
GAZİ ÜNİVERSİTESİ
Gazi
1926

DECISION-MAKING PROCESS FOR HOUSE PLAN LAYOUT GENERATION THROUGH DEEP LEARNING ALGORITHMS: A HYBRID MODEL PROPOSAL

Gizem ÖZEROL ÖZMAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN ARCHITECTURE DEPARTMENT

GAZİ UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MARCH 2024

ETHICAL STATEMENT

I at this moment declare that in this thesis study, I prepared the thesis writing rules of Gazi University Graduate School of Natural and Applied Sciences;

- All data, information, and documents presented in this thesis have been obtained within the scope of academic rules and ethical conduct,
- All information, documents, assessments, and results have been presented by scientific ethical conduct and moral rules,
- All material used in this thesis that is not original to this work has been exhaustively cited and referenced,
- No change has been made in the data used,
- The work presented in this thesis is original,

or else, I admit all loss of rights to be incurred against me.

Gizem ÖZEROL ÖZMAN

29/03/2024

DECISION-MAKING PROCESS FOR HOUSE PLAN LAYOUT GENERATION THROUGH DEEP LEARNING ALGORITHMS: A HYBRID MODEL PROPOSAL

(Ph.D. Thesis)

Gizem ÖZEROL ÖZMAN

GAZİ UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

March 2024

ABSTRACT

With the advancements in technology, Artificial Intelligence (AI), and Deep Learning (DL) provide designers with new perspectives in design frameworks. In this context, designers are noted to prioritize the development of more efficient and expedited design models through these technologies. Generative Adversarial Networks (GANs) provide designers with new opportunities for exploring new ways of design. When utilizing this technology, designers prioritize their design tools, processes, evolving ideas, and methods. The heterogeneity of the design challenge and representations was explored in "How Designerly Way of Thinking". Analyzing "How AI/Machine Learns" from a theoretical perspective helps to comprehend the link between Machine Learning (ML) algorithms and human learning processes. The thesis looks into the parallels between how people learn and comprehend information and the fundamental learning mechanisms of artificial neural networks. Architects consider how they can collaborate effectively using design technology tools. The major question addressed in the thesis is, "How do AI and the Designers Collaborate?" This framework addresses fundamental inquiries regarding the architect's involvement in and interpretation of plan layouts generated by GANs. Additionally, in the object detection technique plan layouts were predicted after the generation of plan layouts through GANs to create, analyze, and label spatial plan layouts. It seeks to recognize the images produced by the ML algorithm and generate computable plan layouts for the architect's final plan assessment. When the computable plan layouts are analyzed, a hybrid decision support system method for architects is suggested, utilizing a rule-based evaluation and classification approach.

Science Code	:	80117
Key Words:	:	Machine Learning(ML), Generative Adversarial Neural Networks(GANs), Object Detection, Decision Making Process, Space Plan Layout Generation
Page Number	:	207
Supervisor	:	Prof. Dr. Semra ARSLAN SELÇUK
Co-Advisor	:	Prof. Dr. Arzu GÖNENÇ SORGUÇ

DERİN ÖĞRENME ALGORİTMALARI ARACILIĞI İLE KONUT PLANI YERLEŞİMİ İÇİN KARAR VERME SÜRECİ: HİBRİT BİR MODEL ÖNERİSİ

(Doktora Tezi)

Gizem ÖZEROL ÖZMAN

GAZİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

Mart 2024

ÖZET

Günümüzde, dijitalleşmeyle birlikte veri miktarı hızla artmakta ve bu verilerin analizi ve kullanımı giderek önem kazanmaktadır. Veri setlerini etkili bir şekilde işleyebilmek ve değerli bilgilere dönüştürebilmek için derin öğrenme algoritmaları büyük bir önem taşımaktadır. Derin Öğrenme (DL), Makine Öğrenmesinin (DL) alt dalı olarak karmaşık veri yapılarını anlamak ve öğrenmek için kullanılan Yapay Zeka (YZ) tekniklerini içermektedir. Özellikle, karar destek sistemlerinden başlayarak görüntü işleme ve üretme süreçlerine kadar birçok alanda kullanılan DL algoritmaları, mimarlık alanında da büyük ilgi görmektedir. Mimarlık, tasarımın temelinde insan algısını ve mekan ilişkilerini anlama ve bu doğrultuda çözümler üretme sürecidir. DL algoritmaları, mimarların bu süreçte daha verimli ve yaratıcı olmalarına yardımcı olabilmektedir. Özellikle, Üretken Çekişmeli Ağlar (GANs) gibi algoritmalar, tasarımcılara yeni ve farklı tasarım seçenekleri sunarak ve üretkenliklerini artırmaktadır. GAN'lar, tasarım sürecinde farklı alternatifler üretirken mimarların kriterlerine uygunluğunu değerlendirebilir ve böylece tasarımcılara önemli bir karar destek sistemi sağlanabilmektedir. Algoritmaların eğitilmesi ve sonuçların yorumlanabilirliği önem taşımaktadır. Tasarımcılar, üretilen sonuçları doğru bir şekilde yorumlayabilmeli ve mekan tasarımı üzerindeki etkilerini anlayabilmelidirler. Bu nedenle, YZ algoritmalarının çalışma prensipleri ve sonuçlarının nasıl elde edildiği konusunda açıklık ve anlaşılırlık sağlamak önemlidir. Bu bağlamda, "Yapay Zeka ve Tasarımcılar Nasıl İşbirliği Yapar?" sorusu altında YZ algoritmalarının tasarımcılarla nasıl işbirliği yapabileceği ve üretilen sonuçların nasıl yorumlanabilir ve açıklanabilir olabileceği üzerine tez çalışması geliştirilmiştir. Tasarımcıların düşünme biçimleri, bilgiye erişim yöntemleri ve tasarım araçlarını kullanma şekilleri de bu çalışmaların odak noktasıdır. Ayrıca, mekan tasarımının temel parametreleri ve değerlendirme kriterleri de araştımamızın önemli bir parçasını oluşturmuştur. Sonuç olarak, DL algoritmaları, mimarların tasarım süreçlerinde daha yaratıcı ve etkili olmalarına olanak tanırken, karar destek sistemi model önerisi olarak tasarımcı ile işbirlikli bir şekilde hibrit bir model önerisi bu tez kapsamında ortaya konulmuştur.

Bilim Kodu	:	80117
Anahtar Kelimeler	:	Makine Öğrenimi (ML), Üretken Çekişmeli Sinir Ağları (GAN'lar), Nesne Tespiti, Karar Verme Süreci, Mekansal Planı Yerleşimi Oluşturma
Sayfa Adedi	:	207
Danışman	:	Prof. Dr. Semra ARSLAN SELÇUK
Co-Advisor	:	Prof. Dr. Arzu GÖNENÇ SORGUÇ

ACKNOWLEDGEMENTS

This thesis was carried out with the contributions of "TÜBİTAK 2211-C Domestic Priority Areas Scholarship". First of all, I would like to thank TÜBİTAK for their contributions. I would also like to thank my esteemed Prof. Dr. Semra ARSLAN SELÇUK for her unwavering help during my thesis study. Additionally, I would like to thank my family for their support.

CONTENTS

Page

Page

Page

LIST OF TABLES

Table **Page**

Table **Page**

LIST OF FIGURES

Figure **Page**

Figure **Page**

Figure **Page**

Figure **Page**

Figure **Page**

SYMBOLS AND ABBREVIATIONS

The symbols and abbreviations used in this study are presented below, along with their explanations.

Symbols	**Explanations**
loss	Loss Function
[D(G(z))]	Generator Loss

Abbreviations	**Explanations**
AI	Artificial Intelligence.
BBOX	Bounding Box
CNN	Convolutional Neural Network
cGAN	Conditional Generative Adversarial Neural Network
DCGAN	Deep Convolutional Neural Network
DNN	Deep Neural Network
DL	Deep Learning
ML	Machine Learning
GANs	Generative Adversarial Neural Networks
R-CNN	Region-Based Convolutional Neural Networks
TOKI	Housing Development Administration of the Republic of Türkiye
MSE	Mean Squared Error

1. INTRODUCTION

The interpretation and thinking of designers about the information they have acquired in their experiential memory should be considered together with the whole design process. As designers' ways of thinking and experiential acquisitions change with the tools they use, and the way they develop their designs evolves with their processes, it has been observed that the knowledge learned develops with the process rather than the targeted design focus. In general, it is possible to say that designers and architects transform this part of themselves (design) as a 'representation' while realizing their designs, which they see as a 'trace' of themselves.

Today, the tools designers and architects use in the design process are also transformed into 'representations,' and each goal gradually becomes a tool. Every design that is desired to be achieved turns into a tool and forms a cycle. The idea that the designer transforms both the item he designs and the vehicle he designs into 'representation' fits into a complete framework with the act of 'designing.' When we consider Kant's (1999) treatment of the transformation of action into an end together with the act of 'designing,' it leads us to infer that action can also be transformed into a means of ‘representation’ into knowledge.

Experiential knowledge acquired through process and time is multi-dimensional. With Deleuze's (2007) statement that knowledge can be multi-dimensional and that the process, context, and conditions affect knowledge acquisition, approaches to knowledge that are disconnected from the acquired process have been incomplete. So, the knowledge designers acquire is also diversified with 'problem-based learning situations based on problems related to the design process' as expressed in the inferences of 'learning by doing' theorist Kolb (2007, 7; 2014, 67). It is known that architects, from a rational perspective, develop space constructions and space organizations based on human perception while designing space. While producing the design space, the architect, who evaluates the analyses carried out with the research and determines the parameters, creates drafts bypassing all this information through his filter and decides on the final design by choosing the most suitable design that adapts to all conditions.

With the developing digital age, the design tools architects use changed how they think, i.e., their 'learning process' and 'way of doing.' The process, which started with 2D drawing tools,

is gradually developing as 3D modeling tools, computational design tools, parametric design tools, and artificial intelligence-supported design tools by the technological development and theory of the age.

Humans turning towards themselves and taking their knowledge and thinking as a reference has also improved their decision-making process. From Aristotle's first process of knowledge acquisition, such as inductive, primary classification, decomposition, and clustering of similarities, human beings perceived nature mimetically, and with Descartes' philosophy, human beings became involved in self-oriented inquiries. They began to produce self-referential inferences (referring to reason). As humans discovered their way of learning in their own 'representation,' they found 'artificial neural networks,' the latest technological link to integrate this way of learning. The system's working principle, which is biologically 'mimetic' to the human nerve cell, is like philosophy's first knowledge acquisition processes, consistent and outputs.

The first 'singular neural networks' developed could reproduce Aristotle's simple assessment that objects belonging to the 'singular' class were easy to discriminate. However, in philosophy, in classifying objects known as 'plural,' in cases with less information, the solution could be developed by inductive methods with the inability to differentiate. This indicates that 'singular neural networks' cannot deal with the fundamental problem of machine learning: the complexity of information. Because of these results, inductive methods for machine learning have also started to be developed.

While using small amounts of data works for simple predictions, as data becomes more complex, learning becomes more complicated, and different methods and algorithms are created. For this reason, various algorithms have been developed, including decomposition, fusion, classification, inductive and deductive approaches. To better learn and train the network on multiple data, models “such as supervised, unsupervised learning, semi-supervised, reinforcement learning,” on-the-fly incremental learning capability (batch learning versus online learning), and instance-based and model-based learning have been developed.

Today, 'big data' constitutes a memory in a digitized world. However, using this data is carried out with ‘deep learning’ algorithms, a subset of machine learning.’ As the

information that needs to be trained for the networks has become more complex, the classification methods have diversified, and the data processing processes have diversified along with the learning models by taking expert opinions when necessary. Interdisciplinary deep learning methods have given rise to deep neural networks (DNN) utilized for tasks like image processing, generation, and completion, particularly in decision support systems.

Problem situation / Description of the topic

Advanced technology and modern design approaches in architecture have provided designers with an innovative perspective through the utilization of Artificial Intelligence and Deep Learning. In this context, designers have focused on creating more generative and faster design models by integrating their professional knowledge with this field (Ozerol and Semra, 2022). In the general progression of these studies, 'spatial plan layout generation' and 'the search for generative design' are the most researched topics at the interface of machine learning and architecture. This indicates a significant search point for 'deep learning algorithms' that allow the architect to create a ‘decision support system’ and enable rapid generation based on many options. GAN networks are popular for their ability to generate generative design models and are considered among the latest advancements in machine learning (ML) techniques. Rather than knowledge-based systems, designers are now looking for systems that will allow them to differentiate this knowledge and produce different knowledge. Designers have used these algorithms differently, seeking systems that can be selected and decided among many alternatives. However, due to training the networks, there may be differences between the measured values/metrics and the designers' search for the best design space based on their criteria. Consequently, a situation that will also determine the scope and hypothesis of this thesis occurs. "How Do AI and Designers Collaborate? " Although the 'interpretability' and 'explainability' of AI algorithms are developed by understanding how the algorithm works, the 'interpretability and explainability' of the generated visuals in terms of architecture make a difference. What is important here is how the results are obtained and how the expert opinion is professionally interpreted. The thesis's hypothesis, main questions, and minor sub-topic questions are expressed in the graph in Figure 1.1. Within the scope of the thesis, these questions are discussed within a theoretical framework. All results were evaluated based on these questions within the scope of the studies.

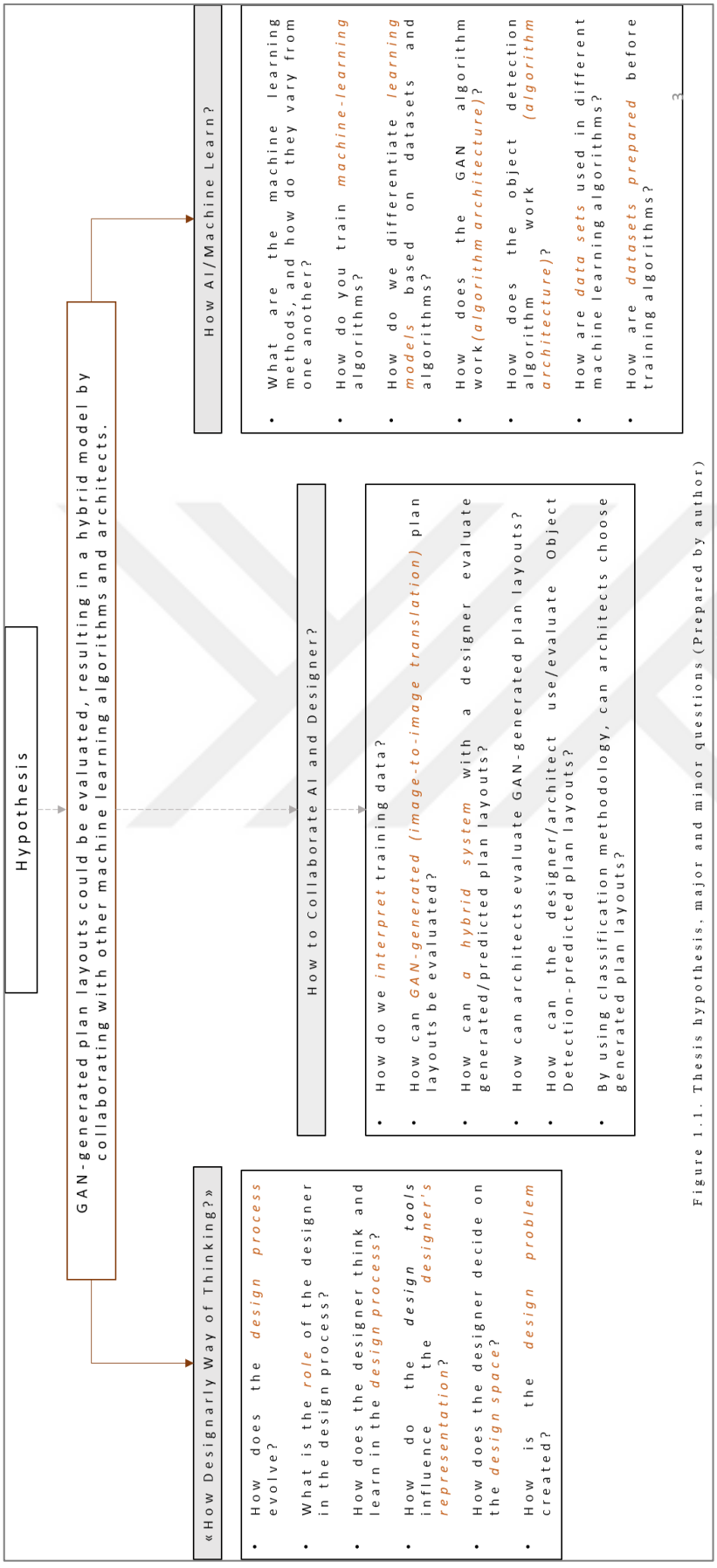


Figure 1.1. Thesis hypothesis, major and minor questions (Prepared by author)

The structure of the thesis

For this reason, situations such as how the designer thinks, how they access and interpret information, and how the way of thinking differs by using a design tool were considered. The sub-questions about how human beings access knowledge and gain diversity in knowledge acquisition with the contributions of philosophy and science to the search for knowledge starting from nature are gathered under the title " How Designerly Way of Thinking ?". The study examined how designers learn throughout the temporal cycle, how this differs from "designing," and how the tool transitions from action to aim. The design space principles and decision-making process from design to model are also covered in this study. In addition, research on housing design, which has been widely discussed in the modern period, has been carried out. This item explains why the plan layouts utilized for the study's application process were chosen and methods for determining spatial quality that affect the final evaluation process.

As a result, the transformation of the understanding of 'imitation,' one of the first learning methods of human beings, into a tool that simulates itself by turning towards itself with similar motivation reveals a different form of learning. Although artificial neural networks (ANN) are represented like the biological neurons found in the human nervous system, in their fundamental principle, they emerge with a resemblance to human learning styles and methods. These topics are explained in detail under the title "How Artificial Intelligence/Machine Learns?" by descending from general to specific, starting from the title of Artificial Intelligence (AI), followed by "ML, ANN, DNN, and DL." All topics about how algorithms work, how they learn with which data types, and how the learning results are evaluated are covered under this heading.

On the other hand, these measurable algorithms have recently generated systems where relating system limits cause mistakes, and the computer can retrain itself. A similar method is also considered in human learning. These methods include defining the problem, then reconsidering the problem, and repeatedly continuing the learning process. The fact that the results generated, which are open-ended in ML algorithms, can be completed and quantified by other systems (algorithms) at points where the system cannot complete itself is considered 'explainability and interpretability.' In ML algorithms, most classification algorithms emerge with the interpretability of the system from the outside, which is expressed by the concept

of a 'white box,' and the results are interpretable. However, in cases where it is not fully understood how the system works, the outputs need to be interpretable. XAI algorithms have been created to manage the system and comprehend its functioning. The algorithms comprise image-to-image translation techniques that operate as 'black boxes.' The emphasis changes in the last evaluation from the GANs-generated image process to the architect's interpretation and quantification of the generated image for their purposes. The subtitle "How Do AI and Designers Collaborate?" includes all these issues and research (Figure 1.2).

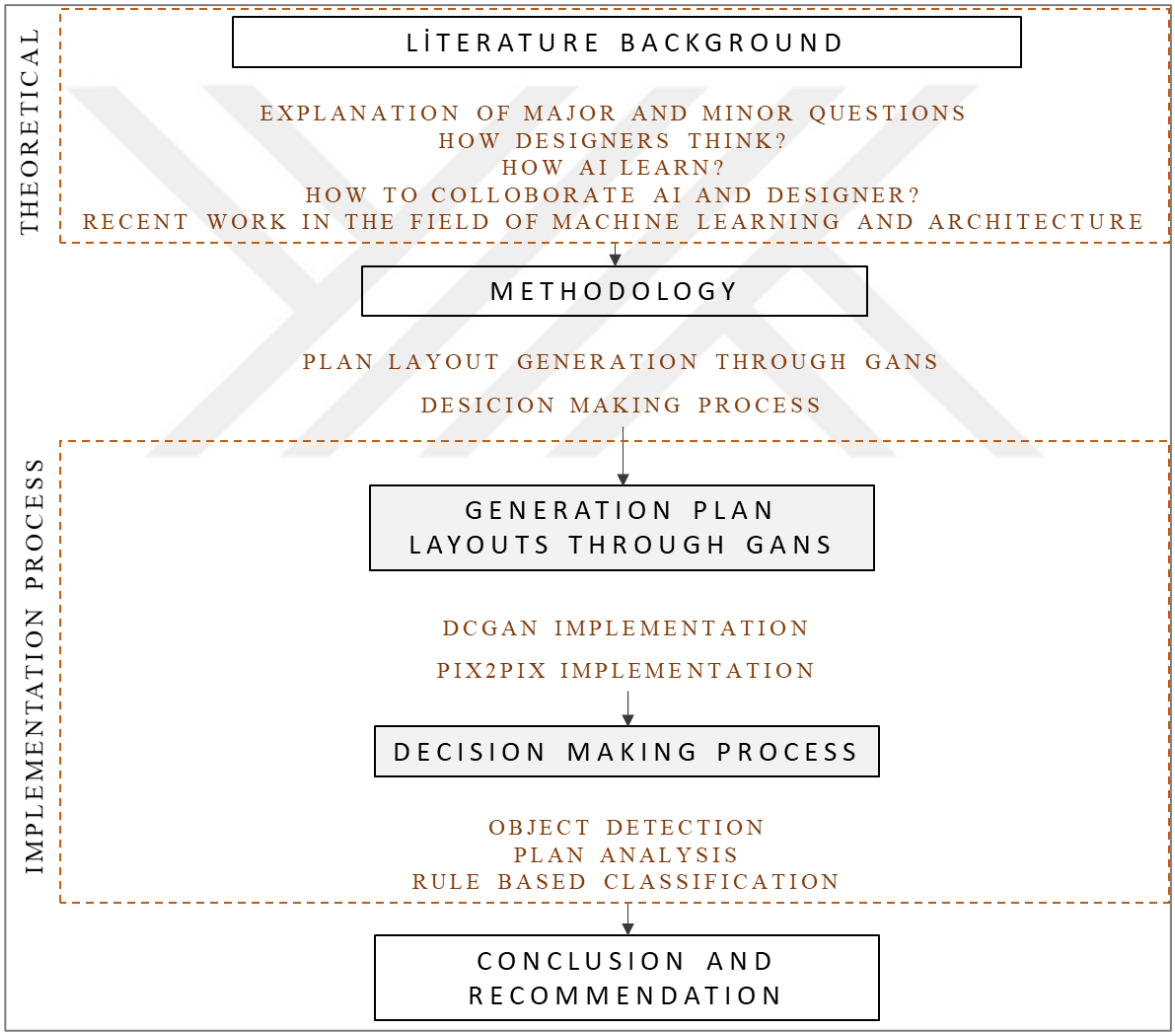


Figure 1.2. Thesis framework (prepared by author)

Finally, the literature review concluded with research at the intersection of AI and architectural design. Here, it is examined how architects generate similar algorithms discovered for the implementation process of their studies. It is investigated how design space can be generated together with GANs. Research on how datasets are created for 2D

plan layout generation methods of design space and what are considered evaluation criteria are elaborated in this part.

Following the theoretical framework, the study aims to make the results of plan layouts generated with GAN networks measurable and transform them into a system for decision-making assistance. The 'Methodology' section of the thesis discusses the application process and algorithms used. The study used DCGAN and Pix2pix (CGAN) algorithms for spatial plan layout generation. In the first process of the study, the datasets were reshaped based on different GAN algorithms. The first visual dataset was generated as input-ground truth from a part of easily accessible open-source TOKI[1] plan typologies; the second dataset was generated as input-ground truth from open plan layouts. According to the training results of the datasets, whether meaningful and analyzable spatial plan layouts are realized is evaluated and interpreted by comparing the metric values obtained from the algorithms. The variety of spatial plan layouts generated became essential, and the issue of whether another artificial intelligence system recognizes the generated image also arose.

The predicted/generated plan layouts were identified using the 'Object Detection' process in the domain of 'Computer Vision.' Here, the way artificial intelligence identifies the image it generates has gained significance. While the plan layouts defined by the object detection process as BBOX are made more definitive and used by designers, specific main criteria have been determined for the decisions they can make. In the object detection process, the dataset was organized in a way suitable for reclassification by using the Roboflow interface while determining the class of the plan layouts. Again, different object detection algorithms were tested, and comparisons determined the algorithm that gave the best prediction and prediction scores. The BBOX information of the plan layouts detected/predicted in Roboflow was converted into.json data and transferred to the Rhino/Grasshopper program, a computable design tool, to be computationally measurable. For spatial design and spatial organization to be quantifiable, the predicted images generated based on the processes of the algorithms were re-evaluated based on specific parameters. The results were re-evaluated according to criteria such as quality of space (degree of enclosure, compactness, isovist), space relationship (connectivity, interlocking space, space linked by common space), and path space relationship (partially, continually circulation). The parameters created to

[1] https://www.toki.gov.tr/ornek-konut-tipleri-ve-planlari

evaluate the spatial quality were collected as separate data to help the architect's decision support system. A rule-based classifying method was used to convert the spatial plan layout study results into a system to help make decisions. Consequently, answers to the thesis questions emphasized at the beginning were sought by evaluating the results of all the algorithms used in the application process within themselves and by considering all these findings and inferences to transform them into a decision-support model within the flow chart.

2. LITERATURE BACKGROUND

Literature research can progress in a connected manner by including thesis questions in sub-headings. First, under the title "How Designerly Way of Thinking?" this chapter questions how the architect's 'knowledge' is formed and how it changes with which tools while explaining how a human, a designer, and an architect think and how the architect participates in "designing." " How Artificial Intelligence/Machine Learns?" describes how ML uses data, learns from data, and uses algorithms with different conditions for these purposes, how they work, and how outputs are evaluated. This section covers AI, ML, and DL algorithms from general to specific. The subtitle "How Do AI and Designers Collaborate?" covers all architecture-design ML tools. One of the most researched topics in recent years is how designers and architects use Deep Learning algorithms, interpret the results, and choose ML tools.

2.1. "How Designerly Ways of Thinking?[2]"

Philosophically, the human connection with nature has evolved by comprehending nature and discerning the interconnections within it. Plato (427–347 B.C.) asserted that the understanding and comprehension of natural phenomena can be achieved through the mind, provided that they can be quantitatively measured and described using geometric shapes. Nevertheless, Aristotle (384–322 B.C.) only interpreted Plato's theory of ideas formally. Moreover, Haşlakoğlu (2022) argued that this perception was deceptive as Plato's concept aligned with the notion of form that was conceived in conjunction with its function. The field known as epistemology, which encompasses the philosophy of knowledge, is concerned with the nature and acquisition of knowledge (Çüçen 2012).

According to Bachelard (2013, p. 7), the modern era introduces the concept of knowledge being correctable, which plays a role in the formation of scientific knowledge. Because of these concrete judgments formed in the mind, it can be defined with abstract concepts. Bachelard (2013, p. 7) asserts that the concept of knowledge as "absolute truth" limits the conceptual diversity of epistemology, thereby constraining analytical thinking.

[2] İnterview with Prof. Dr. Arzu Gönenç Sorguç.

In the words of Bachelard, "modern science... is not full of things "heard, perceived, experienced, imagined" but of things "thought, designed, and technical" (2013, p. 9).

With modern philosophy, Descartes (1596–1650) focused on the human self. "This desire of Descartes led all modern philosophers to focus on the theory of knowledge and to investigate the possibilities of human knowledge, what human beings can and cannot know, and how to obtain accurate knowledge" (Çüçen, 2012). According to Plato (427–347 BC), true knowledge is the knowledge that the mind can comprehend and is independent of experimentation. Aristotle (384–322 BC), on the other hand, bases the source of knowledge on experiment and observation. According to him, true knowledge is knowledge that is acquired through the senses and can be verified by experiment. However, Descartes emphasized the importance of questioning known knowledge through "skepticism," highlighting the necessity of doubting the accuracy of knowledge and the absence of "certain knowledge."

"The human being who possesses science, knowledge, or reason either watches something, in the old sense of the word, or contemplates something, produces something, brings something into being, or finally performs an act, or behavior." Because of these characteristics, Aristotle characterizes these three types of knowledge, science, or way of thinking that human beings possess as: 1) "seeing" (theoretical, theoretical, theoretical); 2) "doing, creating, producing" (poetic, industrial); and 3) "doing an act, performing an action or behavior" (practical, practical) knowledge, science, or way of thinking, type of reason" (Arslan, 2007). While Aristotle differentiates and distinguishes between learning by doing and making and production based on practice, Plato places all kinds of making in the same place together with the productive effect. The critical point here is that the acquired knowledge is based on the relationship between the producer and the product. With Descartes' definition of man as a thinking being, man focused on himself to acquire and learn knowledge. According to Kant (1724–1804), knowledge is obtained through the consistency between the theoretically produced thought of the human mind and the experientially obtained knowledge (Kant, 1999). Kant's new concepts, theoretical thought (a priori knowledge) and experiential knowledge (a posteriori knowledge), complement each other and constitute the basis of human knowledge (Deleuze, 2007). However, from Kant's point of view, it is possible to have different results under different conditions. An important step toward the primacy of experiential knowledge was taken by Edmund Husserl (1859–1938)at

the turn of the twentieth century. Husserl rejected the idea that knowledge could exist apart from human experience (Husserl, 1997). Deleuze (2007) argues that the fact that time (or process) is not addressed by Kant suggests that knowledge acquisition is still limited. Dewey (1908) emphasized that the pragmatic formation of knowledge is based on experience. With this transformation, knowledge acquisition develops with experiential inferences. Piaget (1972), a 20th-century cognitive psychologist, draws pragmatic conclusions about the concept of knowledge based on children's periodic developmental psychology. In line with this information, it is known that cognitive development and learning are influenced by physical experiences interacting with the environment. Deleuze came up with the idea of a "rhizome" , it has multiple dimensions, and lacks a defined input and output. According to Baker (2017), the rhizome, the "root stem," has no beginning, no end, and no source. For this reason, it is stated that knowledge does not constitute a quantifiable form. This knowledge can be diversified with a variety of objects, actions, and outcomes under various circumstances.This situation demonstrates how knowledge can be diversely organized into networks. In addition to all these, Foucault (2022) stated that knowledge has different pragmatic formations for individuals and society. Goodman (1978) examined the problems of philosophy in the reconception of the inductive approach. Goodman (1978–1992) not only developed an important perspective on the relationship between combining, decomposing but also connected between linguistic images and learning with his scientific empiricist approach (Figure 2.1).

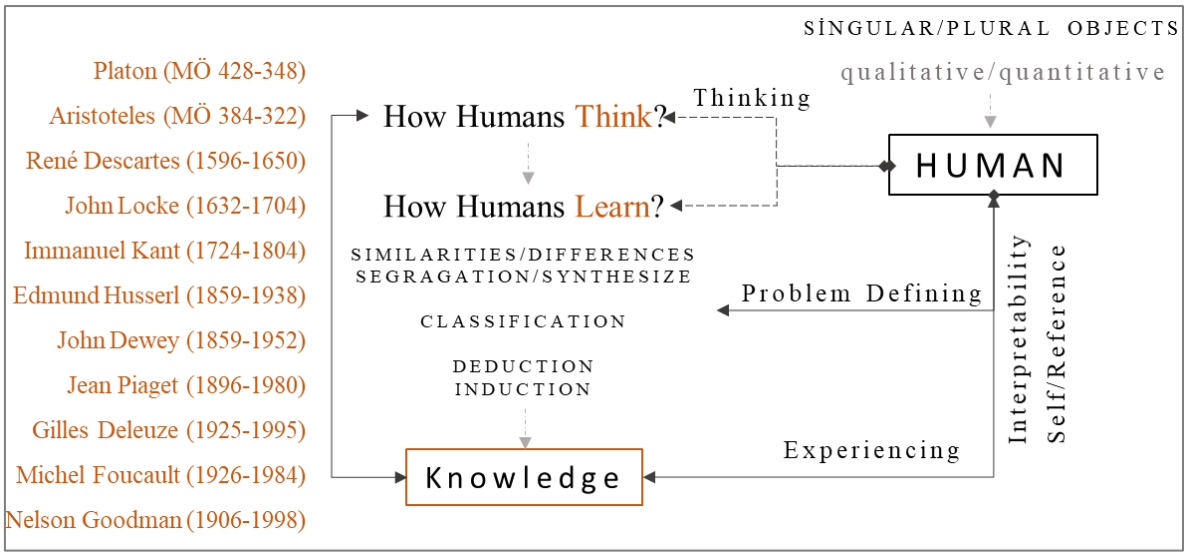


Figure 2.1. ‘Knowledge’ notion (prepared by the author)

2.1.1. Designing

Descartes' most important inference that puts the human at the center of philosophy is the expression "I think, therefore I am". With this expression, the basic requirement of human beings' existence is the act of ‘thinking’ (Descartes, 2013). According to Heidegger (2008), "thinking" requires the act of designing, and this act emerges through the decomposition and recombination of what is designed. In order to understand how designers think, it is first necessary to examine how design emerged conceptually and how the act of design relates to the designer. Cross (1982) states that we have entered a period in which what it means to be 'designing' rather than 'scientific' or 'artistic' is being discussed.

The first notional usage of the word "design" was in the sense of sign or leaving a trace in the 14th century. "Design" is both a noun and a verb in English. As a noun, it has various meanings related to purpose, plan, and structure. As a verb, "to design" means creating, planning, or shaping something. The word originates from the Latin word "signum," which means "sign" and is related to the German word "Zeichen." Originally, "design" meant "to draw a sign." (Flusser and Collars, 1995). Babadağ and Haşlakoğlu (2021b) use the term "Designo" to explain that the designer's action of "designing" represents the traces on the designed object. This term is of Italian origin and means that the creation of the work bears the personal traces of the designer.

According to Ritel (1988), the scope of designed spaces is extensive, and the knowledge used in design is diverse, covering all aspects of human experience. Ritel (1988) mentions that the designer creates concepts in their mind before making their design, and to perceive these concepts as accurate, they need tools helping to transfer their ideas such as sketches, models, and mathematical models. Cross (2006b) argues that architects find design as exploratory rather than the most appropriate solution to a given problem. Cross (2006b) defines 'design is emergent' as the design process evolves alongside the solution and problem processes. Schön (1987) examined the relationship between a designer's skill and theoretical knowledge and focused on the interaction between practice and theory in design. Schön (1987) expressed his perspective on the design using the concept of "reflective activity" and related ideas such as "reflective practice," "reflection in action," and "knowing in action." The behaviours associated with these concepts exemplify what contingency writers call "situated action" and "situated cognition." In the "reflection in action" concept, the

processing and thinking processes are usually supportive and interdependent. Engaging in experimental activities and reflecting on their results develops thinking in actions, and inquiries. Reflection is nourished by the act of doing and the results it produces. According to Schön (1983), these two factors have a reciprocal relationship; each factor influences the other and sets its limits. Reflection in action is a cognitive process in which one reflects on one's actions as they occur. Schön (1983) reveals that skilled professionals often possess knowledge that exceeds their ability to articulate it (Schön, 1983, p. 8). This exemplifies the fundamental and universally valid distinction between practical and theoretical knowledge. According to Schön (1983), design is considered one of several activities encompassing reflective practice.

Representation

"Representation is a form of representation/signalling that substitutes for the original notion, signalling or implying that concept. In other words, the presentation of the concept makes it available to perception. As Kant stated, a concept without perception is blind (Kant,1999). In light of this definition, the questions "What does design represent?" or "What representational value of design?" can be asked" (Babadağ and Haşlakoğlu, 2021b). "Design primarily represents its designer. Based on this concept, it can be said that every work represents both the individual and their ethos (identity) and the community to which the individual is connected and its ethnos (belonging) in terms of collective consciousness (Babadağ and Haşlakoğlu, 2021b)" From this context, it can be interpreted that representation does not start from a single represented and is constantly joined and derived (Figure 2.1). According to Claeys (2022), nowadays, Leon Battista Alberti's "architect's role is to design" has turned into training architects to think in representational concepts. With these concepts, architects have started to develop and create with representational tools.

However, the thought can be multiplied by adding abstract concepts through the conceived in the purpose-tool relationship. According to Kant (1999), the rule of practice is the product of reason; in this rule, action is an end. When the act of designing becomes a tool, the knowledge gained by the designer from this cycle gains importance. "As in nature, contingency and necessity are intertwined in design. The object/thing/tool owes its existence to its purpose. The function necessary for the purpose represents necessity, while the form is contingent for this function" (Babadağ and Haşlakoğlu, 2021). For designers, a tool is an

object or a way of doing that will provide a transmission so that the idea can become a real factor.

The concept of "designer's actions" examines how enhancements that serve specific purposes can be achieved by observing the behaviours of designers. These enhancements are associated with various representations of the goals or functions the designs aim to achieve (Woodbury and Burrow, 2006). Cross (2006b) defines the continuous interaction between the designer's internal and external representations as a reflective state. The statement 'Design is reflective' refers to the constant development of feedback during the sketching stage, which involves revising, rejecting, or approving forms (Cross, 2006b). "The ultimate goal is to become the means to another end once it is achieved (Adorno and Horkheimer, 2010: p. 124, 210; Babadağ and Haşlakoğlu, 2021b)." The designer's conceptual design is concretely perceived by acquiring quality through the tools. According to Schon (1992), the design process consists of movement, and this process continues as an ongoing process. The designer is responsible for reflecting on all of these relationships, including the relationship between the purpose and the result, making inferences, and identifying and analyzing problems (Figure 2.2).

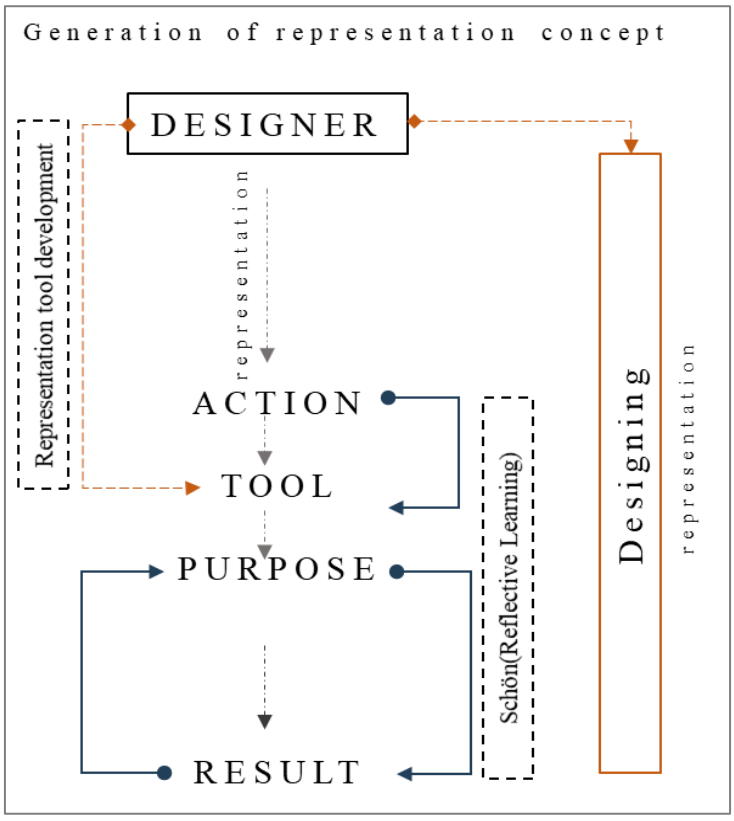


Figure 2.2. Generation of ‘representation’ concept

By setting time as an axis of scale, designers can create different target points and integrate them to the design space in physical and virtual environments (Woodbury and Burrow, 2005).

2.1.2. “Designerly ways of knowing”

While architects produce design problems, they also produce solutions by subdividing the problem into subgroups. In his article "Designerly ways of knowing," Cross (2006a) argues that designers learn from specific situations, like defining abstract ideas and expressions with "codes," to make them tangible objects. This is because designers take a solution-oriented approach to design problems. Designers rely on ‘sketching’ as it helps in discovering unexpected outcomes and surprises that may arise during the design process. According to Schön (1992), one of the distinguishing of design thinking is to have a "reflective conversation with the situation". The 'dialectics of sketching,' as described by Goldschmidt (1991), is a two-way conversation between seeing that' (reflective criticism) and seeing as' (analogical reasoning and reinterpretation of the sketch). According to Cross (2006), the designer thinks of transforming an abstract concept into a concrete concept in a sketch-like way. Alexander (1964, 1979) created 'constructive diagrams' and 'pattern language' by transforming abstract concepts into concrete ideas. Hillier and Leaman (1976) stated that an artificial 'language' is to create and learn a kind of code that transforms 'thoughts' into 'words.'"In effect, the designer learns to 'speak' a language - to perform a helpful operation between disparate domains (sounds and meanings in language, artifacts, and needs in design) through a code or system of codes that structure that connection" (Cross, 2006b).

Design problem

Eastman (1968), to solve the design problems, has acquired experiential knowledge about evaluating different alternatives and semantic expression matches by decomposing the design into 'units.' We can see that semantic expressive schemes gain importance in identifying and solving design problems. "Problem-solving processes in design operate on a knowledge structure organized nodally around concepts that define physical configurations" (Eastman, 1968). The knowledge mentioned here is determined by the relationship between the defined problems and their solutions. Chan (1989) mentions that with the definition of design problems described in the design process, "the question arises whether cognitive

phenomena (i.e., knowledge representation, control structure, and build and test) can remain unchanged in an undefined domain, or whether there are specific relationships that bring these factors together." According to Chan (1989), design problems should be defined as a tree structure, from the general to the specific. Considering that plan layouts are composed of units, to focus on a unit, it is necessary to think together with its details. In other words, as the main point narrows, the details increase.

According to the 'learning by doing' method theorist Kolb (2007, 7; 2014, 67), it is emphasized that the structural transformation of knowledge is realized through the processual transformation of experiences. The knowledge base is concretized through reflection and activation of experiences in the experiential learning process. Although it is perceived as a Kantian approach with the acquisition of knowledge in the focus of experience, it appears as a theory that moves away from Kant due to the inclusion of the process and the fact that knowledge changes and develops continuously with feedback without focusing only on the outcome of the experience. Combining the transformational character of knowledge in the experiential learning process with computational thinking is possible. Computational thinking provides a framework for thinking about and guiding the interaction of this set of expertise with the ability to create complex layouts, forms, or structures. Alexander (1977) argued that the theory of design problem-solving is not about solving each stage and problem of design separately but rather about overcoming individual issues one after the other in a flow. The "Self-Directed Learning" approach in Hmelo's (2004) PBL resource influences the process of understanding reality. It plays a supportive role in identifying and correcting the shortcomings of features, the development of assumptions. In dual evaluation situations, evaluation is done to calculate the facts and reproduce the hypothesis. It is possible to determine this situation by combining the practices of design disciplines with the analysis of the knowledge acquired through experiential learning.

The architectural design process has changed regarding design methodology with various movements and software developments provided by technology. According to Kolarevic and Klinger (2013), 20th-century architecture witnessed a period that offered the possibility of physically realizing complex geometric ideas by moving away from formal and physical boundaries. Programs developed for parametric design went beyond traditional CAD software, enabling the creation of forms with complex geometry. We can see how the role

of computational design has evolved, and the relationship between design and computation has expanded significantly. According to Claeys (2022), the designer simplifies and utilizes the complexity of nature, from rationality to his/her own experiential memory, through the decisions he/she makes. Therefore, the number of forms may theoretically be infinite, but their functionality is limited within a certain range to determine the most effective form. It is stated that the designer can solve the ambiguity in his/her perception with cognitive limitations, and these limitations help him/her to cope with the difficulties in the number of data, called "incompleteness, self-reference, ambiguity".

Chan (1989) divides the space design problem into three main components, the first, physical elements of the architectural elements, the second, the design constraints, and the third component the creation of knowledge that will guide the solution. The design criteria and constraints defined in architecture and the functional relationships (design units) create general design knowledge. Although architects and designers typically follow a strategy of decomposition reassembly, it is possible to identify different design phases where the design is temporarily finalized at a certain level of detail (Akin, 1978; 1994; Sing et al., 2012). They mentioned the need for a hierarchical representation of design knowledge to define design problems (Akin, 1978;1986; Chan, 1989) (Figure 2.3).

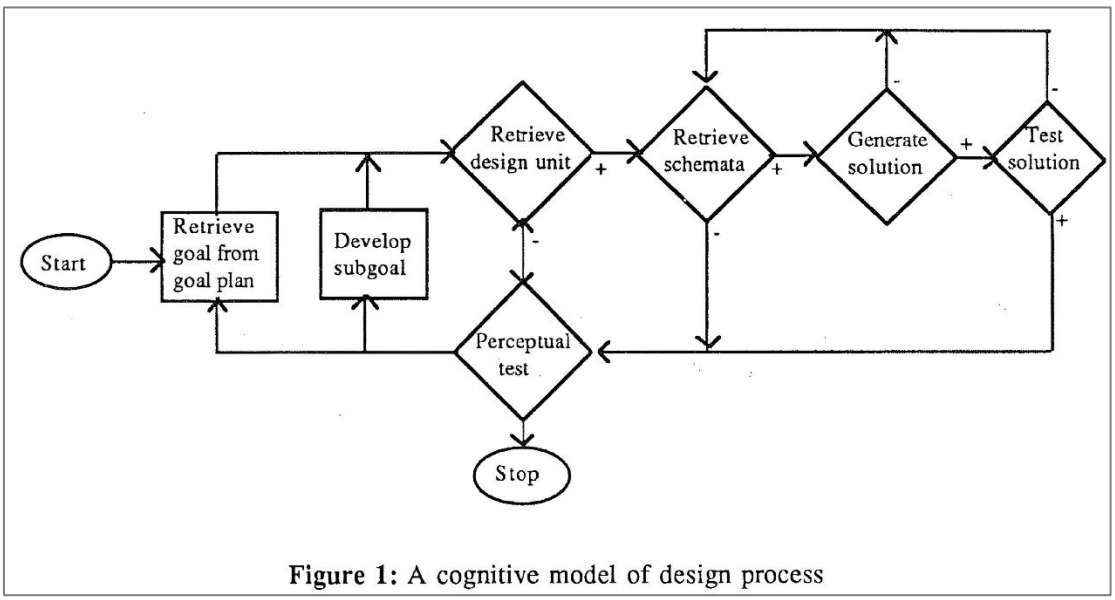


Figure 1: A cognitive model of design process

Figure 2.3. Cognitive model of design process (Akin, 1978;1986; Chan, 1989)

According to the theorists, architects move forward by creating subgoals without reaching the main goal, especially with pre-solution models, which are derived from experiential

knowledge. It is possible for them to check this pre-assessment with (visual information), which is mostly in their memory. The result must be produced and tested following all constraints and rules. Lawson (2016) refers to the generator as the primary constraint of the design. Plan layouts or multiple alternatives generated and evaluated based on these constraints. According to the degrees of the constraints expressing the conditions, sub-constraints are referred to as secondary generators.

According to Lawson (2016), Tom Markus (1969b) and Tom Maver (1970) developed a flow chart to map the terms related to the architectural design process (Figure 2.4). They focused on the fundamental concepts of analysis, synthesis, evaluation, and decision-making. Analysis is the sequencing and structuring of the problem. Synthesis is characterized by the attempt to move forward and create an answer - a solution - to the problem. Evaluation involves the critical assessment of the proposed solutions against the objectives identified in the analysis phase.

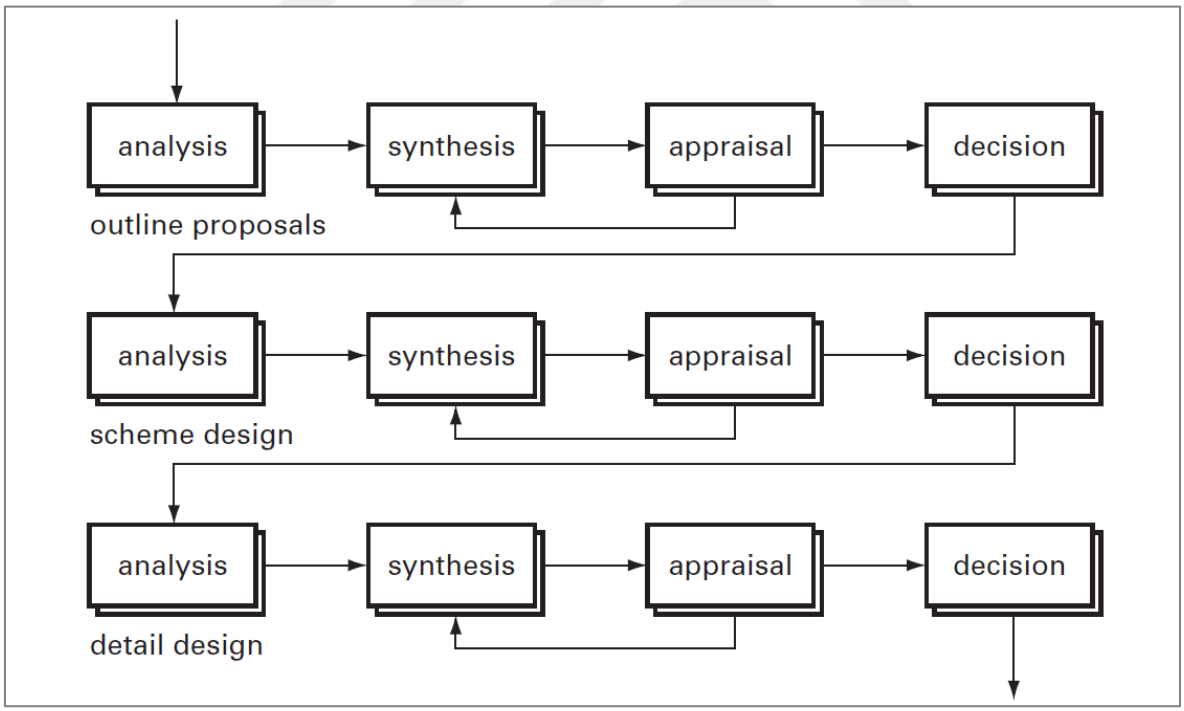


Figure 2.4. The Markus/Maver map of the design process (Lawson, 2016)

According to Oxman (2017), design tools make it easier to tackle complex problems using computational models. Oxman (2016) discusses how digital design tools have changed how designers interact with representation and their tools. The designer's new position has

shiftedto a tool generator for design environments, from the idea to the designing process (Figure 2.5).

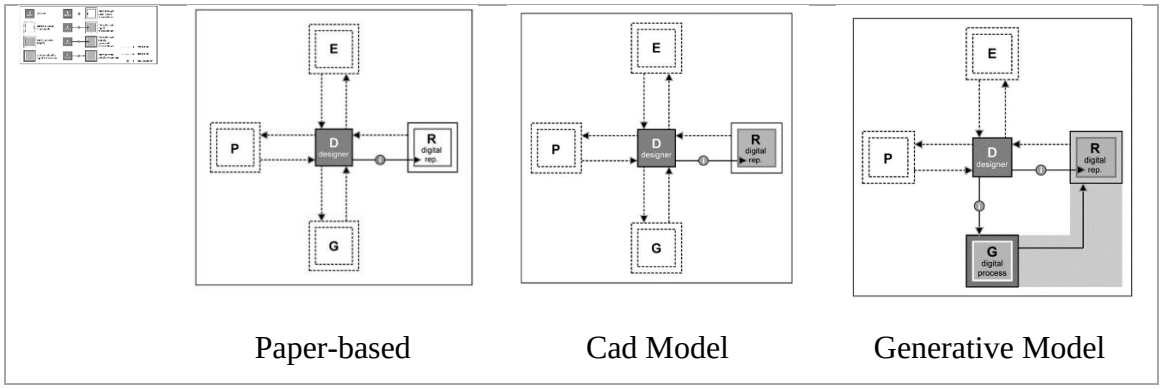


Figure 2.5. Evaluation representation models from paper-based to generative model (Oxman, 2017)

According to Woodburry and Burrow (2005), designers not only create the space they design but also the way they design it. In human-machine interaction, this approach develops in a computational way to produce solutions to all problems (Woodburry and Burrow, 2005).

Design space

Nolberg-Schulz (1979,2019) states that the 'genius loci' as a conceptual 'genius loci' from a phenomenological point of view are created by embodying the already pre-existing spirit of the place and making it a 'place.' Concrete environments come into existence by being displaced by actions and phenomena. Schulz (1979) mentioned that their measurable functional states can be expressed in spatial dimensions for places that transform with actions to take place. Architectural theorist Boudon (2015) states that the design space is the final stage that the architectural space creates to create architectural knowledge. The emphasis on design space as a perceived and experienced space is similarly interpreted by Poincare (2015). With modern mathematicians, the concepts of space and time became more complex, and many new types of space were defined in the language of mathematics. Mathematicians presented various concepts of space, and new concepts such as non-Euclidean spaces, configuration spaces, abstract space, and topology emerged against the Platonic tradition (Lefebvre, 2012). The mathematical theorist Poincare (2015) emphasizes that design space is not only geometrically defined, as Kant says, 'a priori' but can be obtained through time-dependent experience Boudon (2015) states that the object of the

architect's architectural knowledge produces the design space (that can be experienced, perceived), and that this situation can exist in the architect's mind before it turns into architectural space. According to Akin (2006), the design methods movement of the 1960s was aimed at exploring a 'problem' space that we can retrospectively call design space.

The methods of representing the design were investigated by exploring the design spaces and creating design alternatives by computer. However, for these expression methods to be computable, they were expressed with mathematical symbols. Woodbury and Burrow (2006) defined design spaces distinguishing between implicit and explicit spaces, and evaluated them according to criteria such as integrity, discrimination, alternative approaches, and the combination of exploration methods. Furthermore, their research emphasized that the design space should be located to the extent that it can provide the designer with a "compelling, effective, and feasible" solution to the design problem.

2.1.3. Designing space in architecture

According to Arslan (2007), Aristotle interpreted Plato's idea as the idea that all living and non-living things in nature have forms and that these forms, which are facts, transform and develop in such a way that they can form both themselves and their copies. However, according to Haşlakoğlu, contrary to Aristotle's interpretation of Plato's idea, it is seen that objects do not only consist of form, but that the idea expresses both the form and the function of the object (inanimate being). According to Babadağ and Haşlakoğlu (2021a), Louis Sullivan, in his explanation of 'form follows function', it is interpreted that form emerges as a reflection of the visible state of the function, apart from the fact that the function is thought first and then the form is formed. "Form connects itself to function, but this does not mean that the form is predetermined, or in other words, function does not define a single form, but offers a set of options that vary according to the context"(Babadağ and Haşlakoğlu, 2021b). The concept of space design can include many different definitions. In addition to three-dimensional geometric definitions, experientially and perceptually diversified spaces have brought about various definitions of space. According to Hillier (2007), the form-follows-function relationship in architecture often fails in mass housing production. Hillier (2007) states that one of the reasons for these failures is that modernism focuses on the form-function relationship rather than the form-and-meaning relationship. The first step in the process of designing space is to determine the issues that are associated with the design.

After that, one must combine the relative conditions with the environmental factors, and based on this, decisions are made regarding the form and the spatial relationships. The development of these decisions takes place within a considerably wider range of parameters.

Form and space relation

According to Ching (2023), architectural elements and physical and environmental contexts can be interpreted in different ways and classifications. The fact that a vertical wall is an architectural element not only expresses the transition and difference between the functions in the space but also shows that it can define the area where the circulation is constrainted. In addition, it also indicates that these transitions constitute the fluidity and continuity of the circulation. Horizontal and vertical elements in architecture are used as boundary elements in open layouts, especially to increase the flexibility of the space. However, according to Ching (2023), vertical elements are more influential than horizontal architectural elements because they clearly define spatial limitations, such as the perception of enclosure and privacy. Vertical elements (walls, columns, etc.) also act as boundary elements between the interior and exterior. The structural characteristics of these vertical elements may be structurally and materially different, but when the four vertical elements come together, they form an enclosure. In Ching's (2023) studies on the correlation between form and space, he emphasizes the role of the openings created by linear vertical elements in the plan and section in determining the quality of the space. According to Ching (2023), the light received through the openings and transpearency between these elements increases the space's quality. Within this framework, it is evident that the relationship between these components directly affects the space's quality and that this relationship determines the overall character of the space. "*Enclosure(4 planes), Compactness, Quality of Space(Openings Degree)*"(Ching,2023).

Spatial relationship

Architectural space organization formations can be gathered under more than one title. The process of assessing spaces based on their proximity and distance from one another (spatially), as well as their relations (accessibility), is referred to as space connectivity. According to Ching (2023), elements such as spaces within spaces, interlocking spaces, the relationship of spaces (both visually and physically), and the relationship of other spaces

(semi-private), including common spaces and connection through common spaces, classify variations in the organization of spaces. However, when approached holistically in space organizations, it is possible to talk about the relationship between spaces such as centralized, linear, radial, clustered, and grid organization in plural space formations and the relationship with the external environment. "*Connectivity, Space Within Space, Interlocking Spaces, Spaces Lınked By Common Spaces(Semi-Private)*"(Ching,2023).

Path space relationship

Circulations work as an essential backbone of the spaces. They provide the connection between spaces as well as the separation of spaces. While this continuous flow can sometimes be included within certain boundaries, it can also consist of undefined areas defined by the movement within the space itself. The primary process here is where the circulation starts, how it relates to spaces, and how it ends. Formally, it is possible to talk about the formations of circulation as linear, radial, spiral, grid, network, and composite both in the city and the buildings. "*Configuration of the path, Circulation Movement Partial/Continually*"(Ching,2023).

Space design in modern house architecture

Different spatial organizations have been considered in the modern period. While Adolf Loos's approach to spatial planning and the spatial relationships between the different levels of modernism offer inwardly oriented living possibilities with various spatial organizations on each floor, Le Corbusier's '*Le Plan Libre*' approaches have revealed planning approaches that give importance to architecture with their outwardly oriented fluidity and different spatial organizations on each floor (Colomina and Risselada,1988). Despite the differences in semantics and approach, interior space's organization and volumetric relationships began to be constructed interlocking. Both pioneering architects emphasized the transparency between spaces for the users and the realism of spaces with perspectives captured in the perception of space together with functional priorities. Colomina (2017) states that in Adolf Loos's modern organization of space "*Raumplan*," the differential levels and heights of spaces, which create spaces for differences in housing plans, are related to the three-dimensional circulation relationship between rooms. The architect has established a spatial relationship by hiding the circulation in the space when the rooms need to provide privacy

through their functions. With Le Corbusier, it is observed that spatial relations in the architecture of the modern period are organized with fluid circulations. In the '*Maison Domino*' approach, it is essential that certain elements, such as the column grid system, remain fixed in the design of the house and that the spatial relationships are designed according to functionality (Herby,1991). In this context, the construction of the structure and its design, which can change and transform with the other dwelling functions, has gained importance. These approaches have influenced architects in terms of flexible plan design ideas (Figure 2.6).

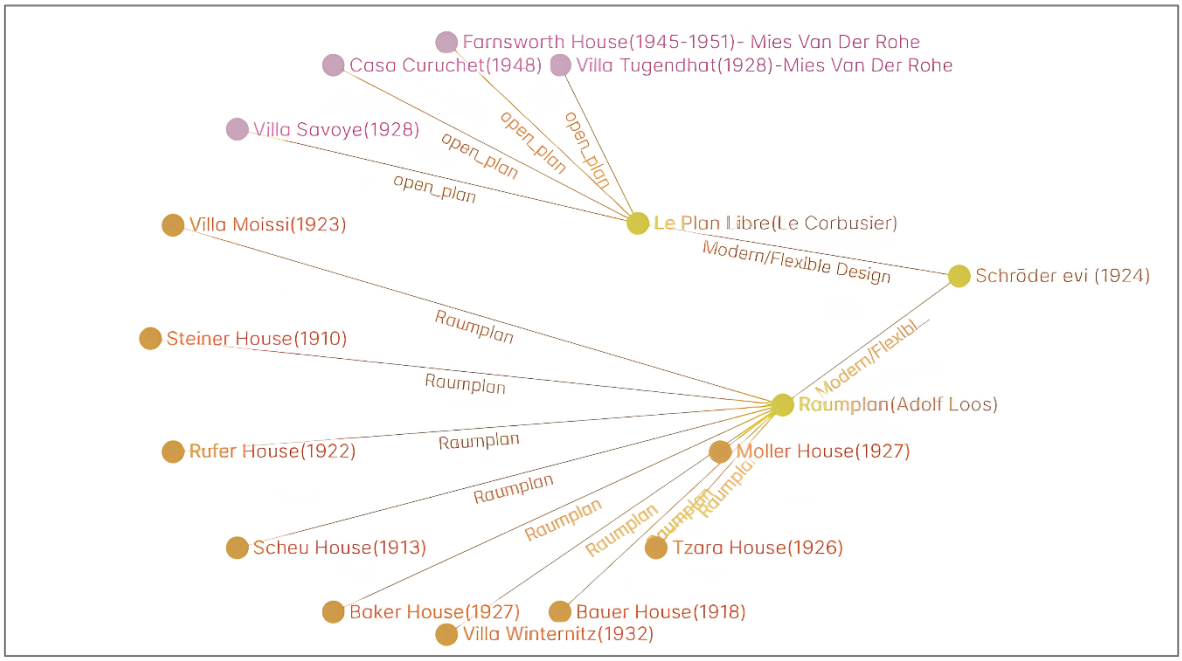


Figure 2.6. Mapping of modern house (Le Corbusier and Adolf Loos) (prepared by author)

Along with industrialization, Walter Gropius (1883-1969) discovered that flexible design models of socio-economic housing could be diversified with standardization and modularity, which was accomplished in the Aurbach House design (Schneider and Till, 2007). Buckminster Fuller, like Walter Gropius, adopted the idea of open-plan, flexibility, and diversification in housing design, with housing proposals that could be used in mass productions with industrialization, as well as fast production and individualization against the dense population, which is one of the critical problems of the modernization period. Ludwig Mies Van Der Rohe, another pioneer of the modern era, expands the boundary between inside and outside in his transparent residential design, which he demonstrates with the material and open-plan space design in Farnsworth House. He took this approach to the next level with the design of the Barcelona Pavilion. He explored the interplay between

defining and dissolving spatial boundaries and how this interplay affects our perception and understanding of space, using the building's elements vertically and horizontally (Clemence,2006). Fuller states that architectural visuals usually consist of horizontal and vertical sections that may need to be revised to fully reflect the architect's three-dimensional creative thinking (Edmondson, 2012).

Space design in TOKI mass housing design

The residential settlement in Turkey existed, with the settlements of traditional Turkish residential houses built in the cities until the 19th century. When the ownership of the land was in the hands of individuals, the structural reform of the cities was realized due to the intense migration from villages to cities with modernization and westernization. Land use was changed, and the mass housing process was initiated with "condominium". It gave a chance to individuals that could be shared by multiple users rather than a single individual. Tekeli (1979) divides Turkey's apartmentization process into two phases. Between 1930 and 1955, only wealthy citizens could build their apartment buildings and rent them to middle-income people. Individuals with higher income levels continued to live in single or duplex housing. "After the Second World War, the concept of social housing policy expanded its scope and led to a generalization of state intervention in this field" (Keleş, 1966). The other period of housing policy development was the end of World War II when land plots were divided into "condominiums" for middle-income people to meet the housing needs of the increasing city population. With the increase in the number of apartment buildings in the city, the "build and sell" logic was developed, and housing sales on a price basis were in high demand to meet the urbanization. The first initial processes of mass housing, which have come to the present day, started in these periods, that’s why the construction of more than one parcel increased rapidly. "The cooperative approach, the first examples of which were seen in 1935, became one of the leading sectors in housing production in Turkey between 1945 and 1965" (Cengizkan, 2009). "The period 1963–1967 was the period of the First Five-Year Development Plan. In this period, an understanding of housing production that is healthy, economical, and accommodates a large number of people was emerged. Luxury housing was tried to be restricted, and public housing was encouraged" (Raporu, 2009). After the 2000s, the change policies in public housing started to be addressed in terms of disaster periods. "In this plan period, since 2003, within the framework of the Emergency Action Plans of the Governments, the Housing Development Administration (TOKI) was

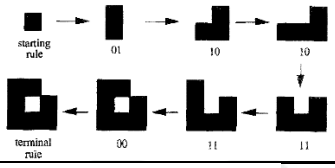
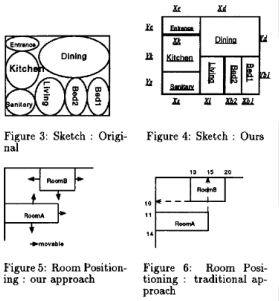
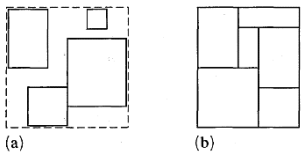
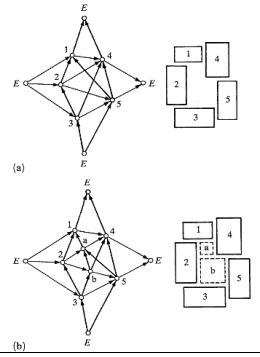
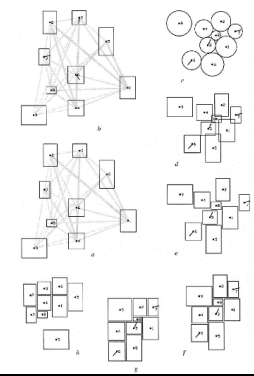
assigned the task of housing production within the scope of renovation and transformation programs" (Raporu, 2009). In 2014 and afterward, mass housing production and transformation projects increased, especially for low-income people. "Including the "Neighborhood Concept" among its basic production approaches, TOKI takes the sustainability of social solidarity as a basis for this type of production and aims to realize housing production in a way to meet social needs in this direction" (TOKI, 2020). Gradually increasing the scale of TOKI social housing and its environments, schools, mosques, and commercial units are now included in the planning of TOKI social housing and its surroundings. Today, the documents of the tenders realized by TOKI in 2020 have been collected, and the plan typologies of each city have been determined. These typologies, which are determined according to multi-user family profiles, are selected based on building population density. It has been discovered that there are 55 different typologies in total. These typologies were formed by differentiating the entrances or circulation points of the same typologies.

2.1.4. Space layout representation

Representation techniques in architecture are implemented using drawing tools. These drawings can be conveyed through technical representations comprising plans, sections, perspectives, and site plans. Various representations are created based on the method of solving the "*Design Space Problem*." According to Baykan (1991), defining the design challenge involves identifying the design constraints. According to Baykan (1991), these space layout representations are divided into "*grid-based, drawing-based, and relational representations*." The grid frame system is divided into sub-grid systems with specific ratios in grid-based representation. It can be expressed as the fragmentation of a whole volume. Drawing-based representation focuses on the shape and size of convex shapes. Relational representation refers to the current representation model known as graph-based representation. Flemming (1989) created plan layouts using a method known as packing arrangement (rectangle alignment), but he made representations using a rule-based model. According to Gero et al. (1996), space layout planning involves creating a representation that depends on the geometric arrangement of space elements and their topological relationships. Rule-based approaches, shape grammars, and optimization methods, such as evolutionary algorithms, also use this dual representation (Gero, 1996). According to Charman (1993), the problem of spatial planning is based on constraints related to spaces in

a way that depends on the location of the spaces. He stated that the most essential constraint is the geometric constraint. Similarly, in his study, Perez (1990) prioritized geometric forms as the primary constraint. Honda and Mizoguchi (1995) start by evaluating a unit of the spatial layout problem with all the constraints of the space (units) and solve the whole plan layout by evaluating it according to "constraints such as Distance, Position, Orientation, adjacency, spatial, view, and path."

Table 2.1. Space layout representation models

	Rule-based definition (Gero et.al.,1996)
	Constraint-based Automation plan layout (Honda and Mizoguchi,1995)
	Alignment of rectangles with spaces (Flemming,1989)
	Ortogonal ve marked structure (Flemming,1989)
	Topological representation (Arvin and House ,2002)

Space layout design research has been studied extensively, utilizing various artificial intelligence algorithms such as genetic algorithms, evolutionary algorithms, and other AI methodologies. One of these studies is the "LOOS" algorithm, which consists of a hierarchical generation and testing method used to generate solutions constraining the design model by establishing design limitations (Coyne, 1989; Baykan, 1991; Flemming et al., 1992). Rectangles combined according to collision detection conditions. Also, they are positioned relative to each other based on distance proximity relations (Arvin and House, 2002). These studies are frequently used in space plan layout representation methods. In a part of our study, this method was used in some plan layouts as a convergence study with the Rhino/Grasshopper-Kangaroo plug-in. Ko et al. (2023) mentioned that AI and space layout generation methods are now hybridized, and the evaluation of the produced results is based on rule-based or pattern topology recognition.

Graph representation of plan layout

Graph theory is a crucial tool in multiple scientific and technical fields. This approach is used in various disciplines as linear expressions in a mathematical framework to simulate system interactions, to describe fundamental particles and to represent geometric spaces. Harary (1969) applied graph theory to demonstrate how matrix systems might be used to evaluate the diversity of node states and the connections between these nodes. The links between the nodes can be classified as either direct or indirect. The adjacency matrix system can show the closeness of relationships, enabling simple adjustment of link length and weight between nodes (Harary, 1969). Architects begin by expressing their ideas through various diagrams, including blueprints and bubble diagrams. Grason (1971) devised a graph representation technique based on the computer program elements of space planning in architectural design. This expression method involved placing spatial arrangement principles in the directions (north, south, east, west) and developing a graphical presentation methodology that used spatial alignments. March (1971) investigated using the graph theory developed by the Swiss mathematician Euler (1707-1782). March et al. (1971) studied the transfer of shifts and neighbouring connections between spaces. This study centers on examining transitions between various states and the connections among adjacent constituents. It includes analyzing the matrix system and using a visual plan layout generation. Placing nodes in rooms follows a network structure corresponding to the spatial relationships of distance and proximity. By defining the rooms as a spatial area, the closeness

circumstances can be represented by lines based on the relationships between their dimensions and openings. March et al. (1971) study thoroughly investigated the architectural drawings created by renowned architects such as Le Corbusier and Palladio. They employed a rule-based approach to examine the proportional relationships in the plans illustrated in these drawings. Korf (1977) and Lynes (1977) pioneered the application of graph theory by utilizing bubble diagrams that are commonly employed in architecture. One can incorporate orientation connecting points both within and externally to achieve spatial design.These proportional relationships were established by duplicating the connections between the longer and shorter sides in how the spaces converge. Figure 2.7 shows a chronological mapping of graph-based representation theorists and the work produced.

