UNIVERSITETI POLIS

ICCSM 2025

BOOK OF PROCEEDINGS

1ST INTERNATIONAL CONFERENCE
COMPUTER SCIENCES AND MANAGEMENT

WHERE DIGITAL & BUSINESS BECOME HUMAN

26-27 June 2025 | Tirana, Albania

ISBN 9789928347123

DOI 10.37199/c41000300

CONFERENCE CHAIR

Assoc. Prof. Merita Toska, POLIS University

PARTNER UNIVERSITIES

POLIS University, Albania
Université Lyon 2, France
Università Telematica San Raffaele, Italy
University of Western Macedonia, Greece
International University of Sarajevo, Bosnia & Herzegovina
Mother Teresa University, North Macedonia
Gebze Technical University, Turkey
Public International Business College, Kosovo
Rochester Institute of Technology – RIT Global Campus, Kosovo
Co-PLAN, Institute for Habitat Development, Albania
AI Hub Albania, Albania
Luralux, Albania

ORGANISING COMMITTEE

Dr. Blerta Mjeda
Dr. Emiliano Mankolli
Msc. Sonila Murataj
Msc. Andia Vllamasi
Msc. Klejda Hallaci
Msc. Erilda Muka
Msc. Armela Reka

SCIENTIFIC COMMITTEE

Prof. Dr. Jérôme Darmont, Université Lumière Lyon 2 (France)
Prof. Dr. Lydia Coudroy de Lille, Université Lumière Lyon 2 (France)
Prof. Dr. Jim Walker, Université Lumière Lyon 2 (France)
Prof. Dr. Besnik Aliaj, POLIS University, (Albania)
Prof. Dr. Daniela Sica, San Raffaele Roma University, (Italy)
Prof. Dr. Stefania Supino, San Raffaele Roma University, (Italy)
Prof. Dr. Arbana Kadriu, South East European University (North Macedonia)
Prof. Dr. Ing. Lejla Abazi-Bexheti, South East European University (North Macedonia)
Prof. Dr. Yusuf Sinan Akgül, Gebze Technical University (Turkey)
Assoc. Prof. Dr. Galia Marinova, Technical University of Sofia (Bulgaria)
Assoc. Prof. Dr. Vasil Guliashki, Technical University of Sofia (Bulgaria)
Assoc. Prof. Mehmet Göktürk, Gebze Technical University (Turkey)
Assoc. Prof. Yakup Genç, Gebze Technical University (Turkey)
Assoc. Prof. Habil Kalkan, Gebze Technical University (Turkey)
Assoc. Prof. Dr. Godiva Rëmbeci, POLIS University (Albania)
Assoc. Prof. Dr. Xhimi Hysa, POLIS University (Albania)
Assoc. Prof. Dr. Merita Toska, POLIS University (Albania)
Assoc. Prof. Dr. Sotir Dhamo, POLIS University (Albania)
Dr. Gennaro Maione, San Raffaele Roma University, (Italy)
Dr. Nicola Capolupo, San Raffaele Roma University, (Italy)
Dr. Benedetta Esposito, San Raffaele Roma University, (Italy)
Dr. Venera Demukaj, Rochester Institute of Technology (Kosovo)
Dr. Emil Knezović, International University of Sarajevo (BiH)
Dr. Šejma Aydin, International University of Sarajevo (BiH)
Dr. Azra Bičo, International University of Sarajevo (BiH)
Dr. Šejma Aydin, International University of Sarajevo (BiH)
Dr. Azra Bičo, International University of Sarajevo (BiH)
Dr. Hamza Smajić, International University of Sarajevo (BiH)
Dr. Panagiotis Kyratsis, University of Western Macedonia (Greece)
Dr. Delina Ibrahimaj, Minister of State for Entrepreneurship and Business Climate (Albania)
Dr. Elona Karafili, POLIS University (Albania)
Dr. Emi Hoxholli, POLIS University (Albania)
Dr. Shefqet Suparaku, POLIS University (Albania)
Dr. Manjola Hoxha, POLIS University (Albania)
Dr. Elsa Toska, POLIS University (Albania)
Dr. Emiliano Mankolli, POLIS University (Albania)

