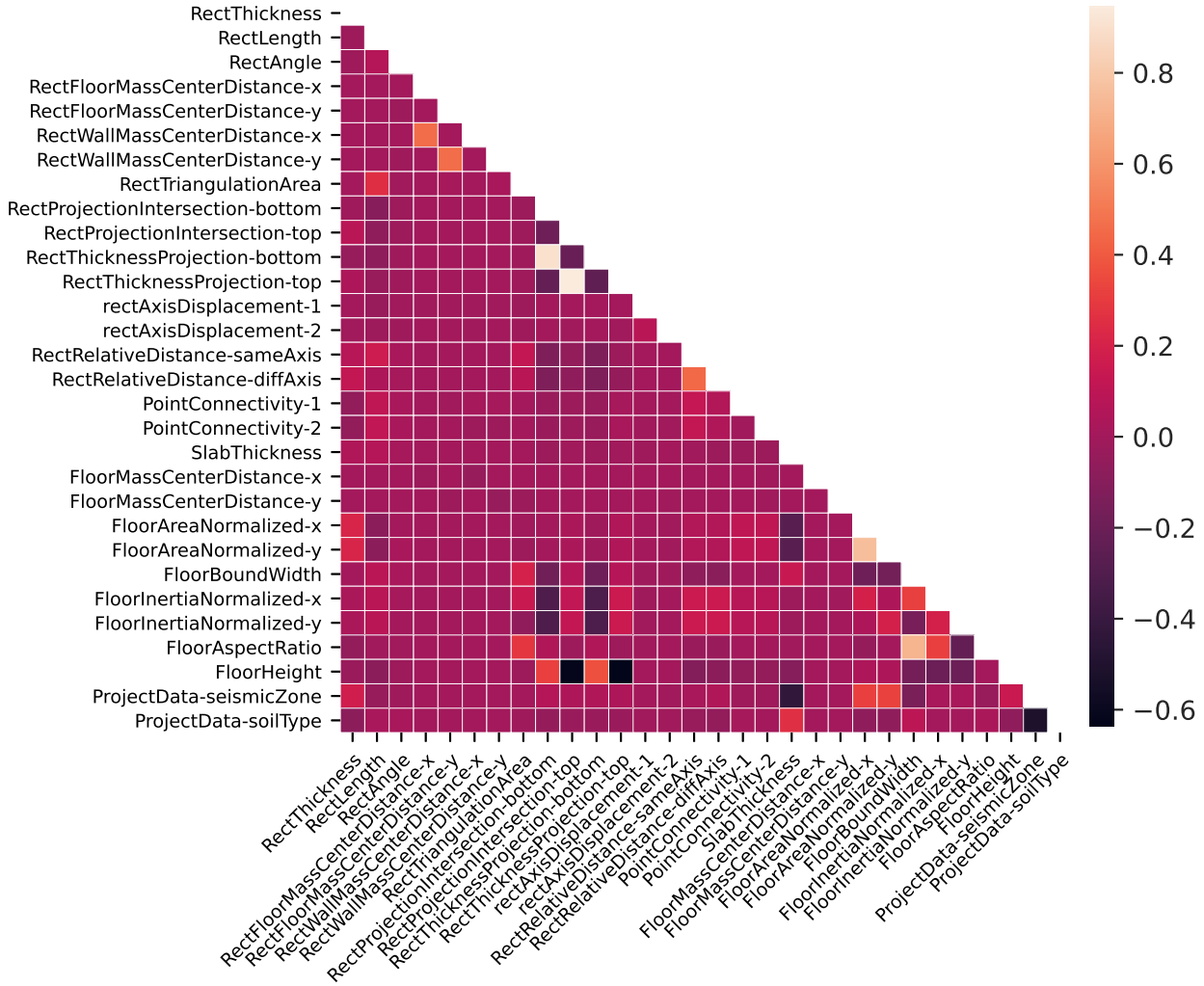


Figure 3.3: Correlation matrix of 30 rectangle features (RENE) in architecture with data augmentation (dataset C).

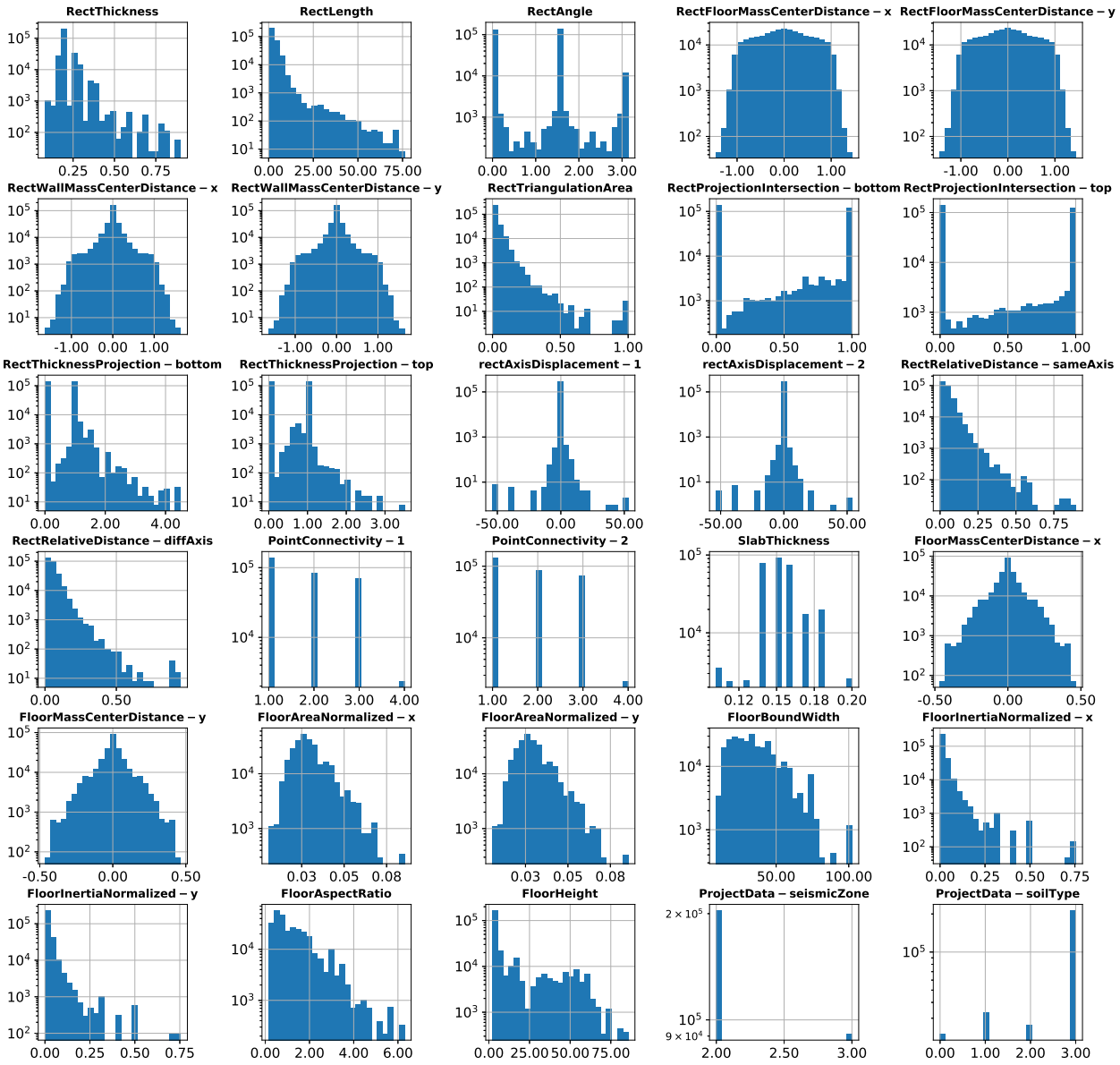


Figure 3.4: Histogram of 30 rectangle features (RENE) in architecture with data augmentation (dataset C).

3.3. Feature association

Each wall must be associated between the architectural and engineering plans to create the regression's model input/output vector, that is, given a rectangle in architecture r_A, a rectangle r_B must be found in the engineering plans that better represents the pair (r_A, r_B). This problem was solved by calculating a heuristic score S_c (Eq. 3.3) assigning the engineering rectangles with a value above the mean plus a standard deviation for a given architectural element; in this way, a single r_A can be associated with several rectangles r_B that share a similar score. S_c considers the weighted sum of six different factors, such as the distance between the geometric centers $\boldsymbol{d_G} = \chi\left(\frac{d(r_{AG}, r_{BG})}{D}\right)$, the smallest

distance between the polygons $\boldsymbol{d_P} = \begin{cases} \chi(dP_{AB}) & dP_{AB} < 0.75D \\ S_c(r_A, r_B) \to 0 & dP_{AB} \geq 0.75D \end{cases}$, the orthogonal distance between the strong axes of the rectangles $\boldsymbol{d_A} = \chi\left(\frac{d_\perp(r_{AP}, r_{BP})}{0.5D}\right)$, the angular difference $\boldsymbol{d_\theta} = \begin{cases} \chi\left(\frac{\Delta\theta}{30°}\right) & \Delta\theta < 45° \\ S_c(r_A, r_B) \to 0 & \Delta\theta \geq 45° \end{cases}$, the difference of the length $\boldsymbol{d_L} = \chi\left(\frac{|r_{AL} - r_{BL}|}{r_{AL}}\right)$, and finally the percentage of the polygons' areas intersection $\boldsymbol{d_I} = \frac{\cap(Area_{r_A}, Area_{r_B})}{Area_{r_A}}$, such that:

$$S_c(r_A, r_B) = 0.1d_G + 0.3d_P + 0.1d_A + 0.25d_\theta + 0.2d_L + 0.05d_I \tag{3.3}$$

Where $D = 0.1\sqrt{B^2 + H^2}$ corresponds to 10% of the diagonal distance with a mean of 5.3 ± 1.51 m, B and H correspond to the width and height of the floor's bounding box. $\chi(x) = 1 - |\tanh x|$ is the proposed nonlinear normalization function, which gives greater importance to values around 0 and almost zero-value outside the range $(-3, 3)$, as illustrated in Figure 3.5. r_{AG} and r_{BG} belongs to the geometric centers of rectangles r_A and r_B (similar notation for other cases). $d(x, y)$ is the Euclidean distance between two polygons on the Cartesian plane. $dP_{AB} = min(d(r_{AP}, r_{BP}))$, where r_{AP} and r_{BP} are the polygons of the rectangles composed of four vertices. $d_\perp(r_{AP}, r_{BP})$ is the orthogonal distance between the main axes of the polygons. $\Delta\theta = |\theta_{r_A} - \theta_{r_B}|$ corresponds to the angular difference, where θ_{r_A} and θ_{r_B} are the angles of the rectangles with respect to the minor axis of the building plan. r_{AL} and r_{BL} are the lengths of the rectangles. Finally, $Area_{r_A}$ and $Area_{r_B}$ are the representation of the rectangle's polygon area. Since the heuristic mechanism compares each rectangle with all the rectangles on its floor (n), the algorithm cost is $O(n^2)$, where n is 74 on average.

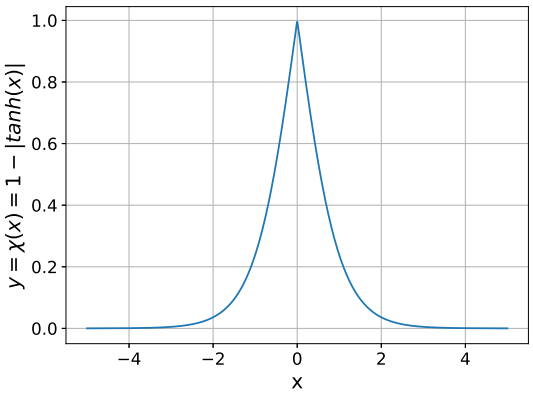


Figure 3.5: Normalization function $\chi(x)$ proposed in the calculation of the association score S_c.

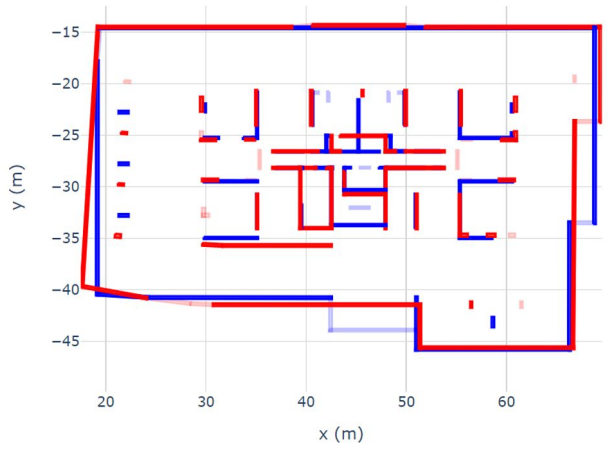
The score $S_c(r_A, r_B)$ can take any value between 0 and 1; however, after reviewing the function's behavior, a minimum tolerable value of 0.45 was established. The association is not performed for values below this limit; that is, the rectangles in architecture that do not have an engineering-related rectangle were associated with an element of zero thickness and length, indicating that such architectural elements should disappear in the engineering plans. The original idea was to consider only the closest rectangle (d_P), but this approximation does not provide satisfactory results. From Eq. 3.3, the highest weights belong to the distance

between polygons d_P, the angular difference d_θ, and the length difference d_L, representing 75% of the final score. If only those parameters are chosen for the association, the score results differ by $8.1 \pm 4.1\%$; however, the pairing quality decreases. Table 3.3 summarizes the total association pairs in the architectural and engineering datasets; for each case, 11% of the architectural plans' rectangles disappear in the engineering plans (no association).

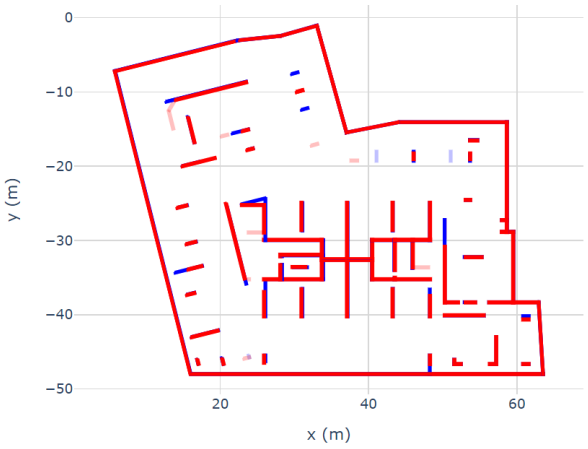
Table 3.3: The number of associations for each dataset.

Dataset	N° rectangles	N° no association
A	36,916	3,956
B	147,651	15,798
C	295,302	31,643

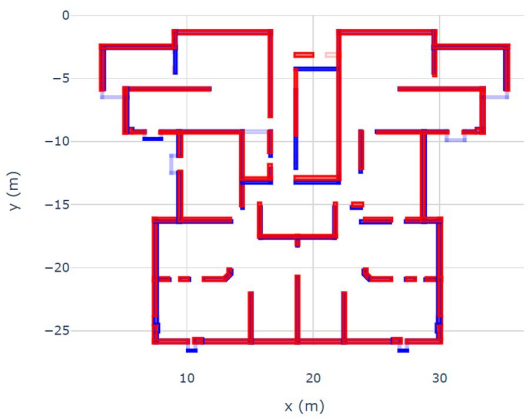
Figure 3.6 illustrates the association results for two different building floors. The elements that disappeared can be identified, illustrated with a light color, both in architecture (blue) and engineering (red) plans. Figures 3.6a and 3.6b corresponds to a basement level, 3.6c to a first-floor level, and Figure 3.6d to a typical floor type.



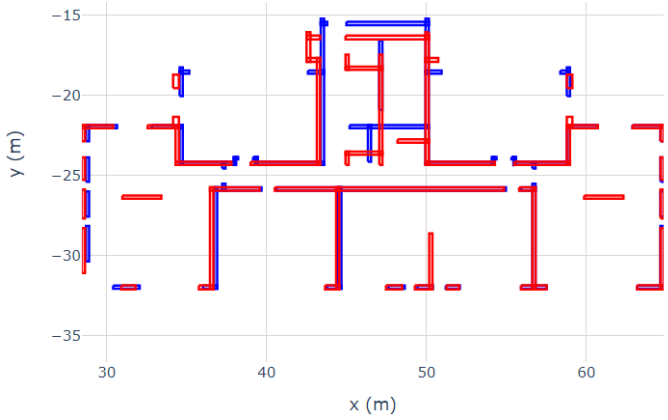
(a) Basement



(b) Basement



(c) First-floor level



(d) Typical floor level

Figure 3.6: Association examples between architecture (blue) and engineering (red) plans.

Chapter 4

ANN-based model

Artificial neural networks (ANN) are among the most common machine learning algorithms, consisting of three or more interconnected neuron layers of which its connection weights are updated through the training phase to minimize the prediction error. The algorithm learns the complex interactions between the input and output domains through this process, creating a hyperdimensional mapping later used to predict new values.

This chapter summarizes the proposed network architecture based on ANN and the model results while predicting the thickness and length of the wall segments. In the whole process, the aim is to maximize the coefficient of determination (R^2), which contrasts the difference between the output and target. This value is improved through the model training, where it learns the information embedded in thousands of geometric and topological rectangle features, encoding the DNA of the Chilean residential building design.

4.1. ANN model formulation

A model based on artificial neural networks (ANN) was proposed, named Sequential model, assembled by an input layer of 30 features detailed in section 3.1 (RENE), six fully connected (FC) hidden layers with 1024 neurons each, and an output layer with the values to predict the thickness (*RectThickness*) and length (*RectLength*) of the wall rectangles. ANN has been used because of the simplicity of assembling complex model architectures by joining different pieces (layers, number of neurons, and regularization) and the tools there exist to manipulate them reliably.

The input and output data were normalized in the range 0–1 using Min-Max scaling transformation. The model's regularization was carried out with three batch normalization layers (BN) [11] inserted at the network's beginning. The dropout regularizing layer [43] was not applied as it worsens performance for ratios of 0.2 to 0.5. The Keras library in Python [44] with TensorFlow-GPU as a backend was used for the implementation. The layer activation function corresponds to ReLU [45], the loss function is set as the Mean Square Error $MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$, and the optimizer was Adam with a learning rate (LR) α of 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-7}$ [46]. Uniform Glorot [47] was used as the weight initializer; no regularization was applied in the kernel's hidden layers. The model, illustrated in Figure 4.1, has a total of 5,290,106 parameters, where 4156 of them are not trainable.

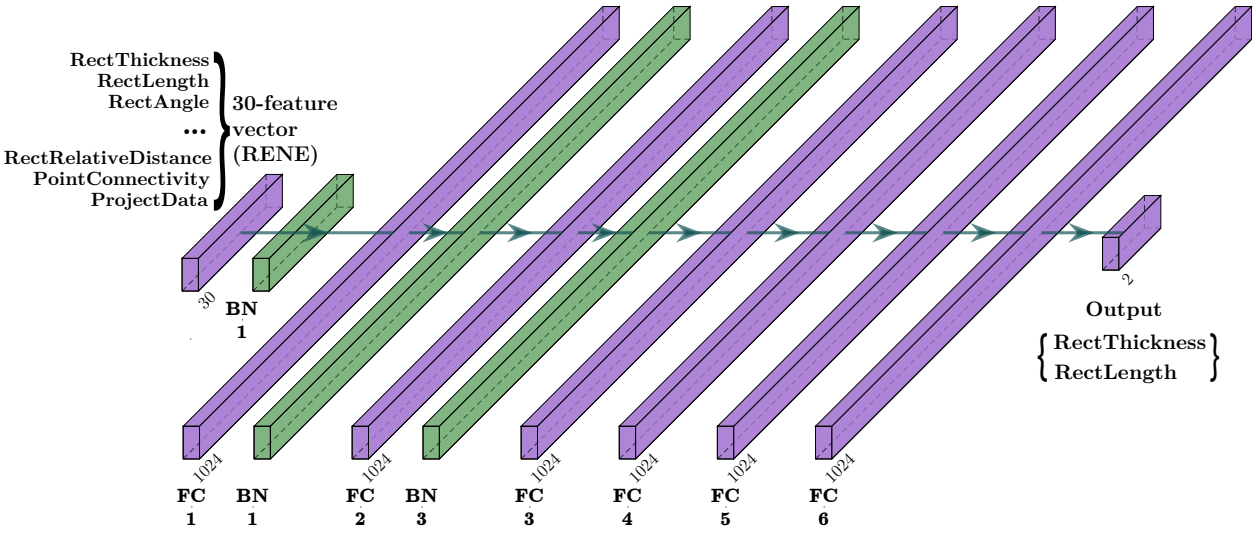
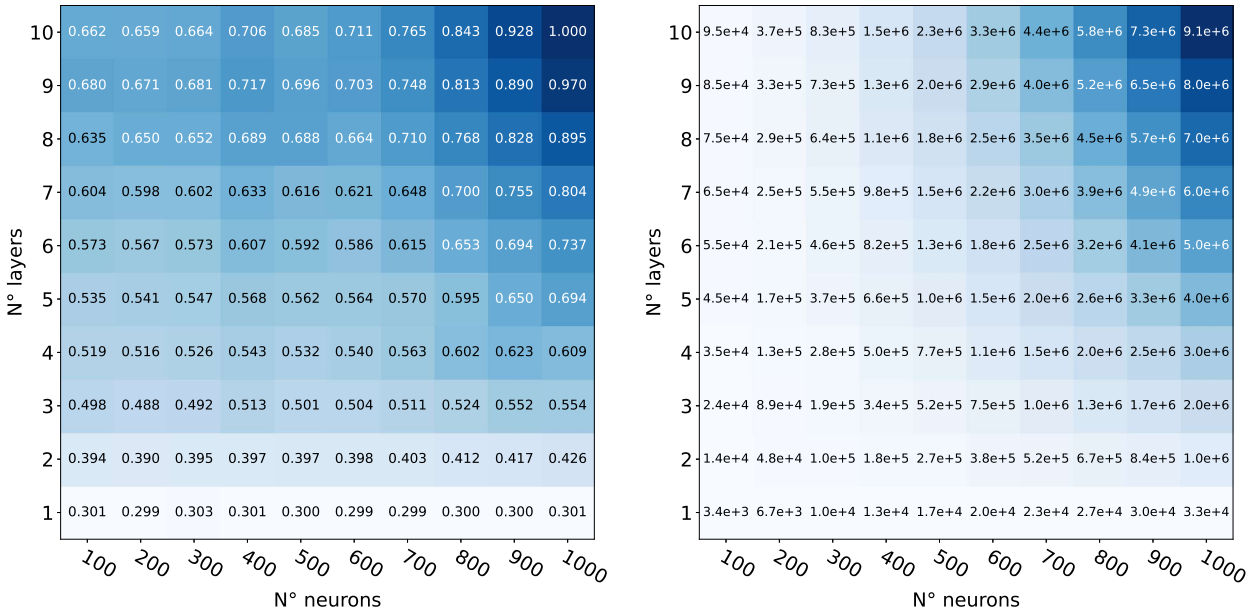


Figure 4.1: Scheme of the deep neural network Sequential model architecture for the wall thickness and length regression.

Since the number of neurons, hidden and regulation layers modify the model's predictability capacity, different combinations were manually tested to heuristically find the best configuration in terms of the model's coefficient of determination. No optimization algorithms were used to determine these hyper-parameters that maximize the model response [48] due to the dataset's large size. In terms of the ANN algorithm cost, it increases exponentially with the number of layers and neurons. Figure 4.2 illustrates the cost for several combinations of layers and neurons in terms of the training time per epoch (Figure 4.2a) and model number of parameters (Figure 4.2b), using a batch size of 64; in both situations, the larger the architecture, more extensive is the time it takes to train and the memory needed.



(a) Normalized epoch training time **(b)** Model number of parameters

Figure 4.2: Artificial neural network costs considering the Sequential architecture but with different combinations of layers and number of neurons.

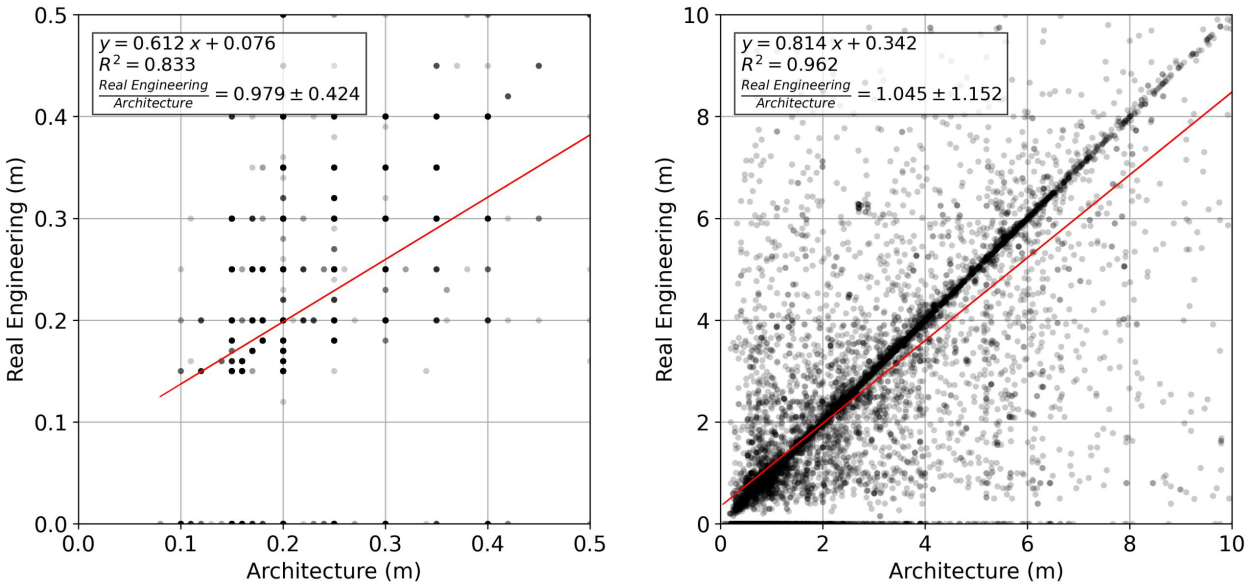
The model's training was carried out with mini-batches of size 64 and a maximum of 100 epochs. After the end of each epoch, the batch data was randomly reordered to reduce the variance. Early stopping regularization was used to avoid overfitting, with a tolerance of 40 epochs. Also, to accelerate the convergence, Keras *ReduceLROnPlateau* monitor was used for reducing the learning rate of the model optimizer if the loss metric does not improve in a margin of 15 epochs, with a decrease factor of 10^{-1} and a minimum delta of 10^{-4} for the monitored value. The dataset partition in training and testing was 70% and 30% in random order, respectively; on the other hand, from the training dataset, 20% of the data was used for validation, and 80% to train the model.