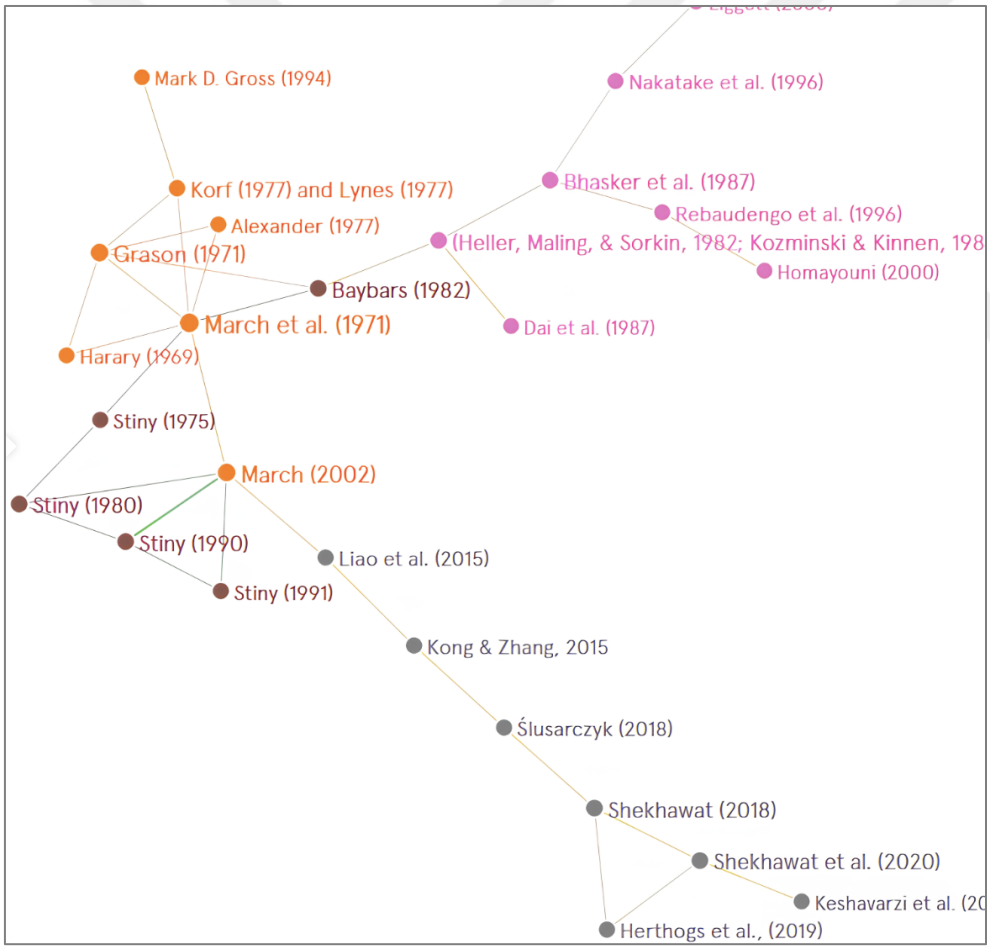


Figure 2.7. Graph plan layout representation graphical and chronological representation of the theoretical treatment of pioneering work (prepared by the author)

Generating spatial relationships enhances the city's development by producing a pattern through the connections the designed building forms with its surroundings. Alexander (1977) examined the evolution of urban structure by using graphic-based structural

formations and pattern analysis on cities. Gross (1994) devised his design methodology using computer software that analyzes hand-drawn sketches and identifies spatial relationships illustrated in diagrams. In the second phase, he studied these sketches using established frameworks and visually represented the images of these frameworks and their spatial connections (Gross, 1995) (Figure 2.8).

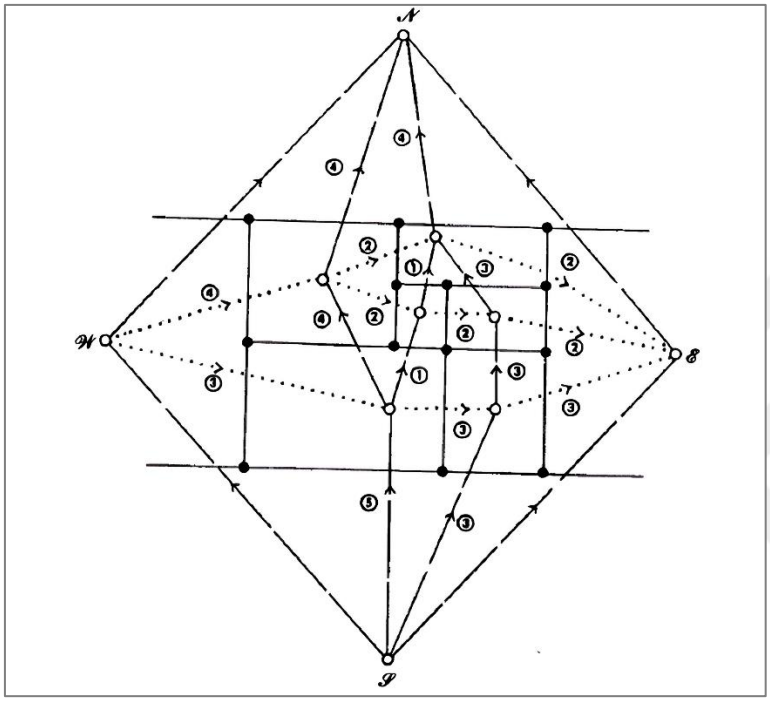


Figure 2.8. Illustrates the spatial relations based on the proximity of distances by linking to axes in the nodes of North, South, East, and West (Gross, 1995)

In his work published, March (2002) focused on Euclidean spatial transformations and demonstrated spatial organizations using symmetry axes, rotations, and reflections. He used mapping techniques and graph layout representations to illustrate his findings. This rule-based project is derived from the framework of Boolean algebra, which serves as the foundation of shape grammars. Baybars (1982) proposed an alternative viewpoint regarding the relationships among plan and graph theory (Baybars, 1982). The impact of features such as internal courtyards and circulation on the modification of the spatial, graphic plan layout was demonstrated by contrasting it with the method of graph mapping of spaces based ontheir neighborhood relationships in the plans (Figure 2.9). Establishing a connection between the locations was facilitated by the interaction between the concepts of 'present' and 'absent.'

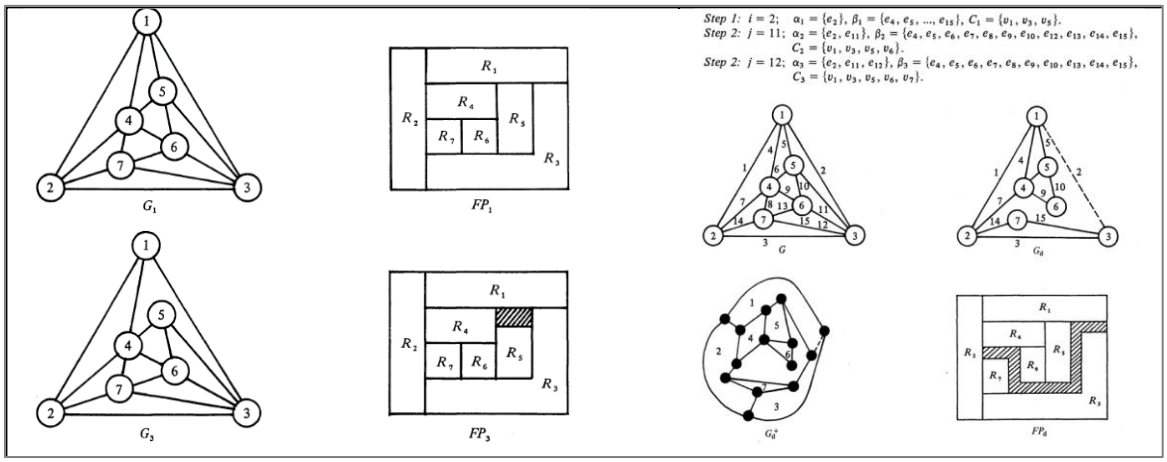


Figure 2.9. The circulation area reorganizes the neighborhood relations between the spaces

The researchers employed the Rectangular dualization approach (Heller et al., 1982; Kozminski and Kinnen, 1984) to generate a dual graph layout for mapping their plans. The researchers attempted to organize the spaces into subcategories and groups by relating the units of length and width and using area computations (Heller et al., 1982). Bhasker et al. (1987) introduced a planar triangulated graph (PTG), a method to illustrate graphs using a rectangular plane. In this representation, dashed lines connect nodes to indicate spaces without transition between adjacent spaces, while solid lines indicate spatial connections with clear transitions. They also proposed the path digraph (PDG) method representing directed transitions between nodes (Figure 2.10). Utilizing the building block layout software pioneered at the University of Berkeley, Dai et al. (1987) positioned solid blocks onto the graphical layout that was initially constructed using blocks, emphasizing the topological allignment.

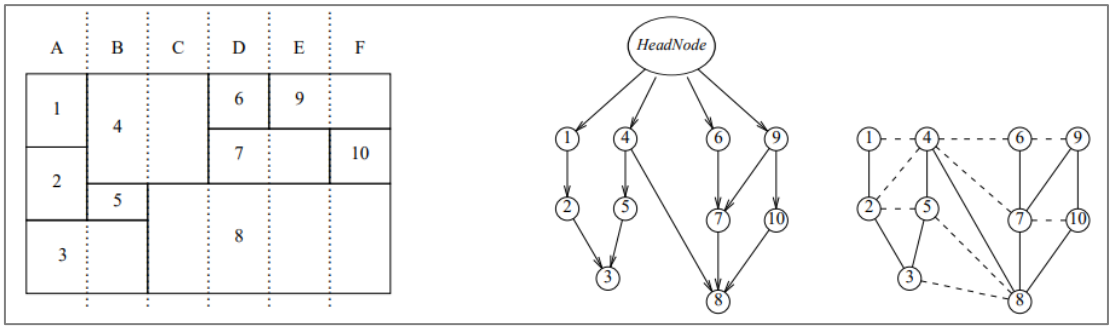


Figure 2.10. Linear and non-linear relationships between spaces expressed as PDG and PTG graphs with linear changes (Bhasker et al., 1987)

In their study, Nakatake et al. (1996) attempted to include modular room forms with specific volumes into a grid system that they separated using the BSG approach (Nakatake et al.,

1996). Given that this system is implemented as a package within bounds, the adjacency graph system was employed to establish the proximity relationships between the rooms in the grid (Figure 2.11). Rebaudengo et al. (1996) and other researchers utilized a hybrid adjacency graph layout system in the dataset to enhance the fitness function used for optimization in producing plans using genetic algorithms.

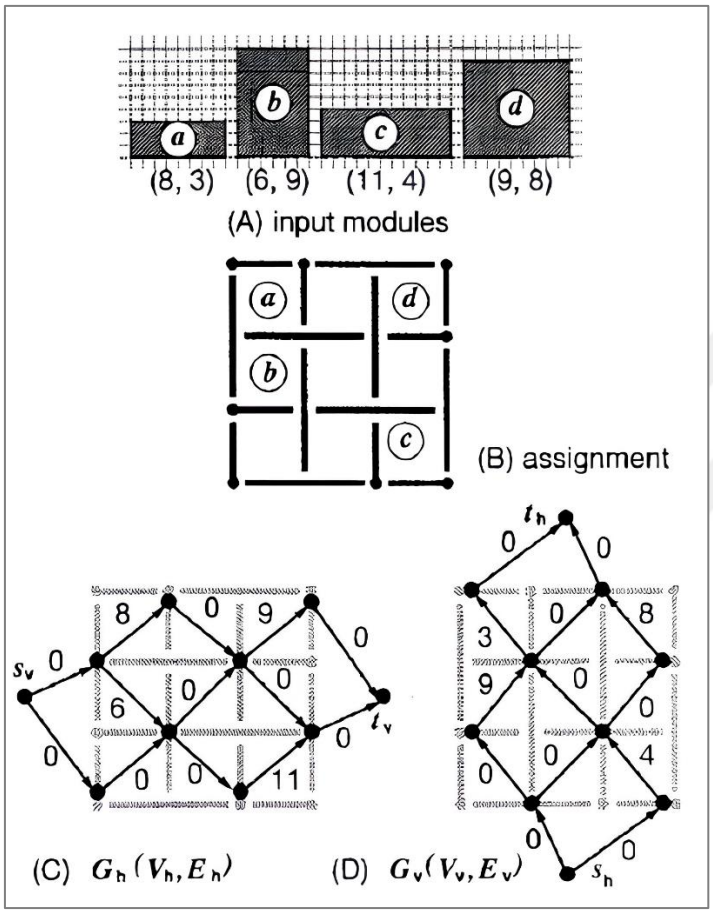


Figure 2.11. Relationships of modular room systems placed with gridal adjacency graph layout, (a) input modules, (b) assignment, (c) and (d) nodes (Nakatake et al.,1996)

Liggett (2000) states that when analyzing the development of plan layout systems from the past to the 2000s, graph representations were referred to as the "dual of a planar graph" with block plan layouts. This was done in order to better understand the evolution of these systems. Initially, a hierarchical tree graph expression was utilized in order to represent the connections that existed between the spaces. In spite of this, other types of rectangular block plans were eventually incorporated (Figure 2.12).

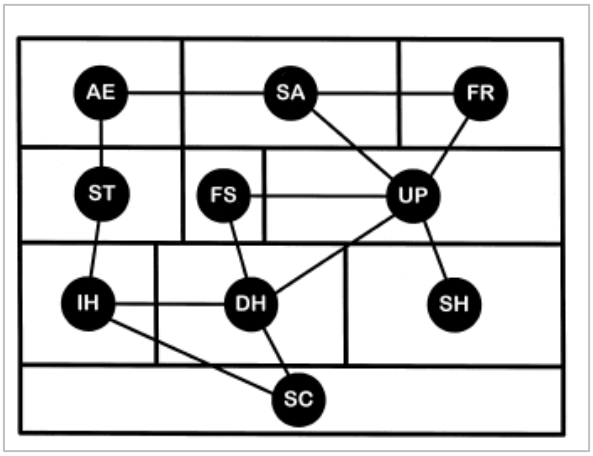


Figure 2.12. Rectangular block plan types (Liggett, 2000)

Ślusarczyk (2018) devised the "HL-graphs" technique, a hierarchical graphical arrangement system for organizing local architectural plan layouts. The following study focused on performing tests implementing Rectangular Floor Plan (RFP) pattern configurations based on the block plan layout method. Within the scope of his research, Shekhawat (2018) expanded upon the previous investigation by incorporating RFP with a variety of theorems and assessing its performance in comparison to graphical layout methodologies. Using a technique known as SAGA, Herthogs et al. (2019) investigated the relationship between the flexibility and adaptability characteristics of rooms in the process of creating plans. Figure 2.13 illustrates the results of their investigation.

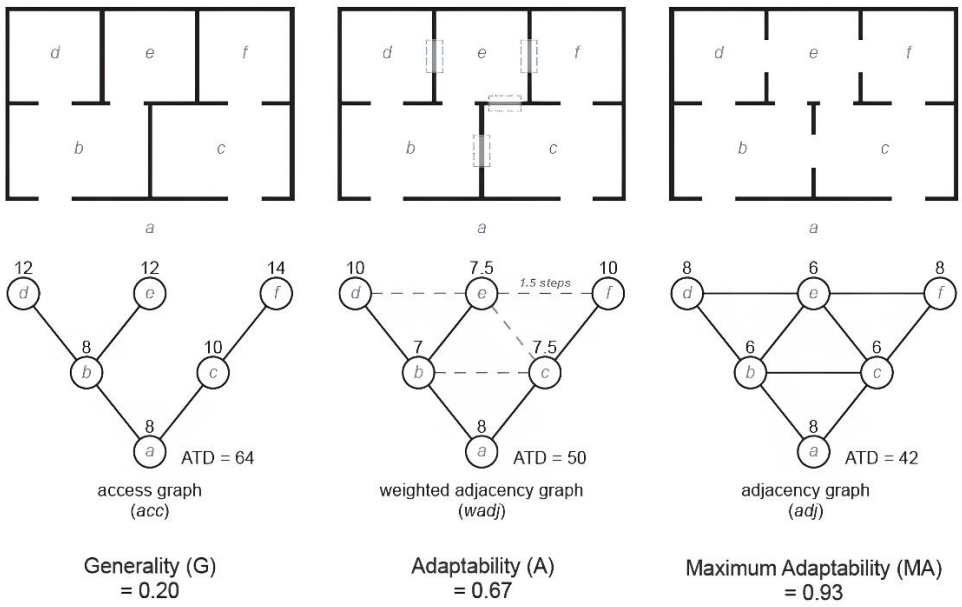


Figure 2.13. The way the proximity relations between spaces are shown according to the hierarchical structuring of SAGA (Herthogs et al., 2019)

2.2. How Artificial Intelligence/Machine Learns?

In Aristotle's philosophy, the fact that Aristotle states that in materialistic approaches, the complexity of everything in nature and the universe is characterized as ‘monist or pluralist’ by a superior force called reason and intelligence creates a world of duality and opposition. According to Atademir (1974), Aristotle's theory that the variability of everything in the universe can be predicted is based on a comparison of the theory that existence does not change with the search for truth (reality) and the fact that the essence of existence is not included in the flow. Aristotle's division of the universe of objects into "perceptible by the senses(sensible)" and "imperceptible by the senses(intangible)" and the fact that knowledge can be obtained through the senses with the thought coming from the object makes an essential contribution to the process of accessing knowledge by human perception today. Many theorists on spatial perception and spatial cognition with the senses have discussed the relationship of three-dimensional volumes that are perceptible by senses through this basic philosophy of Aristotle.

According to Aristotle, mimesis emerges by developing the first signs of human learning with this fundamental emotion. According to Arslan (2007), Aristotle claimed that the instinct of imitation is an innate ability of human beings, that this ability has a feature that distinguishes human beings from other animals, and that human beings acquire their first knowledge through this ability. The knowledge that comes with learning is reinforced by the human understanding of imitation and the understanding of simulating oneself. At the last point of today's technology, just like the transformations in works of art and the progress in philosophy, humans have started to imitate their environment by simulating (painting, using virtual reality). However, in parallel with the developing technology and theories, at the final point, when human beings imitate themselves and focus on themselves, they have moved to a dimension such as artificial intelligence. According to Leich (2022), Artificial intelligence (AI) is an essential tool to understand how architects think and how they train their thoughts.

AI and artificial neural networks (ANNs) were developed by studying and replicating the human brain and cognitive system processes and mimicking the operations of biological nervous systems and neuron networks. However, the idea that a machine's access to knowledge is just like a human's can be understood with an Aristotelian approach. In the universe of the senses, the information transferred from the senses to the mind and

intelligence allows a person to interpret the objects they see. Instead of relying solely on accessing information, neural networks also consider the level of certainty associated with that information. This means that in training studies, each neuron within the network finds not just a single factor but also its ability to distinguish and predict individual objects reliably. As "big data" becomes digitally possible, artificial intelligence is equipped with different algorithms to parse complex information in a world of complex information. In this way, the ability to distinguish, classify, and interpret objects in intricate patterns has evolved into "deep learning."

With the first observations of nature, humans began to decompose complex ideas. This parsing formed Aristotle's conception of living and non-living things in nature and led to the development of the taxonomy method known as "empirical classification" or "artificial classification." In nature, living and non-living things are classified based on their similarities and differences. In the machine world, perceiving objects from images is also based on their analogous similarities and differences. However, there are numerous factors that influence these classifications and the detection of similarities. An adequate and identifiable data source is required for the machine to perform the classification. In cases where the data is incomplete and poorly defined, neural networks cannot develop accurate predictions. Arslan (2007) argues that Aristotle's lack of knowledge in classification leads to the inability of human beings to define a clear definition of knowledge and, therefore, to classify knowledge. According to Arslan, Aristotle used the "empirical world" concept instead of defining it precisely to define knowledge as “not classified.” In this empirical world, definitions can be made due to the knowledge obtained through inductive approaches. Inductive learning occurs in deep learning algorithms through various methods “such as supervised, unsupervised, semi-supervised, reinforcement, online (incremental) learning capabilities, instance-based, and model-based learning”. These methods are developed by considering the learning process of deep neural networks, including how the trained data is processed and prepared and how it interacts with the network. Heynen (2011) defined *novelty* as "fundamental undecidability" in his interpretation of Lacoue-Labarthe's interpretation of mimesis (imitation) in contemporary theory, giving it a new meaning. The fact that the relationship between the imitated and the imitator cannot be clearly defined and that the "similarities and differences" cannot be determined with the representation of technology today becomes significant in art and architecture. According to Bolojan (2022), the results of AI algorithms, basically based on the human thought system, are innovative

and creative products that are different from human perception. This is different from trying to imitate human intelligence's perception of creative thinking (Figure 2.14).

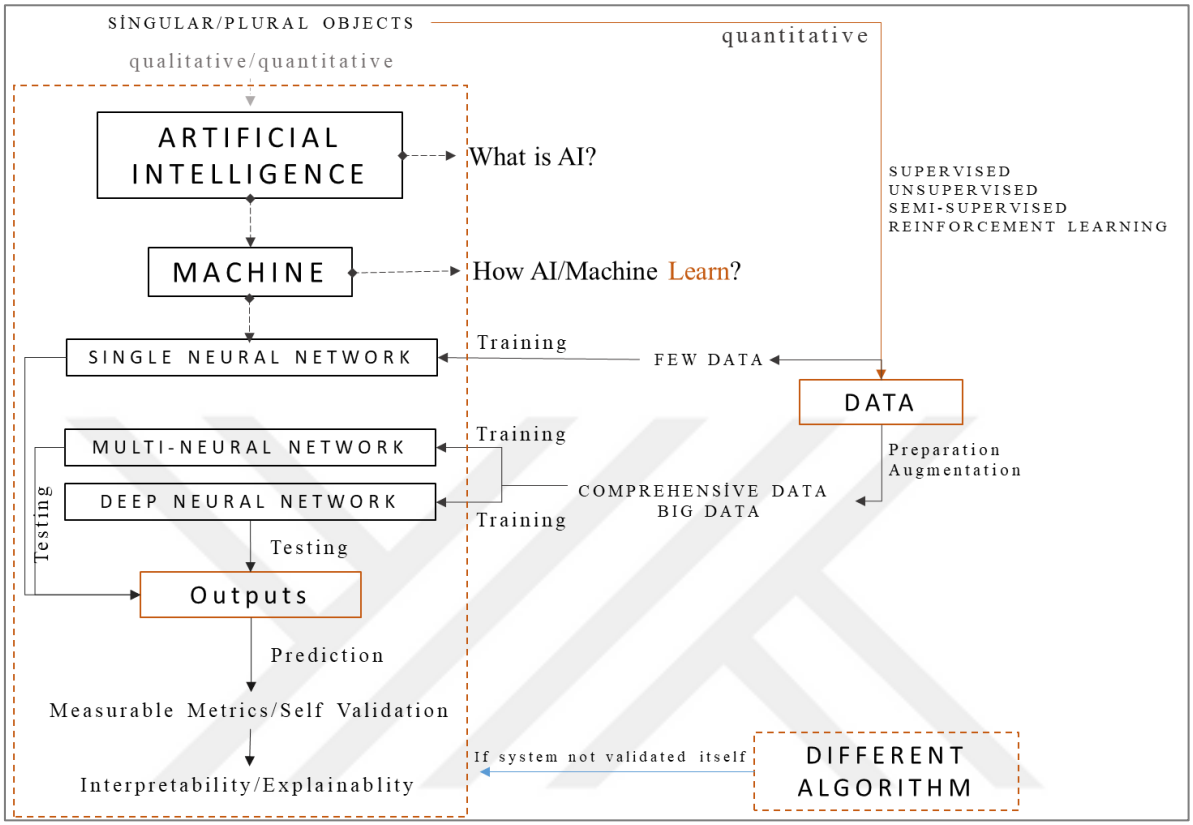


Figure 2.14. “How does AI/machine learns?” graphical explanation (prepared by author)

Similarly, the last point reached with generative artificial intelligence algorithms has gained importance in discussing the "real" or "fake" status of many products reproduced by imitation. The generated images are evaluated based on the "similarities or differences" of the products with the images trained to the networks first, and their “real” or “fake” status is determined.

2.2.1. Artificial intelligence (AI)

Human intelligence can comprehend complex thoughts, maintain environmental adaptation, gain experience by living, and solve problems through various thinking methods. This cognitive and behavioural process has been designed in computer language and implemented as a technological advance called "artificial intelligence." The logic of the computer's functioning can be associated with physical symbols and patterns as a simulation of the human brain. The Dartmouth conference in 1955 featured the work of computer scientists

John McCarthy and Claude Shannon on Turing machine intelligence (Penman, 2018). The conference covered topics such as "the use of machine language, abstractions and concept formation, solving human problems, and machine self-improvement" (Penman, 2018). Elements of AI are presented in Figure 2.15.

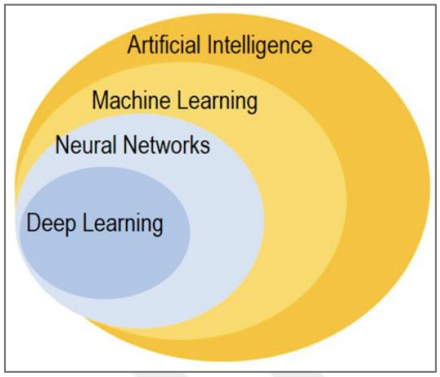


Figure 2.15. Artificial intelligence element's graphic composition (prepared by author)

AI technologies are evolving today across numerous industries. Some application areas of AI include expert systems, robotics, imitating human senses, virtual reality, massive data processing and data mining, face and speech recognition, and concluding available data (Mukhamediev et. al, 2022) (Figure 2.16).

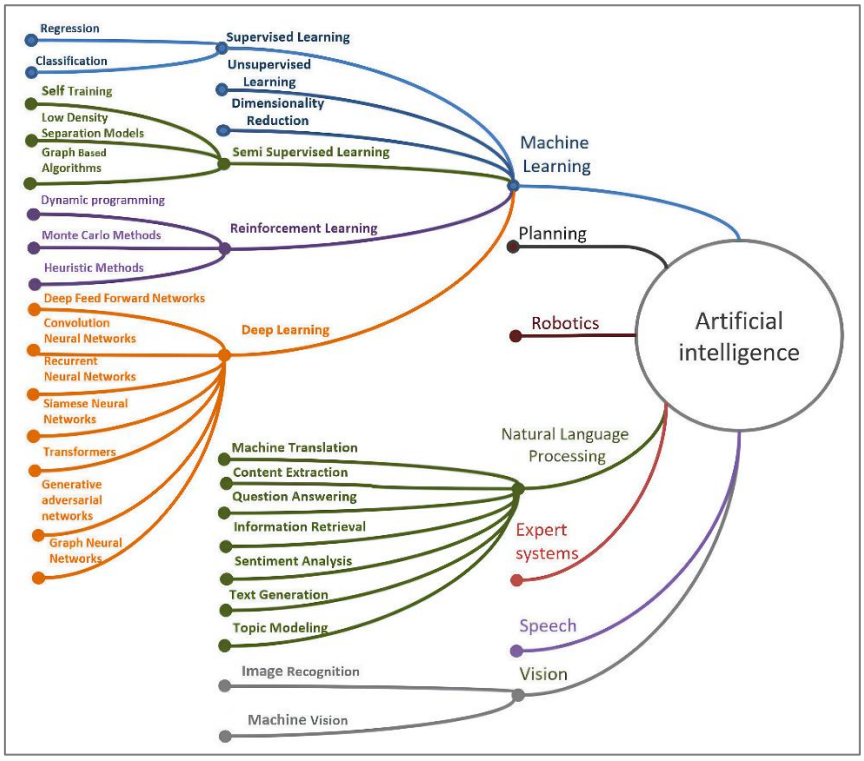


Figure 2.16. AI technologies (Mukhamediev et. al, 2022)

AI algorithms vary in software and hardware based on data types and learning models.

2.2.2. Machine learning

Géron (2017) describes Machine Learning as teaching machines to learn from data. In artificial intelligence, systems can autonomously acquire knowledge, enhancing their ability to adapt and evolve through experiences. Arthur Samuel (1959) defines machine learning as the discipline that enables machines to learn without being specifically programmed. The learning journey begins with significant and insignificant data, guiding the system to unearth hidden insights invisible to the human eye and make more informed decisions in the future. A noteworthy aspect of machine learning methods is their dependence on data samples characterized by specific values, guiding the learning process accordingly. Hence, the meticulous documentation of research data in a sample set format becomes imperative, capturing the known qualifications and associated values. This systematic approach empowers the use of available data to elucidate uncertainties through new examples, ensuring a practical and effective application within the framework of the machine learning discipline. Géron (2017) classifies machine learning into modes, such as "supervised, unsupervised, semi-supervised, and reinforcement learning." This classification underscores the adaptability of machine learning algorithms to different learning scenarios. Supervised, unsupervised, semi-supervised, and reinforcement learning methods train artificial neural networks. The learning rules are defined below.

Supervised learning: This process provides a model with input data and its corresponding desired output. This generates the model's answer, which we then compare to the desired one. The key lies in comparing and adjusting: we modify the model's internal weights, essentially fine-tuning its understanding, to increase the likelihood of it providing the correct response in similar situations in the future (Fyfe, 1996). All the data, labeled as inputs and outputs, becomes the teaching material. A supervisor is found to help the ANN learn, and all the data to be taught to the network are defined as inputs/outputs. It produces outputs as determined by the supervisor. The two most common supervised learning applications are classification and regression (Kim, 2017).

Unsupervised learning: Since no supervisor can help the network learn, the network parses the given input values independently. The network reorganizes the structure of this data by

itself. The network is expected to learn how to determine the relationships between each neuron and the relationships between them on its own.

Reinforcement learning: Input data is provided to the network. During the network training, if incorrect results occur, the network is retrained until it obtains the correct results. In the learning-based process, there is no continuous supervision during the network training and data processing. Based on the results, the supervisor makes decisions and updates the parameters of the network until the correct results are obtained Miraftabzadeh et.al (2021) (Figure 2.17).

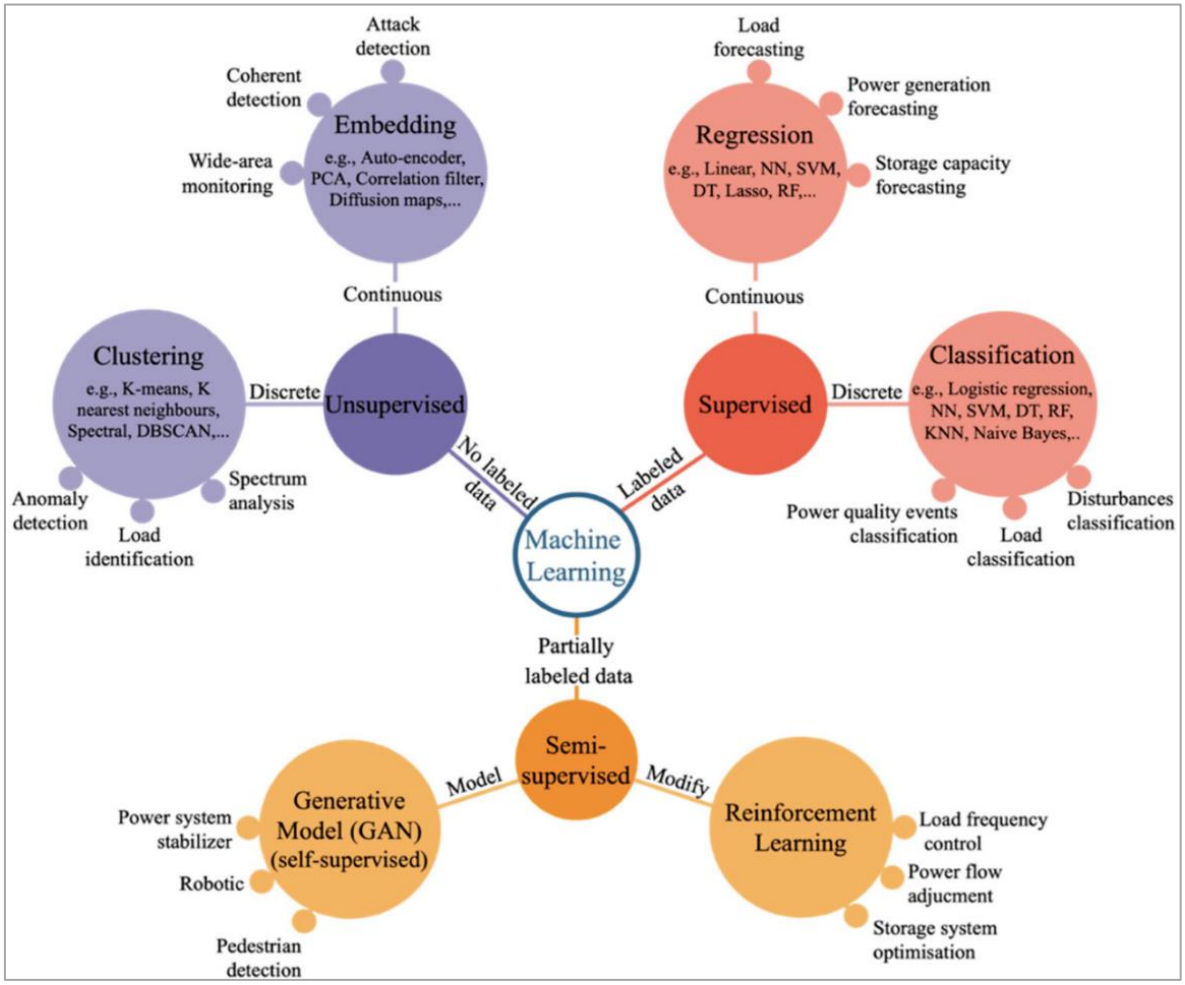


Figure 2.17. Learning models (Miraftabzadeh et al.,2021)

Machine learning methods are notable for using data samples defined by specific values, shaping the learning process accordingly. As outlined by Friedman et al.(2006), the methodology depicted in Figure 2.18 provides a framework for labeling data, with its structure dependent on either supervised or unsupervised learning. The primary method

aligns with the overarching objective, selecting sample learning techniques based on varying variables.

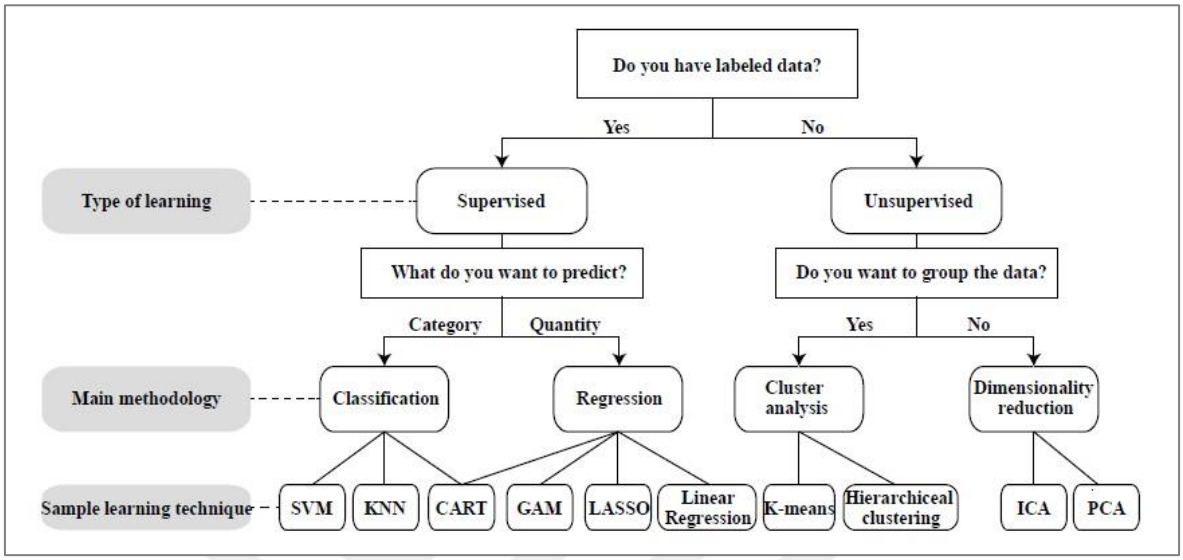


Figure 2.18. Methodology of ANN, CNN, GAN

The meticulous recording of research data in a sample set format, including known qualifications and corresponding values, is essential. This approach facilitates the practical and effective utilization of available data to address ambiguous points with new examples within the machine learning framework. The neural network's evolution began with the single-layer model, progressed through the started with a shallow neural network, and progressed to the complexity of DNN. Kim (2017) underlines that the critical role of deep neural networks was recognized in the middle of the 2000s, around a dozen years after the emergence of shallow neural networks.

Artificial neural networks (ANN)

Within machine learning, ANN emerged as a branch inspired by the intricate functioning of biological nerve cells. Fyfe (2005) contextualizes ANN within a set of innovative techniques designed to replicate the information processing networks found in biology, positing that our human expertise is intricately tied to the underlying hardware of our brains. When new data is input into a network trained with learning rules determined by specific datasets, the result is derived from interpreting previously assimilated information. The perceptron, invented by Rosenblatt in 1958 for pattern classification, consists of interconnected neurons communicating through weighted connections(w). The vector used for input is denoted as

(x), and the architecture of the network is determined by a specific math equation (Figure 2.19).

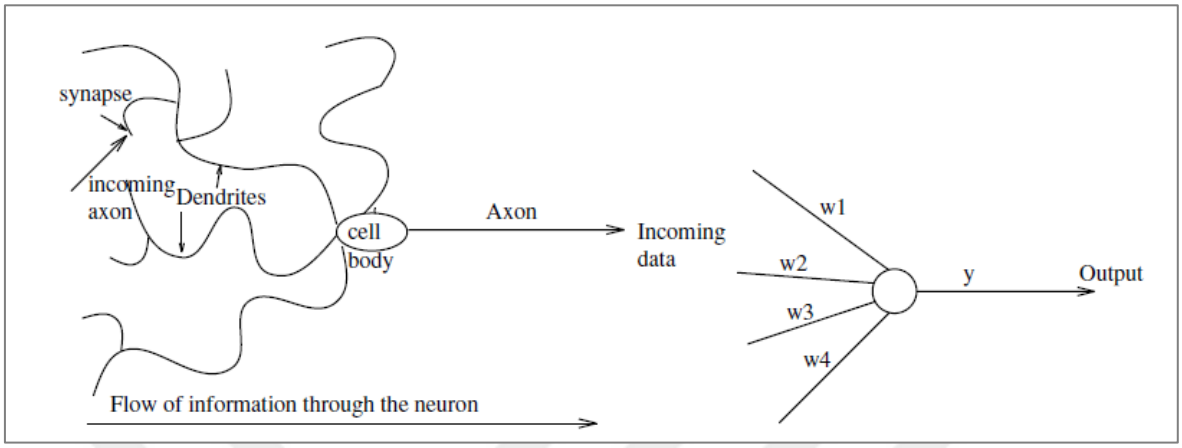


Figure 2.19. On the left side, there is a schematic illustrating an actual neuron, whereas on the right side, there is a diagram of an artificial neuron. The weights of the artificial neuron are used to represent the synaptic efficiency

The architecture of ANN models, demonstrated in Figure 2.20, encompasses a spectrum from single-layer to multi-layer configurations and extends to the complexity of DL models. Although learning rules for these networks have been briefly explained, a more detailed description follows.

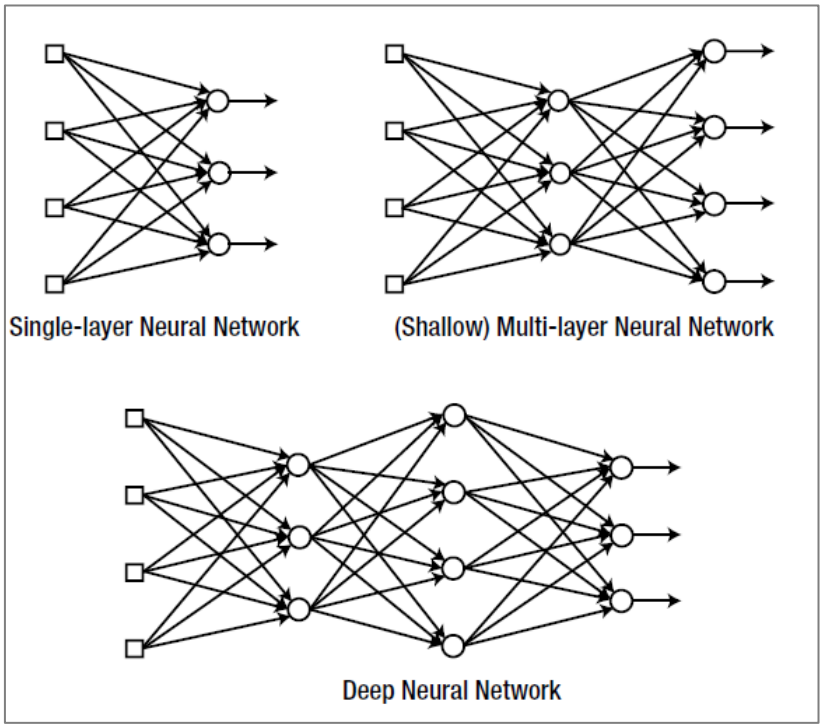


Figure 2.20. The neural network's branches are determined by the layered architecture

Under specific conditions, it becomes necessary to scrutinize training datasets, mainly when a single neural network falls short. When conventional artificial neural networks face obstacles in their progression due to inspection data, the recourse often involves utilizing multiple neural networks, specifically those associated with deep learning. A comprehensive exploration of the complexities and advancements in DL is available in the following subheading.

Deep learning

In ML, deep learning comprises multi-layered artificial neural networks incorporating supervised and unsupervised learning components. It is essential to recognize that deep learning is not a standalone technology; rather, it represents using multi-layered artificial neural networks, commonly called deep neural networks (DNNs), to address diverse problems. These problems can range from spatial recognition to image analysis, as indicated by (Turner et al.,) Conducting investigations in deep learning requires extensive datasets with a substantial number of examples to train networks effectively. Researchers, mindful of these training datasets, possess the ability to modify the approach to obtaining outputs. Algorithms, designed by their utilities, explicitly conform to concurrent benchmarks.

Meanwhile, training the datasets could take a long or short time. The important thing is to make the layers work in progress. That is why, according to data, the effectiveness of deep neural networks is implemented in “Convolutional neural network (CNN), Recurrent neural network (RNN), and Generative Adversarial Networks (GANs).” Using these algorithms, after training your data, you can validate it. The investigation into the constitution of ANN and DNN centers on understanding how machines perceive and calculate, primarily leveraging extensive datasets to mimic human vision through computer vision, image processing, and objective detection.

2.3. Informed Decision Making

"Informed machine learning" involves incorporating additional human knowledge, skill, or pre-existing knowledge into conventional machine learning procedures. This method aims to integrate human expertise into machine learning models to enhance the effectiveness, reliability, and significance of the findings they produce.

2.3.1. Generative adversarial neural networks(GANs)

The methods of learning used in the baseline are determined depending on the supervised or unsupervised learning options, and there are sample learning models that are preferred depending on the variables (Lee et al., 2019). According to the labelization stage, which is the data processing process, supervised or unsupervised learning techniques can be preferred. Among these learning models, unsupervised and semi-supervised learning models are mainly used with GAN networks. The dataset does not need to be processed or fully processed as in the supervised learning model. A "noise vector" is given for the image processing process during the training of GAN networks. This randomized noise vector allows the generated images to pass through a multilayer perceptron and be discriminated by a multilayer discriminator. GAN models consist of noising and denoising processes. High-quality photos are achieved by applying Gaussian noise to the image data, followed by a progressive removal process. Goodfellow (2014) discovered that generative models are mappings from simple noises to data and that the "secret code" formulation proceeds based on this data. The formulation of the "secret code" of diffusion models is a sequence of progressively denoised images. The inverse technique eliminates noise. It generates the noise by adding Gaussian noise along the Markov chain. However, this process produces T times noisy samples. The pictures of excellent quality are gradually produced via the Gaussian noise input $xT \sim N(0, I)$. The Unet architecture is essential for distributing this noisy vector (Ronnebeger et al., 2015; Bombach et al., 2022). The Markov chain envisages that the probability of each probability in its cycle in the probability index works together with the probability of other units. The rule for interaction between each unit is clear. The probability continues until the rule is realized. For this reason, its use in generative networks is predicted to continue until the denoising process is completed when reproducible images are produced together with the Gaussian noise function. UNET in the CGAN algorithm reduces this process by recalculating the component of each patch and combining it again with the completion of this set of rules, which helps to obtain more explicit images with lower Gaussian noise.

DCGAN architecture

The main feature distinguishing DCGAN (Deep Convolutional Generative Adversarial Network) from other neural networks is its multiple hidden layers. Therefore, in artificial

neural networks, the calculation of the input data, together with the weights in these layers, takes place together with the loss function. Optimization involves retraining the network according to the state in which the outputs are generated (Figure 2.21) (Chollet, 2018).

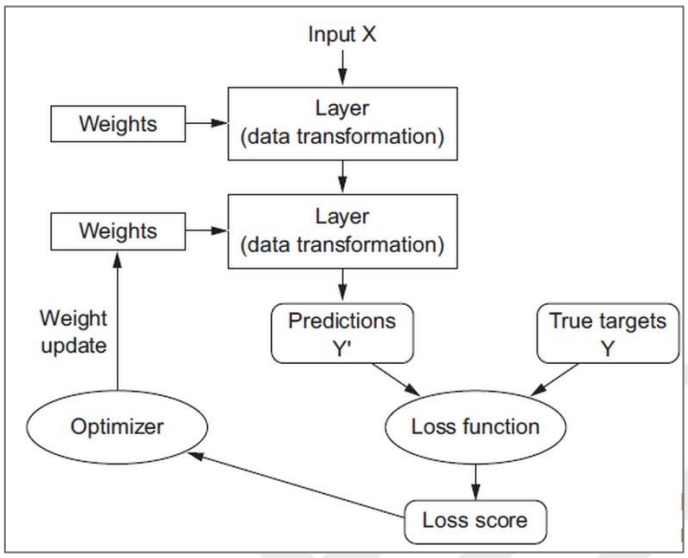


Figure 2.21. The procedure involves determining the network's layers by considering the weights of the inputs and subsequently optimizing them based on the values of the loss function

The development of DCGAN was carried out by Radford et al. (2015). The generator is processed together with the random noise vector to generate images; these generated images are visually discriminated based on being real or fake, and this cycle continues until the real image is captured. The modification of using strided convolutions instead of pooling layers in the discriminator presents several advantages. By replacing pooling with stridden convolutions, the discriminator avoids the reduction in resolution, preserving valuable information crucial for distinguishing real from fake data. Stridden convolutions facilitate downsampling while maintaining information flow, enhancing the discriminator's ability to detect fake samples. The generator employs fractional-strided convolutions for upsampling, enabling it to learn intricate details and generate high-resolution outputs.

Another critical enhancement involves batch normalization for both the generator and discriminator. This technique standardizes inputs to each layer, expediting training and enhancing stability, particularly in deeper architectures. It mitigates issues such as vanishing or exploding gradients commonly encountered in neural networks, leading to smoother convergence and improved overall performance. Eliminating connected hidden layers in

deeper architectures improves stability and training efficiency. This adjustment avoids excessive complexity, reducing the risk of overfitting and challenges in learning long-range dependencies. Focusing on convolutional layers aligns better with the spatial nature of images, allowing the network to capture local and global features effectively. Finally, the choice of activation functions, such as ReLU and LeakyReLU, is tailored to the task. ReLU is employed in the generator for all layers except the output, where Tanh ensures values lie between -1 and 1, suitable for image pixel intensities. In the discriminator, LeakyReLU is chosen to prevent neuron death, enabling the flow of small gradients and facilitating the learning of delicate features.

The noise-generating process in GANs occurs randomly following the labeling stage per the operational concept. The process starts by combining the generated image with the photographs stored in the database. Discriminators and generators are the two fundamental components that comprise GANs (Figure 2.22). The generator gains the capacity to create believable data samples. The events generated act as adverse instances for the discriminator to learn from. The discriminator is trained to know the difference between the synthetic data generated through the generator and the genuine data. The discriminator penalizes the generator for creating counterfeit results (URL-1).

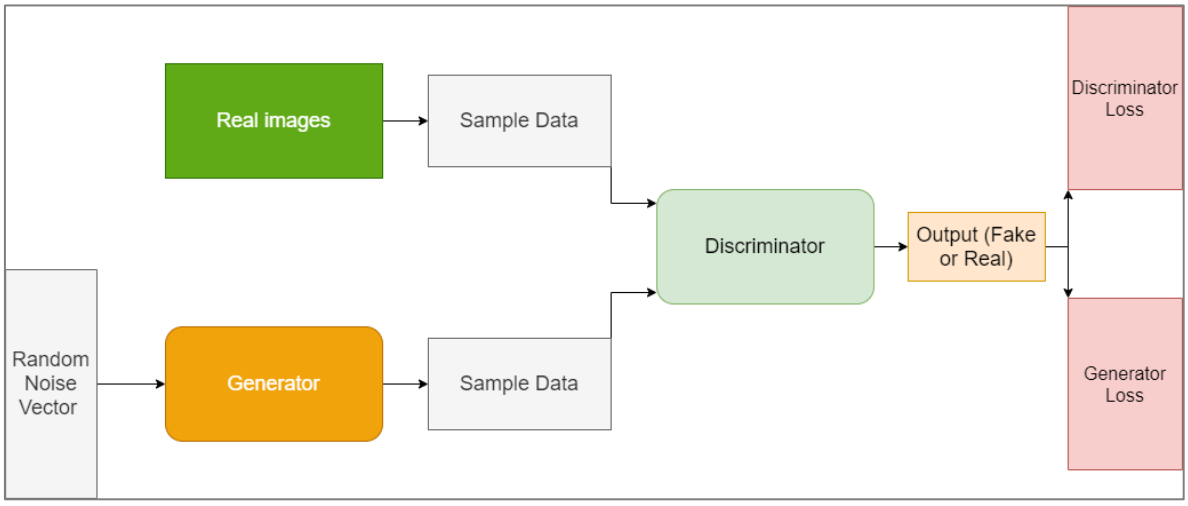


Figure 2.22. The architecture of DCGAN (URL-1)

The generated data is directly linked to the input supplied by the discriminator. The Discriminator's filtering, obtained by backpropagation, serves as a signal to update the Generator's weights. This process ensures the network continues training until it reaches an ideal state for successive generations.

Loss function

The loss function is crucial for optimizing learning and generation in machine learning systems. It is essential to recognize its limitations and consider the intricate interaction of many components throughout the optimization process. As this value gets closer to zero, the results get more optimistic.

"min G max D V (D, G) = Ex∼pdata(x) [log D(x)] + Ez∼pz(z) [log(1 − D(G(z)))]" (Goodfellow, 2014) (2.1)

Generator loss [D(G(z))]

The Generator works to optimize this function. It seeks to maximize the Discriminator's performance for its generated instances (URL-1).

Discriminator loss D(x) - D(G(z))

The Discriminator's goal is to maximize this function. It aims to maximize the difference between its performance with real and fake cases (URL-1).

Activation function

A different algorithm has been developed to complicate the training process of artificial neural networks beyond linear regression. The differentiability of these algorithms becomes crucial in the network's complex structure, surpassing the limits imposed by the linearity of the functions used. "A rectified linear unit (ReLU) is a function meant to zero out negative values, whereas a sigmoid 'squashes' arbitrary values into the [0, 1] interval outputting something that can be interpreted as a probability" (Chollet, 2018). While single neural networks often use linear and sigmoid functions, deep neural networks prefer RELU (Rectified Linear Unit) and LEAKY RELU activation functions (Feng, et.al.2020) The functions are used according to the learning model of the network to be trained. Sigmoid activation function is mostly preferred for single neural networks while RELU and Leaky Relu activation functions are preferred for complex neural networks. (Figure 2.23).

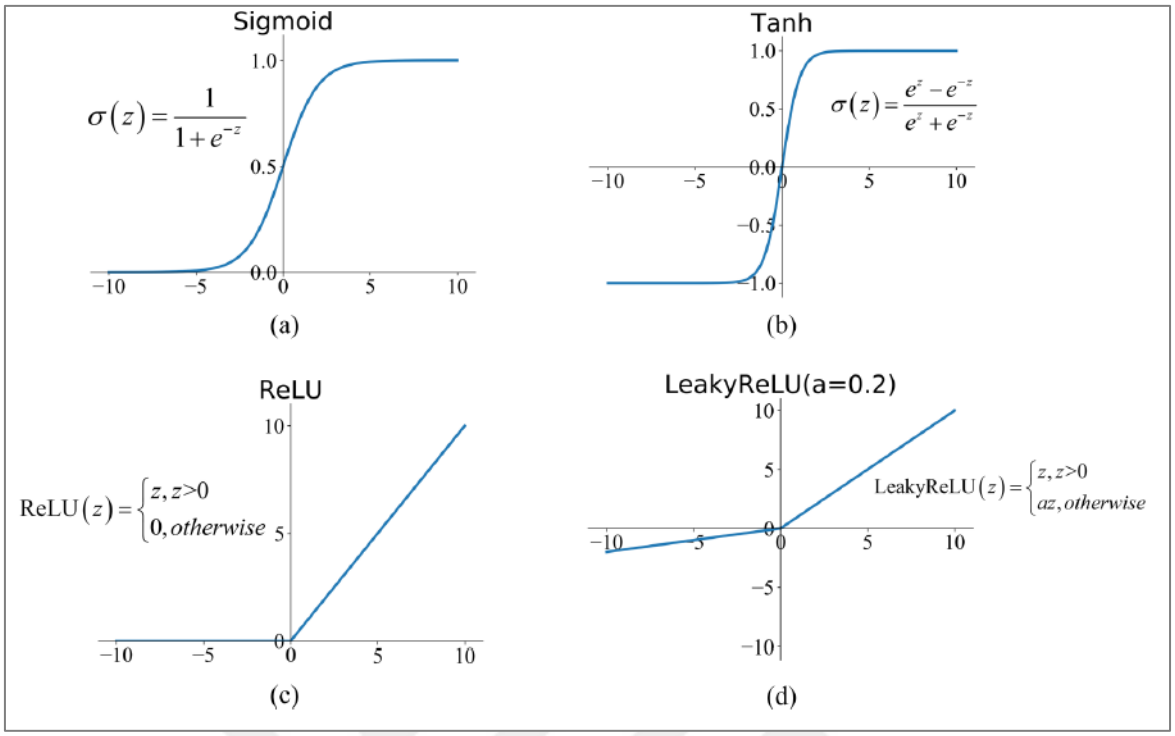


Figure 2.23. Activation functions (Feng, et.al.2020)

Since our own dataset will be used within the scope of the study, the Leaky Relu activation function was preferred.

Pix2pix (cGAN)

In obtaining another image using raster image-based data, translation between data takes place. During this transfer, the data is reproduced using real images and labels, masks, and segmented images (using semantic segmentation) created from real images or by giving them input values to the networks without processing. This duplication and decomposition process varies according to the algorithm's operation. Supervised and unsupervised image-to-image translation processes have been explained methodologically in the previous sections. Figure 2.24 shows the positions of algorithms such as DCGAN and CGAN (Lupion et al.2022) which we have used to classify GAN networks. The Pix2pix (CGAN) algorithm was developed by Isola et al. (2017), and some of the basic concepts in the architecture of this algorithm should be explained in sub-headings because these basic concepts and functions affect the primary evaluation criteria in the algorithm. “The Pix2pix algorithm consists of a generator with a U-Net-based architecture and a discriminator modeled as a convolutional *PatchGAN classifier*” (Isola et al., 2017).

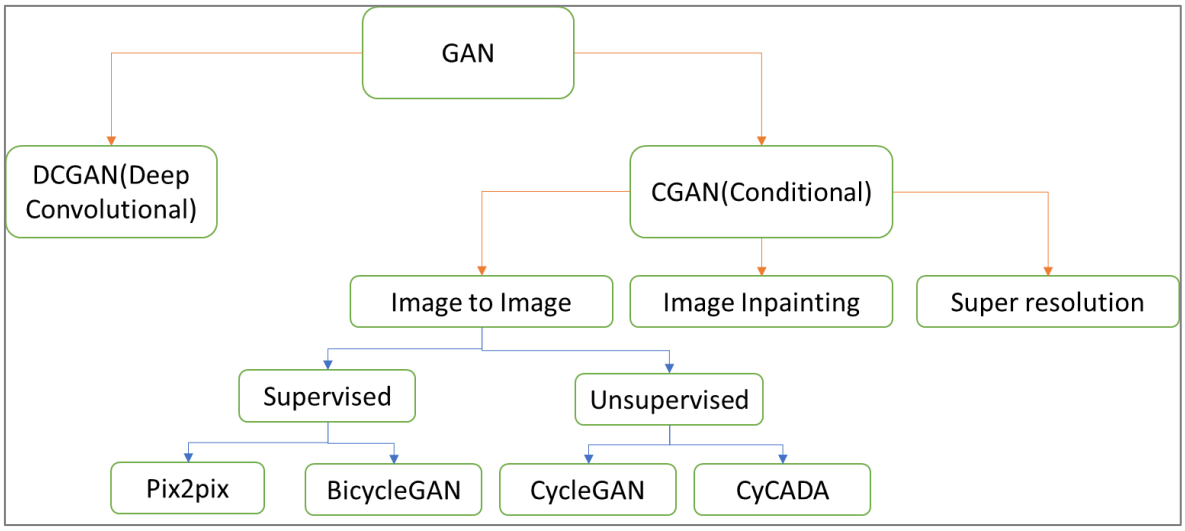


Figure 2.24. Different GAN algorithms (edited by the author)

U-Net

"The UNet structure is a U-shaped architecture consisting of 3 main parts (Figure 2.25). The variable k in the input data represents the image dimensions, each having 256*256 layers. UNet sections are:

1)Contracting path-decoder (contracting path-decoder)
2)Narrow passage section
3) It consists of an expansion section (decoder) (expansive path-encoder) (Isola et al., 2017) (Figure 2.25)."

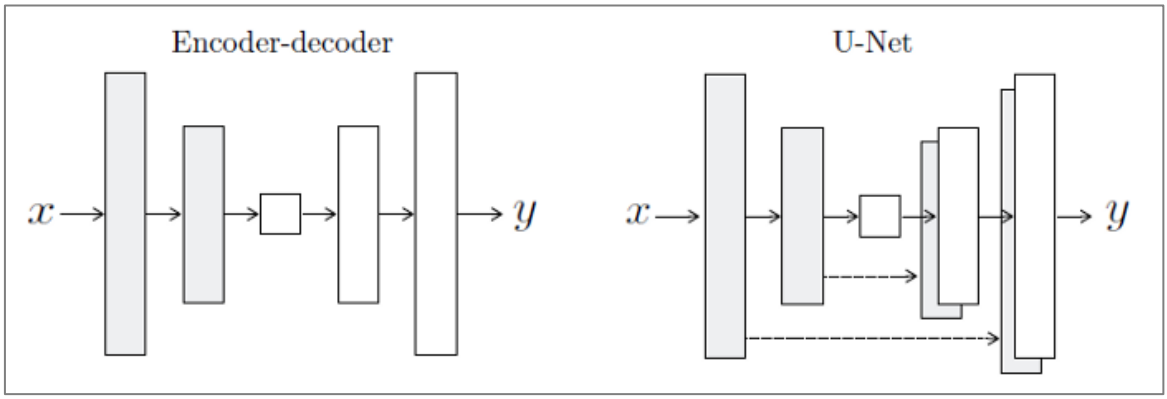


Figure 2.25. The U-Net architecture consists of an encoder-decoder structure with skip links

The narrowing section adheres to a standard convolutional neural network layout and comprises numerous narrowing blocks, as seen in Figure 2.26. Each block processes an input

picture through a convolutional layer, applies a rectified linear unit (ReLU) activation function, subsamples using a stride, and then performs maximum concatenation.

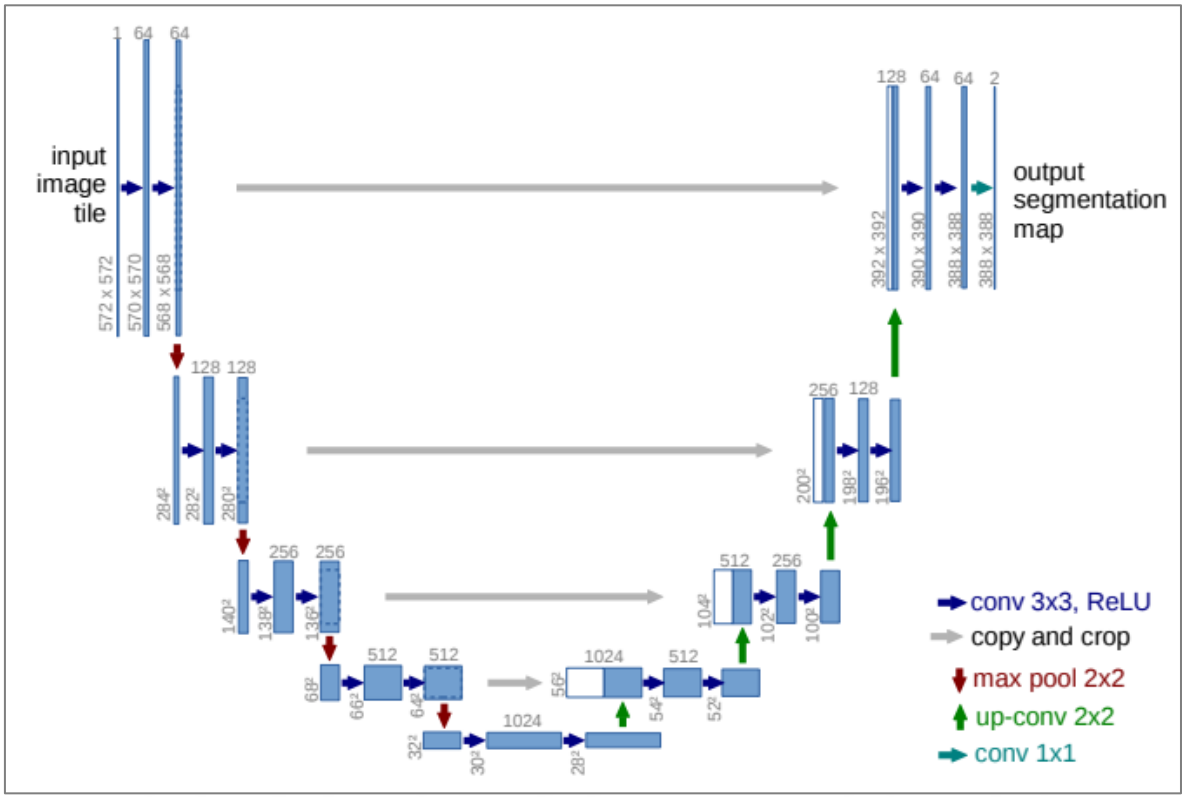


Figure 2.26. Pooling process in the U-net as described by Ronneberger, Fischer, and Brox in 2015.

The stride shift determines how many pixels the applied filter shifts over the main image. The narrow gate section combines the narrowing and widening layers. Like the contraction phase, the expansion section comprises many blocks. Within every unit, the input image undergoes convolutional layer processing followed by upsampling layer processing. Unlike the contraction blocks, the total amount of feature maps is reduced by half to maintain symmetry. The supplied image is interwoven into the respective contraction layer. This guarantees that the acquired features from the contraction phase organize the new image. The quantity of expansion blocks is equal to the amount of contraction blocks. The output of the expansion block is subsequently sent into another convolution layer, with the number of feature maps matching the quantity required of segments or classes.

The loss function for the generated images is the sigmoid cross-entropy. The technique calculates the L1 loss and the mean absolute error (MAE) between the predicted and target

(URL-2) (Figure 2.27). This mistake guarantees that the produced image resembles the target image in structure.

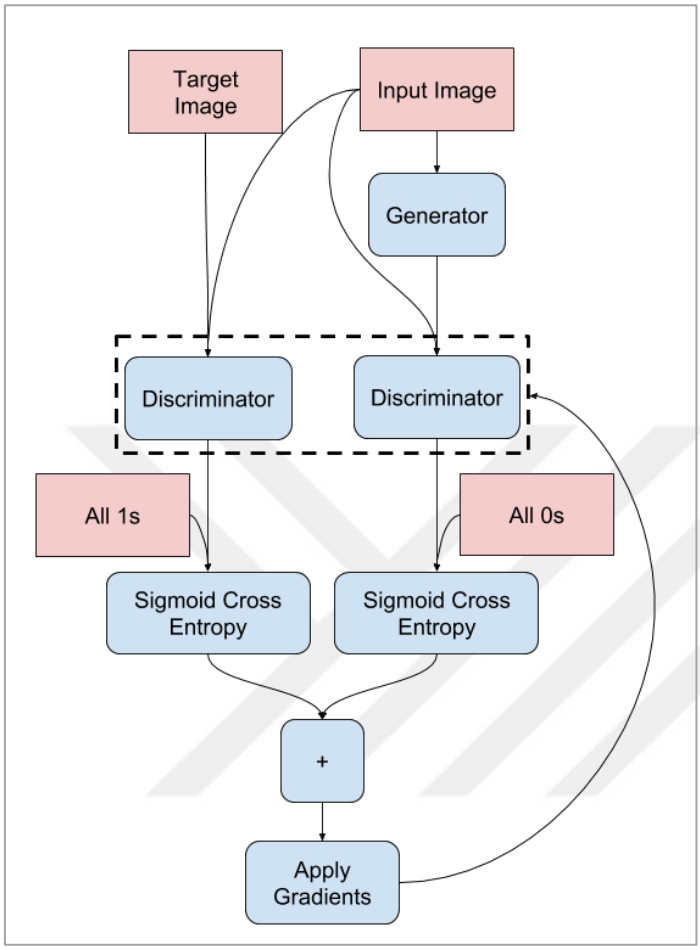


Figure 2.27. Training model of GAN (URL-2)

The algorithm's performance during training can be evaluated by comparing the loss values. These graphs consist of 4 graphs defined in the algorithm, namely Generation Loss Function 1,2 and Discrimination Loss Function 1,2. According to the evaluation logic of the graphs, it is thought that the generation and discrimination functions of the network are approaching to produce the most accurate image at the points where these values are the closest.

Loss function

The loss function calculates the designed model's performance and error rate. The final layer of deep networks is where the loss value is specified. The loss function calculates errors by transforming the problem into an optimization problem and is referred to as an objective function or cost function in optimization terminology. The loss quantifies the discrepancy

between the model's forecast and the actual data. If our model does not forecast well, the discrepancy between the actual and anticipated values will be significant, resulting in a large loss.

Conversely, a well-performing model will yield a low loss value. A loss of 0 will occur if it is identical. We anticipate a proficient model to exhibit a loss value nearing 0. Understanding graphs is more complex when working with a GAN (or a pix2pix-like cGAN) than with a primary classification or regression model. Indicators to monitor: - Whether gen_gan_loss or disc_loss decreases significantly, it suggests one model is overpowering the other, hindering successful training of the combined model.

Discrimination loss

The discriminator's role is to assess the resemblance between the input and unfamiliar images. The unidentified image may be part of the dataset, like the target image, or an output image generated by the producer. In the Pix2Pix network, the PatchGAN discriminator is a unique element to categorize individual (N x N) photos as genuine or fraudulent (URL-2). The architecture is named "PatchGAN" since each pixel in the discriminator result (30×30 picture) represents a 70×70 section of the source image. It is worth mentioning that due to the input photos being 256×256 in size, there is a significant overlap between them (URL-3).

Total generation loss

This formula (2.2) was determined by Isola et al. (2017) to get the generator loss is as follows:

Total agan loss = gan_loss + LAMBDA * l1_loss, LAMBDA = 100. (2.2)

L1 loss

Realistic generated images will allow classifiers trained on genuine photos to classify synthesized images accurately. The discrepancy between the real and the predicted image must be addressed to transition the unsupervised system to a supervised one.

Generation loss

The generator aims to optimize this function. It seeks to optimize the discriminator's output for counterfeit occurrences (URL-3). However, it has been noted that the Pix2pix algorithm has certain limitations. The generator model employs a 256*256 visual network and generates images by layering them in a specific order: 128*128, 64*64, 32*32, 16*16, 8*8, 4*4, 2*2, and 1*1. This process involves pooling and padding operations to separate the images into individual pixels. It has been noted that this visual model exhibits varying behavior with each iteration of network training.

Measurement metrics

This section describes the measurement metrics used to compare ground truth and synthetic images.

Structural similarity metric (SSIM)

This metric is a statistic that measures the similarity between two photographs using the mean (μ) and standard deviation (σ) rather than the positional discrepancy between pixels (Nilsson and Akenine, 2020). It examines perceptual characteristics, such as masking, brightness, and contrast, and the apparent alteration in the image. The equation (2.3) provides the SSIM similarity calculation between the real picture x and the fake image $\hat{x}$.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y+C_1)(2\sigma_{xy}+C_2)}{(\mu_{2x}+\mu_{2y}+C_1)(\sigma_{2x}+\sigma_{2y}+C_2)} \quad (2.3)$$

Mean squared error (MSE)

This metric is a comparison metric developed by Mihelich et al. The degree of similarity between the ground truth and the predicted values is determined by computing the squared average of the differences between corresponding points. Equation 2.4 provides the general formula for the MSE approach.

$$L2 = MSE(x, \hat{x}) = 1\ n\ \sum(xi - \hat{x}i\ )\ 2\ n\ i{=}1 \quad (2.4)$$

The equation computes the squared variances between each pixel in the real x and fake $\hat{x}$, which is similar to x. The sum is divided by the number of pixels (n) in the images. A small MSE score indicates a strong resemblance between the two images.

Peak signal to noise ratio (PSNR)

This metric proposed by Fardo et al. (2016) calculates the similarity between image pairs logarithmically. The signal is considered as data or a real image. Noise is the error caused by distortion or compression in the data. The PSNR ratio between two images or two signals is calculated in decibels. It is shown in Equation 2.5.

$$PSNR = 10 \log_{10} \left(\frac{R^2}{msE} \right) \tag{2.5}$$

2.3.2. Object detection

Object detection algorithms have been extensively utilized in various sectors, such as self-driving cars, robot vision, and video surveillance, for the past two decades (Zou et al., 2023). The algorithms used in this field have been developed after deep learning algorithms to make classifications and object detection more accurate. After CNN algorithms, R-CNN (Girshick et al., 2014) and Fast R-CNN (Ren et al., 2015) algorithms were developed. Object detection algorithms are used to accurately locate objects in images by classifying objects obtained from digital images (Zou et al., 2023). Although the algorithms used for classification in these algorithms are used for semantic segmentation by the supervised learning method before the dataset, the labelization process must be processed to determine the images belonging to the classes to classify digital images. Especially with the classification and layout of the Bounding Box (BBOX) (Felzenszwalb et al., 2009), regionally predicted classifications have become more clearly perceived. Zhao et al. (2019) categorized generic object detection algorithms into a pair of categories: classification methods utilizing area descriptions and classification methods utilizing regression analysis (Figure 2.28). The 'Yolo' algorithm has high accuracy rates, including the real-time object detection process, thanks to BBOX detections (Redmon et al., 2016). Yolo algorithms have been improved to enable users to use them faster and more effectively (Yar et al., 2023).

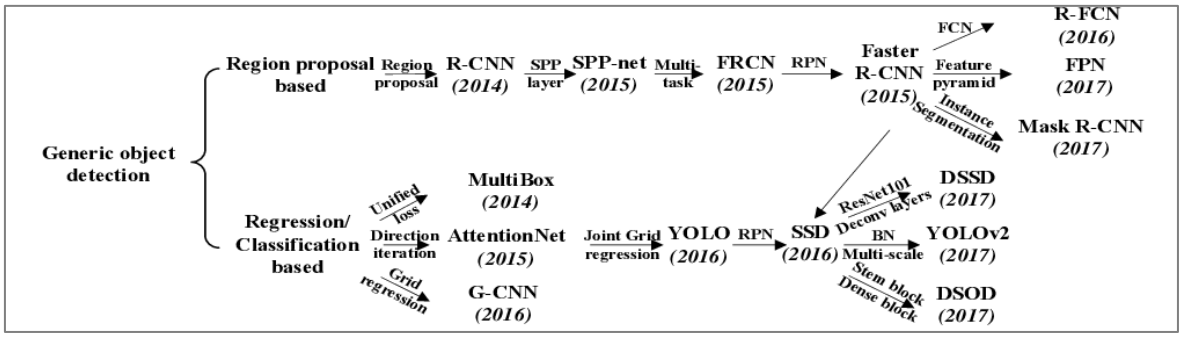


Figure 2.28. Generic object detection techniques (Zhao et al., 2019)

The Yolov5 algorithm was used by Feng et al. (2021), and the highest results in precision, recall, and mAP values were observed in the Yolov5 algorithm from the data augmentation method. Anchor boxes are aligned with reference to their center points (Figures 2.29a and 2.29b).