Dr. Albina Toçilla, POLIS University (Albania)
Dr. Sonia Jojic, POLIS University (Albania)
Dr. Ilda Rusi, POLIS University (Albania)
Dr. Ledian Bregasi, POLIS University (Albania)
Dr. Klodjan Xhexhi, POLIS University (Albania)
Dr. Endri Duro, POLIS University (Albania)
Dr. Remijon Pronja, POLIS University (Albania)
Dr. Vjosë Latifi, International Business Collage Mitrovica (Kosovo)
Dr. Agron Hajdari, International Business Collage Mitrovica (Kosovo)

Table of Contents

20

AI IMAGE GENERATION AND ITS POSSIBLE CONTRIBUTIONS IN ARCHITECTURAL LANGUAGE

DOI: 10.37199/c41000320

Andia VLLAMASI
POLIS University (Tirana, Albania)
andia_vllamasi@universitetipolis.edu.al
ORCID 0009-0007-1730-4130

Skender LUARASI
POLIS University (Tirana, Albania)
skender_luarasi@universitetipolis.edu.al
ORCID 0000-0002-0967-2482

Tamara LUARASI
POLIS University (Tirana, Albania)
tamara_luarasi@universitetipolis.edu.al
ORCID 0009-0002-7449-5491

Abstract

AI, and specifically Machine Learning methodologies, can help the architect's imagination in the design process or the urban planner to develop ideas about urban planning. This paper focuses on generative ML methodologies that generate images from input datasets. The machine is trained to create patterns that help generate new images.

In all of these methodologies, the Generative Adversarial Networks (GANs) model is used, with specific details for each case. The idea is to highlight their specifics, both in concept and in implementation, and to test different metrics for assessing the accuracy of the generative process.

The input datasets are facade images, their sketches, or the combinations of both, and as a result, new images can be generated. The machine learning techniques are used to help us interpret architectural historical concepts, such as the relationship between the natural and the customary. Traditionally, the natural is represented by geometry, while the customary has stood for inherited stylistic languages.

Keywords*:* Design, pattern, generative, algorithm, architectural

I. INTRODUCTION

The development of generative artificial intelligence technologies has opened new horizons in the way architectural forms are conceived and presented (Li, 2024). Formal generation is being revolutionised by the generation of realistic images using sophisticated models like Generative Adversarial Networks (GANs) and other machine learning approaches. The AI-generated images undoubtedly daunt one. By plugging a bunch of natural keywords in the command prompt of AI models like Midjourney, DALL-E, or Stable Diffusion, a myriad of otherworldly images emerge as if out of thin air. The AI-generated images are outputs of neural algorithms "diving" deep into large datasets of 'observations' and intelligently reading and synthesising them (Chaillou, 2022, p. 65). This approach not only enables the formal analysis of complex visual data and the visualisation of projects before physical realisation, but also introduces new, previously unthought-of or unimagined images. This has led both scientists and AI apologists to claim that the latter can generate semantic content 'entirely autonomously' from humans (del Campo, 2022, p. 142). The question, of course, is how we - the designers, in general, can make sense of such output.

AI describes the statistical rather than the epistemological being of a phenomenon. It 'learns' to recognise images by categorising and labelling pixels, or regions in an image, into several groups' (Parente et al, 2023, p. 2). A large amount of information is processed through a series of computational layers comprised of 'neural' parameters or 'dials,' each of which 'filters' the input and 'feeds' it the following layer. Changing these dials will change the probability that the network model gives for the next informational pattern on a given input (IBM Think Topics). Unlike parametric modelling, where the user formulates the parameters, in AI it is the model itself that 'learns' them, and the user guides the general direction of learning through 'high-level settings' or 'hyperparameters' (Cahillou, 2022, p. 65). Out of this 'semantic training' AI 'learns' not only to recognize but also predict and generate new images.