4.2. Model results

The Sequential model results were analyzed in terms of the coefficient of determination (R^2) and the confusion matrix for both network output variables (thickness and length), using only the non-augmented test partition. Without the model's application, the R^2-value for the thickness and length of datasets A, B, and C is 0.833 and 0.962, obtained from comparing the architectural and engineering rectangle's features for each associated pair (section 3.3). Note that the datasets have an identical value as they use the same test partition for comparison purposes, only differing in the training group because of the augmented data; thus, 100%, 25% (1/4), and 12.5% (1/8) of the testing data were used in datasets A, B, and C, respectively.

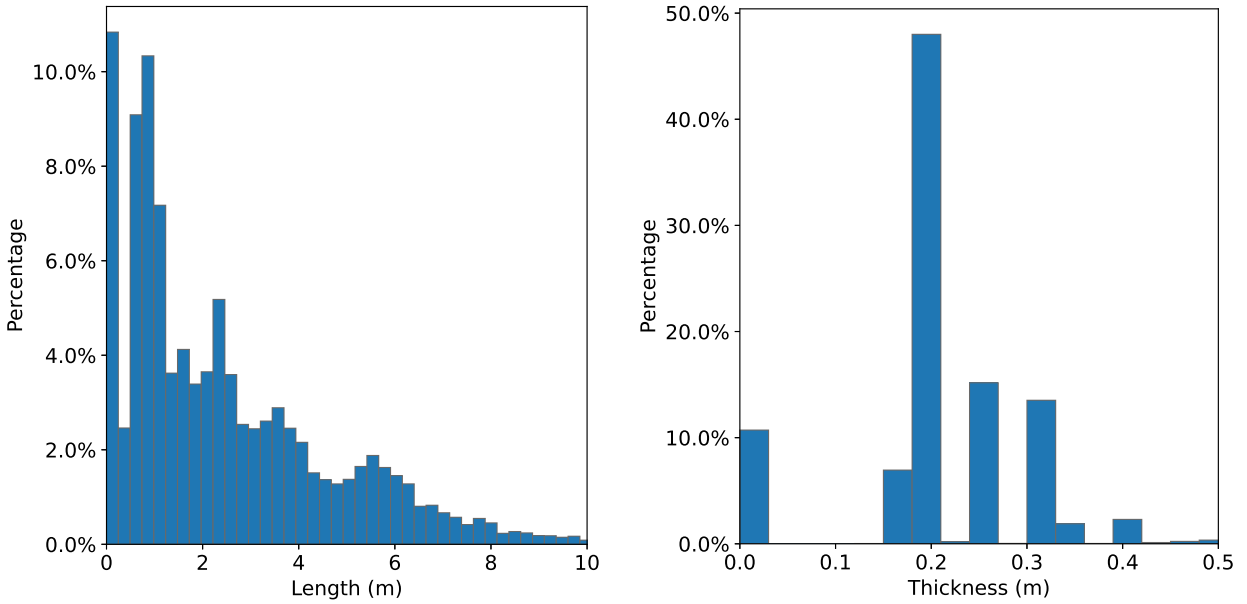
Figure 4.3 illustrates the correlation of dataset C test partition without the model application; it is worth noticing that the thickness (Figure 4.3a) presents a considerable dispersion for values greater than 20 cm, a range that represents 34% of the data, characterized by its discrete nature since typically the values are rounded to the centimeter. Also, near 11% of the rectangles in the engineering vector have a zero-value thickness, due that these objects could not pair with any architectural one during the association process (section 3.3). The length (Figure 4.3b) has a better correlation, with greater dispersion in the case of walls smaller than 5 m, a region that concentrates 83% of the data. The red trend lines in Figure 4.3 also indicate that the wall length in architecture is a reasonable estimate for the length in engineering; however, a worse correlation is observed for the thickness output.

The mean and standard deviation of the ratio between the real engineering and architectural values are also shown in Figure 4.3 as "$\frac{\text{Real Engineering}}{\text{Architecture}}$." It is worth mentioning that the length ratio deviation is high because the extreme values of the distribution are larger, mainly due to outliers of the heuristic association mechanism. 10.9% of the length ratio is equal to zero, same as the percentage of not-associated rectangles in engineering, 16.9% is under 0.5–ratio, and 9.0% is over 1.5–ratio; by contrast, 10.9% of the thickness ratio is equal to zero, 11.1% is under 0.5–ratio, and 3.4% is over 1.5–ratio. The histogram for the thickness and length real engineering values in the test partition, considering the same limits as the correlation graphs, is represented in Figure 4.4.



(a) Thickness **(b)** Length

Figure 4.3: Correlation between architectural and engineering features without applying the Sequential model in the test partition of dataset C.

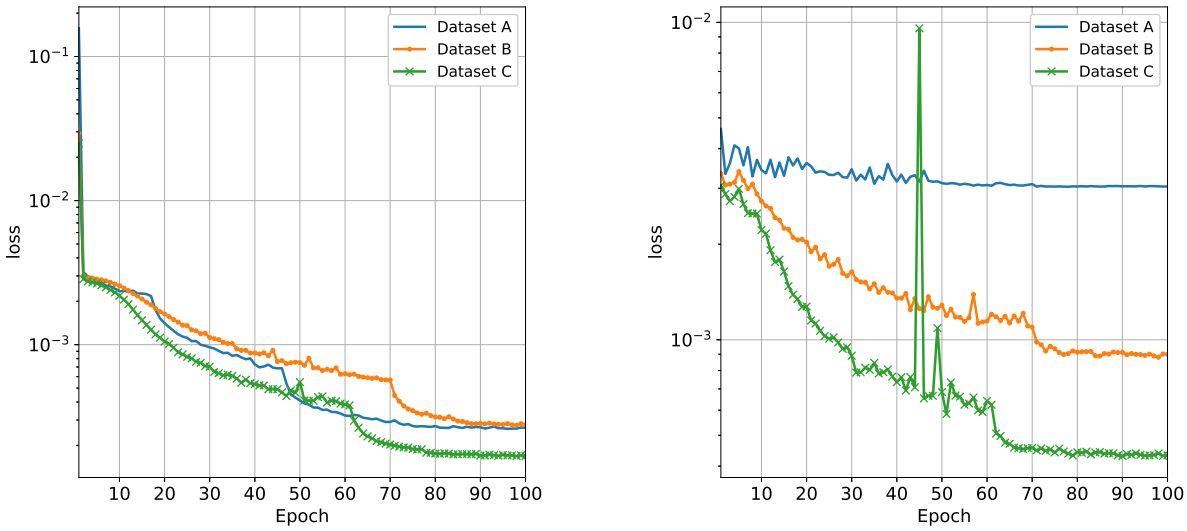


(a) Thickness **(b)** Length

Figure 4.4: Histogram of the engineering features from test partition of dataset C.

The variation of the model training loss function, illustrated in Figure 4.5a, indicates rapid convergence in the first three epochs, achieving a plateau after 90 epochs. From the validation (Figure 4.5b), the generalization capacity with dataset A is lower than the case with data-augmentation datasets B and C, which translates into a lower R^2-value for both predicted variables. Datasets B and C show better behavior in terms of loss decrease as the model trains and validates. By contrast, dataset C presents the best behavior in both training and

validation due to the more significant data variation. Keras ReduceLROnPlateau callback allowed to accelerate the model's convergence, evidenced in the abrupt loss fall after epochs 20 and 60. It was not necessary to stop the training due to overfitting problems (early stopping regularization). When using other losses such as $MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$ (Mean Absolute Error) or $MAPE = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$ (Mean Absolute Percentage Error), the model took much longer to converge, so they were not used.



(a) Training **(b)** Validation

Figure 4.5: Training model loss curves (MSE) for different datasets.

Table 4.1 details the main results in terms of R^2-value for each dataset test partition after running the model, and the time the models took for training[4]. Dataset C obtained the best R^2-value results of 0.995 and 0.994 when predicting the wall rectangle's thickness and length, respectively; however, if the methodology is run ten times using different train-test partitions, the mean R^2-value for thickness and length is 0.99 ± 0.003 and 0.989 ± 0.006, indicating that the model is almost insensitive to the train-test partition selection given the large size of the dataset. For comparison purposes, Table 4.2 shows the R^2 regression model results trained with a total of 26 features for the input vector (architecture) after removing the previously discussed high-correlation features *RectThicknessProjection-bottom*, *RectThicknessProjection-top*, *FloorAreaNormalized-y*, and *FloorAspectRatio* (section 3.2). In this scenario, the model performance worsens (between 1% and 6%) on each test dataset; then, the 30-feature input vector (RENE) is maintained for all analyses.

Table 4.1: Main results of the regression model for each dataset (test partition) for the complete 30-feature input vector (RENE).

Dataset	Train size (with augmentation)	Test size (without augmentation)	R² thickness	R² length	Train time
A	25,841	11,075	0.708	0.902	4 min
B	118,133	11,075	0.965	0.964	20 min
C	206,711	11,075	0.995	0.994	45 min

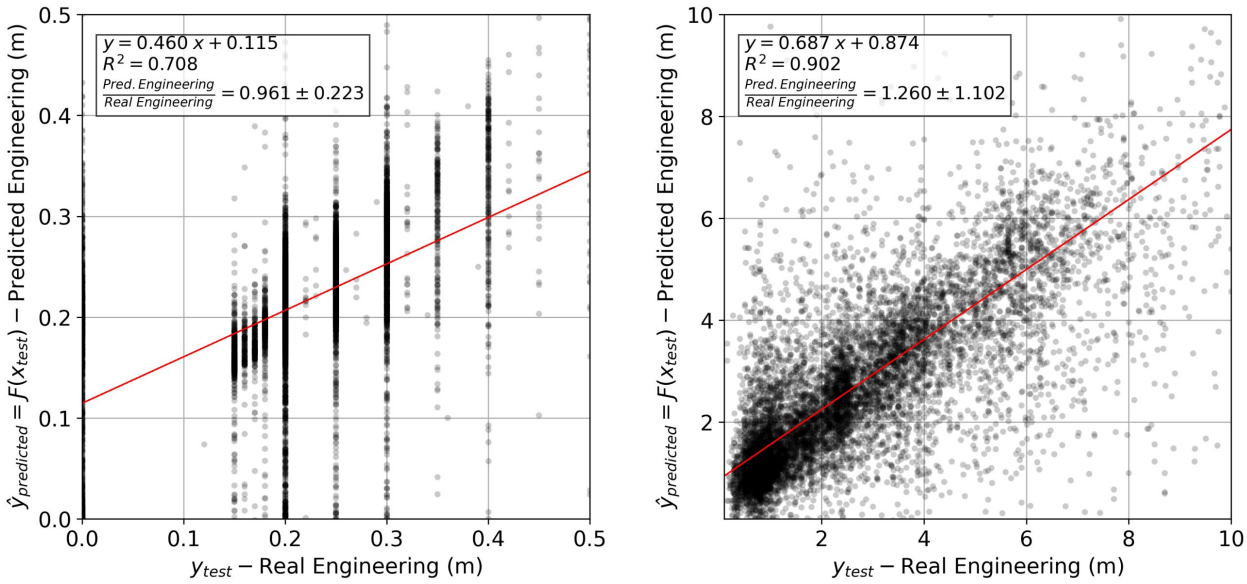
[4] For the thesis development, an Intel® Core™ i7-9750H (12M Cache, 4.5 GHz, 6 cores) CPU, 16GB DDR4-2666 RAM, and NVIDIA® GeForce RTX™ 2070 8GB GDDR6 GPU was used.

Table 4.2: Regression model results for each dataset considering the reduced 26-feature vector as input.

Dataset	R^2 thickness	R^2 length
A	0.676	0.888
B	0.906	0.941
C	0.986	0.989

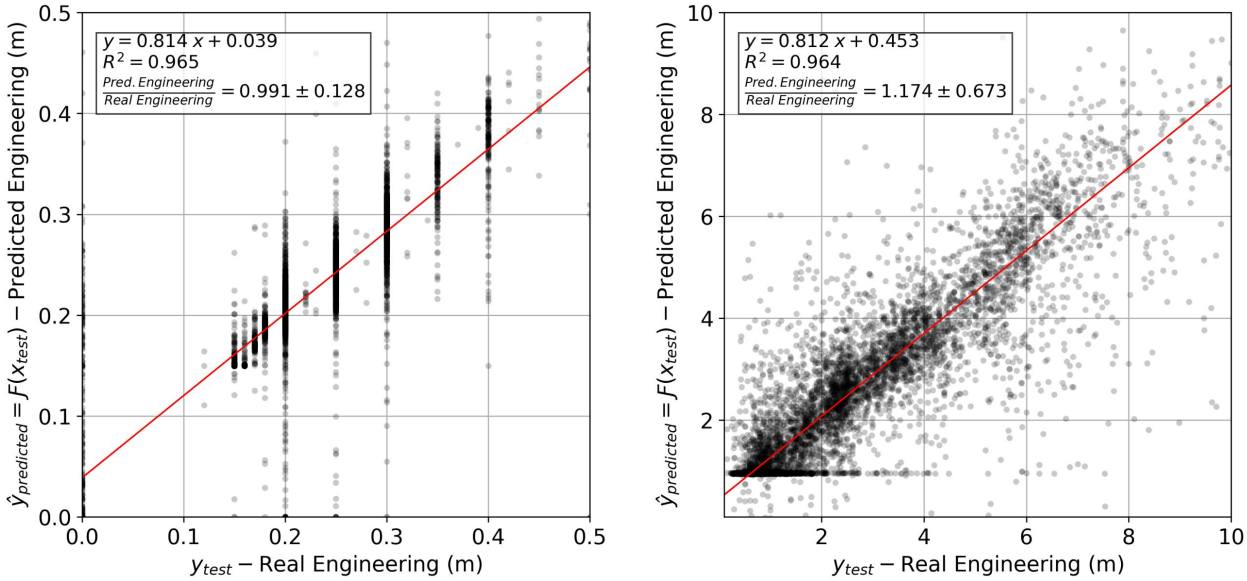
Figures 4.6, 4.7, and 4.8 represent the correlation of the engineering values predicted by the model $\hat{y}_{predicted} = \mathcal{F}(x_{test})$ with the real engineering values y_{test}, where x_{test} corresponds to the value of the architectural features and $\mathcal{F}(x)$ to the model. A better fit of the data is evidenced compared to the case without the model application, as shown in Figure 4.3; on the other hand, incorporating more data in the training partition, such as dataset C, allows a better thickness regression in the range in which the wall disappears (0 m–0.1 m).

Not only the R^2-value is used to measure accuracy, but also the mean and standard deviation of the ratio between the predicted and real values, shown in Figures 15–17 as "$\frac{\text{Pred. Engineering}}{\text{Real Engineering}}$," the standard deviation of the ratio presents a significant reduction with data-augmentation, especially for wall thickness (0.223 to 0.089, for datasets A and C, respectively), exhibiting an improvement of the model with the increased trained data. For display reasons, in Figures 4.6–4.8, the thickness limit is set to a range between 0 cm–50 cm, and the length between 0 m–10 m, regions that concentrate 99% and 98% of the data, respectively.



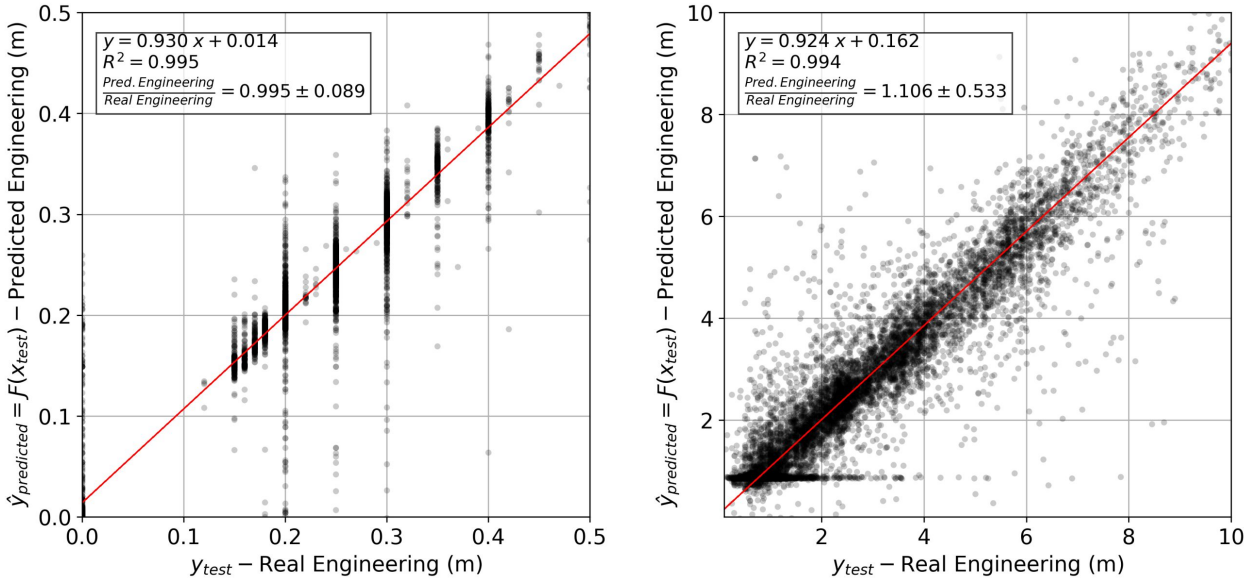
(a) Thickness **(b)** Length

Figure 4.6: Regression model correlation in test partition of dataset A.



(a) Thickness **(b)** Length

Figure 4.7: Regression model correlation in test partition of dataset B.



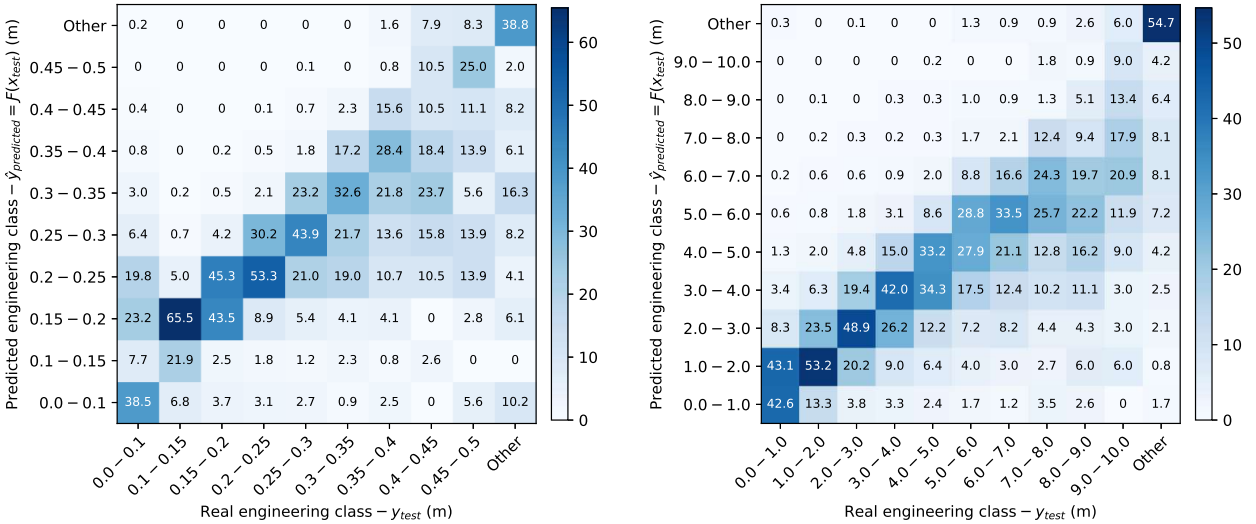
(a) Thickness **(b)** Length

Figure 4.8: Regression model correlation in test partition of dataset C.

If an output discretization for the length and thickness is made, it is possible to construct a confusion matrix that contrasts both predicted and real classes; understanding the class as the range of which each value belongs. Figures 4.9, 4.10, and 4.11 detail the confusion matrices for the three datasets model results; each value indicates the percentage of predicted accuracy, and, for each column, the sum is equal to 100% of the data in the respective real class. A diagonal trend of the data is evident in datasets B and C associated with the high R^2-value; by contrast, in dataset A, there is a marked dispersion for thickness values greater than 25 cm, a region that contains 19% of the data, and an incorrect classification of zero

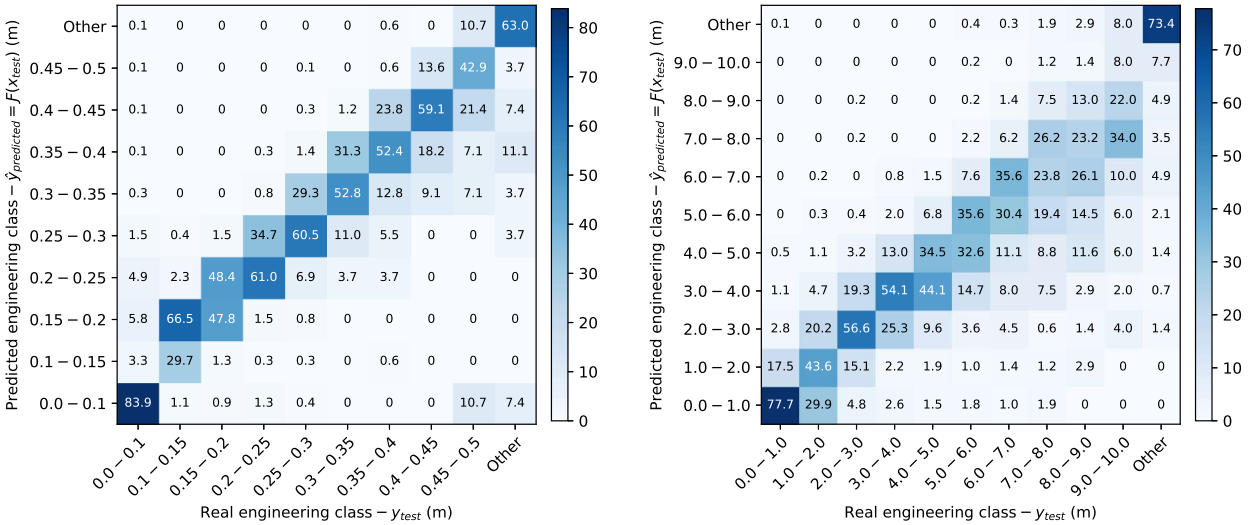
thickness, a phenomenon present when a wall disappears in the engineering plans.

The results indicate that the regression model and the selected features provide an accurate tool for predicting the wall length and thickness based on the initial architectural plans, especially when data augmentation is considered.



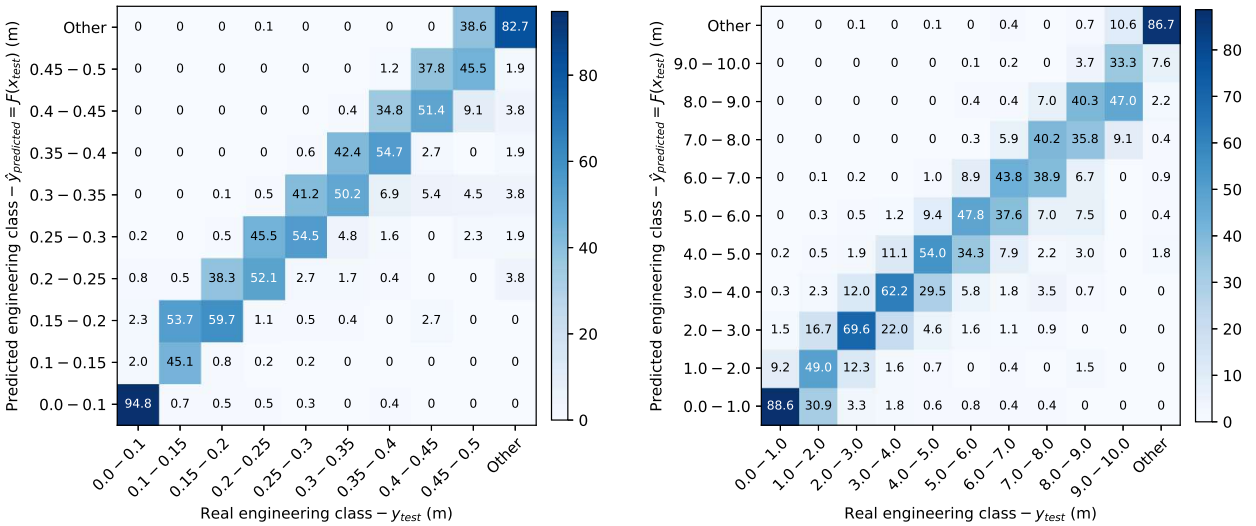
(a) Thickness **(b)** Length

Figure 4.9: Confusion matrix for dataset A test partition.



(a) Thickness **(b)** Length

Figure 4.10: Confusion matrix for dataset B test partition.



(a) Thickness **(b)** Length

Figure 4.11: Confusion matrix for dataset C test partition.

Chapter 5

CNN-based framework

Given that convolutional neural networks allow the calculation of new low and high-level features, they are used to predict new parameters and build a framework that leads to the final engineering plan. To achieve this aim, the rectangle' images of the wall contour feed the convolutional layers, contributing with geometric and topological context not considered by the manual feature calculation; these images have both the requirements and the experience of the engineering design, collected from thousands of different configurations, characterizing the Chilean building.

This chapter explores different applications of CNNs for predicting the displacement parameters and change in plan geometry per rectangle, the generation of engineering images given the architectural input, and its assembly to obtain the final engineering floor plan.

5.1. Proposed framework based on CNN models

As the regressive Sequential ANN model (Chapter 4) does not predict new structural elements or wall rectangle's engineering properties other than the thickness and length, a framework based on convolutional neural network (CNN) models is proposed. This CNN framework generates the final engineering floor plan by combining two independent plan predictions, both of which consider the architectural data as input, such as the numerical 30-feature input vector (RENE, described in section 3.1) and the 64x64 px wall rectangle images (described in section 2.3). In this contribution, the generated plan contains all learned structural considerations made by the engineering office, such as the seismic code, shear wall ratios, layout, and geometry. Architectural decisions are also incorporated in the prediction because the models are trained using the latest engineering floor plan, which each architecture company has previously validated. Thus, the procedure captures the DNA of Chilean shear wall residential building design; but it can be applied to other structural systems or realities based on a different project database (numerical features and images).