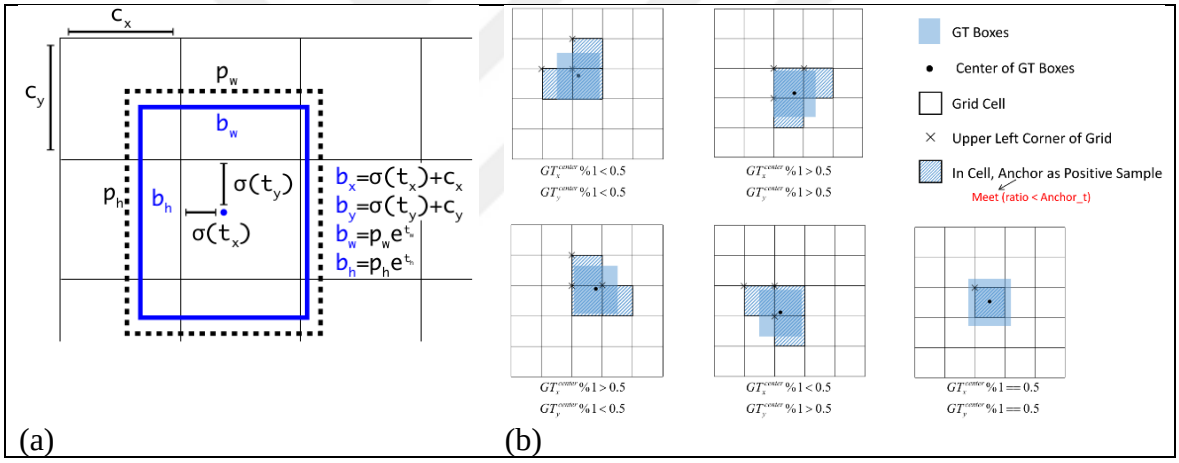


Figure 2.29. (a) Yolov5 shows how BBOX detection is performed, (b) Yolov5 shows the BBOX detection patterns detected on the image divided into grid cells.(URL-3)

The BBOX detection of the predicted image is determined by the x and y coordinates, as well as the width and height, within the constraints of the boxed images. The calculation of this method, on the other hand, is expressed by the notation present in formula 2.6, which can be found below.

$$bx = (2 \cdot \sigma(tx) - 0.5) + cx \; by = (2 \cdot \sigma(ty) - 0.5) + cy \; bw = pw \cdot (2 \cdot \sigma(tw))2 \; bh = ph \cdot (2 \cdot \sigma(th))2 \quad (2.6)$$

The architecture of the Yolov5 algorithm is described below (URL-4) (Figure 2.30). Thanks to this architecture, data augmentation operations are also diversified. Because it is designed

to be quick, accurate, and simple to use, YOLOv5 is a great fit for a wide variety of tasks involving object detection, such as segmentation, and image classification.

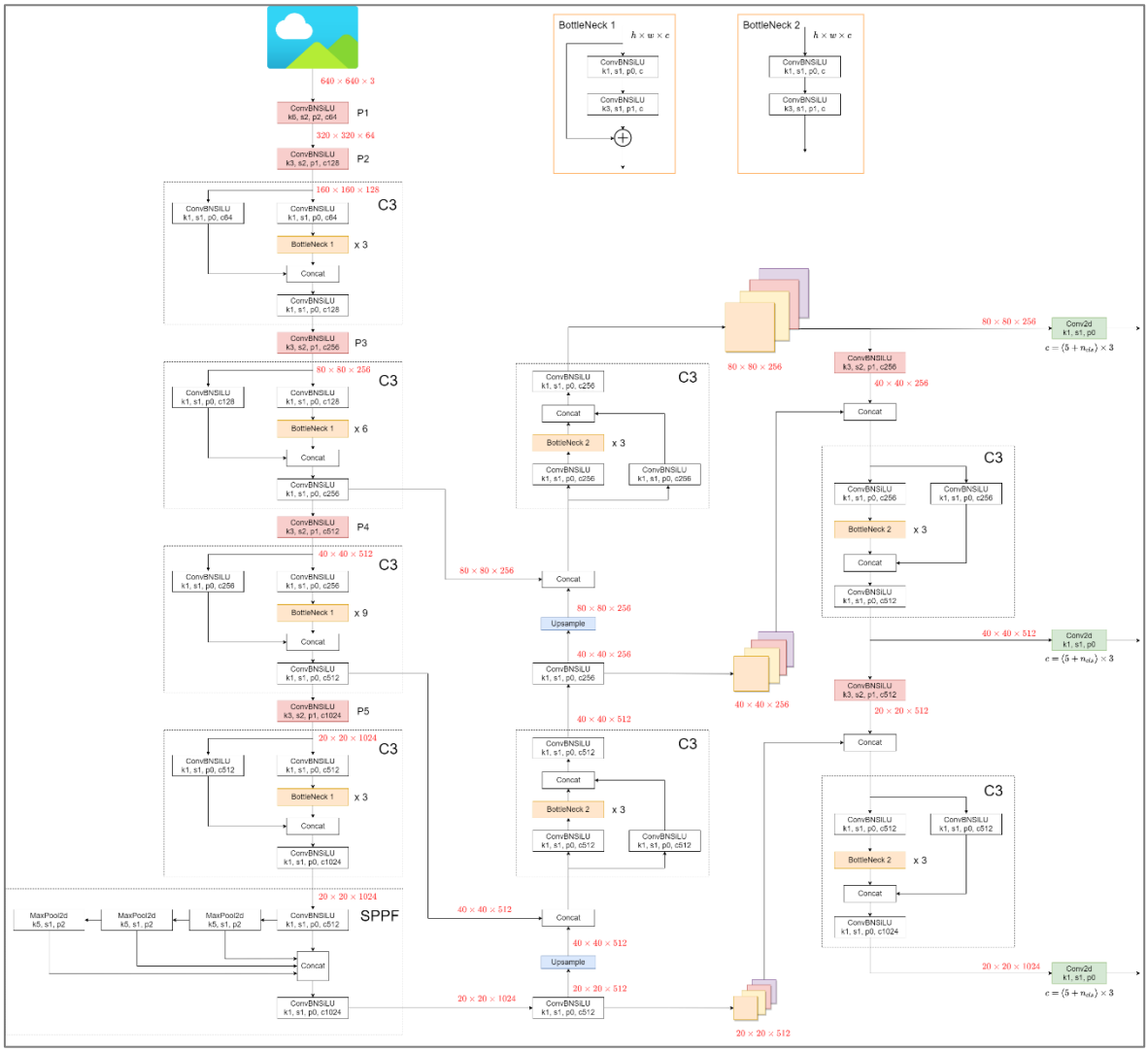


Figure 2.30. Yolov5 architecture (URL-4)

The Yolov8 algorithm is an upgraded version of the Yolov5 algorithm as a classification model (Terven et al., 2023). The Yolov8 algorithm, which has a similar architecture, offers faster object detection and optimizations (URL-5) (Figure 2.31). Heydarian (2022) states that classification techniques are evaluated based on the number of classes. Classification models for multi-class input data are challenging to interpret with binary and regression models.

This new computer vision model, known as YOLOv8, was developed by Ultralytics, the same company that was responsible for developing YOLOv5. Capability to perform object

detection, classification, and segmentation tasks is included in the YOLOv8 model. This support can be accessed through a Python package as well as a command line interface (URL-5).

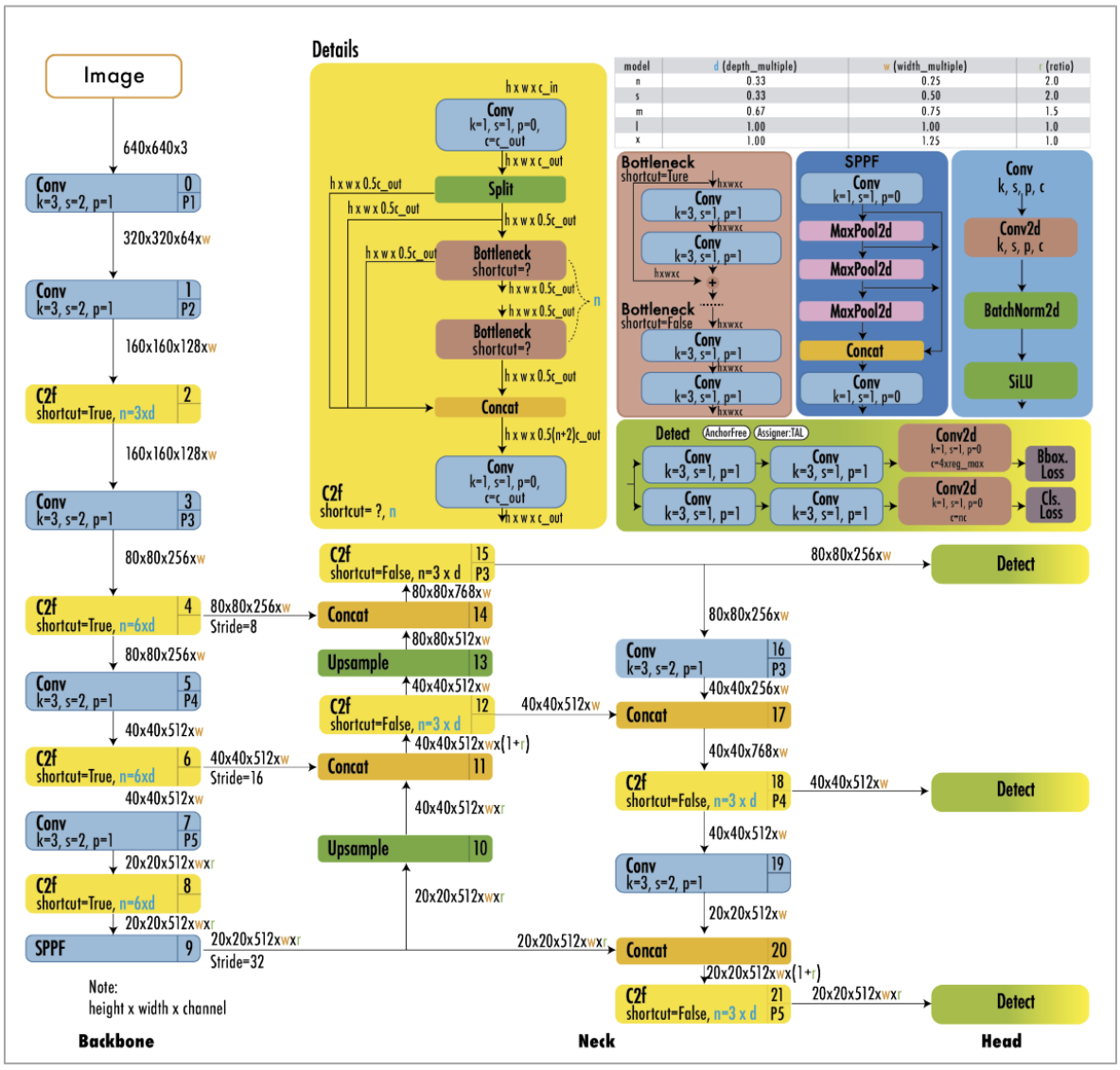


Figure 2.31. Yolov8 architecture (URL-5)

When classifying objects consisting of two courses, it is easier to determine whether the algorithm works correctly. However, the confusion matrix is considered for classification studies consisting of multiple groups of objects (Figure 2.32). This model is created by positioning the actual values and predicted values in the matrix system, considering the correct or incorrect predictions of the classes. However, “TP, TN, FP, FN” values must be known to evaluate this matrix system. Although this matrix is essential for assessing the algorithm, it is insufficient.

		Actual Class	
		Positive (**P**)	Negative (**N**)
Predicted Class	Positive (**P**)	True Positive (**TP**)	False Positive (**FP**)
	Negative (**N**)	False Negative (**FN**)	True Negative (**TN**)

Figure 2.32. Confusion Matrix (URL-6)

When calculating the values of this matrix, the goal is to reveal the accuracy value. This accuracy value is expressed in the following formula (2.7). The formula for calculating the precision value is expressed in (2.8), and the recall formula in (2.10). However, along with all these calculations, the algorithm's accuracy is measured by the F1 score to know whether the algorithm is working correctly (2.10).

$$Accuracy = \frac{TN\ +\ TP}{TN + FP\ +\ FN\ +\ T\ P} \tag{2.7}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.8}$$

$$Recall = \frac{TP}{FN\ +\ T\ P} \tag{2.9}$$

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{2.10}$$

2.4. How Do AI and Designers Collaborate?

In modern philosophy, Descartes emphasized the theory of knowledge by focusing on the ‘human being’ himself. In a framework in which Plato distinguished between the knowledge obtained by reason and the reality of the knowledge acquired by Aristotle through experimental inferences, Descartes emphasized that knowledge should be questioned through "skepticism" by approaching the accuracy of knowledge with doubt (Çüçen, 2012). Nowadays, the certainty gained by deep neural networks due to learning can be evaluated as "Explainable Artificial Intelligence" as they are also prone to errors. The precision and accuracy values may also be inaccurate when training the network. They may not correlate with the results, which may be due to the machine's uncertainty about its errors. Mathematical expressions are composed of numbers and indexes, which are syntactic and expressed symbolically. Gödel (1931) laid the foundations of modern mathematics with his

"Completeness Theorem," which states that if we consider syntax and semantics assets, each set is provably correct. Explainable AI (XAI), a secondary algorithm for checking algorithms in the architecture whose correctness is not sure, is a system created to prove the consistency and accuracy of the generated images. Gödel's "Incompleteness Theorem" forms the basis of this system. Gödel's theorem is based on the idea that a more robust system should check systems to prove their consistency/completeness. However, it may not be possible to talk about the consistency of some systems. This cycle can continue without absolute consistency. The concepts of "explainability" and "interpretability" were developed to understand the working logic of AI. By demystifying the workings of an algorithm, we unlock the ability to analyze its outputs and pinpoint discrepancies or biases. However, for many factors we cannot understand, the "explainability and interpretability" part gains importance. When the inputs and outputs are known, but we do not know and cannot predict how the algorithm or any system works, we encounter the 'black box3' theorem. Nevertheless, as we delve into the algorithm's functionality, comprehending the processes behind predictions and the outcomes' rationale leads us to the 'white box[4]' theory (Giuste et al.,2022) (Figure 2.33). It has a similar meaning in all aspects where this theory is used in the literature. Explainable approaches are developed to allow the system to be predicted and interpreted.

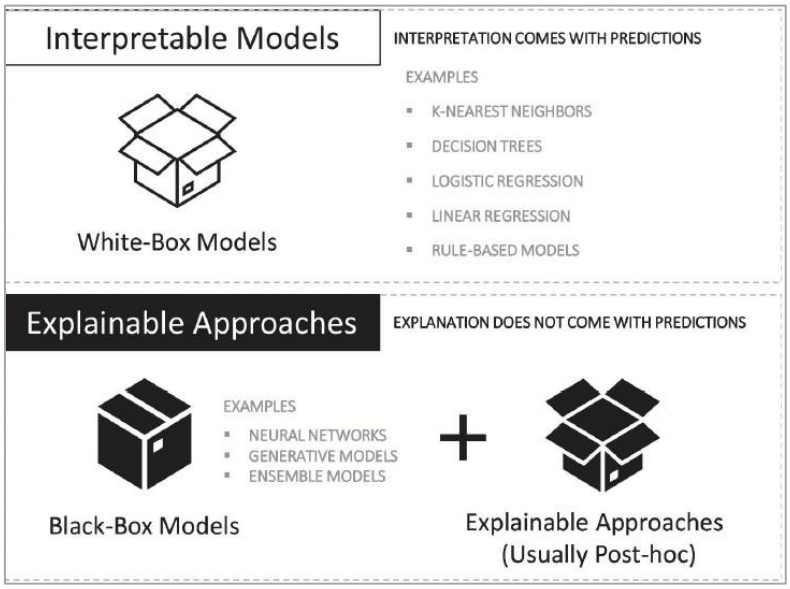


Figure 2.33. Interpretability and explainability of AI (Giuste et. al,2022)

[3] A "black box" system is a working algorithm with inputs and outputs. However, it is not known exactly what is inside and how the system works (Rothman, 2020).
[4] The system can be understood by the user with the results it provides. The results predicted by the algorithms provide information, interpretable and verifiable results (Rothman, 2020).

In 1940, Alan Turing introduced the concept of "explainability" to complex systems' computational and decision-making capabilities by inventing a machine. (Rothman,2020). In this machine, "Errors" explains the system by identifying and solving the problems that the ciphers experience while decrypting. The mention of "error" in the paper would have been unclear to readers, but in a February 1947 address by Turing, the concept of "making mistakes" explained the challenge to machine intelligence raised by Gödel's theorem. The issue raised in the 1950 paper, known as the "Mathematical Objection," argues that no Turing machine, or computer program, can match the success of a human being. By adhering to principles derived from formal axioms, individuals can ascertain the validity of mathematical statements that cannot be proven. Pedro Domingos' observations indicate that the most crucial feature of how the brain learns with the neurons in the brain is that it calculates which connections are caused by which problems and learns through errors (Leach,2022). Machine intelligence is used as an auxiliary tool in mathematical proofs, and humans complement the creative aspect of mathematics. In other words, AI algorithms (machine intelligence) can help automate complex calculations and find errors. Still, mathematical models continue to significantly impact creating issues, developing ideas, and evaluating the correctness of proofs. Ko et al. (2023) mention that nowadays, image to image generation methods offer hybrid methods by using GAN algorithms, GNN algorithms, CNN algorithms together with human involvement (Figure 2.34).

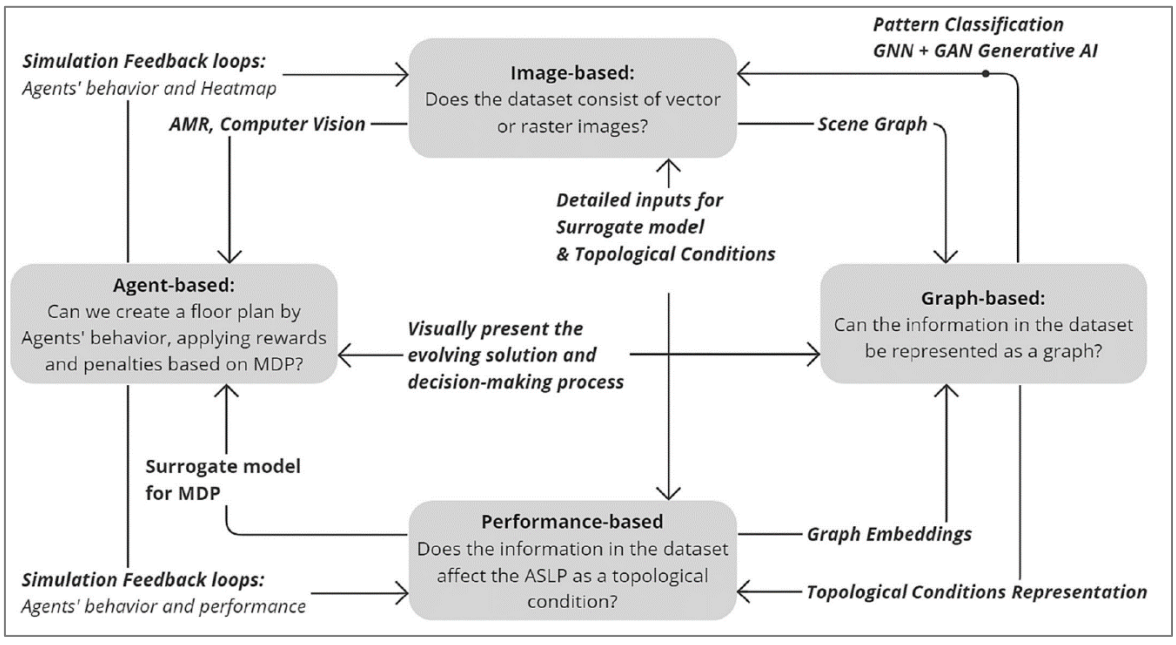


Figure 2.34. Hybrid AI and collaboration approaches

The fact that the GAN networks used in our study are 'image to image translation' methods and that it is not clear how the generated/predicted images are generated and according to what they are generated is an indication that the functioning of the algorithm cannot be explained and interpreted from the outside. Merely assessing the resemblance between the predicted and ground truth based on look alone does not benefit the designer or architect. For this reason, the object detection algorithm, another deep learning algorithm, was used to make the generated plan layouts measurable. The fact that this algorithm is a classification algorithm and that the objects classified in line with the processed data are transparent increases the clarity and accuracy of the predictions developed over the given images. Object detection was preferred because classification-based CNN algorithms have higher explainability/interpretability, and the results give measurable space layout planning values (BBOX detection). The preparation of the data used in the operation of all these algorithms was realized by the architect. The architect was also accountable for the outcome evaluation and interpretation (Figure 2.35).

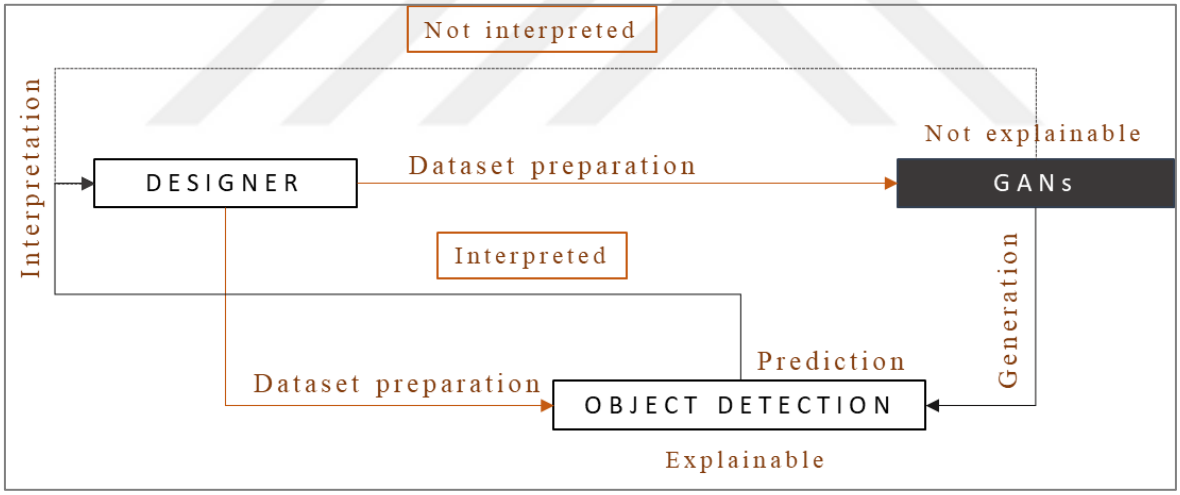


Figure 2.35. “How Do AI and Designers Collaborate?” workflow graphical expression

2.4.1 Related works with architecture and ML

DL has been a potent visual arts, construction, architecture, and urban design tool. DNNs have been utilized to categorize artistic and architectural styles. Saleh et al. categorized visual art genres using computer-assisted techniques, distinguishing paintings by semantic-level characteristics. Llamas et al. (2017) applied CNN to classify photographs of historical buildings in architecture. In different studies, Elgammal and his team investigated the relationship between stylized paintings in art history.

Zhang et al., (2019) and colleagues semantically segmented city elements using Google Street View, focusing on the intersection of machine learning, architecture, and urban design. Cai et al., (2018) proposed detection automation using three models to automate tree cover on Google Street View. Valero et al. developed a labeling system to detect decay in the stone walls of the Royal String Castle Chapel. They performed detection and classification using supervised machine learning algorithms.

A 3D model of the building is generated by segmenting the facade using the acquired data. In another study, under the supervision of a team of experts in architectural history and theory, Yoshimura et al. (2014) conducted a computer visualization of the works of thirty-four architects. They transformed them into an adjunct decision support system. Many images were analyzed to study the architectural style of the works. The computer vision algorithm makes it possible to determine which architect's style the image matches or has similarities.

The synthesis of architectural design and machine learning is having a profound impact. Before deep learning, genetic and heuristic algorithms dominated, but generative design, aligned with generative adversarial networks (GANs), gained prominence due to its flexibility. Researchers use GANs to predict current features using convergent generative design results as ANN input datasets. Sjoberg et al. (2017) evaluated phenotypic genes in Dynamo, using designer preferences as parameters for training ANNs on cluster analysis and population generation. Wilkinson et al. (2014) concentrated on CFD using wind simulations in skyscrapers. As et al. (2018) utilized deep learning to create generative concept designs by analyzing 3D spatial interactions in Revit-based architectural models. Shape grammar and formal analysis guided generative design models, taking spatial size, customer satisfaction, sustainability, and living situation priorities into account for GAN training.

Anderson et al. (2018) devised an algorithm to optimize office plan layouts designed by architects and evaluated the algorithm's outcomes. Huang et al. (2018) utilized the GAN model framework to train machine learning algorithms to recognize floor plans based on color, subsequently generating plans with identical characteristics. Newton et al. (2019) used GANs to create facades at different training levels and clustering styles by using Le Corbusier's drawings. Data enrichment and convolutional filtering methods were integrated to achieve optimum GAN performance.

Cutellic et al. (2019) have created a bridge between neuroscience and architectural design through a brain-computer interface. Integrating machine learning algorithms to address specific assumptions is another common interest of researchers at the intersection of architecture and energy. For example, Hong et al. (2020) investigated the performance of HVAC systems in buildings using an energy-based prediction method. Gaussian Process Regression (GPR) was used to predict low-energy control strategies based on empirical data by Zeng et al. (2020). Studies at the interface of energy and architecture (Liu et al., 2020; Walker et al.,2020; Tekler et al., 2020; Lee et al., 2019). Lee et al. (2019) used a prediction model with daylighting indices. The three-dimensional models influenced the adjustment of the façades based on energy consumption, reflecting realistic materials and features, and provided input to the prediction model. Supervised learning techniques were used to estimate and evaluate the accuracy of the algorithms. Sing et al. introduced a Component-Based Machine Learning (CBML) model using Energy Plus simulation, considering building elements and energy zones to calculate performance. Glani and others have developed a hybrid model for optimization using ANNs and incorporating HVAC systems, merging physical and operational data.

Machine learning methods are utilized in multiple research projects involving BIM. Braun et al. (2019) created an annotated system that utilizes four-dimensional data to synchronize constructional surroundings with BIM models. Rahmian et al. (2020) investigated the integration of BIM with the construction environment, developing a hybrid system that links virtual and physical worlds similar to gaming. CNN models were trained for building element recognition using labeled points from monitoring systems in virtual reality assembly. Feng et al. (2020) created an extensive model that covers the entire building lifetime, focusing on environmental performance from initial design to construction. Managing the design process influences the energy cycle. BIM modeling with Revit facilitates the incorporation of parametric attributes, such as material properties and room measurements, which are then utilized in training fuzzy clustering models. These studies showcase machine learning algorithms' diverse applications and impacts in enhancing BIM practices within the architectural and construction domain.

The combination of HCI and ML has enriched the learning process for robotic design, enabling kinetic and responsive structures. Research by Rossi et al. exemplifies the development of adaptive systems that change according to user requests and environmental conditions.

Integrating reinforcement learning (RL) into the feedback loop between power generation, occupants, shading, and AI further improves these systems' adaptability and optimal control. The work of Oungrinis et al., (2014) linking machine recognition to human behavior and emotions is essential in this context.

The synergy between machine learning and various architectural and construction applications underscores these technologies' broad impact and transformative potential in the field. In construction, machine learning applications have played an essential role in improving safety and predicting accidents. Arciszewski et al. used regression models based on accident records to prevent accidents in construction environments. More recently, Choi et al. developed models to predict fatal accidents and conduct inspections by analyzing accident and injury datasets. Li et al. intersected eye-tracking technology with construction domains using supervised learning algorithms to classify construction operators' mental fatigue and identify focal points. The approach by Koehler et al., (2018) that generates urban design architecture patterns using SG shows how urban design and ML are merging. Using large data from OpenStreetMap and domain syntax analysis, Chang et al. (2019) utilized clustering using K-means unsupervised learning for assessing design concepts based on online submissions and clustering algorithms.

Bibliometric analysis

This study examines the convergence of social sciences, ICTs, and AEC, with a specific emphasis on semantic clustering. Articles were reviewed using Web of Science, CumInCAD, Science Direct, IJAC, and Google Scholar, resulting in a refined selection connected to semantic keywords. The research identified various sub-topics, acknowledging that a single publication can belong to multiple categories, which adds complexity to the analysis. If a study encompasses both generative design and GAN networks, it is unequivocally classified under both areas. As a result, a total of 73 articles were evaluated, and the disciplines of “ML/Energy Performance-Based, ML/Robotic, and ML/CNN” were found to be commonly used throughout the six-year timeframe. Following closely after were “ML/GAN and ML/Generative Design” (Ozerol and Selcuk Arslan, 2022) (Figure 2.36). The articles were classified according to years, subjects, and subsets, which produced insightful results. EPB design has been the most popular area of study in architectural research, focusing on EPB, BIM, and CNN as the main subjects.

Research was conducted intensively in the following areas: computer vision, generative design, GAN, urban studies, and building environments.

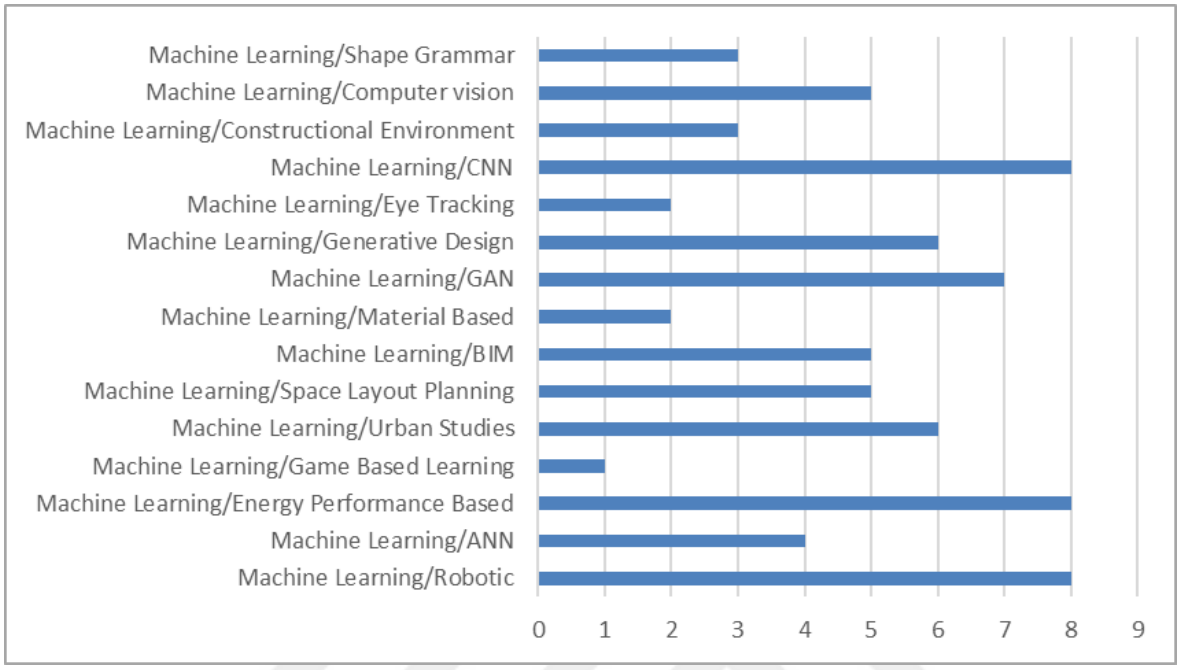


Figure 2.36.Total number of subgroups in a research study (Ozerol and Arslan Selcuk, 2022)

2018 SG was recognized as the most favored subgroup, and Generative design substantially impacted researchers in 2019 (Ozerol and Selcuk Arslan,2022) (Figure 2.37). Notably, in 2018 and 2019, the robotics, computer vision, and generative design subgroups were continually in demand. Throughout these years, research frequently employed the following keywords: architectural design, ML, BIM, GAN, and classification. Subjects advancing through intersections showed a transition towards experimenting with 3D generating methods using machine learning, notably driven by GAN networks. BIM was essential because it provided data for creating 3D models and enabled energy performance simulations or evaluations across various cross-sections.

When looking into the Sankey diagram of articles (ML/architecture), the growing influence of GAN networks on architectural practice was noticed. Urban Studies and Space Layout Planning are emphasized as popular subjects, with the beneficial impact of big data on city planning processes shaping research in these domains. The advancement of Space Layout Planning studies from 2D blueprints to 3D models, especially in combination with GAN networks, is recognized.

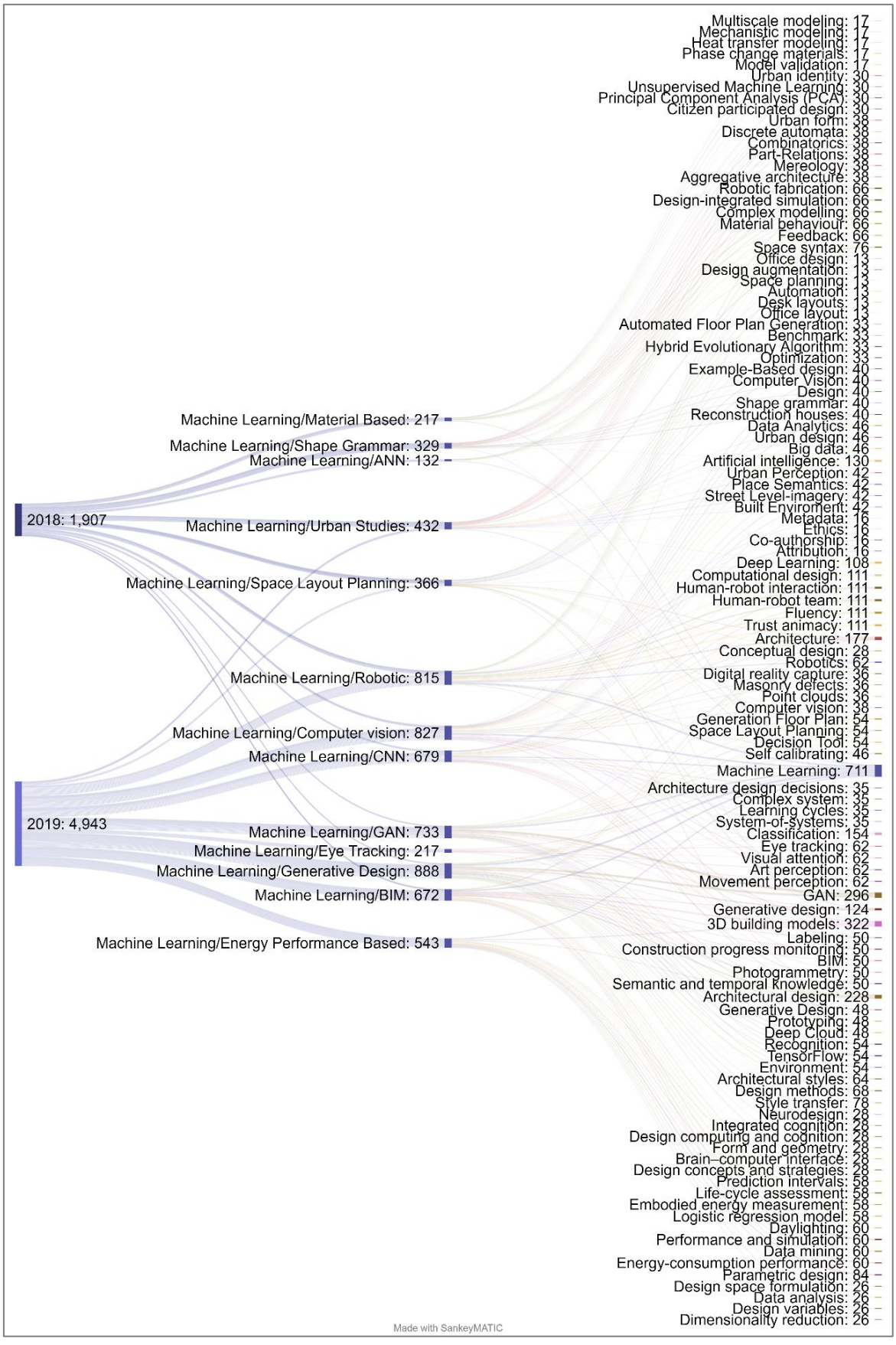


Figure 2.37. Sankey Diagram of Articles (Machine Learning/Architecture) (Ozerol and Arslan Selcuk, 2022)

Construction environment and BIM domains became secondary preferences in 2020, with ML/Energy performance-based remaining a top subgroup. CNN was used more steadily than GANs and widely in dimensional reduction, regression, classification, and cluster analysis (Ozerol and Selcuk Arslan,2022) (Figure 2.38).

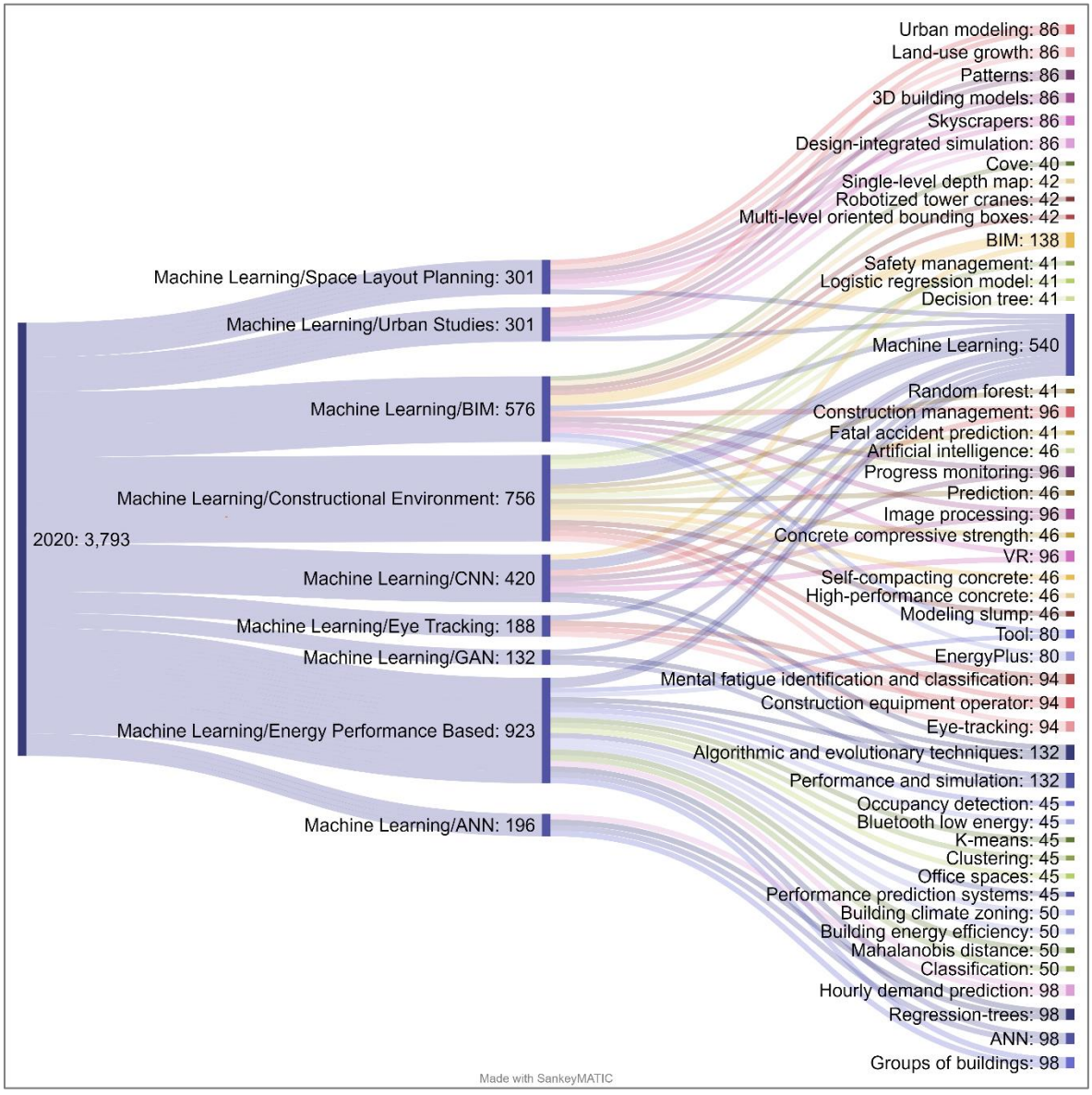


Figure 2.38.The part of 2020 of the sankey diagram (Ozerol and Arslan Selcuk, 2022)

According to the research, the leading areas of research at the confluence of ML and architecture are EPB design, GAN/generative design, computer vision, SG, and spatial layout planning. This suggests that future studies could develop within these ML frameworks. Recent research in these fields has been significantly aided by using DL methods and GANs to improve results. The possibility of creating sustainability-generating

designs is emphasized, especially at the junction with EPB design, GAN, and generative design. Future research aims to investigate the connection between EPB and BIM and utilize BIM's spatially layout planning features for planning and design analysis. Proposing the inclusion of StyleGAN as a labeling approach for support systems to give significant datasets to train GANs and create a more comprehensive range of design alternatives through feasibility assessments.

2.4.2. Current research studies between 2020-2024

Machine learning has brought a new dimension to plan layout generation methods, utilizing artificial intelligence algorithms and computational design techniques. Researchers are exploring different algorithms by using existing plans as a dataset, processing them, and testing various regeneration approaches. In a specific study by Liu et al. (2017), junctions were initially created from wall intersections in plan diagrams and treated as raster images. The colors of rooms were processed with pixel-based definition, and the first layer, consisting of vectorially positioned walls, was generated. The positions of window and door openings were identified. Post-processing with a CNN technique converted the data from a raster file to a vector format, a vital stage in the plan layout creation process (Figure 2.39).

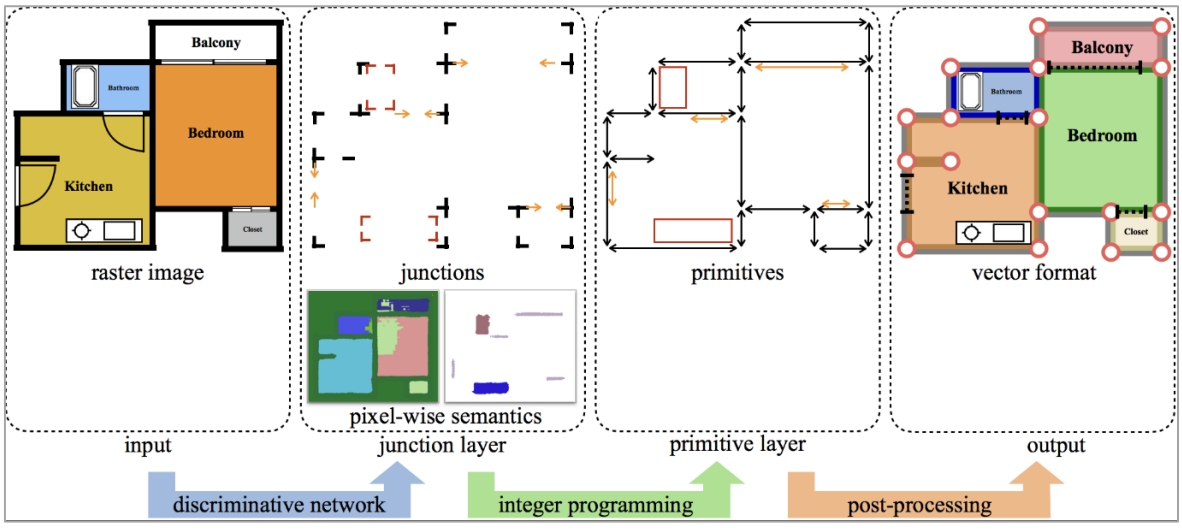


Figure 2.39. The plans generated with CNN, first, the joint formations of the walls were revealed according to the joints of the walls, and then the formation locations of the walls were generated vectorially from these points.

Zeng, Li, Yu, and Fu's (2019) study used both R2V and R3V datasets, focusing on segmenting rooms based on size and location through network prediction using colored and

black-and-white drawings as inputs. Various algorithms, including those from previous studies, were tested and compared in the segmentation process. Creating, segmenting, and predicting the plan layout was tested using the CNN algorithm (Figure 2.40).

Figure 2.40. This study compares the segmentation results of all the algorithms given above (Zeng et al.,2019)

Meanwhile, Chaillou (2019, 2020) used the NVIDIA Pix2pix algorithm for plan layout segmentation on raster images, generating layouts individually and in conjunction with urban-scale parcels. The training model for raster images included pixel-based training, reproducing both walls and various interior space combinations. Liu et al. (2020) trained the network on architectural and urban floorplans and eventually produced floorplans of homes in the training dataset (Figure 2.41).

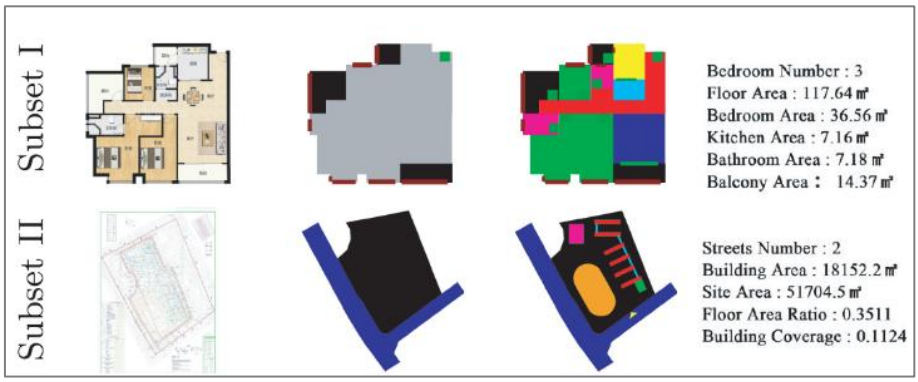


Figure 2.41. AutoALP plan and urban plan layout generation (Liu et al., 2020)

In analyzing datasets, it is common for researchers to try semantic segmentation methods when working with images. Although these methods are usually tested pre-processed,

Shekhawat et al. (2020) have described a process to create RFP layouts, in which points are quantitatively weighted according to their minimum and maximum scores. In developing the Graph2Plan algorithm, Hu et al. (2020) utilized the RPLAN. After detecting plan boundaries and aligning rooms on a dataset, the researchers combined this dataset with the GNN algorithm to create a user interface with a combination of graphical plan layouts (Figure 2.42).

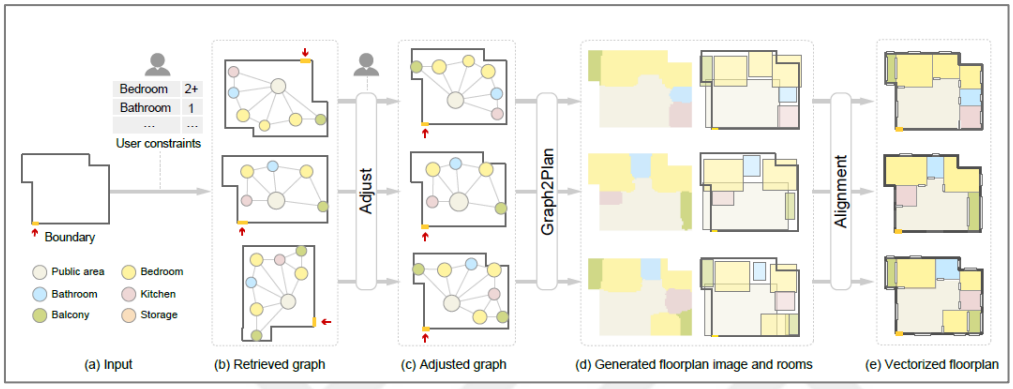


Figure 2.42. Deep neural network detecting boundaries and interaction with bubble diagrams (Hu et.al.,2020)

The HouseGAN study generates plans using the LIFULL house dataset(Nauata et al.,2020) (Figure 2.43). Images are used for boundary detection, and models are compared using testing generation methods based on Conv-MPN and GCN algorithms. In another study, the CNN network trained with the Layout GMN algorithm was found to produce a diagram-based layout in the backpropagation phase (Patil et al., 2021).

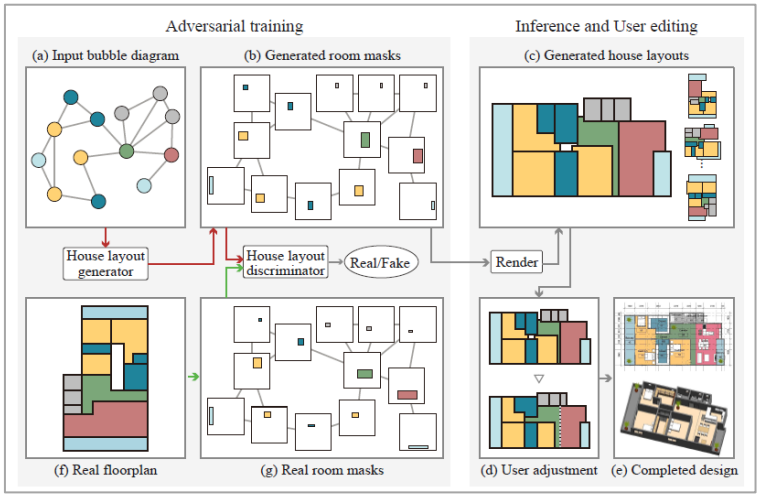
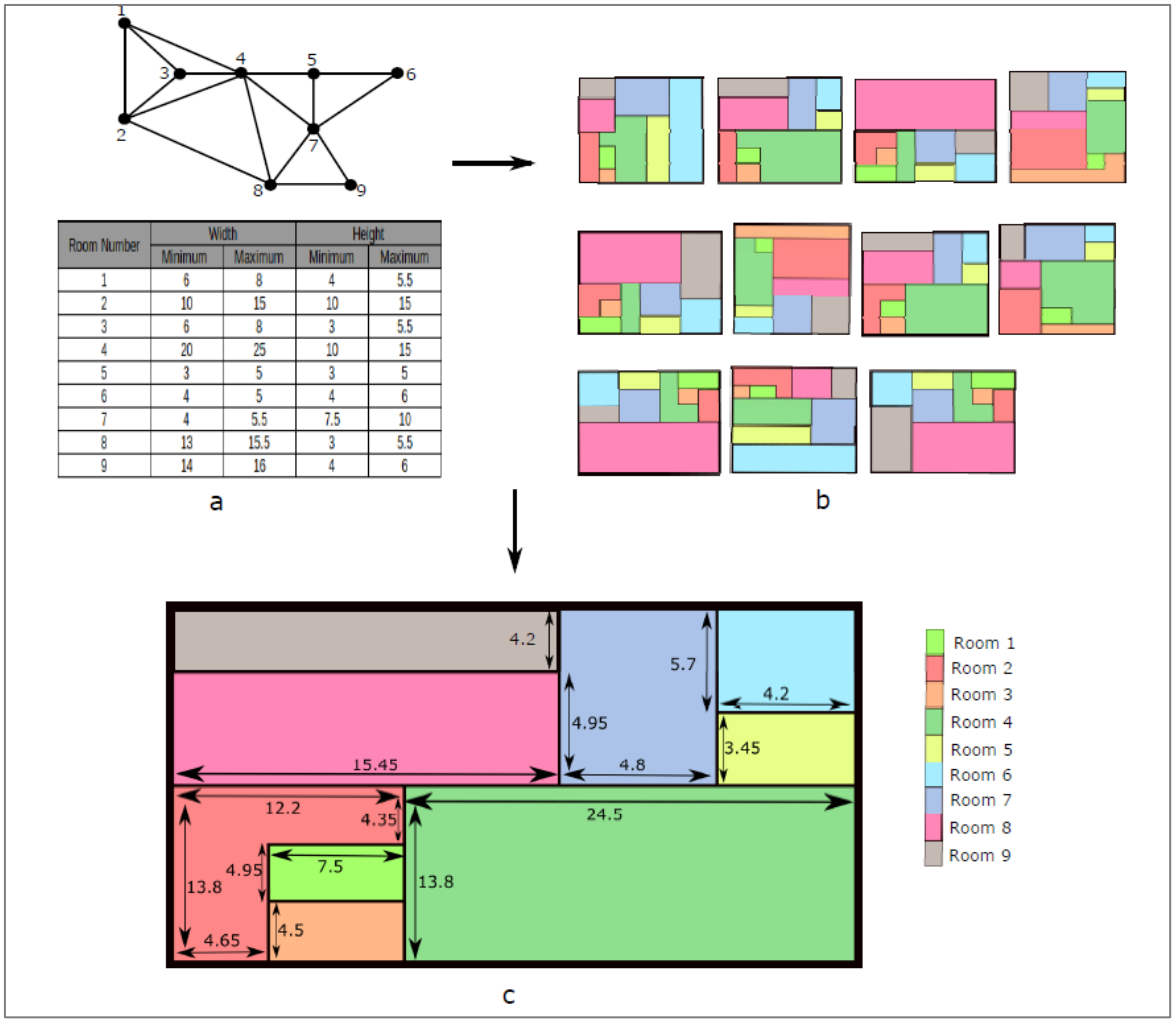


Figure 2.43. Interaction and generation of bubble diagrams and plans with HouseGAN (Nauata,2020)

A different research introduces the GPLAN algorithm, employing the linear optimization method (Shekhawat et al., 2020). Diverging from the authors' previous works, this study incorporates graph analyses with bubble diagrams and integrates training plans determined by their measurements on the network. Shekhawat et al. (2020) addressed this issue by combining RFP layout plan generation through graph layout representation. They analyzed the potential values of rooms and assigned numerical significance to the connections between nodes. (Shekhawat, et. al., 2020). Using this dataset, they integrated the generation of designs with GAN networks (Figure 2.44).



Room Number	Width		Height	
	Minimum	Maximum	Minimum	Maximum
1	6	8	4	5.5
2	10	15	10	15
3	6	8	3	5.5
4	20	25	10	15
5	3	5	3	5
6	4	5	4	6
7	4	5.5	7.5	10
8	13	15.5	3	5.5
9	14	16	4	6

Figure 2.44. RFP plan sistemi ve GANs ile üretilmiş plan layoutu (Shekhawat et al., 2020)

Unlike the architecture of the HouseGAN algorithm of Nauata et al. (2021), in this architecture, a segmentation mask algorithm is added to which doors can be added. The addition of doors brings about transitions between spaces and adjacency estimations. This

algorithm, which goes through a pooling process with the Conv-MPN algorithm, is called the HouseGAN++ algorithm. Luo et al. (2022) describe the properties of architectural floor plans used in architectural design processes and FloorplanGAN, a model proposed to generate these plans. FloorplanGAN utilizes the concept of differentiable imaging with the CNN algorithm to combine the tasks of vector generation and raster separation. Furthermore, this model focuses on raster design generation with GAN, making this method adaptable to other vector generation tasks. The study also developed a differentiable rendering algorithm that simplifies complex computations, focusing on the particular problem of floor plan generation (Figure 2.45).

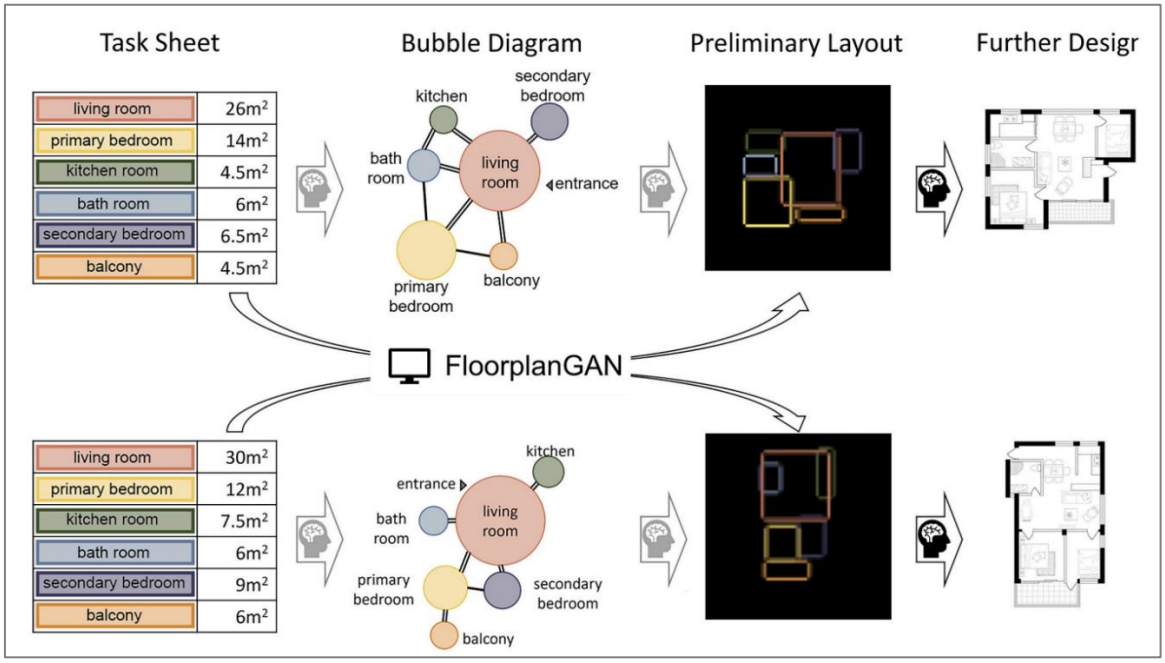


Figure 2.45. FloorplanGAN algorithm Luo et. al.(2022)

Rahbar et al. (2022) presented a novel hybrid method for generating 2D architectural layouts. This approach combines agent-based modeling with deep learning algorithms. The main goal is to allow designers to maintain high regulatory control over the generation process, thus ensuring that the generated results comply with the desired topological and geometric constraints. The featured hybrid method is a combination of two different elements. First, agent-based modeling constructs an agent-based model by mimicking the hierarchical stages, representing the bubble diagram that satisfies the topological conditions. A rule-based algorithm then transforms these bubble diagrams into heat maps. Second, the pix2pix algorithm applies a conditional GAN and deep learning approach to transform the heatmaps

into an architectural spatial layout. This process is supported by the cGAN algorithm, which is trained on a manually generated original dataset (Figure 2.46).

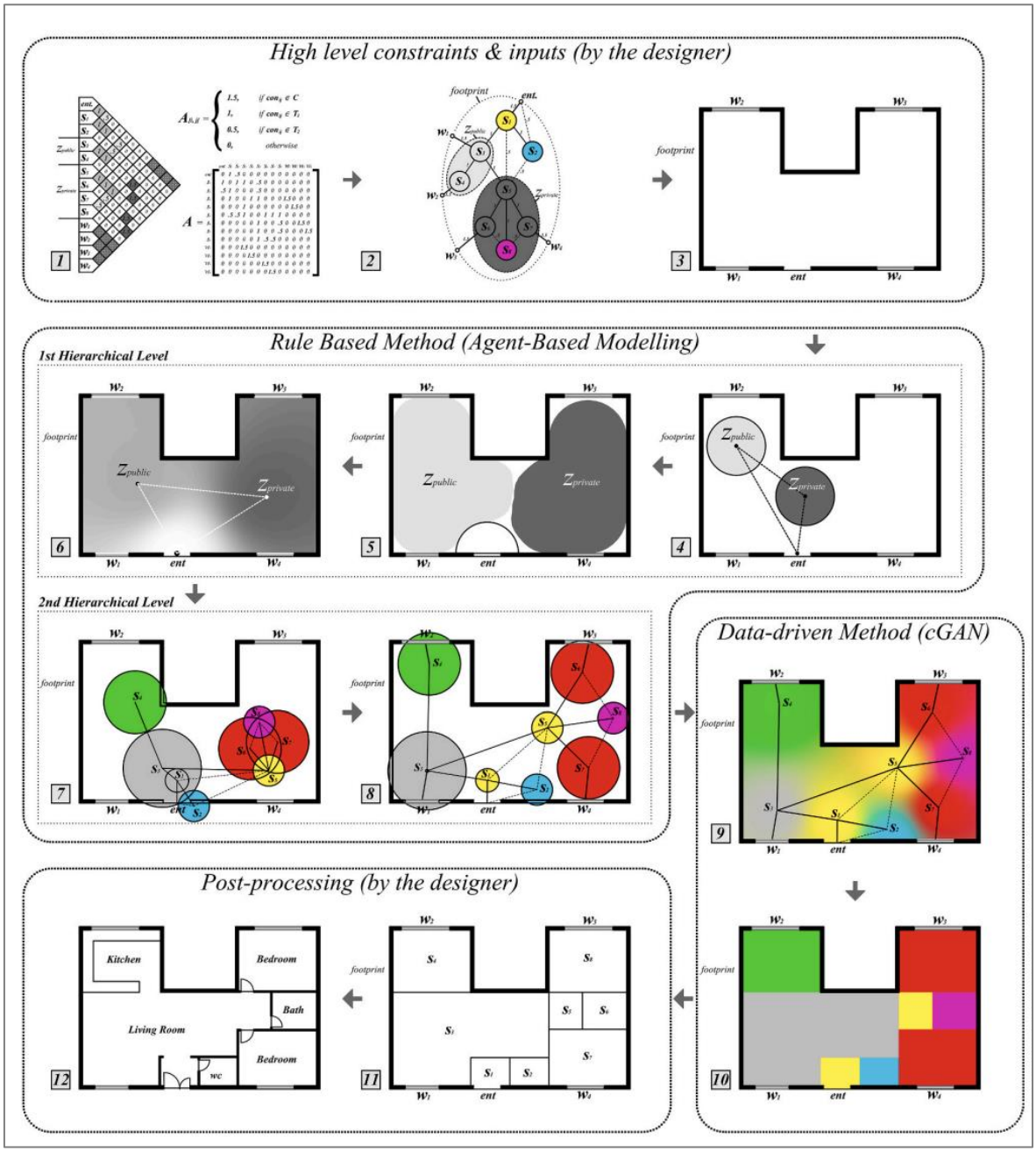


Figure.2.46. Rahbar et. al.(2022) Hibrid model Rulebased method and Cgan generation

Aalaei et al. (2023) proposed a data-driven approach for architects to generate architectural space layouts automatically using GAN networks. The developed pipeline technique allows architects to define building boundaries and bubble diagrams as vector data, exhibiting high accuracy in simulating the RPLAN dataset. It has become an essential tool in architectural floor plan generation. A comparison was also made between a plan layout and walls. First,

spatial plan layouts are generated, followed by a plan layout where the spatial plan layout is generated, and then walls are added.

3. METHODOLOGY

While the architects are designing with traditional methods, they are becoming a part of their designs, which they see as a trace of themselves. Over time, designers' and architects' digital technology tools have evolved from two-dimensional CAD drawings to BIM and computational design tools. With the technologies developing in parallel with these changing and discovered movements and theories, architects have started to change and grow their ways of thinking and making. Architects who think with the tool want to achieve generative design results with multiple alternatives, thanks to the potential to achieve fast results due to time constraints, while reaching the final design proposals.

3.1. Plan Layout Generation Through GANs

As a result of the literature research, 'spatial plan layout' is mainly generated through GAN networks according to the evaluations at the interface of machine learning and architecture. In this framework, DCGAN and Pix2pix(cGAN) algorithms were applied for plan layout generation methods. Two different datasets were used in these two algorithms. The purpose of using these different datasets is to more clearly identify the differences between the plan layouts emerging when TOKI plan images, which constitute an example of spatial planning with clear boundaries, and Open plan images, which have more flexible spatial boundaries, are trained to GAN networks. Since both algorithms are image-to-image translation methods, all generated datasets were converted into image files. For DCGAN, the segmentation process of the datasets was optional. However, for the Pix2pix(cGAN) algorithm, the segmentation process of the datasets was needed. The method of preparing the dataset for both algorithms differed. The evaluation of the new layouts generated was compared with the loss values obtained by the algorithms. Loss values in each algorithm differ. For this reason, the results of the algorithms are first compared within themselves. Afterward, a comparison was made to whether the generated image/predicted image was real or fake (Figure 3.1). The designer's decision-making process regarding the final plan layouts generated through GAN networks varies. This is because, even if the output/generated/predicted images are close to 'real' according to the network results, they may not be a preferable plan layout proposal for the designer.

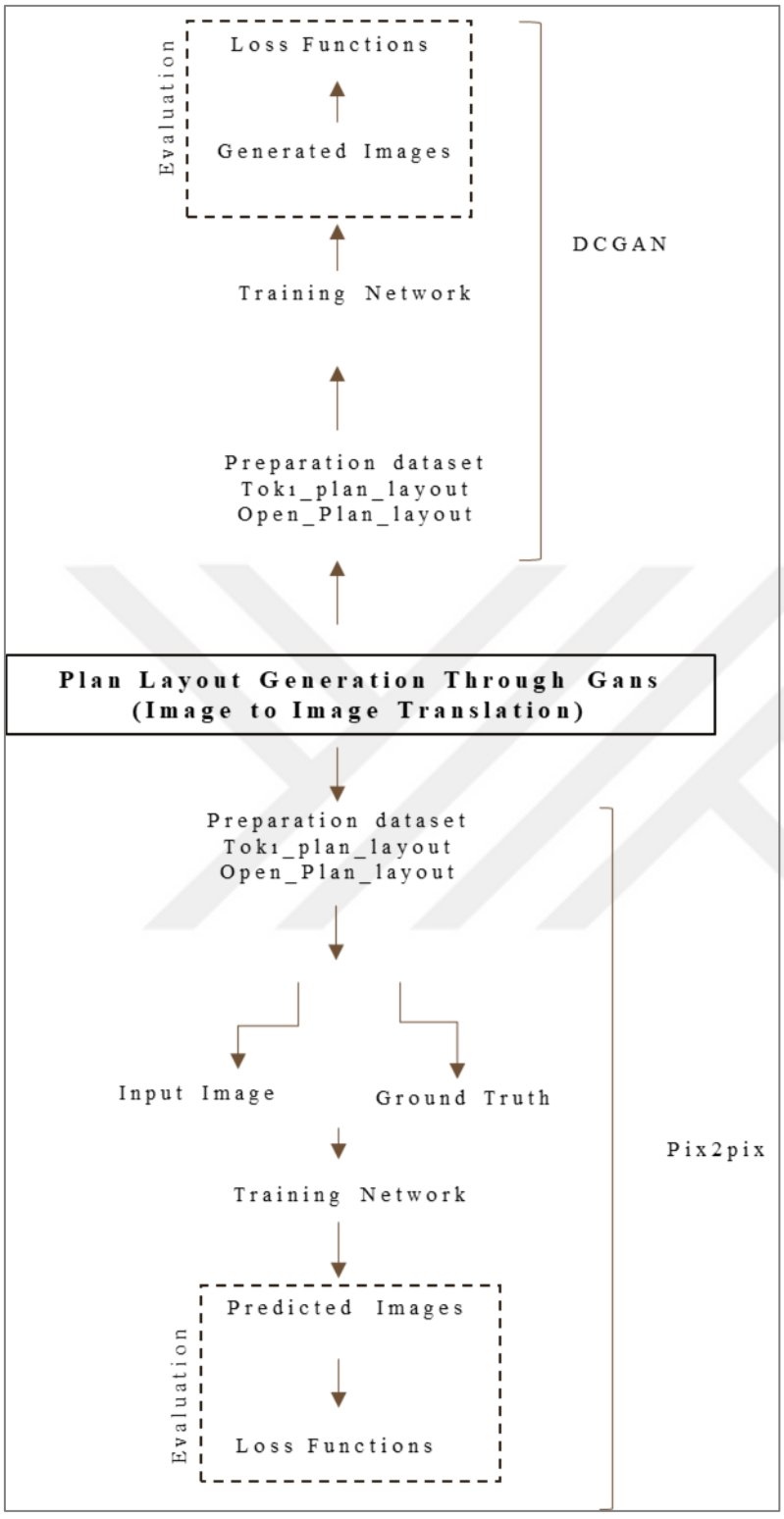


Figure 3.1. Plan layout generation through GANs (prepared by author)

3.2. Decision Making Process

In this study, AI tools were again used to reinterpret the images produced by GANs. However, the architect's evaluation regarding the interpretability of the plan layouts created

in this process was needed. This section aims to re-evaluate and concretize the plan layouts with simulations made with computational design tools. Afterward, the Rule-based classification method, also an AI algorithm, was preferred to evaluate the alternatives in decision-making.

3.2.1. Plan layout detection through object detection

Each predicted image, which is close to the ground truth, may not match the criteria that the architect may prefer. To detect these cases, the generated plan layouts through different datasets are analyzed with other systems in addition to evaluating the machine's learning metrics. This system was determined to be the 'object detection' algorithm. They were re-inspecting generated/predicted images through GANs with another artificial intelligence algorithm, 'Object Detection,' to determine whether the spaces in the plan layouts created were correctly defined. The data obtained from these predicted/detected images were transferred to the Rhino/Grasshopper program, a computational design tool with BBOX detections and .json data. The aim is to make the defined plan layouts measurable and re-evaluable by architects according to specific spatial design criteria (Figure 3.2).

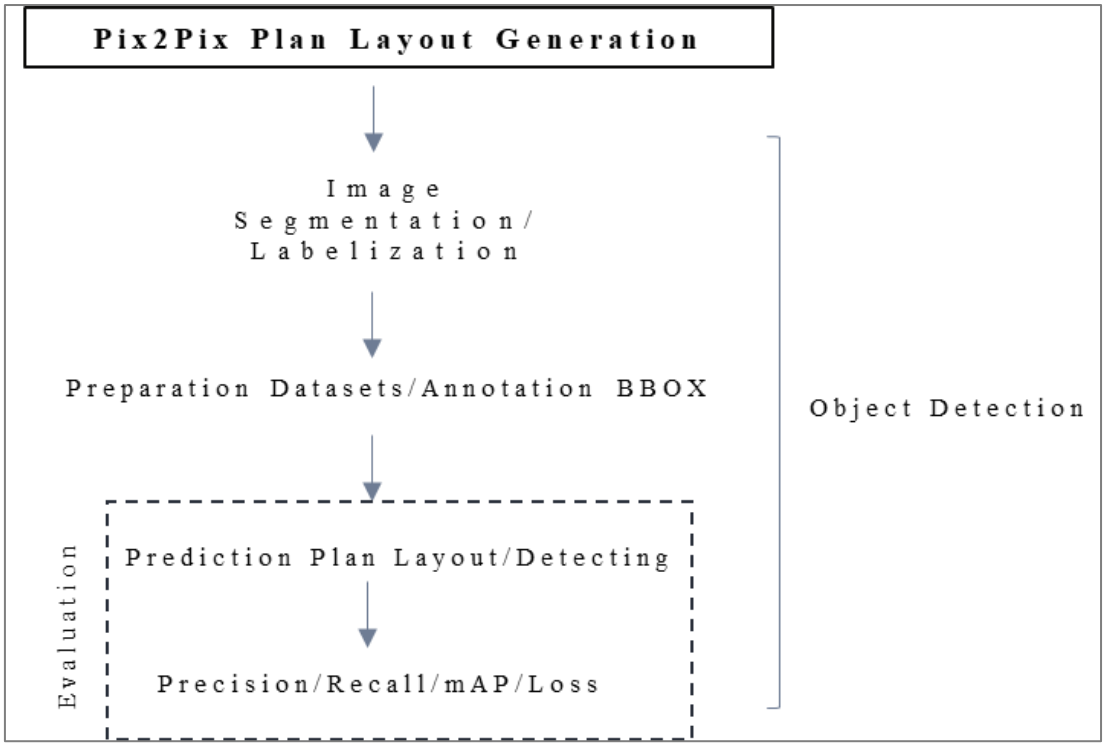


Figure 3.2. Object detection process of generated images (prepared by author)

As a result of the predictions, by checking the images generated by GAN networks with another AI algorithm, object detection, it was determined that the AI could make spatial descriptions of the generated images. However, for the designer, these predictions should have meaningful results. It is aimed to make the generated plans measurable in a concrete way based on specific parameters and to facilitate the designer's choice.