Matias Del Campo has likened AI's ability to learn patterns to the architect's analysis of 'a building's style. The architect would break down the building into its basic components, looking for features such as columns, arches, curves, colours, materials that are associated with specific time periods, school of thoughts, or that can differentiate the image content in a *unique* and *meaningful way*' (del Campo, 2022, 72). Del Campo's claim, however, conflates terms that, historically, have stood for different categories. For example, the curve, as a geometrical entity, would fall under the 'natural,' while the columns and arches, as semantic or stylistic components, would fall under the 'customary.' The aim of such categorization was precisely to match and calibrate formal and content, generative epistemic systems (which are always technological) with meaning.
What is critical in such generative large language models, however, is precisely the very im-possibility of such a distinction. AI tends to erase such a distinction by being used like a black box

that does not account for the architect's intentions and overflows the latter with infinite patterns. This paper aims to use machine learning in such a way as to revitalize and reconceptualize the distinction between form and content, syntax and semantics, the natural and customary. By analysing three GAN extensions, this paper aims to explore the potential of AI in creating visual images of facades of buildings through a generator, on the one hand, and then in matching these images with geometrical sketches through a pixel-to pixel algorithm.

This material focuses on three generative machine learning methodologies that have as their objective the generation of images based on a dataset that is used to train the machine learning algorithm. The machine is trained, creating patterns that help in the generation of new images.

These methodologies consist of:

- Generating new views as a result of training the algorithm with facade views and their details
- Generating a facade view or producing an object detail as a result of training machine learning algorithm with real sketch-photo pairs
- Generating new views similar to those that the machine is trained. In this case algorithm is trained on facade views that are classified and this classification helps in the generation process.

In all three of these methodologies, the Generative Adversarial Networks (GANs) model is used with some specifics for each case. The idea is simply to highlight their specifics, in concept and implementation, and to test some different metrics in assessing the accuracy of the generative process.

II. LITERATURE REVIEW

II.1 Related Work

A Generative Adversarial Network, or GAN, was designed by Ian Goodfellow and his colleagues in 2014, and is a machine learning approach to generative modelling that has served as a basis for many other generative modelling approaches. The original GAN is often called a Vanilla GAN. Its two main components are neural networks: a generator G and a discriminator D. D takes images generated by G as input, along with real images, and is trained to distinguish the generated images from the real ones. G is trained to synthesise images as close to reality as possible. (Goodfellow, 2014)

While the GAN model has a wide range of applications, other generation models are more focused on image generation, which is directly related to the architectural design process. The following models are as follows.

The Progressive Growing GAN model is an extension of the GAN model adapted for large and high-quality images. Its idea is to add new generator and discriminator layers during the training process, going from a low resolution to a higher one and thus accelerating the execution time. (Karras, 2018)

An alternative of GAN model is proposed Kaneko and Harada (2020), which is called noise robust GANs (NR-GANs). In this alternative, the generator is trained with a noise input in addition to training a generator with a clean image input, which solves the problem when there is a lack of information in the input noise, such as the type of distribution, amount of noise, or noise-signal relationship.

While the GAN model uses a pair of images, one real and one synthesized to perform image generation, the CycleGAN (Zhu, 2017) model does not use a pair of images, which is often difficult to obtain. For this, this model uses two generators that pass, the first, an image from one domain to another and the second, returns the transformed image back by comparing it with the first in an attempt to get it as close as possible to the first. A model of a robust noise-generation generative adversarial network (NG-GAN) is proposed by Hossain and Lee (2023). The idea of the proposed model is how to deal with old images, how to obtain noise distribution of the degraded old images inspired by the CycleGAN model.

Deep Convolutional Generative Adversarial Network (DCGAN) were introduced by Alec Radford & Luke Metz & Soumith Chintala (2016) , where an architectural topology of Convolutional GANs were proposed. It has been utilized by artists and designers to create unique and visually captivating artworks, logos, and graphics.

Conditional GAN (cGAN) allows us to condition the network with additional information such as class labels (Mirza, 2014). It means that beside the images, some associative labels are the inputs of this model (facades with roof, facades without roofs).

In their study, Isola et al. (2017) presented Image-to-Image Translation with Conditional Adversarial Networks (PixToPix). This model It is a special case of the conditional GAN where the label is image. The two networks learn the mapping of two images, but also a loss function to train this mapping.

Based on a study, Convolutional Generative Adversarial Network (CGAN-Pix2pix) is the method used to provide a quality assessment of dehazing images.

II.2 GAN Overview

There are two models included in the proposed GAN model, that cooperate and oppose each other, and each is a so-called a multilayer perceptron. (Goodfellow, 2014)

Multilayer perceptron (MLP) are artificial neural networks (X. Li, 2021). Such networks consist of several fully connected layers that transform input data from one dimension to another. There is an input layer, one or more hidden layers, and the output layer.