Figure 5.1 illustrates a general scheme of the proposed framework; two regressive convolutional network models (CNN-A and CNN-B) are used in parallel to predict each rectangle's design parameters in the engineering floor plan from the architectural input (Figure 5.1a), such as the 30-feature vector (RENE) and the 64x64 px images. The CNN-A regressive model (Figure 5.1b) predicts each rectangle's thickness and length. The CNN-B regressive model

(Figure 5.1c) predicts the floor's bounding box geometry change (width and aspect ratio) and the translation of the rectangles between the architectural and engineering floor plans on both main axes. Then, CNN-A and CNN-B outputs are used to assemble the first plan prediction only considering the regressed rectangle's properties, called the Regressive Plan (A_R), procedure shown in Figure 5.1d as a red rhombus. Note that these models predict the rectangle's engineering properties later used to generate the plan, i.e., the regressive CNNs do not predict images; also, the architectural wall's rectangles topology is not explicitly considered, i.e., the relationship between the rectangles is not retained. On the other hand, the second independent plan prediction is performed at image level using a modified version of the U-Net segmentation model [49]. The modified version proposed is denoted as UNET-XY; this model generates the most likely image of the engineering floor plan from the architectural input (30-feature vector and 64x64 px images), illustrated in Figure 5.1e in grayscale, where less probable walls have a light gray color. All generated images are concatenated in the proper position to assemble the second plan prediction, which in this case is called the Likely Image Plan (A_I), shown as a blue rhombus in Figure 5.1f. Ultimately, the Regressive Plan (A_R) and Likely Image Plan (A_I) are combined to lead the Final Engineering Plan (A_P), shown as a $\oplus$ symbol in Figure 5.1g, resulting in the framework output (Figure 5.1h). Next, the independent plan predictions, assemble, and their combination is described.

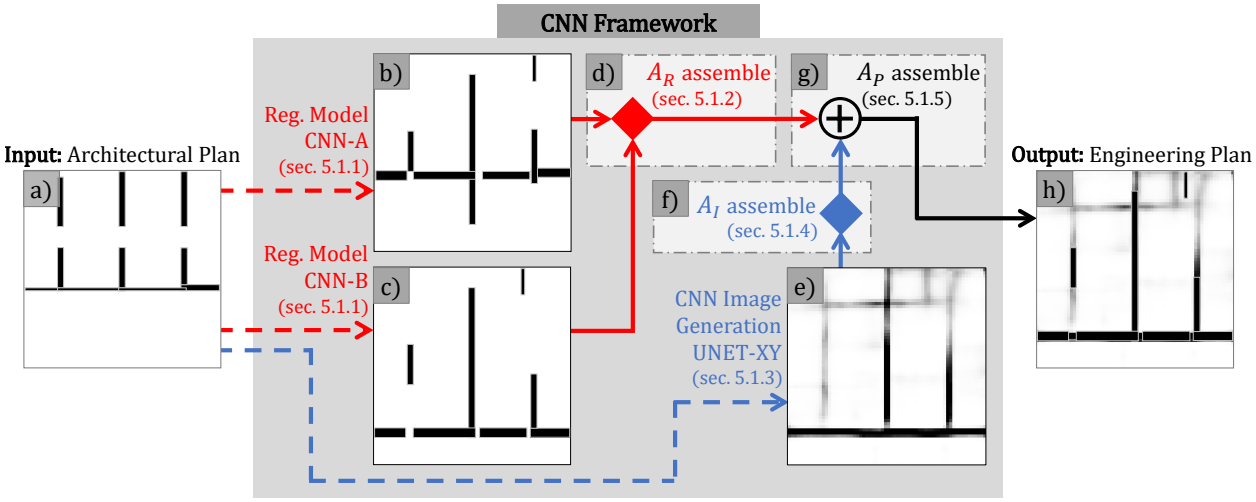


Figure 5.1: Proposed framework based on CNN models to generate the final engineering floor plan by combining two independent floor plan predictions – **(a)** Architectural input data, **(b)** CNN-A regressive model, **(c)** CNN-B regressive model, **(d)** First plan prediction (A_R) assembled using CNN-A and CNN-B models, **(e)** CNN image generation UNET-XY, **(f)** Second plan prediction (A_I) assembled using UNET-XY output images, **(g)** Final engineering floor plan assemble by combining the independently predicted plans, and finally **(h)** The framework output.

5.1.1. Regressive CNN models for engineering parameters prediction (CNN-A and CNN-B)

The CNN-A and CNN-B models are defined to predict different engineering parameters for each rectangle. The former seeks to predict the thickness (*RectThickness*) and length (*RectLength*); by contrast, the latter predicts the translation of each rectangle concerning

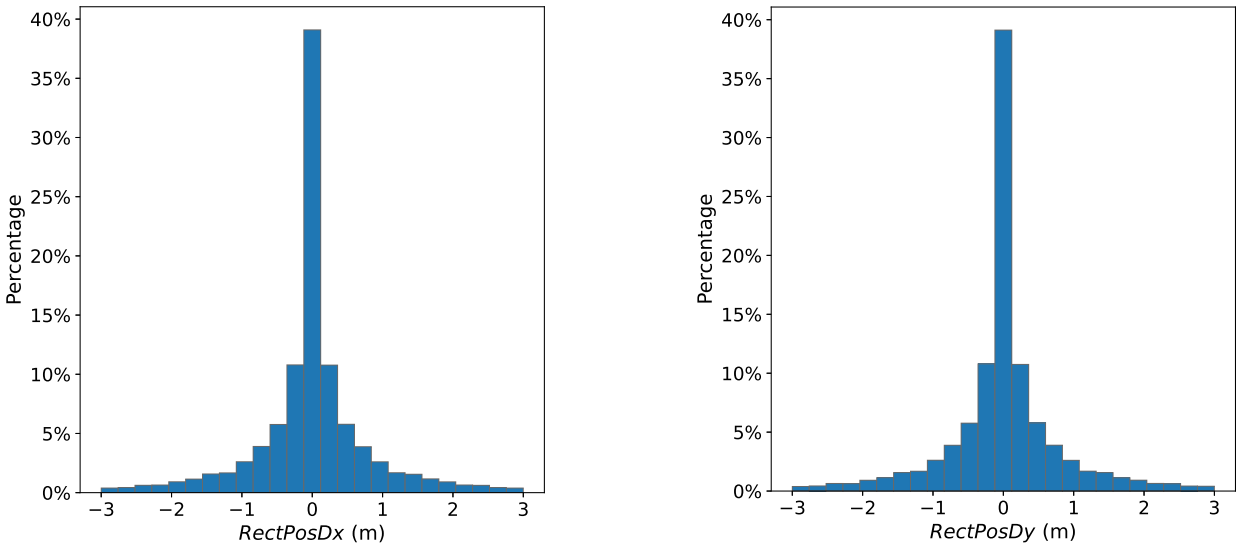
the architectural position (*RectPosDx*, *RectPosDy*), the floor bounding box width (*FloorWidth*), the floor aspect ratio (*FloorAspectRatio*), and the floor offset between the geometric center (GC) and center of mass (MC) on both main axes (*FloorMassCenterDistance-x*, *FloorMassCenterDistance-y*). Some of these features can be observed in Figure 3.1.

Unlike the Sequential model in Chapter 4, CNN-A and CNN-B models present a better performance in terms of the coefficient of determination (R^2) given the incorporation of new features extracted from the images by the convolutional layers. The calculation of the rectangle translation on both axes was carried out considering the relative difference between the position in the architecture and engineering plans concerning the floor mass center:

$$RectPosDx = rMCDx_E \cdot w_E - rMCDx_A \cdot w_A \tag{5.1}$$

$$RectPosDy = rMCDy_E \cdot h_E - rMCDy_A \cdot h_A \tag{5.2}$$

Where $rMCDx$ and $rMCDy$ correspond to the *RectFloorMassCenterDistance* feature (distance between the floor and the rectangle mass centers) on the x and y-axis; $w_A = \frac{FloorWidth_A}{2}$ and $w_E = \frac{FloorWidth_E}{2}$ correspond to the widths of the floor bounding box in architecture (A) and engineering (E) plans, finally $h_A = \frac{w_A}{FloorAspectRatio_A}$ and $w_E = \frac{FloorWidth_E}{2}$ correspond to the bounding box's height in architecture (A) and engineering (E) plans, respectively. It is worth mentioning that the translation was limited to a maximum of 3 m to achieve lower dispersion than the heuristic association mechanism outliers between architecture and engineering wall's rectangles; this limit yields an improved response of the regressive models as the translation range decreases, making smaller values more relevant. Translation values over that limit, typically due to association between rectangles with significant differences in size or position, were set to zero. The translation features ($RectPosDx, RectPosDy$) are described in the histograms of Figure 5.2, wherein the proposed region $(-3, 3)$ m concentrates 85.8% of the data.



(a) Translation on x-axis

(b) Translation on y-axis

Figure 5.2: Translation of the rectangle's center of mass between architecture and engineering plan without normalization.

CNN-B model also considers the rectangle thickness and length as output since such incorporation improves the performance in terms of the R^2-value; however, these output variables are discarded at prediction time as the CNN-A model output is used instead. In both regressive CNN models, the proposed architecture is the same, composed of two different inputs: the 30-feature vector (RENE) normalized between 0 and 1 (Table 3.1) and the 64x64 px image associated with each rectangle (section 2.3). The input image is processed by four convolutional blocks of 64, 32, 32, and 8 filters, with a 3x3 kernel size and 1x1 stride size. A Batch Normalization (BN) layer [11] is applied at the output of each block; subsequently, a dimension reduction is applied with an Average Pooling layer with a pool of size 2x2. The convolutional blocks end in a 1x32 tensor (Flatten), which is concatenated with the 30-feature input vector (RENE) to feed a fully connected network (FC) with six layers with 1024 neurons each, normalized after the first and second layers by a BN layer, to predict two parameters in model CNN-A and eight parameters in model CNN-B; outputs described in Table 5.1. Python Keras library [44] with TensorFlow-GPU as a backend was used for the implementation. The activation function corresponds in all cases to ReLU [45], except in the output layer, which does not consider a non-linear processing unit. The loss function corresponds to the mean square error, $MSE(y_i, \hat{y}_i) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$, and the optimizer corresponds to Adam with a learning rate α of 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-7}$, same values recommended by [44, 46]. Glorot Uniform [47] was used as the weight initializer for all layers, and no regularization was applied in the layer's kernel. A weight factor equal to 1.0 was used for each output in both models to calculate the losses. Figure 5.3 illustrates a schematic of the implemented CNN-A model; the CNN-B model architecture is identical except for the number of output parameters, which is eight in this case.

Table 5.1: Details of the CNN regressive models.

Model	Output variables	N° parameters
CNN-A	RectThickness, RectLength	5,354,086
CNN-B	RectThickness, RectLength, RectPosDx, RectPosDy, FloorWidth, FloorAspectRatio, FloorMassCenterDistance-x, FloorMassCenterDistance-y	5,358,186

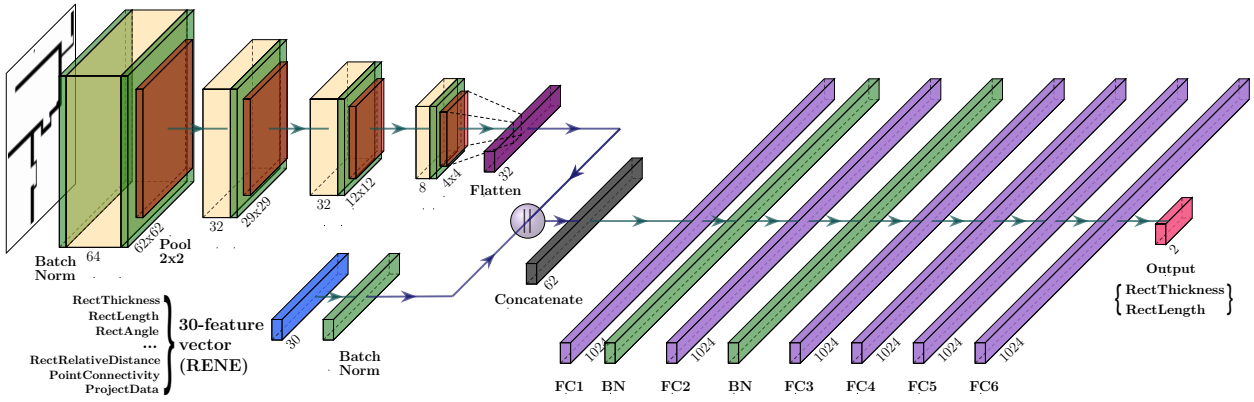


Figure 5.3: CNN-A regressive model architecture.

Because the model performance is affected by the number of neurons and the layer type used, several combinations were tried in a heuristic fashion to find the best results in terms of R^2. Like the Sequential algorithm, the CNN model cost, both in computation time and the number of parameters, grow exponentially according to the number of convolutional layers, the number of neurons in the FC network, and the kernel size. Due to the large volume of the datasets, it was not possible to use an optimization algorithm to determine the hyper-parameters that maximize the performance [48]. In both models (CNN-A, CNN-B), the training was carried out with mini-batches of size 128 in a maximum of 150 epochs; after the end of each epoch, the data was randomly reordered to reduce the variance. For avoiding overfitting, an early stopping regularization with a tolerance of 40 epochs was used. Also, to accelerate the convergence, the Keras monitor ReduceLROnPlateau was considered, which reduces the optimizer learning rate if the loss metric does not improve by a margin of 15 epochs, with a reduction factor of 10^{-1} and a minimum delta of 10^{-4} for the evaluated value. The partition of the dataset (C, described in section 3.2) in training and testing was 70% and 30% in random order, using the same for CNN-A and CNN-B models; from the training dataset, 20% was used for validation, and the remaining 80% for training.

5.1.2. Assemble of the engineering regressive plan (A_R)

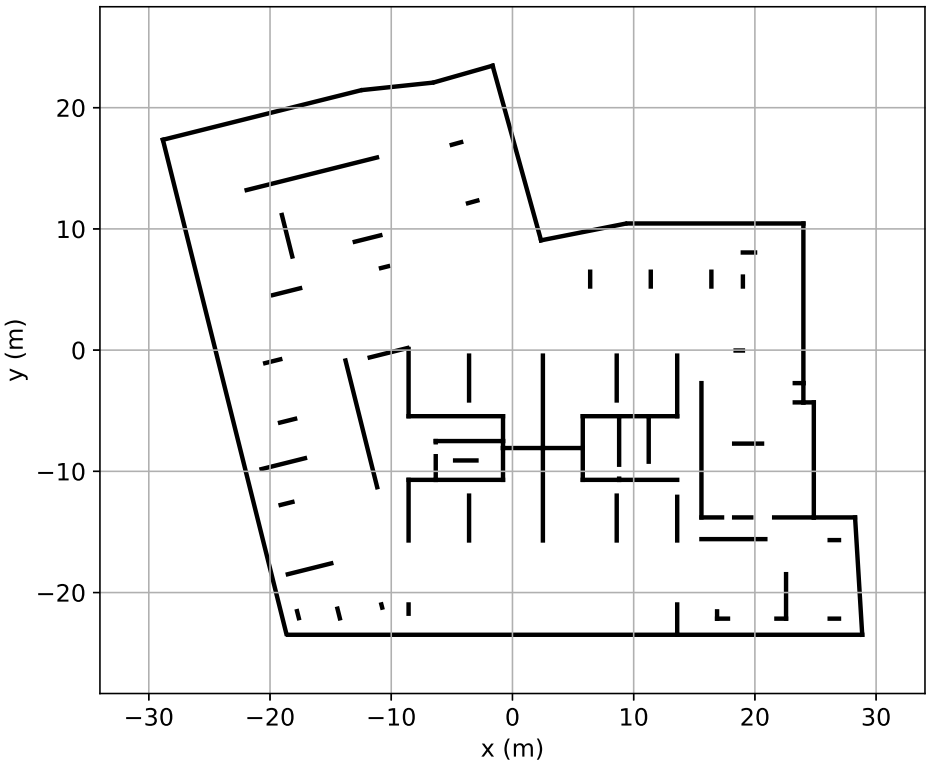
The engineering Regressive Plan (A_R, represented as a red rhombus in Figure 5.1d) is assembled by employing the previous regressive CNN models. Each rectangle's engineering size (thickness and length) is obtained from the CNN-A output, while the engineering rectangle's position (p_x, p_y) is calculated by adding the outputs from the CNN-B model to the architectural position, such that:

$$p_x = \Big(rMCDx + fMCDx\Big) \cdot w + RectPosDx \tag{5.3}$$

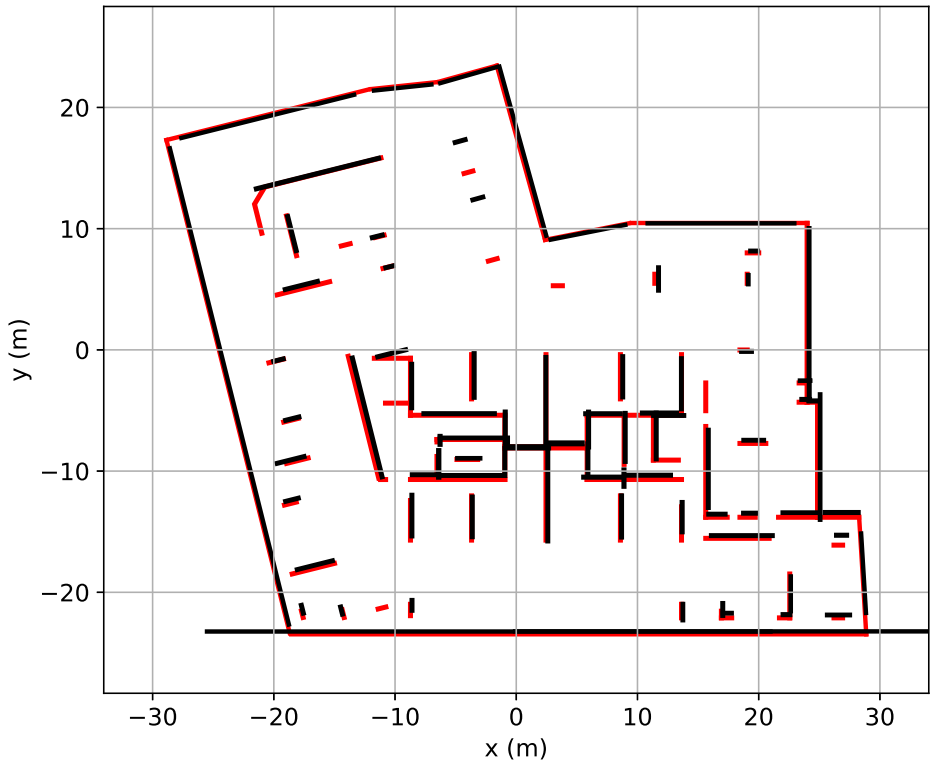
$$p_y = \Big(rMCDy + fMCDy\Big) \cdot h + RectPosDy \tag{5.4}$$

Where $rMCDx$ and $rMCDy$ correspond to the architectural input feature *RectFloorMassCenterDistance* (distance between the floor and rectangle mass centers) on x and y-axis. $fMCDx$ and $fMCDy$ correspond to the CNN-B engineering output feature *FloorMassCenterDistance* (distance between the geometric and mass center of each floor) on the x and y-axis. *RectPosDx* and *RectPosDy* correspond to the relative translation between the architectural and engineering plans predicted by CNN-B. Finally, $w = \frac{FloorWidth}{2}$ and $h = \frac{w}{FloorAspectRatio}$ correspond to the width and height of the floor bounding box, respectively, being $FloorWidth$ and $FloorAspectRatio$ CNN-B outputs.

Figure 5.4 illustrates an example of the engineering Regressive Plan (A_R) assemble using CNN-A and CNN-B models, comparing the architectural input plan (Figure 5.4a) and the predicted plan (Figure 5.4b), considering changes in the thickness, length, and position of the elements. The black rectangles in Figure 5.4b stand for the predicted ones (A_R) while the red rectangles illustrate the real solution; in this case, there are minor differences regarding translation, length, and connectivity, but as discussed in the Results section, if the position or length is not changed, the error between the proposed and real engineering solution increases.



(a) Architectural input plan



(b) Prediction of the A_R plan (black color), compared to the real A_Y plan (red color) in engineering

Figure 5.4: Comparison between an architectural plan, the engineering Regressive Plan (A_R) composed of the rectangles predicted by the CNN-A and CNN-B models, and the real engineering plan (A_Y).

As CNN-A and CNN-B are pure regressive models, they do not allow the proposition of new rectangles because only existing architectural elements are processed to compute the changes to be made (geometry, position) for creating the Regressive Plan (A_R). Thus, the convolutional neural network model UNET-XY, described in the following subsection, is proposed to generate the most likely engineering image floor crop for each rectangle, fed by the 30-feature vector data (RENE) and the architectural image crop (section 2.3), allowing the proposal of new objects not present in the input architectural data, making suggestions about the element's connection, and reinforcing the regressed rectangles in A_R plan.

5.1.3. UNET-XY model for the engineering floor plan prediction

In this section, another prediction of the engineering floor plan is presented but employing a model based on the U-Net fully convolutional image segmentation network [49] to generate the most likely image of the engineering floor plan given the architectural input. The standard image segmentation U-Net model is employed to accomplish predictions for the 4096 pixels of the 64x64 px engineering output image by performing a binary classification task for each pixel; class-0 corresponds to the background, class-1 to the wall rectangles. The classification aims to guess each pixel's proper class in the engineering image from the architectural input, based on the data the model has been trained. The U-Net model's architecture consists of two main sections: encoder (contraction) and decoder (expansion); the encoder section is used to capture the context in the image comprised by several convolutional and max-pooling layers; on the other hand, the decoder section is comprised by many feature channels used to enable precise localization through the transposed convolutions, propagating context information to higher resolution layers. The decoder also combines the feature and spatial information through a sequence of up-convolutions and concatenations with high-resolution features from the encoder, improving localization and reconstructing the output image. Therefore, the expansive path is symmetric to the contracting part and yields an u-shaped architecture [49].

Additional to the standard U-Net model, named UNET in the implementation, two variations were explored. The first corresponds to the prediction of the numerical 30-feature vector (RENE) plus the structure's fundamental period on both axes obtained from the engineering office's structural main results, adding up to 32 output labels to improve the model back-propagation training performance phase. These outputs are calculated by a connected Fully Convolutional (FC) network from the last convolutional layer of the UNET subnet. This model is called UNET-Y and presents a better performance in terms of the average binary accuracy per pixel (frequency in which the predictions coincide with the binary labels) compared with UNET model. The second variation studied is built on UNET-Y, and consists of combining the architectural image with the 30-feature input vector (RENE) in a ratio of 3:1. Thus, an intermediate image is generated to be used later as input in UNET-Y. This model is called UNET-XY, presenting a better result in terms of binary accuracy than the standard UNET and UNET-Y, as shown in the results section. The mentioned models and the details of each configuration are illustrated in Figure 5.5.

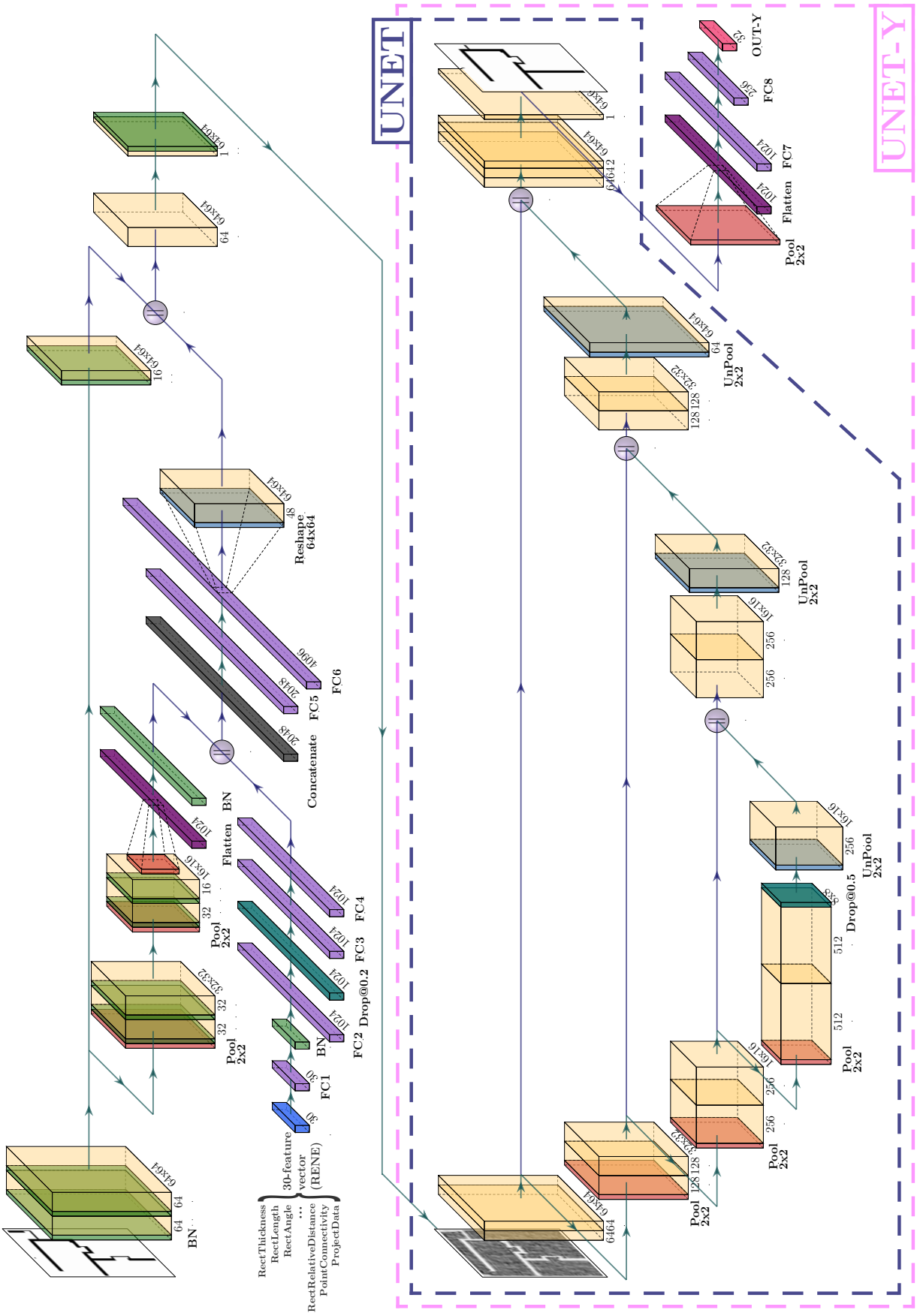


Figure 5.5: UNET-XY model architecture.