3.2.2. Analysing Plans

The BBOX coordinates (x,y, values width, and height lengths) and midpoints are included with the predicted room properties (class) in the JSON data format. Thus, In the Rhino/Grasshopper environment, geometrically transforming these detected plan layouts was quantified and materialized. After the plan layouts were converted into tangible plan layouts, analyses were performed to graphically express the enclosure, openness, circulation connections, and spatial connections of the plan layouts according to their spatial quality values. A final evaluation was made according to these data analyses (Figure 3.3).

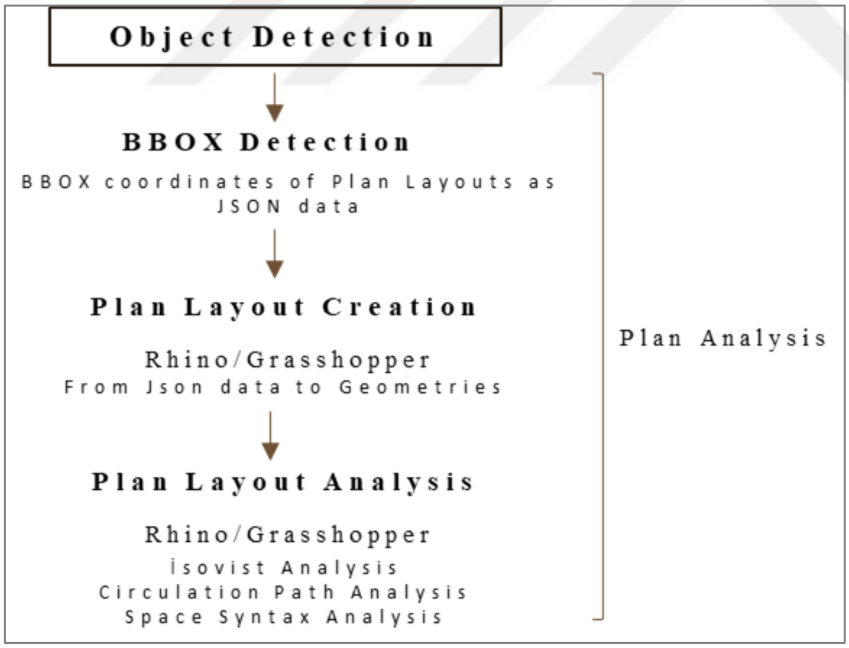


Figure 3.3. Plan analysis workflow (prepared by author)

To summarize, after conducting an analysis of the plan layouts, this evaluation graph was processed in great detail, and numerical values were included in a table that was created for the architect to make a decision. The result values of the analyses are recorded as parameters in the data table.

3.2.3. Classification

Mainly ML algorithms are used for as decision-support systems. These decision support systems are tools that help us in situations where we cannot be sure of their accuracy or in processes where we need to make quick decisions due to time constraints. With the rapid expansion of automation systems in design processes, 'decision support systems' from AI systems have started to be used frequently. The main structure of this process is the re-evaluation of the plan layouts generated within the framework of specific problems in line with the designer's suggestion. Re-assessing the analysis of the design is provided by including expert opinion. Decision support systems, especially with the advancement of technology, have been used to process the information obtained and to act quickly to make the right decision. AI supported classification methods create certain classifications through algorithms such as “decision trees, K-nearest neighborhood/KNN, and fuzzy logic.” Although these classification methods give positive results, to proceed within the framework of specific rules, the rule-based classification model introduced by Holte (1993) is a more precise system created for multi-class systems compared to other systems based on more than one probability and situation.

Our study transferred the BBOX predictions obtained through Object Detection to the Rhino/Grasshopper computational design tool. With the Jswan plug-in, we created plan layouts from the BBOX detections we received in JSON format. With the simulations made in the Rhino/Grasshopper program, the plans were reclassified by evaluating whether the plan layouts have specific spatial qualities. The new data obtained here was revised with expert opinion and transformed into a decision support system where the end users can easily make decisions with the 'Rule-based classification' method, which is an open system in terms of "interpretability" and "explainability" of the resulting plans (Figure 3.4).

The table values resulting from the analysis were utilized for the rule-based evaluation technique. By writing the rules in Python software language in the Google Colab environment, the designer can reach the plan layout option or options he/she wants to obtain by writing the desired conditions (with evaluations of different parameters) from this knowledge base.

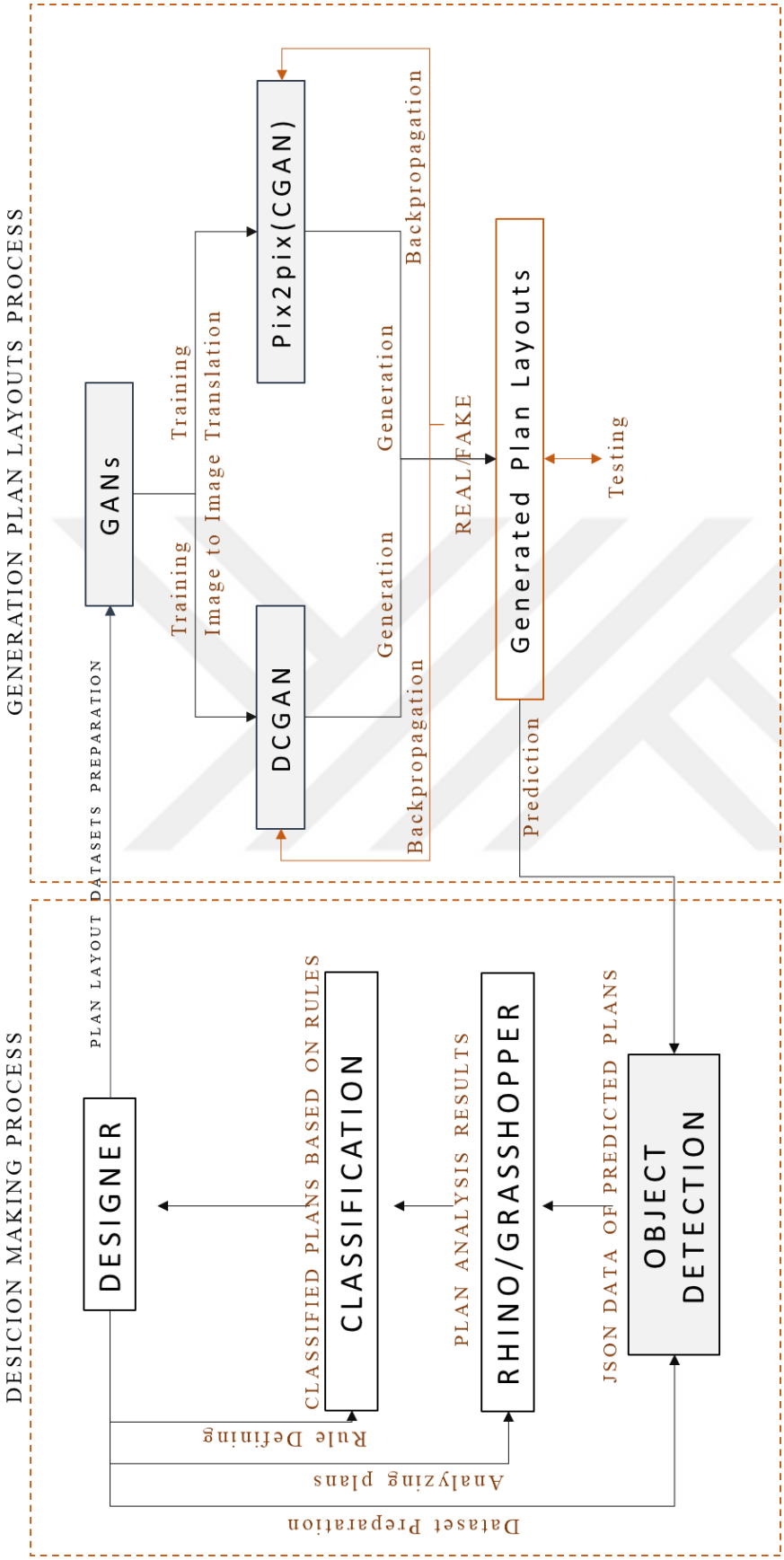


Figure 3.4 Workflow chart from generated image through decision making (prepared by author)

4. IMPLEMENTATION LAYOUT GENERATION THROUGH GANS

The data processing differs depending on the architecture of each machine learning algorithm. For supervised learning and unsupervised learning, the data processing of the image also differs. In unsupervised learning models and algorithms such as DCGAN, knowing the images as RGB and preparing them with ready-made images in .jpeg/.png formats is a valid and sufficient process for training the network without processing the data. However, for a supervised learning process such as CGAN (Pix2Pix), it is essential to develop a learning method for the images produced as a result of training by dividing the data into two parts: input and ground truth.

4.1. DCGAN Implementation

In this study, DCGAN, the most straightforward GAN architecture described earlier, is used. The DCGAN algorithm, with its two main components, generator and discriminator, is used to ascertain whether the generated images are real or fake. The datasets were prepared in color, and for clearer detection, the TOKI plan house was created using the initial segmentation of the dataset. Training plan layouts of different typologies are compared with the results. The architecture of the DCGAN algorithm is demonstrated in Figure 4.1.

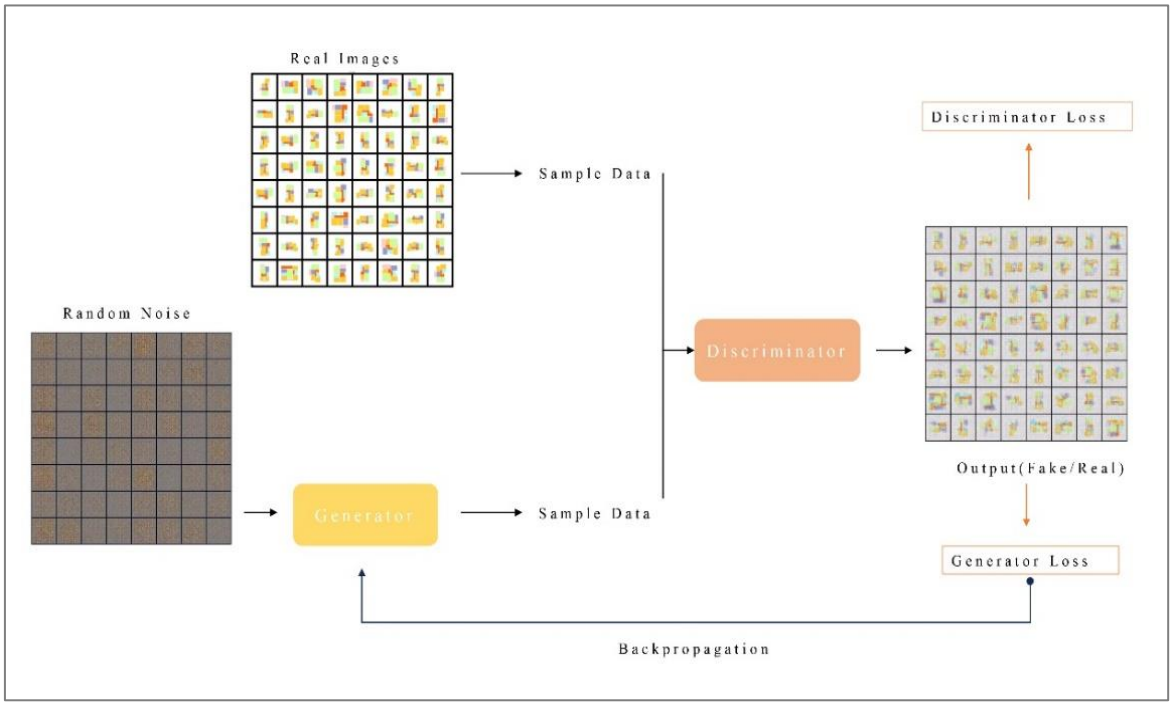


Figure 4.1. The workflow of DCGAN (prepared by the author)

The algorithm was used to train the algorithm on these plans, and then datasets of labeled rotated 90, 180, and 270 degrees and mirrored images were created that could also be used in the Pix2pix algorithm. The experimentation continued on the 'Google Colab' platform, utilizing the GPU operating system to optimize computing speed. Google Colab provided convenient access to our image data saved on the SSD after performing data augmentation processes (see Table A.1). Afterwards, the chosen method used DCGANs to create designs, implemented through a Tensorflow connection. The approach differentiates GANs from other DNNs by establishing connections between the generator and discriminator. Subsequently, the dataset was established, and the network began the training process. As a result, the network was retrained using an expanded dataset consisting of 157 plans. The modified model was again subjected to epoch values of 50, 80, 150, and 500.

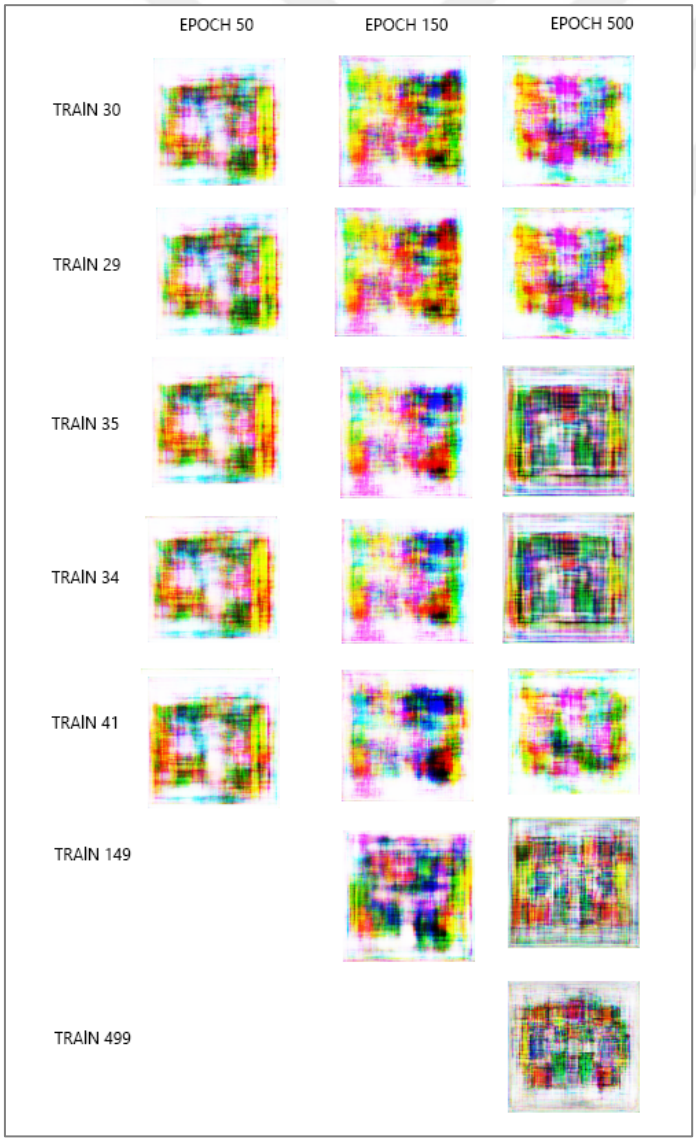


Figure 4.2. Tried generation on 157 plans (prepared by author)

The experiment used the Google Colab tool, integrating the GPU platform to improve processing performance. Google Colab efficiently facilitated access to our image data stored on the drive following the execution of data augmentation operations. DCGANs were subsequently utilized in the selected strategy to generate plans using a Tensorflow linkage. After defining the dataset, the network undertook an extensive training procedure. The investigation involved examining epochs of 50, 80, 150, and 500 to determine where the training set becomes saturated. This is seen in Figure 4.3. Despite training the model for a substantial 499 epochs, achieving accurate generation was still impossible due to the limitations of the small dataset.

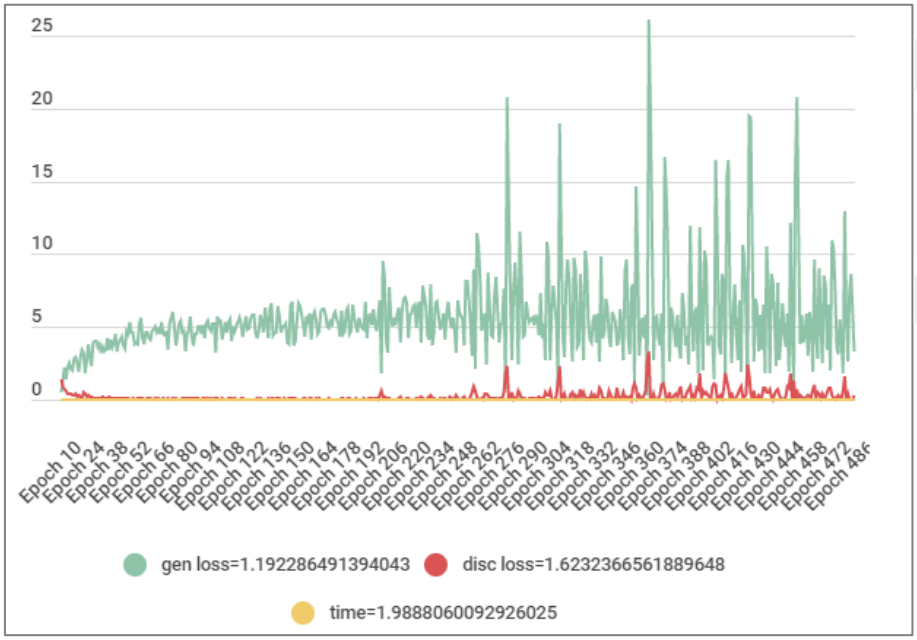
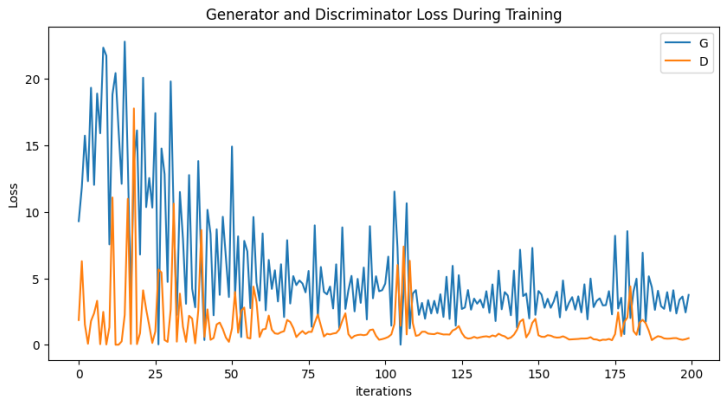
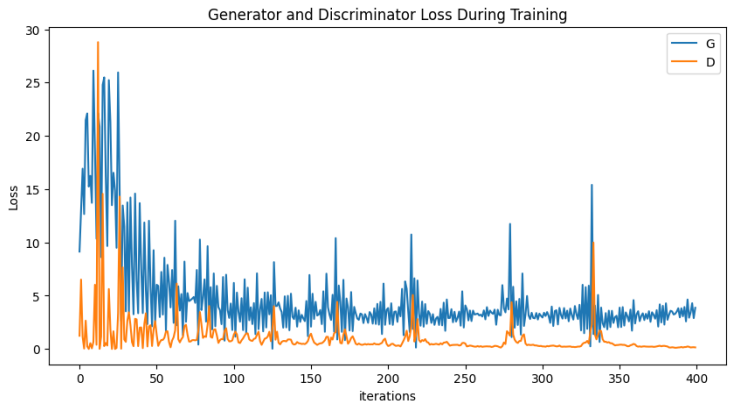
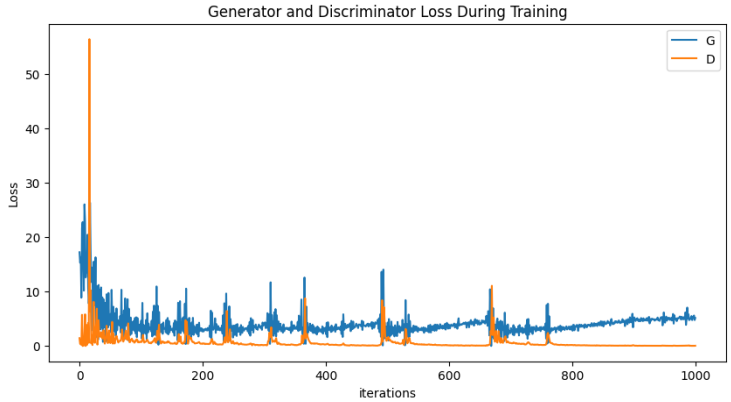
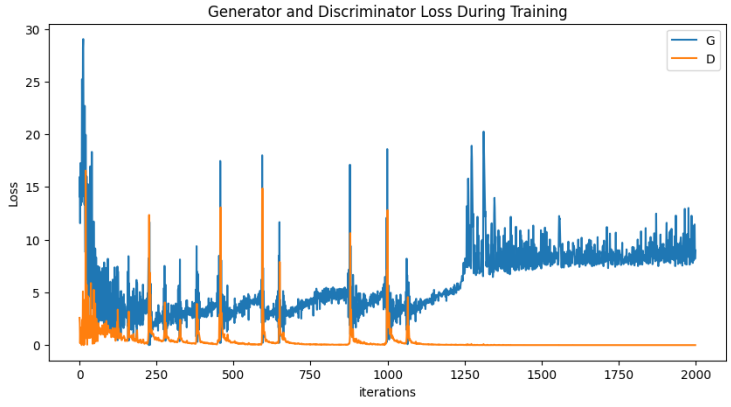


Figure 4.3. After the training process of the network with an epoch value of 500, generation loss function and discrimination loss function values (prepared by author)

4.1.1. TOKI plan dataset training through DCGAN

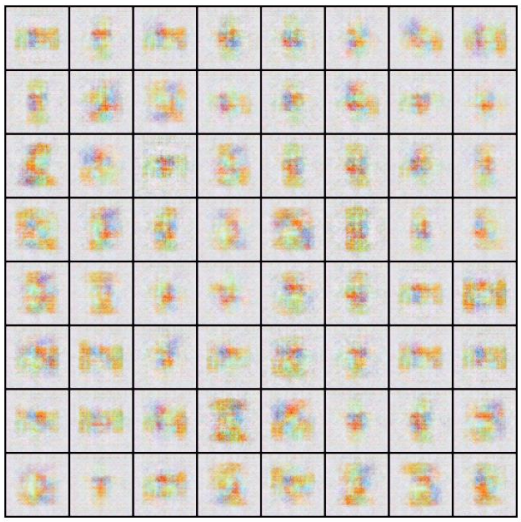
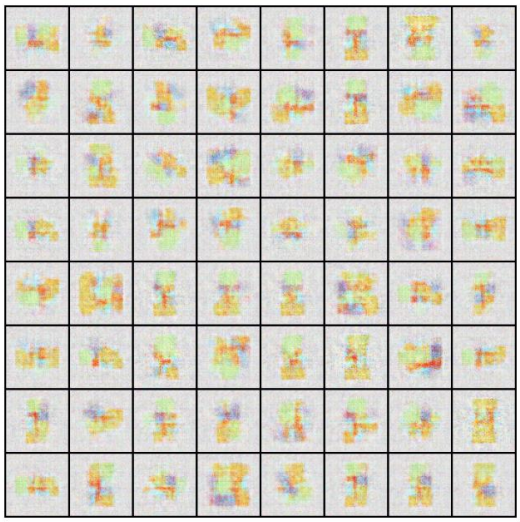
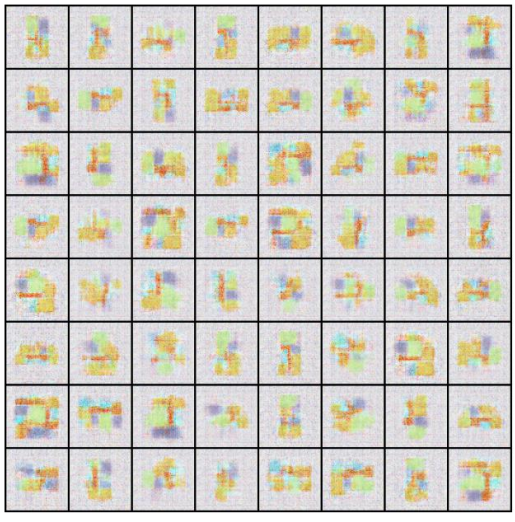
Plan layouts reproduced from one unit of the closed plan layouts of the TOKI plan dataset were trained together with DCGAN. As seen in the open_plan layout training, the network training runs more smoothly as the epoch values increase starting from 100 epochs, and we can understand this situation from the graphical functioning of the Generator and Discriminator Loss values. Especially after 500 and 1000 epochs, the Discriminator is in a balanced graphical value, while the generator is clearer at low values. However, it has been noted that it can generate images that differ from the input images. (Table 4.1).

Table 4.1. TOKI plan datasets training results

Epoch	Time	Generator and Discriminator Loss During Training
100 epoch	28 dk	
200 epoch	47 dk	
500 epoch	1 sa 36 dk	
1000 Epoch	4 sa 50 dk	

After 500 epochs, the images generated by the network became clearer. At 1000 epochs, the images are produced more clearly and generated differently (Table 4.2).

Table 4.2. TOKI plans datasets training results

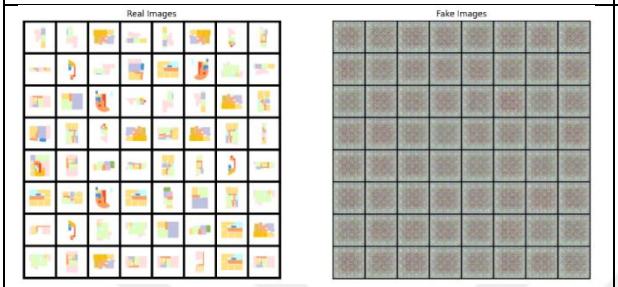
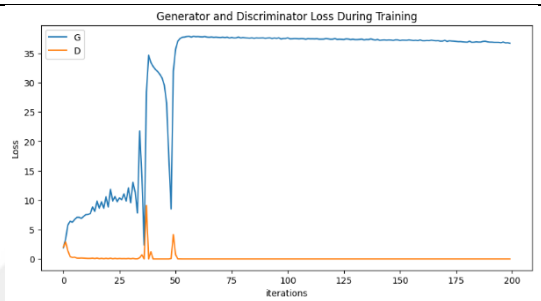
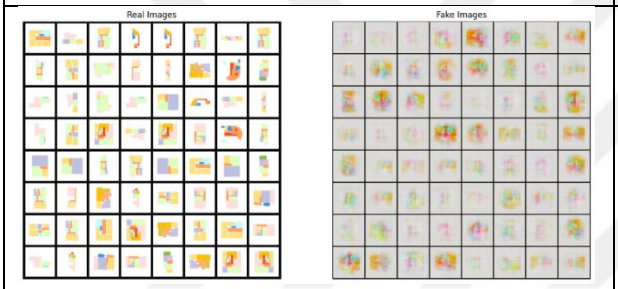
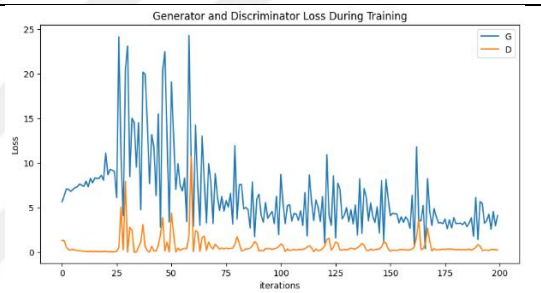
100 Epoch	200 Epoch
	
500 Epoch	1000 Epoch
	

4.1.2. Open house plans training through DCGAN

The dataset was trained using this DCGAN algorithm. The images trained at 100 epochs during the first trial were retrained by reducing the noise function and more favorable results

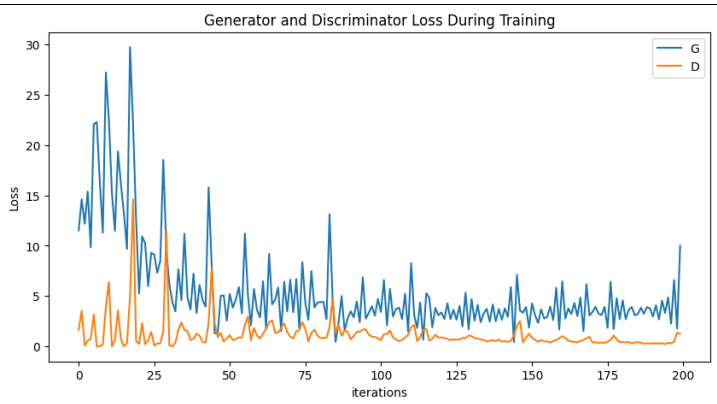
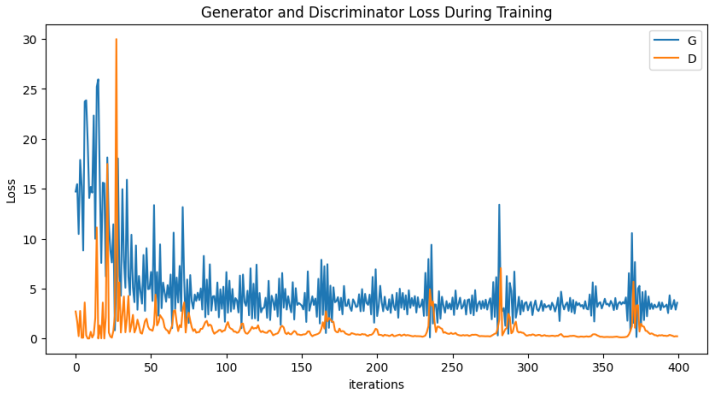
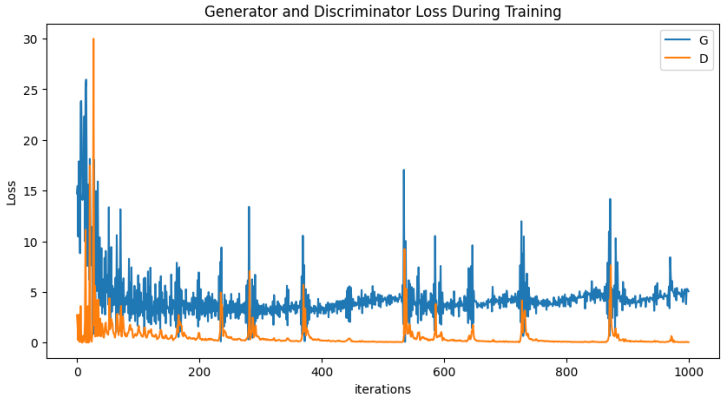
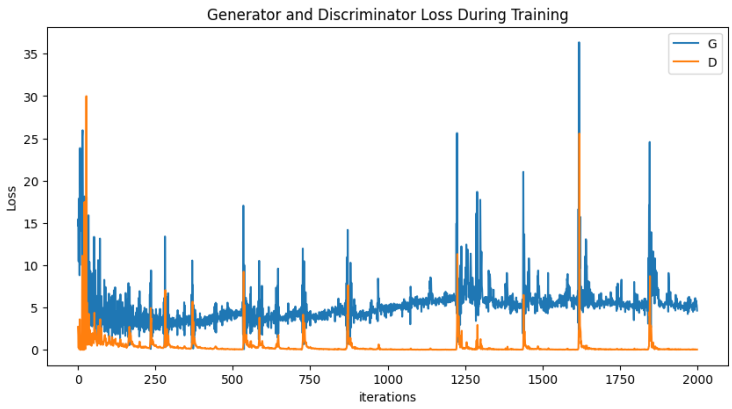
were obtained. The graphical values indicate that the value of the Generator and Discriminator variables are closely aligned (Table 4.3).

Table 4.3. DCGAN 2.case results

Real and Fake Images	Generator and Discriminator Loss Functions
	
	

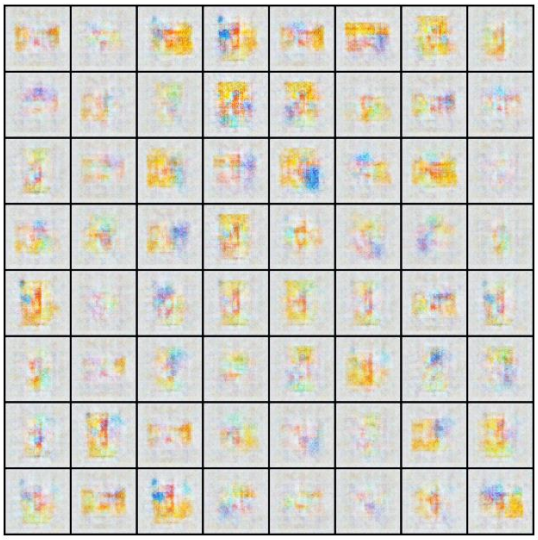
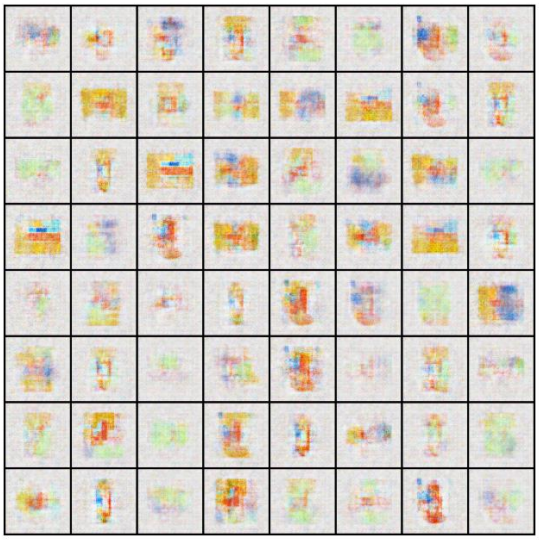
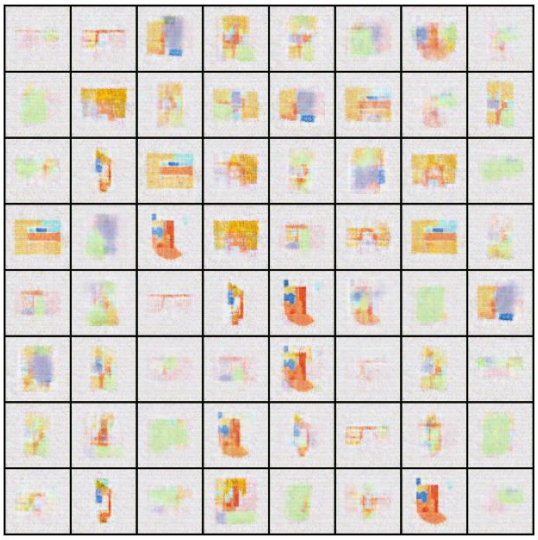
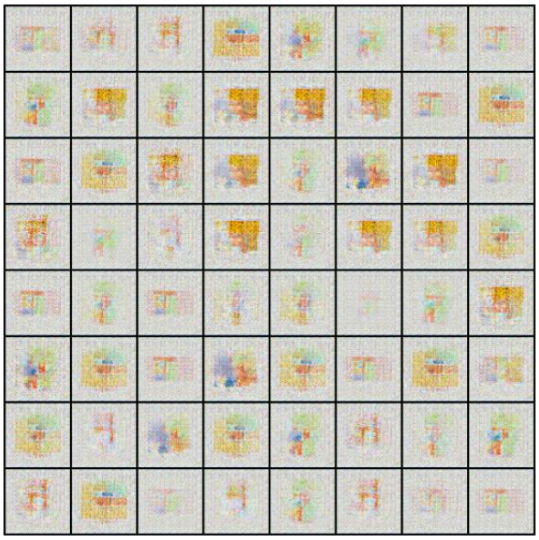
Experiments can be performed at high epoch values. However, the first change in the code was tried by decreasing the random noise vector so that a more realistic image could be generated. Then, the learning rate of the network was increased slightly above its minimum value. The "learning rate" value was avoided to reach its maximum value. Because the network training methods start to vary after overfitting. The training process was prolonged after these values changed. “The input data consists of latent vectors, z, sampled from a conventional normal distribution, and the final result comprises a 3x64x64 RGB image" (URL-7). From 100 epochs, it was changed to 200 and 500, and the change in loss function values is expressed in Table 4.4 along with the training process. The training time increased proportionally from 100 epochs to 500 epochs of the trained dataset. The best generated images are the ones with low generator loss and high discriminator values, which are close to the real image. This is because, when the generator loss is low, it can produce an image that is not very different from the real image. The discriminator should be balanced during the network training period. Because in order to distinguish the generated image, it must be at low and balanced values.

Table 4.4. Open plan datasets training through DCGAN

Epoch	Time	Generator and Discriminator Loss During Training
100 epoch	36 dk	
200 epoch	1 sa 4 dk	
500 epoch	2 sa 36 dk	
1000 Epoch	3 sa 50dk	

Within the framework of this logic, according to the inference made from the training results at 200 epochs and 500 epochs, it is seen that the network can produce and distinguish healthier and closer to authentic images with increasing value. Table 4.5 shows the resulting generated images.

Table 4.5. Open plan datasets training results

100 Epoch	200 Epoch
	
500 Epoch	1000 Epoch
	

The clarity of the images obtained with the resulting loss values is also close to reality. A similar training method and the same algorithm were tested again with different training sets and the same values.

4.2. Pix2pix open plan house layouts generation

For the use of the dataset we created in the Pix2pix algorithm, we trained the network with the dataset we created, not with the MNIST dataset, which uses open-source code. The transition processes of the data we created in the U-NET and Patch-GAN sections, which are two essential components of the Pix2pix algorithm, are critical in terms of the compatibility of the images given by the separation and correct identification of these algorithms (generator and discriminator). The functioning process of the Pix2pix algorithm with the data we used is demonstrated in Figure 4.4.

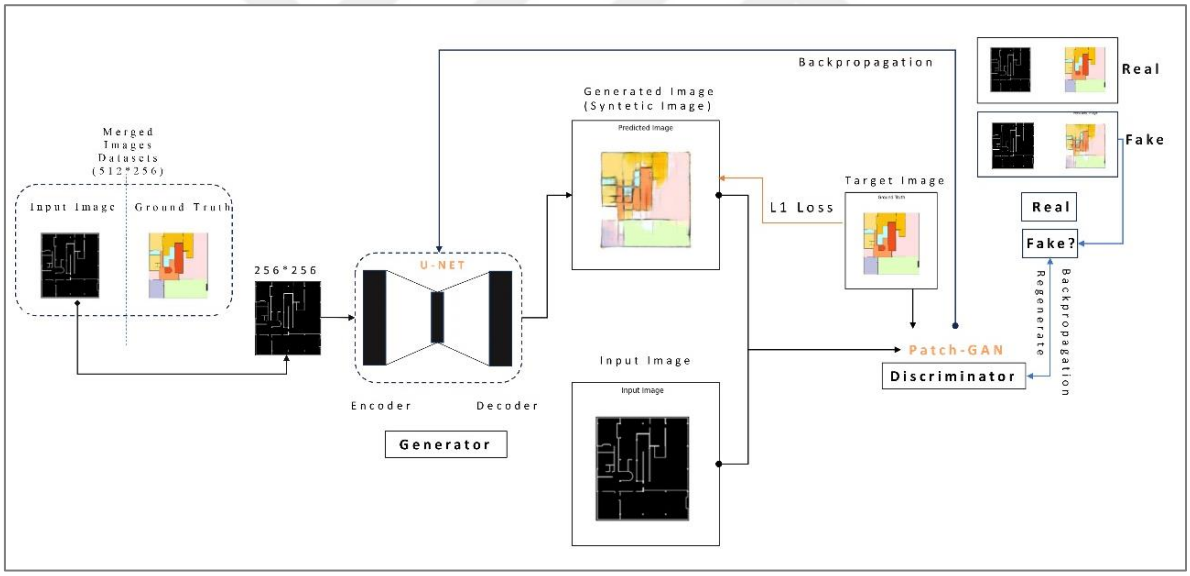


Figure 4.4. How the dataset used with the Pix2pix algorithm works with the algorithm (prepared by author)

Creating the output Loss values as metric values between the input, ground truth, and predicted image more clearly expresses whether the algorithm has undergone the correct training process. Afterward, we saved the images produced by the trained network at each epoch value or specific interval epoch values during the training process as .h5 extension. We created recalling process to re-test the trained images (Figure 4.5).

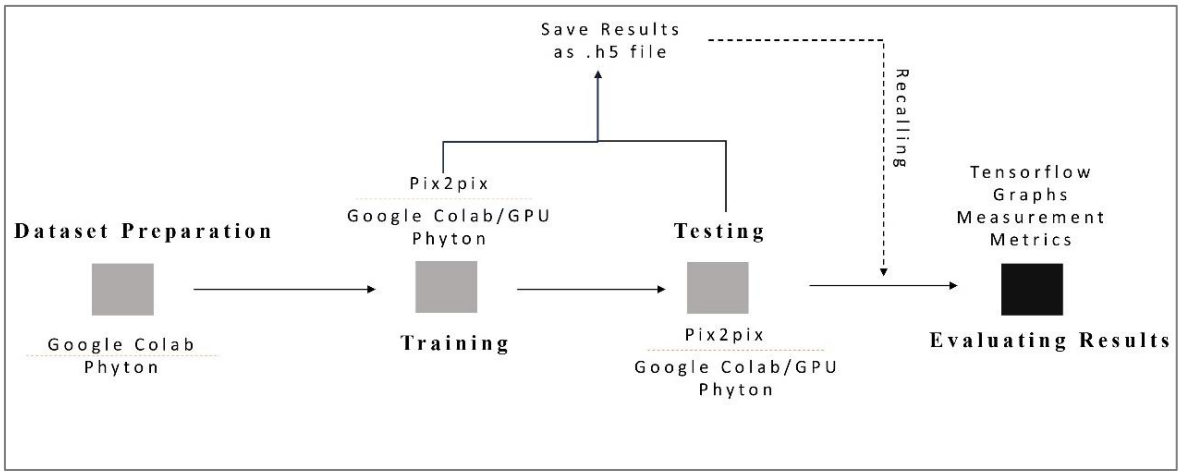


Figure 4.5. Process diagram of how the pix2pix algorithm is trained and how the post-training process takes place (prepared by author)

Generator

Since the generator is created from U-net architecture, we see the upsampling and downsampling process here (Table 4.6.).

Table 4.6. Generator definition (downsampling and upsampling)

```
def Generator():
  inputs = tf.keras.layers.Input(shape=[256,256,3])
down_stack = [
    downsample(64, 4, apply_batchnorm=False), # (bs, 128, 128, 64)
    downsample(128, 4), # (bs, 64, 64, 128)
    downsample(256, 4), # (bs, 32, 32, 256)
    downsample(512, 4), # (bs, 16, 16, 512)
    downsample(512, 4), # (bs, 8, 8, 512)
    downsample(512, 4), # (bs, 4, 4, 512)
    downsample(512, 4), # (bs, 2, 2, 512)
    downsample(512, 4), # (bs, 1, 1, 512)
  ]
  up_stack = [
    upsample(512, 4, apply_dropout=True), # (bs, 2, 2, 1024)
    upsample(512, 4, apply_dropout=True), # (bs, 4, 4, 1024)
    upsample(512, 4, apply_dropout=True), # (bs, 8, 8, 1024)
    upsample(512, 4), # (bs, 16, 16, 1024)
    upsample(256, 4), # (bs, 32, 32, 512)
    upsample(128, 4), # (bs, 64, 64, 256)
    upsample(64, 4), # (bs, 128, 128, 128)
  ]
```

The generator is structured based on U-Net architecture. Each frame in the encoder starts with a convolutional layer and then goes through a batch normalization process and a leaky

ReLU activation. The decoder comprises a transposed convolutional layer, batch normalization, the dropout method, and ReLU activation. Following the U-Net architecture, bypass connections are used among the encoder and decoder.

Discriminator

The Parser utilizes the PatchGAN model in this situation. Every block in the Parser undergoes convolutional processing, then goes through the batch normalization process, then leaky ReLU activation. The final layer outputs a shape of (batch_size, 30, 30, 1). Each 30x30 patch of the final result in this architecture classifies a 70x70 section of the input image, known as PatchGAN. This procedure differentiates the input image from the actual target image. An evaluation compares the input image, meant to be misclassified, with the created image produced by the generator (Figure 4.6). To handle these inputs, the code concatenates them together using the operation " tf. concat([inp, tar], axis=-1)".

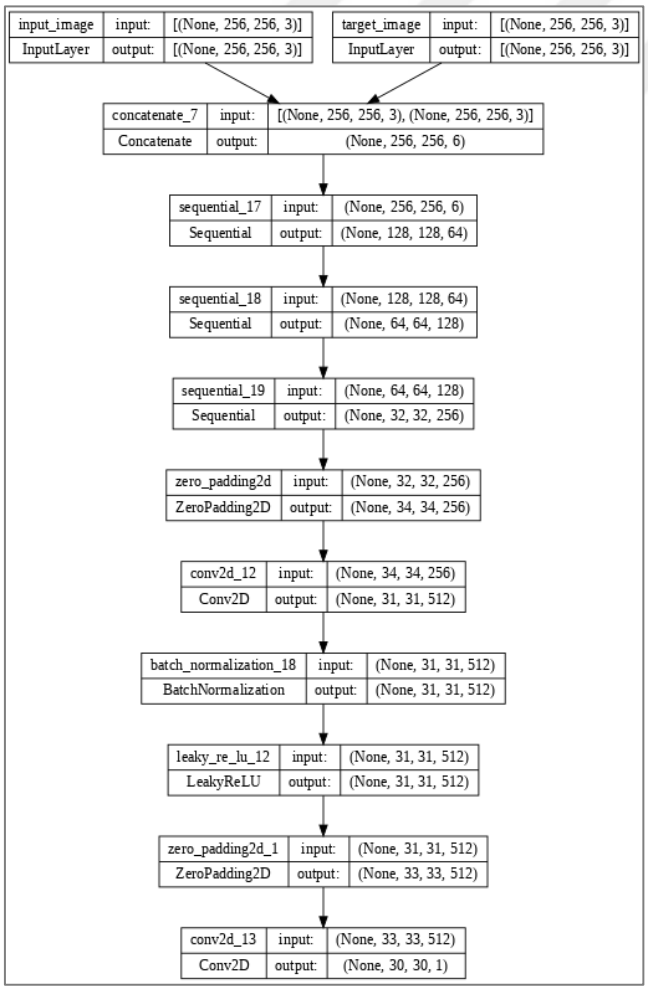


Figure 4.6. The Discriminator architecture of pix2pix algorithm

4.2.1. Open plan datasets preparation

This stage varies since each algorithm has a different way of processing the data or training the network. Supervised and Unsupervised learning methods can sometimes be done by simply scaling and changing the dimensions of the data without processing the data at all, and sometimes, the data must be broken down and converted into a specific format that the network can train with the identification process.

While creating the "Open Plan" dataset, open plan layouts and raumplan layouts were searched online. After collecting the visual data of these plans (.jpg or .png formats), the data was prepared to train the network with the Pix2pix algorithm. The labelization stage was started by identifying and separating the images for the semantic segmentation process. In the sample data preparation methods, in order to produce images from real images, the areas to be generated must be colored and defined. This identification process was realized by coloring the spatial organizations according to the functions of the plan layouts. In decomposing the plan layouts, negative image visuals defined by interior and exterior walls define the area boundaries that spatial boundaries can produce (Figure 4.7).

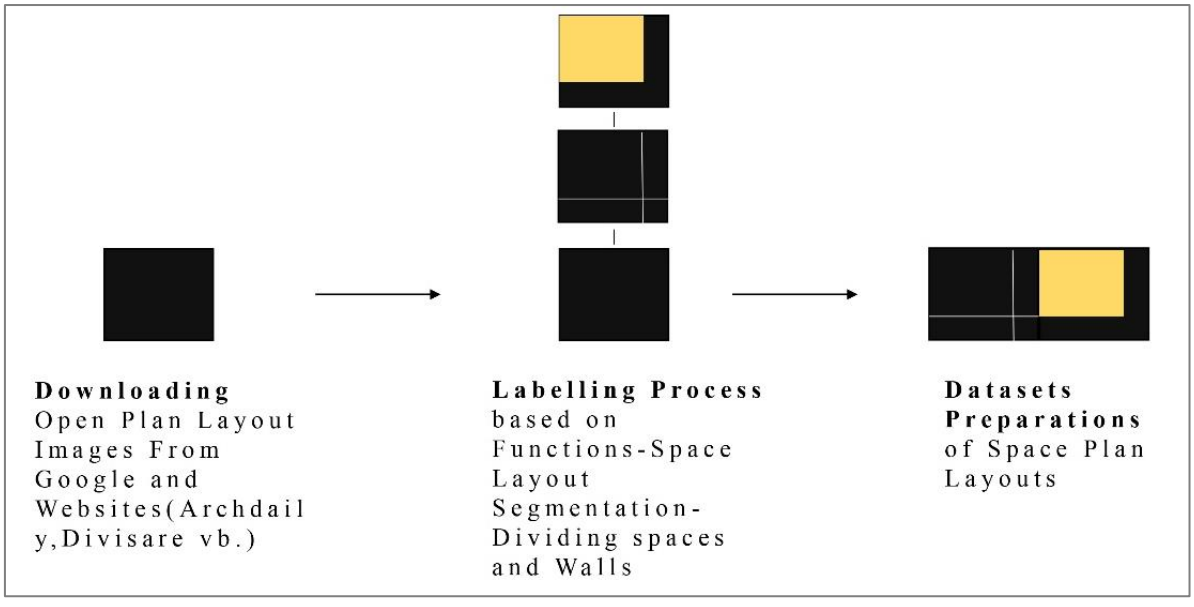
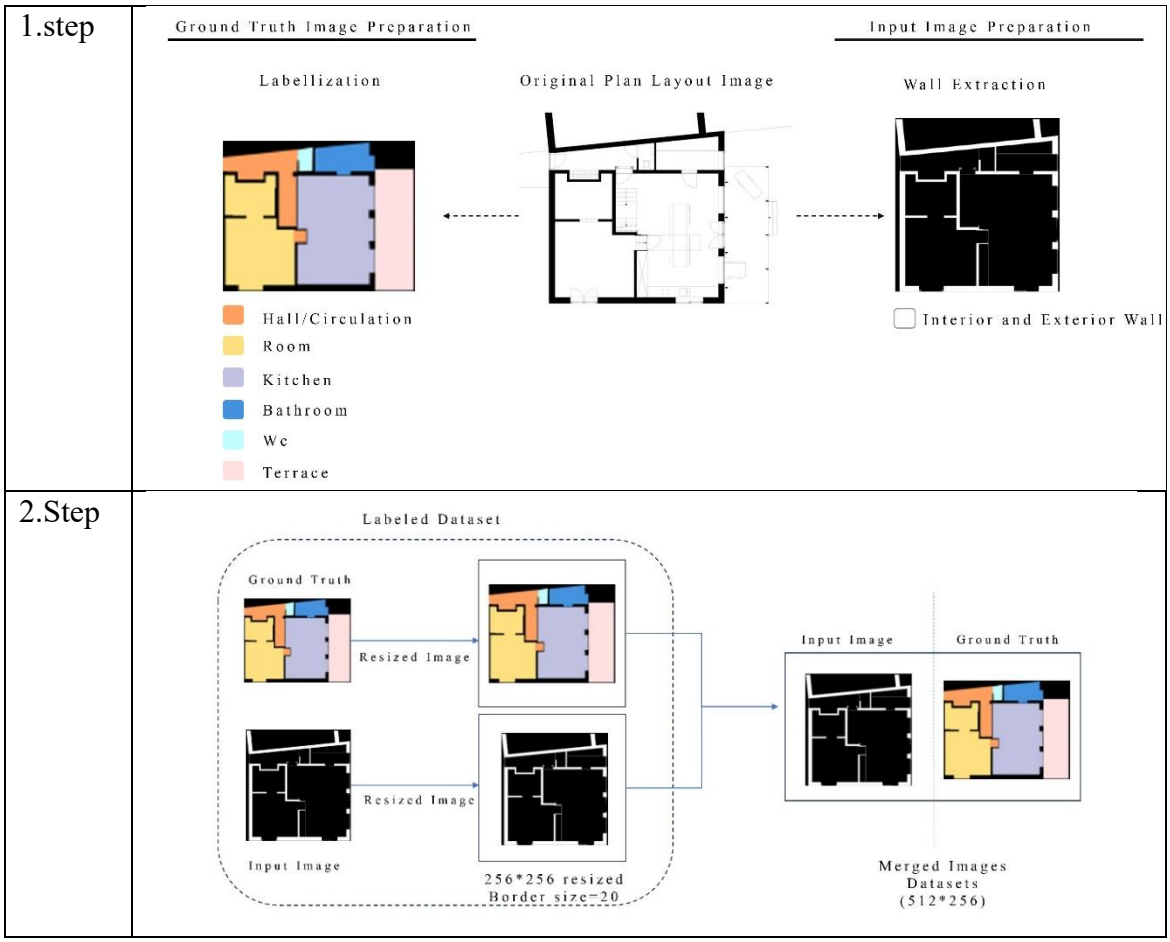


Figure 4.7. Process of datasets preparation (Prepared by author)

Giving spatial boundaries rather than the whole plan boundary also differs from the plan types in our previous study. Because in open-plan layouts, interior partitions (walls) are not included as boundaries. It is essential to differentiate this variable in the training dataset. In

the first stage of Table 4.7, the defined spaces, which are separated according to the functions of the spaces in the labelization process, are an essential stage for generating ground truth.

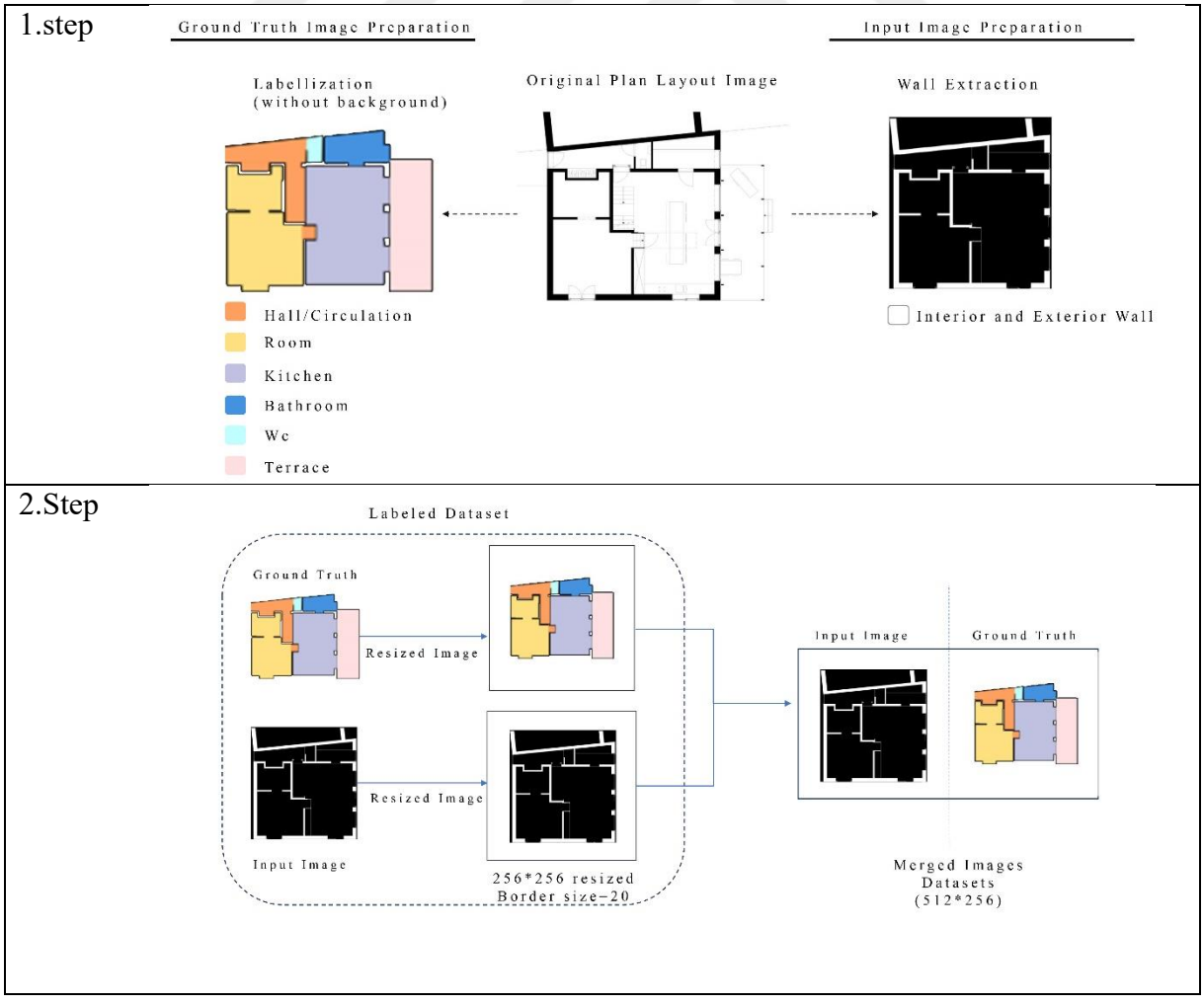
Table 4.7. Open plan dataset preparation (case 1 and case 2) (prepared by author)



The plan layout images that the algorithm tries to generate are based on these ground truth images. The white walls on the black background, defined as input images, are included in the dataset to define the area and area boundaries where the space can be generated. In the second stage, these parsed images had to be scaled, rescaled, and included within certain boundaries. This was done because the pix2pix algorithm requires the images to be of the pre-defined size to work. One of the important methods of GAN networks is to reduce the images to their pixels by fragmenting and dividing them and then recombining these pixels. Thanks to the U-net algorithm, we can see this fragmentation and reassembly. U-net and Patch-GAN algorithms, which are important components of the Pix2pix algorithm, provide an essential factor in understanding how the images generated are processed and how they are generated. In this context, the process of fitting the images into the 256*256 scale that is processed in the algorithm without going through the process of separating the prepared

images as "Ground Truth and Input Image" was realized through the Python code. After this process, the 20cm images were scaled again with Python code after it was determined that creating an outer border inside the image effectively separated the ground truth and input image more smoothly. The final stage of training the network was achieved by combining these decomposed and scaled images through Python code and scaling them to 512*256.After these steps, the dataset was included in the data augmentation process, which is the other process when the dataset needs to be multiplied in case of need. However, during this process, while creating the first dataset, all images were merged by crossing all images rather than "merged images" ground truth and input image similarity. In the second study's modified dataset, the background for the ground truth variable was deleted, and all images were merged with their paired (images with similar boundaries) input values (Table 4.8). It aimed to determine whether the output of the trained network differed with the matching of the changed ground truth variable and the input variable.

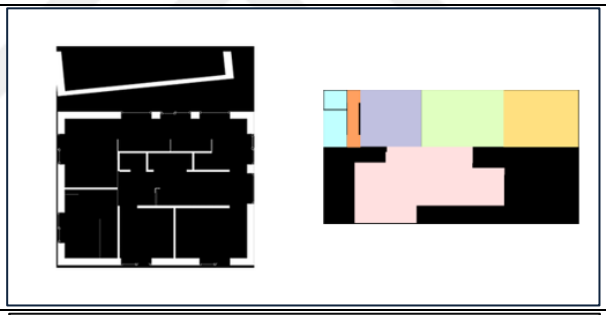
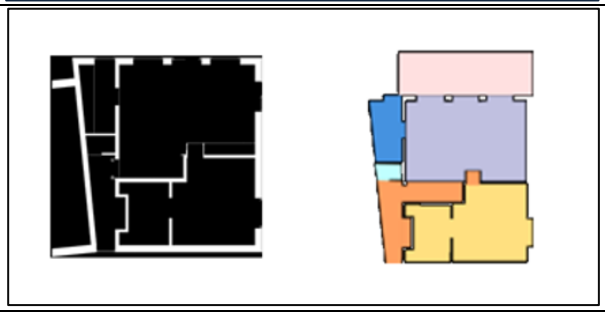
Table 4.8. Open plan dataset preparation (case 3) (prepared by author)



4.2.2. Datasets augmentation

The augmentation of the dataset differs based on the algorithms used. The first dataset used to train Open plan was resized to 256*256 (resized image), and all input values and ground truth images were combined with the crossing technique rather than the paired matching of ground truth for each input value. As a result of the experiment with this first dataset, different plan images were generated after training the network and applying data augmentation was tried with the other method, the augmentation process was carried out with the augmentor package of images rotated simultaneously at 90,270 angles with each input value and its identical ground truth (Table 4.9). After augmenting the datasets, case study 1 comprised 380 images. The ground truth of this dataset was prepared as black-backgrounded in the first process, while the second process was prepared with a white background. For the 3rd case study, 596 data were obtained.

Table 4.9. Data augmentation for open plan datasets

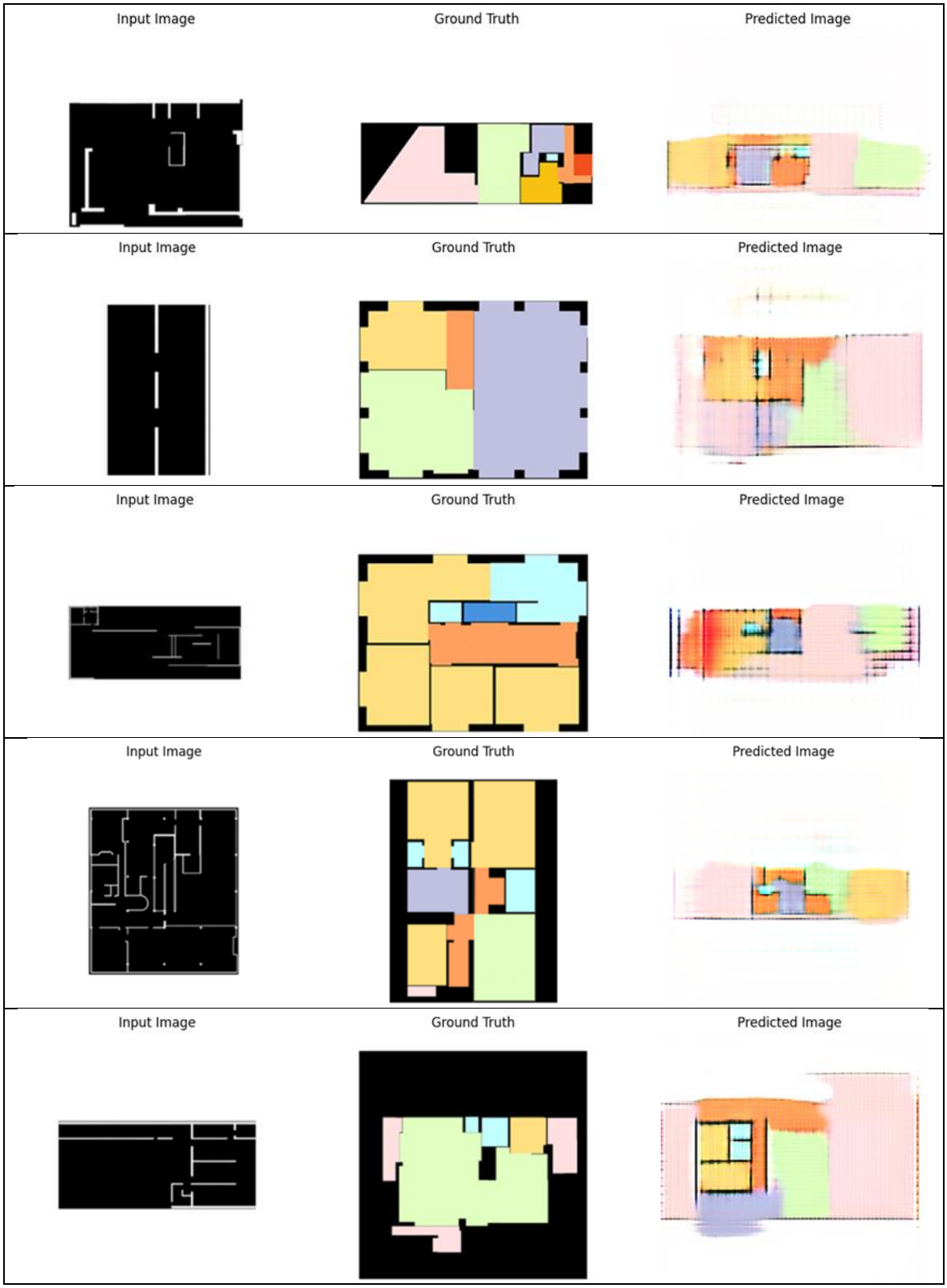
1. Data Augmentation(Creating all input images in the file by crossing all input images in the file with all ground truth images in another file).(case 1)	
2. Data Augmentation (Rotating 90° ve 270° all paired images in one file.(Case 2 and case 3)	

4.2.3. Training open plan datasets

The first trained dataset was trained at 100 epochs. The initial goal was to merge the input image regarding the ground truth image with the cross-reference technique instead of the paired approach. The ground truth image had a black background, and the walls or gaps between the spaces were also marked in black. The fact that the input images are different has also enabled different plan layouts to be generated.

Case 1

Table 4.10. Results of training datasets of open plan 01 datasets (100 epoch)



When interpreting the graphs for discrimination loss, generation total loss, generation GAN loss and generation L1 loss, each shows different aspects of the model and its performance. Below are general interpretations of these loss types:

Discrimination loss graph:

The discrimination loss shows how much the GAN's discriminator has been lost. The discriminator aims to distinguish between the model's outputs and authentic images. Low discrimination loss may indicate that the discriminator struggles to distinguish the generator's outputs effectively. If the discrimination loss is high, the discriminator can better tell the results of the generator apart.

Generation GAN loss (Gen-Gan loss) graph:

Generation GAN loss is a way to determine how much output the GAN's generator is losing. This sets how realistic the result of the generator is. The generator has learned to make realistic results if the GAN loss is low. However, a low GAN loss may also indicate that the timer does not work as well, and the generator can "cheat" more easily.

Generation L1 loss (L1-Gen loss) graph:

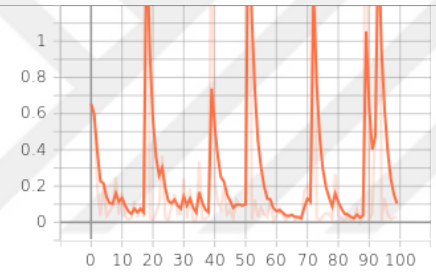
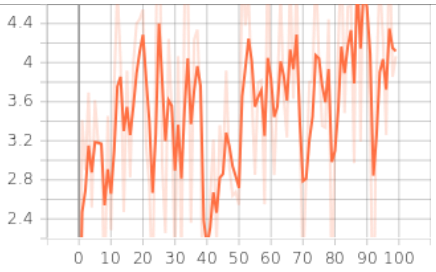
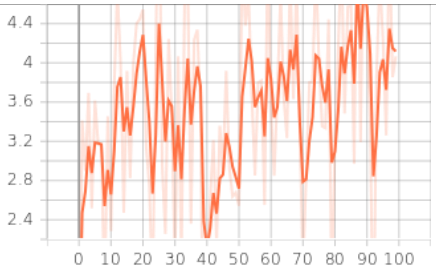
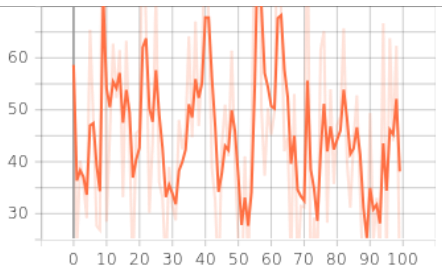
Generation L1 loss checks the closest proximity the generator's output is to the target picture, also known as "ground truth." If the L1 loss is low, it means that the generator is making output that is closer to the goal. A high L1 loss, on the other hand, could mean that the generator is having trouble making output closer to the goal.

Generation total loss(Gen-total loss) graph:

Total generation loss is the cumulative amount of output losses experienced by the generator. It often consists of discriminating loss, GAN loss, and L1 loss. Reduced overall loss often indicates improved generator performance. However, this could be different based on the choices and dataset. Often, when monitoring these loss graphs during training, it is essential to understand how each type of loss changes and in which areas the model shows improvement. Ideally, one should observe a decrease in overall losses over the training process and an improvement towards the desired performance. However, evaluating these

losses together and visually inspecting the model's performance on real-world data is essential. In the Discrimination Loss Function Graph, it is possible to see high values indicating that the discriminator perceives the images produced by the generator. In the Gen_Gan_Loss Function Graph, it can be observed that there are very few low values, which means that the generation of realistic outputs is low to medium. This evaluation assesses the authenticity of the ground truth image and measures the resemblance of the predicted to the ground truth (Table 4.11). When assessing the Gen_L1_Loss Function, the few low values indicate a low resemblance to both the predicted and the target image due to the low level captured in an epoch value in the function. The low values in Gen_Total_Loss values are low and average values, so the generator works well at the average value.