The figure below illustrates such a network, where each node (or neuron) in the input layer represents a feature of the input data, in our case of an image, and has a full connection between a neuron of one layer and all neurons of the previous layer.

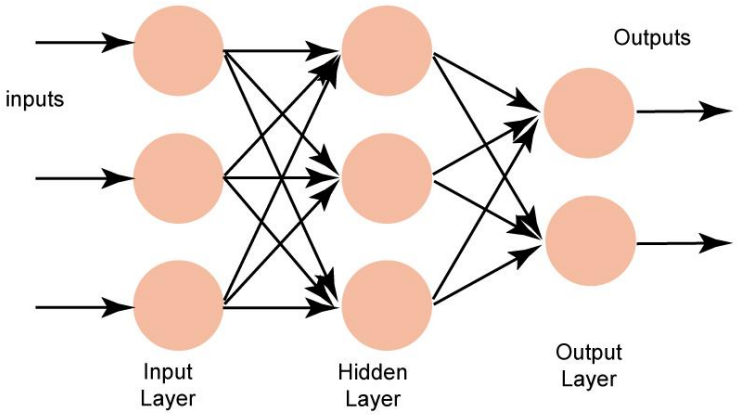


Figure 1. Fully connected neural network
Source: Authors processing

It is important to mention some key point in the paper of the authors. Many authors have used and commented on this paper. Between them in this work (Arjovsky, 2017) is a detailed interpretation of these key points is presented. It is necessary to mention these key points, to have a better understanding of the implementation of GAN model.

According to the authors of the GAN model (Goodfellow, 2014), the GAN model consists of two multi-layers neural networks: Discriminator and Generator.

As input data for the generator, are used some fixed-length vectors of randomly selected values representing the qualities of an image, referred to as noise, with values chosen within the possible bounds that suggest concrete data for their qualities, or sampling these values from a probabilistic distribution of the qualities of the data that is taken a priori. As a result of this input, the generator produces an image that is called a fake image.: noise(z), parameters(g)-> image: $G(z, param_g)$, and this image that will serve as an input for the discriminator.

The discriminator, on the other hand, is also a multi-layer neural network $D(x, param_d)$ that takes as input real images and those generated by the generator, and produces a single numerical output that is the labelling of these images with predicted labels.

The labelling of the images, coming from real data, should be a probability $D(x)$ as close to 1 as possible, and the labeling of a generated image $G(z)$ should be a probability as small as possible D(G(z)), which translates into a value as high as possible of $1 - D(G(z))$. Therefore, the goal of the

discriminator is to provide a value as high as possible to $D(x) * (1 - D(G(z)))$ or to the expression $log(D(x) + log(1 - D(G(z)))$.

In order for the discriminator to become capable of this, it needs to be trained, which means improving (or learning) the parameters of the network. Thus, we have an autonomous training of the discriminator that is carried out through the backpropagation process, where in each step of this process we have an improvement of the network parameters that results in increasing the discriminator's ability to avoid incorrect evaluations.

Backpropagation is conditioned by the mistakes that the discriminator makes and they are calculated. There are two types of errors, discriminator loss and generator loss. The first is related to the error against real images, so the image is real and its labelling is close to 0, while the second is related to the error against generated images, meaning the image is generated and its labelling is close to 1. The sum of these errors, which is based on the differences between the expected labels and those that occur, forces the discriminator to repeat the process from the beginning now with modified parameters.

On the other hand, the generator is interested in generating images as close to reality as possible, thus minimizing the expression $D(G(z)\ or\ maximizing\ (1 - D\big(G(z)\big)\ or\ log(1 - D\big(G(z)\big)$, and this also requires training of the generator, which is carried out through a backpropagation process, a process that modifies the parameters of the neural network of the generator. Everything said is illustrated with the figure 2.