UNET-XY is fed by the 30-feature vector (RENE) and the 64x64 px image from each architectural rectangle. The image is processed by three convolutional blocks of 64x64, 32x32, and 32x16 kernel size, normalized by BN layers, ending in a dimension reduction leading to a 1x1024 size tensor (Flatten). This tensor is concatenated with a 4-layer FC subnet fed with the numerical input vector, regularized by a BN layer and a dropout layer with a ratio of 0.2 [43]. This concatenation is subsequently connected to two FC layers of 2048 and 4096 neurons, followed by a convolutional layer of 48 filters. This 48-filter layer is later added to a 16-filter convolutional layer from the first 64x64 convolutional block to generate the intermediate image, where the ratio 48:16 or 3:1 is evidenced. The intermediate image is then processed by the 4-level UNET-Y subnet, with sizes of 64, 128, 256, and 512 filters with a dropout regularization (ratio of 0.5) in the deepest contractive level. All convolutional layers used have a kernel size of 3x3 and a stride of size 1x1. The ReLU activator was used in all hidden layer outputs, except for the image output where a sigmoidal activation function is used instead; OUT-Y output (32 engineering labels) does not consider a non-linear processing unit. The pooling layers correspond to Average Pooling with a size of 2x2. The loss function for the image corresponds to binary cross-entropy $CE(y_i, \hat{y}_i) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log \hat{y}_i + (1-y_i) \cdot \log(1-\hat{y}_i)$, and MSE for vector Y, with a weight factor of 5 to 1, respectively. The optimizer corresponds to Adam with a learning rate α of 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-7}$, same values recommended by [44, 46]. The weight initializer in all convolutional layers is He Normal [50], while Uniform Glorot is employed in FC layers. In the U-Net layers, L2 was used as a kernel regularizer with a factor of 10^{-4}. The training was carried out with mini-batches of size 100 in a maximum of 15 epochs. Keras metric ReduceLROnPlateau was used to accelerate convergence, with two-epoch patience and a reduction factor of 10^{-1}. The training and validation partition is the same used in the CNN models. UNET-XY comprises a total of 23,833,256 parameters, where 2688 of them are not trainable.

For the sake of comparison and to explore the application of a model architecture based on conditional generative adversarial networks (cGANs), the Pix2Pix model [51] is also implemented, illustrated in Figure 5.6. Pix2Pix model allows the generation of the engineering plan conditioned to the architectural input (30-feature vector and 64x64 px images) through the U-Net generator network and a discriminator network composed of 5 convolutional blocks. These blocks consist of 32, 64, 128, 256, and 256 filters processed by regularizing BN layers, using the *LeakyReLU* activation with an α value of 0.2. An *adversarial loss* is used for the discriminator and composite loss for the combined model in a 100:1 ratio for the generator and discriminator, respectively. The optimizer corresponds to Adam with a learning rate α of $2 \cdot 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-7}$, same values recommended by [44, 51]. The training was carried out with a maximum of 100 epochs and a batch size of 1. However, the implemented Pix2Pix model results are, on average, 80% lower relative to UNET-XY in terms of the binary accuracy metric during the entire training and validation history, which is later detailed in the results section.

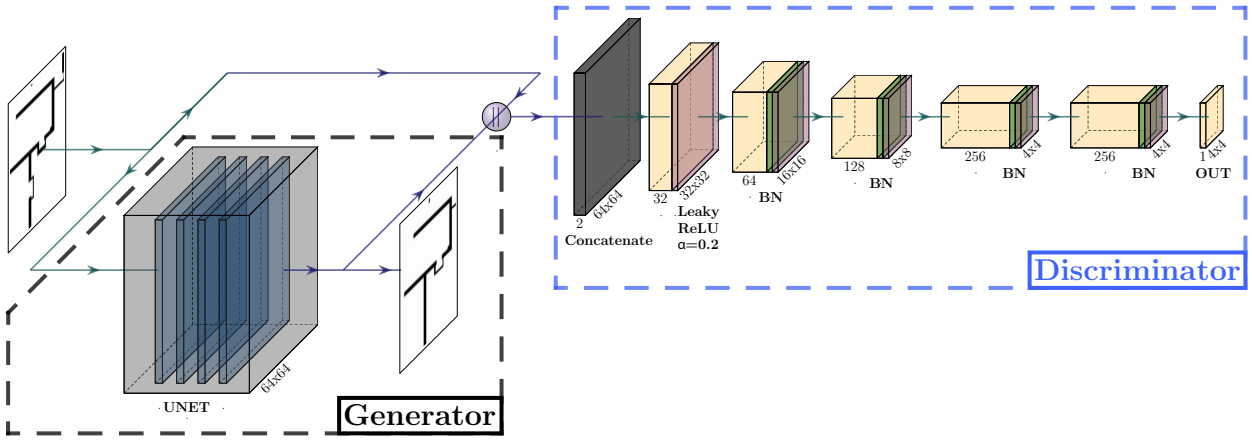


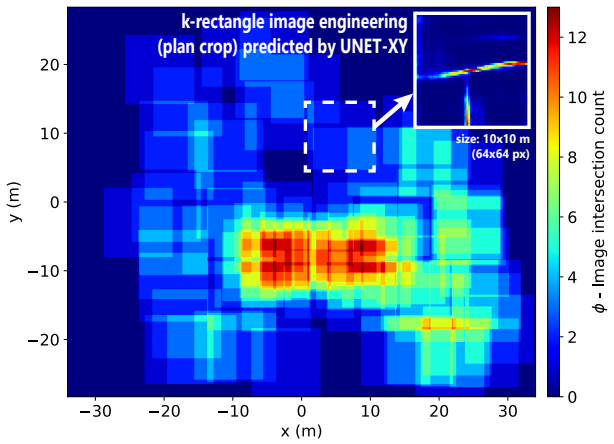
Figure 5.6: Pix2Pix image generation model architecture.

5.1.4. Assemble of the engineering likely image plan (A_I)

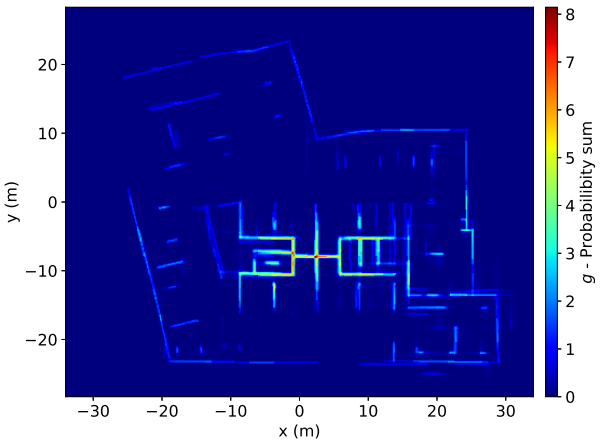
In this section, the UNET-XY model output images are used to assemble the engineering Likely Image Plan (A_I, represented as a blue rhombus in Figure 5.1f). For this purpose, each image predicted by the UNET-XY model is placed in the appropriate position on the engineering plan (p_x, p_y) employing the Eqs. 5.3 and 5.4. Note that the predicted images could be overlapped in this case, requiring the set of a rule to estimate the final pixel value; this rule could be, for example, the pixel average across the overlapped images or more complex metrics described in the following paragraphs. In this study, the engineering plan is considered as a matrix with the same 0.156 m/px factor used to create the images; that is, a 10x10 m region is represented in 64x64 px. The goal then is to identify the matrix that contains the most representative pixel values employing the images generated by the UNET-XY model. The incorporation of the Likely Image Plan (A_I) in the assembling of the Final Engineering Plan (A_P) adds much more information about the wall layout, allowing to predict the presence of new rectangles, reinforcing the existing ones proposed by the regressive models, and offering notions regarding the connection between them.

The assemble process of the A_I matrix consisted of four stages, represented in Figure 5.7. First, the matrix ϕ was created, which identifies the amount of overlapped engineering images (floor plan crops described in section 2.3) predicted by UNET-XY on each pixel of the floor plan, considering that the image of each k-rectangle (denoted as I_k) is located at the mass center $({p_x}^k, {p_y}^k)$ of the respective rectangle (Figure 5.7a). The second stage corresponds to the sum of the binary class value for each pixel of all N images calculated as $g = \sum_{k=1}^{N} \sum_{i=1}^{64} \sum_{j=1}^{64} I_k[i,j]_{[a,b]}$, wherein $I_k[i,j]$ corresponds to the pixel $[i,j]$ of the k-image, added at the position $[a = {p_x}^k - i + 32, b = {p_y}^k - j + 32]$ of the matrix g (Figure 5.7b). Subsequently, in the third stage, the average $\hat{g}[i,j] = \frac{g[i,j]}{\max(\phi[i,j],1)}$ is calculated as the division between the sum g and the number of images ϕ present in the pixel $I_k[i,j]$ (Figure 5.7c). Finally, in the last stage, the matrix $A_I[i,j] = \frac{f(\hat{g}[i,j])}{\max A_I}$ is obtained as the application of a non-linear output function $f(x)$ on each pixel of the $\hat{g}$ matrix. Here, the A_I matrix is normalized between 0 and 1 (Figure 5.7d) when it is divided by its maximum value. It should be noted that the matrices ϕ, g, $\hat{g}$, and A_I have the same border limits as those defined by the floor

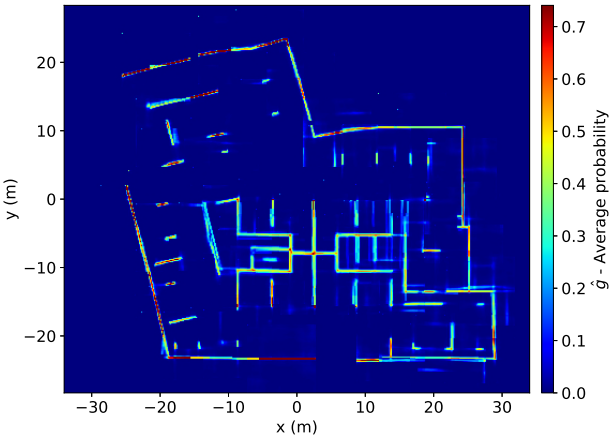
plan predicted by the CNN regressive models (Figure 5.4).



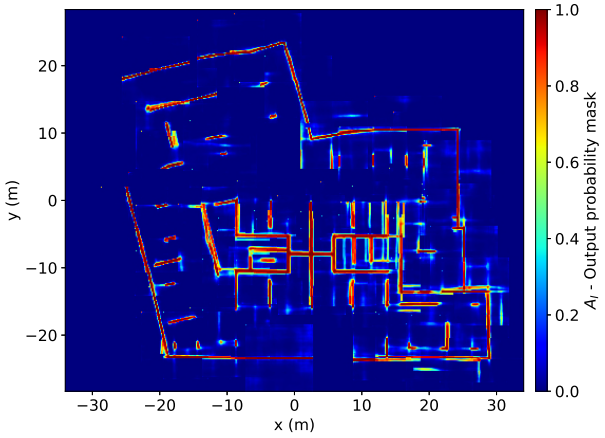
(a) Number of the intersection of the images in the plan, matrix ϕ



(b) Sum of the class values of all images in the plan, matrix g



(c) Average value, matrix $\hat{g}$



(d) Application of non-linear output function and normalization for matrix A_I

Figure 5.7: Assemble procedure of the Likely Image Plan (A_I).

The output function f allows controlling the contrast of the matrix $\hat{g}$ (Figure 5.7d), giving more or less importance to regions of low or high-class likeness given by the images. In particular, nine different functions were tested: $f_{\text{linear}}(x) = x$, $f_{\text{sigmoid}}(x) = \frac{1}{1+e^{-10\cdot(x-0.5)}}$, $f_{\text{sigmoidh}}(x) = \frac{2}{1+e^{-10\cdot x}} - 1$, $f_{\text{tanh}}(x) = \frac{\tanh(10\cdot(x-0.5))+1}{2}$, $f_{\cos}(x) = \frac{\cos((x+1)\cdot\pi)}{2}$, $f_{\text{square}}(x) = x^2$, $f_{\text{quad}}(x) = x^4$, $f_{\text{isquare}}(x) = -(1-x)^2+1$, and $f_{\text{iquad}}(x) = -(1-x)^4+1$. Figure 5.8 illustrate the plots of these functions between the class range 0–1. The $f_{sigmoid}$, f_{tanh}, f_{square}, and f_{quad} functions are conservative since they strongly penalize the region of $\hat{g}$ that presents values lower than 0.5. On the other hand, $f_{sigmoidh}$, $f_{isquare}$, and f_{iquad} allow obtaining more information in the same regions ($\hat{g} < 0.5$) since the penalization is weaker than the previous functions. As discussed in the results section, the best CNN framework results are achieved using the $f_{sigmoidh}$ function.

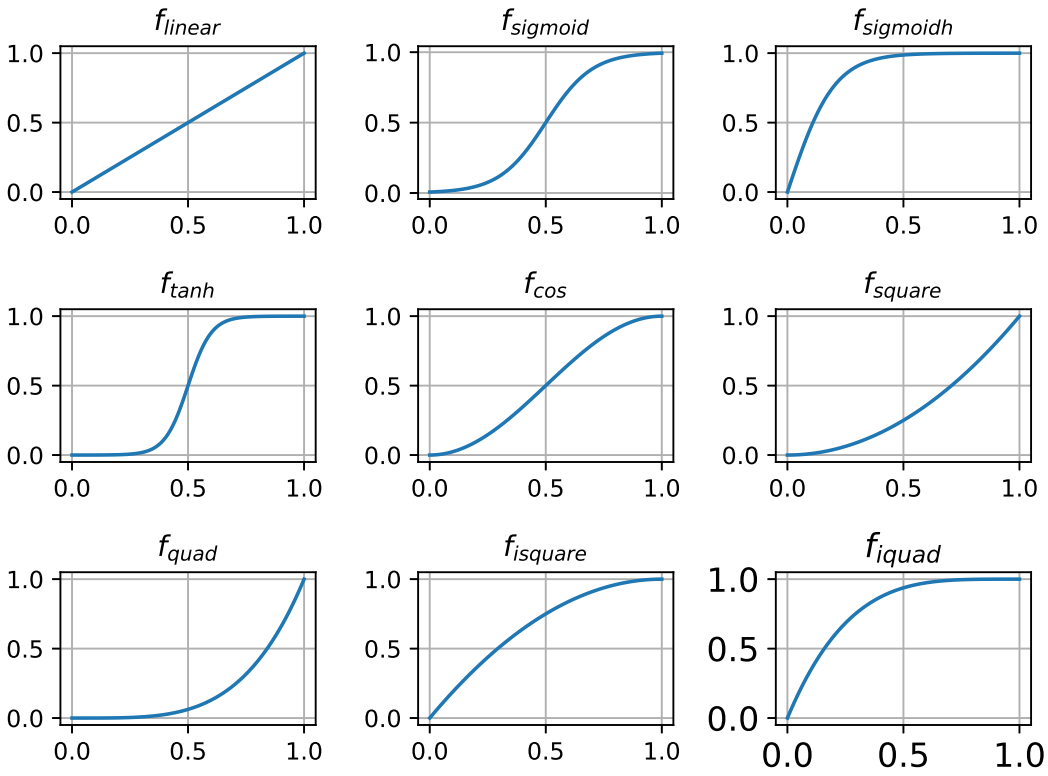
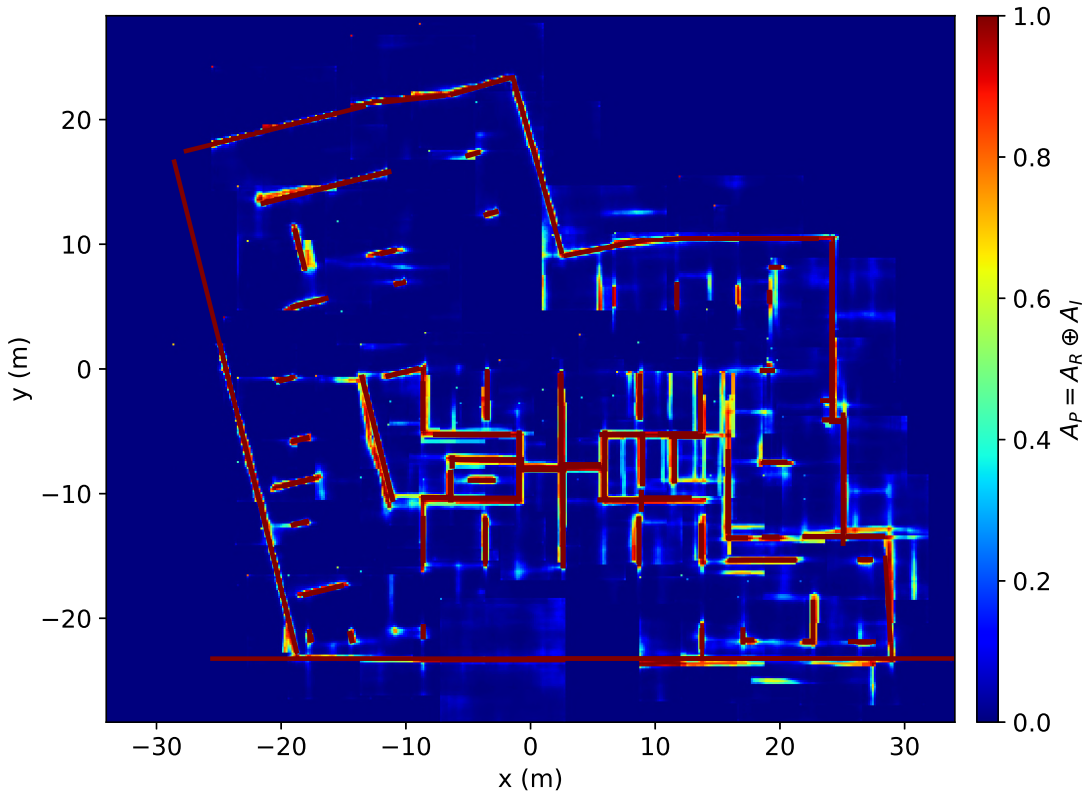


Figure 5.8: Output $f(x)$ functions plot used for the normalization of the $\hat{g}$ matrix.

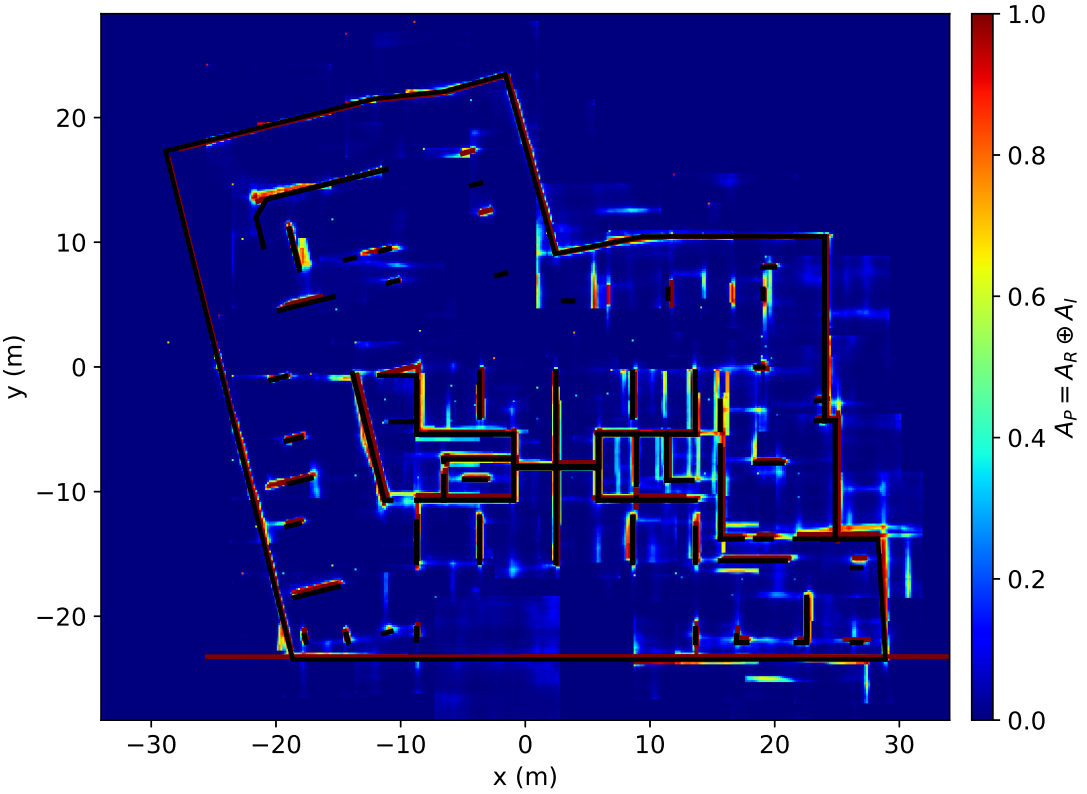
5.1.5. Combining the regressive plan (A_R) and the likely image plan (A_I) to assemble the final engineering plan (A_P)

Finally, the matrix representing the Regressive Plan (A_R) and the matrix representing the Likely Image Plan (A_I) are combined, leading to the Final Engineering Plan (A_P), where $A_P = A_R \oplus A_I$, illustrated in Figure 5.1g. In this study, the sum operator $\oplus$ is defined as $(X \oplus Y)[i,j] = \min(X[i,j] + Y[i,j], 1)$ for each $[i,j]$ position within matrices, such that the possible presence of a wall is captured from combining (adding) A_R and A_I. Figure 5.9 illustrates an example of the final engineering plan assemble, where the combination of A_R and A_I matrices is presented in Figure 5.9a, subsequently compared with the real solution in Figure 5.9b.

Next, the subsection 5.1.6 describes a proposed framework for obtaining the wall rectangles from the Regressive Plan (A_R) and Likely Image Plan (A_I). Finally, section 5.2 describes the results of the whole CNN framework for both regressive models (CNN-A and CNN-B), the likely image generation model (UNET-XY), and the combination of the predicted floor plans A_R and A_I for assembling the Final Engineering Plan (A_P).



(a) Predicted plan A_P, as the sum of A_R and A_I



(b) Comparison of the proposed solution with the real plan (in black)

Figure 5.9: Final Engineering Plan assemble (A_P) by combining the prediction of the Regressive Plan (A_R) and the Likely Image Plan (A_I).

5.1.6. Wall rectangles proposition from the predicted engineering image plans

The developed CNN framework's final process aims to obtain the wall rectangles from the predicted engineering image plans as geometrical objects with defined length, thickness, angle, and position. The wall's rectangle information is more manageable if using a numerical data structure rather than an image domain, which are highly discretized in the sums of pixels, the position cannot be directly obtained, and the overall shape of each rectangle can be merged with another object(s), as shown in Figures 5.7 and 5.9. The rectangle data obtained from the images can be used in structural modeling, analysis, and optimization algorithms such as genetic algorithms[5] (GA) to find the best subset of walls that improve the building performance [26, 27, 53].

Figure 5.10 illustrates the proposed framework, later explained in the following subsections, for obtaining the rectangles from the image plans (Figure 5.10a), considering as input both Regressive Plan (A_R, section 5.1.2) and Likely Image Plan (A_I, section 5.1.4). First, the images are discretized in a collection of fixed-size square regions of 4x4 meters, named patches (Figure 5.10b); this discretization procedure simplifies the comparison between the images, leading to fewer errors in the geometrical shapes recognition. For each patch, the area of the structural elements were obtained in the A_R and A_I plans by the sum of each pixel multiplied by the inverse of the meter/pixel (m/px) factor (Figure 5.10c); the comparison between the patches lead to three different scenarios (Figure 5.10d). First, if $Area(A_R) > Area(A_I)$ the generative UNET-XY model output, which leads to the assembly of the A_I plan (section 5.1.5), does not include new structural elements in the same region compared to the regressive data, that is considered to be reliable; similarly, if $Area(A_R) \sim Area(A_I)$ the regressed rectangle data and the likely plans contain the same amount of structural wall's area, thus, no additional elements should be added. By contrast, if $Area(A_R) < Area(A_I)$ the UNET-XY model predicts additional wall's segments that must be accounted for; in that case, a rectangle(s) is proposed from the difference between $A_I - A_R$ (Figure 5.10e) that maximizes the intersection in the area clusters (Figure 5.10f). For such purpose, an optimization task is performed (Figure 5.10g) that considers geometrical restrictions such as the maximum thickness, length, angle, translation, and context restrictions from the probability measure that each pixel has (section 5.1.4), avoiding to propose rectangles that consider null probability area. Finally, for all patches that satisfy a positive area difference, the optimization task for finding rectangles is performed (Figure 5.10h), finishing the process.

The following subsections detail the patch discretization of a given floor, the comparison between the regressed and image areas, and the rectangle proposition over each new area by finding the maximum for an optimization problem.

[5] A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover, and selection [52].

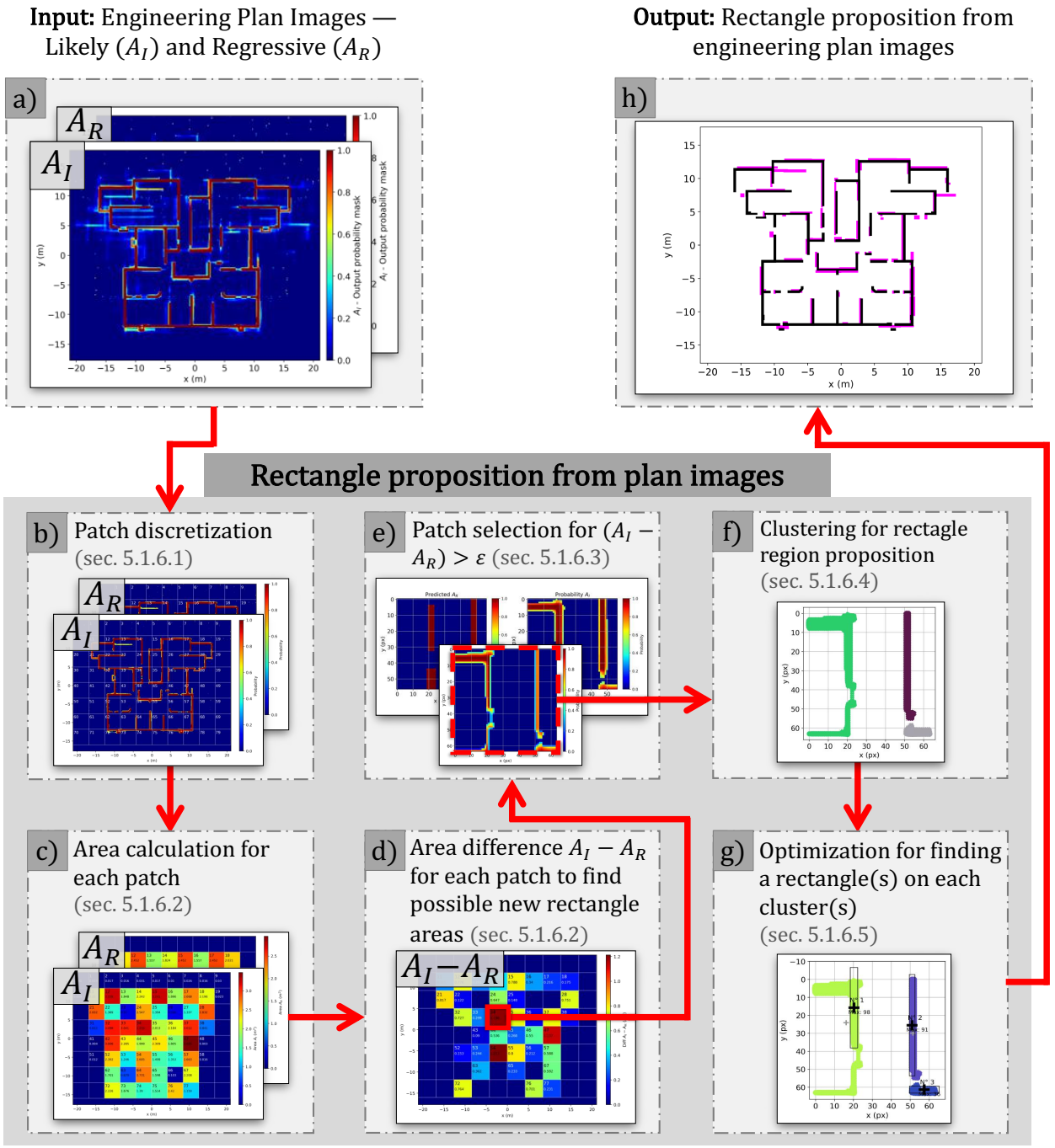


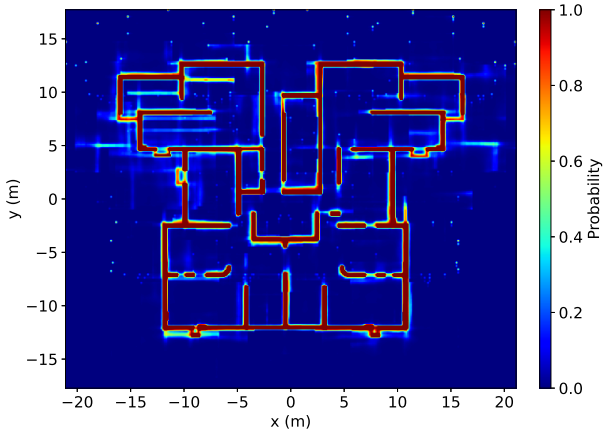
Figure 5.10: Proposed framework for obtaining rectangles from the plan images – **(a)** Image input data, **(b)** Image discretization into 4x4 m patches, **(c)** Area computation for each patch, **(d)** Area difference between patches to find potential new rectangle areas, **(e)** Patch selection from the area difference, **(f)** Clustering for rectangle region proposition, **(g)** Optimization for finding rectangles on each cluster, and finally **(h)** The output where all possible rectangles have been identified from the input images.