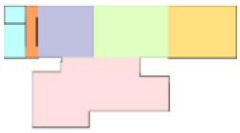
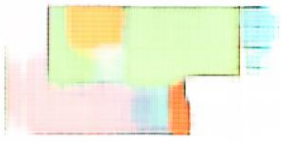
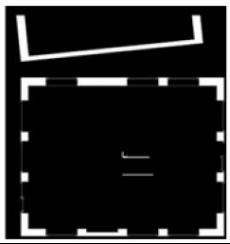
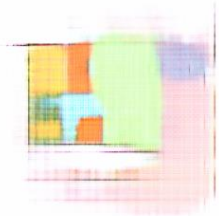
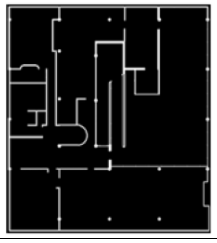
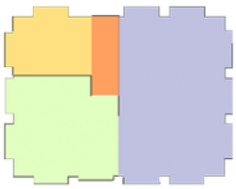
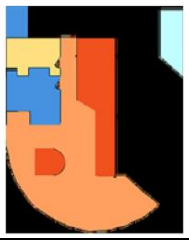
Table 4.11. Loss functions graphs of open plan datasets 01 training results (100 epoch)

Loss Functions	Graphs_Open_Plan_Datasets01 Training
Disc_loss	
Gen_Gan_Loss	
Gen_L1_Loss	
Gen_Total_Loss	

Case 2

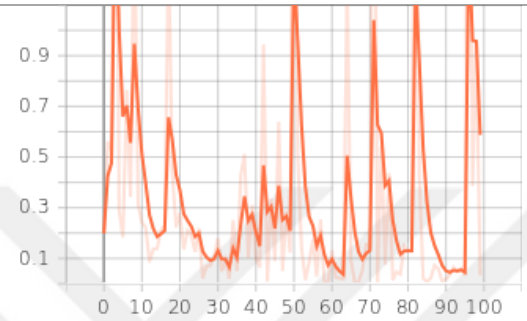
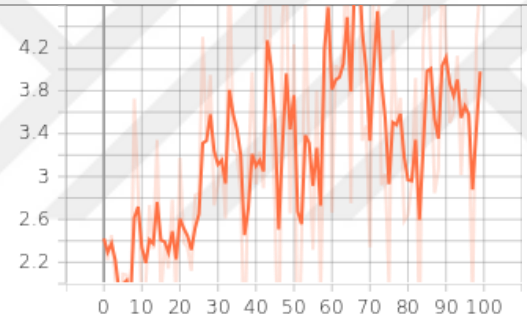
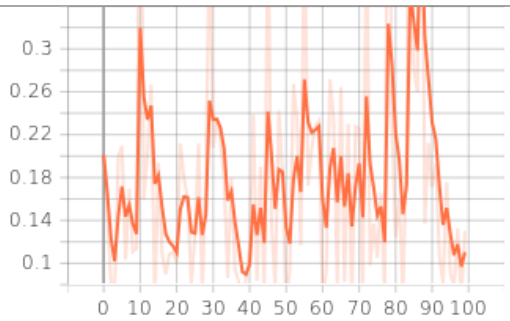
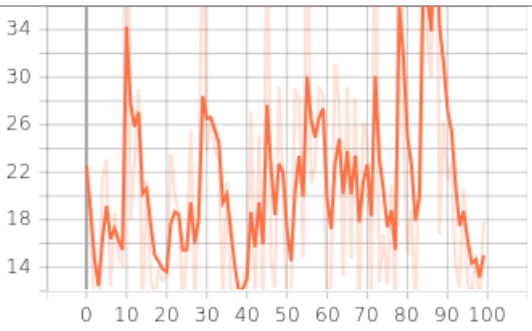
Table 4.12 shows the predicted image (generated image) from training the pix2pix network on Open_Plan_Datasets02 100 epochs.

Table 4.12. Results of training datasets of open plan 02 datasets

Input Image	Ground Truth	Predicted Image
		
Input Image	Ground Truth	Predicted Image
		
Input Image	Ground Truth	Predicted Image
		
Input Image	Ground Truth	Predicted Image
		
Input Image	Ground Truth	Predicted Image
		

In this dataset, images obtained by the crossing method are without changing the ground truth background (black wall and space definition). It is possible to say that the formation between the spaces in the obtained images can realize more flexible "not-compact" plan layout generation. Table 4.13 shows the values obtained after training the network.

Table 4.13. Loss functions graphs of open plan datasets 02 training results (100 epoch)

Loss Functions	Graphs_Open_Plan_Datasets02 Training
Disc_loss	
Gen_Gan_Loss	
Gen_L1_Loss	
Gen_Total_Loss	

From the high values, it is possible that the discriminator can more clearly identify the disparity between the ground truth data and the predicted images after 50 epochs in this network. For the Gen_Gan_Loss value, the higher-than-average value indicates that the network has not learned to generate realistic images. The Gen_L1_Loss value, on the other hand, shows that the network produced less accurate images than the ground truth. The Gen_Total_Loss number signifies that the generator's performance should be improved.

Case 3

Open plan 03 dataset was created by paired images of the input image and the ground truth with the background removed (black wall and space definition). While preparing this dataset, paired images were also prepared by undergoing the same degree of transformation. The network was then trained. All of the findings are displayed in Table 4.14.

Table 4.14. Results of training datasets of open plan 03 datasets

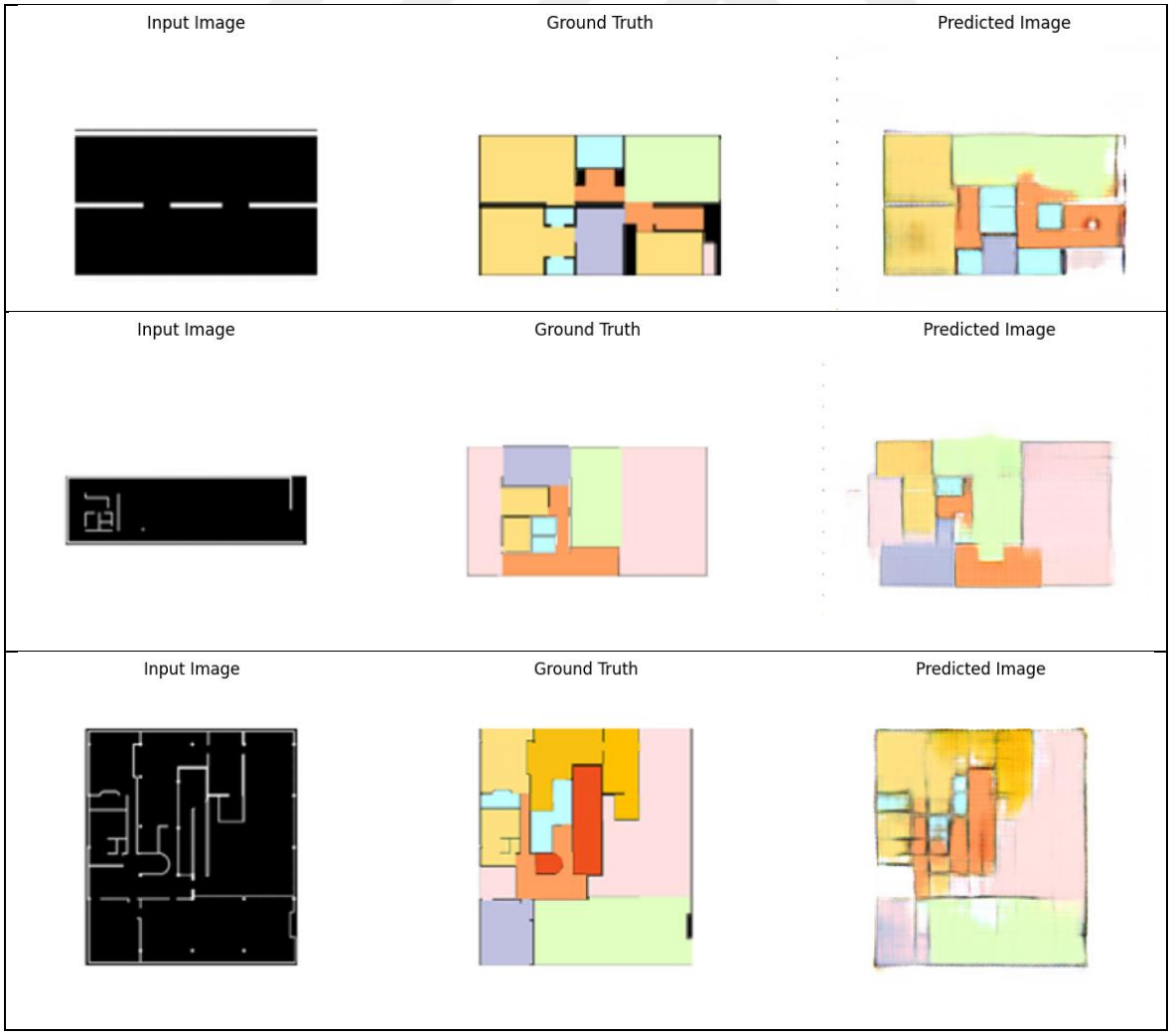
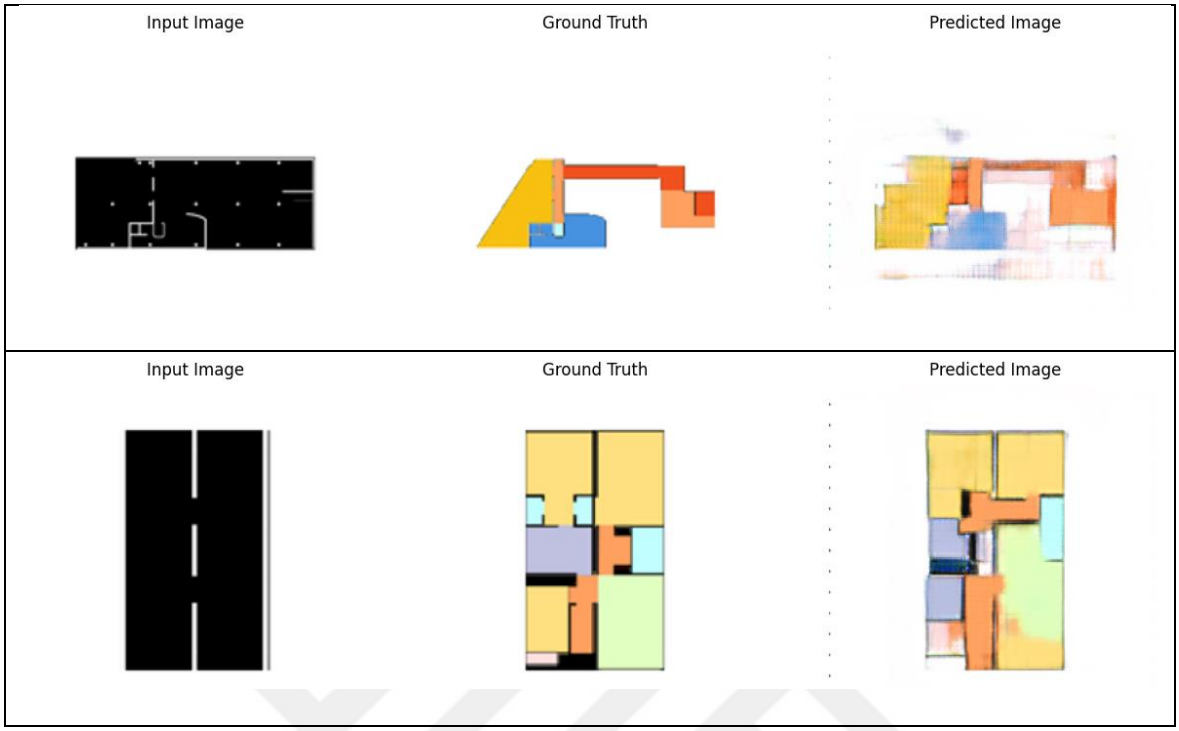
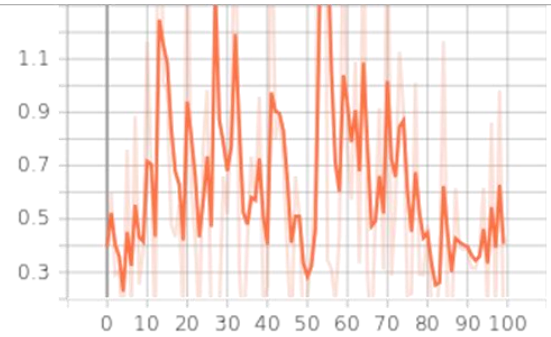
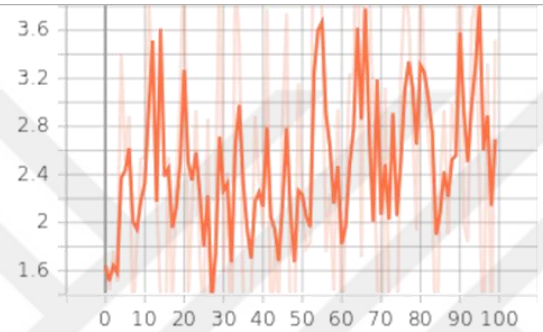
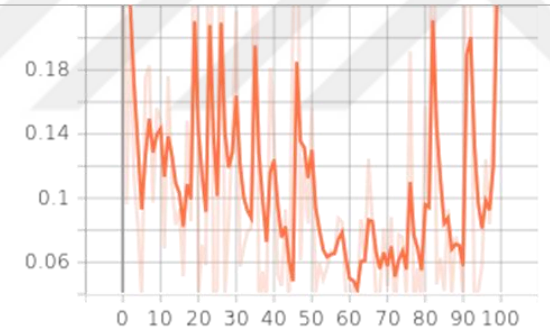
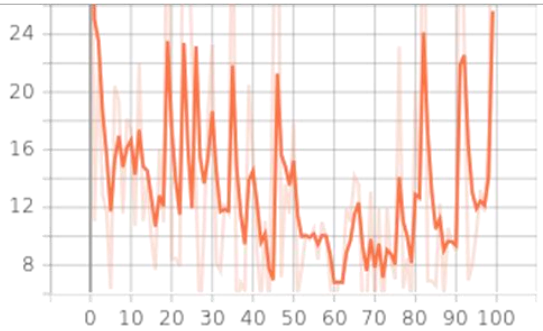


Table 4.14 (Continued). Results of training datasets of open plan 03 datasets



Gen_Gan_Loss is above average, indicating that the network can produce realistic images at an average level. The Gen_L1_Loss value shows that the network could generate images similar to the targeted image (ground truth) after 60 epochs, with low values indicating that the success rate was approached. The Gen_Total_Loss value shows that the generator's performance again shows that the correct generator works well between 60-80 epoch values. It is in parallel with the Gen_L1_Loss value. In Table 4.15 was showed the values obtained from training the network as described above. It is possible to understand from the low values that the discriminator cannot identify the variation between the images predicted by this network and the images of the ground truth. The most successful results in open plan layout generation were achieved in case 3. Looking at the loss values and the test images obtained as a result of the network training, it is seen that the predicted images/generated plan layouts are more perceivable and more feasible plan layouts. However, this success rate does not mean that the generated plan layouts generate more positive results in terms of spatial organization.

Table 4.15. Loss functions graphs of open plan datasets 01 training results (100 epoch)

Loss Functions	Graphs_Open_Plan_Datasets03 Training
Disc_loss	
Gen_Gan_Loss	
Gen_L1_Loss	
Gen_Total_Loss	

4.3. Pix2pix TOKI House Plan Layouts Generation

TOKI house plan layouts are typologically classified from mass housing plan layouts and prepared for the rapid construction process. Since these plan types are generated as tunnel formwork systems (load-bearing-wall), they are created based on closure rather than flexible

plan design. It has been observed that at least one module or more than one module in the designs created in all plan types can create a typology for mass production by symmetrically copying or mirroring on certain axes with transformation.

4.3.1. Datasets preparation

The labelization and semantic segmentation process of each plan layout and the typology of each different unit was completed (Figure 4.8). After this process, the dataset was prepared as ground truth, and the input image and the merging process were started with 256*256 resolution and dimensions to be suitable for the pix2pix algorithm.

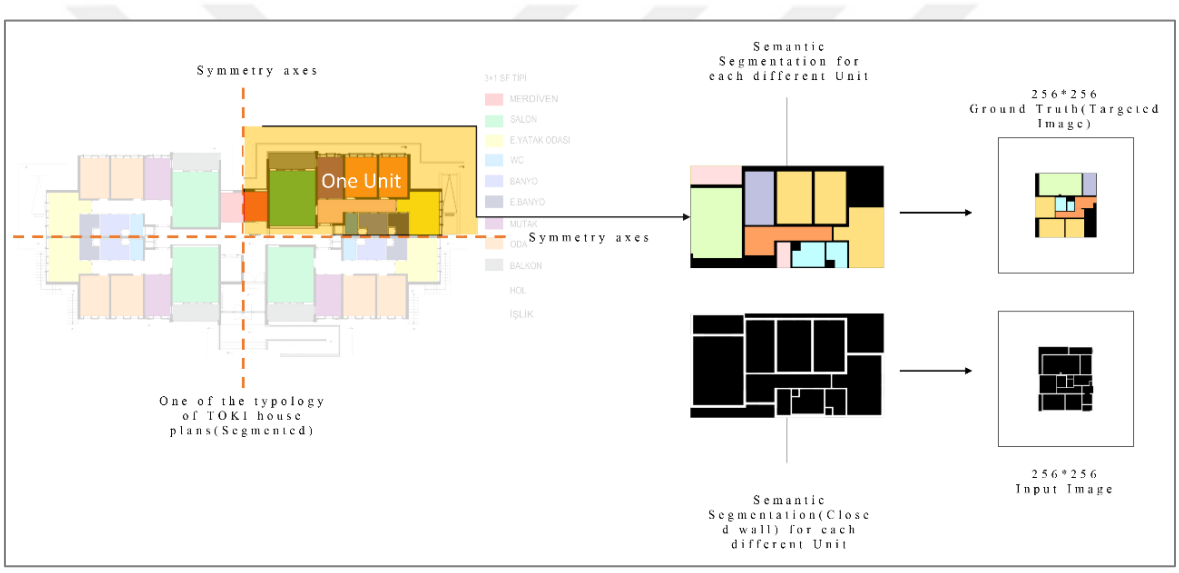


Figure 4.8. The preparation process of TOKI house plan layout(each unit) to pix2pix dataset (prepared by author)

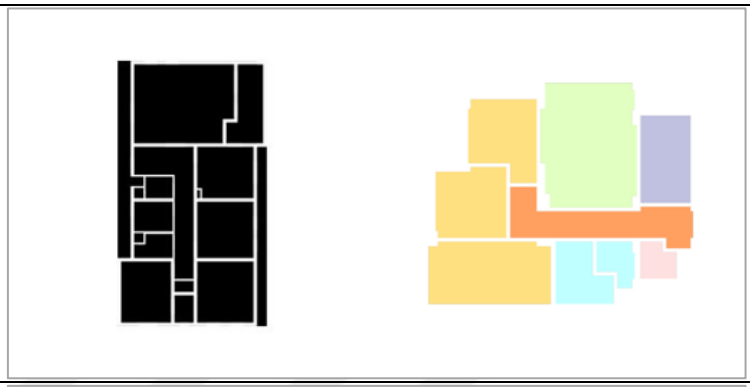
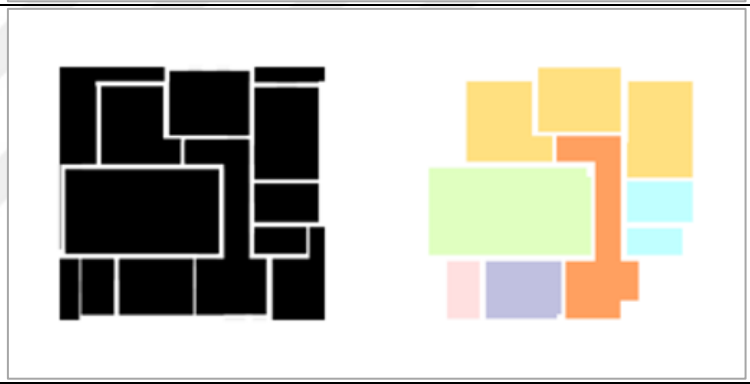
After each unit of the TOKI house plan layout goes through the semantic segmentation and labelization procedure according to Figure 4.7, the input visuals created with the walls as separate layers define the boundaries of the spaces.

4.3.2. Datasets augmentation

Similar processes to the open_plan_layout data augmentation process were used. As seen in Table 4.16, in the first process, the dataset created by differentiating the laws created by differentiating black backgrounded grounded truth images with the crossing technique and matching each ground truth with different input values was used in the first studies.

The second data duplication process was applied by rotating the paired images identically and simultaneously and then merging them. All the augmented datasets were combined into a 512*256 dataset to train the pix2pix algorithm.

Table 4.16. Data augmentation for TOKI house plan one unit

1. Data Augmentation(Creating all input images in the file by crossing all input images in the file with all ground truth images in another file).(1.case and 2. case)	
2. Data Augmentation (Rotating 90° ve 270° all paired images in one file.(3.case)	

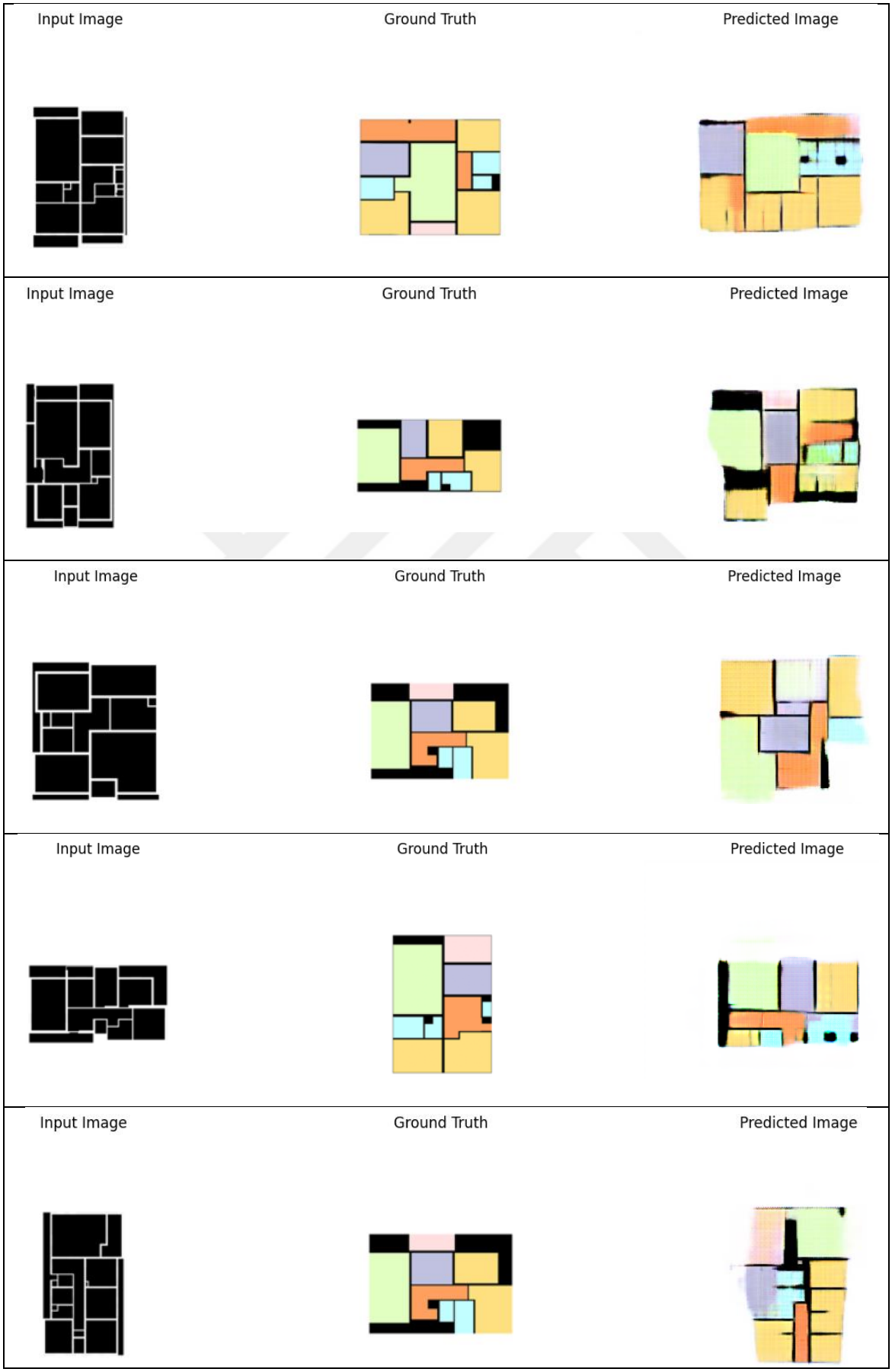
4.3.3. Training datasets

The training process of 3 different datasets is discussed. Case 1 presents the results of the dataset trained at 100 epochs and replicated with the black_backgrounded ground truth replicated with the crossing technique. In Case 2, the ground truth is changed to a white background. In Case 3, the network was trained with white background ground truth using paired images.

Case 1

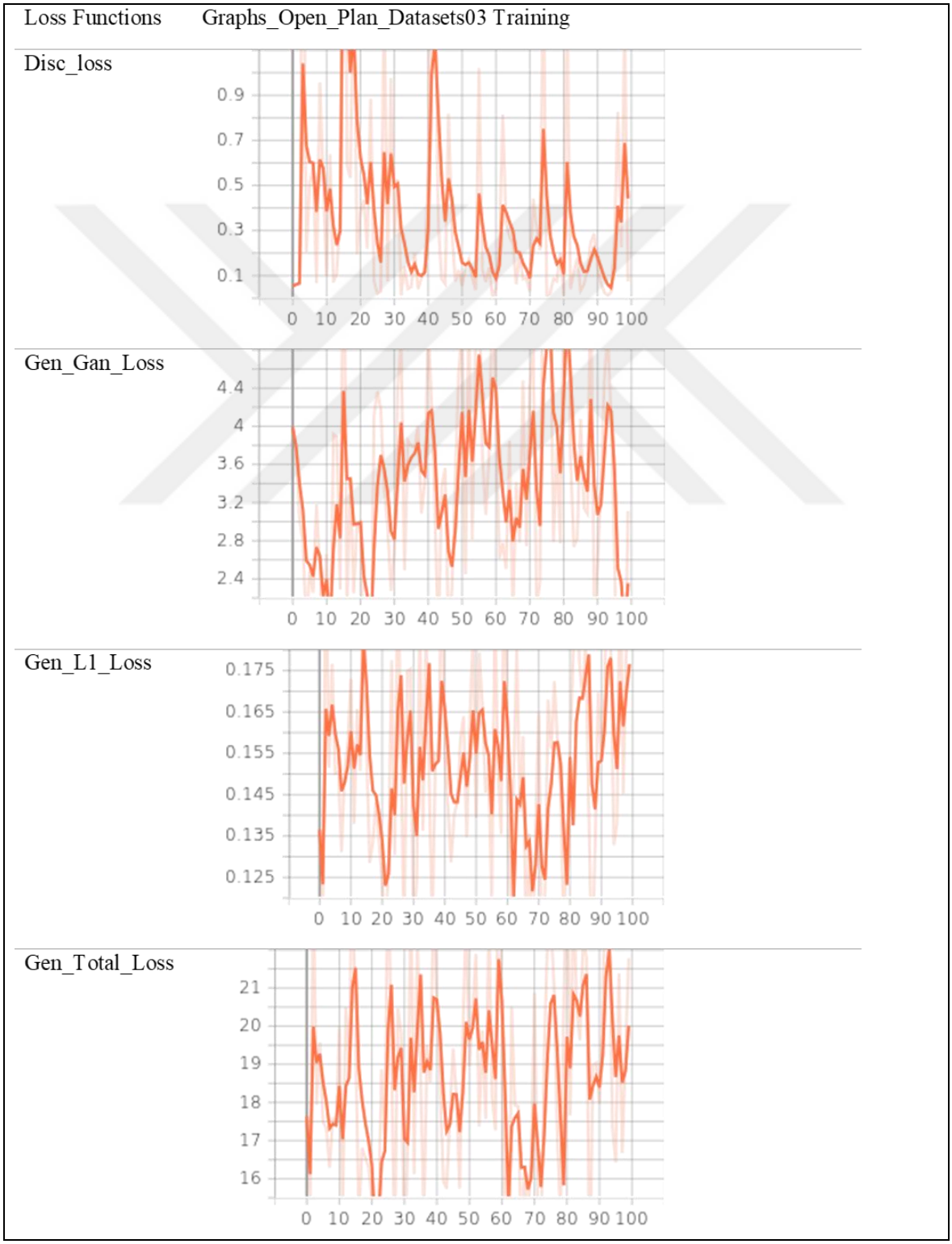
Due to the network's training, it is possible to state that different plan layouts are generated and predicted images that are different from ground truth and input images (Table 4.17).

Table 4.17. Results of training datasets of TOKI house plan 01 datasets (100 epoch)



It is observed that the discriminator value is high in the initial epoch values, indicating that the discriminator works well in the initial values, but decreases in the later values, indicating that the discriminator module cannot discriminate well (Table 4.18).

Table 4.18. Loss Functions Graphs of TOKI house plan dataset 01 Training Results(100 epoch)



Gen_Gan_Loss value is in high value ranges except for a few epoch values. It shows that the loss in the generated images, that is, “the similarity between the ground truth and the predicted image” is too high. As a matter of fact, the images obtained also express this. In the Gen_L1_Loss value graph, the fact that there are lower values in the lower range than the other training models but still high values in the total graph indicates that the generator (generator) has difficulty in producing images similar to the ground truth. The Gen_Total_Loss values show that the performance of the generator works well in the range of 60-80 epoch values, but the generator does not work well in other increasing value ranges.

Case 2

In this training set, the input image and the ground truth (white backgrounded) image were duplicated with the crossing method. In the graph where the Disc_loss value is observed, we can see that as the network is trained, it cannot recognize the generated images; that is, the function of the discriminator decreases functionally. It is observed that the predicted images are entirely distinct from the ground truth and input images (Table 4.19).

Table 4.19. Results of training datasets of TOKI house plan 02 datasets (100 epoch)

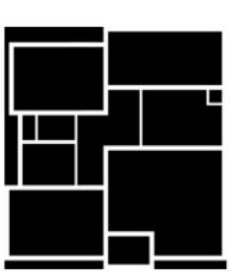
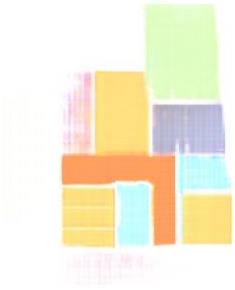
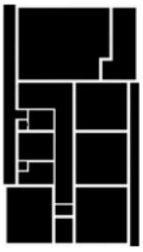
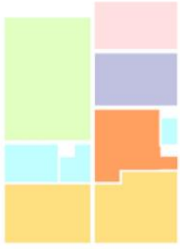
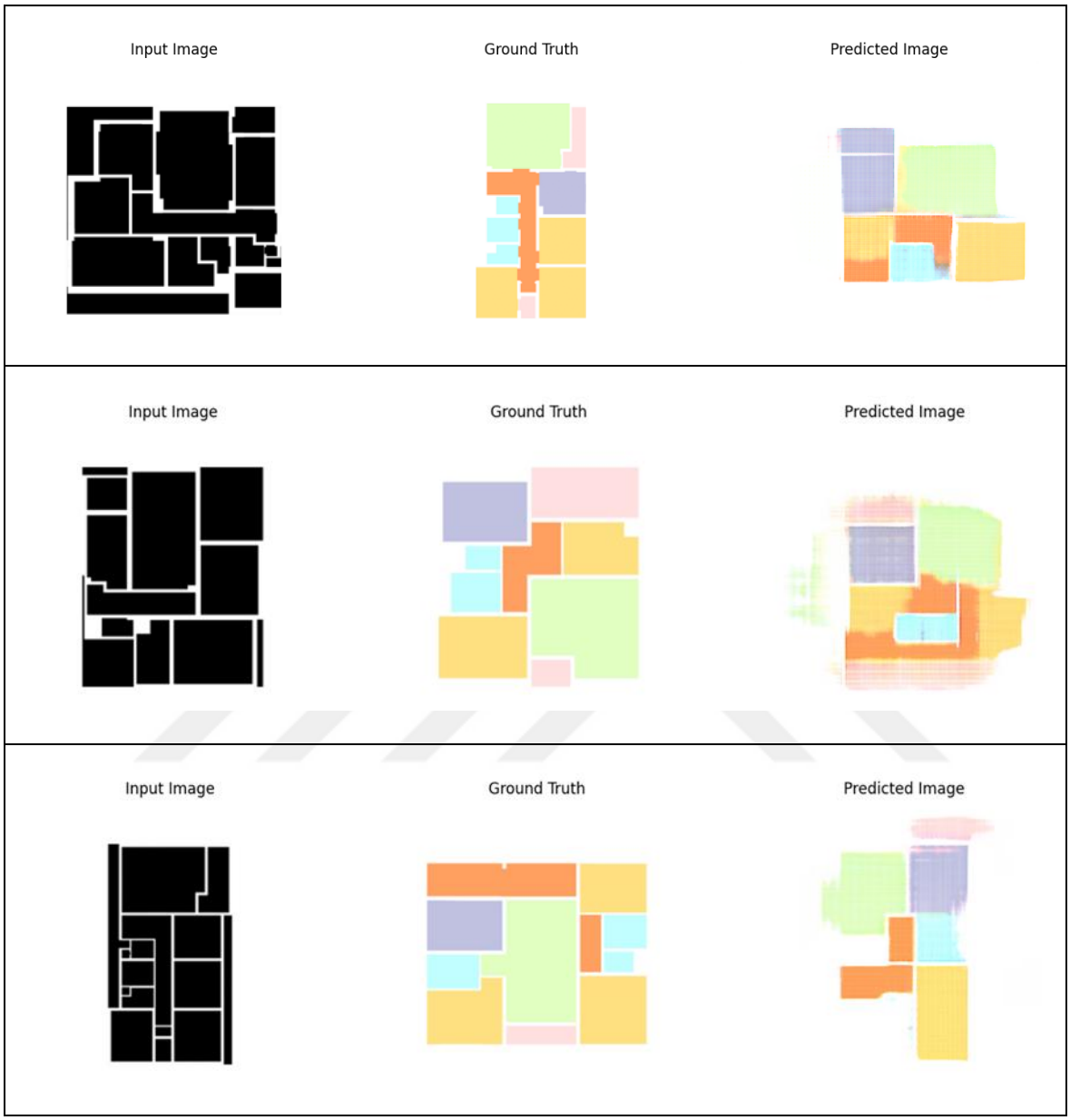
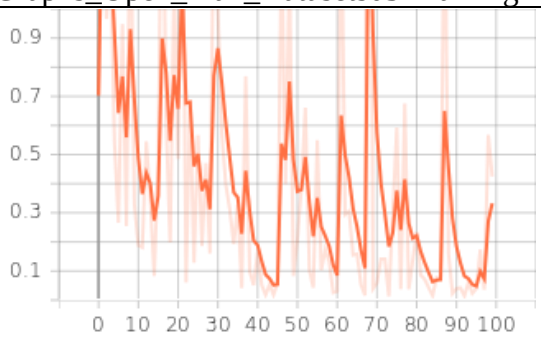
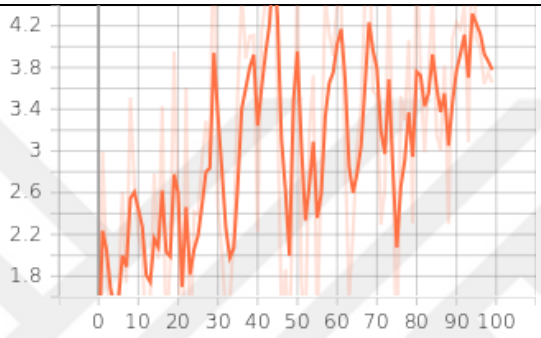
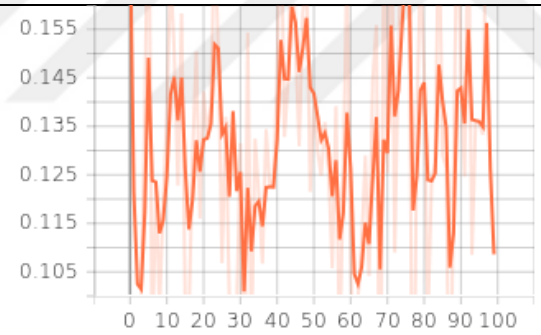
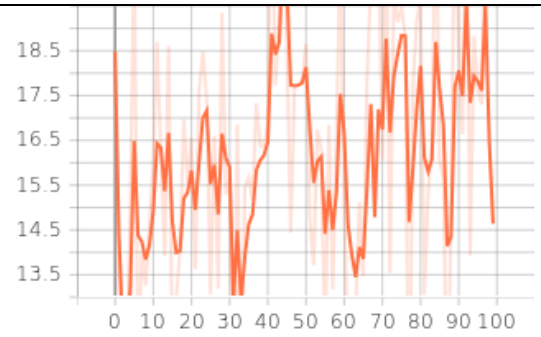
Input Image	Ground Truth	Predicted Image
		
Input Image	Ground Truth	Predicted Image
		

Table 4.19 (Continued). Results of training datasets of TOKI house plan 02 Datasets (100 epoch)



In the Gen_Gan_Loss value graph, while the visual loss is the difference of predicted images and ground truth starting from low, and it is challenging to decide the discrimination process, the gradually increasing value graphs after 30 epochs indicate that the loss increases. Gen_L1_Loss shows that the image generates realistic images at low values, but fake images are generated at high values compared to the ground truth. Gen_Total_Loss value graph shows that the total loss of the image generated through the generator is high. This value graph changes parallel to the L1 graph (Table 4.20).

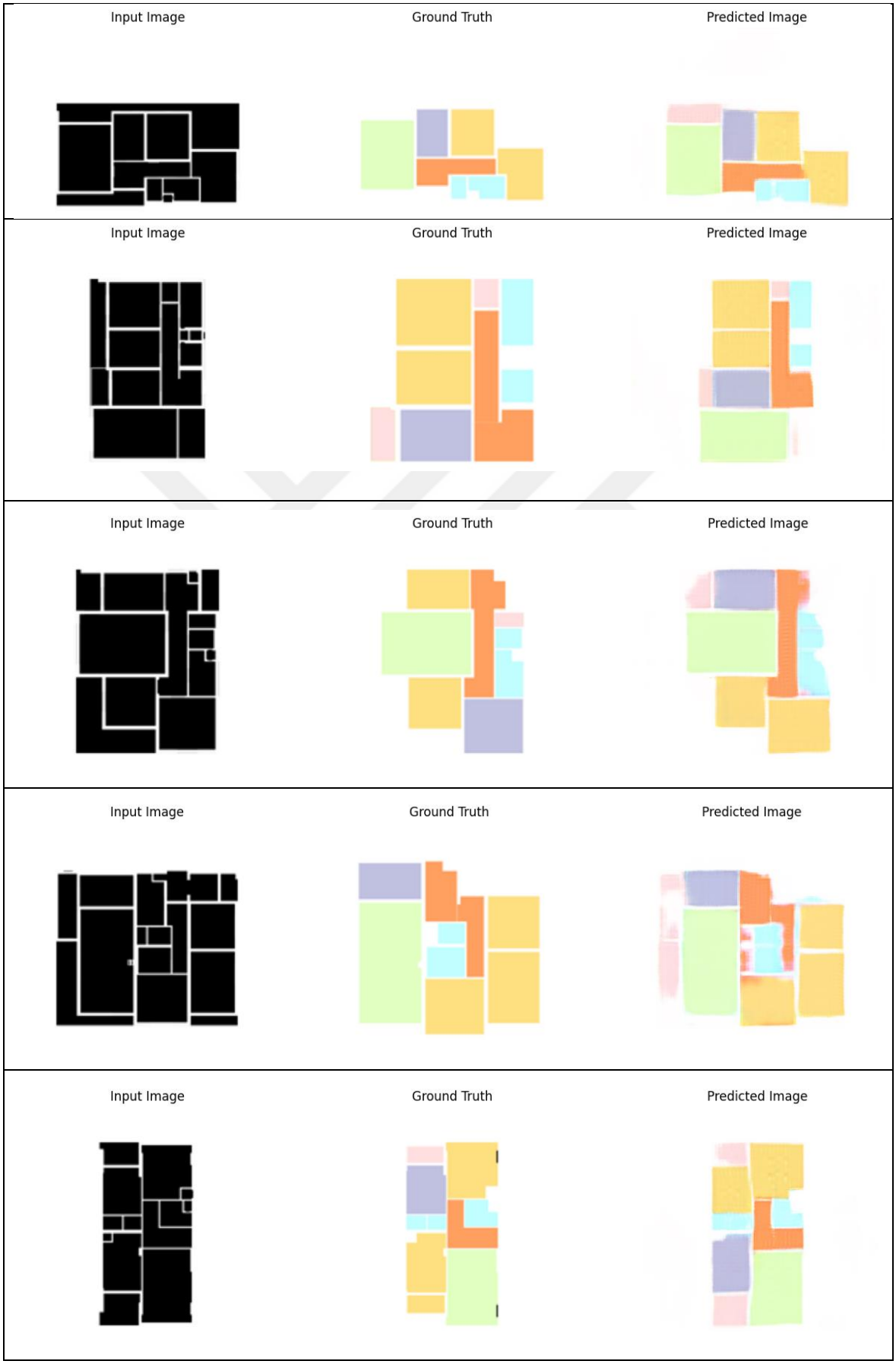
Table 4.20. Loss functions graphs of TOKI house plan 02 datasets (100 epoch)

Loss Functions	Graphs_Open_Plan_Datasets03 Training
Disc_loss	
Gen_Gan_Loss	
Gen_L1_Loss	
Gen_Total_Loss	

Case 3

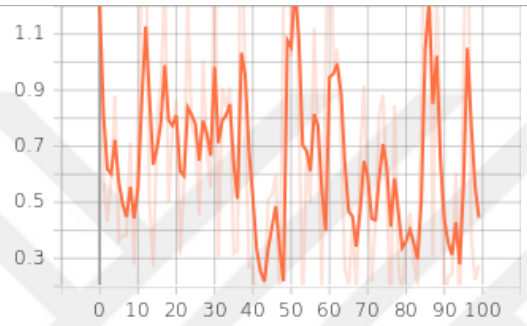
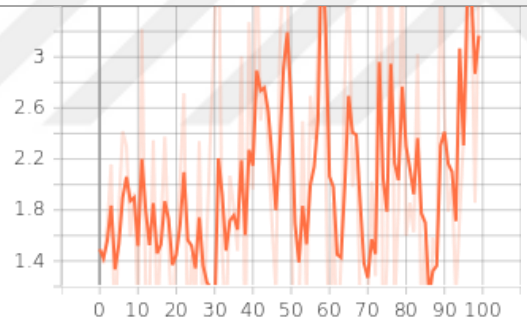
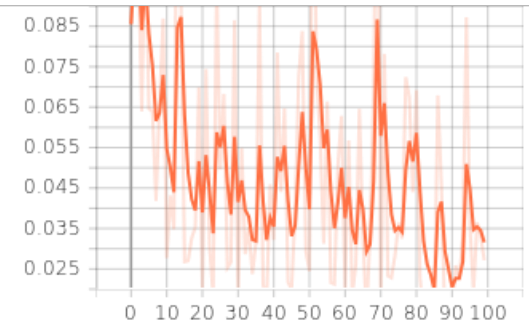
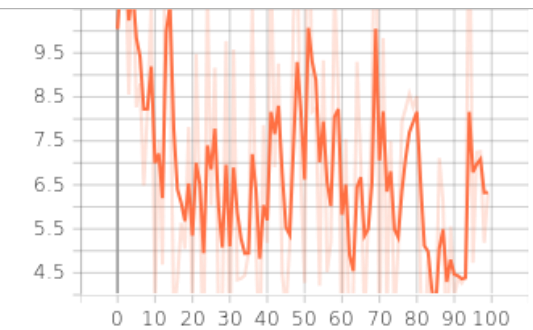
Table 4.21 displays the images divided into input, ground truth (target image), and predicted images.

Table 4.21. Results of training datasets of TOKI house plan 03 datasets (100 epoch)



Finally, the training process was tested on a dataset where the training set consisted of paired images, identical images of the input value, and the ground truth value. This study's ground truth consists of (white backgrounded) images. The predicted images are similar to the ground truths. When we look at the values in Table 4.22, we can see that the discriminator works well according to the high discrimination loss values and value ranges compared to the results of other trained networks.

Table 4.22. Loss functions graphs of open plan datasets 01 training results (100 epoch)

Loss Functions	Graphs_Open_Plan_Datasets03 Training
Disc_loss	
Gen_Gan_Loss	
Gen_L1_Loss	
Gen_Total_Loss	

When we look at the Gen_Gan_Loss values, low levels in the initial epoch values show that the image generates realistic images. The interpretation of the graph of Gen_L1_Loss value, which gradually reaches low values, shows that the generator generates images close to the target image (ground truth) as the epoch value increases.Gen_Total_Loss value shows that the generator works better towards the last epoch values of the graph, which is at low values in parallel with the L1 function graph.

5. TUNING OF THE MODEL

This section aims to evaluate the plan layouts generated by GAN networks. The evaluation of the networks based on their result metrics is not sufficient for the evaluation of 'spatial plan layout.' This is because the generated images are not just images; they define a space. For this reason, the generated/predicted images of the Pix2pix algorithm, which are the most identifiable of the generated plan layouts, were preferred. When these algorithms are considered as 'black box' models, there is no certainty and accuracy value of the results produced by the algorithm.

For this reason, the 'Object Detection' method was used, allowing both the algorithm results to be re-evaluated with another AI algorithm and thus improving the plan layouts' quantifiability. The process of 'classification/decomposition of the defined space function' of the spaces of the plan layouts was carried out. While the BBOX determinations obtained through this process determined the physical boundaries of the plan layout, it also enabled the transition to transforming it into the final stage, where the architect can make decisions. The plan layout estimation information is transferred to Rhino/Grasshopper, and class names (room, bathroom, hall, etc.), x,y, width, and height fields are embedded in .json data and transformed into a quantifiable state. For the architect to make decisions, basic parameters that will help us measure the quality of space in spatial transformations were created and analyzed in Grasshopper depending on these parameters. In the last process, the results of the analyses provided a decision support system to facilitate the architect's choice within the rules determined by the 'rule-based classification' method.

5.1. Object Detection

The object detection algorithm aims to determine whether the boundaries of the data we have obtained as a plan layout are correctly determined. This process is also essential regarding the rules for the decision-making phase. Figure 5.1 provides a visualization of how this process works. The Roboflow interface was used at this stage. The data processed here was also trained here. It was also connected through the Google Colab platform, allowing it to be trained and used again. The object detection method aims to determine whether the functions were correctly defined in the plan layouts resulting from the Pix2pix algorithm and to obtain BBOX data.

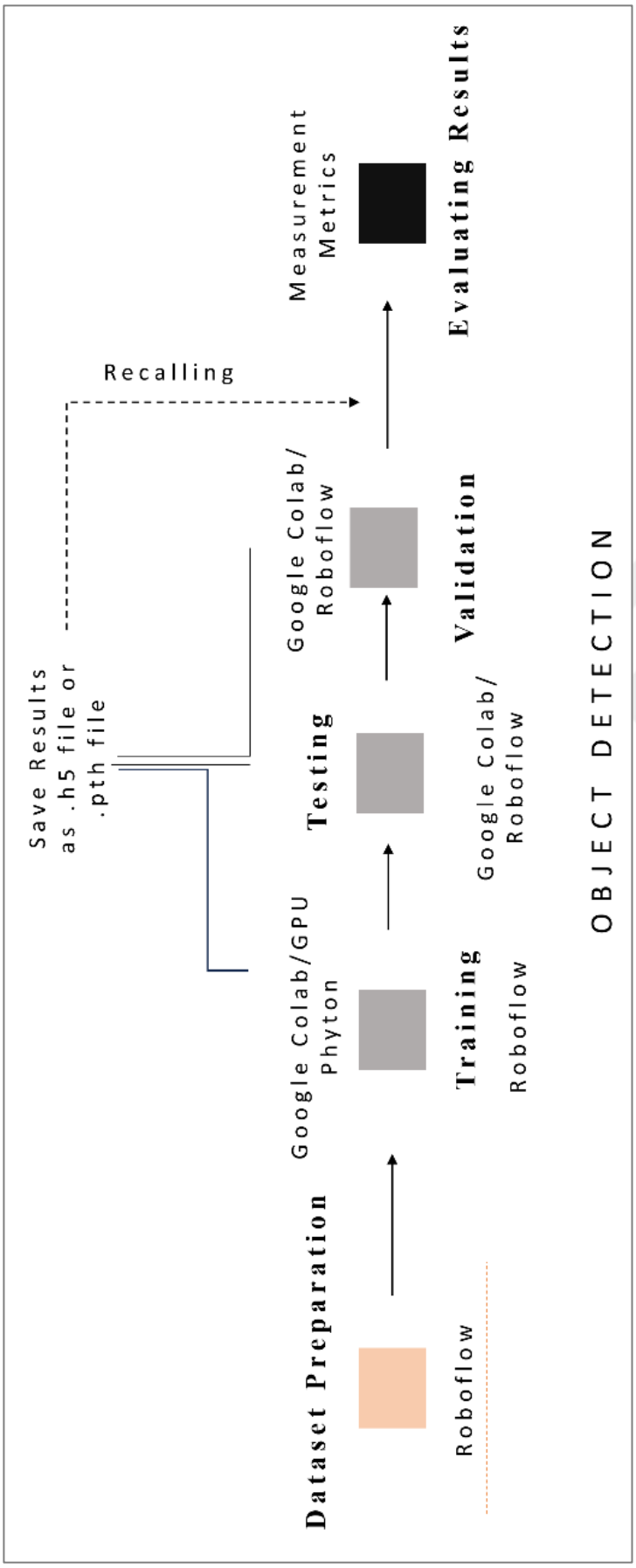


Figure 5.1. Process of object detection (prepared by author)

Initially, using the RGB segmented state of the dataset we prepared for the Pix2pix algorithm, all functions in all plans were processed in BBOX (bounding box) for object detection (Figure 5.2). The network was trained with Roboflow and Google Colab during this process in 100 epochs. In this algorithm, where the dataset can be separated directly into train, test, and validation, the network weights are saved for reuse.

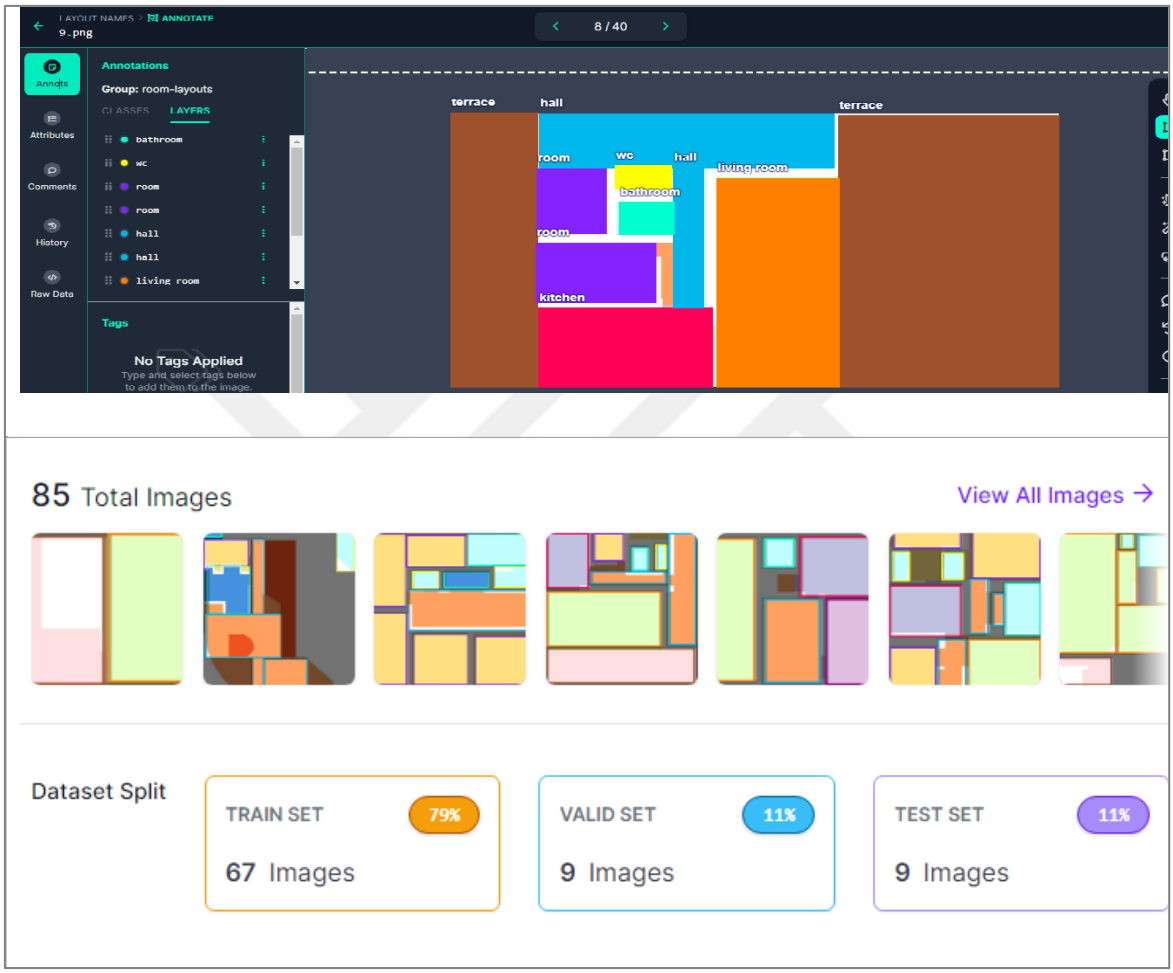


Figure 5.2. Object detection annotation process through roboflow

As a result of the training of the networks after labeling the plan layouts of the plans in all the datasets, we have the re-annotation process based on the variety of spaces from the plan layouts, the variety of the detected spaces in the plan dataset, the box sizes of the Anchor Boxes are given in the x,y coordinates and width and height graph (Figure 5.3). The Anchor Boxes detected here are given as the center point of the rectangle on this grid, as in the process of the yoloV5 algorithm, where the whole image is divided into grids. The coordinates of the BBOX detection formed on it are given as the center point of the rectangle on this grid. However, when the processed dataset is trained with different networks, it is

known that the corner coordinates of the BBOXs determined in case of conversion to other formats are taken as the basis. For this reason, different dataset transformations were created when testing the object detection implementation of the Roboflow-prepared dataset in the Google Colab environment.

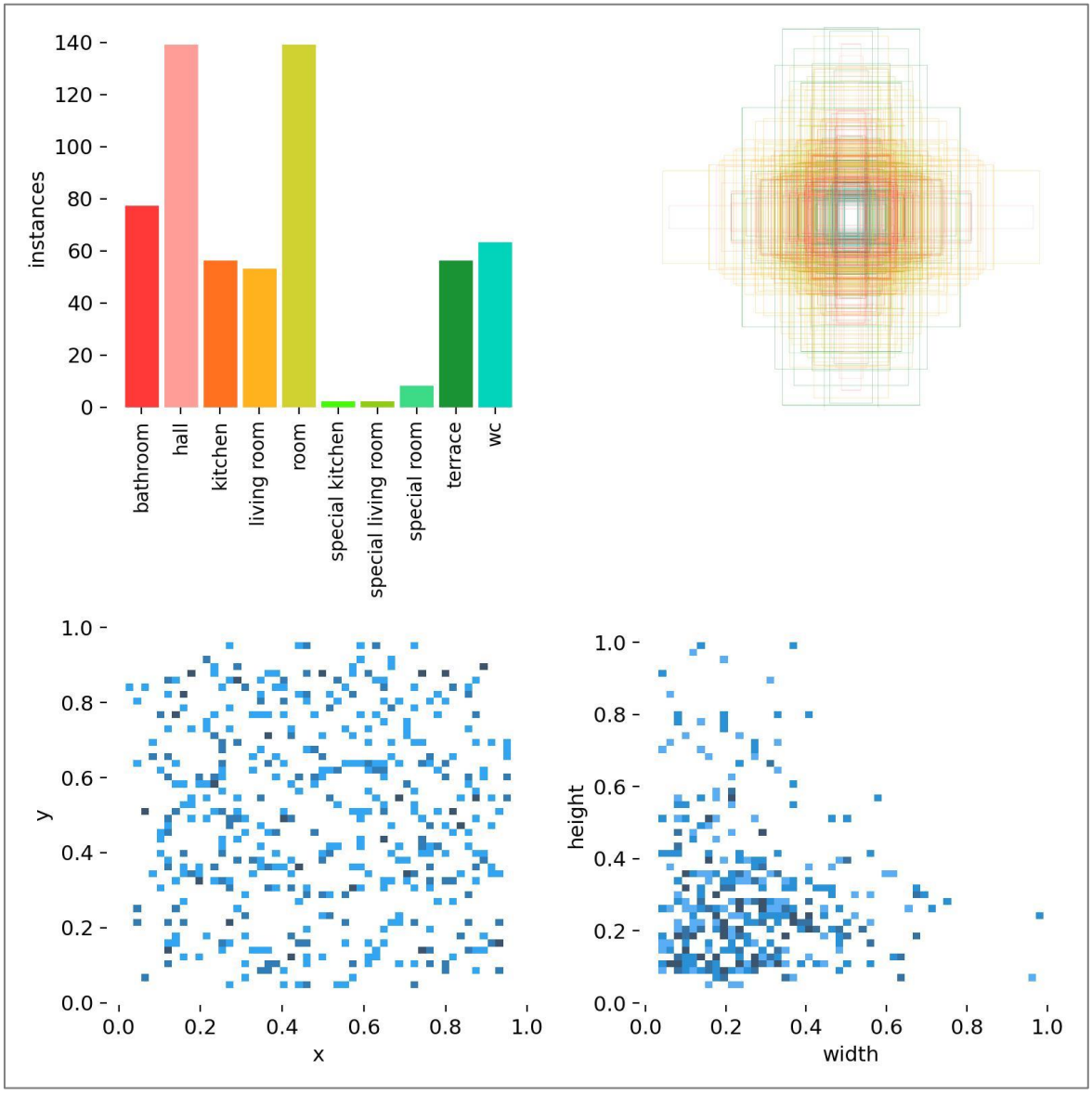
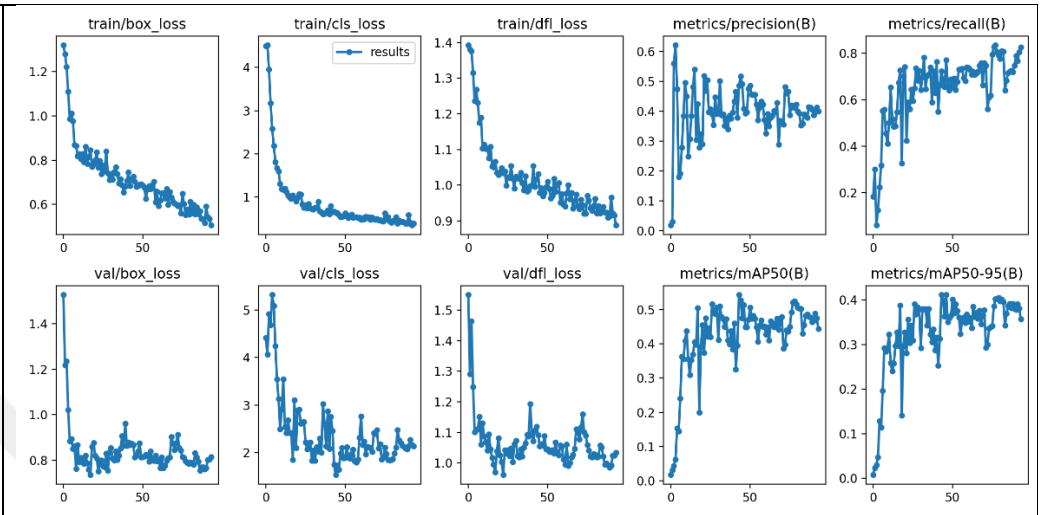
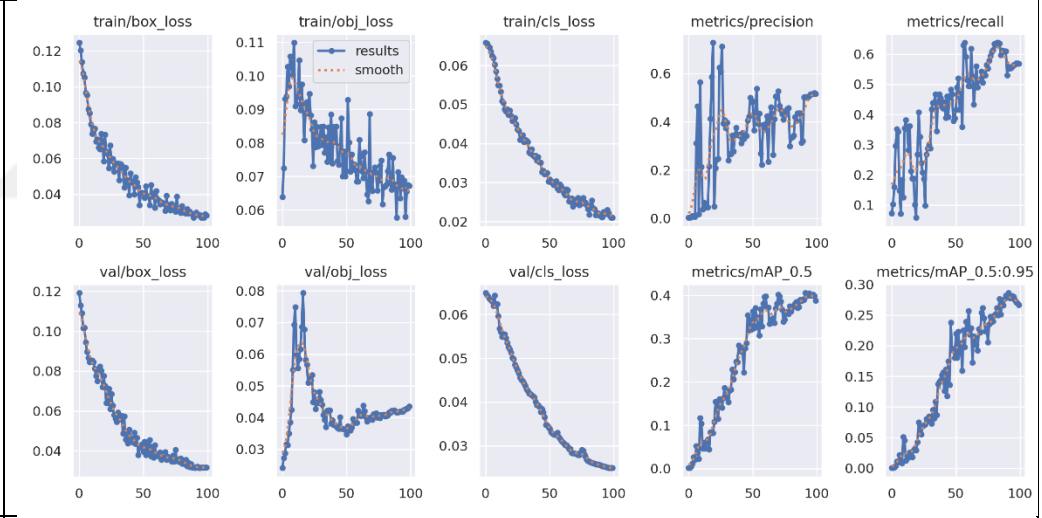
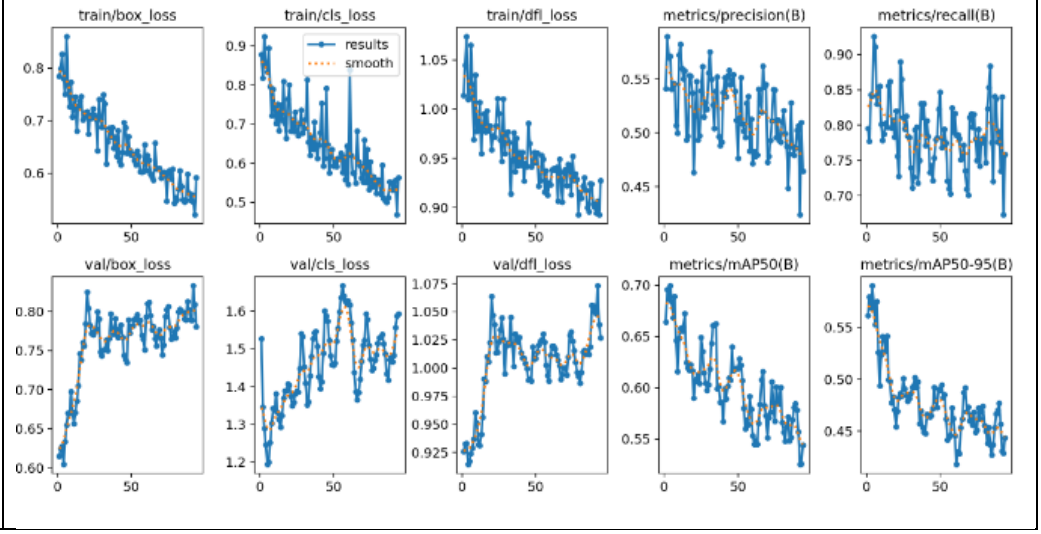


Figure 5.3. Predicted the dataset's spatial variations, coordinates, and BBOX length and width features after training in the Roboflow environment.

After the detection of the relevant objects, accuracy and precision values emerge. A more significant number in the precision/recall value graph indicates a greater accuracy rate in detecting objects. mAP values indicate whether the algorithm is generally successful. The graph of the obtained results is given in Table 5.1. At 100 epochs, the results of the data trained with different algorithms are presented comparatively in the table. Loss values reveal

the loss values of the locations, classes, and objects of the detected boxes. Low loss values indicate that the loss is low.

Table 5.1. Training 100 epoch for object detection

Google Colab-Yolov8	
Google Colab-2-Yolov5	
Roboflow	

It was observed that the precision of the trained network was higher with Yolov8. According to the comparison of the loss values, it is observed that the loss of the detected classes is generally low, but the loss values are the lowest in Yolov8 and the highest in Roboflow. As a result of the network training, it was observed that there was a partial loss of prediction in Roboflow. However, in Google Colab, BBOX room layout detection of all images was achieved in 100 epochs. Figure 5.3 shows that the predicted images are evaluated based on the dataset, and the predicted scores are reported as a Confusion Matrix. The measurement of the results of the classification algorithms varies according to the dataset. While the classification model of a dataset consisting of two classes can be processed with models such as binary, regression, etc., multiple classification models of multiple classes are measured with the confusion matrix model (Heydarian (2022). Thanks to this matrix, we can see how accurate and inaccurate the algorithms are when identifying the classes.

The calculated Accuracy, Precision, Recall, and F1 scores are expressed in these matrices. Accuracy values alone are not sufficient for the evaluation of the matrices. When val/box values are considered, the network with the highest loss value was trained in Roboflow. Yolov5 and Yolov8 trained the most accurate network with the least loss. In fact, Yolov8 gave the network training results with the least loss in detections.

However, when we evaluate all these values not separately, but in a way to get the F1 score, the results of the Roboflow algorithm are more favorable. For this reason, the correct operation of the algorithms can be determined more clearly with the F1 score. As a result of our calculations, F1 values were calculated as 50% for Yolov5, 60% for Yolov8, and 80% for Roboflow.As a result of these values, the best working algorithm is the classification value in Roboflow. At the same time, it was preferred for the future of the study due to its connection to the real time process of obtaining .json data of the predictions/BBOX detections of the tested results.

TP, TN, EP, and FN values are used to evaluate the confusion matrix. These values are used to calculate the formulas mentioned in the Object detection section. The confusion matrices created to clearly see all these values are given in the Table 5.2.

Table 5.2. Confusion matrixes

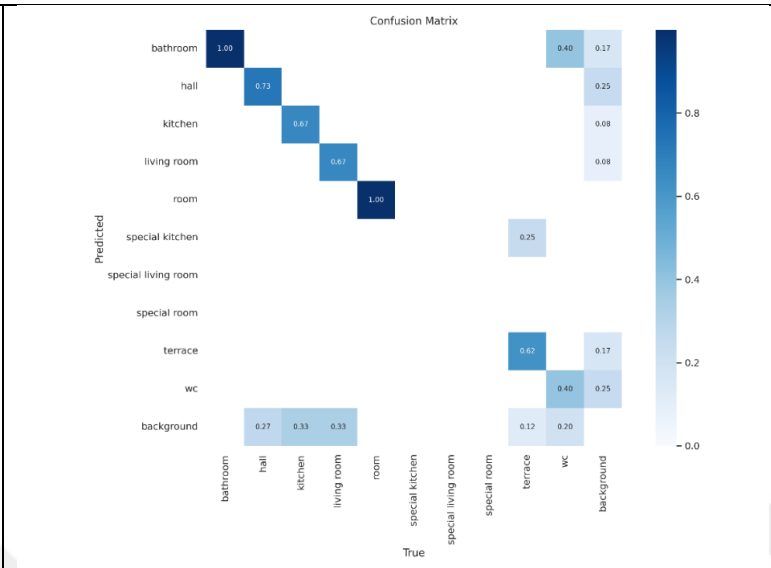
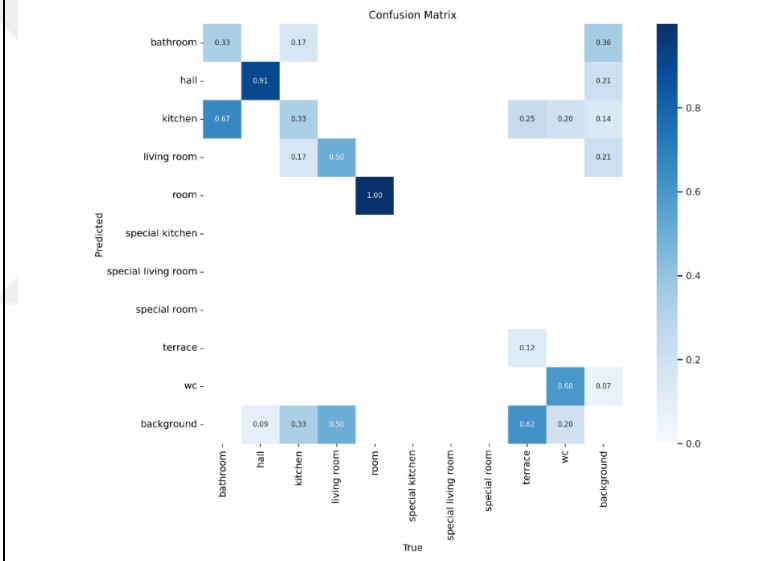
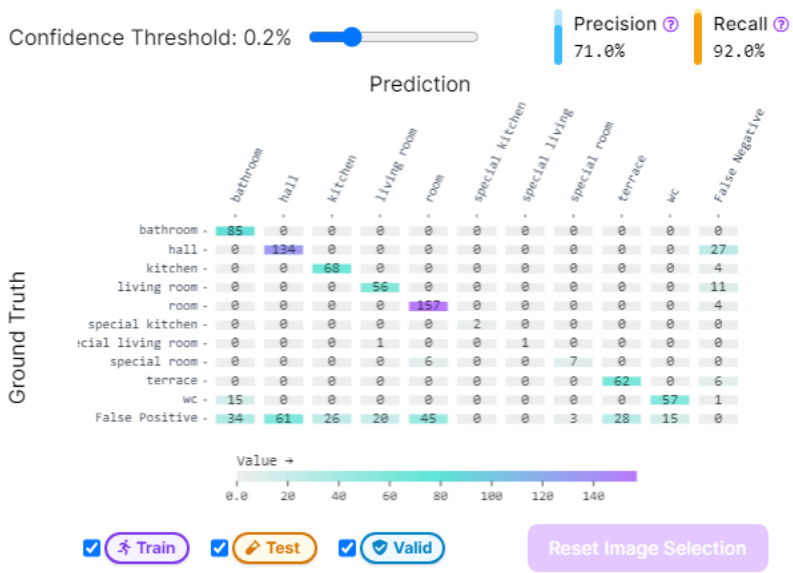
Google Colab-Yolov8	
Google Colab-yolov5	
Roboflow	

Figure 5.4 shows the image detections and scores resulting from the training of the network in Yolov5 in the Google Colab environment on the tested images. Here, it is observed that objects are mostly detected correctly, even if the values obtained from the detected images are low.