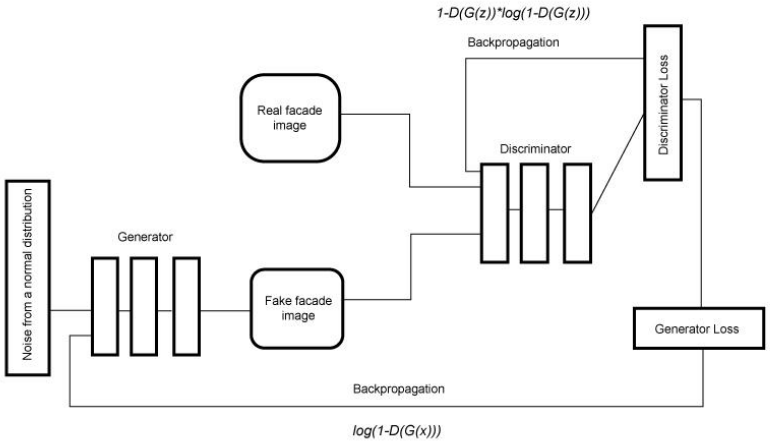


Figure 2. Generative Adversarial Networks (GANs) Model
Source: Authors processing

The achievement of these two objectives by the discriminator and generator has been expressed by the authors finding of:

$$V(D,G) = E_x(x)[logD(x)] + E_z\left[log\ log\ \left(1 - D\big(G(z)\big)\right)\right] \quad (1)$$

Where$E_x, dhe\ E_z$ represent the mathematical expectations of the respective expressions.

According to the author (Goodfellow) of the model, the modifications of the parameters of the discriminator and the generator are made through stochastic gradient, which is an iterative method of modifying the parameters like it is represented in the Figure 3.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 3. Algorithm of the iterative modifications of Discriminator and Generator
Source: Authors processing

In this figure an iterative independent cycle of parameter modifications of the discriminator is observed, and this cycle is included in another cycle where the generator is modified.
For fixed G, the $V(D,G)$ maximum will be achieved when
$D^*(x) = realdataDistribution(x) / [realdataDistribution(x) + generateddataDistribution(x)]$ *(2)*
The global minimum in (1) is achieved when the total error calculated (when the real images are predicted with low probabilities and when generated images are predicted with high probabilities) gets its smallest value.
To measure this error it is used **Jensen-Shannon (JS) divergence**, which measures the differences between two distributions of real data and generated data.
Jensen-Shannon (JS) Divergence Formula is:

$$JS(D||G) = 1/2 * KL(P||M) + 1/2 * KL(Q||M) \ (3)$$

where M is the average distribution:

$$M = (D + G)/2$$

and the **Kullback-Leibler (KL) divergence** is:

$$KL(D||G) = \sum D(x)\, log[D(x)/G(x)]$$

D(x) is the distribution of the predictions of the real images and G(x) the distribution of predictions of the generated images. Using this method, it is concluded that the global minimum is achieved when two distributions are identical.

III. METODOLOGY AND DATA

III.1 Implementation of the GAN model

III.1.1 The internal structure of the generator and discriminator and its implementation

A fully connected neural network is composed by some layers, where each layer is the input of the next layer. A layer is composed by neurons and every neuron of a layer is connected with each neuron of the its input layer like the figure 1 shows.

Each neuron is in fact a non-linear function, called perceptron neuron as well, and performs two steps:

- Finds the sum of the outputs of previous layer multiplied by the weights of previous layer, that can be presented as: $z_j = \sum_i \quad w_i \cdot x_i + b_j$
- Applies the activation function, which is a non-linear function:

The Figure 4 shows the layer created after the input layer by these two steps.

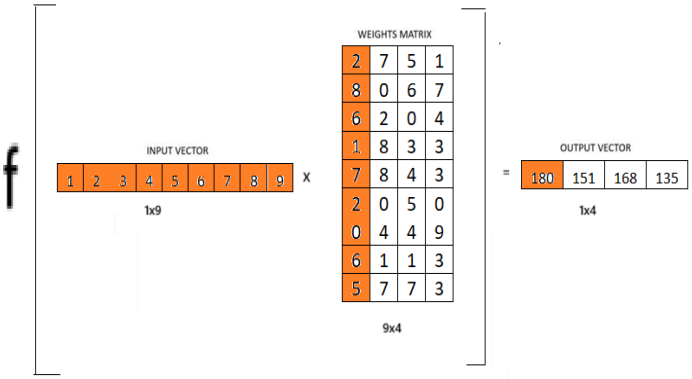


Figure 4. The steps of perceptron neuron (reprinted from *Fully Connected Layer vs. Convolutional Layer: Explained*, Built In, n.d., https://built-in.com/…).