5.1.6.1. Image floor plan discretization (patches)

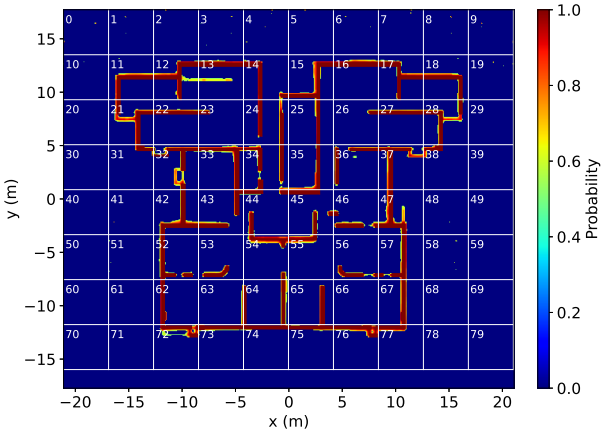
As previously discussed, the rectangles are proposed considering a subregion of the plan; this approach improves the optimization algorithm for finding rectangles, as it takes less time and resources as the search space is dramatically reduced. Similarly, a subregion with the same coordinates between image plans makes it possible to compare the structural elements between the Regressive Plan image (A_R) and the Likely Plan image (A_I).

Next, each floor plan image was discretized in a square grid of $N = 4$ m, each 4x4 m region called in this work as a patch. This patch size can vary depending on the engineer design decisions and the data it has to capture; in this case, it was obtained by a manual inspection of the dataset's wall distribution, and 4 meters is sufficient to capture the surroundings of the assembled plan while keeping a small search space for the developed algorithms. A reduced search space improves the response of the optimization algorithms and avoids the presence of complex objects as they are cut in several regions. The patches were stored in a 64x64 px image (matrix) to achieve a resolution similar or better than the one used for creating the wall rectangle's images (section 2.3); thus, the m/px factor (meter per pixel) equals 0.0625. A minimum threshold $\delta = 0.5$, obtained through a trial and error procedure, was also applied to the likely's image probability output; this limit removes noise and wall segments with a low confidence value. δ can be manually configured depending on the design principles of the engineering office; high-value thresholds are more conservative than the low ones, filtering the wall segments not predicted by the likely plan image with a high probability class.

Figure 5.11 illustrates the patch creation example; Figure 5.11a shows a likely floor plan image with no minimum threshold. On the other hand, Figure 5.11b details the same plan but considering the $\delta = 0.5$ limit and the enumerated patches, reaching a total of 80; in this case, the probability threshold removes the image noise, the rectangles outside the floor perimeter, and other invalid structural elements, mostly considered as false positives from the UNET-XY model's output.



(a) Floor likely engineering image plan (A_I) without any probability thresholds

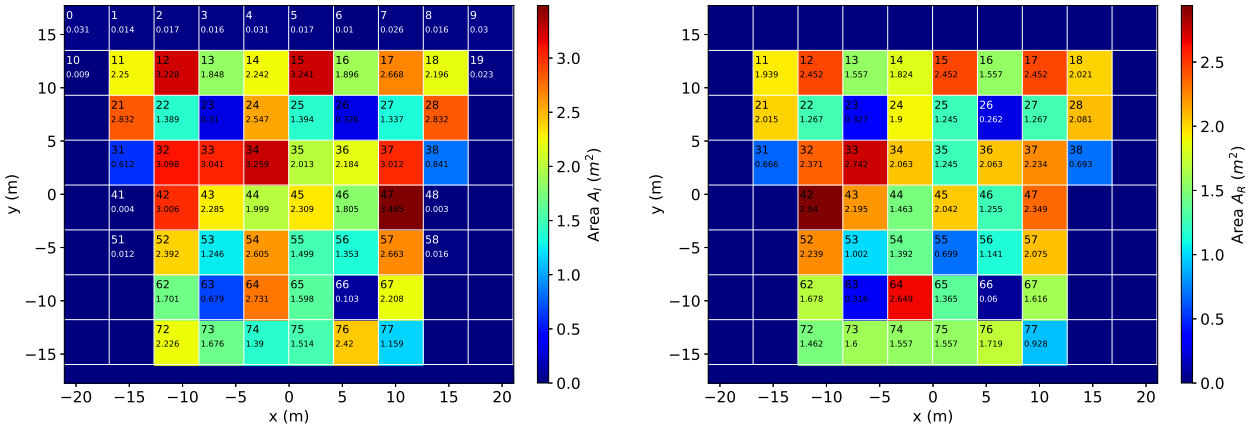


(b) Patches of the region, enumerated from 0 (top-left corner) to 79 (bottom-right), considering a $\delta = 0.5$ probability threshold limit

Figure 5.11: Example of patch discretization from a basement floor type.

5.1.6.2. Area computation for each patch

From each patch, the structural element's area of the Regressive Plan (A_R) and the Likely Image Plan (A_I) is computed, obtained by the sum of all pixel values, later multiplied by the inverse of the pixel per meter factor of the images, in this case being 0.0625 m/px. The area comparison allows for decision-taking about searching for new rectangles in the likely image plan; only if $Area(A_I) > Area(A_R)$ then one or more rectangles will be searched in the difference $A_I - A_R$. For avoiding numerical issues and ignoring similar areas between plans, a minimum difference $\varepsilon = 0.09$ m^2 was considered; that is, if the difference between the image A_I and regressed rectangle's area A_R is lower than the area of an $\varepsilon = 30 \times 30$ cm wall's rectangle (which is the minimum column section in ACI 318-19 [54]), then the patch is discarded. Figure 5.12 details an example of the area computation for both input plans; Figure 5.12a shows each A_I area patch and Figure 5.12b for each A_R patch. Figure 5.13 illustrates the example of $A_I - A_R$ for each patch considering the ε minimum area threshold.



(a) Area patch for A_I image plan

(b) Area patch for A_R image plan

Figure 5.12: Example of area computation for each patch.

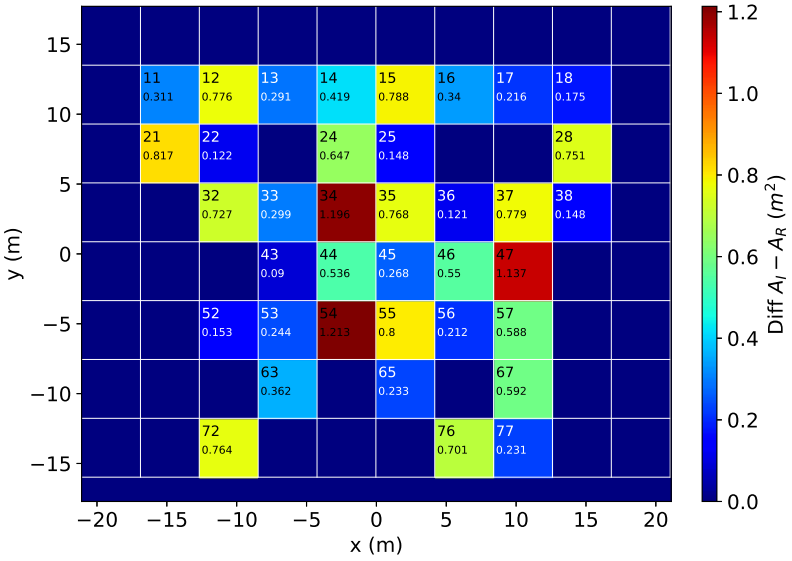


Figure 5.13: Example of the difference between A_I and A_R area patches. considering the minimum $\varepsilon = 0.09$ m^2 area threshold.

5.1.6.3. Patch selection for finding potential rectangle areas

The following process of the rectangle proposition framework is selecting the patches where the area difference $A_I - A_R$ exceeds the minimum threshold ε (Figure 5.13). Figure 5.14 details three examples for the patch A_I, A_R, and the difference $A_I - A_R$. It can be observed that the resulting area difference yields a complex geometry with irregular objects; also, the pixel class probability distribution (being class-0 the background, and class-1 the wall rectangles) is not uniform. Therefore, the number of rectangles to be proposed in the resulting area difference and their initial position to start the optimization mechanism for fine-tuning each proposition's geometry must be known. For achieving such a goal, a clustering method, described in the following subsection, was performed in the area difference for identifying the individual regions to propose a single rectangle that maximizes the intersection with the probability distribution (the likeness of each class).

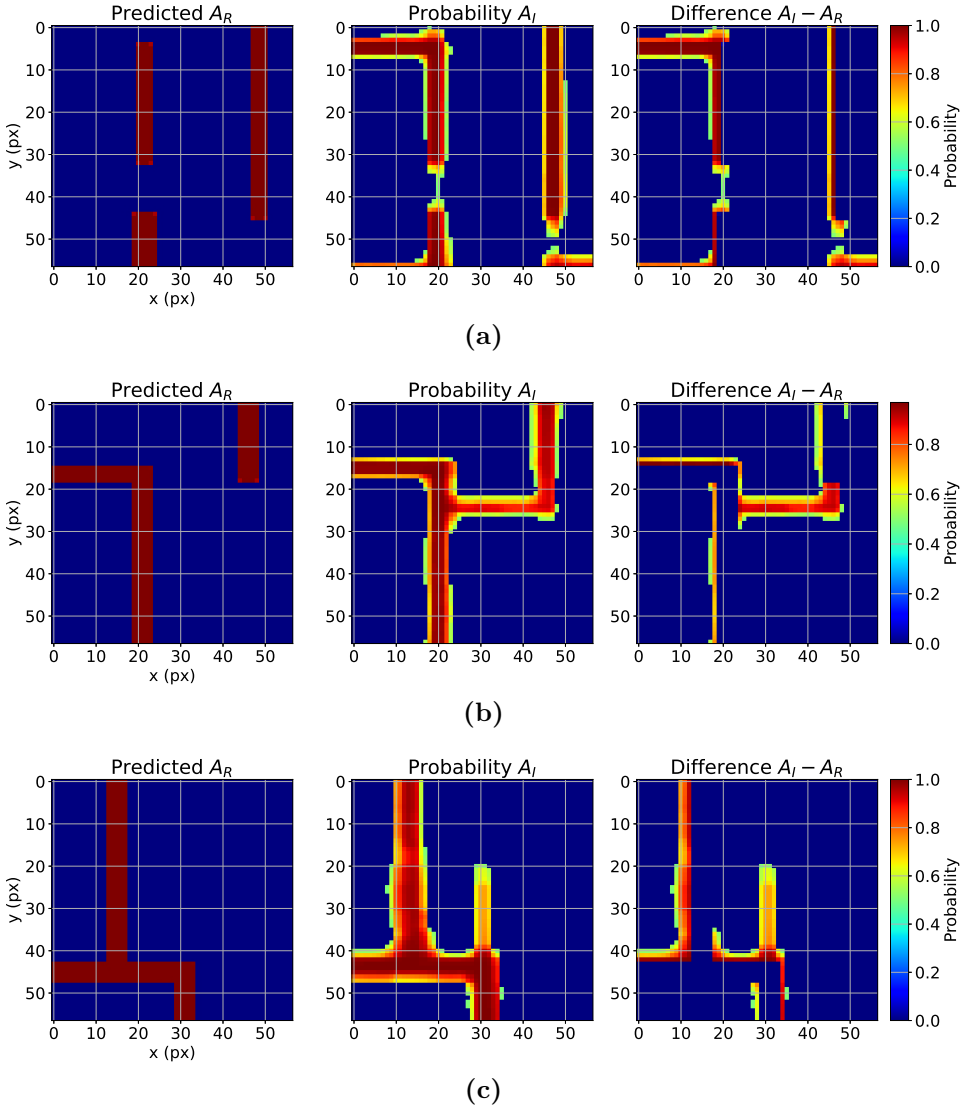


Figure 5.14: Example of three different patch selection where $A_I - A_R > \varepsilon$. For each figure, the left stands for the regressive plan A_R, the center for the A_I likely image, and the right for the difference $A_I - A_R$ between both images.

5.1.6.4. Area difference clustering for finding rectangle regions

With a focus on finding the total number of regions for proposing a rectangle that better fits the area distribution of the difference $A_I - A_R$, a clustering method was performed. Clustering is the task of grouping an object set, in this case, the probability area of the wall class likeness, in such a way that objects in the same group (called a cluster) are more similar (in some way) than to those in other groups (clusters). In this sense, clustering the area difference leads to obtaining the independent wall regions for estimating a new rectangle. Several clustering algorithms were studied, such as K-Means, Affinity Propagation [55], Mean Shift [56], Spectral Clustering [57], Agglomerative Clustering, OPTICS [58], and Birch [59]. However, those algorithms do not achieve good results due to noise and numerical problems related to the data distribution; also, not only the position of each pixel in the patch matrix must be considered, but also the class likeness value.

For the reasons mentioned above, the density-based clustering algorithm DBSCAN [60] lead to the best results; in this case, the density was considered as the class likeness probability resulting from $A_I - A_R$, values which range from 0 (background) to 1 (wall). For the implementation, SciPy [61] and Sklearn [62] Python libraries were used[6], leading to an algorithm complexity of $O(n^2)$ where n is the number of points; however, with spatial indexing (kd-tree or r-tree), the complexity is $O(n \log n)$. Figure 5.15 illustrates the clustering output of three examples shown in Figure 5.14; it can be noticed that the clusters are separated (in different colors), ranging from points, lines, L-shaped, and C-shaped polygons, to complex geometries. Also, subfigures (b) and (c) show the cases where a rectangle is split into two different regions (vertical one in $X = 10-13$ px, $Y = 10-64$ px in Figure 5.14b, and vertical in $X = 10-13$ px, $Y = 0-48$ px in Figure 5.14c). In these cases, the clustering and the continuation of splitted rectangles are preserved between patches.

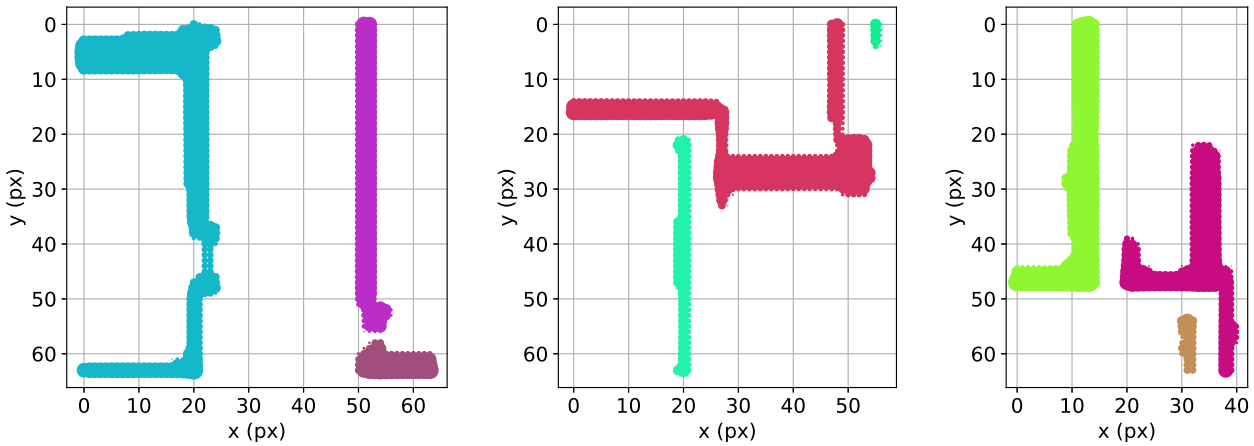
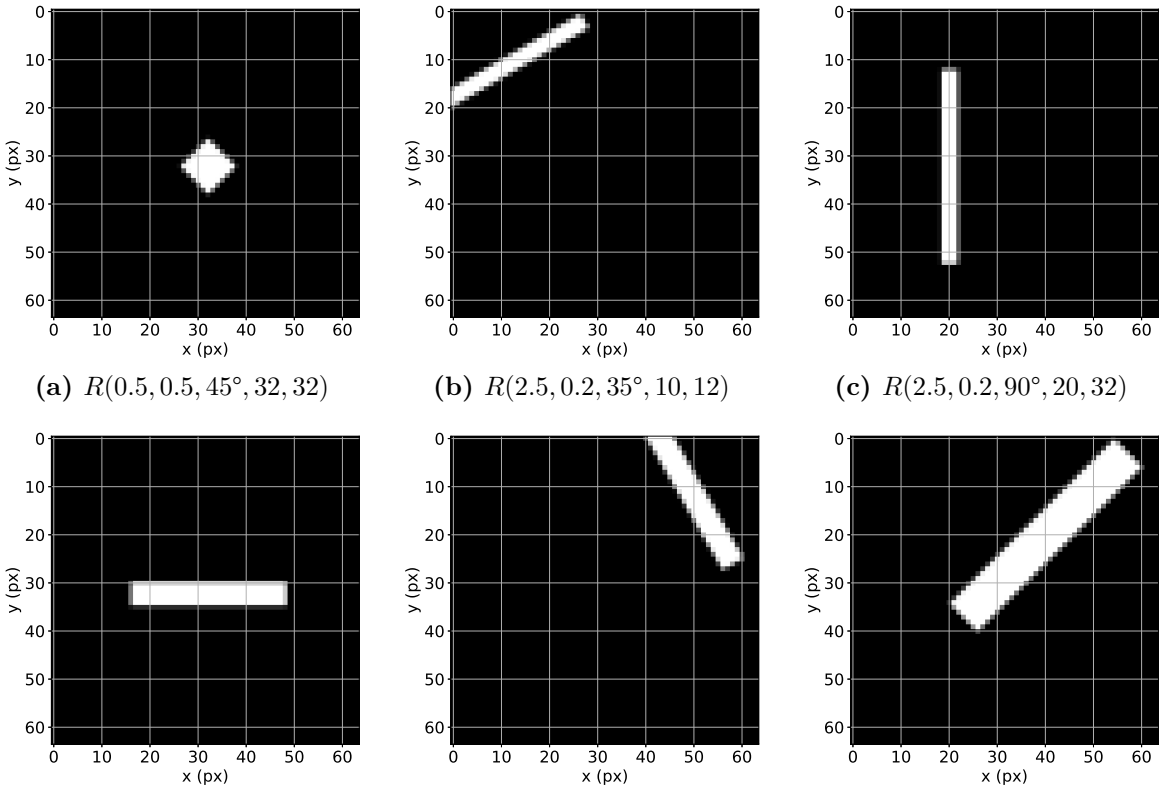


Figure 5.15: Clustering results of each $A_I - A_R$ area difference; illustrated in different colors.

[6] For implementing the DBSCAN in the Python Sklearn library, the *eps* parameter, which accounts for the maximum distance between two samples for one to be considered as in the neighborhood of the other, was set to $eps = 1.0$ for achieving the best results. This value was obtained through a trial and error procedure, considering the shape and results of the cluster output for different input scenarios.

5.1.6.5. Optimization algorithm for finding a rectangle on each cluster

The framework's final process consists of proposing a rectangle over each cluster that better maximizes the class probability likeness area and minimizes the intersection with the background class. In this sense, a rectangle is generated by a function $R(L, t, \theta, c_x, c_y)$, where L stands for the length (m), t for the thickness (m), θ for the angle (degrees), c_x for the x-axis center position (px), and c_y for the y-axis center position (px). For numerical purposes, the R function creates a matrix with the same 0.0625 m/px factor of the patches, 64x64 px, and 4x4 m length, constructed using OpenCV [37] and Shapely [40] Python libraries; the matrix contains only two numbers, 0 for modeling the background, and 1 for defining the rectangle object. Figure 5.16 illustrates an example of 6 different matrix outputs from the R function, ranging from horizontal, diagonal, and vertical objects with rectangular or square shapes.



(a) $R(0.5, 0.5, 45°, 32, 32)$ **(b)** $R(2.5, 0.2, 35°, 10, 12)$ **(c)** $R(2.5, 0.2, 90°, 20, 32)$

(d) $R(2, 0.3, 0°, 32, 32)$ **(e)** $R(2, 0.3, -30°, 50, 12)$ **(f)** $R(3, 0.5, 45°, 40, 20)$

Figure 5.16: Example of six matrix outputs from the $R(L, t, \theta, c_x, c_y)$ rectangle generator function.

The optimization task for finding the best value of L, t, θ, c_x, and c_y for each cluster C_i that approximates a rectangle $R(L, t, \theta, c_x, c_y)$ is performed by maximizing Eq. 5.5. This objective function considers the sum of each $[j, k]$ pixel of the 64x64 matrices resulting from the intersection of the rectangle R (a matrix, same as the examples shown in Figure 5.16) and the cluster C_i, penalized by the intersection of the rectangle and the complement of the cluster $1 - C_i$ (the external region), considering a λ factor of 0.5.

$$\max - \sum_{j=1}^{64} \sum_{k=1}^{64} \left(\Big(R \otimes C_i \Big)_{[j,k]} - \lambda \cdot \Big(R \otimes (1 - C_i) \Big)_{[j,k]} \right) \tag{5.5}$$

The penalizing λ factor was obtained by a trial-and-error procedure which took into account the cluster's output for several examples. However, that value can be fine-tuned by finding the best value which maximizes the intersection between the clusters and rectangles in the whole database. The $\otimes$ operator corresponds to the matrix element-wise multiplication, which describes the intersection of both matrices.

COBYLA (Constrained optimization by linear approximation) [63] algorithm was used to optimize Eq. 5.5. This method finds the solution that maximizes the objective function by performing a linear approximation considering an upper and lower bounds for each variable, whose described in Table 5.2, where $N = 4$ m stands for the width/height of each patch. If the bounds are not correctly fitted to the data distribution, the optimization mechanism does not improve as it tends to converge in local maxima; for that reason, mins&max were set for each variable. On the other hand, if the objective value defined by Eq. 5.5 is negative, or the resulting rectangle is lower than an object of 0.2×0.2 meters, the proposition is discarded.

Table 5.2: Description and bounds for the rectangle generation variables.

Variable	Description	Min. Bound	Max. Bound
L	Length (m)	0.2	N
t	Thickness (m)	0.2	0.5
θ	Angle (degrees)	-90	90
c_x	X-axis center (px)	$-\frac{N}{4}$	$\frac{N}{4}$
c_y	Y-axis center (px)	$-\frac{N}{4}$	$\frac{N}{4}$

For the initial guess, the L value starts as 75% of the maximum between the cluster's bounding box width and height, the thickness t starts at 20 cm, the rectangle's angle θ was set from the linear fit slope of the cluster's points, and for the center (p_x, p_y) a grid search was performed with a 15% offset of N margin for searching the best point that maximizes the area intersection, considering the cluster's center as a first approach. The computing time taken for each patch was 35 seconds on average; thus, a complete floor composed of 150 patches would need half an hour to complete if done in a single thread execution. However, the framework can be implemented in parallel, as the rectangle proposition is an independent problem that does not consider the surrounding rectangles in the algorithm's execution. For this reason, the framework was implemented considering 12 concurrent threads, which let solve a complete floor in under 5 minutes.

Figure 5.17 shows the results of nine-patches rectangle propositions, which contain up to 30 clusters. The center of each cluster is illustrated with a gray cross; similarly, the rectangle proposals are shown as a black box, where its center is marked with a black cross. Case (a) illustrates three clusters, the bottom left in $X = 60$ px, $Y = 65$ px captures a square cluster, a typical shape in the database, present in the example cases (d) and (f). The dark-sea-green cluster in case (a), with an L-shape, is captured in a rectangle near its center in $X = 50$ px, $Y = 15$ px, splitting the source cluster into two groups which can be later processed in a second iteration of the algorithm. L-shaped clusters are also common within the database; for example, case (i) illustrates an L-shaped cluster (in purple), where its rectangle proposition only considers a dense portion in $X = 3$ px and $Y = 12$ px. The light-blue cluster in case

(a), with a straight-line shape, is approximated by a long rectangular shape, with center $X = 35$ px and $Y = 5$ px, where its extreme ends are not captured while performing the optimization mechanism. These long-shaped clusters are also common within the database patches results; for example, cases (b) to (h) show several examples of rectangles that fit a linear cluster. Finally, case (e) shows an example of a non-trivial cluster, illustrated in blue color; a rectangle with center $X = 50$ px and $Y = 15$ px is proposed, but as the target cluster yields a complex shape, there are many areas not covered. Generally, the complex clusters must be processed with several rectangles; thus, after subtracting the proposals, the whole framework must be applied again for covering the cluster leftovers.