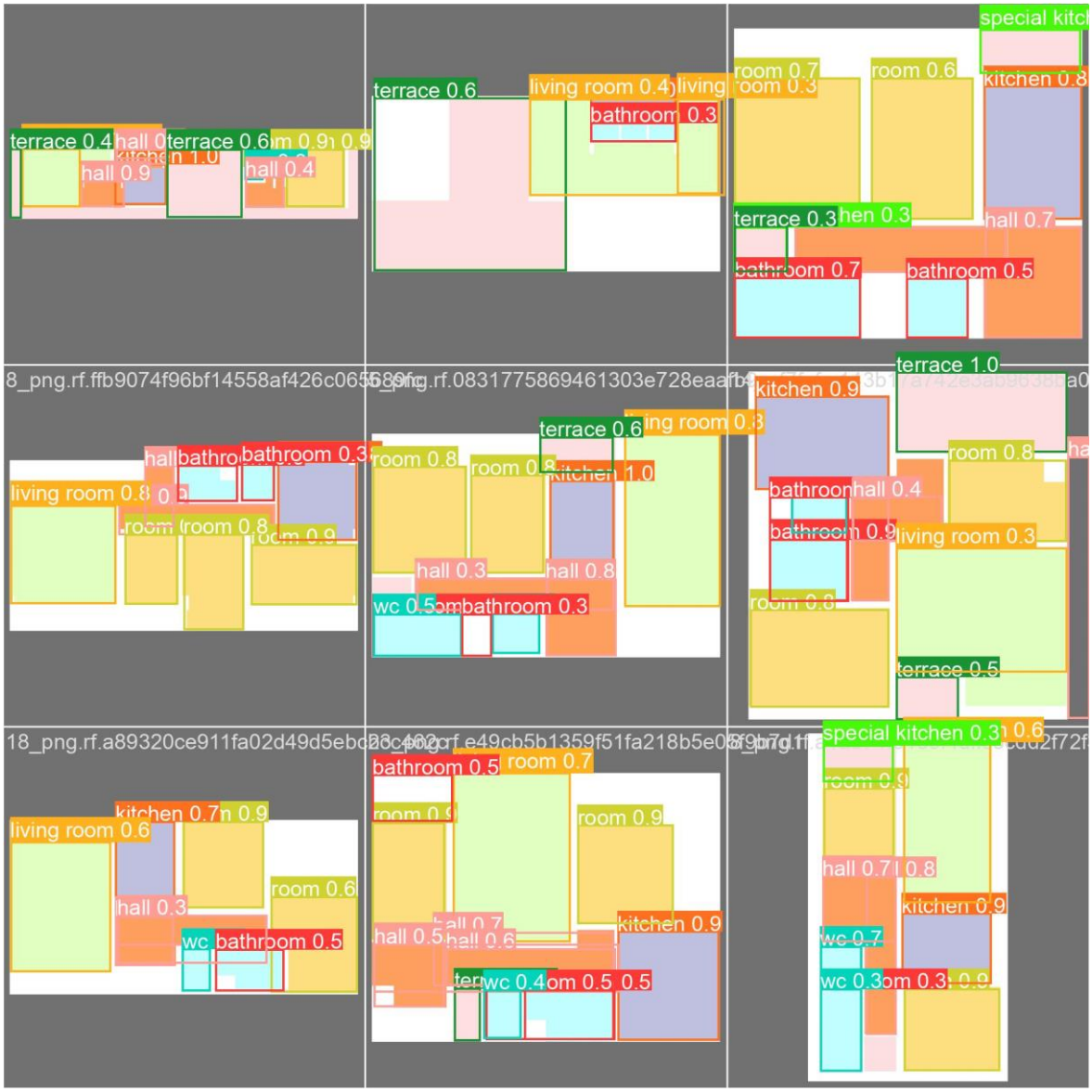
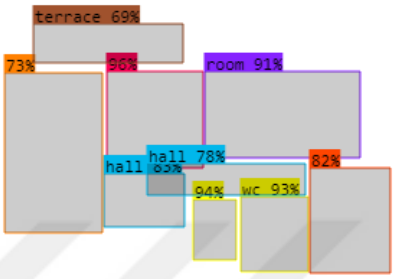
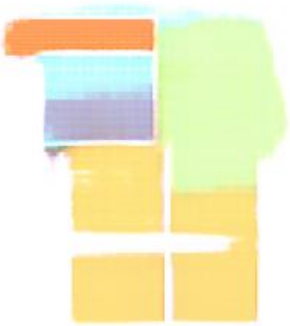
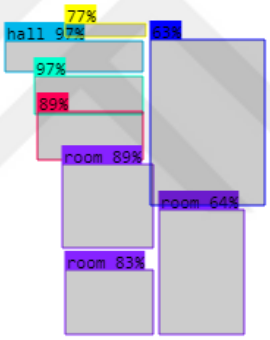
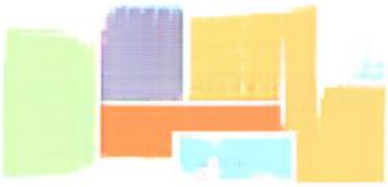
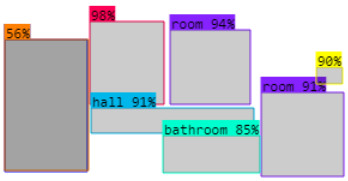


Figure 5.4. Google Colab-Yolov5 training results of object detection results tested with class data

Afterward, an interface was created to determine the room layout of the generated images obtained with the Pix2pix algorithm. In this interface, the predictions of the added images are provided (Figure 5.5). The images detected by the network trained in Roboflow as a result of the test and their scores are given in Table 5.3. It was observed that the detected BBOX object detections correctly identified the locations of the objects along with their

coordinates. In particular, the real-time object detections performed in Roboflow gave fast results, and the .json data of the tested images were correctly given, along with the values and locations of all classes.

Table 5.3. Roboflow bbox predictions of plan layouts generated with pix2pix algorithm

Positional formats for frameworks and packages that display bounding boxes can be different. Based on the properties of the response JSON object, the following rules can always be used together to make a bounding box.

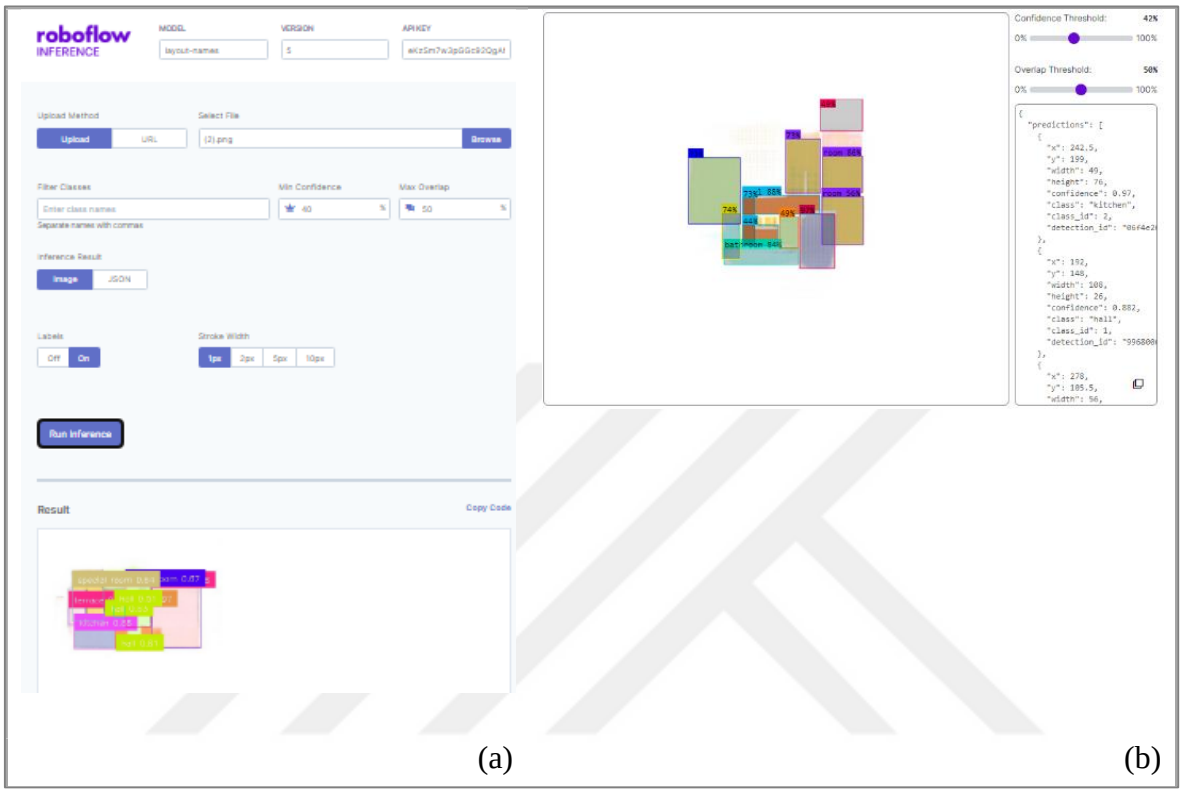


Figure 5.5 (a) Roboflow web application interface example (BBOX and predictions of detected visual images), (b) prediction result obtained with Json data

The obtained BBOX Json data was asked to predict the objects by applying a min.40% Confidence Treshold. Plan layouts that fall below this prediction and cannot be completed will be eliminated in the next stage.

5.2. Analysing Plan Layouts

The predicted visuals from Roboflow extracted the predicted functions with the .json data and BBOX locations. This .json data was transferred to the Rhino/Grasshopper program. The process of perceiving and transforming this data as an object was provided with the Jswan plug-in. Afterward, the BBOXes defined on top of each other were eliminated by reducing them with C#. In cases where there were problems with the (x,y) coordinates of all the BBOXs obtained, convergence was performed with the Kangaroo plug-in. After this process, the drawings obtained were transferred to the Rhino environment with the Elefront

plug-in. The Planbee plug-in was used to determine the possible openings of the plan layouts created here and to evaluate them in line with the analysis (Figure 5.6).

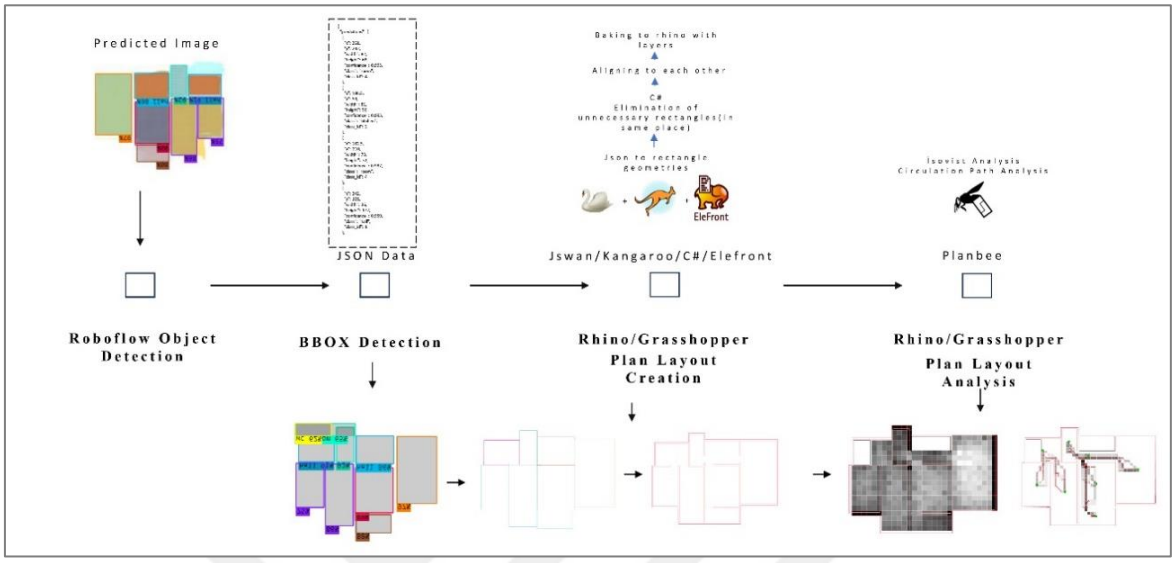


Figure 5.6. Workflow for plan layout analysis (Prepared by author)

Thanks to this plug-in, the continuity of the circulation of the plans, their fragmentation, and the openings that can measure the quality of the spaces were determined by isovity analysis. In addition to the analysis process, graph-based layouts of the spaces were also obtained to detect the connectivity of the connections between the functions with the space syntax analysis method. The json data obtained from Roboflow was transformed geometrically for rectangle detection with the Jswan plug-in. Figure 5.7 shows the Rhino/Grasshopper definition as defined.

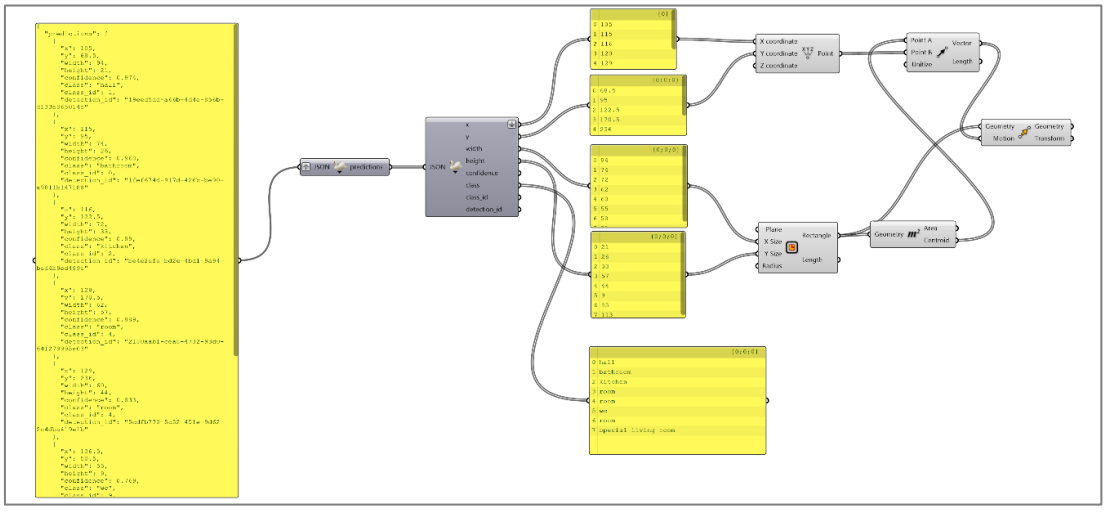


Figure 5.7. Json to Jswan plug-in process

The parameters and classids of all rectangles (x,y, width, height) were parsed and moved to the corner coordinate center. In the case of nested or distant rectangle situations in the plan, layouts moved to this corner coordinate, and additional definitions were added to the definition. With C#, the extra rectangle in the same position (valid for rectangles detected twice at the same point in the prediction process during object detection) was eliminated. Then, with the Kangaroo plug-in, the plan layouts that were too far apart were brought together with the predicted image and aligned with the convergence process. See Figure 5.8 to understand how the definition works.

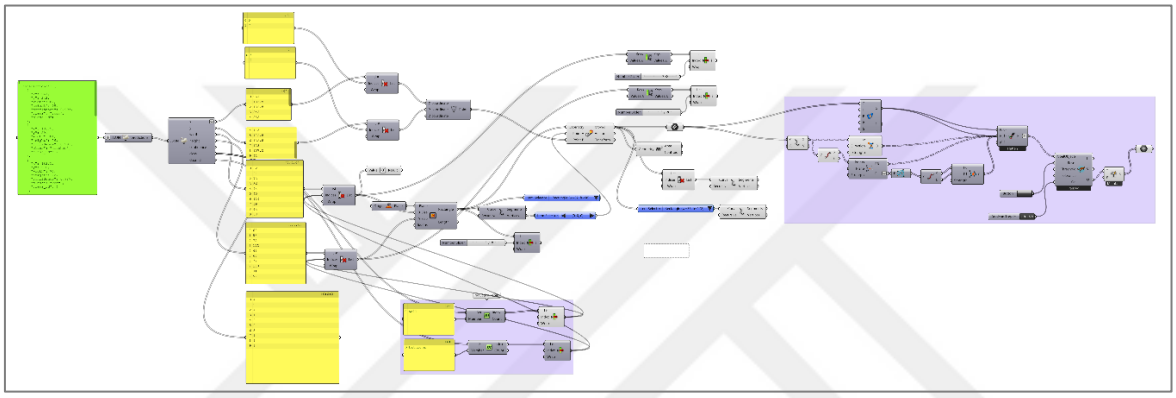


Figure 5.8. Definition from jswan to kangaroo plug-ins

After this process, the plan layout creation process of the room surfaces created in Grasshopper was realized. These plan layouts' interior and exterior walls were created with reference to the visuals produced by GAN networks. Afterward, simulations of isovist values and path analysis values were obtained with the PlanBee plug-in of these plan layouts transferred to the Rhino environment. At the same time, daylighting analyses were also determined according to the openness and closure levels of the spaces. The (x,y) value coordinates defined in the BBOX determinations by converting them to Json version have changed to define the corner points. These defined coordinate data created plan layouts with the reconstructed BBOX definitions.

PlanBee plug-in was used for plan analysis in the Rhino/Grasshopper environment. It aims to measure the space's quality with the analyses obtained with this plug-in. Visibility analysis shows both the openness of the space in the interior space and determines whether the viewing angle between the spaces is wide in spatial perception. This measure is significant for understanding the quality of spatial organization. As a result of the analyses, the general visibility value of the plan was determined by evaluating the remapped values in the range

of 0 and 1 values of the plans that gave more positive results in visibility by taking the average (average) value range (Figure 5.9).

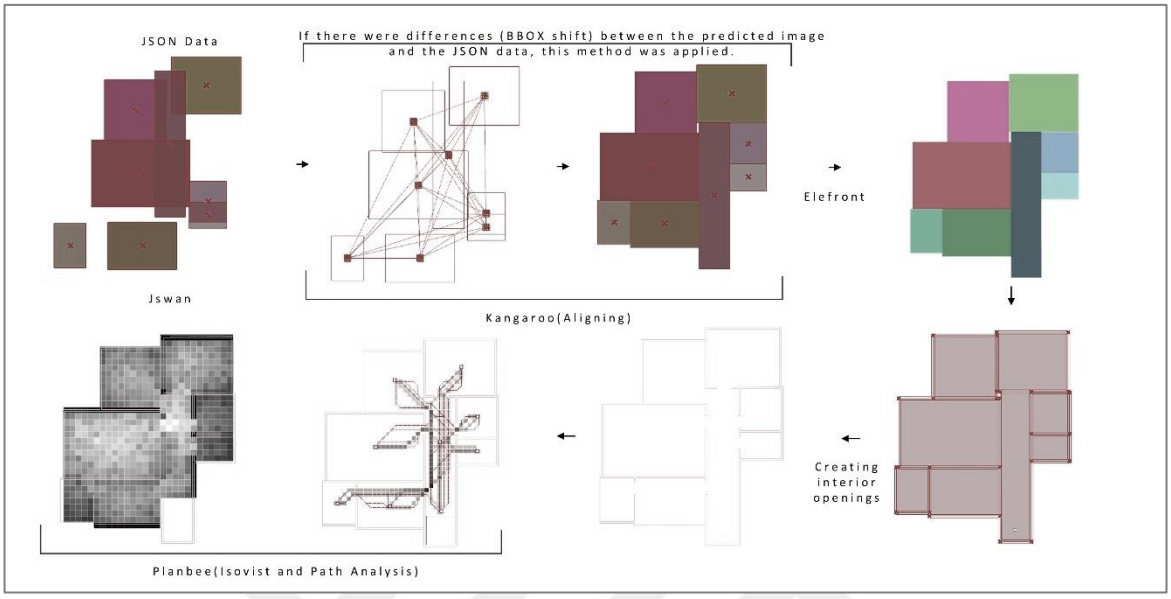


Figure 5.9. Workflow for tangible plan layouts (prepared by author)

It is crucial to consolidate these studies under the categories of spatial connection and spatial relationships to enhance comprehension of spatial organization and the quantity and variety of places. In order to understand space connectivity and space relations, the IDs, names, locations and number of spaces we obtained from Json data are given. In line with this information, the Space Syntax method was used so that designers could more clearly understand and make sense of the difference between the images produced. In this process realized in Rhino/Grasshopper, more than one method was tried. Space Syntax plug-in and SpaceChase plug-ins were tried. However, more accurate representations were obtained with the plug-in-free method. Figure 5.10 shows the analysis and Rhino/Grasshopper definitions.

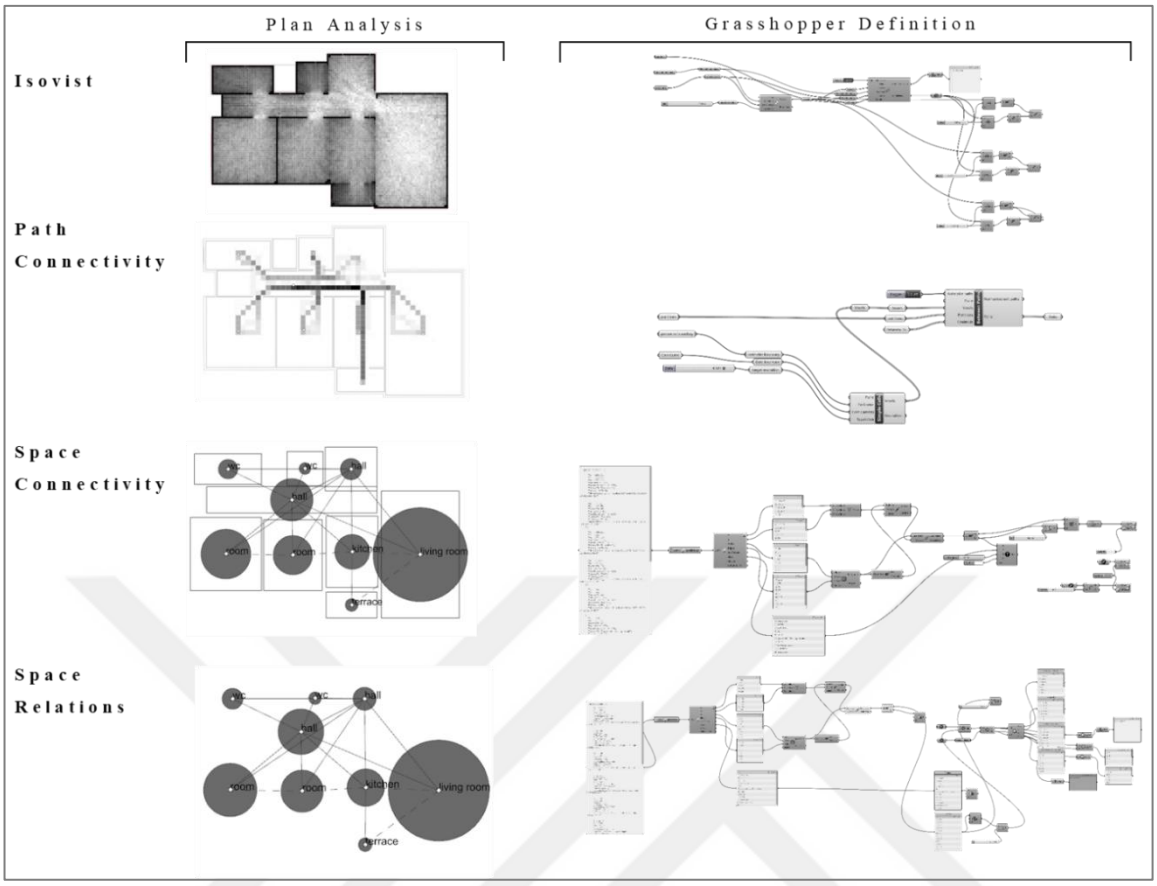


Figure 5.10. Plan analysis results and rhino/grasshopper definition (prepared by author)

The data obtained from these analyses were processed into an Excel spreadsheet. The processed data contains rules for designers to facilitate decision-making. 45 photos were chosen from the Pix2pix estimated images to establish these rules. The BBOX detections of these images were obtained using the object detection process. In converting these BBOXs into Json data, mirror operation was applied on the x and y axes while transferring them to Rhio/Grasshopper. This process was not changed because of axial symmetry. Figure 5.11 and Figure 5.12 shows most of the plans shown. Other plans are included in the appendices.

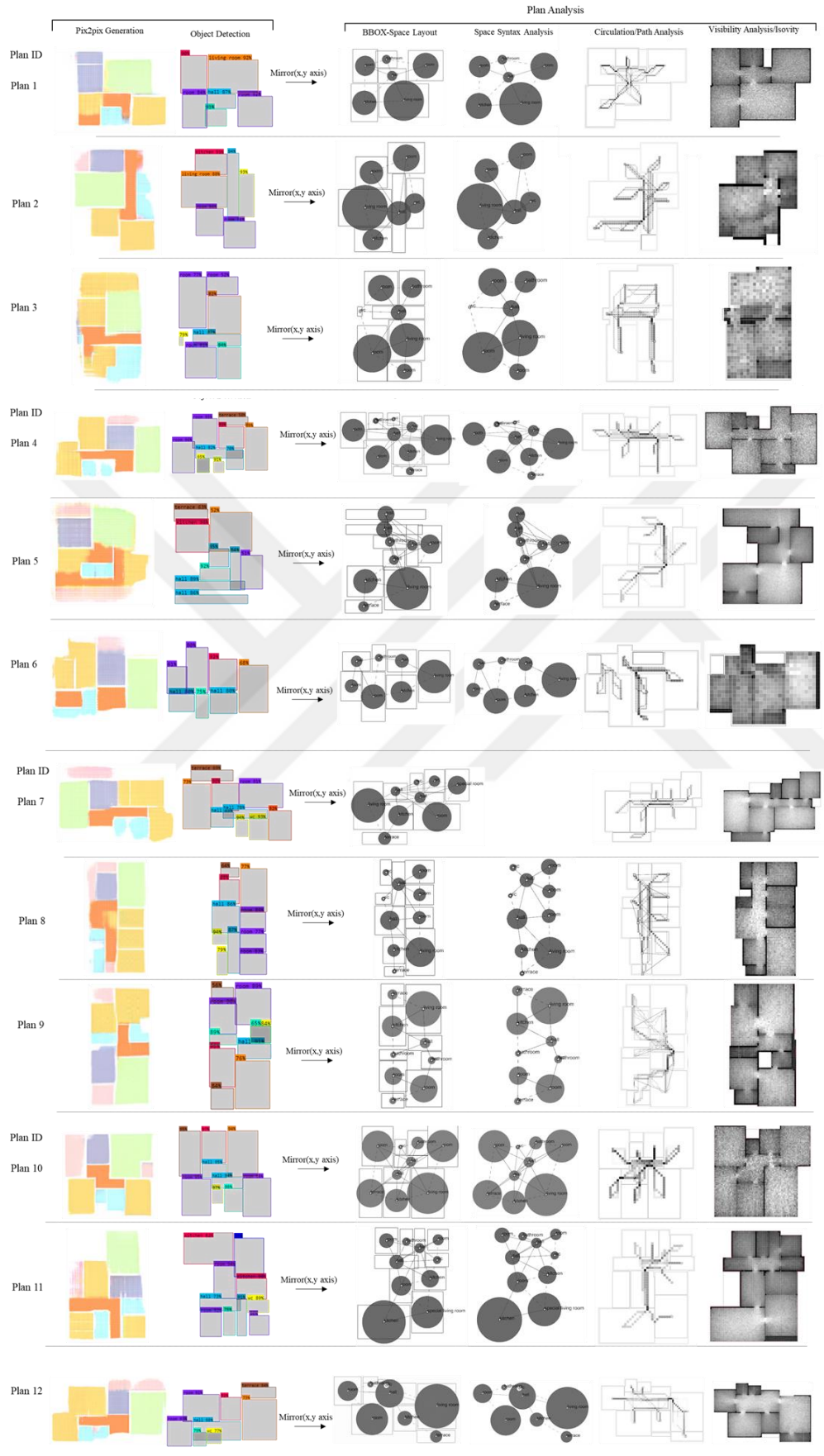


Figure 5.11. The process from predicted ımages through pix2pix to plan layout analysis

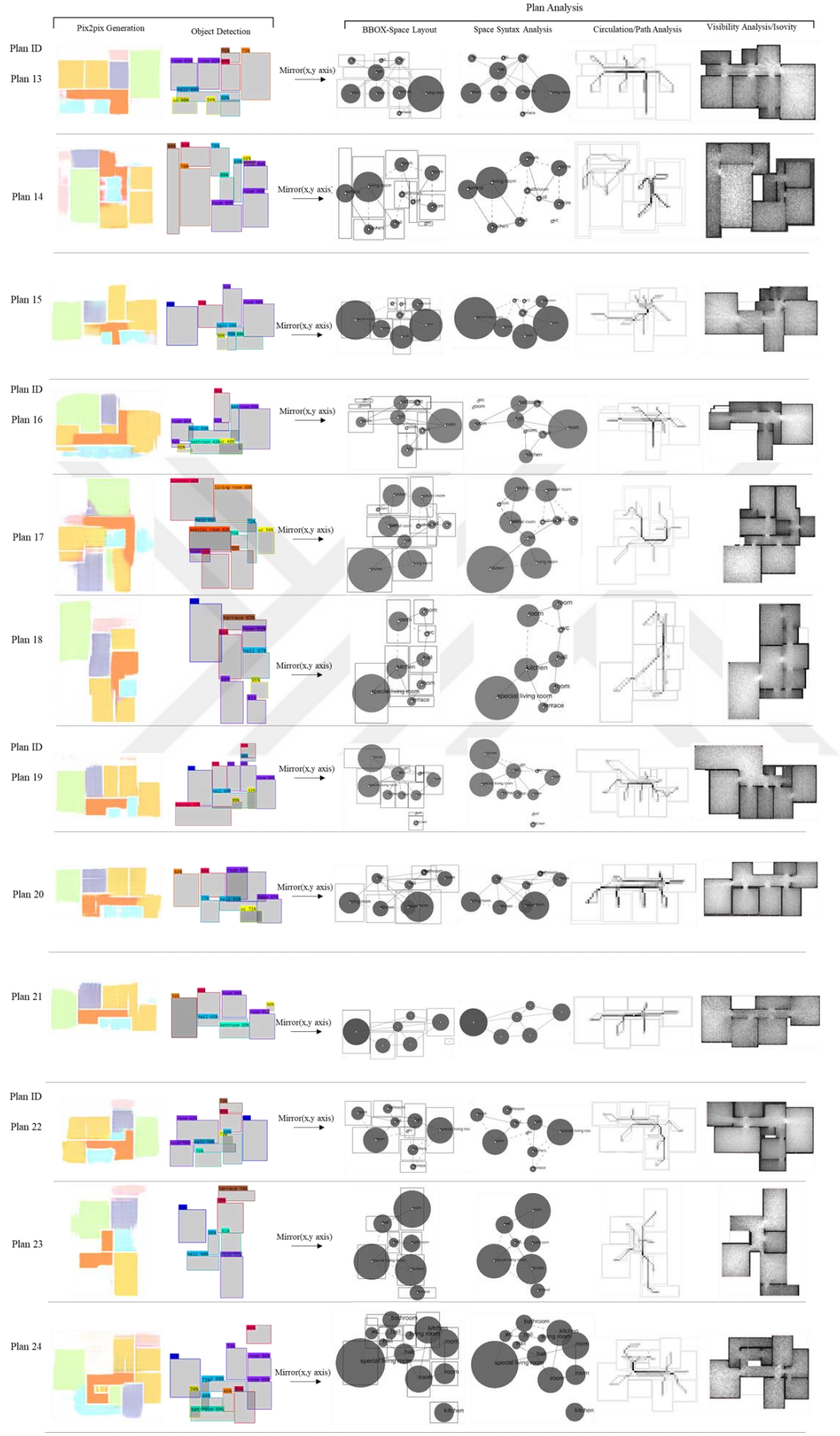


Figure 5.12. The process from predicted ımages through pix2pix to plan layout analysis

5.3. Classification

In order to benefit from the decision support system, architects must assess the quality of space potential associated with the architectural plan arrangement. As we mentioned, further analysis of "Form And Space Relation, Spatial Relationship, Path Space Relationship" values as spatial quality criterion values can be revealed. An expert evaluated the analyses with Rhino/Grasshopper, a computational design tool. Thanks to this tool, we processed the numerical values we obtained from the analysis as a dataset. The titles in the table in the evaluation process of the plan layouts produced from Pix2pix, whose data was created, are given in Figure 5.17. Function names are generated from the place names used in the labelization process. This data was also obtained from JSON data in a defined way for each plan. Circulation connections that provide continuity between spaces were identified as another parameter of spatial quality to be measured.

For this reason, interior walls and openings (interior walls and openings were created by the predicted visual) were drawn to determine the circulation connections with the highest probability. In particular, the "hall" circulation connection, which is present in most of the plans produced with the training of TOKI plans, is present in some of the plans produced with open plan training. In some plans where these circulation connections are absent, the spatial limitation "hall" is not defined simply because the spatial boundaries of the circulation are not clearly defined. For this reason, an evaluation parameter was created according to the number of spaces with a direct connection to common space "Spaces linked by Common Space," which is an essential factor in the user's or designer's decision-making, the "hall," which is the space defined in the circulation connections, and the number of connections of other spaces associated with this space. According to this number, depending on the number of spaces, it can be observed that if the number of spaces connected to the "hall" overlaps, it can connect all spaces. In other possibilities, if more than one circulation connection is defined in spatial diversity, the plan layout system emerges according to those diversities.

For this reason, whether the circulations are continuous or partial is included as separate parameters as 'Continous Circulation' and 'Partially Circulation' outside the "hall" space limitation. "Path" is an evaluation parameter if there are single or more (partial) circulations in the plan layout. "PathBranches" is the parameter calculated as the path lengths between

the spaces in the simulation run through Grasshopper. The higher this numerical value, the larger the space is, or there is no easy access. Most of the time, it is possible to see that the spaces are intertwined, allowing the transition from one space to another. Such parameters are processed as numerical data as "Interlocking Space." These data were obtained mainly in BBOX detections due to Object Detection estimations. Clear boundaries between spaces and unclearly defined boundaries between spaces are very close to reality in BBOX. Finally, to measure the spatial quality, the openness of the space was considered. The more vertical and horizontal elements that divide the plan layout, the more compactness/enclosure of the space is increased. This parameter also decreases the quality of the space. The evaluation of this situation is included under the "Enclosureness" parameter. Finally, an isovity analysis was performed to assess the visibility of these plans defined by their interior and exterior walls. Here, this analysis was made by predicting that spacious spatial areas with high visual access and physical access in the interior can be defined in human perception. By averaging these value ranges for each plan, it was defined that the higher ones offer high-quality spaces in terms of the "visibility" parameter.

Table 5.4. Analysis parameters for rule-based classification

'PlanID', 'Room', 'Bathroom ', 'Hall', 'Kitchen', 'Living_room',
'Terrace', 'Wc', 'Special _room', 'Spaces_l_Common_Place',
'Continous Circulation', 'Partially Circulation', 'Path',
'PathBranches', 'Visibility', 'Enclosureness', 'Interlocking Spaces'

The processed data were imported into Google Colab for the rule-based selection algorithm and using functions created within the determined rules. Appendix.10 contains the imported data table (Appendix.10). In Table 5.4, we see that all the parameters above are evaluated within certain constraints and conditions and used to create rules. All analyzed plan layouts are classified for each rule.

Table 5.4. Rule Definition for Classification

Rule1	def rule_01_room_based(row):
	if row['Room'] == 0 or row['Bathroom '] == 0 or row['Kitchen'] == 0:
	return "The plan is not a house plan"
	elif row['Room'] >= 1 and row['Bathroom '] > 0 and row['Kitchen'] >= 1 and row['Living_room'] >= 1:
	return "This plan could be a house plan/livable"
	else:
	return "NONE"

Rule2	def rule_02_circulation_based(row):
	if row['Continous Circulation'] == 1 and row['Partially Circulation'] == 0 and row['Path'] == 1 :
	return "Circulation movement is continuous"
	elif row['Continous Circulation'] == 0 and row['Partially Circulation'] > 0:
	return "Circulation movement is partially separated"
	elif row['PathBranches'] <= 42:
	return "This plan is suitable for accessibility."
	else:
	return "The plan does not exist"

Rule3	def rule_03_functional_based(row):
	if row['Room'] >= 1 and row['Bathroom '] >= 1 and row['Kitchen'] >= 1 and row['Living_room'] >= 1:
	return "This plan is a house plan"
	elif row['Room'] >= 1 and row['Bathroom '] >= 1 and row['Kitchen'] >= 1 and row['Living_room'] >= 1 and row['Terrace'] >= 1:
	return "This plan includes a terrace"
	elif row['Living_room'] >= 1 or row['Special _room'] >= 1 and row['Bathroom '] >= 1 or row['Wc'] >= 1 or row['Terrace'] >= 1:
	return "This plan could be a different plan/but not for long term living"
	else:
	return "NONE"

Rule4	def rule_04_spacerelation_based(row):
	if row['Hall'] >= 2 and row['Path'] == 1:
	return "This plan could be open house plan"
	elif row['Hall'] >= 1 and row['Bathroom '] >= 1 and row['Kitchen'] >= 1 and row['Living_room'] >= 1 and row['Spaces_l_Common_Place'] >= 3:
	return "The hall constitutes the main circulation connection."
	elif row['Hall'] >= 1 and row['Path'] == 1:
	return "This plan could be compact."
	else:
	return "NONE"

Rule5	def rule_05_spacelayout_based(row):
	if row['Spaces_l_Common_Place'] >= 3 and row['PathBranches'] > 42:
	return "This plan could not be suitable for accessibility/spaces are big and complex relationships ."
	elif row['Path'] == 2 and row['Partially Circulation'] > 0:
	return "This plan is NOT suitable for accessibility/The spaces are divided and have separate entrances."
	else:
	return "NONE"

Rule6	def rule_06_spacelayout_based(row):
	if row['Interlocking Spaces'] >= 1 and row['Enclosureness'] < 3:
	return "This plan could be unique."
	elif row['Interlocking Spaces'] == 0 and row['Enclosureness'] < 3:
	return "This plan could be unique."
	else:
	return "NONE"

Rule7	def rule_07_spacelayout_based(row):
	if row['Visibility'] >= 0.5 and row['Enclosureness'] <= 2:
	return "Space quality is high in visible conditions."
	elif row['Visibility'] < 0.5 and row['Enclosureness'] > 2 and row['Enclosureness']< 5:
	return "Space quality is low"
	elif row['Visibility'] < 0.5 and row['Enclosureness'] == 5:
	return "Space quality is very low"
	else:
	return "NONE"

The classification results obtained according to the specified rules are presented in Table 5. 5. The evaluation of these results was carried out with verbally definable statements. According to the result graph in Result_1, "Room, Bathroom, and Kitchen" are the functions that should be present in standard housing plans under minimum conditions. However, as the result of this rule reveals, Pix2pix has produced plan layouts that still need to have these three functions in most of the plans. Eighteen plan layouts meet these conditions, and 18 do not meet these conditions at all. The "NONE" class plan layouts contain definitions outside the rule's constraints. In the Result_2 result graph, results are drawn according to the rule function that can be inferred about the fragmented, continuous circulation and its accessibility in the whole space. Thirty-seven plan layouts were found to have continuous circulation, and two plan layouts had the highest accessibility. According to Result_3 evaluation, to determine the plan diversity, it is said that if the functions that should be included in the standard housing plan meet the minimum conditions, it can be a housing plan.

In contrast, a definition has been developed based on interpreting different plan layouts outside the standard that offer other spatial function diversity. According to the Result_5 evaluation, it was determined that 26 of the plans have large volumes and complex spatial relationships. For this reason, they were considered weak in terms of accessibility. Four plans were analyzed as two different spaces within the same volume due to their separated circulations and the lack of interaction. While measuring this, as a rule, the fragmentation of the circulation was taken as a basis. In Result_6, it was determined that the plan layout and 17 of the realized plan examples were at this level with the determination that there are many intertwined spaces. Despite this spatial relationship, the whole space has an open space. Few dividing elements separate the spaces due to the low level of "enclosures."

In Result_7, it is seen that the plans in which the height ratios of the viewing angles in the interior space and the spatial organizations in which the open space relationship is established are evaluated as high in terms of spatial quality. While 15 plans were found to be at a reasonable level in terms of spatial quality by meeting both of these two constraints, it is observed that the plan layouts evaluated as compact are evaluated as low enough not to be recommended in terms of spatial quality if the visibility analysis is also low. Six plans meet this constraint.

Table 5.5. Classification results

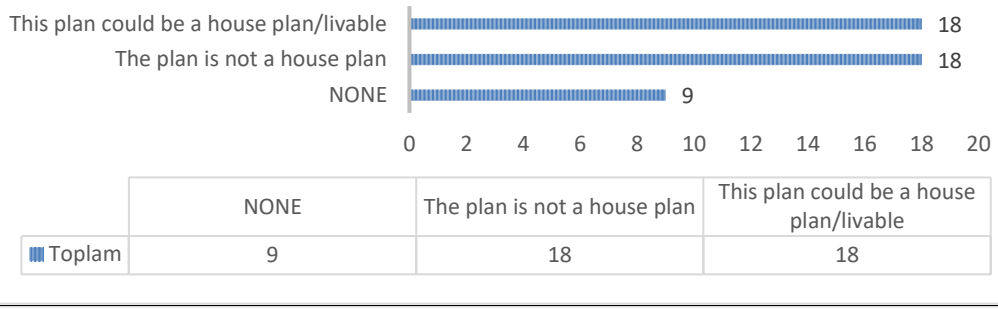
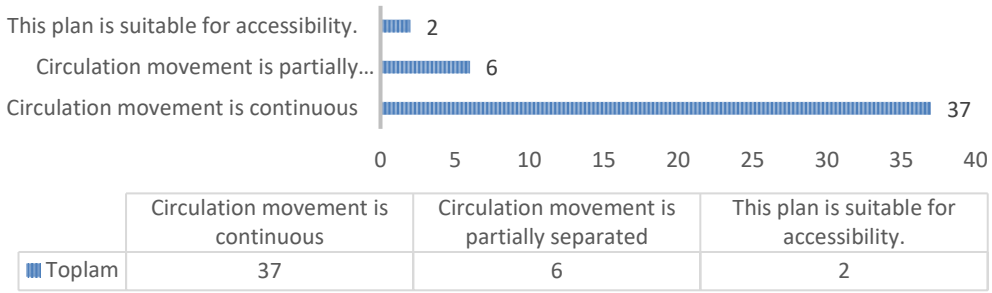
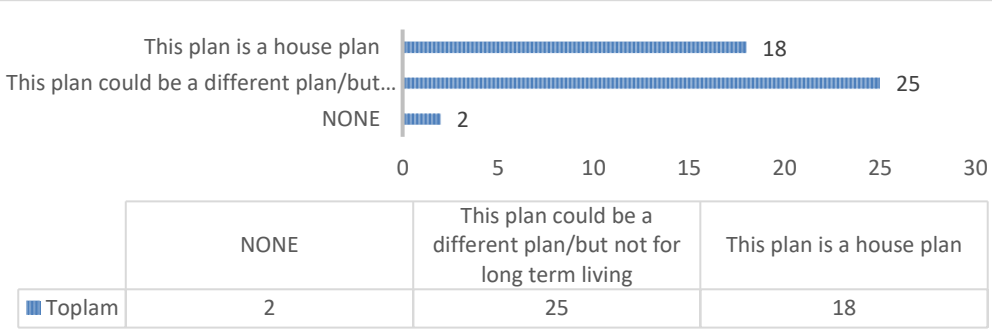
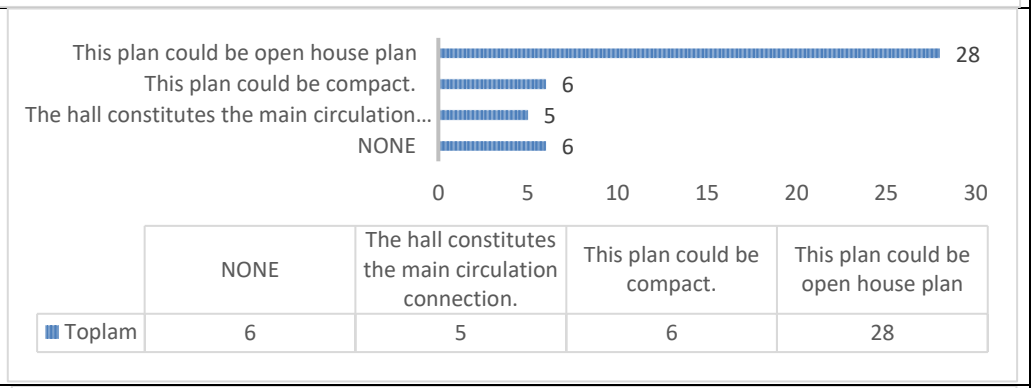
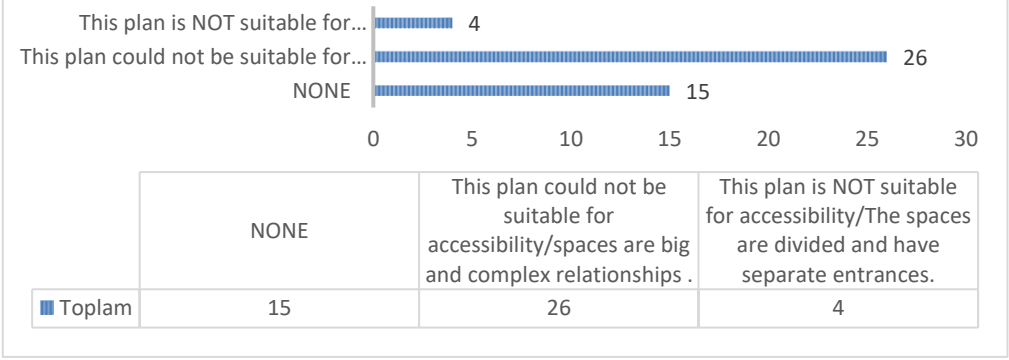
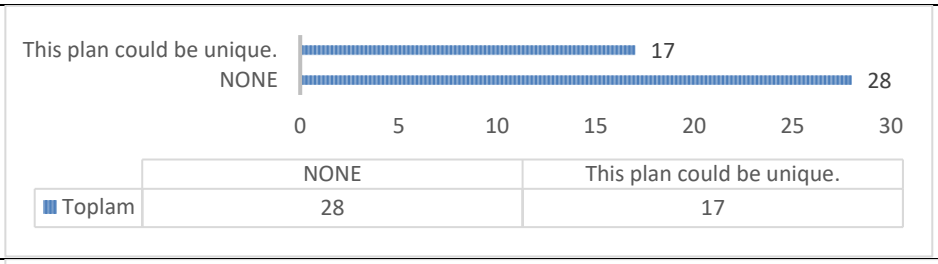
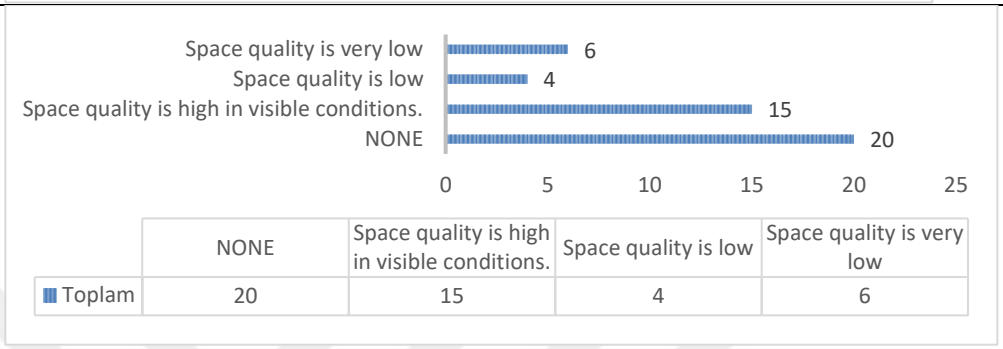
Result_1	
Result_2	
Result_3	
Result_4	
Result_5	

Table 5.5. (Continued). Classification results

Result_6	
Result_7	

All charts are essential in measuring the consistency of rules and enabling the user to make decisions on the generated plan drafts quickly. The classifications demonstrate this importance. This system, in which the desired plan drafts can be determined with the selection tools of the specified parameters, has also created a more predictable structure. Figure 5.18 shows a graph of all the plans that are data for classification, their PlanIDs, and the class to which they belong according to the results of the rules.

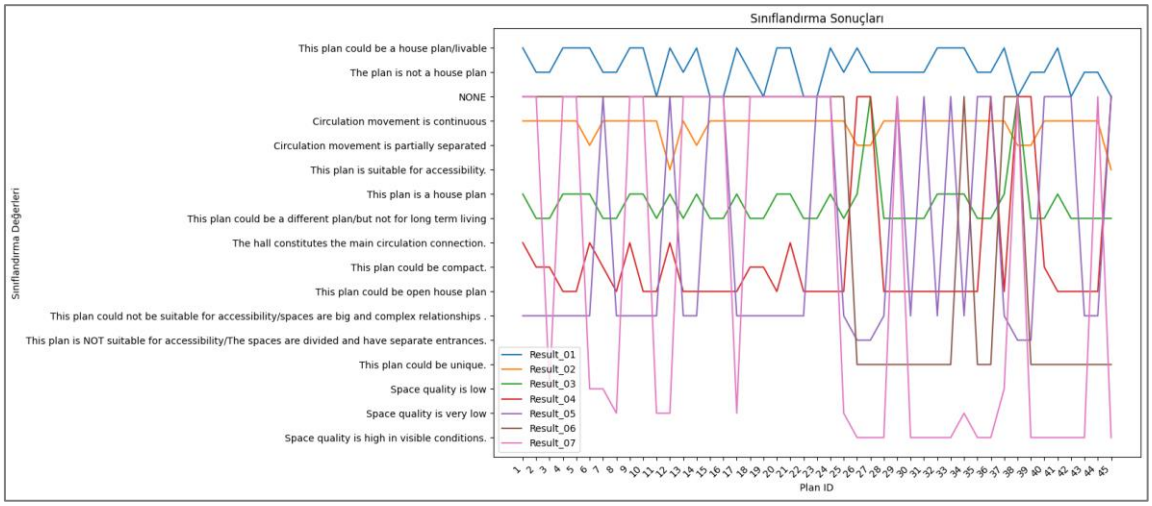


Figure 5.14. Chart showing which class they fit into according to which rules according to their PlanID

In interpreting this line graph, selecting and evaluating the intersection of positively selected plan identities (plan layouts) in all rules is essential.

6. CONCLUSION AND RECOMMENDATIONS

Within the scope of this thesis, studies have been carried out on the contributions of AI technologies, the latest technology today, for designers and architects. In the theoretical research, starting with the human way of thinking and access to knowledge, the thinking processes of designers and architects and the methods of design realization using the knowledge they have acquired have been examined. In the context of the design and the designing process, the point is that once the objectives have been achieved, the action is toward the final product, they become a means to something else is emphasized in the research of Adorno & Horkheimer (2010) and Babadağ and Haşlakoğlu (2021b). According to Schön (1992), this design process consists of a cycle in which the designer makes inferences based on the relationship between purpose and result and continues in a continuous change process. With the interpretation that representation can be constantly added and produced, designers have played an active role in developing and producing representational tools. The knowledge that the designer, the architect, acquires in 'designing' increases in parallel with the productive state of design.

For this reason, architects have changed their representation methods, first with 2D and 3D software and then with computable design tools in the digital age concerned with producing a productive design model. With these tools, the ability to think and produce has changed methodologically. With computational design tools, original and productive design models based on multiple alternatives have started to be developed. With the proliferation of alternatives, the designer and the architect have been able to choose among the alternatives. Because, within the specified parameters, designers and architects can reach their final designs by choosing alternatives. In this context, the issues of what knowledge is and how it is obtained and produced through representation play an essential role.

There are various definitions of space; experiential and perceptual diversity has limited spaces to three-dimensional geometric definitions and revealed different perspectives of space. In spatial organization and design, designers have changed their way of thinking based on the view that the transformation of form and function follow each other with the modern period, taking into account user behavior and perceptions. Housing design has gained different meanings over time, especially with the influence of the modern period, with various discussions and design approaches to define people's personal and private spaces.

People's perception of privacy changed as they lived in the city. With it, their spatial perception wholly changed in the transformation to functional and simple spaces that fit the lines of the modern movement. With the transparent and fluid circulation of the spaces, it has been observed that the elimination of indoor-outdoor distinction and the change of boundaries have been observed. With the change in how people live in the city with the postmodern period and the differentiation of needs, the act of living together collectively and the perception of collective housing emerged, and housing designs were tried to be developed based on minimum living needs in minimum space limits. Typologies limit the variability and diversity of spatial quality. The architect can go beyond these limits by adding different original values in housing design. At the same time, he/she can choose from multiple alternatives for himself/herself by making a quick decision. This way of thinking and designing is realized with the latest technology, artificial intelligence (AI) algorithms. With the increase of 'knowledge accumulation' in the digital world, 'big data' has provided opportunities for machines to learn and designers, as in many disciplines, to discover new productive methods and change the decision-making process.

According to Leich (2022), AI is an essential tool for understanding the thinking processes of architects and how they develop their thinking. The primary objective is to determine how the 'learning algorithms of artificial intelligence,' which humans try to mimic their learning style, work and produce, how they transform the data they use, and how they reinterpret the results they generate together with other systems.

This study compares artificial intelligence (AI) learning styles and their sub-sets, machine learning, and deep learning algorithms from a general framework to specific examples. The relationship between the learned information and "big data" is analyzed through the way big data is defined, its number, and the way it is parsed. This method also uncovers the learning technique used with big data and how it affects the results of the trained model. Recent research has determined that GAN networks are the most often utilized DL algorithm at architecture and machine learning interfaces. In particular, plan layout generation methods are realized with these networks. Our study covers creating and evaluating the plan layouts of housing designs that architects intensively examine. The tools used in plan layout generation are the DCGAN and Pix2pix algorithms. It has been observed that the generated/predicted plan layouts are trained and tested by training and testing the networks using a dataset of visual images. The plan layout generation is realized with the generated

visual images and the tested images. Here, two different datasets were created for comparison. A module of TOKI plan layouts is used in the first dataset, which consists of housing plans, while the other dataset contains the plan layouts of open-plan housing designs. The preparation of these datasets varies according to the algorithm. This is because the dimensions, perceptions, and methods by which each algorithm reduces visual images vary. In the process of evaluating the generated images, we obtained loss values so that we could understand how the networks work. However, these values do not change the generated images 'real' or 'fake' status. The machine cat perceives the generated image as ' real' when it evaluates it at a level similar to the dataset it receives as input and perceives it as 'fake' when it is different. This is basically how GAN networks work, but since DCGAN is the first basic GAN algorithm, algorithms such as Pix2p.

In the datasets prepared for DCGAN, the plan layouts were labeled and separated with different colors based on the functions in the plan layouts. Subsequently, the number of datasets grew in tandem with the understanding that the number of datasets is essential for the network to be trained correctly. It is known that this algorithm works on two main models: 'generator' and 'discriminator.' Both models pass all visual images through convolutional padding processes. The discriminator detects the image generated by the generator, evaluates whether it is similar or dissimilar to other images in the given train set, and identifies the discriminated image as 'real' or 'fake' based on this 'similarity' or 'dissimilarity.' If the image is 'fake,' this process continues again. This basic logic of parsing and merging, identifying similarities and differences, is based on CNN algorithms. Human learning is similar in that it involves identifying and classifying similarities and differences and converting them into information. However, the visual image produced by the network is no different from any other image. The epoch values increased in the DCGAN plan layout generation when we looked at the loss value graphs and the generated plan layouts. According to the defined formula, the closer the discriminator and generator values are to each other, the more 'real' the generated image is defined. At 100 epoch values, this could not be realized for both datasets. At 200, 500, and 1000 epochs, it was found that more explicit images were finally generated, and more favorable loss values were observed. However, in the process we call 'overfitting,' it is observed that when the network performs more training processes with the same dataset, the tested images do not differ. This can also be seen in the graphs. In the TOKI plan dataset training, 1000 epochs cause the network to perform 'overfitting.' A value of 500 epochs can provide ideal production. The 'Training

Open_Plan dataset' shows that it can reach the overfitting value at 500 epochs because the dataset is more minor. Between 200 epochs and 500 epochs, it can perform a more favorable generation. After applying this basic GAN algorithm(DCGAN), it was found that none of the generated plan layouts could generate fuzzy and fully clear plan layouts with the noisy vector factor used in the network training process. For this reason, we continued our study with the Pix2pix algorithm for the plan layout generation process.

The Pix2pix algorithm was implemented by preparing, processing, and augmenting the dataset, ultimately for the algorithm. In order to bring the datasets to the appropriate environment, they were applied to the labelization process in DCGAN, and color allocations and function definitions were brought to the spaces by the functions. In addition, in two different datasets of housing plans, the issue of how the visuals generated by converting the structure into an input value, especially the divider elements that are important in the division and delimitation of spaces, can be produced in framed or non-framed systems has also been examined. The transition processes of the data we generate in the U-NET and Patch-GAN parts, which are essential components of the Pix2pix algorithm, become necessary for separating and correctly identifying these algorithms (generator and discriminator). The generated loss values represent the metric values between the input, ground truth, and predicted image, indicating whether the algorithm has been trained correctly. The input and ground truth images must be prepared by merging 512*256 during the dataset's creation. In completing this process, three different types of datasets were created, and we also created differences within the datasets. The ground truth, where the background is entirely white so that the walls and structure are perceived as white, and the ground truth, where the background is black for the walls, are also perceived as black. The dataset is prepared as input (space boundaries are specified, with walls) and ground truth (labeled version of the original plan layout/black background or white background). In addition, different datasets were created as paired-unpaired in the input-ground truth combination while bringing the size to 512*256. In paired datasets, the input and ground truth data are identical; in other words, they are generated from the same plan layout. In this dataset, the data augmentation is only created by synchronizing the data at 90, 180, and 270 degrees. In the unpaired dataset, instead of input-ground truth matching, the dataset is augmented by crossing an input with more than one ground truth. These data augmentation methods were applied for both TOKI and open_plan layouts. While creating the dataset, the merging operations and data

augmentation operations for the pix2pix algorithm were performed in Python on the Google Colab platform. In each dataset, these variations were included as 3 case studies.

In the algorithm, in the complex process, the images were fragmented (upsampling and downsampling) thanks to the U-NET algorithm. The generator applies both of these processes here. Then, the resulting images go through the bottleneck process, which takes place between the encoding and decoding processes. PatchGAN is in the evaluation stage of reassembling the fragmented images. The primary purpose of PatchGAN is to check the overall compatibility of the generated images and to provide a detailed level of realism. This makes it possible to provide more localized and detailed feedback by evaluating each patch separately. This approach can be efficient in applications where attention to detail in some areas of the images is essential.

The result graphs evaluate the generated plan or predicted layouts according to their loss values. All these loss functions are evaluated to measure and understand whether the network model is well trained, whether there is a loss of data from the ground truth while training the dataset, whether the network is over- or under-trained (overfitting), and whether the generator and discriminator are working well (understanding the difference between real and fake images). To assess the training outcomes of all datasets, all trained datasets were evaluated at 100 epochs. When the result graphs and predicted layouts of the trained datasets are evaluated, it is observed that the generators and discriminators of the networks trained with paired datasets generally work well, and the image quality of the result outputs is high. Even though the predicted images deviate from the ground truths, it is observed that the indoor plan layout is generated by preserving the given input boundaries. These conclusions were reached for both plan models. However, based on the paired images on which the TOKI plan layouts were trained, it was observed that the visual quality was higher, and the plan layouts were more expressive with colors. In the unpaired and crossed datasets training results, the fact that the generated images, which the generator perceives as different from the ground truth while performing network training, do not resemble the discriminator part and do not start to express a value. This is because all the generated/predicted plan layouts in this format were found to generate completely independent and dissimilar images from the ground truth within the boundaries of the input data. The aim is to determine how different and unique the generated/predicted images (plan layouts) will be when the ground truths and datasets are differentiated. The visuals produced from unpaired images had high

incompleteness rates and needed to be completed in terms of resolution, resulting in low-quality plan layouts in terms of readability.

However, the evaluation process of all these generated/predicted plan layouts should not be limited to loss values and the network's measurement metrics. The fact that the generated images are plan layouts and are generated with spatial descriptions does not make them quantifiable. For spatial quality to be quantifiable or analyzable, the parameters of Ching (2023) were used. These parameters are the relationship between spaces, the effect of form and function on the relationship between spaces, and the accessibility of spaces. These parameters were used to objectively measure and compare the quality of spaces in the information assessment process. The parameters the architect examined, especially in designing the house, and evaluated while deciding on the final design were evaluated as deep learning algorithms handled them. It is predicted that decision support systems that can contribute to the architect's temporal process and shorten the process can change positively with the support of DL algorithms. Whether the results of DL are goal-oriented should be the architect's decision and should be concluded with the parametric evaluations it brings. However, when these algorithms are evaluated in terms of 'explainability' and 'interpretability,' they provide comprehensibility with the contribution of expert opinion.

- In order to spatially evaluate the plan layouts produced by GANs, the question of whether the results produced by this network system are identifiable and measurable when checked by another network system was sought to be answered.

Object detection algorithms analyzed the generated plan layouts to see if the spaces divided by functions in the plan layouts could be detected.

- Here, it is emphasized that another artificial intelligence algorithm (Object Detection) can identify the images produced by an artificial intelligence algorithm (Pix2pix/cGAN), and another system can verify and explain the system.

The Object Detection process is based on the CNN algorithm, a classification algorithm. The label process/annotation process stage that creates the dataset is essential. Because the machine evaluates what to identify and how to recognize class discrimination in a dataset for this reason, we reprocessed all the datasets we used in the training of the pix2pix dataset

through the Roboflow interface and performed the classification of the plan layouts of all datasets based on the functions. In order to achieve the appropriate number of the dataset, the object detection method in different algorithms was implemented by the flip operation at certain angles, and these algorithms were compared. In the Google Colab environment, the information from the dataset created in Roboflow was extracted, and Yolov5 and Yolov8 algorithms were used. Of these algorithms, Yolov8 gave more accurate results in classification and plan layout definition depending on the measurement metrics. At the same time, Roboflow's internal training model was also tested. The confusion matrices, Accuracy, Precision, Recall, and F1 values were compared, and it was found that Roboflow identified well at the same epoch value. Thanks to the simplicity of its interface, the detection of the tested images was carried out quickly. The .json extensions of the BBOX data, which determine the prediction rates, names, and locations of all predicted/detected places, could be obtained. This process was applied for each plan investigated. The accuracy rates of the detected/predicted functions/classes are high. However, by converting the coordinate systems of the BBOX rectangles into .json data from center point to corner sub-coordinate coordinates, the first stage of transforming the plan layout into a measurable state was realized.

All spatial plan layout data that reached the measurable stage was converted from .json data to a tangible plan layout view through Rhino/Grasshopper, a computational design tool. All data transferred here remained the same. Because the BBOX class detection rectangles extracted from the .json data provide simultaneous access to the 'x,y coordinates, and width and height length' values, the detected 'class' values (room, living room, WC, bathroom, etc.) that are, their functions and the 'area' measurement values of these spaces. In transforming the plan layouts into an analyzable state to make them fully measurable, plan layout generations were carried out depending on the locations and openings of the interior and exterior partition boundaries (walls) in the predicted visuals.

- It emphasizes how the designer and machine learning can collaboratively generate, interpret, and decide on layouts and the machine as a tool.

Collecting these analyses under the headings of space connectivity and space relations is essential to comprehend the spatial organization and relationships between spaces. Information such as space IDs, names, locations, and the number of spaces obtained from

JSON data were used to understand space connectivity and space relations. The space syntax method was applied in the Rhino/Grasshopper program to help designers understand the difference between visuals, and more than one method was tried in this process.

Visibility analysis determines the width of the viewing angle in spatial perception by showing the interior space's openness and evaluating the spatial organization's quality. Using the Grasshopper/PlanBee plugin, the quality of space was measured through analysis. The analysis results were evaluated by averaging the remapped values between 0 and 1 to determine the overall visibility values of the plans that provide more positive visibility. Thanks to all the analysis methods applied, all parameters and the analysis results obtained are shown in an Excel spreadsheet. The contribution of this data to the evaluation process is essential because it is aimed to evaluate the generated plan layouts according to their spatial quality and, based on these evaluations, to present the plan layouts produced since the production of GAN networks as a decision support system for architects. Decision support systems are included in artificial intelligence algorithms as explainable and interpretable classification methods. When all these methods are evaluated, conditional and multiple rule-based classification processes are emphasized. The rule-based classification system assists the architect in the final decision-making process.

The rule sets consist of functions written in Google Colab/Python and are composed of seven rules. The main objective here is to determine that the intersections of multiple measurable parameters can determine spatial quality or that spatial organization can be evaluated with the inputs resulting from spatial planning, that spatial openness and closeness should be decided by reading spatial openness and closeness together with circulation connections in user perception, and that there may or may not be housing design according to minimum functional needs. All these determinations and evaluations are included with seven rules and separate classifications based on these rules. The result graphs can also illustrate situations that meet more than one rule and belong to more than one class. The results of the rules are expressed as Result_1, Result_2, Result_3, Result_4, Result_5, Result_6, Result_7.

According to Result_1, essential functions such as "Room, Bathroom, and Kitchen," which should be present in standard housing plans, are not present in many plan layouts generated by Pix2pix. In addition, the plan layouts classified as "NONE" consist of definitions outside the specified constraints. In Result_2, conclusions were drawn according to the rule

functions regarding whether the circulation is fragmented or continuous and accessibility in the space. Result_3 and other result evaluations include definitions for different plan layouts. In Result_5, it was found that the plans have large volumes and complex spatial relationships. It is also noted that some plans have different spaces within the same volume, and the circulation is segregated. In Result_6, it was stated that the plan layouts were evaluated depending on the number of intertwined spaces and the lack of divider elements despite being open spaces. In Result_7, it is stated that the plans with a combination of interior viewing angles and open space relationship are evaluated as high in spatial quality. If the visibility of the plan layouts evaluated as compact is low, it is stated that they need to be higher to be recommended in terms of spatial quality.

Given all these considerations, the spatial representation of the plan layouts was decided by a rule-based classification method utilizing the intersection of different parameters. This method relationship suggests the relationship between the architect's opinion and the "explainability" and "interpretability" of the results of algorithms such as 'image to image translation' methods (Pix2pix). The generated visuals' quantifiable, classifiable, and distinguishable qualities can accelerate the decision-making process and reach the goal.

Recommendations

Suggestions for future studies based on the algorithms used and their results;

- It is known that algorithms such as DCGAN and pix2pix, developed in the last ten years, are still used in many studies as image-to-image translation methods and in the architecture interface. However, in addition to these algorithms in GAN networks, algorithms such as Pix2pixHD (image-to-image translation) and HauseGAN have been added and developed. The HouseGAN algorithm, as a breaking point, can generate outputs with graph-based representation layout training by including more than one algorithm and GCN/GNN algorithms. This algorithm was also tested at one stage of the study. However, the algorithm could not be retrained as it was difficult to integrate our dataset. HouseGAN++ algorithms were also added to this algorithm.
- Together with these algorithms, research has been carried out to create ready-made web platforms and plan layout generation by converting them into user interfaces.

- While conducting the study, many studies on 3D generation were encountered. Models such as FrankenGAN are being developed, where space layouts are also generated from 3D models. Similar 3D space layout generation generations can be realized.
- Classification algorithms are constantly evolving to get speedy results. However, looking at the recent studies known as RuleXAI, it is predicted that this algorithm combined with rule-based classification can be used as an evaluation method as an explainable artificial intelligence algorithm combined with rule-based selection.
- During the study, it was observed that machine learning tools (plug-ins) have been integrated into the Rhino/Grasshopper program in the last ten years. It has been observed that these algorithms, which are still used as classification and decision support systems, are increasingly being combined with productive systems.

In addition to the proposed algorithms, housing designs are evaluated in this thesis as a suggestion for generative design models. However, GAN networks can be used for image completion processes and plan layouts. With these networks, plan layouts and sections other than housing plan layouts can also be used in educational tools. In parallel with such studies, the plan layout productions of site plans and layout layouts of interior furnishings have also been included in the processes produced with GAN networks. Here, the question arises as to why the visuals to be produced are produced and for what purpose they will be used. In future studies, new suggestions can be made on how the networks work and how the results should be evaluated, especially in plan layout productions as an evaluation criterion.

REFERENCES

Ahuja, S., and Chopson, P. (2020). Automation and Machine Learning in Architecture: A New Agenda for Performance-Driven Design. *Architectural Design*, *90*(2), 104-111.

Akhtar, N., and Ragavendran, U. (2020). Interpretation of intelligence in CNN-pooling processes: a methodological survey. *Neural computing and applications*, 32(3), 879-898.

Akin, O. (1978). How do architects design. *Artificial Intelligence and Pattern Recognition in Computer Aided Design*, IFIP, 65-98.

Akın, Ö. (2006). The whittled design space. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 20(2), 83–88. doi:10.1017/S0890060406060069

Alexander, C. (1977). *A pattern language: towns, buildings, construction*: Oxford university press.

Allan MacLean , Richard M. Young , Victoria M.E. Bellotti andThomas P. Moran (1991) Questions, Options, and Criteria: Elements of Design Space Analysis, *Human–Computer Interaction*, 6:3-4, 201-250

Anderson, C., Bailey, C., Heumann, A., and Davis, D. (2018). Augmented space planning: Using procedural generation to automate desk layouts. *International Journal of Architectural Computing*, 16(2), 164-177. doi:10.1177/1478077118778586

Arciszewski, T., Michalski, R. S., and Dybala, T. (1995). STAR methodology-based learning about construction accidents and their prevention. *Automation in Construction*, 4(1), 75-85.

Argota Sánchez-Vaquerizo, J., and Cardoso Llach, D. (2019). The Social Life of Small Urban Spaces 2.0: Three Experiments in Computational Urban Studies. In *Computer-Aided Architectural Design." Hello, Culture" 18th International Conference, CAAD Futures 2019, Daejeon, Republic of Korea*, June 26–28, 2019, Selected Papers 18 (pp. 295-310). Springer Singapore.

Arslan, A. (2007). *İlkçağ felsefe tarihi 3: Aristoteles*. İstanbul: İstanbul Bilgi Üniversitesi Yayınları, 7.

As, I., Pal, S., and Basu, P. (2018). Artificial intelligence in architecture: Generating conceptual design via deep learning. *International Journal of Architectural Computing*, 16(4), 306-327. doi:10.1177/1478077118800982

Atademir, H. R. (1974). *Aristo'nun mantık ve ilim anlayışı*. Ankara: Ankara Üniversitesi İlahiyat Fakültesi.

Babadağ, M., and Haşlakoğlu, O. (2021a) . Bilgi ve Değer Bağlamında "İyi Tasarım" Kavramının İzinin Sürülmesi. *Kaygı. Bursa Uludağ Üniversitesi Fen-Edebiyat Fakültesi Felsefe Dergisi*, *20*(1), 171-192.

Babadağ, M., and Haşlakoğlu, O. (2021b). Tasarımda Değer-Yarar İlişkisi ve Temsil Problemi Üzerine Bir İrdeleme. *Planlama*, *31*(3).

Baker, U. (2017). *Yüzeybilim fragmanlar*. İletişim Yayınları.

Bachelard, G. (2013). *Bilimsel zihnin oluşumu*. Baskı,(Çev: A. Tümertekin) İstanbul: İthaki Yayınları, İstanbul.

Banham, R. (1980). *Theory and design in the first machine age*. MIT press.

Baybars, I. (1982). The generation of floor plans with circulation spaces. *Environment and Planning B: Planning and Design*, *9*(4), 445-456.

Baykan, C. A. (1991). *Formulating spatial layout as a disjunctive constraint satisfaction problem* (Doctoral dissertation, Carnegie Mellon University).

Behrends, E. (2000). *Introduction to Markov chains* (Vol. 228). Braunschweig/Wiesbaden: Vieweg.

Bhasker, J., and Sahni, S. (1987). A linear time algorithm to check for the existence of a rectangular dual of a planar triangulated graph. *Networks*, *17*(3), 307-317.

Bidgoli, A., and Veloso, P. (2019). DeepCloud. The Application of a Data-driven, Generative Model in Design. *arXiv preprint* arXiv:1904.01083.

Biggs, N., Lloyd, E. K., and Wilson, R. J. (1986). *Graph Theory, 1736-1936*: Oxford University Press.

Boudon, P. (2015). *Mimari mekan üzerine: mimarlık epistomolojisi üzerine deneme*. Janus Yayıncılık.

Braun, A., and Borrmann, A. (2019). Combining inverse photogrammetry and BIM for automated labeling of construction site images for machine learning. *Automation in Construction*, 106, 12. doi:10.1016/j.autcon.2019.102879

Brown, N. C., and Mueller, C. T. (2019). Design variable analysis and generation for performance-based parametric modeling in architecture. *International Journal of Architectural Computing*, 17(1), 36-52.

Brugnaro, G., and Hanna, S. (2017, October). Adaptive robotic training methods for subtractive manufacturing. In *Proceedings of the 37th annual conference of the association for computer aided design in architecture (ACADIA)* (pp. 164-169). Acadia Publishing Company.