Source: Author's processing

Where the activation function f can be one of the functions: Sigmoid, Tanh, ReLU (Rectified Linear Unit) function, etc. (Shiv Ram Dubey, 2021). The implementation code of a fully connected layer depends on the Python framework. For all the examples below, it is selected the TensorFlow framework of python and Keras. Layers library on it. In this framework the placeholders of the data are so called tensors. “In TensorFlow, tensors are the basic building blocks used to represent data.

A tensor can be thought of as a multi-dimensional array, similar to a matrix but with an arbitrary number of dimensions. Tensors can hold various data types, including integers, floating-point numbers, and strings." (Bader, 2009)

The implementation of fully connected layer for the GAN model would be:

Create a dense layer with 128 units

layer = tf.keras.layers.Dense(units=128, activation='relu'), and the generator and discriminator would be composed of some lines of this type.

Because there are generation models more focused on image generation than GAN model, some implementation details are provided related to some specifics of them in the following sections.

III.1.2 The implementation of GAN model extensions related to some specifics

The common characteristic of these modes is the use of convolutional layers in their model. In difference with the fully connected layer, in a convolutional layer, each neuron of a layer applies a convolutional operation to the input layer. The neuron uses a sliding window (called kernel), and represents the space occupied by this window in the input layer as a sum of a dot product between the values in the input layer and values in the window. In this type of layer (convolutional one), a neuron is not connected with all the neurons of the input layer. Figure 5 represent a convolutional operation of a neuron. The neuron with value 48, uses the dot multiplication of the space in orange colour of the input layer and the kernel to get this value. Output is called a feature map too.

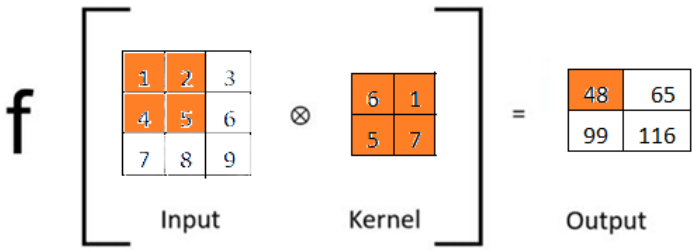


Figure 5. Convolutional operation (adapted from *Conditional Generative Adversarial Network*

Source:https://www.geeksforgeeks.org/conditional-generative-adversarial-network/).

Some key moments will be mentioned for the models DCGAN, CGAN and PixToPix:

- Model structure: Discriminator and Generator
- Definition of the discriminator loss and generator loss
- Training process of discriminator and generator

III.2 DCGAN model structure

DCGAN model generates new images. The generator starts with a noise, and the discriminator compares the generated image from the noise by the generator, and a real image. The process progresses like in GAN model and the result will be some new images.

- *Discriminator* structure in tensorflow keras.layers:

```
#inputs are colored images
def discriminator
 image = tf.keras.layers.Input(shape=(img_size, img_size, 3)) # [(None, 32, 32, 3)]
 i_output = tf.keras.layers.Conv2D(128,(3,3 ),strides=(2, 2), padding='same')(image)
                                        # (None, 16,18,128)
 i_output = tf.keras.layers.LeakyReLU(alpha=0.2)( i_output)   # (None, 16,18,128)
 i_output = tf.keras.layers.Conv2D(128, (3, 3),strides=(2,
2),padding='same')(i_output)
                                        # (None, 8,8,128)
 i_output = tf.keras.layers.LeakyReLU(alpha=0.2)(i_output)    # (None, 8,8,128)
 i_output = tf.keras.layers.Flatten()(i_output)            # (None, 1,8192)
 i_output = tf.keras.layers.Dropout(0.4)(i_output)          # (None,1,8192)
 i_output = tf.keras.layers.Dense(1, activation='sigmoid')(i_output)# (None,1)
 model = Model(image, i_output)
 return model
d_model = build_discriminator()
d_model.summary()
```

Some notes about discriminator structure:

1. Input (shape= (img_size, img_size, 3): creates the tensor for the image with dimensions [None, 32, 32, 3] (here the image_size=32), None represent the size of data
2. Conv2D (128, (3,3), strides= (2, 2), padding='same') (image):
convolutional layer creates 128 features maps with dimensions (16, 16), or an output of dimensions (None, 16, 16, 128); the formula is applied

$$O = \lfloor \frac{n - f + 2 \cdot p}{s} \rfloor + 1$$

where, O: Output size, n: Input size (height or width) =32, f: kernel size =3, p: Padding =1, s: Stride =2