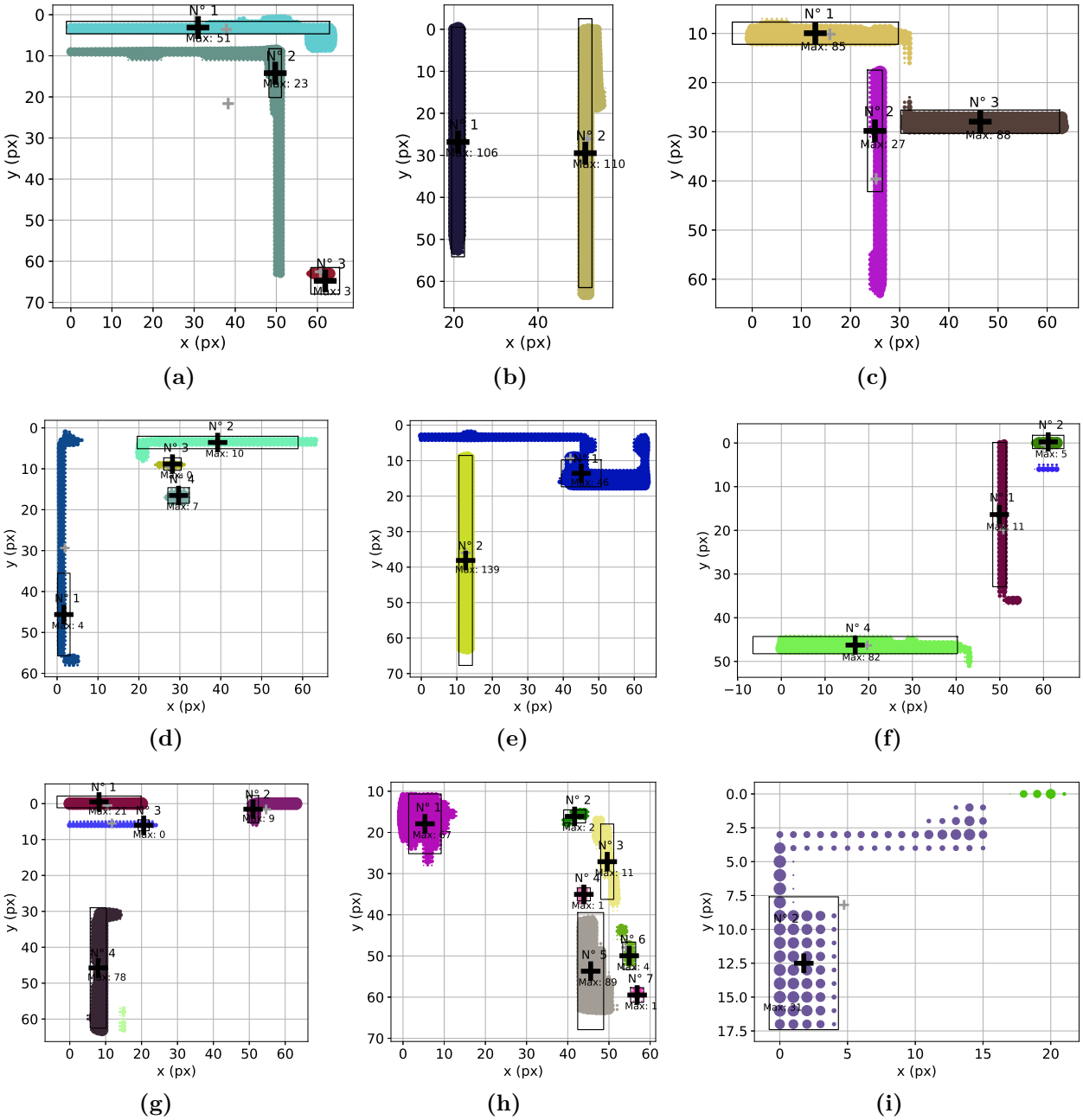


Figure 5.17: Rectangle output from a selection of nine patches, featuring a total of 30 clusters.

It is important to note that the rectangle proposition framework can be applied several times until area convergence; thus, all clusters are considered. In this case, the new rectangles proposed from the optimization mechanism are incorporated into the A_R plan, repeating the process. Typically, an entire floor does not need more than one pass to fulfill the area difference for each patch, with a maximum found of three passes. Figure 5.18 shows the rectangles' example for two different floors, a first-floor level (Figure 5.18a), which needed one pass, and a basement-type level (Figure 5.18c), completed with two passes; Figures 5.18b and 5.18d compare the real engineering solution for both cases. In case (a), six new propositions are evident, marked with circles. In case (c), five propositions are marked, where the optimization model identified new rectangles present in the image plan. Finally, the new rectangles (in pink) can be later merged and filtered to avoid intersections with the actual solution (black), dramatically reducing the number of proposals. The filtering can be solved through a score that considers the presence of other's proposals, the distance or intersection with regressed rectangles, among others.

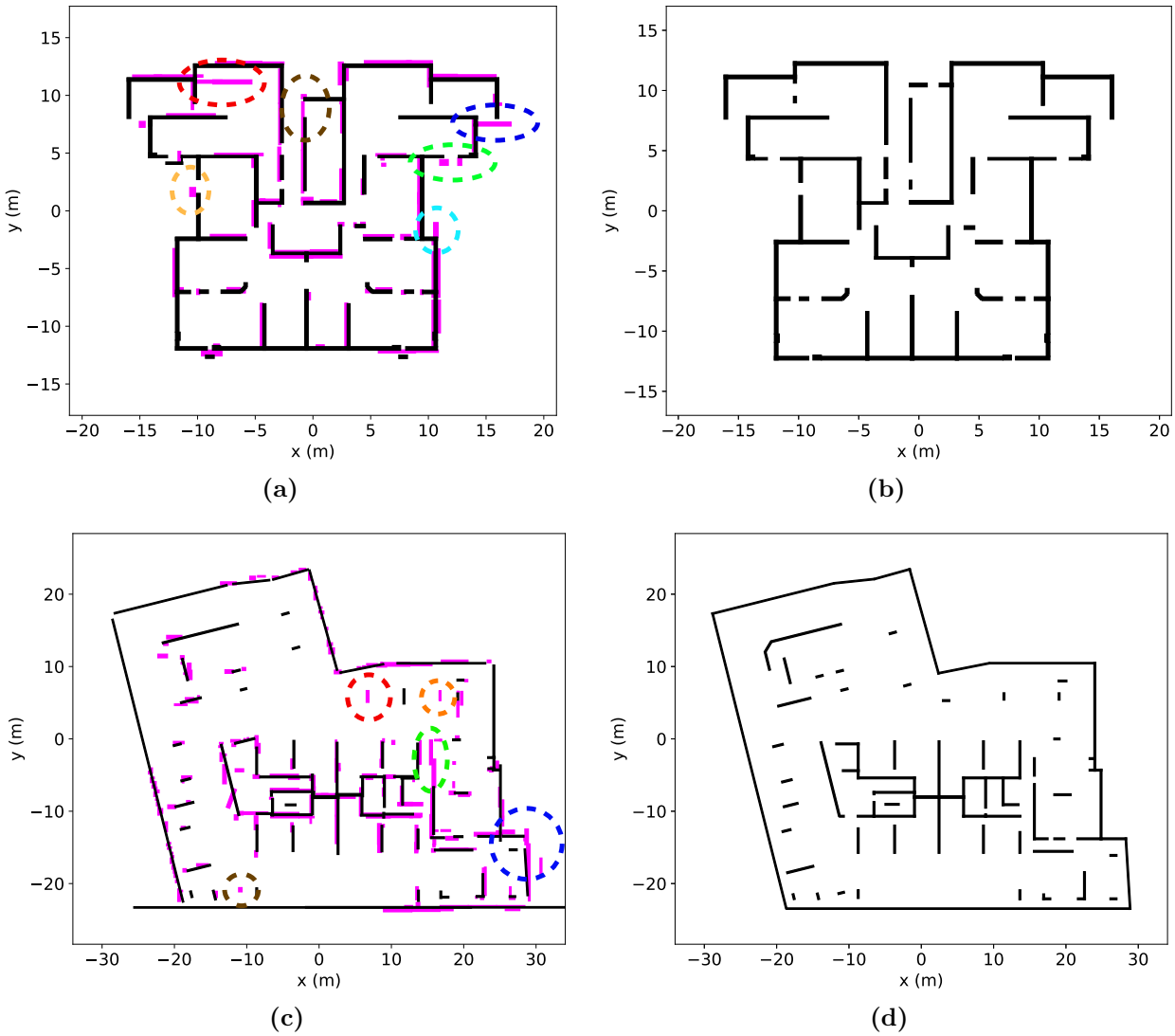


Figure 5.18: Floor's rectangle output proposition for all patches (in pink), compared to the regressive rectangle data from A_R output (in black). The right images show the real engineering solution for case (a) and (c).

5.2. CNN framework results

5.2.1. Regressive CNN-A and CNN-B models

The CNN-A and CNN-B model results were analyzed in terms of the coefficient of determination (R^2) and the confusion matrix, which contrasts the predicted and real class in bins that group ranges of values for each variable (e.g., the thickness). Dataset C (section 3.2) was used to obtain and analyze the results. Table 5.3 details the R^2-value for each parameter before applying the model between architecture and engineering values in the testing dataset without augmented data; it should be noted that $RectPosDx$ and $RectPosDy$ are not determined since these parameters only exist in engineering. It is also worth mentioning that the floor's geometric properties' parameters present a high initial correlation.

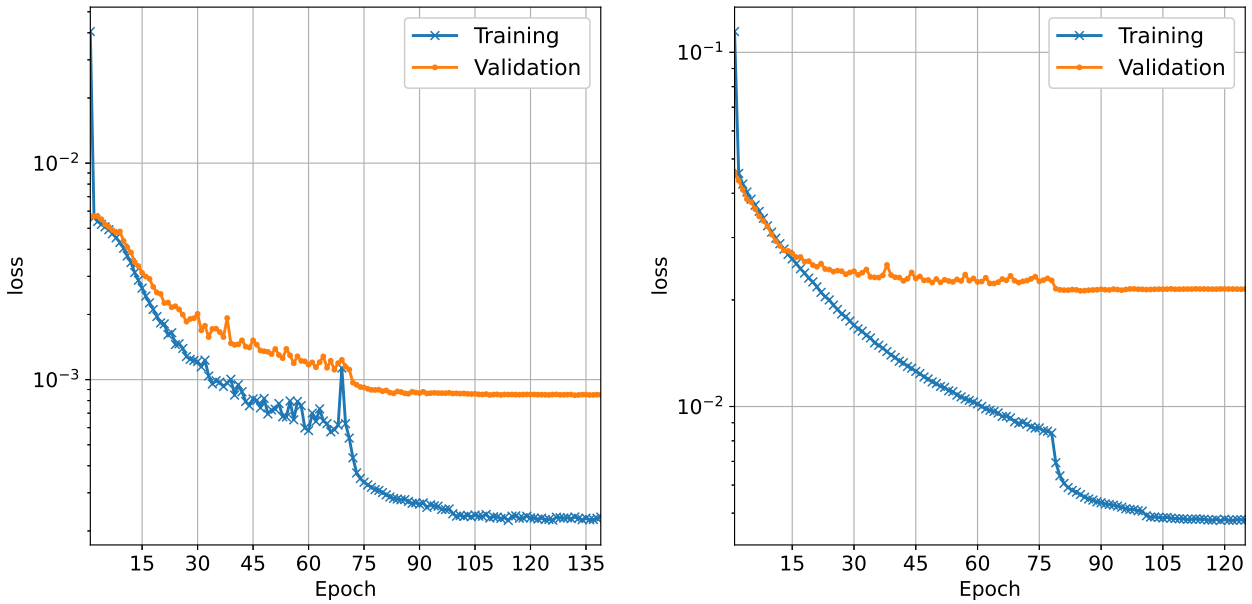
Table 5.3: R^2-value for each dataset C parameter, without the application of the model.

Parameter	R^2 Test dataset
RectThickness	0.826
RectLength	0.959
RectPosDx	–
RectPosDy	–
FloorWidth	1.000
FloorAspectRatio	0.999
FloorMassCenterDistance-x	0.985
FloorMassCenterDistance-y	0.987

Figure 5.19 illustrates the loss variation in both models' training process, showing a rapid convergence for the first three epochs. The effect of Keras ReduceLROnPlateau callback is reflected in the rapid loss decrease in the training curve at 72 and 98 epochs for model CNN-A and at 78 and 100 epochs for model CNN-B, with training times of 111 and 94 minutes, respectively[7]. Due to overfitting problems, the early stopping regularizer ended the training in 139 epochs for model A and 125 epochs for B. When using other loss functions such as $MAE(y_i, \hat{y}_i) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$ (Mean Absolute Error), or $MAPE(y_i, \hat{y}_i) = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$ (Mean Absolute Percentage Error), the models took much longer to converge, so they were not used.

Table 5.4 details the R^2-value in the test dataset without augmented data, comparing the result with ANN-based Sequential model, which considers the exact output parameters, with an architecture composed of 6 hidden layers of 1024 neurons, two layers of BN, Adam optimizer, ReLU activator, and MSE loss. A better result in the CNN-B model is observed due to the convolutional layers' image features. The CNN-A model obtained similar results to the Sequential model, so its use does not improve the methodology.

[7] Same as the development of ANN, an Intel® Core™ i7-9750H (12M Cache, 4.5 GHz, 6 cores) CPU, 16GB DDR4-2666 RAM, and NVIDIA® GeForce RTX™ 2070 8GB GDDR6 GPU was used.



(a) CNN-A model **(b)** CNN-B model

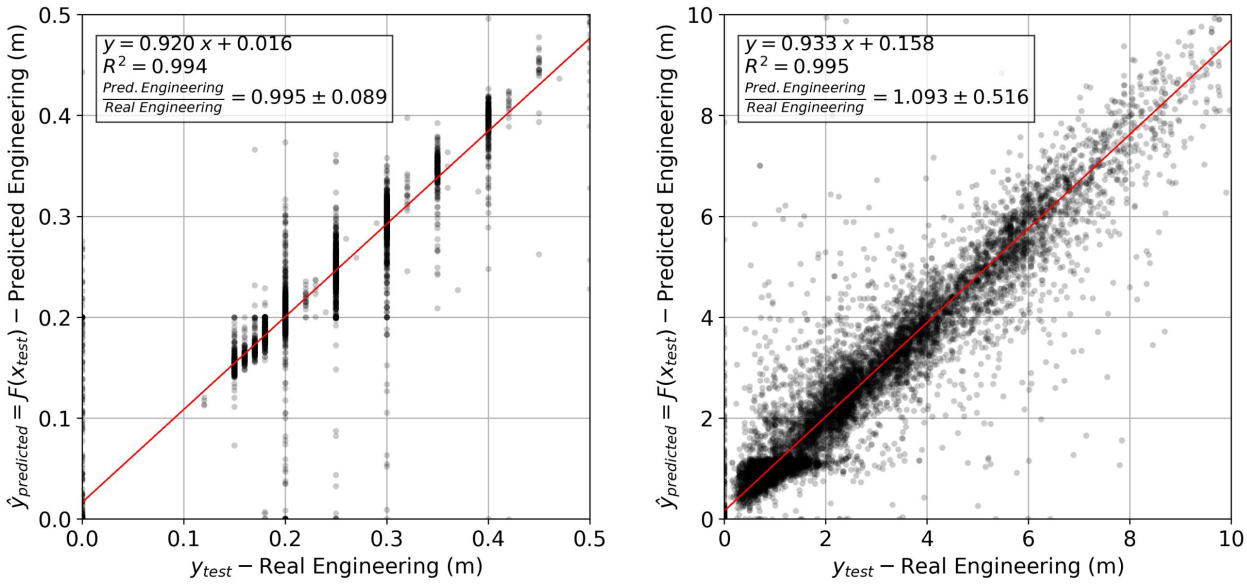
Figure 5.19: Loss curves (MSE) of the regressive CNN models.

Table 5.4: R^2-value of the regression for models CNN-A and CNN-B in the test dataset C.

Model	Parameter	R^2 Sequential Model	R^2 CNN Model
CNN-A	RectThickness	0.995	0.994
	RectLength	0.994	0.995
CNN-B	RectThickness	0.922	0.941
	RectLength	0.938	0.968
	RectPosDx	0.789	0.875
	RectPosDy	0.781	0.830
	FloorWidth	1.000	1.000
	FloorAspectRatio	1.000	1.000
	FloorMassCenterDistance-x	1.000	1.000
	FloorMassCenterDistance-y	1.000	1.000

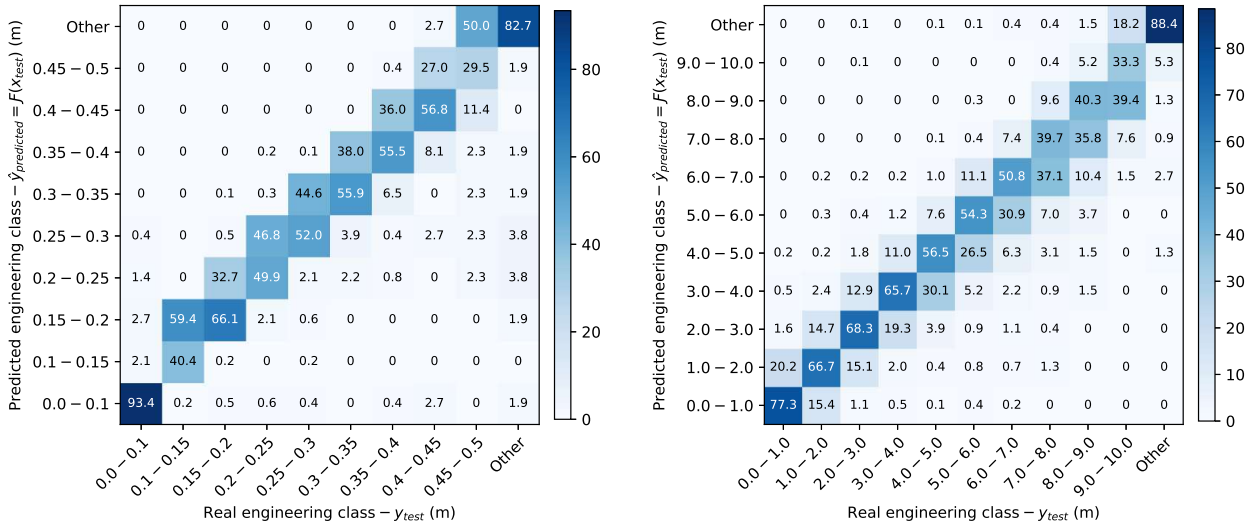
Figures 5.20 and 5.22 present the correlation results of the values predicted by the model $\hat{y}_{predicted} = \mathcal{F}(x_{test})$ against the real values in engineering (y_{test}), where x_{test} corresponds to the value of the architectural features, and $\mathcal{F}(x)$ to the model used. There is a good correlation associated with the geometric properties of thickness and length and the floor plan geometry prediction. The R^2-value is used to measure accuracy and the mean and standard deviation of the ratio between the predicted value over the real value, shown in Figures 5.20 and 5.22 as "$\frac{\text{Pred. Engineering}}{\text{Real Engineering}}$." Regarding the translation prediction, there is a quasi-linear response (Figures 5.22a and 5.22b), with a linear representation close to $y = 0.6x$; an important translation prediction is evidenced in rectangles that should not move, showing data in a cross-shaped trend on the axes $(y_{test}, 0)$ and $(0, y_{predicted})$.

Figures 5.21 and 5.23 show the correlation matrices for each of the outputs of models CNN-A and CNN-B; each value indicates the percentage of predicted accuracy, where the sum of each column is equivalent to 100% of the data in the respective real class; a good correlation is observed in all the predicted parameters. The ranges used in each correlation graph and confusion matrices represent $98 \pm 1.43\%$ of the data.



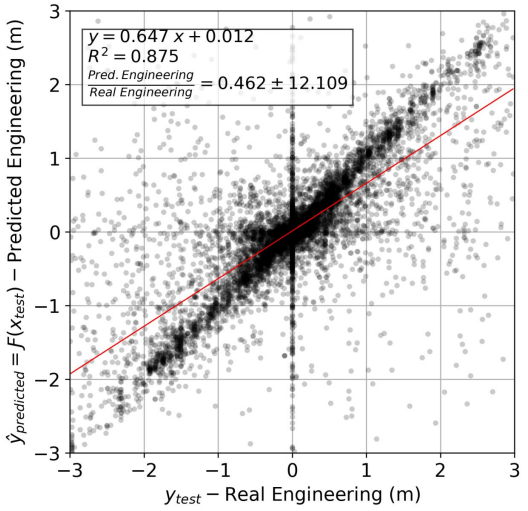
(a) Thickness **(b)** Length

Figure 5.20: Correlation results of the CNN-A model.

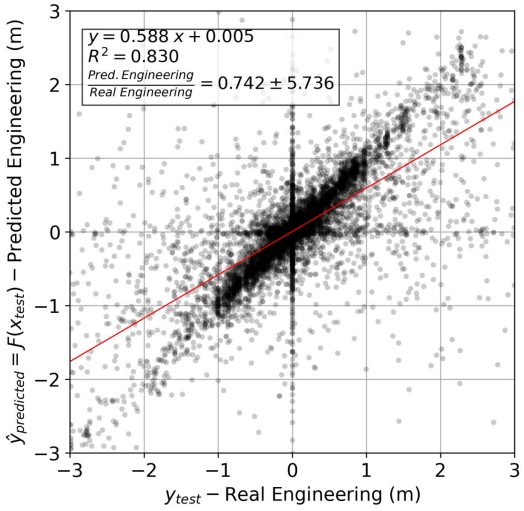


(a) Thickness **(b)** Length

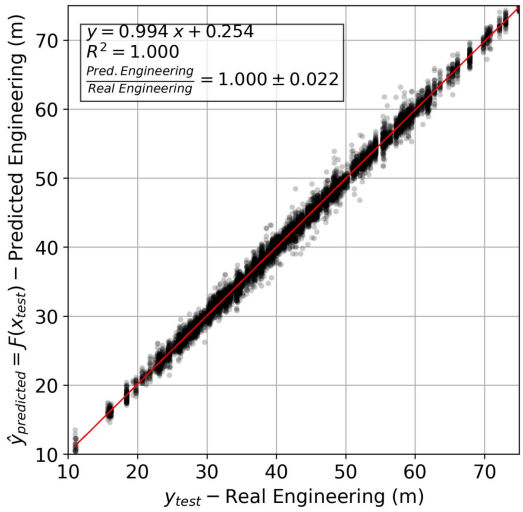
Figure 5.21: CNN-A model confusion matrices.



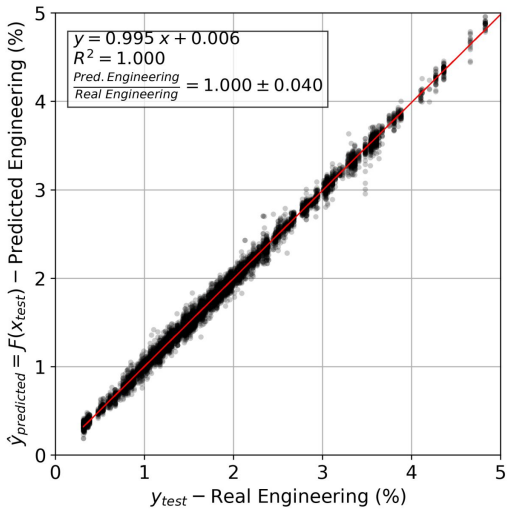
(a) *RectPosDx*



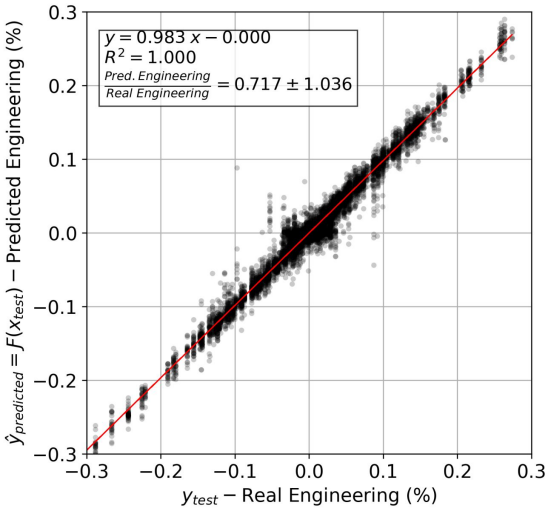
(b) *RectPosDy*



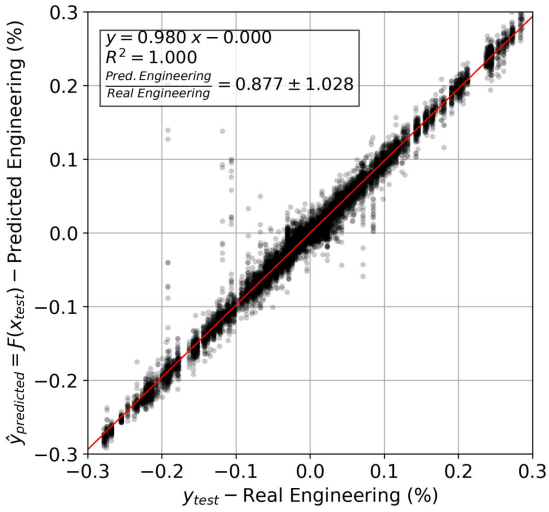
(c) Floor plan width



(d) Floor plan aspect ratio

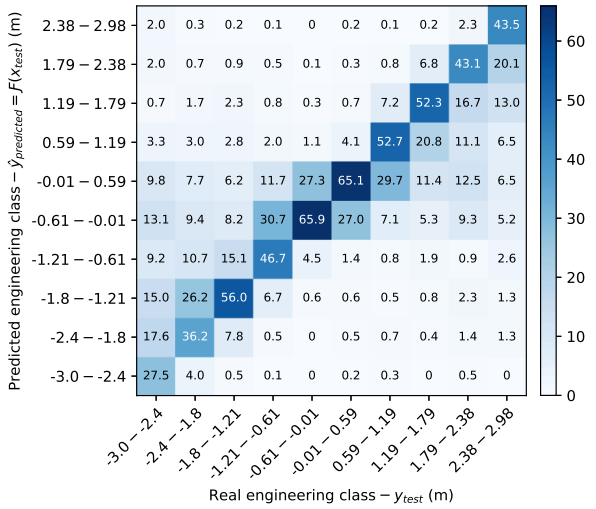


(e) Normalized distance between MC and GC, x-axis (*FloorMassCenterDistance-x*)

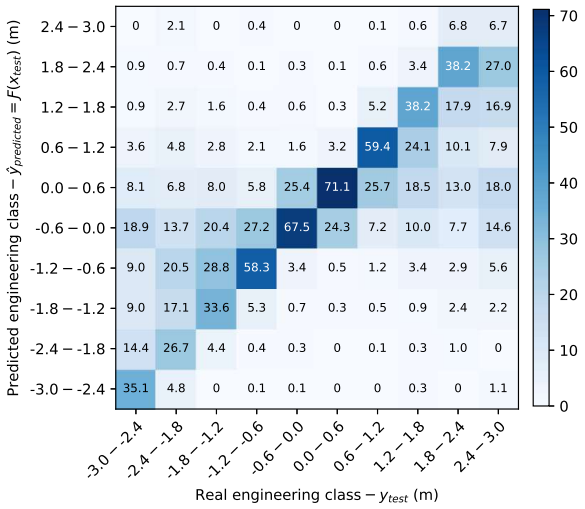


(f) Normalized distance between MC and GC, y-axis (*FloorMassCenterDistance-y*)

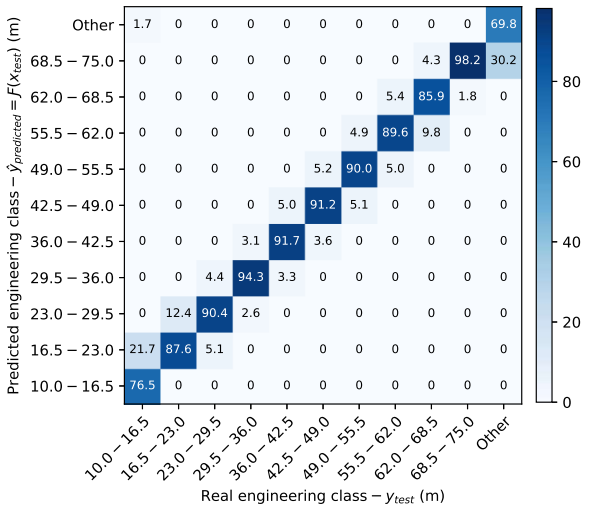
Figure 5.22: Correlation results of the CNN-B model.