Cai, B. Y., Li, X., Seiferling, I., and Ratti, C. (2018, July). Treepedia 2.0: applying deep learning for large-scale quantification of urban tree cover. In *2018 IEEE International Congress on Big Data (BigData Congress)* (pp. 49-56). IEEE.

Cengizkan, A. (2009). Kültür nesnesi olarak konut ve politik aktörlerin arka bahçesi olarak konut üretimi. *Mimarlık Dergisi*, 345(1).

Chaillou, S. (2019). *AI + Architecture* | Towards a New Approach.

Chaillou, S. (2020). *Archigan: Artificial intelligence x architecture*. In Architectural intelligence (pp. 117-127): Springer.

Chan, C. S. (1989). Cognition in design process. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society* (pp. 291-298). Hillsdale, NJ: Lawrence Erlbaum.

Chang, K.-H., Cheng, C.-Y., Luo, J., Murata, S., Nourbakhsh, M., and Tsuji, Y. (2021). Building-GAN: Graph-Conditioned Architectural Volumetric Design Generation. *arXiv preprint* arXiv:2104.13316.

Chang, M.-C., Buˇs, P., Tartar, A., Chirkin, A., and Schmitt, G. (2018). Big-data informed citizen participatory urban identity. In *Computing for a Better Tomorrow: The 36th International Conference on Education and Research in Computer Aided Architectural Design in Europe (ECAADe 2018)* 2(June 2019) (pp. 669–678).

Chang, S., and Jun, H. (2019). Hybrid deep-learning model to recognise emotional responses of users towards architectural design alternatives. *Journal of Asian Architecture and Building Engineering*, 11. doi:10.1080/13467581.2019.1660663

Chatzikonstantinou, I., and Sariyildiz, I. S. (2017). Addressing design preferences via auto-associative connectionist models: *Application in sustainable architectural Fagade design. Automation in Construction*, 83, 108-120. doi:10.1016/j.autcon.2017.08.007

Charman, P. (1993, September). Solving Space Planning Problems. In *Ai'93-Proceedings Of The 6th Australian Joint Conference On Artificial Intelligence* (p. 454). World Scientific.

Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. *arXiv preprint* arXiv:1706.05587.

Choi, J., Gu, B., Chin, S., and Lee, J.-S. (2020). Machine learning predictive model based on national data for fatal accidents of construction workers. *Automation in Construction*, 110, 102974.

Ching, F. D. (2023). *Architecture: Form, space, and order*. John Wiley and Sons.

Chollet, F. (2018). *Deep learning with Python* (Vol. 361): Manning New York.

Clemence, P. (2006). *Mies van der Rohe's Farnsworth House*, Schiffer Pub.

Colomina, B., and Kılıç, A. U. (2017). *Mahremiyet ve kamusallık: kitle iletişim aracı olarak modern mimari.* Metis Yayınları.

Colomina, B., and Risselada, M. (1988). *Raumplan versus Plan Libre, Le Corbusier and Adolf Loos 1919-1930.*

Copeland, B. J. (2004). *The essential turing*: Clarendon Press.

Coyne, R.F., and Flemming, U. (1990). Planning in design synthesis: Abstraction-Based LOOS (ABLOOS). *Proc. Fifth Int. Conf. AI in Engrg.* (Gero, J., Ed.), 1:Design.

Cross, N. (1982). Designerly ways of knowing. Design studies, 3(4), 221-227.

Cross, N. (2006a). Designerly ways of knowing (pp. 1-13). Springer London.

Cross, N. (2006b). Understanding design cognition. *Designerly ways of knowing*, 77-93.

Cutellic, P. (2019). Towards encoding shape features with visual event-related potential based brain–computer interface for generative design. *International Journal of Architectural Computing*, 17(1), 88-102. doi:10.1177/1478077119832465

Çüçen, A. K. (2012). *Bilgi felsefesi*. Sentez.

Dai, W.-M., Sato, M., and Kuh, E. S. (1987). A dynamic and efficient representation of the building-block layout. Paper presented at the *24th ACM/IEEE Design Automation Conference.*

Deleuze, G. (2007). *Kant Üzerine Dört Ders*, çev. Baker, İstanbul: Kabalcı Yayınevi.

Dewey, J. (1908). What does pragmatism mean by practical?. *The journal of philosophy, psychology and scientific methods*, 5(4), 85-99.

Doyle, S., and Senske, N. (2018). Digital provenance and material metadata: Attribution and co-authorship in the age of artificial intelligence. *International Journal of Architectural Computing*, 16(4), 271-280. doi:10.1177/1478077118800887

dPrix, W., Schmidbaur, K., Bolojan, D., and Baseta, E. (2022). The legacy sketch machine: from artificial to architectural intelligence. *Architectural Design*, *92*(3), 14-21.

Dutta, S., Cai, Y., Huang, L., and Zheng, J. (2020). Automatic re-planning of lifting paths for robotized tower cranes in dynamic BIM environments. *Automation in Construction*, 110, 102998.

Duschinsky, R. (2012). *Tabula rasa and human nature*. Philosophy, 87(4), 509-529.

Eastman, C. M. (1968). *Explorations of the cognitive processes in design* (pp. 1-108). Carnegie-Mellon University. Department of Computer Science.

Edmondson, A. C. (2012). *A Fuller explanation: The synergetic geometry of R. Buckminster Fuller*. Springer Science and Business Media.

Elgammal, A., Liu, B., Kim, D., Elhoseiny, M., and Mazzone, M. (2018). The shape of art history in the eyes of the machine. Paper presented at the *Thirty-Second AAAI Conference on Artificial Intelligence.*

Faircloth, B., Welch, R., Tamke, M., Nicholas, P., Ayres, P., Sinke, Y. and Ramsgaard Thomsen, M. (2018). Multiscale modeling frameworks for architecture: Designing the unseen and invisible with phase change materials. *International journal of architectural computing*, *16*(2), 104-122.

Fan, Z., Chen, T., Wang, P., and Wang, Z. (2022). CADTransformer: Panoptic Symbol Spotting Transformer for CAD Drawings. Paper presented at the *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Fan, Z., Zhu, L., Li, H., Chen, X., Zhu, S., and Tan, P. (2021). FloorPlanCAD: a large-scale CAD drawing dataset for panoptic symbol spotting. Paper presented at the *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Foucault, M. (2022). *Bilginin arkeolojisi*. EA Publish.

Feng, D. C., Liu, Z. T., Wang, X. D., Chen, Y., Chang, J. Q., Wei, D. F., and Jiang, Z. M. (2020). Machine learning-based compressive strength prediction for concrete: An adaptive boosting approach. *Construction and Building Materials*, 230, 11. doi:10.1016/j.conbuildmat.2019.117000

Feng, K. L., Lu, W. Z., and Wang, Y. W. (2019). Assessing environmental performance in early building design stage: *An integrated parametric design and machine learning method. Sustainable Cities and Society*, 50, 15. doi:10.1016/j.scs.2019.101596

Feng, Z., Guo, L., Huang, D., and Li, R. (2021, May). Electrical insulator defects detection method based on yolov5. In *2021 IEEE 10th Data Driven Control and Learning Systems Conference (DDCLS)* (pp. 979-984). IEEE.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2009). Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, *32*(9), 1627-1645.

Flagg, H. G. (1964). *Boolean Algebra and its Application*. New York. In: John Wiley and Sons, Inc.

Flemming, U. (1989). More on the representation and generation of loosely packed arrangements of rectangles. *Environment and Planning B: Planning and Design*, *16*(3), 327-359.

Flemming, U., Baykan, C. A., Coyne, R. F., and Fox, M. S. (1992). Hierarchical Generate-and-Test vs Constraint-Directed Search: A comparison in the context of layout synthesis. *Artificial Intelligence in Design'92*, 817-838.

Flusser, V., and Cullars, J. (1995). On the word design: An etymological essay. *Design Issues*, 11(3), 50-53.

Friedman, J. H. (2006). Recent advances in predictive (machine) learning. *Journal of classification*, 23(2), 175-197.

Fyfe, C. (2005). *Artificial neural networks. In Do smart adaptive systems exist?* (pp. 57-79): Springer.

Gao, X., Guo, X., and Lo, T. (2023). M-StruGAN: An Automatic 2D-Plan Generation System under Mixed Structural Constraints for Homestays. *Sustainability*, *15*(9), 7126.

Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems:* O'Reilly Media, Inc.

Giuste, F., Shi, W., Zhu, Y., Naren, T., Isgut, M., Sha, Y., ... and Wang, M. D. (2022). Explainable artificial intelligence methods in combating pandemics: A systematic review. *IEEE Reviews in Biomedical Engineering*.

Goetschalckx, M. (1992). An interactive layout heuristic based on hexagonal adjacency graphs. *European Journal of Operational Research, 63*(2), 304-321.

Goldschmidt, G. (1991). The dialectics of sketching. *Creativity research journal*, 4(2), 123-143.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Bengio, Y. (2014). Generative adversarial nets. Paper presented at the *Advances in neural information processing systems.*

Goodman, N. (1978). *Ways of worldmaking (Vol. 51).* Hackett Publishing.

Goodman, N., Douglas, M., and Hull, D. L. (1992). How classification works: *Nelson Goodman among the social sciences.*

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).

Grason, J. (1971). An approach to computerized space planning using graph theory. Paper presented at the *Proceedings of the 8th Design automation workshop*.

Gross, M. D. (1994). Recognizing and interpreting diagrams in design. Paper presented at the *Proceedings of the workshop on Advanced visual interfaces*, Bari, Italy. https://doi.org/10.1145/192309.192330

Gross, M. D. (1995). Indexing visual databases of designs with diagrams. *Visual Databases in Architecture*, 1-14.

Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S., and Yang, G. Z. (2019). XAI—Explainable artificial intelligence. *Science robotics*, *4*(37), eaay7120.

Guo, Y., Liu, Y., Georgiou, T., and Lew, M. S. (2018). A review of semantic segmentation using deep neural networks. *International journal of multimedia information retrieval*, *7*, 87-93.

Harary F, Norman RZ (1953). *Graph theory as a mathematical model in social science*. Institute for Social Research, Ann Arbor

Haşlakoğlu O., (11 Mayıs,2017). *"Platon Düşüncesinde Mimesis Eleştirisi* [1. Oturum]" ,Zeynep Gökgöz mod. Sanat ve Felsefe İhtisas Seminerleri Söyleşisi.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. Paper presented at the *Proceedings of the IEEE conference on computer vision and pattern recognition.* pp. 770-778

Hebly, A. (1991). *The 5 points and Form. En Raumplan versus Plan Libre*, coord. Max Risselada. Delft: Delft University Press.

Heidegger, M. (2008). *Düşüncenin çağırdığı*. Say Yayınları.(pp.23,25)

Heller, W. R., Maling, K., and Sorkin, G. (1982). The planar package planner for system designers. Paper presented at the *19th Design Automation Conference*.

Herthogs, P., Debacker, W., Tunçer, B., De Weerdt, Y., and De Temmerman, N. (2019). Quantifying the generality and adaptability of building layouts using weighted graphs: the SAGA method. *Buildings, 9*(4), 92.

Heydarian, M., Doyle, T. E., and Samavi, R. (2022). MLCM: Multi-label confusion matrix. *IEEE Access, 10*, 19083-19095.

Hillier, B. (2007). *Space is the machine: a configurational theory of architecture*. Space Syntax.

Hillier B, Musgrove J, O'Sullivan P. (1972) Knowledge and design. Environmental design: research and practice 2. In: *Proceedings of EDRA 3/AR 8 conference*; p. 29

Hillier, B., Leaman, A., Stansall, P., and Bedford, M. (1976). Space syntax. *Environment and Planning B: Planning and design*, *3*(2), 147-185.

Homayouni, H. (2000). *A survey of computational approaches to space layout planning (1965-2000).* Department of Architecture and Urban Planning University of Washington.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11, 63-90.

Hong, T., Wang, Z., Luo, X., and Zhang, W. (2020). State-of-the-Art on Research and Applications of Machine Learning in the Building Life Cycle. *Energy and Buildings*, 109831.

Honda, K., and Mizoguchi, F. (1995, February). Constraint-based approach for automatic spatial layout planning. In *Proceedings the 11th Conference on Artificial Intelligence for Applications* (pp. 38-45). IEEE.

Hu, R., Huang, Z., Tang, Y., Van Kaick, O., Zhang, H., and Huang, H. (2020). Graph2plan: Learning floorplan generation from layout graphs. *ACM Transactions on Graphics (TOG)*, 39(4), 118: 111-118: 114.

Hmelo-Silver, C. E. (2004). Problem-based learning: What and how do students learn?. *Educational psychology review*, 16, 235-266.

Huang, W., and Zheng, H. (2018). Architectural drawings recognition and generation through machine learning. Paper presented at the Proceedings of the *38th Annual Conference of the Association for Computer Aided Design in Architecture*, Mexico City, Mexico.

Internet: *Gan Model*. URL1:https://neptune.ai/blog/generative-adversarial-networks-gan-applications. Accessed: March 13, 2024.

Internet:*Pix2pixmodelarchitecture*.URL2:https://www.tensorflow.org/tutorials/generative/pix2pix?hl=tr) Accessed: March 13, 2024.

Internet:*Yolov5architecture*.URL3:https://docs.ultralytics.com/tr/yolov5/tutorials/architecture_description/) Accessed: March 13, 2024.

Internet:*Yolov5algorithm*.URL4:https://docs.ultralytics.com/tr/yolov5/tutorials/architecture_description/)Accessed: March 13, 2024.

Internet:*Yolov8algorithm*.URL5:https://github.com/openmmlab/mmyolo/tree/main/configs/yolov8, 2023. Accessed: March 13, 2024.

Internet: *Confusion matrix explanation*. URL6: https://www.ml-science.com/confusion-matrix. Accessed: March 13, 2024.

Internet: *DCGAN*. URL7: https://www.tensorflow.org/tutorials/generative/dcgan?hl=tr Accessed: March 13, 2024.

Jo, J. H., and Gero, J. S. (1998). Space layout planning using an evolutionary approach. *Artificial intelligence in Engineering*, *12*(3), 149-162.

Kant, I. (1999). *Pratik Aklın Eleştirisi*, Türkiye Felsefe Kurumu, 3. Baskı, Ankara.

Karadag, I., Güzelci, O. Z., and Alaçam, S. (2022). EDU-AI: a twofold machine learning model to support classroom layout generation. *Construction Innovation*, (ahead-of-print).

Keshavarzi, M., and Rahmani-Asl, M. (2021). Genfloor: Interactive generative space layout system via encoded tree graphs. *arXiv preprint arXiv:2102.10320*.

Keleş, R. Y. (1966). Sosyal Konut Politikası Kavramı Üzerinde Bir Deneme ve Türkiye de Sosyal Konut Politikası. *Ankara Üniversitesi SBF Dergisi*, 21(02).

Kim, P. (2017). Matlab deep learning. With Machine Learning, *Neural Networks and Artificial Intelligence*, 130.

Koehler, D., Saleh, A., Sheghaf, L., Hua, Y., and Chuwei, Z. (2018). Mereologies-Combinatorial Design and the Description of Urban Form.

Korf, R. (1977). A shape independent theory of space allocation. *Environment and Planning B: Planning and Design*, *4*(1), 37-50.

Kotnik, T. (2010). Digital architectural design as exploration of computable functions. *International journal of architectural computing*, *8*(1), 1-16.

Kozminski, K., and Kinnen, E. (1984). An algorithm for finding a rectangular dual of a planar graph for use in area planning for VLSI integrated circuits. Paper presented at the *21st Design Automation Conference Proceedings.*

Kolarevic, B., and Klinger, K. *(Eds.). (2013). Manufacturing material effects: rethinking design and making in architecture*. Routledge.

Ko, J., Ennemoser, B., Yoo, W., Yan, W., and Clayton, M. J. (2023). Architectural spatial layout planning using artificial intelligence. *Automation in Construction*, *154*, 105019.

Kolb, D. A. (2007). *The Kolb learning style inventory*. Boston, MA: Hay Resources Direct.

Kolb, D. A. (2014). *Experiential learning: Experience as the source of learning and development*. FT press.

Lawson, B. (2006). *How Designerly Ways of Thinking: The design process demystified*. Routledge.

Leach, N. (2022). Architectural Hallucinations: What Can AI Tell Us About the Mind of an Architect?. *Architectural Design*, *92*(3), 66-71.

Leach, N. (2022). In the mirror of AI: what is creativity?. *Architectural Intelligence*, *1*(1), 15.

Lee, J., Boubekri, M., and Liang, F. (2019). Impact of Building Design Parameters on Daylighting Metrics Using an Analysis, Prediction, and Optimization Approach Based on Statistical Learning Technique. *Sustainability*, 11(5), 21. doi:10.3390/su11051474

Lee, S., Hwang, Y., Jin, Y., Ahn, S., and Park, J. (2019). Effects of Individuality, Education, and Image on Visual Attention: Analyzing Eye-tracking Data using Machine Learning. *Journal of Eye Movement Research*, 12(2), 22. doi:10.16910/jemr.12.2.4

Lefebvre, H. (2012). *From the production of space*. In Theatre and performance design (pp. 81-84). Routledge.

Li, G., Muller, M., Thabet, A., and Ghanem, B. (2019). Deepgcns: Can gcns go as deep as cnns?. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 9267-9276).

Li, J., Li, H., Umer, W., Wang, H., Xing, X., Zhao, S., and Hou, J. (2020). Identification and classification of construction equipment operators' mental fatigue using wearable eye-tracking technology. *Automation in Construction*, 109, 103000.

Li, X., Liu, Y., Lian, L., Yang, H., Dong, Z., Kang, D., ... and Keutzer, K. (2023). Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 17535-17545).

Liao, K., de Vries, B., Kong, J., and Zhang, K. (2015). Pattern, Cognition and Spatial Information Processing. Paper presented at the *International Conference on Computer-Aided Architectural Design Futures.*

Liggett, R. S. (2000). Automated facilities layout: past, present and future. *Automation in Construction*, *9*(2), 197-215.

Lin, J. Y., Wan, H. Y., and Cui, Y. T. (2020). Analyzing the spatial factors related to the distributions of building heights in urban areas: A comparative case study in Guangzhou and Shenzhen. *Sustainable Cities and Society*, 52, 9. doi:10.1016/j.scs.2019.101854

Liu, C., Wu, J., Kohli, P., and Furukawa, Y. (2017). Raster-to-vector: Revisiting floorplan transformation. Paper presented at the *Proceedings of the IEEE International Conference on Computer Vision.*

Liu, Y., Lai, Y., Chen, J., Liang, L., and Deng, Q. (2020). SCUT-AutoALP: A Diverse Benchmark Dataset for Automatic Architectural Layout Parsing. *IEICE Transactions on Information and Systems*, 103(12), 2725-2729.

Llach, D. C., Bidgoli, A., and Darbari, S. (2017). Assisted automation: Three learning experiences in architectural robotics. *International Journal of Architectural Computing*, 15(1), 87-102. doi:10.1177/1478077117691627

Llamas, J., M Lerones, P., Medina, R., Zalama, E., and Gómez-García-Bermejo, J. (2017). Classification of architectural heritage images using deep learning techniques. *Applied Sciences*, 7(10), 992.

Lozano-Perez, T. (1990). *Spatial planning: A configuration space approach* (pp. 259-271). Springer New York.

Lynes, J. (1977). Windows and floor plans. *Environment and Planning B: Planning and Design, 4*(1), 51-55.

MacLean, A., Young, R. M., Bellotti, V. M., and Moran, T. P. (2020). Questions, options, and criteria: Elements of design space analysis. In *Design rationale* (pp. 53-105). CRC Press.

March, L. (2002). Architecture and Mathematics since 1960. *Nexus Network Journal, 4*(3).

March, L. and Steadman, P. (2020). *The geometry of environment: an introduction to spatial organization in design*: Routledge.

Markus, T. A. (1969b). The role of building performance measurement and appraisal in design method. Design methods in Architecture. London, Lund Humphries

Maver, T. W. (1970). *Appraisal in the building design process. Emerging Methods in Environmental Design and Planning*. Cambridge Mass, MIT Press.

McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, C. E. (2006). A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4), 12-12.

Miraftabzadeh, S. M., Longo, M., Foiadelli, F., Pasetti, M., and Igual, R. (2021). Advances in the application of machine learning techniques for power system analytics: A survey. *Energies, 14*(16), 4776.

Mitchell, T. M. (1997). Artificial neural networks. *Machine learning*, 45, 81-127.

Nakatake, S., Fujiyoshi, K., Murata, H., and Kajitani, Y. (1996). Module placement on BSG-structure and IC layout applications. Paper presented at the *Proceedings of International Conference on Computer Aided Design.*

Narahara, T. (2017). *Collective Construction Modeling and Machine Learning: Potential for Architectural Design*. Brussels: Ecaade-Education and Research Computer Aided Architectural Design Europe.

Nauata, N., Chang, K.-H., Cheng, C.-Y., Mori, G., and Furukawa, Y. (2020). House-gan: Relational generative adversarial networks for graph-constrained house layout generation. Paper presented at the *European Conference on Computer Vision.*

Negroponte, N. (1969). Toward a theory of architecture machines. *Journal of Architectural Education*, 23(2), 9-12.

Newton, D. (2019). Generative Deep Learning in Architectural Design. *Technology|Architecture + Design*, 3(2), 176-189. doi:10.1080/24751448.2019.1640536.

Norberg-Schulz, C. (2019). *Genius loci: towards a phenomenology of architecture* (1979). Historic Cities: Issues in Urban Conservation, 8, 31.

Nisztuk, M., and Myszkowski, P. B. (2019). Hybrid Evolutionary Algorithm applied to Automated Floor Plan Generation. *International Journal of Architectural Computing*, 17(3), 260-283. doi:10.1177/1478077119832982

Obeso, A. M., Vázquez, M. S. G., Acosta, A. A. R., and Benois-Pineau, J. (2017). Connoisseur: classification of styles of Mexican architectural heritage with deep learning and visual attention prediction. Paper presented at the Proceedings of the *15th international workshop on content-based multimedia indexing.*

Oungrinis, K.-A., and Liapi, M. (2014). Spatial Elements Imbued with Cognition: A possible step toward the "Architecture Machine". *International Journal of Architectural Computing*, 12(4), 419-438. doi:10.1260/1478-0771.12.4.419

Ozerol, G., and Arslan Selçuk, S. (2022). Machine learning in the discipline of architecture: A review on the research trends between 2014 and 2020. *International Journal of Architectural Computing*, 14780771221100102.

Oxman, R. (2006). Theory and design in the first digital age. *Design Studies*, 27(3), 229e265.

Patil, A. G., Li, M., Fisher, M., Savva, M., and Zhang, H. (2021). LayoutGMN: Neural Graph Matching for Structural Layout Similarity. Paper presented at the *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Penman, S. S. D. (2018). *Ludus ex machina: toward computational play* (Doctoral dissertation, Massachusetts Institute of Technology).

Piaget, J. (1972). Development and learning. *Reading in child behavior and development*, 38-46.

Prathap, G., and Afanasyev, I. (2018). Deep learning approach for building detection in satellite multispectral imagery. Paper presented at the *2018 International Conference on Intelligent Systems (IS)*.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Radziszewski, K., and Iop. (2017). Artificial Neural Networks as an Architectural Design Tool-Generating New Detail Forms Based On the Roman Corinthian Order Capital. *In World Multidisciplinary Civil Engineering-Architecture-Urban Planning Symposium - Wmcaus* (Vol. 245). Bristol: Iop Publishing Ltd.

Rahimian, F. P., Seyedzadeh, S., Oliver, S., Rodriguez, S., and Dawood, N. (2020). On-demand monitoring of construction projects through a game-like hybrid application of BIM and machine learning. *Automation in Construction*, 110, 14. doi:10.1016/j.autcon.2019.103012

Raman, R., and D'Souza, M. (2019). Decision learning framework for architecture design decisions of complex systems and system-of-systems. *Systems Engineering*, 22(6), 538-560. doi:10.1002/sys.21517

Raporu, Ö. İ. K. (2009). Yerel Yönetimler. *Katılımcılık ve Kentsel Yönetim Komisyon Raporu, Kentleşme Şurası, Bayındırlık ve İskan Bakanlığı*, Ankara.

Rebaudengo, M., and Reorda, M. S. (1996). GALLO: A genetic algorithm for floorplan area optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 15*(8), 943-951.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, *28*.

Rodrigues, R. C., and Duarte, R. B. (2022). Generating floor plans with deep learning: A cross-validation assessment over different dataset sizes. *International Journal of Architectural Computing*, *20*(3), 630-644.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 10684-10695).

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. Paper presented at the International Conference on Medical image computing and computer-assisted intervention.

Rothman, D. (2020). *Hands-On Explainable AI (XAI) with Python: Interpret, visualize, explain, and integrate reliable AI for fair, secure, and trustworthy AI apps*. Packt Publishing Ltd.

Rittel, H. W. (1988). *The reasoning of designers* (p. 12). Boston, MA, USA: IGP.

Saleh, B., Abe, K., Arora, R. S., and Elgammal, A. (2016). Toward automated discovery of artistic influence. *Multimedia Tools and Applications*, 75(7), 3565-3591.

Schneider, T., and Till, J. (2007). *Flexible housing*. Architectural press.

Schön, D. A. (1987). Educating the reflective practitioner, toward a new design for teaching and learning in the professions. San Francisco, CA: Jossey-Bass Inc.

Schön, D. A. (1987a). Educating the reflective practicioner. Washington, DC.: American Educa-tional Research Association.

Schön, D. A. (1983). The reflective practitioner: How professionals think in action. New York: Basic Books. (Reprinted in 1995).

Shekhawat, K. (2018). Enumerating generic rectangular floor plans. *Automation in Construction, 92*, 151-165.

Shekhawat, K., Upasani, N., Bisht, S., and Jain, R. (2020). GPLAN: Computer-Generated Dimensioned Floorplans for given Adjacencies. arXiv preprint arXiv:2008.01803.

Singh, A. (2020). *Hands-On Python Deep Learning for the Web*: Packt Publishing.

Singh, V., and Gu, N. (2012). Towards an integrated generative design framework. *Design studies*, 33(2), 185-207.

Sjoberg, C. H. (2017). *Machine Learning Models for Directed Curation of Design Solution Space* (Doctoral dissertation, The University of North Carolina at Charlotte).

Ślusarczyk, G. (2018). Graph-based representation of design properties in creating building floorplans. *Computer-Aided Design, 95*, 24-39.

Sonmez, N. O. (2018). A review of the use of examples for automating architectural design tasks. *Computer-Aided Design*, 96, 13-30. doi:10.1016/j.cad.2017.10.005

Steinfeld, K., Park, K., Menges, A., and Walker, S. (2019). A Framework for the Application of Machine Learning to Generative Architectural Design, and a Report of Activities at Smartgeometry 2018. In J. H. Lee (Ed.), *Computer-Aided Architectural Design: Hello, Culture, Caad Futures 2019* (Vol. 1028, pp. 32-46). Cham: Springer International Publishing Ag.

Stiny, G. (1980). Introduction to shape and shape grammars. *Environment and Planning B: Planning and Design, 7*(3), 343-351.

Stiny, G. (1990). What is a design? *Environment and Planning B: Planning and Design, 17*(1), 97-103.

Stiny, G. (1991). The algebras of design. *Research in Engineering Design, 2*(3), 171-181.

Stiny, G. N. (1975). *Pictorial and formal aspects of shape and shape grammars and aesthetic systems*: University of California, Los Angeles.

Takizawa, A., and Furuta, A. (2017). 3D spatial analysis method with first-person viewpoint by deep convolutional neural network with omnidirectional RGB and depth images. In *Proceedings of the 35th International Conference on Education and Research in Computer Aided Architectural Design in Europe (eCAADe)[Volume 2]*. eCAADe.

Tamke, M., Nicholas, P., and Zwierzycki, M. (2018). Machine learning for architectural design: Practices and infrastructure. *International Journal of Architectural Computing*, 16(2), 123-143. doi:10.1177/1478077118778580

Tarabishy, S., Psarras, S., Kosicki, M., and Tsigkari, M. (2020). Deep learning surrogate models for spatial and visual connectivity. *International Journal of Architectural Computing*, 18(1), 53-66. doi:10.1177/1478077119894483

Tekler, Z. D., Low, R., Gunay, B., Andersen, R. K., and Blessing, L. (2020). A scalable Bluetooth Low Energy approach to identify occupancy patterns and profiles in office spaces. *Building and Environment*, 171, 12. doi:10.1016/j.buildenv.2020.106681

Tekeli, İ. (1979). Türkiye kentlerinde apartmanlaşma sürecinde iki aşama. *Çevre Mimarlık ve Görsel Sanatlar Dergisi*, 4.

Terven, J., Córdova-Esparza, D. M., and Romero-González, J. A. (2023). A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *Machine Learning and Knowledge Extraction*, *5*(4), 1680-1716.

Till, J., and Schneider, T. (2005). Flexible housing: the means to the end. *ARQ: architectural research quarterly*, *9*(3-4), 287-296.

Valero, E., Forster, A., Bosche, F., Hyslop, E., Wilson, L., and Turmel, A. (2019). Automated defect detection and classification in ashlar masonry walls using machine learning. *Automation in Construction*, 106, 14. doi:10.1016/j.autcon.2019.102846

Vazquez, A. N., and Jabi, W. (2017). Investigations in robotic-assisted design: Strategies for symbiotic agencies in material-directed generative design processes. *International Journal of Architectural Computing*, 15(1), 70-86. doi:10.1177/1478077117691626

Vazquez, A. N., and Jabi, W. (2019). Robotic assisted design workflows: a study of key human factors influencing team fluency in human-robot collaborative design processes. *Architectural Science Review*, 62(5), 409-423. doi:10.1080/00038628.2019.1660611

Walker, S., Khan, W., Katic, K., Maassen, W., and Zeiler, W. (2020). Accuracy of different machine learning algorithms and added-value of predicting aggregated-level energy performance of commercial buildings. *Energy and Buildings*, 209, 14. doi:10.1016/j.enbuild.2019.109705

Wilkinson, S., and Hanna, S. (2014). Approximating computational fluid dynamics for generative tall building design. *International Journal of Architectural Computing*, 12(2), 155-177.

Wu, W., Fu, X.-M., Tang, R., Wang, Y., Qi, Y.-H., and Liu, L. (2019). Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics* (TOG), 38(6), 1-12.

Woodbury, R. F., and Burrow, A. L. (2006). Whither design space?. Ai Edam, 20(2), 63-82.

Woodbury, R., Datta, S., and Burrow, A. (2000). Erasure in design space exploration. *Artificial Intelligence in Design'00*, 521-543.

Yang, L., Lyu, K. L., Li, H. L., and Liu, Y. (2020). Building climate zoning in China using supervised classification-based machine learning. *Building and Environment*, 171, 11. doi:10.1016/j.buildenv.2020.106663

Yang, G., Feng, W., Jin, J., Lei, Q., Li, X., Gui, G., and Wang, W. (2020, December). Face mask recognition system with YOLOV5 based on image recognition. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)* (pp. 1398-1404). IEEE.

Yar, H., Khan, Z. A., Ullah, F. U. M., Ullah, W., and Baik, S. W. (2023). A modified YOLOv5 architecture for efficient fire detection in smart cities. *Expert Systems with Applications*, *231*, 120465.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018, July). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 974-983).

Yoon, S. W., Kim, I., Lee, K. C., and Ieee. (2018). The Architectural Design of Storage System for Power Data Management. In *2018 Ieee International Conference on Big Data and Smart Computing* (pp. 736-738). New York: Ieee.

Yoshimura, Y., Cai, B., Wang, Z., and Ratti, C. (2019). Deep Learning Architect: Classification for Architectural Design Through the Eye of Artificial Intelligence. Paper presented at the *International Conference on Computers in Urban Planning and Urban Management.*

Zandavali, B. A., and Jimenez Garcia, M. (2019). Automated brick pattern generator for robotic assembly using machine learning and images. *In Blucher Design Proceedings* (Vol. 7, pp. 217-226). Editora Blucher.

Zeng, A., Ho, H., and Yu, Y. (2020). Prediction of building electricity usage using Gaussian Process Regression. *Journal of Building Engineering*, 28, 101054.

Zeng, Z., Li, X., Yu, Y. K., and Fu, C. W. (2019). Deep floor plan recognition using a multi-task network with room-boundary-guided attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 9096-9104).

Zhang, F., Zhou, B., Liu, L., Liu, Y., Fung, H. H., Lin, H., and Ratti, C. (2018). Measuring human perceptions of a large-scale urban region using machine learning. *Landscape and Urban Planning*, 180, 148-160.

Zhang, H. (2019). 3D Model Generation on Architectural Plan and Section Training through Machine Learning. *Technologies*, 7(4), 14. doi:10.3390/technologies7040082

Zhang, Y., Fei, G. Z., and Shang, W. Q. (2017). *3D Architecture Facade Optimization Based on Genetic Algorithm and Neural Network.* New York: Ieee.

Zheng, Z., Li, J., Zhu, L., Li, H., Petzold, F., and Tan, P. (2022). GAT-CADNet: Graph Attention Network for Panoptic Symbol Spotting in CAD Drawings. Paper presented at the *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

Zhao, Z. Q., Zheng, P., Xu, S. T., and Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, *30*(11), 3212-3232.

Zou, Z., Chen, K., Shi, Z., Guo, Y., and Ye, J. (2023). Object detection in 20 years: A survey. *Proceedings of the IEEE*.

APPENDICES

APPENDIX 1. Google Colab_ DCGAN_Python_Code

A DCGAN is a direct extension of the GAN described above, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively. It was first described by Radford et. al. in the paper `Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks <https://arxiv.org/pdf/1511.06434.pdf>`. **The discriminator is made up of strided `convolution <https://pytorch.org/docs/stable/nn.html#torch.nn.Conv2d>` layers,** `batch norm <https://pytorch.org/docs/stable/nn.html#torch.nn.BatchNorm2d>` layers, and `LeakyReLU <https://pytorch.org/docs/stable/nn.html#torch.nn.LeakyReLU>` activations. The input is a 3x64x64 input image and the output is a scalar probability that the input is from the real data distribution. The generator is comprised of `convolutional-transpose <https://pytorch.org/docs/stable/nn.html#torch.nn.ConvTranspose2d>` layers, batch norm layers, and `ReLU <https://pytorch.org/docs/stable/nn.html#relu>` activations. The input is a latent vector, z, that is drawn from a standard normal distribution and the output is a 3x64x64 RGB image. The strided conv-transpose layers allow the latent vector to be transformed into a volume with the same shape as an image. In the paper, the authors also give some tips about how to setup the optimizers, how to calculate the loss functions, and how to initialize the model weights, all of which will be explained in the coming sections.

```
from __future__ import print_function
#%matplotlib inline
import argparse
import os
import random
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torchvision.utils as vutils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML

# Set random seem for reproducibility
manualSeed = 999
#manualSeed = random.randint(1, 10000) # use if you want new results
print("Random Seed: ", manualSeed)
random.seed(manualSeed)
torch.manual_seed(manualSeed)
```

```
Random Seed:  999
<torch._C.Generator at 0x7d8ac8121270>
```

Inputs

Let's define some inputs for the run:

- **dataroot -** the path to the root of the dataset folder. We will talk more about the dataset in the next section
- **workers -** the number of worker threads for loading the data with the DataLoader
- **batch_size -** the batch size used in training. The DCGAN paper uses a batch size of 128
- **image_size -** the spatial size of the images used for training. This implementation defaults to 64x64. If another size is desired, the structures of D and G must be changed. See `here <https://github.com/pytorch/examples/issues/70>` for more details
- **nc -** number of color channels in the input images. For color images this is 3
- **nz** - length of latent vector
- **ngf -** relates to the depth of feature maps carried through the generator
- **ndf -** sets the depth of feature maps propagated through the discriminator
- **num_epochs -** number of training epochs to run. Training for longer will probably lead to better results but will also take much longer
- **lr** - learning rate for training. As described in the DCGAN paper, this number should be 0.0002
- **beta1 -** beta1 hyperparameter for Adam optimizers. As described in paper, this number should be 0.5
- **ngpu -** number of GPUs available. If this is 0, code will run in CPU mode. If this number is greater than 0 it will run on that number of GPUs

APPENDIX 1(continued). Google Colab_ DCGAN_Python_Code


```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Root directory for dataset
dataroot = "/content/drive/MyDrive/DCGAN/OPEN_PLANS"

# Number of workers for dataloader
workers = 4

# Batch size during training
batch_size = 128

# Spatial size of training images. All images will be resized to this
#   size using a transformer.
image_size = 64

# Number of channels in the training images. For color images this is 3
nc = 3

# Size of z latent vector (i.e. size of generator input)
nz = 10


# Size of feature maps in generator
ngf = 64

# Size of feature maps in discriminator
ndf = 64

# Number of training epochs
num_epochs = 1000

# Learning rate for optimizers
lr = 0.0005

# Beta1 hyperparam for Adam optimizers
beta1 = 0.5

# Number of GPUs available. Use 0 for CPU mode.
ngpu = 1
```

```
dataroot
```

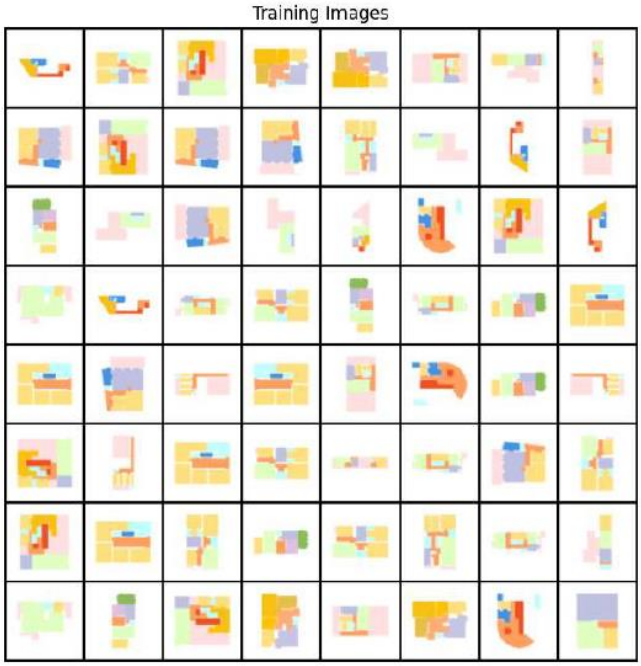
```
'/content/drive/MyDrive/DCGAN/OPEN_PLANS'
```

Düzenlemek için çift tıklayın (veya enter tuşuna basın)

```
# We can use an image folder dataset the way we have it setup.
# Create the dataset
dataset = dset.ImageFolder(root=dataroot,
                           transform=transforms.Compose([
                               transforms.Resize(image_size),
                               transforms.CenterCrop(image_size),
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                           ]))
# Create the dataloader
dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
                                         shuffle=True, num_workers=workers)

# Decide which device we want to run on
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cpu")

# Plot some training images
real_batch = next(iter(dataloader))
plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch[0].to(device)[:64], padding=2, normalize=True).cpu(),(1,2,0)))
```

APPENDIX 1(continued). Google Colab_ DCGAN_Python_Code


```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker proces
  warnings.warn(_create_warning_msg(
<matplotlib.image.AxesImage at 0x7d89e37bf820>
```



Implementation

With our input parameters set and the dataset prepared, we can now get into the implementation. We will start with the weigth initialization strategy, then talk about the generator, discriminator, loss functions, and training loop in detail.

Weight Initialization

From the DCGAN paper, the authors specify that all model weights shall be randomly initialized from a Normal distribution with mean=0, stdev=0.2. The ``weights_init`` function takes an initialized model as input and reinitializes all convolutional, convolutional-transpose, and batch normalization layers to meet this criteria. This function is applied to the models immediately after initialization.

```
# custom weights initialization called on netG and netD
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)
```

Generator

The generator, G, is designed to map the latent space vector (z) to data-space. Since our data are images, converting z to data-space means ultimately creating a RGB image with the same size as the training images (i.e. 3x64x64). In practice, this is accomplished through a series of strided two dimensional convolutional transpose layers, each paired with a 2d batch norm layer and a relu

APPENDIX 1(continued). Google Colab_ DCGAN_Python_Code

activation. The output of the generator is fed through a tanh function to return it to the input data range of $[-1,1]$. It is worth noting the existence of the batch norm functions after the conv-transpose layers, as this is a critical contribution of the DCGAN paper. These layers help with the flow of gradients during training. An image of the generator from the DCGAN paper is shown below.

.. figure:: /_static/img/dcgan_generator.png
   :alt: dcgan_generator

Notice, the how the inputs we set in the input section (*nz*, *ngf*, and *nc*) influence the generator architecture in code. *nz* is the length of the z input vector, *ngf* relates to the size of the feature maps that are propagated through the generator, and *nc* is the number of channels in the output image (set to 3 for RGB images). Below is the code for the generator.

```
# Generator Code

class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 4 x 4
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*4) x 8 x 8
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*2) x 16 x 16
            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. (ngf) x 32 x 32
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. (nc) x 64 x 64
        )

    def forward(self, input):
        return self.main(input)
```

Now, we can instantiate the generator and apply the `weights_init` function. Check out the printed model to see how the generator object is structured.

```
# Create the generator
netG = Generator(ngpu).to(device)

# Handle multi-gpu if desired
if (device.type == 'cuda') and (ngpu > 1):
    netG = nn.DataParallel(netG, list(range(ngpu)))

# Apply the weights_init function to randomly initialize all weights
#  to mean=0, stdev=0.2.
netG.apply(weights_init)

# Print the model
print(netG)
```

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(10, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

APPENDIX 1(continued). Google Colab_ DCGAN_Python_Code


```
        (8): ReLU(inplace=True)
        (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (11): ReLU(inplace=True)
        (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (13): Tanh()
      )
    )
```

Discriminator

As mentioned, the discriminator, D, is a binary classification network that takes an image as input and outputs a scalar probability that the input image is real (as opposed to fake). Here, D takes a 3x64x64 input image, processes it through a series of Conv2d, BatchNorm2d, and LeakyReLU layers, and outputs the final probability through a Sigmoid activation function. This architecture can be extended with more layers if necessary for the problem, but there is significance to the use of the strided convolution, BatchNorm, and LeakyReLUs. The DCGAN paper mentions it is a good practice to use strided convolution rather than pooling to downsample because it lets the network learn its own pooling function. Also batch norm and leaky relu functions promote healthy gradient flow which is critical for the learning process of both G and D.

Discriminator Code

```
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 64 x 64
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*4) x 8 x 8
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 4 x 4
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

Now, as with the generator, we can create the discriminator, apply the `weights_init` function, and print the model's structure.

```
# Create the Discriminator
netD = Discriminator(ngpu).to(device)

# Handle multi-gpu if desired
if (device.type == 'cuda') and (ngpu > 1):
    netD = nn.DataParallel(netD, list(range(ngpu)))

# Apply the weights_init function to randomly initialize all weights
#  to mean=0, stdev=0.2.
netD.apply(weights_init)

# Print the model
print(netD)
```

```
Discriminator(
  (main): Sequential(
```

APPENDIX 1(continued). Google Colab_ DCGAN_Python_Code


```
        (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (1): LeakyReLU(negative_slope=0.2, inplace=True)
        (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (4): LeakyReLU(negative_slope=0.2, inplace=True)
        (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (7): LeakyReLU(negative_slope=0.2, inplace=True)
        (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
        (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (10): LeakyReLU(negative_slope=0.2, inplace=True)
        (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
        (12): Sigmoid()
      )
    )
```

Loss Functions and Optimizers

With D and G setup, we can specify how they learn through the loss functions and optimizers. We will use the Binary Cross Entropy loss (`BCELoss <https://pytorch.org/docs/stable/nn.html#torch.nn.BCELoss>`__) function which is defined in PyTorch as:

$$\begin{align}\ell(x, y) = L = \{l_1,\dots,l_N\}^\top, \quad l_n = - \left[y_n \cdot \log x_n + (1 - y_n) \cdot \log (1 - x_n) \right]\end{align}$$

Notice how this function provides the calculation of both log components in the objective function (i.e. $log(D(x))$ and $log(1-D(G(z)))$). We can specify what part of the BCE equation to use with the y input. This is accomplished in the training loop which is coming up soon, but it is important to understand how we can choose which component we wish to calculate just by changing y (i.e. GT labels).

Next, we define our real label as 1 and the fake label as 0. These labels will be used when calculating the losses of D and G, and this is also the convention used in the original GAN paper. Finally, we set up two separate optimizers, one for D and one for G. As specified in the DCGAN paper, both are Adam optimizers with learning rate 0.0002 and Beta1 = 0.5. For keeping track of the generator's learning progression, we will generate a fixed batch of latent vectors that are drawn from a Gaussian distribution (i.e. fixed_noise) . In the training loop, we will periodically input this fixed_noise into G, and over the iterations we will see images form out of the noise.

```
# Initialize BCELoss function
criterion = nn.BCELoss()

# Create batch of latent vectors that we will use to visualize
#  the progression of the generator
fixed_noise = torch.randn(64, nz, 1, 1, device=device)

# Establish convention for real and fake labels during training
real_label = 1
fake_label = 0

# Setup Adam optimizers for both G and D
optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.999))
```

Training

Finally, now that we have all of the parts of the GAN framework defined, we can train it. Be mindful that training GANs is somewhat of an art form, as incorrect hyperparameter settings lead to mode collapse with little explanation of what went wrong. Here, we will closely follow Algorithm 1 from Goodfellow's paper, while abiding by some of the best

APPENDIX 1(continued). Google Colab_ DCGAN_Python_Code


```
# Training Loop

# Lists to keep track of progress
img_list = []
G_losses = []
D_losses = []
iters = 0

print("Starting Training Loop...")
# For each epoch
for epoch in range(num_epochs):
    # For each batch in the dataloader
    for i, data in enumerate(dataloader, 0):

        ############################
        # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
        ###########################
        ## Train with all-real batch
        netD.zero_grad()
        # Format batch
        real_cpu = data[0].to(device)
        b_size = real_cpu.size(0)
        label = torch.full((b_size,), real_label, device=device)
        # Forward pass real batch through D
        output = netD(real_cpu).view(-1)
        # Calculate loss on all-real batch
        # Daha sonra label'ı Float türüne dönüştürün
        label = label.float()

        errD_real = criterion(output, label)
        # Calculate gradients for D in backward pass
        errD_real.backward()
        D_x = output.mean().item()

        ## Train with all-fake batch
        # Generate batch of latent vectors
        noise = torch.randn(b_size, nz, 1, 1, device=device)
        # Generate fake image batch with G
        fake = netG(noise)
        label.fill_(fake_label)
        # Classify all fake batch with D
        output = netD(fake.detach()).view(-1)
        # Calculate D's loss on the all-fake batch
        errD_fake = criterion(output, label)
        # Calculate the gradients for this batch
        errD_fake.backward()
        D_G_z1 = output.mean().item()
        # Add the gradients from the all-real and all-fake batches
        errD = errD_real + errD_fake
        # Update D
        optimizerD.step()

        ############################
        # (2) Update G network: maximize log(D(G(z)))
        ###########################
        netG.zero_grad()
        label.fill_(real_label)  # fake labels are real for generator cost
        # Since we just updated D, perform another forward pass of all-fake batch through D
        output = netD(fake).view(-1)
        # Calculate G's loss based on this output
        errG = criterion(output, label)
        # Calculate gradients for G
        errG.backward()
        D_G_z2 = output.mean().item()
        # Update G
        optimizerG.step()

        # Output training stats
        if i % 50 == 0:
            print('[%d/%d][%d/%d]\tLoss_D: %.4f\tLoss_G: %.4f\tD(x): %.4f\tD(G(z)): %.4f / %.4f'
                  % (epoch, num_epochs, i, len(dataloader),
                     errD.item(), errG.item(), D_x, D_G_z1, D_G_z2))

        # Save Losses for plotting later
        G_losses.append(errG.item())
        D_losses.append(errD.item())

        # Check how the generator is doing by saving G's output on fixed_noise
        if (iters % 500 == 0) or ((epoch == num_epochs-1) and (i == len(dataloader)-1)):
            with torch.no_grad():
                fake = netG(fixed_noise).detach().cpu()
            img_list.append(vutils.make_grid(fake, padding=2, normalize=True))
```

APPENDIX 1(continued). Google Colab_ DCGAN_Python_Code


```
    iters += 1

Starting Training Loop...
[0/1000][0/2]   Loss_D: 2.7033  Loss_G: 14.7140 D(x): 0.5791    D(G(z)): 0.8108 / 0.0003
[1/1000][0/2]   Loss_D: 0.2303  Loss_G: 10.4573 D(x): 0.9320    D(G(z)): 0.1003 / 0.0000
[2/1000][0/2]   Loss_D: 0.0731  Loss_G: 15.2119 D(x): 0.9464    D(G(z)): 0.0034 / 0.0000
[3/1000][0/2]   Loss_D: 3.6065  Loss_G: 23.7410 D(x): 0.9988    D(G(z)): 0.9259 / 0.0000
[4/1000][0/2]   Loss_D: 0.0002  Loss_G: 19.6564 D(x): 0.9999    D(G(z)): 0.0001 / 0.0000
[5/1000][0/2]   Loss_D: 0.6691  Loss_G: 15.1837 D(x): 1.0000    D(G(z)): 0.3620 / 0.0000
[6/1000][0/2]   Loss_D: 0.3898  Loss_G: 22.3344 D(x): 0.9977    D(G(z)): 0.2454 / 0.0000
[7/1000][0/2]   Loss_D: 11.1266 Loss_G: 25.2111 D(x): 1.0000    D(G(z)): 0.9889 / 0.0000
[8/1000][0/2]   Loss_D: 1.2902  Loss_G: 15.9578 D(x): 0.4497    D(G(z)): 0.0000 / 0.0002
[9/1000][0/2]   Loss_D: 3.6001  Loss_G: 15.6018 D(x): 1.0000    D(G(z)): 0.8077 / 0.0000
[10/1000][0/2]  Loss_D: 2.1320  Loss_G: 6.2179  D(x): 0.2805    D(G(z)): 0.0001 / 0.0160
[11/1000][0/2]  Loss_D: 0.6096  Loss_G: 11.4821 D(x): 0.7548    D(G(z)): 0.0017 / 0.0000
[12/1000][0/2]  Loss_D: 0.0816  Loss_G: 7.6109  D(x): 0.9930    D(G(z)): 0.0662 / 0.0033
[13/1000][0/2]  Loss_D: 2.2765  Loss_G: 0.8127  D(x): 0.2276    D(G(z)): 0.0045 / 0.7002
[14/1000][0/2]  Loss_D: 1.7240  Loss_G: 18.0347 D(x): 0.9896    D(G(z)): 0.4172 / 0.0000
[15/1000][0/2]  Loss_D: 0.5963  Loss_G: 4.2763  D(x): 0.9984    D(G(z)): 0.4222 / 0.0213
[16/1000][0/2]  Loss_D: 4.2019  Loss_G: 7.8118  D(x): 0.0664    D(G(z)): 0.0001 / 0.0033
[17/1000][0/2]  Loss_D: 1.9683  Loss_G: 15.9012 D(x): 0.8927    D(G(z)): 0.7368 / 0.0000
[18/1000][0/2]  Loss_D: 0.6626  Loss_G: 4.3505  D(x): 0.8824    D(G(z)): 0.3781 / 0.0311
[19/1000][0/2]  Loss_D: 2.3616  Loss_G: 5.1054  D(x): 0.2139    D(G(z)): 0.0026 / 0.0275
[20/1000][0/2]  Loss_D: 0.9978  Loss_G: 9.3168  D(x): 0.9271    D(G(z)): 0.5379 / 0.0004
[21/1000][0/2]  Loss_D: 1.3014  Loss_G: 6.2587  D(x): 0.9411    D(G(z)): 0.6560 / 0.0062
[22/1000][0/2]  Loss_D: 0.4766  Loss_G: 3.4888  D(x): 0.8139    D(G(z)): 0.1960 / 0.0477
[23/1000][0/2]  Loss_D: 1.7344  Loss_G: 2.7596  D(x): 0.2907    D(G(z)): 0.0239 / 0.1344
[24/1000][0/2]  Loss_D: 1.2053  Loss_G: 4.9391  D(x): 0.4118    D(G(z)): 0.0144 / 0.0254
[25/1000][0/2]  Loss_D: 0.8565  Loss_G: 6.6648  D(x): 0.7492    D(G(z)): 0.3213 / 0.0057
[26/1000][0/2]  Loss_D: 1.4027  Loss_G: 13.3551 D(x): 0.8720    D(G(z)): 0.6550 / 0.0000
[27/1000][0/2]  Loss_D: 1.3129  Loss_G: 6.6545  D(x): 0.9359    D(G(z)): 0.6606 / 0.0082
[28/1000][0/2]  Loss_D: 2.3355  Loss_G: 9.4278  D(x): 0.9427    D(G(z)): 0.8128 / 0.0002
[29/1000][0/2]  Loss_D: 1.8605  Loss_G: 5.5925  D(x): 0.8937    D(G(z)): 0.6794 / 0.0127
[30/1000][0/2]  Loss_D: 0.8771  Loss_G: 3.6825  D(x): 0.6392    D(G(z)): 0.2796 / 0.0465
[31/1000][0/2]  Loss_D: 0.4898  Loss_G: 4.0806  D(x): 0.7507    D(G(z)): 0.1502 / 0.0264
[32/1000][0/2]  Loss_D: 1.3095  Loss_G: 1.0319  D(x): 0.3854    D(G(z)): 0.1086 / 0.4252
[33/1000][0/2]  Loss_D: 2.8310  Loss_G: 2.9533  D(x): 0.1105    D(G(z)): 0.0061 / 0.1223
[34/1000][0/2]  Loss_D: 0.7350  Loss_G: 3.6066  D(x): 0.6200    D(G(z)): 0.1046 / 0.0539
[35/1000][0/2]  Loss_D: 1.1169  Loss_G: 2.7543  D(x): 0.4274    D(G(z)): 0.0427 / 0.1007
[36/1000][0/2]  Loss_D: 3.6133  Loss_G: 5.8285  D(x): 0.0569    D(G(z)): 0.0007 / 0.0080
[37/1000][0/2]  Loss_D: 2.0041  Loss_G: 5.9074  D(x): 0.9451    D(G(z)): 0.8077 / 0.0113
[38/1000][0/2]  Loss_D: 1.8091  Loss_G: 6.3533  D(x): 0.9364    D(G(z)): 0.7407 / 0.0042
[39/1000][0/2]  Loss_D: 0.7440  Loss_G: 2.9827  D(x): 0.7429    D(G(z)): 0.3116 / 0.0778
[40/1000][0/2]  Loss_D: 0.5830  Loss_G: 3.7552  D(x): 0.7333    D(G(z)): 0.1781 / 0.0384
[41/1000][0/2]  Loss_D: 0.6485  Loss_G: 3.8431  D(x): 0.7083    D(G(z)): 0.2175 / 0.0266
[42/1000][0/2]  Loss_D: 0.9647  Loss_G: 2.8944  D(x): 0.5672    D(G(z)): 0.2279 / 0.0838
[43/1000][0/2]  Loss_D: 1.6287  Loss_G: 2.1311  D(x): 0.2596    D(G(z)): 0.0192 / 0.1861
[44/1000][0/2]  Loss_D: 1.2589  Loss_G: 2.4027  D(x): 0.4025    D(G(z)): 0.0928 / 0.1268
[45/1000][0/2]  Loss_D: 1.2759  Loss_G: 2.7783  D(x): 0.3674    D(G(z)): 0.0240 / 0.1028
[46/1000][0/2]  Loss_D: 0.4660  Loss_G: 4.2505  D(x): 0.8044    D(G(z)): 0.1913 / 0.0224
[47/1000][0/2]  Loss_D: 0.6834  Loss_G: 5.6042  D(x): 0.8840    D(G(z)): 0.3873 / 0.0068
[48/1000][0/2]  Loss_D: 0.8604  Loss_G: 4.9275  D(x): 0.9242    D(G(z)): 0.5086 / 0.0139
[49/1000][0/2]  Loss_D: 0.7806  Loss_G: 5.1854  D(x): 0.8469    D(G(z)): 0.4267 / 0.0102
[50/1000][0/2]  Loss_D: 1.3152  Loss_G: 6.6472  D(x): 0.9334    D(G(z)): 0.6723 / 0.0043
[51/1000][0/2]  Loss_D: 1.1144  Loss_G: 5.7903  D(x): 0.9291    D(G(z)): 0.5934 / 0.0071
[52/1000][0/2]  Loss_D: 0.9072  Loss_G: 4.9558  D(x): 0.9227    D(G(z)): 0.5085 / 0.0195
[53/1000][0/2]  Loss_D: 0.6893  Loss_G: 4.0460  D(x): 0.7871    D(G(z)): 0.2954 / 0.0318
[54/1000][0/2]  Loss_D: 0.6171  Loss_G: 2.6043  D(x): 0.6864    D(G(z)): 0.1810 / 0.0897
[55/1000][0/2]  Loss_D: 1.4700  Loss_G: 1.0285  D(x): 0.2871    D(G(z)): 0.0320 / 0.4263
[56/1000][0/2]  Loss_D: 0.7448  Loss_G: 3.6636  D(x): 0.5409    D(G(z)): 0.0393 / 0.0637
```

Results

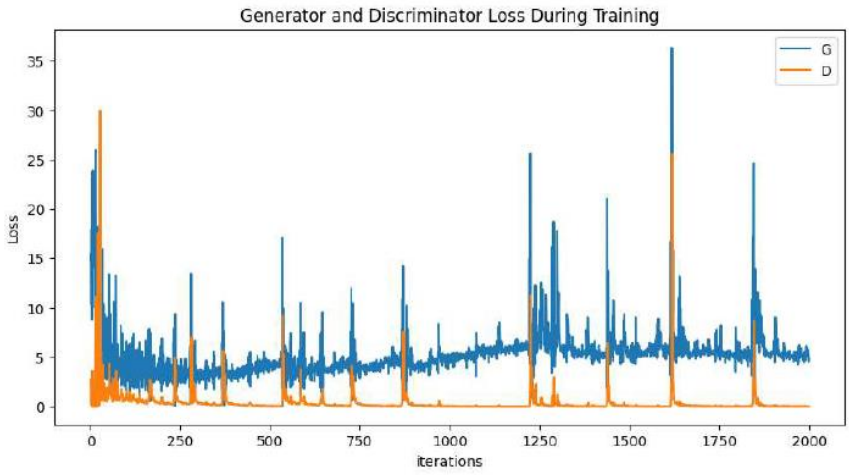
Finally, lets check out how we did. Here, we will look at three different results. First, we will see how D and G's losses changed during training. Second, we will visualize G's output on the fixed_noise batch for every epoch. And third, we will look at a batch of real data next to a batch of fake data from G.

Loss versus training iteration

Below is a plot of D & G's losses versus training iterations.

```
plt.figure(figsize=(10,5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(G_losses,label="G")
plt.plot(D_losses,label="D")
plt.xlabel("iterations")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

APPENDIX 1(continued). Google Colab_ DCGAN_Python_Code



Visualization of G's progression

Remember how we saved the generator's output on the fixed_noise batch after every epoch of training. Now, we can visualize the training progression of G with an animation. Press the play button to start the animation.

```
#%%capture
fig = plt.figure(figsize=(8,8))
plt.axis("off")
ims = [[plt.imshow(np.transpose(i,(1,2,0)), animated=True)] for i in img_list]
ani = animation.ArtistAnimation(fig, ims, interval=1000, repeat_delay=1000, blit=True)

HTML(ani.to_jshtml())
```

APPENDIX 2. Google Colab_ Data Augmentation for Pix2pix_Python_Code


```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install Augmentor
```

```
Collecting Augmentor
  Using cached Augmentor-0.2.12-py2.py3-none-any.whl (38 kB)
Requirement already satisfied: Pillow>=5.2.0 in /usr/local/lib/python3.10/dist-packages (from Augmentor) (9.4.0)
Requirement already satisfied: tqdm>=4.9.0 in /usr/local/lib/python3.10/dist-packages (from Augmentor) (4.66.1)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from Augmentor) (1.23.5)
Installing collected packages: Augmentor
Successfully installed Augmentor-0.2.12
```

```
import Augmentor
import os
import shutil
from PIL import Image
import numpy as np
```

Documentation on Augmentor can be found here: https://github.com/mdbloice/Augmentor

```
pathA = '/content/drive/MyDrive/open_plan_04/A'
pathB = '/content/drive/MyDrive/open_plan_04/B'
```

```
p = Augmentor.Pipeline(pathA)
# Point to a directory containing ground truth data.
# Images with the same file names will be added as ground truth data
# and augmented in parallel to the original data.
p.ground_truth(pathB)

# Add operations to the pipeline:
p.rotate90(probability=0.25)
p.rotate270(probability=0.25)
p.flip_left_right(probability=0.25)
p.flip_top_bottom(probability=0.25)
```

```
Initialised with 19 image(s) found.
Output directory set to /content/drive/MyDrive/open_plan_04/A/output.
                                                    19 ground truth image(s) found.
```

```
IMAGE_COUNT = 300

p.sample(IMAGE_COUNT)
```

```
Processing <PIL.Image.Image image mode=RGBA size=595x1181 at 0x7A76B4217610>: 100%|██████████| 300/300 [00:23<00:00, 12.50 Samples/s
```

APPENDIX 3. Google Colab_ Data Preparation for (256*256) Pix2pix_Python_Code


```
from PIL import Image, ImageOps
import os

# Set the path to the folder containing the images to be resized
input_path = '/content/drive/MyDrive/open_plan_04/A/output'
output_path = '/content/drive/MyDrive/open_plan_04/R2'

# Set the desired output size
size = (256, 256)

# Set the desired border size
border_size = 50

# Check if the output directory exists, create it if necessary
if not os.path.exists(output_path):
    os.makedirs(output_path)

# Loop through all the images in the folder
for file_name in os.listdir(input_path):
    try:
        # Open the image using PIL
        image = Image.open(os.path.join(input_path, file_name))

        # Resize the image while maintaining the aspect ratio
        image.thumbnail(size, Image.ANTIALIAS)

        # Create a new image with a white background and paste the resized image onto it
        new_image = Image.new('RGB', size, (255, 255, 255))
        new_image.paste(image, ((size[0] - image.size[0]) // 2, (size[1] - image.size[1]) // 2))

        # Add a border to the image
        bordered_image = ImageOps.expand(new_image, border=border_size, fill='white')

        # Save the new image with the same file name in the output directory
        output_file_path = os.path.join(output_path, file_name)
        bordered_image.save(output_file_path)
        print(f"Processed and saved: {output_file_path}")
    except Exception as e:
        print(f"Error processing {file_name}: {e}")
```

APPENDIX 4. Google Colab_ Data Preparation for Input and Ground Truth Images of Pix2pix Algorithm_Python_Code


```
import os
import shutil

# Kaynak dizini belirtin
source_folder = "/content/drive/MyDrive/open_plan_04/R2"

# Hedef dizinleri belirtin
newPathA = "/content/drive/MyDrive/open_plan_04/A2/"
newPathB = "/content/drive/MyDrive/open_plan_04/B2/"

# Hedef dizinleri oluşturun
if not os.path.exists(newPathA):
    os.makedirs(newPathA)

if not os.path.exists(newPathB):
    os.makedirs(newPathB)

# Kaynak dizinindeki tüm dosyaları kontrol et
for filename in os.listdir(source_folder):
    try:
        # Dosyanın adına göre ayır ve kopyala
        if '_groundtruth_' in filename:
            guid = filename[-16:]
            shutil.copyfile(os.path.join(source_folder, filename), os.path.join(newPathA, guid))
        elif 'A_original' in filename:
            guid = filename[-16:]
            shutil.copyfile(os.path.join(source_folder, filename), os.path.join(newPathB, guid))
    except Exception as e:
        print(f"Error processing {filename}: {e}")

# Ayrılan dosyaların adlarını kontrol et
A_files = [f[:-4] for f in os.listdir(newPathA)]
B_files = [f[:-4] for f in os.listdir(newPathB)]

print("A Filenames Found: ", A_files)
print("B Filenames Found: ", B_files)
```

```
A Filenames Found:  ['c59eb9e6c57e', '6d580f412482', '864972ea2e64', 'fc7dc95697c9', '8a286567cc7e', '5712c562c884', 'abbd75522dbe',
B Filenames Found:  ['c59eb9e6c57e', '6d580f412482', 'fc7dc95697c9', '864972ea2e64', '8a286567cc7e', '5712c562c884', 'abbd75522dbe',
```

after merging all images now rescalling

```
import os
from PIL import Image

# Değişkenleri ayarla (gerekirse değiştir)
newPathA = "/content/drive/MyDrive/open_plan_04/B2"
newPathB = "/content/drive/MyDrive/open_plan_04/A2"
combinedPath = "/content/drive/MyDrive/open_plan_04/Combined"


# Scaling factor for resizing images
scaling_factor = 0.5  # Adjust this value based on your requirements

for filename in os.listdir(newPathA):
    try:
        # Construct the file paths for the corresponding images in A_files and B_files
        image_path_A = os.path.join(newPathA, filename)
        image_path_B = os.path.join(newPathB, filename)

        # Open the images using PIL
        im1 = Image.open(image_path_B)
        im2 = Image.open(image_path_A)

        # Scale images
        im1 = im1.resize((int(im1.width * scaling_factor), int(im1.height * scaling_factor)))
        im2 = im2.resize((int(im2.width * scaling_factor), int(im2.height * scaling_factor)))

        # Resize im1 to fit half of the final image width
        im1 = im1.resize((256, 256))

        # Resize im2 to fit the other half of the final image width
        im2 = im2.resize((256, 256))

        # Combine images horizontally using PIL
        dst = Image.new('RGB', (512, 256))
        dst.paste(im1, (0, 0))          # Paste im1 at the top-left corner
        dst.paste(im2, (256, 0))        # Paste im2 to the right of im1
```

APPENDIX 4 (Continued). Google Colab_ Data Preparation for Input and Ground Truth Images of Pix2pix Algorithm_Python_Code


```
import os
from PIL import Image

# Değişkenleri ayarla (gerekirse değiştir)
newPathA = "/content/drive/MyDrive/open_plan_04/B2"
newPathB = "/content/drive/MyDrive/open_plan_04/A2"
combinedPath = "/content/drive/MyDrive/open_plan_04/recombined"

for filename in os.listdir(newPathA):
    try:
        # Construct the file paths for the corresponding images in A_files and B_files
        image_path_A = os.path.join(newPathA, filename)
        image_path_B = os.path.join(newPathB, filename)

        # Open the images using PIL
        im1 = Image.open(image_path_B)
        im2 = Image.open(image_path_A)

        # Get the original dimensions of the images
        width1, height1 = im1.size
        width2, height2 = im2.size

        # Ensure both images have the same height
        min_height = min(height1, height2)
        im1 = im1.resize((int(width1 * min_height / height1), min_height))
        im2 = im2.resize((int(width2 * min_height / height2), min_height))

        # Combine images horizontally using PIL
        dst = Image.new('RGB', (im1.width + im2.width, min_height))
        dst.paste(im1, (0, 0))           # Paste im1 at the left
        dst.paste(im2, (im1.width, 0))   # Paste im2 to the right of im1

        # Save the combined image
        combined_filename = "Combined_" + filename
        combined_path = os.path.join(combinedPath, combined_filename)
        dst.save(combined_path)

        print(f"Combined and saved {filename} to {combined_path}")
    except Exception as e:
        print(f"Error processing {filename}: {e}")
```

```
Combined and saved 9376a078c07b.png to /content/drive/MyDrive/open_plan_04/recombined/Combined_9376a078c07b.png
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
from PIL import Image
import os

# Set the path to the folder containing the left images
left_path = '/content/drive/MyDrive/open_plan_2/B2'

# Set the path to the folder containing the right images
right_path = '/content/drive/MyDrive/open_plan_2/A2'

# Set the output path for the merged images
output_path = '/content/drive/MyDrive/open_plan_2/NEW'

# Loop through all the left images in the folder
for left_filename in os.listdir(left_path):
    # Check if the file is an image
    if left_filename.endswith('.jpg') or left_filename.endswith('.png'):
        # Open the left image using PIL
        left = Image.open(os.path.join(left_path, left_filename))

        # Loop through all the right images in the folder
        for right_filename in os.listdir(right_path):
            # Check if the file is an image
            if right_filename.endswith('.jpg') or right_filename.endswith('.png'):
                # Open the right image using PIL
                right = Image.open(os.path.join(right_path, right_filename))

                # Resize the left image to fit the right
                left_width, left_height = left.size
                right_width, right_height = right.size
                new_left_width = int(left_width * right_height / left_height)
                new_left_height = right_height
                resized_left = left.resize((new_left_width, new_left_height))

                # Create a new output image by pasting the left on the right
                output_image = Image.new('RGB', (left_width+right_width, right_height), (255, 255, 255))
                output_image.paste(resized_left, (0, 0))
                output_image.paste(right, (new_left_width, 0))

                # Save the output image to the specified output path
                output_filename = left_filename.split('.')[0] + '_' + right_filename.split('.')[0] + '.png'
                output_image.save(os.path.join(output_path, output_filename))
```

```
from PIL import Image
import os

# Set the directory where the images are located
img_dir = '/content/drive/MyDrive/open_plan_2/NEW'

# Set the new size for the images
new_size = (512, 256)

# Loop through all the images in the directory
for filename in os.listdir(img_dir):
    # Open the image
    img = Image.open(os.path.join(img_dir, filename))

    # Resize the image
    img = img.resize(new_size)

    # Save the resized image, overwriting the original file
    img.save(os.path.join(img_dir, filename))
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code

Copyright 2019 The TensorFlow Authors.

Licensed under the Apache License, Version 2.0 (the "License");

Licensed under the Apache License, Version 2.0 (the "License");

Kodu göster

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Sun Nov 19 09:40:17 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   53C    P8    10W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Sun Nov 19 09:40:18 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   53C    P8    10W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
  print('Not using a high-RAM runtime')
else:
  print('You are using a high-RAM runtime!')
```

```
Your runtime has 13.6 gigabytes of available RAM

Not using a high-RAM runtime
```

```
from google.colab import drive
drive.mount('/content/drive')
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
Mounted at /content/drive
```

Pix2Pix

View on TensorFlow.org Run in Google Colab View source on GitHub Download notebook

Nate - Additional Notes:

This notebook is an edited version of the Tensorflow 2.X Pix2Pix sample. It's designed to load a formatted dataset from Google Drive, train a new model, and save a Keras model back to Google Drive in the ".h5" model format.

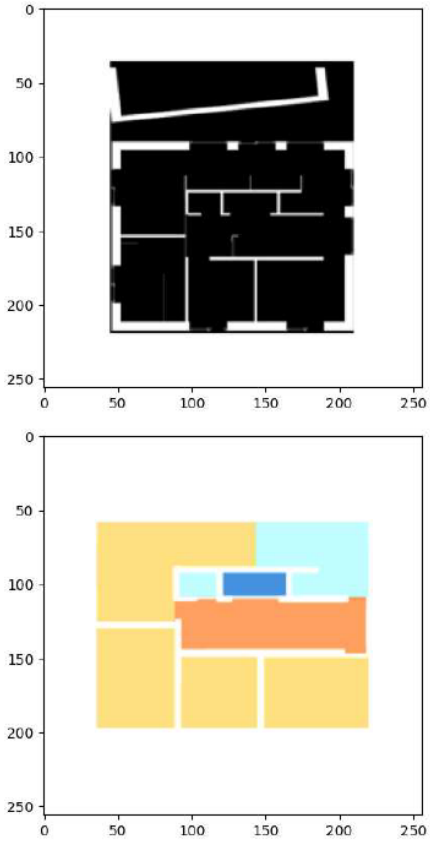
Most of the original notes and model diagrams have been removed from the notebook so the model can be re-executed faster after re-connecting to Colab. See the original notebook for better documentation.