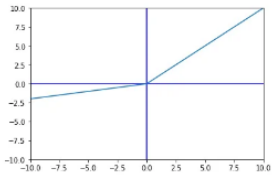
The second Conv2 produces the output of (None, 8, 8,128), based on the same formula, but now n: Input size (height or width) =16

3. LeakyReLU(alpha=0.2) (i_output)

It is applied the formula to avoid linearity

f(x) = alpha * x if x < 0

f(x) = x if x >= 0



4. Flatten () (i_output)

The feature maps are converted into a 1D array (1, 8192) where 8192=8x8x128

5. Dropout (0.4) (i_output)

The Dropout layer with rate=0.4 ensures that 40% of the neurons in the preceding layer are randomly deactivated during training to enhance the robustness of neural network model.

6. Dense (1, activation='sigmoid') (i_output)

This line provides one dimensional tensor and the activation function is applied. The mathematical formula for a sigmoid function is given by: f(x) = 1 / (1 + exp(-x)). It transforms numbers to values typically between 0 and 1.

- *Generator* structure in tensorflow keras.layers:

```
#Inputs are noises defined randomly from a normal distribution
        noise = tf.keras.layers.Input(shape=(noise_dim,))   # [(None, 100)], noise_dim=100
        n_nodes = 128 * 8 * 8
        g_output = tf.keras.layers.Dense(n_nodes)(noise)     # (None, 8192), 8192=128x8x8
        g_output = tf.keras.layers.LeakyReLU(alpha=0.2)( g_output)# (None, 8192)
        g_output = tf.keras.layers.Reshape((8, 8, 128))( g_output)   #(None, 8, 8, 128)
        g_output = tf.keras.layers.Conv2DTranspose(128,4,4),strides=(2,2),padding='same')
                                        (g_output) # (None, 16, 16, 128)
        g_output = tf.keras.layers.LeakyReLU(alpha=0.2)(gen)       #(None, 16, 16, 128)
        g_output = tf.keras.layers.Conv2DTranspose(128,(4,4),strides=(2,2), padding='same')
                                        (g_output) #(None, 32, 32, 128)
        n_output = tf.keras.layers.LeakyReLU(alpha=0.2)( g_output)  #(None, 32, 32, 128)
        g_output = tf.keras.layers.Conv2D(3,(8,8),activation='tanh',padding='same')(g_output)
                                            # (None, 32, 32, 3)
        model = Model (noise, g_output)
```

Some notes about generator structure:

1. Input(shape=(noise_dim,)

   creates the tensor for a fixed vector with size noise_dim (here noise_dim=100)

2. n_nodes = 128 * 8 * 8

   defines the dimension of the generated image at the starting point

3. Dense(n_nodes)(noise) provides one dimensional tensor for the noise

4. Reshape ((8, 8, 128))
   Reshape the tensor in correspondence with the tensors used by discriminator
5. Conv2DTranspose (128,4,4), strides= (2,2), padding='same')(l_output)

The need for transposed convolutions generally arises from the need to use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution. (Odena, 2016) he following formula calculates the output dimension in this layer based on some parameters mentioned below: (Contributors, In PyTorch 2.7. , 2025)

$$n_{out} = (n_{in} - 1)s - 2p + (f - 1) + p + 1 = (n_{in} - 1)s - p + f$$

where

- n_{out} *is the output width or height* $(same\ in\ both\ dimensions)$
- n_{in} *is the input width or height* $(same\ in\ both\ dimensions)$
- s *is the stride* $(same\ in\ both\ dimensions)$
- p *is the padding* $(same\ in\ both\ dimensions)$
- f *is the filter dimensions* $(same\ in\ both\ dimensions)$

Padding is calculated: (Liu, 2018)

$$if\ \ n_{in}\%s == 0$$
$$p = max(f - s, 0)$$
$$else$$
$$p=max(f-(n_{in}\%S),0$$

for the above case:

$$n_{in}\%S=8\%2=0 \quad then\ p=4-2=2$$
$$And\ \ n_{out} = (n_{in} - 1)s - p + f = (8 - 1) * 2 - 2 + 4 = 16$$