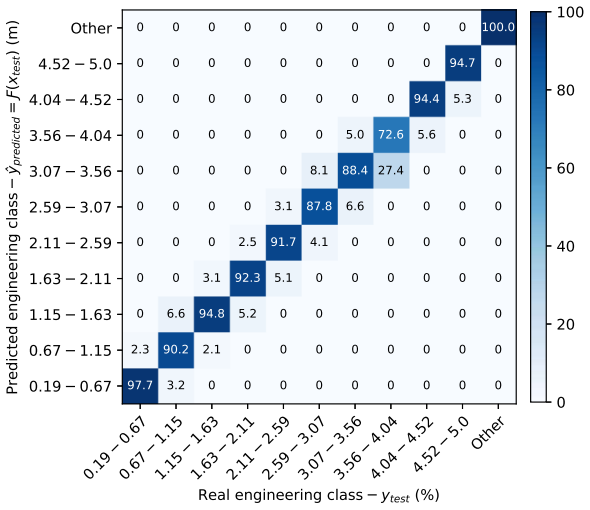
(a) *RectPosDx*



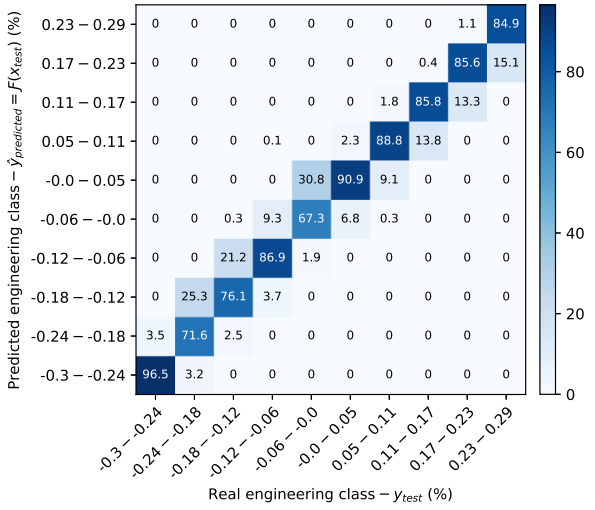
(b) *RectPosDy*



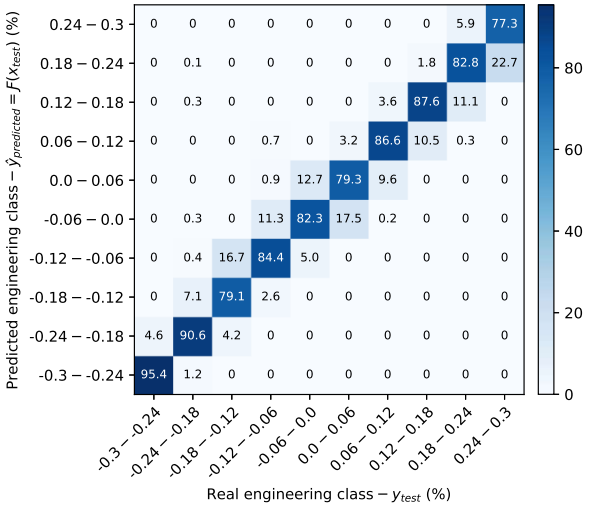
(c) Floor plan width



(d) Floor plan aspect ratio



(e) Normalized distance between MC and GC, x-axis (*FloorMassCenterDistance-x*)

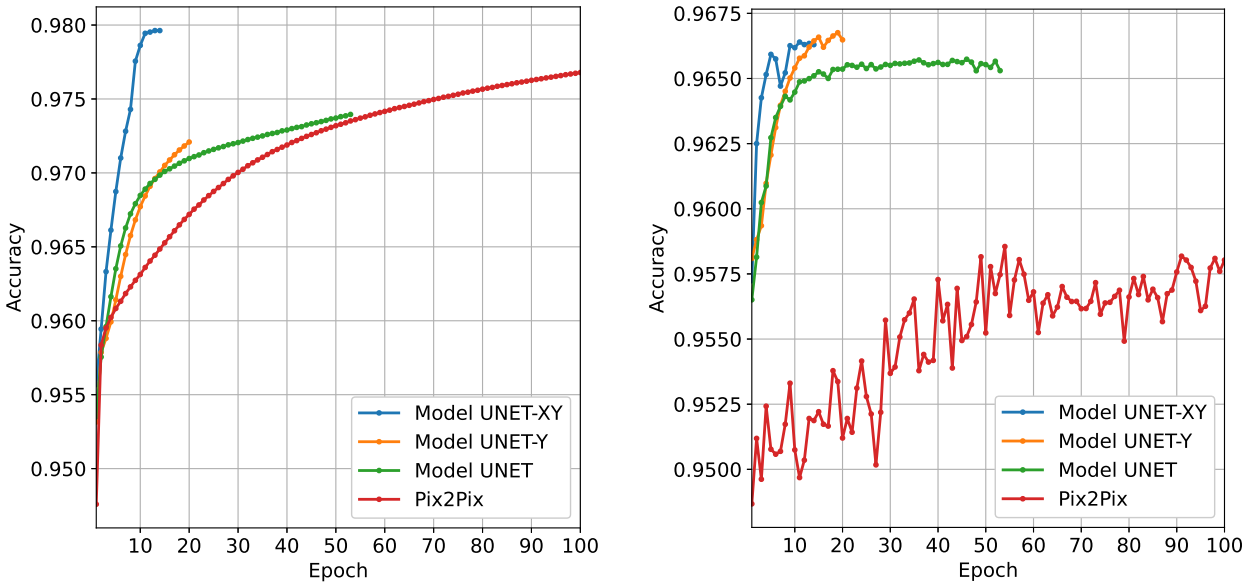


(f) Normalized distance between MC and GC, y-axis (*FloorMassCenterDistance-y*)

Figure 5.23: CNN-B model confusion matrices.

5.2.2. Likely image plan generation

Figure 5.24 shows the variation of the UNET, UNET-Y, UNET-XY, and Pix2Pix models' binary accuracy metric during the training process (Figure 5.24a) and testing (Figure 5.24b). The best values are achieved with the UNET-XY model in the least number of epochs, followed by UNET-Y, UNET, and finally, the Pix2Pix model, wherein the latter presents a high accuracy oscillation in testing. Table 5.5 presents the training time, the total number of epochs, and the mean accuracy plus the standard deviation for the training and test datasets in the final epoch (Figure 5.24). Although UNET-XY achieves similar results as UNET-Y for the test case, the training process is faster, takes fewer epochs to converge, and has a better distribution. It is worth noticing that without model application, an accuracy close to 95% is achieved, similar to the percentage of pixels that do not represent walls, being the dominant image class.



(a) Training, total images = 206,711

(b) Validation, total images = 88,591

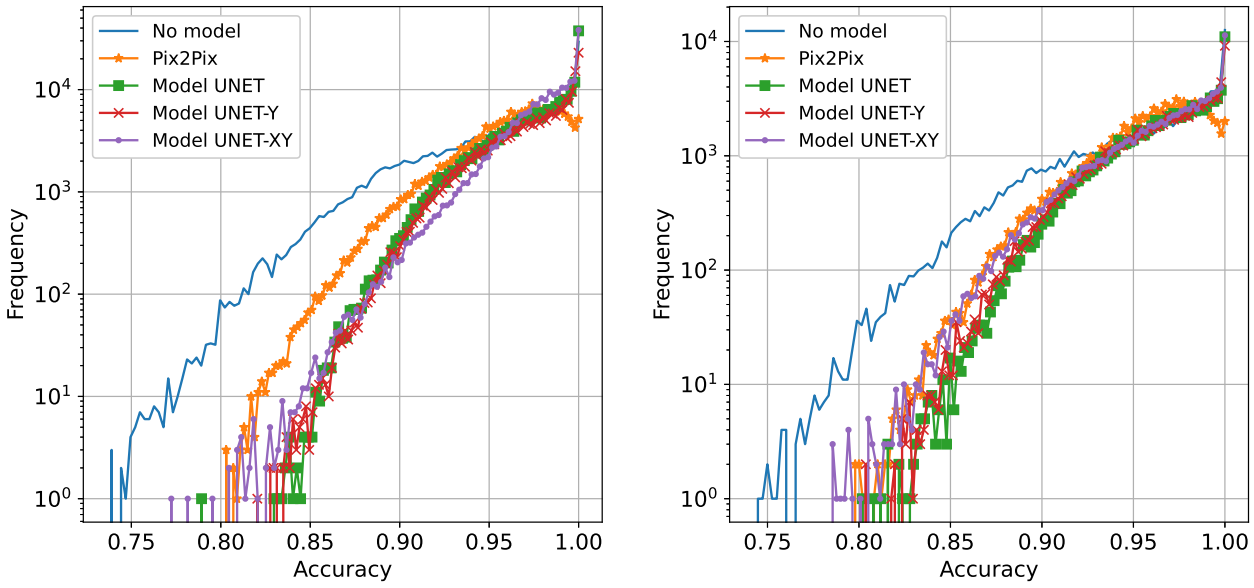
Figure 5.24: Binary accuracy curves of the engineering image for the different predictive models UNET, UNET-Y, UNET-XY, and Pix2Pix.

Table 5.5: Main training results of the likely engineering image models.

Model	Train epochs	Train time (min)	Train avg. accuracy	Test avg. accuracy
No model	–	–	0.956 ± 0.042	0.956 ± 0.043
UNET-XY	15	154	0.980 ± 0.022	0.966 ± 0.024
UNET-Y	20	165	0.972 ± 0.026	0.966 ± 0.029
U-Net	52	496	0.973 ± 0.026	0.965 ± 0.028
Pix2Pix	100	1,060	0.977 ± 0.029	0.958 ± 0.030

The histogram in Figure 5.25 illustrates the binary accuracy in the training and test datasets. It is observed many cases with a value close to 1.0, mainly associated with images

that are identical in architecture and engineering, or cases with few or no rectangles; this, added to the significant asymmetry in the distribution of the pixel classes, makes the training of the model more complicated, since it tends to generate the same image without applying any transformation. In this sense, UNET-XY transforms the input image forcing the model to learn while improving the training performance. However, it does not significantly improve the generalization capacity (evaluation with new data) but slightly increases the binary accuracy standard deviation, and takes less time to train, as shown in Table 5.5. Thus, for deployment purposes or further development, the UNET-XY model is recommended.

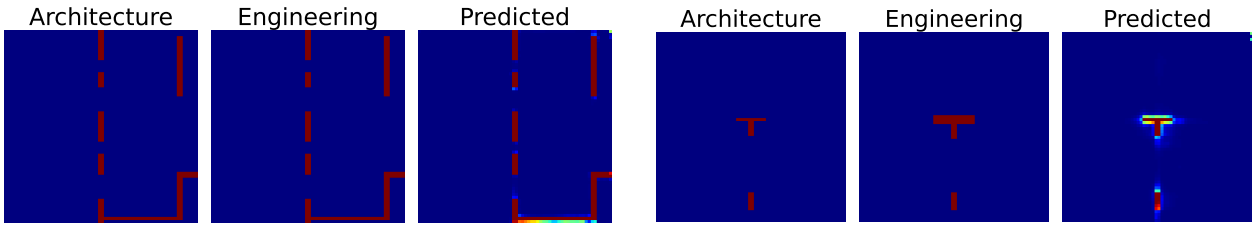


(a) Train **(b)** Test

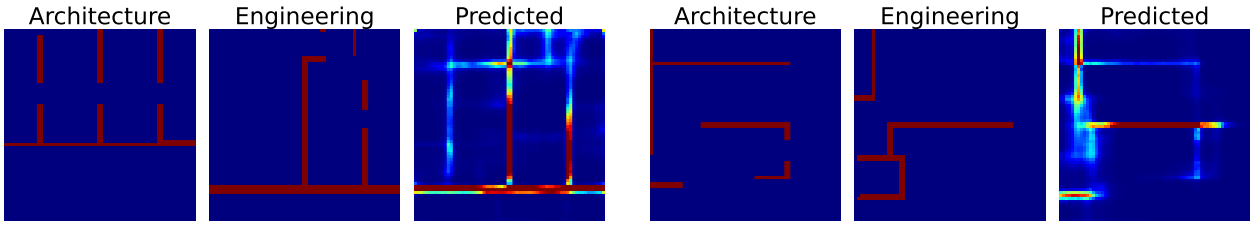
Figure 5.25: Histogram accuracy of the predicted images in the train and test dataset.

Figure 5.26 shows a selection of 8 images predicted by the UNET-XY model in the test dataset. Cases (a) and (b) correspond to those examples wherein the image of both architecture and engineering are very similar in the general floor plan geometry, presenting differences only in the element thickness or slight translations. These cases produce a high binary accuracy value above 0.99, corresponding to 27% of the data; in case (b), it is observed that the model predicts a larger thickness in the origin (central) rectangle, which corresponds to the proper rectangle used in engineering. On the other hand, in case (c), the horizontal wall below the origin rectangle is predicted, in addition to maintaining some features of the original architectural image (left vertical rectangle). Case (d) maintains the origin rectangle while eliminating some architectural elements such as the upper horizontal wall. Case (e) predicts a translation of the two upper rectangles of the hallway. In case (f), the model can predict the elevator shaft wall, keeping most of the elements defined in architecture present in the engineering floor plan. In case (g), although the architectural and engineering images are very different, the model can predict the two horizontal walls and the vertical one, eliminating several elements. Finally, (h) correspond to a case where the image between architecture and engineering presents a considerable variation, so the image's prediction model has a poor performance, achieving an accuracy lower than 0.9, representing 5% of the test dataset; these cases occur for two main reasons, one associated with the fact that the floor plan of the

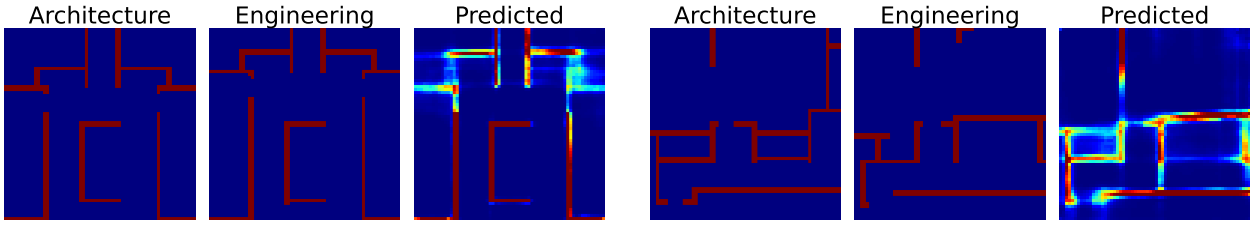
first architecture version and the latest engineering version presents a significant variation, or because of a wrong association between the architecture and engineering rectangles given the heuristic formulation used in the creation of these pairs (section 2.3), which induces the model to learn a conceptually wrong transformation of the image.



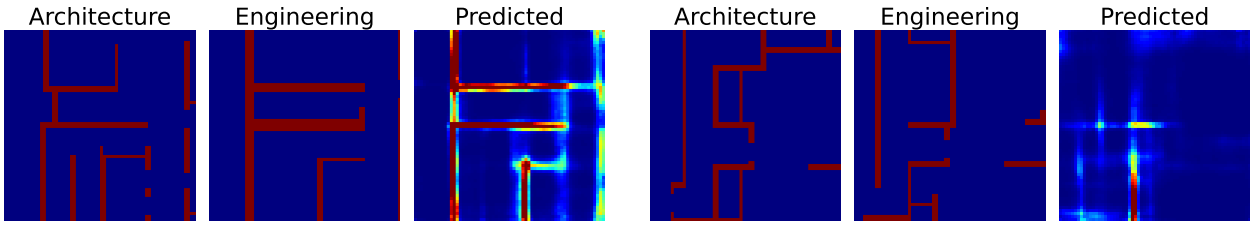
(a) Model: 0.999, No model: 1.000 **(b)** Model: 0.993, No model: 0.992



(c) Model: 0.972, No model: 0.881 **(d)** Model: 0.970, No model: 0.927



(e) Model: 0.965, No model: 0.939 **(f)** Model: 0.953, No model: 0.891



(g) Model: 0.942, No model: 0.853 **(h)** Model: 0.901, No model: 0.846

Figure 5.26: Selection of 8 examples of the likely image generation in engineering using the UNET-XY model in the test dataset, along the binary accuracy metric with and without the model's application.

5.2.3. Assemble of the predicted final engineering plan (A_P)

For the study of the CNN regression model results (Regressive Plan matrix A_R), and the UNET-XY image generation results (Likely Image Plan matrix A_I) in the assembling of the Final Engineering Plan (matrix A_P), the metric IoU (Intersection over Union), commonly applied in object detection problems, was used. IoU corresponds to the quotient between the intersection of the real engineering floor plan's rectangles (called matrix A_Y) and the predicted floor plan A_P, divided by the sum of the actual and predicted solutions. In this framework, the predicted solution can be $A_P = A_R$ (using only the regressive results through CNN-A and CNN-B models, described in section 5.1.2), $A_P = A_I$ (using only the images generated by UNET-XY, described in section 5.1.4), or $A_P = A_R \oplus A_I$ (the combination of both models, described in section 5.1.5). The value of IoU varies between 0 (null intersection) and 1 (perfect intersection). Given that the problem was treated in a matrix form, IoU was calculated as $\frac{A_Y \oplus A_P}{A_Y \otimes A_P}$, where the operator $\otimes$ corresponds to the matrix element-wise multiplication.

For evaluating the IoU metric, the mean and the standard deviation were calculated using the 165 database projects, composed of 477-floor plans in architecture (basement, first floor, and typical floor). The complete results are presented in Table 5.6, wherein the value obtained in the case without applying any model and the combinations between CNN models (CNN-A, CNN-B) and UNET-XY models (likely image generation) are described. For evaluation purposes, the output function $f_{sigmoidh}$ was used. It is possible to observe that applying only CNN-A or CNN-B model does not decrease the error, nor does it increase the average value of IoU; however, when using both models in combination, this contributes to reducing the variance, which is consistent with the improvement of the R^2-value for the thickness, the length, the rectangle's translation, and the floor plan's geometric properties. On the other hand, the application of the image allows increasing the average IoU in all cases.

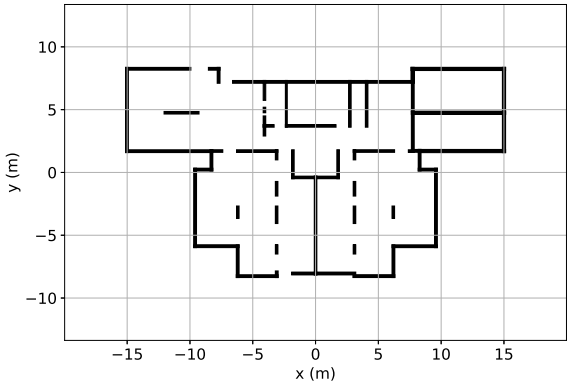
For the study of the effect of the output function in the generation of the image given by the UNET-XY model, the average IoU was calculated considering the nine output functions proposed (section 5.1.4), without considering the rectangles of the prediction (Matrix A_R); thus, $f_{\text{linear}} = 0.19 \pm 0.092$, $f_{\text{sigmoid}} = 0.144 \pm 0.097$, $f_{\text{sigmoidh}} = 0.234 \pm 0.097$, $f_{\text{tanh}} = 0.123 \pm 0.101$, $f_{\cos} = 0.167 \pm 0.096$, $f_{\text{square}} = 0.141 \pm 0.087$, $f_{\text{quad}} = 0.087 \pm 0.071$, $f_{\text{isquare}} = 0.21 \pm 0.097$, and $f_{\text{iquad}} = 0.227 \pm 0.098$. The best result was obtained with $f_{sigmoidh}$.

Table 5.6: Value of the IoU metric for different combinations of the regressive CNN and UNET-XY models for assembling the final engineering floor plan.

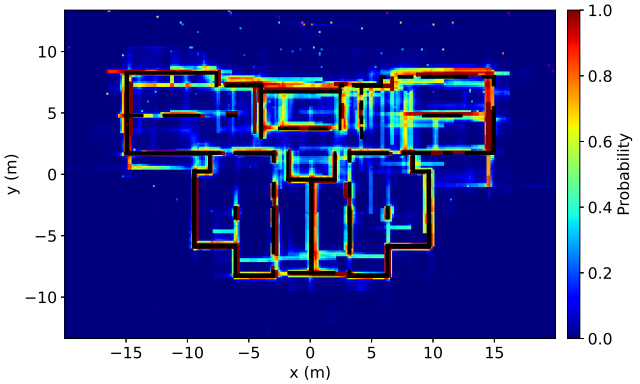
Regressive model	**Use of likely image generation**	
	No	**Yes**
None	0.471 ± 0.282	0.596 ± 0.255
CNN-A	0.463 ± 0.273	0.597 ± 0.253
CNN-B	0.313 ± 0.240	0.486 ± 0.245
CNN-A + CNN-B	0.424 ± 0.185	0.627 ± 0.174

Figures 5.27 through 5.30 show different results of the floor plan prediction with different IoU levels; Figure 5.27 presents a superior case, with an IoU = 0.787 (77th percentile), where the high intersection between the predicted and real solution is visualized. In this case, the model can predict new elements such as horizontal rectangles at $X = (-15, -5)$ and $Y = 5$, the reinforcement of the connections between elements, and the presence of horizontal walls at $X = (-6, 6)$, $Y = -6.5$ suggested by UNET-XY. Figure 5.28 represents an above-average case, the elimination of the diagonal walls in the perimeter of the stairway box between $X = (-4, 4)$ and $Y = (2, 6)$ stands out, and the correct prediction of the vertical walls in $X = -10$, $Y = (-10, -5)$ and $X = 10$, $Y = (-10, -5)$. The images also allow modeling the upper walls between $X = (-2.5, 0)$ and $Y = (7, 11)$, which are not present in the architecture floor plan. In Figure 5.29, there is a case below the average, with an IoU = 0.6 (38th percentile), which highlights the remarkable similarity of the walls of the central hallway, in $X = (-2, 18)$ and $Y = (0, 3)$, in addition to a correct prediction of the walls under the stairway box, at $X = (-10, -5)$ and $Y = (-3, 0)$. Finally, Figure 5.30 shows an example within the lower spectrum of the distribution, IoU = 0.413 (11th percentile), representing those buildings that present a significant variation between the architectural and engineering floor plan. For example, the following situations reduce the accuracy: (a) the difference in the direction of the perimeter walls; (b) the diaphragm area at the level of the stair is similar, but with a different location, which produces an error in the prediction; (c) the incorporation of images generates a large number of walls within the hallway region that does not correspond to the real floor plan, like the horizontal walls at $Y = -8$ m that are maintained from architecture without being eliminated by UNET-XY.

In general, the results indicate that most cases present an excellent correlation associated with high values of IoU, which provides not only an accurate prediction of the walls geometry, but also the probable location and dimensions of new walls that are not anticipated in the architectural plans.

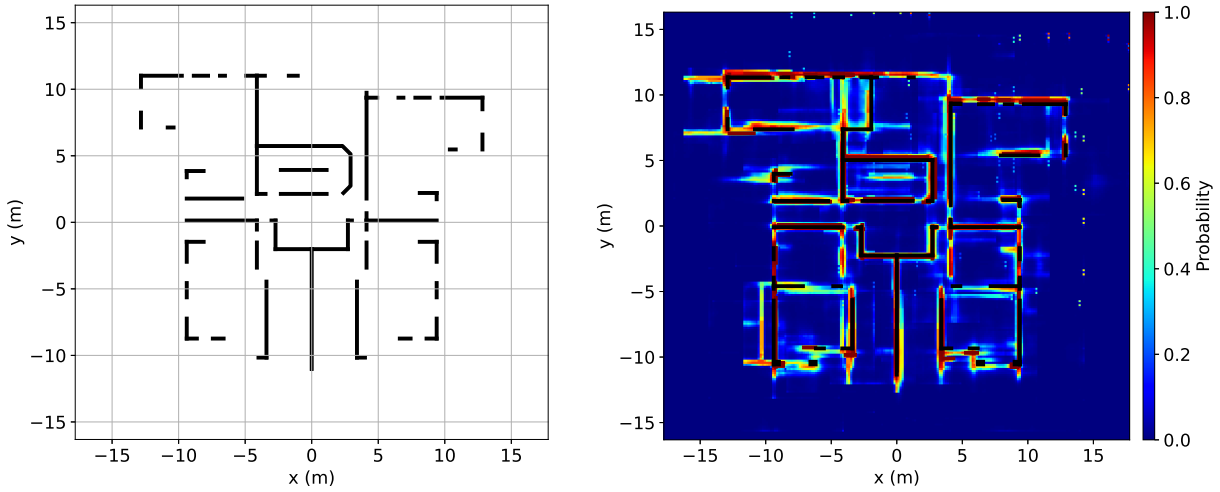


(a) Architectural plan



(b) Predicted plan, comparison with engineering (black)

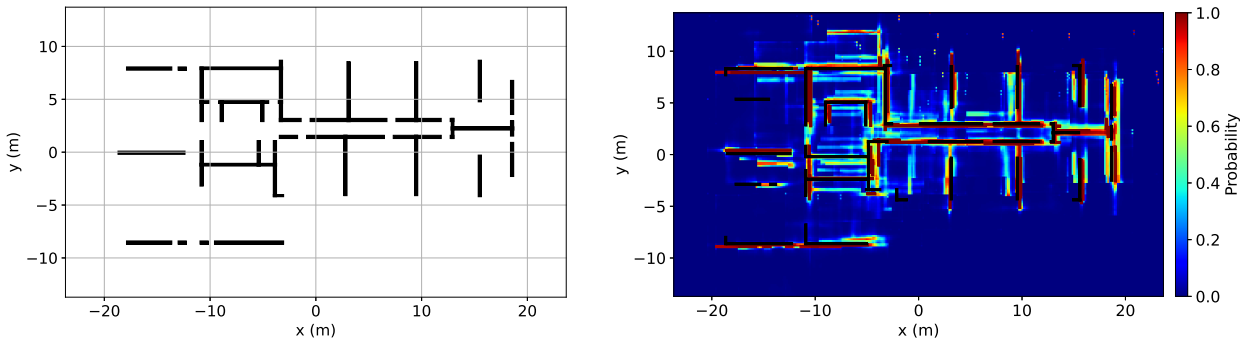
Figure 5.27: Example of plan prediction, IoU = 0.787, percentile 77th. IoU without model = 0.631.