The only changes (so far) are in the image loader code, and the model saving. This notebook still saves checkpoints.

```
import tensorflow as tf

import os
import time

from matplotlib import pyplot as plt
from IPython import display
import numpy as np


# PATH = '/content/drive/MyDrive/open_plan_04'

PATH = '/content/drive/MyDrive/open_plan_04'

BUFFER_SIZE = 400
BATCH_SIZE = 1
IMG_WIDTH = 256
IMG_HEIGHT = 256


def load(image_file):
  image = tf.io.read_file(image_file)
  image = tf.image.decode_jpeg(image)

  w = tf.shape(image)[1]

  # 'w' should be 256:
  w = w // 2

  # Reversing to match our image formatting:
  input_image = image[:, :w, :]
  real_image = image[:, w:, :]

  input_image = tf.cast(input_image, tf.float32)
  real_image = tf.cast(real_image, tf.float32)

  return input_image, real_image


inp, re = load(PATH+'/recombinedTrain/Combined_0aa7cdea952e.png')
# casting to int for matplotlib to show the image
plt.figure()
plt.imshow(inp/255.0)
plt.figure()
plt.imshow(re/255.0)
```

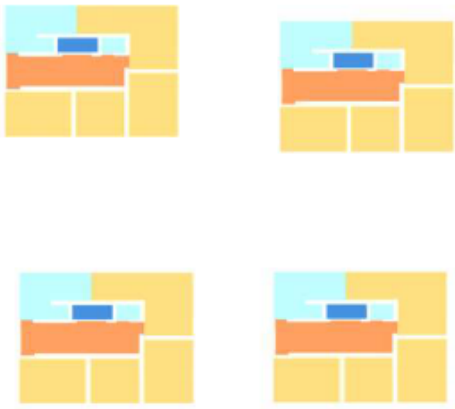
APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
<matplotlib.image.AxesImage at 0x79d87dc91900>
```



APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
def resize(input_image, real_image, height, width):
  input_image = tf.image.resize(input_image, [height, width],
                                method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
  real_image = tf.image.resize(real_image, [height, width],
                               method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)

  return input_image, real_image

def random_crop(input_image, real_image):
  stacked_image = tf.stack([input_image, real_image], axis=0)
  cropped_image = tf.image.random_crop(
      stacked_image, size=[2, IMG_HEIGHT, IMG_WIDTH, 3])

  return cropped_image[0], cropped_image[1]

# normalizing the images to [-1, 1]

def normalize(input_image, real_image):
  input_image = (input_image / 127.5) - 1
  real_image = (real_image / 127.5) - 1

  return input_image, real_image

@tf.function()
def random_jitter(input_image, real_image):
  # resizing to 286 x 286 x 3
  input_image, real_image = resize(input_image, real_image, 286, 286)

  # randomly cropping to 256 x 256 x 3
  input_image, real_image = random_crop(input_image, real_image)

  if tf.random.uniform(()) > 0.5:
    # random mirroring
    input_image = tf.image.flip_left_right(input_image)
    real_image = tf.image.flip_left_right(real_image)

  return input_image, real_image


plt.figure(figsize=(6, 6))
for i in range(4):
  rj_inp, rj_re = random_jitter(inp, re)
  plt.subplot(2, 2, i+1)
  plt.imshow(rj_re/255.0)
  plt.axis('off')
  # print(rj_inp)
plt.show()
```



APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
def load_image_train(image_file):
  input_image, real_image = load(image_file)

  input_image, real_image = random_jitter(input_image, real_image)
  input_image, real_image = normalize(input_image, real_image)

  return input_image, real_image

def load_image_test(image_file):
  input_image, real_image = load(image_file)
  input_image, real_image = resize(input_image, real_image, IMG_HEIGHT, IMG_WIDTH)
  input_image, real_image = normalize(input_image, real_image)

  return input_image, real_image
```

Input Pipeline

```
train_dataset = tf.data.Dataset.list_files(PATH+'/recombinedTrain/*.png')
train_dataset = train_dataset.map(load_image_train,
                                  num_parallel_calls=tf.data.experimental.AUTOTUNE)
train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.batch(BATCH_SIZE)

test_dataset = tf.data.Dataset.list_files(PATH+'/recombinedTest/*.png')
test_dataset = test_dataset.map(load_image_test)
test_dataset = test_dataset.batch(BATCH_SIZE)


for example_input, example_target in test_dataset.take(1):
  print(np.asarray(example_input))
```

```
[[[[1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]
   ...
   [1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]]

  [[1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]
   ...
   [1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]]

  [[1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]
   ...
   [1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]]

  ...

  [[1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]
   ...
   [1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]]

  [[1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]
   ...
   [1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]]

  [[1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]
   ...
   [1. 1. 1.]
   [1. 1. 1.]
   [1. 1. 1.]]]]
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code

Build the Generator

- The architecture of generator is a modified U-Net.
- Each block in the encoder is (Conv -> Batchnorm -> Leaky ReLU)
- Each block in the decoder is (Transposed Conv -> Batchnorm -> Dropout(applied to the first 3 blocks) -> ReLU)
- There are skip connections between the encoder and decoder (as in U-Net).

```
OUTPUT_CHANNELS = 3


# @tf.function()
def downsample(filters, size, apply_batchnorm=True):
  initializer = tf.random_normal_initializer(0., 0.02)

  result = tf.keras.Sequential()
  result.add(
      tf.keras.layers.Conv2D(filters, size, strides=2, padding='same',
                             kernel_initializer=initializer, use_bias=False))

  if apply_batchnorm:
    result.add(tf.keras.layers.BatchNormalization())

  result.add(tf.keras.layers.LeakyReLU())

  return result


down_model = downsample(3, 4)
down_result = down_model(tf.expand_dims(inp, 0))
print (down_result.shape)
```

```
(1, 128, 128, 3)
```

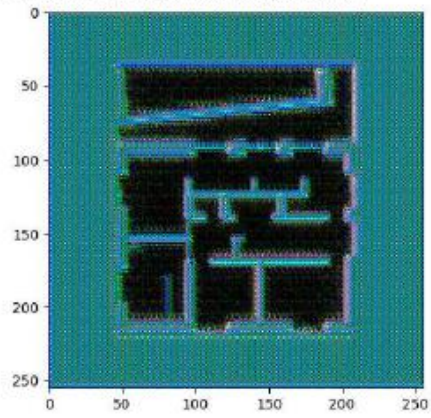
```
# @tf.function()
def upsample(filters, size, apply_dropout=False):
  initializer = tf.random_normal_initializer(0., 0.02)

  result = tf.keras.Sequential()
  result.add(
    tf.keras.layers.Conv2DTranspose(filters, size, strides=2,
                                    padding='same',
                                    kernel_initializer=initializer,
                                    use_bias=False))

  result.add(tf.keras.layers.BatchNormalization())

  if apply_dropout:
      result.add(tf.keras.layers.Dropout(0.5))

  result.add(tf.keras.layers.ReLU())

  return result


up_model = upsample(3, 4)
up_result = up_model(down_result)
print (up_result.shape)
```

```
(1, 256, 256, 3)
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
# @tf.function()
def Generator():
  inputs = tf.keras.layers.Input(shape=[256,256,3])

  down_stack = [
    downsample(64, 4, apply_batchnorm=False), # (bs, 128, 128, 64)
    downsample(128, 4), # (bs, 64, 64, 128)
    downsample(256, 4), # (bs, 32, 32, 256)
    downsample(512, 4), # (bs, 16, 16, 512)
    downsample(512, 4), # (bs, 8, 8, 512)
    downsample(512, 4), # (bs, 4, 4, 512)
    downsample(512, 4), # (bs, 2, 2, 512)
    downsample(512, 4), # (bs, 1, 1, 512)
  ]

  up_stack = [
    upsample(512, 4, apply_dropout=True), # (bs, 2, 2, 1024)
    upsample(512, 4, apply_dropout=True), # (bs, 4, 4, 1024)
    upsample(512, 4, apply_dropout=True), # (bs, 8, 8, 1024)
    upsample(512, 4), # (bs, 16, 16, 1024)
    upsample(256, 4), # (bs, 32, 32, 512)
    upsample(128, 4), # (bs, 64, 64, 256)
    upsample(64, 4), # (bs, 128, 128, 128)
  ]

  initializer = tf.random_normal_initializer(0., 0.02)
  last = tf.keras.layers.Conv2DTranspose(OUTPUT_CHANNELS, 4,
                                         strides=2,
                                         padding='same',
                                         kernel_initializer=initializer,
                                         activation='tanh') # (bs, 256, 256, 3)

  x = inputs

  # Downsampling through the model
  skips = []
  for down in down_stack:
    x = down(x)
    skips.append(x)

  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    x = tf.keras.layers.Concatenate()([x, skip])

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)


generator = Generator()
# tf.keras.utils.plot_model(generator, show_shapes=True, dpi=80)


gen_output = generator(inp[tf.newaxis,...], training=False)
plt.imshow(gen_output[0,...])
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers
<matplotlib.image.AxesImage at 0x79d87cbfa770>
```



APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code

- **Generator loss**
  - It is a sigmoid cross entropy loss of the generated images and an **array of ones**.
  - The paper also includes L1 loss which is MAE (mean absolute error) between the generated image and the target image.
  - This allows the generated image to become structurally similar to the target image.
  - The formula to calculate the total generator loss = gan_loss + LAMBDA * l1_loss, where LAMBDA = 100. This value was decided by the authors of the paper.

The training procedure for the generator is shown below:

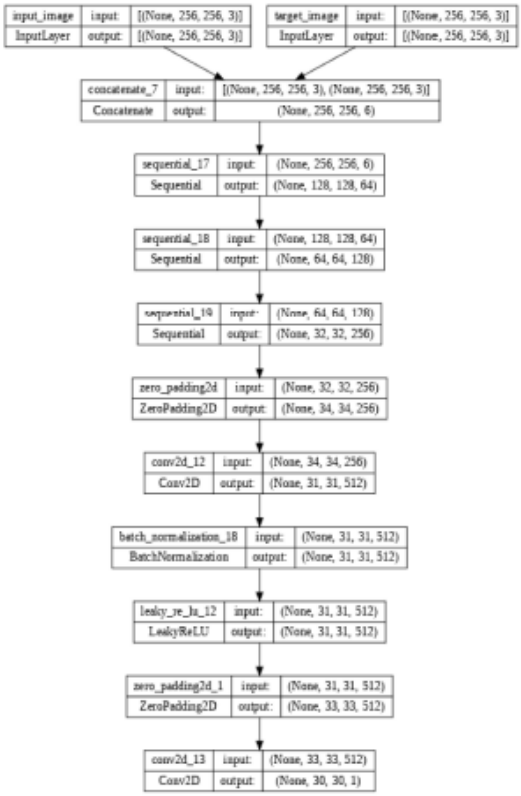
```
LAMBDA = 100

def generator_loss(disc_generated_output, gen_output, target):
  gan_loss = loss_object(tf.ones_like(disc_generated_output), disc_generated_output)

  # mean absolute error
  l1_loss = tf.reduce_mean(tf.abs(target - gen_output))

  total_gen_loss = gan_loss + (LAMBDA * l1_loss)

  return total_gen_loss, gan_loss, l1_loss
```

Build the Discriminator

- The Discriminator is a PatchGAN.
- Each block in the discriminator is (Conv -> BatchNorm -> Leaky ReLU)
- The shape of the output after the last layer is (batch_size, 30, 30, 1)
- Each 30x30 patch of the output classifies a 70x70 portion of the input image (such an architecture is called a PatchGAN).
- Discriminator receives 2 inputs.
  - Input image and the target image, which it should classify as real.
  - Input image and the generated image (output of generator), which it should classify as fake.
  - We concatenate these 2 inputs together in the code (`tf.concat([inp, tar], axis=-1)`)

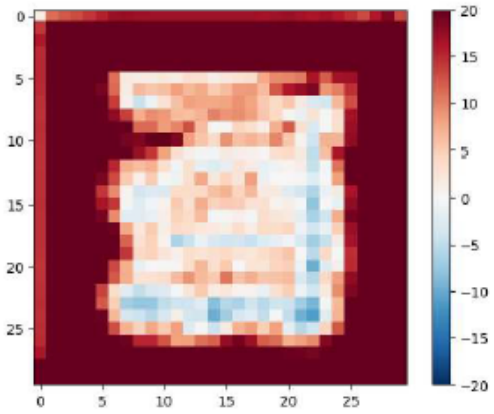
```
def Discriminator():
  initializer = tf.random_normal_initializer(0., 0.02)

  inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_image')
  tar = tf.keras.layers.Input(shape=[256, 256, 3], name='target_image')

  x = tf.keras.layers.concatenate([inp, tar]) # (bs, 256, 256, channels*2)

  down1 = downsample(64, 4, False)(x) # (bs, 128, 128, 64)
  down2 = downsample(128, 4)(down1) # (bs, 64, 64, 128)
  down3 = downsample(256, 4)(down2) # (bs, 32, 32, 256)

  zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3) # (bs, 34, 34, 256)
  conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                                kernel_initializer=initializer,
                                use_bias=False)(zero_pad1) # (bs, 31, 31, 512)

  batchnorm1 = tf.keras.layers.BatchNormalization()(conv)

  leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)

  zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu) # (bs, 33, 33, 512)

  last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                kernel_initializer=initializer)(zero_pad2) # (bs, 30, 30, 1)

  return tf.keras.Model(inputs=[inp, tar], outputs=last)

discriminator = Discriminator()
tf.keras.utils.plot_model(discriminator, show_shapes=True, dpi=64)
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code




```
disc_out = discriminator([inp[tf.newaxis,...], gen_output], training=False)
plt.imshow(disc_out[0,...,-1], vmin=-20, vmax=20, cmap='RdBu_r')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x79d87cb5aa10>
```




```
loss_object = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
def discriminator_loss(disc_real_output, disc_generated_output):
  real_loss = loss_object(tf.ones_like(disc_real_output), disc_real_output)

  generated_loss = loss_object(tf.zeros_like(disc_generated_output), disc_generated_output)

  total_disc_loss = real_loss + generated_loss

  return total_disc_loss
```

Define the Optimizers and Checkpoint-saver

```
generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
```

```
checkpoint_dir = './training_checkpoints_100'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)
```

Generate Images

Write a function to plot some images during training.

- We pass images from the test dataset to the generator.
- The generator will then translate the input image into the output.
- Last step is to plot the predictions and **voila!**

Note: The `training=True` is intentional here since we want the batch statistics while running the model on the test dataset. If we use training=False, we will get the accumulated statistics learned from the training dataset (which we don't want)

```
def generate_images(model, test_input, tar):
  prediction = model(test_input, training=True)
  plt.figure(figsize=(15,15))

  display_list = [test_input[0], tar[0], prediction[0]]
  title = ['Input Image', 'Ground Truth', 'Predicted Image']

  for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.title(title[i])
    # getting the pixel values between [0, 1] to plot it.
    plt.imshow(display_list[i] * 0.5 + 0.5)
    plt.axis('off')
  plt.show()
```

```
for example_input, example_target in test_dataset.take(3):
  generate_images(generator, example_input, example_target)
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code

Input Image Ground Truth Predicted

Input Image Ground Truth Predicted

Input Image Ground Truth Predicted

Training

- For each example input generate an output.
- The discriminator receives the input_image and the generated image as the first input. The second input is the input_image and the target_image.
- Next, we calculate the generator and the discriminator loss.
- Then, we calculate the gradients of loss with respect to both the generator and the discriminator variables(inputs) and apply those to the optimizer.
- Then log the losses to TensorBoard.

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
EPOCHS = 120

import datetime
log_dir="logs/"

summary_writer = tf.summary.create_file_writer(
  log_dir + "fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))


@tf.function
def train_step(input_image, target, epoch):
  with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
    gen_output = generator(input_image, training=True)

    disc_real_output = discriminator([input_image, target], training=True)
    disc_generated_output = discriminator([input_image, gen_output], training=True)

    gen_total_loss, gen_gan_loss, gen_l1_loss = generator_loss(disc_generated_output, gen_output, target)
    disc_loss = discriminator_loss(disc_real_output, disc_generated_output)

  generator_gradients = gen_tape.gradient(gen_total_loss,
                                          generator.trainable_variables)
  discriminator_gradients = disc_tape.gradient(disc_loss,
                                               discriminator.trainable_variables)

  generator_optimizer.apply_gradients(zip(generator_gradients,
                                          generator.trainable_variables))
  discriminator_optimizer.apply_gradients(zip(discriminator_gradients,
                                              discriminator.trainable_variables))

  with summary_writer.as_default():
    tf.summary.scalar('gen_total_loss', gen_total_loss, step=epoch)
    tf.summary.scalar('gen_gan_loss', gen_gan_loss, step=epoch)
    tf.summary.scalar('gen_l1_loss', gen_l1_loss, step=epoch)
    tf.summary.scalar('disc_loss', disc_loss, step=epoch)
```

The actual training loop:

- Iterates over the number of epochs.
- On each epoch it clears the display, and runs `generate_images` to show it's progress.
- On each epoch it iterates over the training dataset, printing a '.' for each example.
- It saves a checkpoint every 20 epochs.

```
def fit(train_ds, epochs, test_ds):
  for epoch in range(epochs):
    start = time.time()

    display.clear_output(wait=True)

    for example_input, example_target in test_ds.take(1):
      generate_images(generator, example_input, example_target)
    print("Epoch: ", epoch)

    # Train
    for n, (input_image, target) in train_ds.enumerate():
      print('.', end='')
      if (n+1) % 100 == 0:
        print()
      train_step(input_image, target, epoch)
    print()

    # saving (checkpoint) the model every 10 epochs
    # if (epoch + 10)  == 0:
    #  checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time taken for epoch {} is {} sec\n'.format(epoch + 1,
                                                        time.time()-start))
  checkpoint.save(file_prefix = checkpoint_prefix)
  # New save method for h5 format:
  # Uncomment for training in Colab instances:
  h5_dir = '/content/drive/MyDrive/open_plan_04/Combined'
  h5_name = 'generator_model_003_edges2daisies_061720.h5'
  generator.save(h5_dir+h5_name)
```

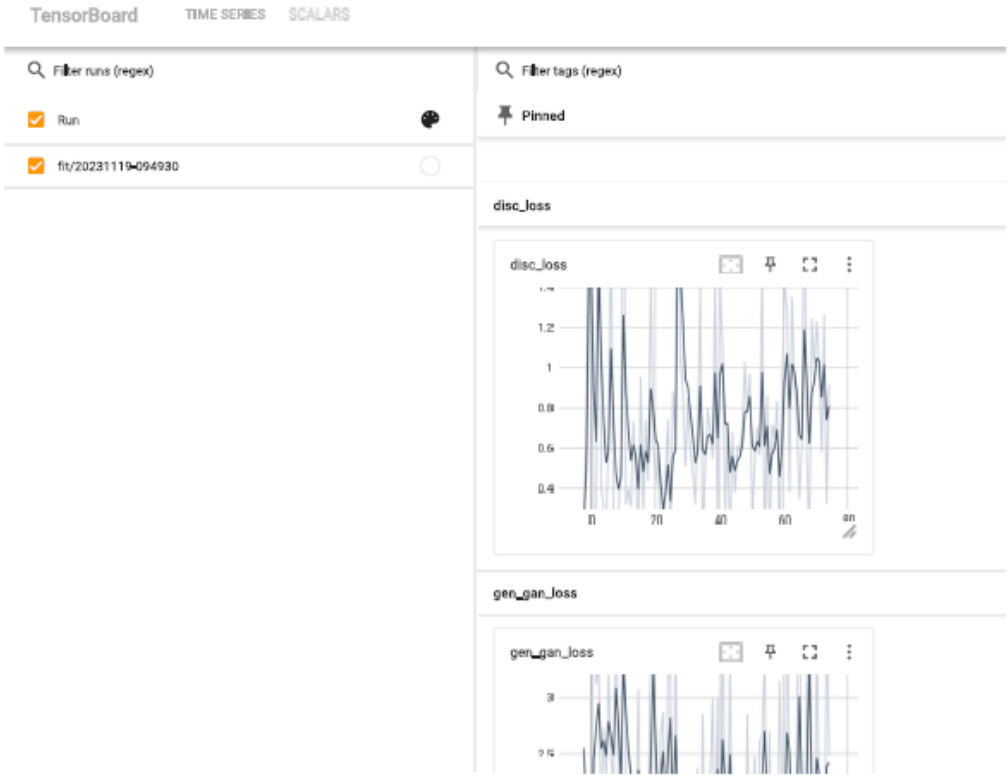
Now run the training loop:

```
fit(train_dataset, EPOCHS, test_dataset)
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
Epoch:  108
```

```
%load_ext tensorboard
%tensorboard --logdir {log_dir}
```




```
# Clear any logs from previous runs
!rm -rf ./logs/

# Save the model back to Google Drive:
generator.save('/content/drive/MyDrive/open_plan_04/generator_model_002_epochs_100.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty u
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
h5_dir = '/content/drive/MyDrive/open_plan_04'
h5_name = 'generator_model_003_061620.h5'
generator.save(h5_dir+h5_name)
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty
```

Restore the latest checkpoint and test

```
!ls {checkpoint_dir}
```

```
checkpoint  ckpt-1.data-00000-of-00001  ckpt-1.index
```

```
# restoring the latest checkpoint in checkpoint_dir
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x79d40377aa40>
```

Generate using test dataset

```
# Run the trained model on a few examples from the test dataset
# for inp, tar in test_dataset.take(10):
#   generate_images(generator, inp, tar)
```

```
# This works! :
generator.save('/content/drive/MyDrive/open_plan_3/generator_model_002_epochs_100.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty
```

```
# Recreate the exact same model, including its weights and the optimizer
new_model = tf.keras.models.load_model('/content/drive/MyDrive/open_plan_04/generator_model_002_epochs_100.h5')

# Show the model architecture
new_model.summary()
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
Model: "model"
```

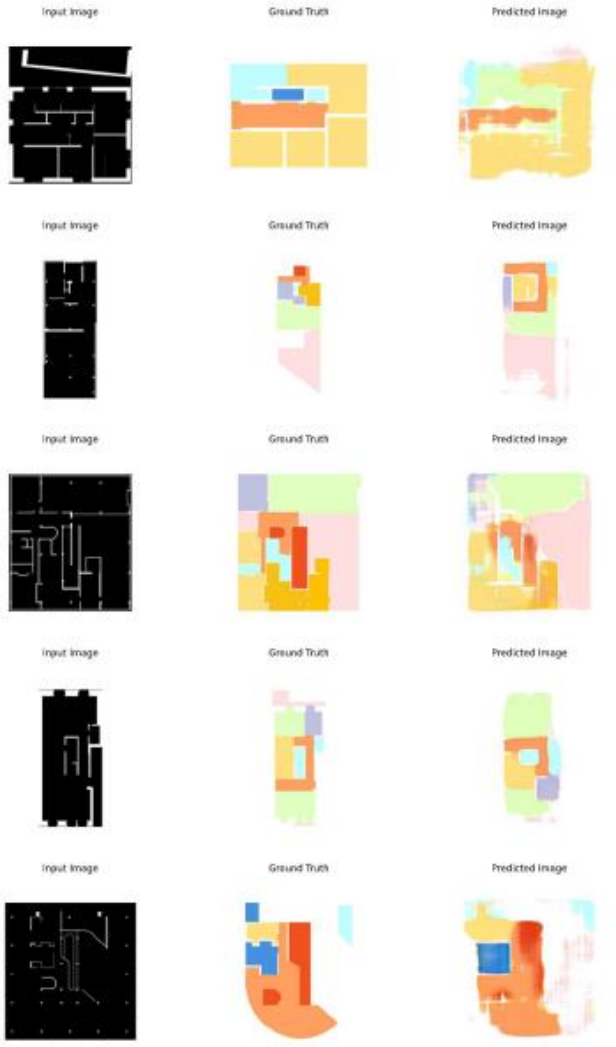
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)]	0	[]
sequential_2 (Sequential)	(None, 128, 128, 64)	3072	['input_1[0][0]']
sequential_3 (Sequential)	(None, 64, 64, 128)	131584	['sequential_2[0][0]']
sequential_4 (Sequential)	(None, 32, 32, 256)	525312	['sequential_3[0][0]']
sequential_5 (Sequential)	(None, 16, 16, 512)	2099200	['sequential_4[0][0]']
sequential_6 (Sequential)	(None, 8, 8, 512)	4196352	['sequential_5[0][0]']
sequential_7 (Sequential)	(None, 4, 4, 512)	4196352	['sequential_6[0][0]']
sequential_8 (Sequential)	(None, 2, 2, 512)	4196352	['sequential_7[0][0]']
sequential_9 (Sequential)	(None, 1, 1, 512)	4196352	['sequential_8[0][0]']
sequential_10 (Sequential)	(None, 2, 2, 512)	4196352	['sequential_9[0][0]']
concatenate (Concatenate)	(None, 2, 2, 1024)	0	['sequential_10[0][0]', 'sequential_8[0][0]']
sequential_11 (Sequential)	(None, 4, 4, 512)	8390656	['concatenate[0][0]']
concatenate_1 (Concatenate)	(None, 4, 4, 1024)	0	['sequential_11[0][0]', 'sequential_7[0][0]']
sequential_12 (Sequential)	(None, 8, 8, 512)	8390656	['concatenate_1[0][0]']
concatenate_2 (Concatenate)	(None, 8, 8, 1024)	0	['sequential_12[0][0]', 'sequential_6[0][0]']
sequential_13 (Sequential)	(None, 16, 16, 512)	8390656	['concatenate_2[0][0]']
concatenate_3 (Concatenate)	(None, 16, 16, 1024)	0	['sequential_13[0][0]', 'sequential_5[0][0]']

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code

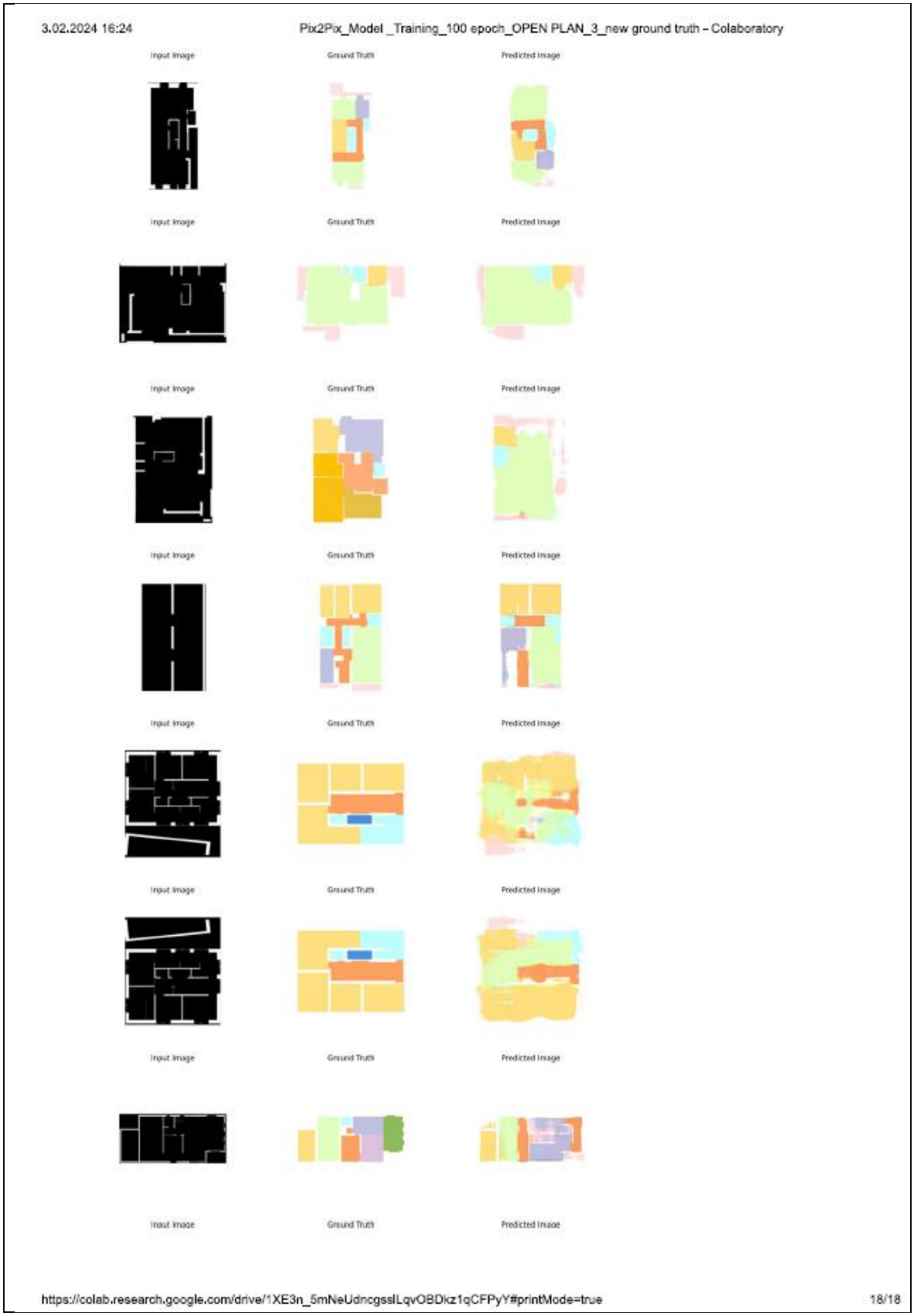

```
sequential_14 (Sequential)  (None, 32, 32, 256)         4195328   ['concatenate_3[0][0]']

concatenate_4 (Concatenate  (None, 32, 32, 512)         0         ['sequential_14[0][0]',
)                                                                   'sequential_4[0][0]']

sequential_15 (Sequential)  (None, 64, 64, 128)         1049088   ['concatenate_4[0][0]']

concatenate_5 (Concatenate  (None, 64, 64, 256)         0         ['sequential_15[0][0]',
)                                                                   'sequential_3[0][0]']

sequential_16 (Sequential)  (None, 128, 128, 64)        262400    ['concatenate_5[0][0]']

concatenate_6 (Concatenate  (None, 128, 128, 128)       0         ['sequential_16[0][0]',
)                                                                   'sequential_2[0][0]']
```

```
for inp, tar in test_dataset.take(5):
  generate_images(new_model, inp, tar)
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code



APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code



APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
# Use the Google Colab tools to connect to Google Drive. If you're running this notebook locally, you should download your `.h5` model ar

import tensorflow as tf

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# The path to the .h5 checkpoint that was saved after the training loop:
MODEL_PATH = '/content/drive/MyDrive/open_plan_2/generator_model_002_epochs_100.h5'

# Location of the image you want to test with. The images should be an RGB .png or .jpg file, scaled to 256x256 pixels
INPUT_IMAGE_PATH = '/content/drive/MyDrive/open_plan_04/B2f001b0d07d05.png'

model = tf.keras.models.load_model(MODEL_PATH)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
```

```
model.summary()
```

```
 sequential_3 (Sequential)   (None, 64, 64, 128)          131584    ['sequential_2[0][0]']

 sequential_4 (Sequential)   (None, 32, 32, 256)          525312    ['sequential_3[0][0]']

 sequential_5 (Sequential)   (None, 16, 16, 512)          2099200   ['sequential_4[0][0]']

 sequential_6 (Sequential)   (None, 8, 8, 512)            4196352   ['sequential_5[0][0]']

 sequential_7 (Sequential)   (None, 4, 4, 512)            4196352   ['sequential_6[0][0]']

 sequential_8 (Sequential)   (None, 2, 2, 512)            4196352   ['sequential_7[0][0]']

 sequential_9 (Sequential)   (None, 1, 1, 512)            4196352   ['sequential_8[0][0]']

 sequential_10 (Sequential)  (None, 2, 2, 512)            4196352   ['sequential_9[0][0]']

 concatenate (Concatenate)   (None, 2, 2, 1024)           0         ['sequential_10[0][0]',
                                                                      'sequential_8[0][0]']

 sequential_11 (Sequential)  (None, 4, 4, 512)            8390656   ['concatenate[0][0]']

 concatenate_1 (Concatenate  (None, 4, 4, 1024)           0         ['sequential_11[0][0]',
 )                                                                    'sequential_7[0][0]']

 sequential_12 (Sequential)  (None, 8, 8, 512)            8390656   ['concatenate_1[0][0]']

 concatenate_2 (Concatenate  (None, 8, 8, 1024)           0         ['sequential_12[0][0]',
 )                                                                    'sequential_6[0][0]']

 sequential_13 (Sequential)  (None, 16, 16, 512)          8390656   ['concatenate_2[0][0]']

 concatenate_3 (Concatenate  (None, 16, 16, 1024)         0         ['sequential_13[0][0]',
 )                                                                    'sequential_5[0][0]']

 sequential_14 (Sequential)  (None, 32, 32, 256)          4195328   ['concatenate_3[0][0]']

 concatenate_4 (Concatenate  (None, 32, 32, 512)          0         ['sequential_14[0][0]',
 )                                                                    'sequential_4[0][0]']

 sequential_15 (Sequential)  (None, 64, 64, 128)          1049088   ['concatenate_4[0][0]']

 concatenate_5 (Concatenate  (None, 64, 64, 256)          0         ['sequential_15[0][0]',
 )                                                                    'sequential_3[0][0]']

 sequential_16 (Sequential)  (None, 128, 128, 64)         262400    ['concatenate_5[0][0]']

 concatenate_6 (Concatenate  (None, 128, 128, 128)        0         ['sequential_16[0][0]',
 )                                                                    'sequential_2[0][0]']

 conv2d_transpose_8 (Conv2D  (None, 256, 256, 3)          6147      ['concatenate_6[0][0]']
 Transpose)

==================================================================================================
Total params: 54425859 (207.62 MB)
Trainable params: 54414979 (207.58 MB)
Non-trainable params: 10880 (42.50 KB)
__________________________________________________________________________________________________
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
img = tf.io.read_file(INPUT_IMAGE_PATH)
img = tf.io.decode_png(img, channels=3)
img = tf.image.convert_image_dtype(img, tf.float32)
img = tf.image.resize(img, (256, 256), antialias=True)
img.shape
```

```
TensorShape([256, 256, 3])
```

```
import matplotlib.pyplot as plt

# Assuming 'img' is already defined
plt.imshow(img)
plt.axis('off')
plt.show()
```

```
input_image = tf.expand_dims(img, axis=[0])
input_image.shape
```

```
TensorShape([1, 256, 256, 3])
```

```
# prediction = model.predict(input_image)
prediction = model(input_image, training=True)
prediction
```

```
<tf.Tensor: shape=(1, 256, 256, 3), dtype=float32, numpy=
array([[[[0.9997685 , 0.9999108 , 0.9999677 ],
         [0.9999953 , 0.99998844, 0.9999923 ],
         [0.9999811 , 0.99982953, 0.99990356],
         ...,
         [1.        , 1.        , 1.        ],
         [0.99999964, 0.99999946, 1.        ],
         [0.9978524 , 0.9985728 , 0.997842  ]],

        [[0.9999973 , 0.99999917, 0.99999917],
         [1.        , 1.        , 1.        ],
         [1.        , 0.99999976, 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [0.9999921 , 0.999989  , 0.99998146]],

        [[0.999998  , 0.99999857, 0.9999996 ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [0.99999714, 0.99999666, 0.9999989 ]],

        ...,

        [[0.9999989 , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [0.9999191 , 0.99996644, 0.99987644]],

        [[0.99999905, 0.99999964, 1.        ],
         [1.        , 1.        , 1.        ],
```

APPENDIX 6. (Continued) Google Colab_ Pix2Pix_Python_Code


```
                [1.        , 1.        , 1.        ],
                ...,
                [1.        , 1.        , 1.        ],
                [1.        , 1.        , 1.        ],
                [0.9999725 , 0.99999267, 0.99999535]],

               [[0.9993966 , 0.9998953 , 0.9999533 ],
                [0.9999997 , 0.99999964, 1.        ],
                [0.9999995 , 0.99999964, 1.        ],
                ...,
                [0.99999964, 0.9999997 , 1.        ],
                [0.9999994 , 0.9999995 , 1.        ],
                [0.99963844, 0.99968934, 0.9999617 ]]]], dtype=float32)>
```

```
prediction.shape
```

```
TensorShape([1, 256, 256, 3])
```

```
import matplotlib.pyplot as plt
plt.imshow(prediction[0, :, :, :])
plt.axis('off')
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integer
```

APPENDIX 7. Google Colab_ Yolov5_Python_Code

Custom Training with YOLOv5

In this tutorial, we assemble a dataset and train a custom YOLOv5 model to recognize the objects in our dataset. To do so we will take the following steps:

- Gather a dataset of images and label our dataset
- Export our dataset to YOLOv5
- Train YOLOv5 to recognize the objects in our dataset
- Evaluate our YOLOv5 model's performance
- Run test inference to view our model at work

Step 1: Install Requirements

```
#clone YOLOv5 and
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow

import torch
import os
from IPython.display import Image, clear_output  # to display images

print(f"Setup complete. Using torch {torch.__version__} ({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else 'CPU
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 16078, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 16078 (delta 6), reused 9 (delta 1), pack-reused 16056
Receiving objects: 100% (16078/16078), 14.64 MiB | 21.95 MiB/s, done.
Resolving deltas: 100% (11038/11038), done.
/content/yolov5
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 190.6/190.6 kB 3.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.6/3.6 MB 13.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 645.8/645.8 kB 18.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 6.5 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
imageio 2.31.6 requires pillow<10.1.0,>=8.3.2, but you have pillow 10.1.0 which is incompatible.
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.3/63.3 kB 1.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.7/178.7 kB 6.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 kB 7.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.1/49.1 MB 13.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.8/67.8 kB 8.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.2/72.2 kB 9.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.5/54.5 kB 7.4 MB/s eta 0:00:00
Setup complete. Using torch 2.1.0+cu118 (Tesla T4)
```

```
from roboflow import Roboflow
rf = Roboflow(model_format="yolov5", notebook="ultralytics")
```

APPENDIX 7. Google Colab_ Yolov5_Python_Code


```
                 all          9         48      0.502      0.553      0.399      0.278

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      93/99      1.59G    0.02844    0.06971    0.02189         66        416: 100% 5/5 [00:00<00:00,  5.56it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  3.96it/s]
                   all          9         48      0.502      0.554        0.4      0.278

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      94/99      1.59G    0.02845    0.06645    0.02265         56        416: 100% 5/5 [00:00<00:00,  5.11it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  4.32it/s]
                   all          9         48      0.507      0.559      0.405      0.281

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      95/99      1.59G    0.02689    0.06873    0.02304         67        416: 100% 5/5 [00:01<00:00,  4.52it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  3.81it/s]
                   all          9         48      0.514      0.564        0.4       0.28

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      96/99      1.59G    0.02785    0.05789    0.02218         52        416: 100% 5/5 [00:00<00:00,  5.05it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  6.24it/s]
                   all          9         48      0.515      0.566        0.4      0.274

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      97/99      1.59G    0.02691    0.06548    0.02148         43        416: 100% 5/5 [00:00<00:00,  6.87it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  8.53it/s]
                   all          9         48      0.517       0.57      0.402      0.271

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      98/99      1.59G    0.02883    0.06738    0.02102         61        416: 100% 5/5 [00:00<00:00,  6.91it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  7.14it/s]
                   all          9         48      0.519      0.569      0.398       0.27

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      99/99      1.59G    0.02806    0.06724    0.02098         59        416: 100% 5/5 [00:00<00:00,  6.78it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  8.45it/s]
                   all          9         48      0.516      0.569      0.388      0.267

100 epochs completed in 0.043 hours.
Optimizer stripped from runs/train/exp4/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/exp4/weights/best.pt, 14.4MB

Validating runs/train/exp4/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7037095 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00, 11.32it/s]
                   all          9         48      0.509      0.532      0.401      0.284
              bathroom          9          3      0.212      0.667      0.235      0.168
                  hall          9         11      0.508      0.727       0.62       0.31
               kitchen          9          6      0.484      0.629      0.392      0.289
           living room          9          6       0.27        0.5       0.22      0.124
                  room          9          9      0.524          1      0.647      0.559
               terrace          9          8          1          0      0.257      0.226
                    wc          9          5      0.563        0.2      0.437      0.309
Results saved to runs/train/exp4
```

Evaluate Custom YOLOv5 Detector Performance

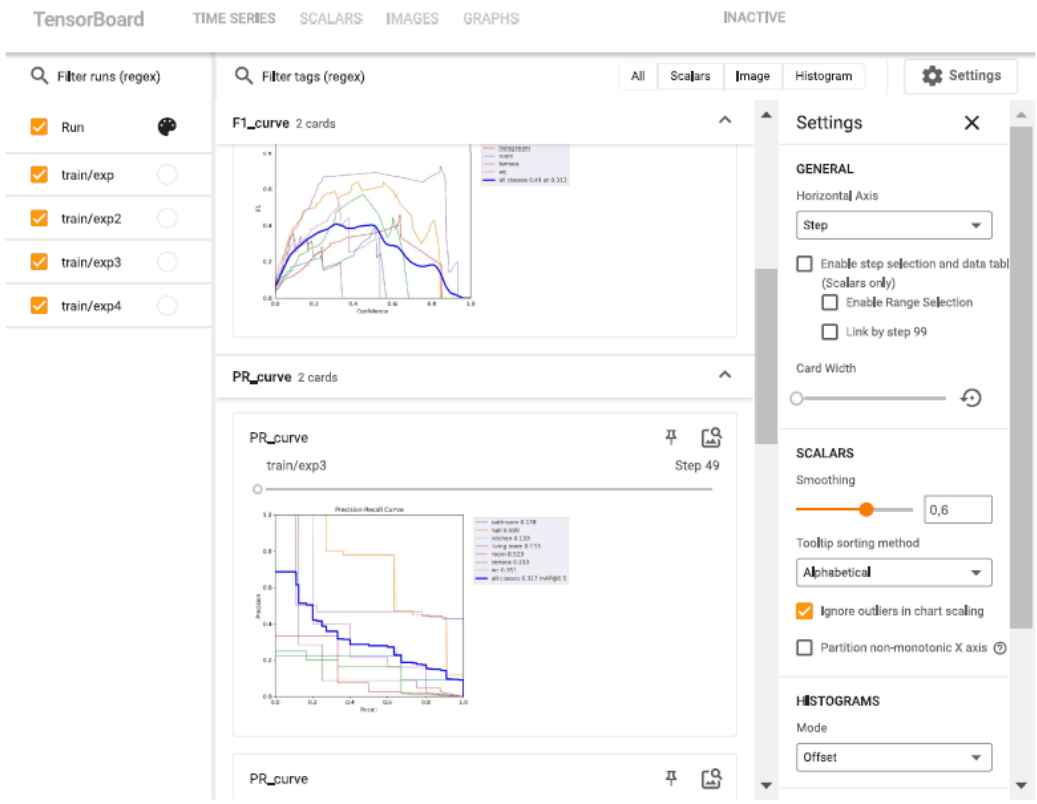
Training losses and performance metrics are saved to Tensorboard and also to a logfile.

If you are new to these metrics, the one you want to focus on is `mAP_0.5` - learn more about mean average precision here.

```
# Start tensorboard
# Launch after you have started training
# logs save in the folder "runs"
%load_ext tensorboard
%tensorboard --logdir runs
```

APPENDIX 7. Google Colab_ Yolov5_Python_Code


```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 2564), started 0:07:42 ago. (Use '!kill 2564' to kill it.)
```



Run Inference With Trained Weights

Run inference with a pretrained checkpoint on contents of `test/images` folder downloaded from Roboflow.

```
!python detect.py --weights runs/train/exp/weights/best.pt --img 416 --conf 0.1 --source {dataset.location}/test/images
```

```
detect: weights=['runs/train/exp/weights/best.pt'], source=/content/yolov5/layout-names-4/test/images, data=data/coco128.yaml, imgs
YOLOv5 🚀 v7.0-247-g3f02fde Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)

Fusing layers...
Model summary: 157 layers, 7037095 parameters, 0 gradients, 15.8 GFLOPs
image 1/9 /content/yolov5/layout-names-4/test/images/11_png.rf.a188389bba1cf3bc2b5f43306c7754d1.jpg: 352x416 8 halls, 5 living rooms
image 2/9 /content/yolov5/layout-names-4/test/images/13_png.rf.b71c58af0828cec3a3ad707fed654ec1.jpg: 320x416 2 halls, 5 living rooms
image 3/9 /content/yolov5/layout-names-4/test/images/15_png.rf.bc2b1e4b332f32232db1f04c9a6048a1.jpg: 320x416 1 bathroom, 3 halls, 7
image 4/9 /content/yolov5/layout-names-4/test/images/16_png.rf.c9d574ec94b908d655ced9256a7b0c6c.jpg: 256x416 1 bathroom, 6 halls, 2
image 5/9 /content/yolov5/layout-names-4/test/images/17_png.rf.9acf69810ca898c513f3bba4ef43501e.jpg: 320x416 1 bathroom, 3 halls, 6
image 6/9 /content/yolov5/layout-names-4/test/images/19_png.rf.241e1d925f03f730501ec7717d7fbd27.jpg: 288x416 2 kitchens, 9 living r
image 7/9 /content/yolov5/layout-names-4/test/images/19_png.rf.6650a2657cb9efbe85a86c3d2d6d868b.jpg: 160x416 8 bathrooms, 14 halls,
image 8/9 /content/yolov5/layout-names-4/test/images/20_png.rf.8398d9b20bd01e138aad8f9f507a29e2.jpg: 416x224 4 bathrooms, 13 halls,
image 9/9 /content/yolov5/layout-names-4/test/images/7_png.rf.c978074cf146fa79604257769f4bbb97.jpg: 288x416 1 bathroom, 5 halls, 3
Speed: 0.3ms pre-process, 34.4ms inference, 8.6ms NMS per image at shape (1, 3, 416, 416)
Results saved to runs/detect/exp
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

APPENDIX 8. Google Colab_ Yolov8_Python_Code

Install YOLOv8

⚠️ YOLOv8 is still under heavy development. Breaking changes are being introduced almost weekly. We strive to make our YOLOv8 notebooks work with the latest version of the library. Last tests took place on **27.01.2023** with version **YOLOv8.0.20**.

If you notice that our notebook behaves incorrectly - especially if you experience errors that prevent you from going through the tutorial - don't hesitate! Let us know and open an issue on the Roboflow Notebooks repository.

YOLOv8 can be installed in two ways-from the source and via pip. This is because it is the first iteration of YOLO to have an official package.

```
# Pip install method (recommended)

!pip install ultralytics==8.0.20

from IPython import display
display.clear_output()

import ultralytics
ultralytics.checks()
```

```
Ultralytics YOLOv8.0.20 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 26.9/78.2 GB disk)
```

```
# Git clone method (for development)

# %cd {HOME}
# !git clone github.com/ultralytics/ultralytics
# %cd {HOME}/ultralytics
# !pip install -e .

# from IPython import display
# display.clear_output()

# import ultralytics
# ultralytics.checks()


from ultralytics import YOLO

from IPython.display import display, Image
```

CLI Basics

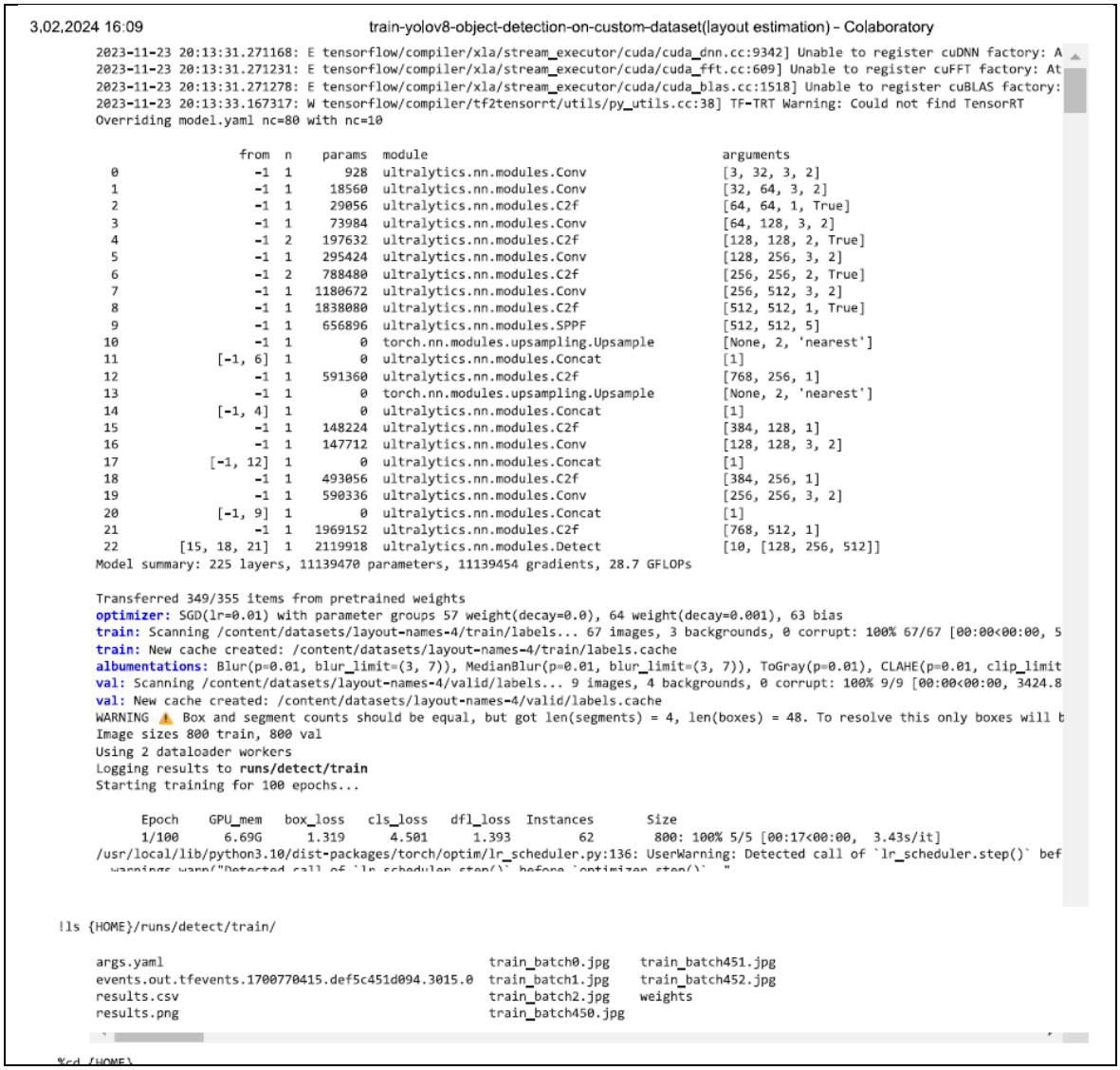
1 hücre gizli

Custom Training

```
%cd {HOME}

!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=100 imgsz=800 plots=True
```

```
/content
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8s.pt to yolov8s.pt...
100% 21.5M/21.5M [00:00<00:00, 267MB/s]

Ultralytics YOLOv8.0.20 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
yolo/engine/trainer: task=detect, mode=train, model=yolov8s.yaml, data=/content/datasets/layout-names-4/data.yaml, epochs=100, p
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 119MB/s]
```

APPENDIX 8. Google Colab_ Yolov8_Python_Code



3,02,2024 16:09 train-yolov8-object-detection-on-custom-dataset(layout estimation) - Colaboratory

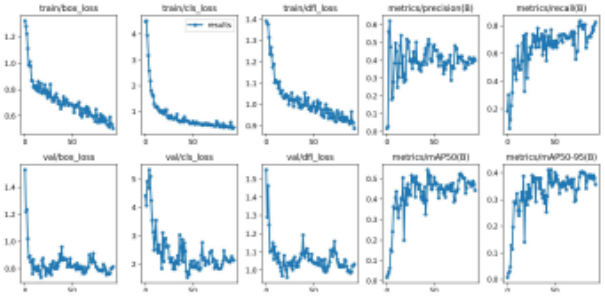
```
2023-11-23 20:13:31.271168: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342] Unable to register cuDNN factory: A
2023-11-23 20:13:31.271231: E tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] Unable to register cuFFT factory: At
2023-11-23 20:13:31.271278: E tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518] Unable to register cuBLAS factory:
2023-11-23 20:13:33.167317: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
Overriding model.yaml nc=80 with nc=10

                   from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.Conv                  [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.Conv                  [32, 64, 3, 2]
  2                  -1  1     29056  ultralytics.nn.modules.C2f                   [64, 64, 1, True]
  3                  -1  1     73984  ultralytics.nn.modules.Conv                  [64, 128, 3, 2]
  4                  -1  2    197632  ultralytics.nn.modules.C2f                   [128, 128, 2, True]
  5                  -1  1    295424  ultralytics.nn.modules.Conv                  [128, 256, 3, 2]
  6                  -1  2    788480  ultralytics.nn.modules.C2f                   [256, 256, 2, True]
  7                  -1  1   1180672  ultralytics.nn.modules.Conv                  [256, 512, 3, 2]
  8                  -1  1   1838080  ultralytics.nn.modules.C2f                   [512, 512, 1, True]
  9                  -1  1    656896  ultralytics.nn.modules.SPPF                  [512, 512, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11            [-1, 6]  1         0  ultralytics.nn.modules.Concat                [1]
 12                  -1  1    591360  ultralytics.nn.modules.C2f                   [768, 256, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14            [-1, 4]  1         0  ultralytics.nn.modules.Concat                [1]
 15                  -1  1    148224  ultralytics.nn.modules.C2f                   [384, 128, 1]
 16                  -1  1    147712  ultralytics.nn.modules.Conv                  [128, 128, 3, 2]
 17           [-1, 12]  1         0  ultralytics.nn.modules.Concat                [1]
 18                  -1  1    493056  ultralytics.nn.modules.C2f                   [384, 256, 1]
 19                  -1  1    590336  ultralytics.nn.modules.Conv                  [256, 256, 3, 2]
 20            [-1, 9]  1         0  ultralytics.nn.modules.Concat                [1]
 21                  -1  1   1969152  ultralytics.nn.modules.C2f                   [768, 512, 1]
 22        [15, 18, 21]  1   2119918  ultralytics.nn.modules.Detect                [10, [128, 256, 512]]
Model summary: 225 layers, 11139470 parameters, 11139454 gradients, 28.7 GFLOPs

Transferred 349/355 items from pretrained weights
optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.001), 63 bias
train: Scanning /content/datasets/layout-names-4/train/labels... 67 images, 3 backgrounds, 0 corrupt: 100% 67/67 [00:00<00:00, 5
train: New cache created: /content/datasets/layout-names-4/train/labels.cache
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01), CLAHE(p=0.01, clip_limit
val: Scanning /content/datasets/layout-names-4/valid/labels... 9 images, 4 backgrounds, 0 corrupt: 100% 9/9 [00:00<00:00, 3424.8
val: New cache created: /content/datasets/layout-names-4/valid/labels.cache
WARNING ⚠ Box and segment counts should be equal, but got len(segments) = 4, len(boxes) = 48. To resolve this only boxes will b
Image sizes 800 train, 800 val
Using 2 dataloader workers
Logging results to runs/detect/train
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      6.69G      1.319      4.501      1.393         62        800: 100% 5/5 [00:17<00:00,  3.43s/it]
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:136: UserWarning: Detected call of `lr_scheduler.step()` bef
  warnings.warn("Detected call of `lr_scheduler.step()` before `optimizer.step()`. "
```

```
!ls {HOME}/runs/detect/train/

args.yaml                                       train_batch0.jpg    train_batch451.jpg
events.out.tfevents.1700770415.def5c451d094.3015.0  train_batch1.jpg    train_batch452.jpg
results.csv                                     train_batch2.jpg    weights
results.png                                     train_batch450.jpg
```

```
%cd {HOME}
```

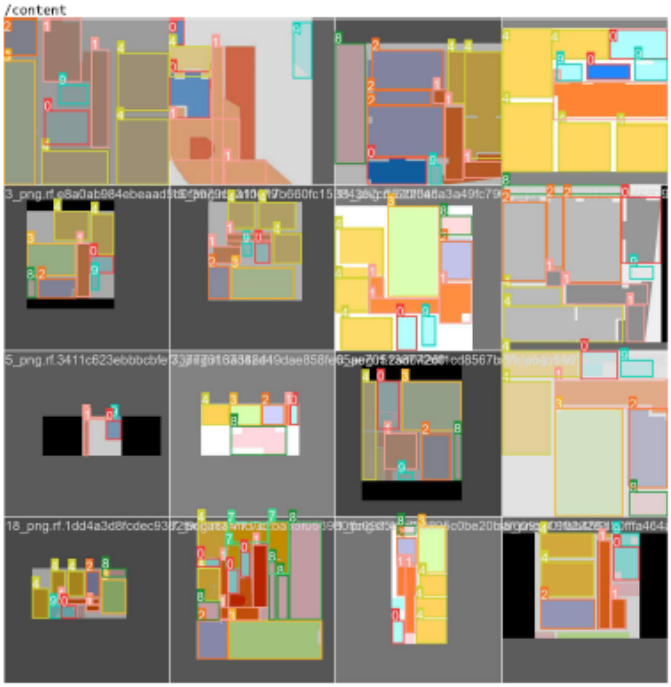
APPENDIX 8. Google Colab_ Yolov8_Python_Code


```
/content
```



```
%cd {HOME}
Image(filename=f'{HOME}/runs/detect/train/train_batch450.jpg', width=800)
```

```
/content
```



Validate Custom Model

```
%cd {HOME}

!yolo task=detect mode=val model={HOME}/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml

/content
2023-11-23 20:33:50.920106: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342] Unable to register cuDNN factory: Atte
2023-11-23 20:33:50.920158: E tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] Unable to register cuFFT factory: Attem
2023-11-23 20:33:50.920197: E tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518] Unable to register cuBLAS factory: Att
2023-11-23 20:33:52.209402: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
Ultralytics YOLOv8.0.28 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11129454 parameters, 0 gradients, 28.5 GFLOPs
val: Scanning /content/datasets/layout-names-4/valid/labels.cache... 9 images, 4 backgrounds, 0 corrupt: 100% 9/9 [00:00<?, ?it/s]
WARNING ⚠️ Box and segment counts should be equal, but got len(segments) = 4, len(boxes) = 48. To resolve this only boxes will be u
```

APPENDIX 9. Rule_Based Classification


```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import ipywidgets as widgets
from IPython.display import display

# Google Drive bağlantısını gerçekleştirme
from google.colab import drive
drive.mount('/content/drive')

# Excel dosyasının yolu
excel_file_path = '/content/drive/MyDrive/ANALİZ SONUÇLARI.xlsx'

# Excel dosyasını oku
df = pd.read_excel(excel_file_path)
```

```
Mounted at /content/drive
```

```
print(df.columns)
```

```
Index(['PlanID', 'Room', 'Bathroom ', 'Hall', 'Kitchen', 'Living_room',
       'Terrace', 'Wc', 'Special _room', 'Spaces_l_Common_Place',
       'Continous Circulation', 'Partially Circulation', 'Path',
       'PathBranches', 'Visibility', 'Enclosureness', 'Interlocking Spaces'],
      dtype='object')
```

```
class house_plan:
    def rule_01_room_based(row):
        if row['Room'] == 0 or row['Bathroom '] == 0 or row['Kitchen'] == 0:
            return "The plan is not a house plan"
        elif row['Room'] >= 1 and row['Bathroom '] > 0 and row['Kitchen'] >= 1 and row['Living_room'] >= 1:
            return "This plan could be a house plan/livable"
        else:
            return "NONE"
```

```
class circulation_path_relationship:
      def rule_02_circulation_based(row):
          if row['Continous Circulation'] == 1 and row['Partially Circulation'] == 0 and row['Path'] == 1 :
              return "Circulation movement is continuous"
          elif row['Continous Circulation'] == 0 and row['Partially Circulation'] > 0:
              return "Circulation movement is partially separated"
          elif row['PathBranches'] <= 42:
              return "This plan is suitable for accessibility."
          else:
              return "The plan does not exist"
```

```
class functional_diversity:
      def rule_03_functional_based(row):
          if row['Room'] >= 1 and row['Bathroom '] >= 1 and row['Kitchen'] >= 1 and ['Living_room'] >= 1 :
              return "This plan is a house plan"
          elif row['Room'] >= 1 and row['Bathroom '] >= 1 and row['Kitchen'] >= 1 and ['Living_room'] >= 1 and ['Terrace'] >= 1 :
              return "This plan could be a house plan/livable"
          elif row['Living_room'] >= 1 or row['Special _room'] >= 1 and row['Bathroom '] >= 1 or row['Wc '] >= 1 or row['Terrace'] >= 1
              return "This plan could be a different plan/but not for long term living"
          else:
              return "NONE"
```

```
class SpaceRelationship:
    @staticmethod
    def rule_04_spacerelation_based(row):
        if row['Hall'] >= 2 and row['Path'] == 1:
            return "This plan could be open house plan"
        elif row['Hall'] >= 1 and row['Bathroom '] >= 1 and row['Kitchen'] >= 1 and row['Living_room'] >= 1 and row['Spaces_l_Common_Pl
            return "The hall constitutes the main circulation connection."
        elif row['Hall'] >= 1 and row['Path'] == 1:
            return "This plan could be compact."
        else:
            return "NONE"
```

APPENDIX 9. (Continued) Rule_Based Classification


```
            return "This plan is NOT suitable for accessibility/The spaces are divided and have separate entrances."
        else:
            return "NONE"
    def rule_06_spacelayout_based(row):
        if row['Interlocking Spaces'] >= 1 and row['Enclosureness'] < 3:
            return "This plan could be unique."
        elif row['Interlocking Spaces'] == 0 and row['Enclosureness'] < 3:
            return "This plan could be unique."
        else:
            return "NONE"

class VisibleQuality:
    @staticmethod
    def rule_07_spacelayout_based(row):
        if row['Visibility'] >= 0.5 and row['Enclosureness'] <= 2:
            return "Space quality is high in visible conditions."
        elif row['Visibility'] < 0.5 and row['Enclosureness'] > 2 and row['Enclosureness']< 5:
            return "Space quality is low"
        elif row['Visibility'] < 0.5 and row['Enclosureness'] == 5:
            return "Space quality is very low"
        else:
            return "NONE"


# Kuralları uygula
df['Result_01'] = df.apply(house_plan.rule_01_room_based, axis=1)
df['Result_02'] = df.apply(circulation_path_relationship.rule_02_circulation_based, axis=1)
df['Result_03'] = df.apply(functional_diversity.rule_03_functional_based, axis=1)
df['Result_04'] = df.apply(SpaceRelationship.rule_04_spacerelation_based, axis=1)
df['Result_05'] = df.apply(space_layout_relationship.rule_05_spacelayout_based, axis=1)
df['Result_06'] = df.apply(space_layout_relationship.rule_06_spacelayout_based, axis=1)
df['Result_07'] = df.apply(VisibleQuality.rule_07_spacelayout_based, axis=1)


# Sınıflandırma sonuçlarını göster
print(df[['Result_01', 'Result_02', 'Result_03', 'Result_04', 'Result_05', 'Result_06', 'Result_07']])
```

```
                                    Result_01  \
0   This plan could be a house plan/livable
1              The plan is not a house plan
2              The plan is not a house plan
3   This plan could be a house plan/livable
4   This plan could be a house plan/livable
5   This plan could be a house plan/livable
6              The plan is not a house plan
7              The plan is not a house plan
8   This plan could be a house plan/livable
9   This plan could be a house plan/livable
10                                    NONE
11  This plan could be a house plan/livable
12             The plan is not a house plan
13  This plan could be a house plan/livable
14                                    NONE
15                                    NONE
16  This plan could be a house plan/livable
17             The plan is not a house plan
18                                    NONE
19  This plan could be a house plan/livable
20  This plan could be a house plan/livable
21                                    NONE
22                                    NONE
23  This plan could be a house plan/livable
24             The plan is not a house plan
25  This plan could be a house plan/livable
26             The plan is not a house plan
27             The plan is not a house plan
28             The plan is not a house plan
29             The plan is not a house plan
30             The plan is not a house plan
31  This plan could be a house plan/livable
32  This plan could be a house plan/livable
33  This plan could be a house plan/livable
34             The plan is not a house plan
35             The plan is not a house plan
36  This plan could be a house plan/livable
37                                    NONE
38             The plan is not a house plan
39             The plan is not a house plan
40  This plan could be a house plan/livable
41                                    NONE
42             The plan is not a house plan
43             The plan is not a house plan
44                                    NONE

                                   Result_02  \
0           Circulation movement is continuous
```

APPENDIX 9. (Continued) Rule_Based Classification

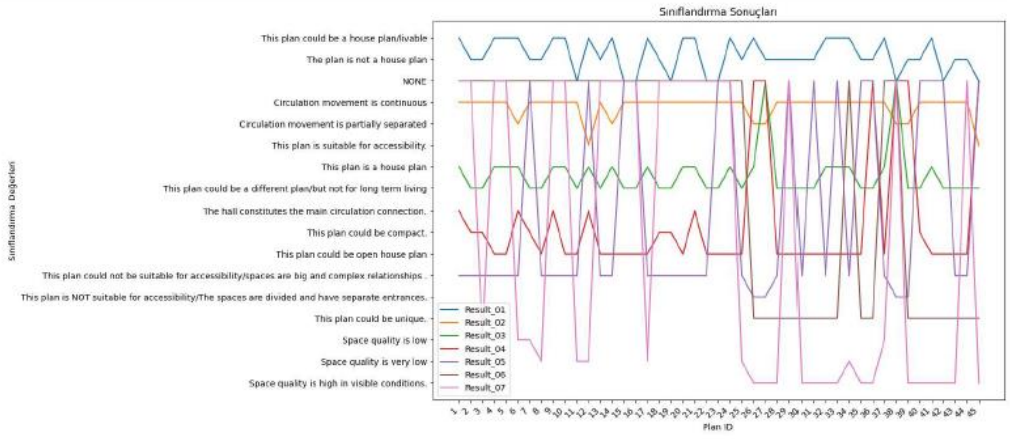
12,02,2024 12:05 class_rules - Colaboratory

```
1           Circulation movement is continuous
2           Circulation movement is continuous
3           Circulation movement is continuous
4           Circulation movement is continuous
5   Circulation movement is partially separated
6           Circulation movement is continuous
7           Circulation movement is continuous
8           Circulation movement is continuous
9           Circulation movement is continuous
```

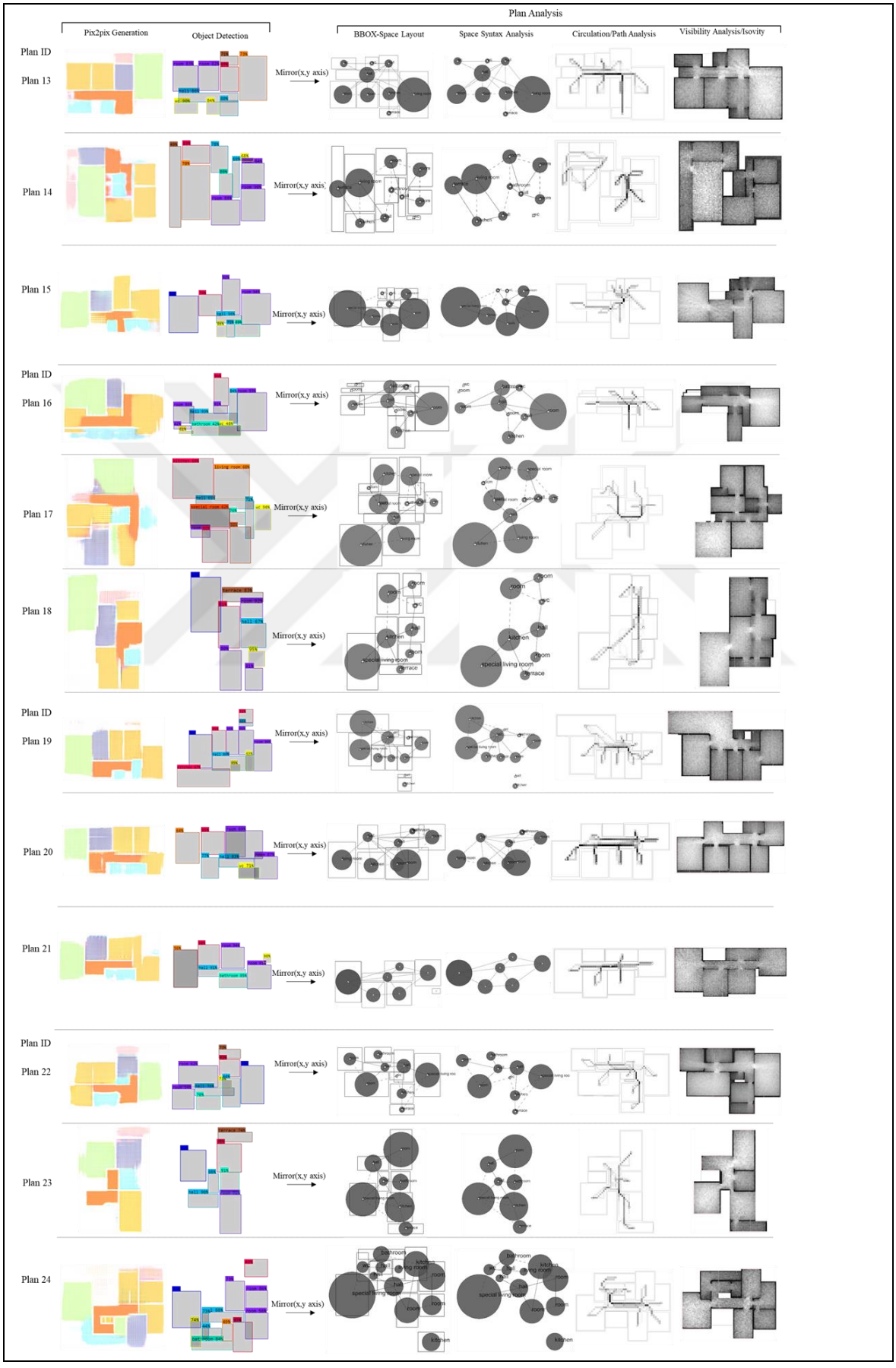
```
# Sınıflandırma sonuçlarını Excel dosyasına kaydet
df[['Result_01', 'Result_02', 'Result_03', 'Result_04', 'Result_05', 'Result_06', 'Result_07']].to_excel('/content/drive/MyDrive/Sinifl
```

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Sınıflandırma sonuçlarını al
classification_results = df[['Result_01', 'Result_02', 'Result_03', 'Result_04', 'Result_05', 'Result_06', 'Result_07']]

fig, ax = plt.subplots(figsize=(12, 8))
for column, classifications in classification_results.items():
    sns.lineplot(x=df.index, y=classifications, label=column)

ax.set_xticks(df.index)  # Tüm x değerlerini göster
ax.set_xticklabels(df['PlanID'], rotation=45, ha='right')  # PlanID'leri x ekseni etiketleri olarak kullan

plt.title('Sınıflandırma Sonuçları')
plt.xlabel('Plan ID')
plt.ylabel('Sınıflandırma Değerleri')
plt.legend()
plt.show()
```



APPENDIX 10. Pix2pix Generation, Object Detection and Analysis Results



Gazili olmak ayrıcalıktır