(a) Architectural plan

(b) Predicted plan, comparison with engineering (black)

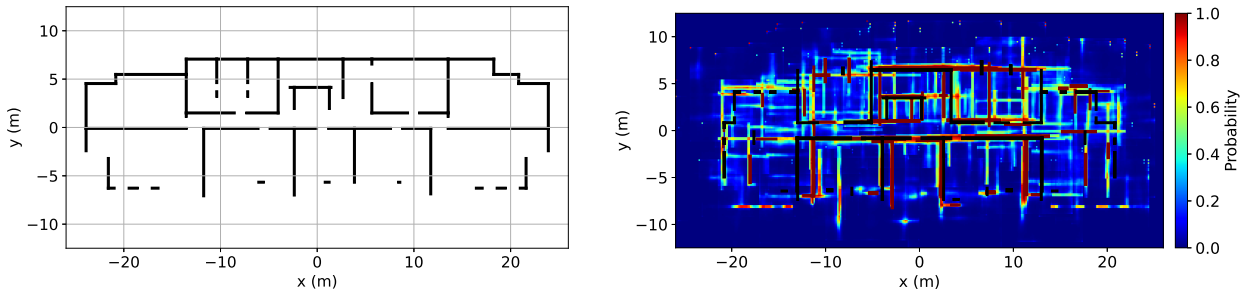
Figure 5.28: Example of plan prediction, IoU = 0.699, percentile 58th. IoU without model = 0.388.



(a) Architectural plan

(b) Predicted plan, comparison with engineering (black)

Figure 5.29: Example of plan prediction, IoU = 0.6, percentile 38th. IoU without model = 0.446.



(a) Architectural plan

(b) Predicted plan, comparison with engineering (black)

Figure 5.30: Example of plan prediction, IoU = 0.413, percentile 11th. IoU without model = 0.038.

Chapter 6

Conclusions

Structural design is a task that requires high interaction between the architecture and engineering teams when defining each of the walls distributed within the floor plans of a building. This process inherently generates a large amount of data, which can be used to predict the variation of wall dimension and location when comparing the first architecture and the final engineering plans, aside from predicting new walls due to engineering requirements.

Given the recent development of artificial intelligence (AI) and computational capacity, a machine learning (ML) algorithm, based on artificial neural networks (ANN), was used to develop a methodology for the conceptual design of the wall layout of Chilean residential buildings. Neural networks in this sense allow modeling complex interaction scenarios between an input and output domains, in this case, the hyperdimensional mapping between the architectural and engineering plan information, allowing to encode the knowledge, experience, and know-how of the engineering design process embedded in the floor plans from an adequately processed database.

As a first approach of developing a ML-based structural wall layout design, a database of 165 Chilean residential projects of reinforced concrete shear wall buildings was created, considering a rectangular wall discretization. For each rectangle, a total of 30 numerical features were calculated (RENE), accounting for geometric and topological properties. These features fed a regressive ANN model to predict the engineering values of thickness and length from the architectural input, archiving remarkable results in terms of the coefficient of determination (R^2) of 0.995 and 0.994 for the thickness and length, respectively. The model performance shows that the rectangular discretization and the computed features adequately represent each wall's geometric and semantic information; moreover, the association mechanism between architectural and structural objects, although based on a complex heuristic formula, allows a correct assignment of the pairs. However, a regressive model of this nature does not incorporate spatial detail or contextual information of each wall perimeter, and also, the prediction of other parameters such as the wall translation has a poor performance.

As convolutional neural networks (CNN) allow for obtaining more contextual and non-trivial features is that a CNN framework was proposed to generate the final engineering floor plan. For such purpose, two independent plan predictions were combined, which consider the

architectural data as input, such as the numerical 30-feature input vector (RENE) and the surrounding walls' image for each rectangle. The proposed image size is 64x64 pixels (px), where the source rectangle is centered and covers a 10x10 meters (m) region of its respective floor plan. The first plan prediction is assembled using two regressive CNN models, CNN-A and CNN-B, fed by the 30-feature vector (RENE) and the 64x64 px images. CNN-A model allows the prediction of the rectangles' thickness and length, with an R^2-value of 0.994 and 0.995, respectively. CNN-B model predicts the element's translation between the architectural and engineering floor plan with an R^2-value of 0.875 and 0.83 for the x and y-axis. CNN-B also predicts the geometric properties of the bounding box of the engineering floor plan with an R^2-value equal to 1.0 for the width, aspect ratio, and the offset between the geometric center and the mass center on both main axes. The second plan prediction is assembled using a model (UNET-XY) that generates a likely engineering floor plan image for each rectangle given the same architectural input mentioned above; UNET-XY reached an average binary accuracy metric of 0.966 in the test dataset. In this context, it is observed that the incorporation of the images for each rectangle improves the performance of the model prediction by adding a new spatial dimension to the input, allowing the calculation of new low, medium, and high-level features due to the model's convolutional layers that account for more properties at geometric and topological levels. Such more complex properties cannot be captured with only the original 30-feature vector (RENE).

Both independently predicted plans (from CNNs & UNET-XY) are combined to lead the final engineering floor plan, with a mean result in terms of the intersection over union (IoU) of 0.627 ± 0.174. This generated plan allows predicting the wall's rectangles design parameters and proposes new structural elements not present in architecture, making the methodology an excellent candidate to accelerate the building wall layout's early conceptual design.

Despite the results obtained by the methodologies, artificial intelligence-based models have great potential for improvement because more data can be incorporated into the dataset, allowing better distribution and larger edge cases, or new algorithms can be studied on the fly. Moreover, the proposed framework not only allows studying Chilean buildings, since the models are data-type agnostic; that is, they do not discriminate different types of structures. Also, to incorporate other styles or realities, it is enough to study the feature calculation mechanism and the element assignment process to be predicted. This means that new research into these applications has great potential for industry development, decreasing the design time and suggesting new solutions to the engineering team.

Further work could consider dimension estimation for the new structural elements and their impact on design considerations such as building displacement demand, building dynamic characteristics, and force distribution. The heuristic association between architectural and engineering objects also has significant room for improvement, as more geometric and topological constraints can be applied; for such task, a regressive or classification model can be trained for finding the closest wall considering the restrictions discussed in this work, such as the distance or angle similarity. The feature calculation to describe the wall's rectangles can also be improved, for example, incorporating more context and spatial relationships; other structural elements can also be considered as these objects also influence the wall's design, for example, beams or slab shafts. Finally, different ML algorithms can be studied

to obtain a more refined representation of the image prediction between an architectural and engineering domain, especially in the cases where there is a considerable semantic difference between the floor plans.

List of Terms

Adam	Adaptive Moment Estimation, an optimization algorithm. 25, 37, 42, 57
AI	Artificial Intelligence. 1, 2, 68
ANN	Artificial Neural Network. i, ii, 2–6, 17, 25, 26, 34, 57, 68
API	Application Programming Interface. 17
BN	Batch Normalization. 25, 37, 42, 57
CE	Cross Entropy. 42
cGAN	Conditional Generative Adversarial Networks. 42
CNN	Convolutional Neural Network. i, ii, 2–6, 34, 35, 37, 38, 42, 44, 47, 65, 68, 69
CNN-A	CNN-A regressive model for predicting the thickness and length of the wall rectangles. 34–38, 40, 45, 57–59, 65, 69
CNN-B	CNN-B regressive model for predicting the thickness, length, and displacement of the wall rectangles, and predicts the floor bounding box geometrical properties. 34–38, 40, 45, 57–59, 65, 69
COBYLA	Constrained Optimization BY Linear Approximation. 54
DBSCAN	Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. 52
DNA	Deoxyribonucleic acid; in this work, however, it refers to encode the essential expressions of a given domain. 2, 4, 25, 34
EA	Evolutionary Algorithms. 47
FC	Fully Connected. 25, 37, 38, 40, 42
GA	Genetic Algorithm. 47
GC	Geometric Center. 18, 19, 36
GPU	Graphics Processing Unit. 25, 37

HTML	HyperText Markup Language, the standard markup language for documents designed to be displayed in a web browser. 10
IoU	Intersection Over Union. 65, 66, 69
Javascript	Javascript is a high-level, often just-in-time compiled, and multi-paradigm general-purpose programming language. 10
Keras	Keras is an open-source software library that provides a Python interface for artificial neural networks. 25, 27, 29, 37, 38, 42, 57
L2	L2 normalization. 42
LR	Learning Rate. 25
MAE	Mean Absolute Error. 29, 57
MAPE	Mean Absolute Percentage Error. 29, 57
Matplotlib	Matplotlib is a library for creating static, animated, and interactive visualizations in Python. 14
MC	Center of Mass. 17–19, 36
ML	Machine learning. 2, 3, 6, 68, 70
MSE	Mean Square Error. 25, 37, 42, 57
OpenCV	Open Source Computer Vision Library, is a library of programming functions mainly aimed at real-time computer vision. 14, 53
OPTICS	Ordering Points To Identify the Clustering Structure. 52
PHP	PHP is a general-purpose scripting language especially suited to web development. 10
Pix2Pix	Pix2Pix is a Generative Adversarial Network, or GAN, model designed for general purpose image-to-image translation. 42, 62
PNG	Portable Network Graphics. 10
Python	Python is an interpreted, high-level and general-purpose programming language. 10, 17, 25, 37, 52, 53
RC	Reinforced Concrete. 3
ReduceLROnPlateau	Reduce learning rate on plateau Keras training monitor. 27, 29, 38, 42, 57
ReLU	Rectified Linear Unit. 25, 37, 42, 57

RENE	Regression Engineering Neural Estimator. 3, 4, 6, 17, 21, 25, 29, 34, 37, 40, 42, 68, 69
SciPy	SciPy is a free and open-source Python library used for scientific computing and technical computing. 52
Sequential	ANN-based Sequential model for predicting the thickness and length of the wall rectangles. 3, 4, 25, 27, 34, 38, 57
Shapely	Shapely is a BSD-licensed Python package for manipulation and analysis of planar geometric objects. It is based on the widely deployed GEOS (the engine of PostGIS) and JTS (from which GEOS is ported) libraries. 19, 53
Sklearn	Scikit-learn is a machine learning library for the Python programming language. 52
SL	Supervised Learning. 2
SQL	Structured Query Language. 10
SVM	In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. 3
TensorFlow	TensorFlow is an end-to-end open source platform for machine learning. 25, 37
U-Net	Fully convolutional image segmentation model. 35, 40, 42, 62
UL	Unsupervised Learning. 2
UNET	U-Net model implementation. 40, 62
UNET-XY	Image generation model based on UNet for predicting the floor plan image for each wall rectangle. 35, 40, 42, 43, 45, 47, 49, 62, 63, 65, 66, 69
UNET-Y	UNET model variation. 40, 42, 62

Bibliography

[1] National Institute of Standards and Technology (NIST), "Comparison of U.S. and Chilean Building Code Requirements and Seismic Design Practice 1985–2010," NIST GCR 12-917-18, 2012, https://nehrp.gov/pdf/nistgcr12-917-18.pdf (visited on 01/07/2021).

[2] Massone, L. M., Bonelli, P., Lagos, R., Lüders, C., Moehle, J., and Wallace, J. W., "Seismic Design and Construction Practices for R.C. Structural Wall Buildings," Earthquake Spectra, vol. 28, pp. 245–256, 2012, doi:10.1193/1.4000046.

[3] Massone, L. M., "Fundamental principles of the reinforced concrete design code changes in Chile following the Mw 8.8 earthquake in 2010," Engineering Structures, vol. 56, pp. 1335–1345, 2013, doi:10.1016/j.engstruct.2013.07.013.

[4] Lagos, R., Lafontaine, M., Bonelli, P., Boroschek, R., Guendelman, T., Massone, L. M., Saragoni, R., Rojas, F., and Yañez, F., "The quest for resilience: The Chilean practice of seismic design for reinforced concrete buildings," Earthquake Spectra, vol. 37, no. 1, pp. 26–45, 2021, doi:10.1177/8755293020970978.

[5] Yang, J., Jang, H., Kim, J., and Kim, J., "Semantic Segmentation in Architectural Floor Plans for Detecting Walls and Doors," in 2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), pp. 1–9, IEEE, 2018, doi:10.1109/CISP-BMEI.2018.8633243.

[6] Shahin, M. A., "State-of-the-art review of some artificial intelligence applications in pile foundations," Geoscience Frontiers, vol. 7, pp. 33–44, 2016, doi:10.1016/j.gsf.2014.10.002.

[7] Samuel, A. L., "Some studies in machine learning using the game of checkers," IBM Journal of Research and Development, vol. 3, pp. 210–229, 1959, doi:10.1147/rd.441.0206.

[8] Siam, A., Ezzeldin, M., and El-Dakhakhni, W., "Machine learning algorithms for structural performance classifications and predictions: Application to reinforced masonry shear walls," Structures, vol. 22, pp. 252–265, 2019, doi:10.1016/j.istruc.2019.06.017.

[9] Alpaydin, E., "Introduction to Machine Learning," 2014, https://mitpress.mit.edu/books/introduction-machine-learning-third-edition (visited on 01/07/2021).

[10] LeCun, Y., Bengio, Y., and Hinton, G., "Deep learning," Nature, vol. 521, pp. 436–444, 2015, doi:10.1038/nature14539.

[11] Goodfellow, I., Bengio, Y., and Courville, A., "Deep learning," MIT Press, 2016, https://www.deeplearningbook.org/ (visited on 01/07/2021).

[12] Khan, A., Sohail, A., Zahoora, U., and Qureshi, A. S., "A survey of the recent architectures of deep convolutional neural networks," Artificial Intelligence Review, vol. 53, pp. 5455–5516, 2020, doi:10.1007/s10462-020-09825-6.

[13] Salehi, H. and Burgueño, R., "Emerging artificial intelligence methods in structural engineering," Engineering Structures, vol. 171, pp. 170–189, 2018, doi:10.1016/j.engstruct.2018.05.084.

[14] Ye, X. W., Dong, C. Z., and Liu, T., "A Review of Machine Vision-Based Structural Health Monitoring: Methodologies and Applications," Journal of Sensors, vol. 2016, pp. 1–10, 2016, doi:10.1155/2016/7103039.

[15] Ye, X. W., Jin, T., and Yun, C. B., "A review on deep learning-based structural health monitoring of civil infrastructures," Smart Structures and Systems, vol. 24, no. 5, pp. 567–585, 2019, doi:10.12989/sss.2019.24.5.567.

[16] Flah, M., Nunez, I., Ben Chaabene, W., and Nehdi, M. L., "Machine Learning Algorithms in Civil Structural Health Monitoring: A Systematic Review," Archives of Computational Methods in Engineering, 2020, doi:10.1007/s11831-020-09471-9.

[17] Sun, L., Shang, Z., Xia, Y., Bhowmick, S., and Nagarajaiah, S., "Review of Bridge Structural Health Monitoring Aided by Big Data and Artificial Intelligence: From Condition Assessment to Damage Detection," Journal of Structural Engineering, vol. 146, 2020, doi:10.1061/(ASCE)ST.1943-541X.0002535.

[18] Sun, H., Burton, H. V., and Huang, H., "Machine learning applications for building structural design and performance assessment: State-of-the-art review," Journal of Building Engineering, vol. 33, p. 101816, 2021, doi:10.1016/j.jobe.2020.101816.

[19] Gupta, T. and Sharma, R. K., "Structural Analysis and design of buildings using neural network: A review," International journal of engineering and managment sciences, vol. 2, no. 4, pp. 216–220, 2011, http://scienceandnature.org/IJEMS/IJEMS-Vol2(4)-Oct2011/IJEMS_V2(4)6.pdf (visited on 01/07/2021).

[20] Xie, Y., Ebad Sichani, M., Padgett, J. E., and DesRoches, R., "The promise of implementing machine learning in earthquake engineering: A state-of-the-art review," Earthquake Spectra, vol. 36, pp. 1769–1801, 2020, doi:10.1177/8755293020919419.

[21] Akinosho, T. D., Oyedele, L. O., Bilal, M., Ajayi, A. O., Delgado, M. D., Akinade, O. O., and Ahmed, A. A., "Deep learning in the construction industry: A review of present status and future innovations," Journal of Building Engineering, vol. 32, p. 101827, 2020, doi:10.1016/j.jobe.2020.101827.

[22] Hong, T., Wang, Z., Luo, X., and Zhang, W., "State-of-the-art on research and applications of machine learning in the building life cycle," Energy and Buildings, vol. 212, p. 109831, 2020, doi:10.1016/j.enbuild.2020.109831.

[23] Feng, D. and Feng, M. Q., "Computer vision for SHM of civil infrastructure: From dynamic response measurement to damage detection – A review," Engineering Structures, vol. 156, pp. 105–117, 2018, doi:10.1016/j.engstruct.2017.11.018.

[24] Xu, Y. and Brownjohn, J. M. W., "Review of machine-vision based methodologies for displacement measurement in civil structures," Journal of Civil Structural Health Mon-

itoring, vol. 8, pp. 91–110, 2018, doi:10.1007/s13349-017-0261-4.

[25] Dong, C.-Z. and Catbas, F. N., "A review of computer vision–based structural health monitoring at local and global levels," Structural Health Monitoring, vol. 20, pp. 692–743, 2021, doi:10.1177/1475921720935585.

[26] Zhang, Y. and Mueller, C., "Shear wall layout optimization for conceptual design of tall buildings," Engineering Structures, vol. 140, pp. 225–240, 2017, doi:10.1016/j.engstruct.2017.02.059.

[27] Tafraout, S., Bourahla, N., Bourahla, Y., and Mebarki, A., "Automatic structural design of RC wall-slab buildings using a genetic algorithm with application in BIM environment," Automation in Construction, vol. 106, p. 102901, 2019, doi:10.1016/j.autcon.2019.102901.

[28] Lou, H., Gao, B., Jin, F., Wan, Y., and Wang, Y., "Shear wall layout optimization strategy for high-rise buildings based on conceptual design and data-driven tabu search," Computers & Structures, vol. 250, p. 106546, 2021, doi:10.1016/j.compstruc.2021.106546.

[29] Talatahari, S. and Rabiei, M., "Shear wall layout optimization of tall buildings using Quantum Charged System Search," Frontiers of Structural and Civil Engineering, vol. 14, pp. 1131–1151, 2020, doi:10.1007/s11709-020-0660-1.

[30] Bailer, C., Habtegebrial, T., Varanasi, K., and Stricker, D., "Fast feature extraction with CNNs with pooling layers," arXiv, 2018, arXiv:1805.03096.

[31] Pizarro, P. N. and Massone, L. M., "Structural design of reinforced concrete buildings based on deep neural networks," Engineering Structures, vol. 241, p. 112377, 2021, doi:10.1016/j.engstruct.2021.112377.

[32] Pizarro, P. N., Massone, L. M., Rojas, F. R., and Ruiz, R. O., "Use of convolutional networks in the conceptual structural design of shear wall buildings layout," Engineering Structures, vol. 239, p. 112311, 2021, doi:10.1016/j.engstruct.2021.112311.

[33] Chilean National Standards Institute (INN), "NCh 433: Seismic design of buildings (In Spanish)," 2012, doi:10.1061/9780784413647.ap01.

[34] Lagos, R., Küpfer, M., Lindenberg, J., Bonelli, P., Saragoni, R., Guendelman, T., Massone, L. M., Boroschek, R., and Yañez, F., "Seismic Performance of High-rise Concrete Buildings in Chile," International Journal of High-Rise Buildings, vol. 1, no. 3, pp. 181–194, 2012.

[35] Fonseca S., E., "Structuring analysis based on architectural and engineering plans for buildings with reinforced concrete walls in Chile (in Spanish)," Civil engineering thesis, Universidad de Chile, 2020, http://repositorio.uchile.cl/handle/2250/175779 (visited on 01/07/2021).

[36] Barber, C. B., Dobkin, D. P., and Huhdanpaa, H., "The Quickhull Algorithm for Convex Hulls," ACM Transactions on Mathematical Software, vol. 22, no. 4, pp. 469–483, 1996, doi:10.1145/235815.235821.

[37] Bradski, G., "The OpenCV Library," Dr Dobb's Journal of Software Tools, vol. 25, pp. 120–125, 2000, https://opencv.org/ (visited on 01/07/2021).

[38] Chew, L. P., "Constrained delaunay triangulations," in Proceedings of the 3rd Annual Symposium on Computational Geometry, SCG 1987, (New York, USA), pp. 215–222, ACM Press, 1987, doi:10.1145/41958.41981.

[39] Loze, M. K. and Saunders, R., "Two simple algorithms for constructing a two-dimensional constrained Delaunay triangulation," Applied Numerical Mathematics, vol. 11, pp. 403–418, 1993, doi:10.1016/0168-9274(93)90062-V.

[40] Gillies, S. and Others, "Shapely: manipulation and analysis of geometric objects," 2020, https://shapely.readthedocs.io/ (visited on 01/07/2021).

[41] MINVU, "D.S. N61. Building seismic design code, replacing D.S N117 of 2010 (in Spanish)," Decree of Chilean Ministry of Housing and Urbanism, Government of Chile, 2011, https://www.minvu.gob.cl/elementos-tecnicos/decretos/d-s-n-61-v-y-u-2011/ (visited on 01/07/2021).

[42] He, Z., Xie, L., Chen, X., Zhang, Y., Wang, Y., and Tian, Q., "Data augmentation revisited: Rethinking the distribution gap between clean and augmented data," arXiv, 2019, arXiv:1909.09148.

[43] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R., "Improving neural networks by preventing co-adaptation of feature detectors," arXiv, 2012, arXiv:1207.0580.

[44] Keras Team, "Keras: Deep learning for humans," 2021, https://keras.io (visited on 01/07/2021).

[45] Nair, V. and Hinton, G. E., "Rectified linear units improve Restricted Boltzmann machines," ICML 2010 - Proceedings, 27th International Conference on Machine Learning, pp. 807–814, 2010, doi:10.5555/3104322.3104425.

[46] Kingma, D. P. and Ba, J., "Adam: A Method for Stochastic Optimization," arXiv, 2014, arXiv:1412.6980.

[47] Glorot, X. and Bengio, Y., "Understanding the difficulty of training deep feedforward neural networks," Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, vol. 9, no. 9, pp. 249–256, 2010, https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf (visited on 01/07/2021).

[48] Tran-Ngoc, H., Khatir, S., De Roeck, G., Bui-Tien, T., and Abdel Wahab, M., "An efficient artificial neural network for damage detection in bridges and beam-like structures by improving training parameters using cuckoo search algorithm," Engineering Structures, vol. 199, p. 109637, 2019, doi:10.1016/j.engstruct.2019.109637.

[49] Ronneberger, O., Fischer, P., and Brox, T., "U-Net: Convolutional Networks for Biomedical Image Segmentation," Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. Lecture Notes in Computer Science, vol. 9351, pp. 234–241, 2015, doi:10.1007/978-3-319-24574-4_28.

[50] He, K., Zhang, X., Ren, S., and Sun, J., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in 2015 IEEE International Conference on Computer Vision (ICCV), pp. 1026–1034, IEEE, 2015, doi:10.1109/ICCV.2015.123.

[51] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A., "Image-to-Image Translation with Conditional Adversarial Networks," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5967–5976, IEEE, 2017, doi:10.1109/CVPR.2017.632.

[52] Mitchell, M., An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press, 1996, https://mitpress.mit.edu/books/introduction-genetic-algorithms (visited on 01/07/2021).

[53] Kicinger, R., Arciszewski, T., and Jong, K. D., "Evolutionary computation and structural design: A survey of the state-of-the-art," Computers & Structures, vol. 83, pp. 1943–1978, 2005, doi:10.1016/j.compstruc.2005.03.002.

[54] ACI, "ACI 318-19, Building Code Requirements for Structural Concrete and Commentary," American Concrete Institute, 2019, https://www.concrete.org/ (visited on 01/07/2021).

[55] Frey, B. J. and Dueck, D., "Clustering by Passing Messages Between Data Points," Science, vol. 315, pp. 972–976, 2007, doi:10.1126/science.1136800.

[56] Comaniciu, D. and Meer, P., "Mean shift: a robust approach toward feature space analysis," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, pp. 603–619, 2002, doi:10.1109/34.1000236.

[57] Ng, A. Y., Jordan, M. I., and Weiss, Y., "On spectral clustering: analysis and an algorithm," Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, pp. 849–856, 2011, doi:10.5555/2980539.2980649.

[58] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J., "OPTICS: Ordering Points To Identify the Clustering Structure," in Proceedings of the 1999 ACM SIGMOD international conference on Management of data - SIGMOD '99, (New York, New York, USA), pp. 49–60, ACM Press, 1999, doi:10.1145/304182.304187.

[59] Zhang, T., Ramakrishnan, R., and Livny, M., "BIRCH: an efficient data clustering method for very large databases," ACM SIGMOD Record, vol. 25, pp. 103–114, 1996, doi:10.1145/235968.233324.

[60] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X., "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," AAAI, no. KDD-96 Proceedings, 1996, doi:10.5555/3001460.3001507.

[61] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Waltm, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, L., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., and van Mulbregt, P., "SciPy: Fundamental Algorithms for Scientific Computing in Python," Nature Methods, vol. 17, pp. 261–272, 2020, https://www.scipy.org/ (visited on 01/07/2021).

[62] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-learn: Machine Learning

in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011, https://sklearn.org/ (visited on 01/07/2021).

[63] Powell, M. J. D., "A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation," in Advances in Optimization and Numerical Analysis, pp. 51–67, Dordrecht: Springer Netherlands, 1994, doi:10.1007/978-94-015-8330-5_4.