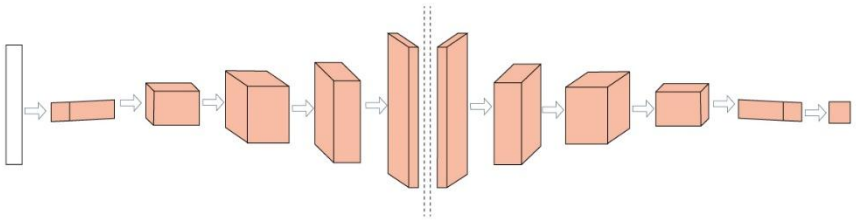


Figure 6. Illustration of GAN structure
Source: Authors processing

III.3 Definition of the discriminator loss and generator loss

An important element of the GAN model is the error calculation during the training process. These errors dictate the updating of the parameters in each step and backpropagation of this process.

- *JS Divergence*

As it is mentioned above this approach finds an "average" distribution between distribution of the discriminator predictions and generator prediction, and uses this distribution as a reference distribution against the two first, finding the differences with them like it is shown in the figure.

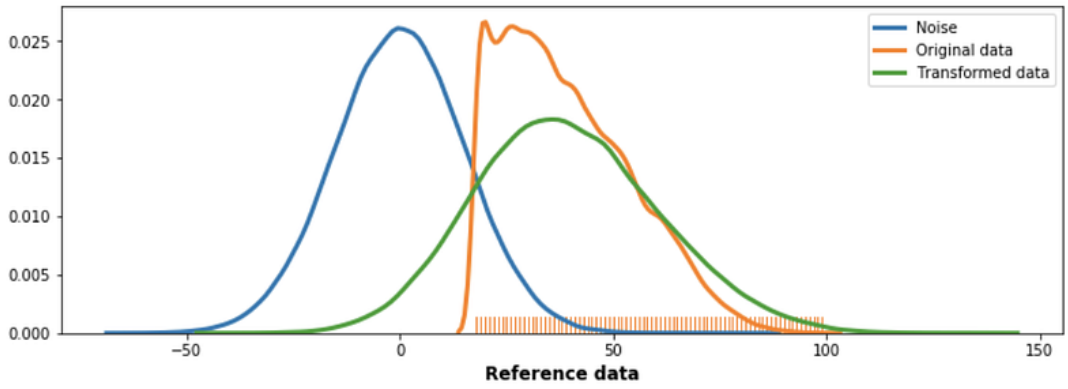


Figure 7. Jensen–Shannon divergence - Search Images
Source: Authors processing

The implementation of this approach is:

```
def jensen_shannon_divergence(d, g):
    # The values outside (0,1] are eliminated
    d = tf.clip_by_value(p, 1e-10, 1.0)
    g = tf.clip_by_value(q, 1e-10, 1.0)
    # the "average" distribution
    m = 0.5 * (d + g)
    # Compute the KL divergence for each part
    kl_dm = tf.reduce_sum(p * tf.math.log(d / m), axis=-1)
    kl_gm = tf.reduce_sum(q * tf.math.log(g / m), axis=-1)

    # JS divergence as average of the two Kullback-Leibler divergences
    js_divergence = 0.5 * (kl_dm + kl_gm)

    return js_divergence

def discriminator_loss_js(real, fake):
    real_loss = jensen_shannon_divergence(tf.ones_like(real), real)
    fake_loss = jensen_shannon_divergence(tf.zeros_like(fake), fake)
    total_loss = real_loss + fake_loss
    return total_loss
```

```
def generator_loss_js(preds):
  return jensen_shannon_divergence(tf.ones_like(preds), preds)
```

- *Binary cross-entropy*

It is considered the binary cross-entropy approach too for the error calculation, which calculates the error by the formula as follows:

$$BCE = -\frac{1}{N}\sum_{i=1}^{n} [y_i log(p_i) + (1 - y_i) log(1 - p_i)] \quad (4)$$

where: N is the data size, y_i is the actual binary label (0 or 1) of the i^{th} real image, p_i is the predicted probability of the i^{th} real images. It quantifies the uncertainty of the prediction distribution measuring the average of the differences between the real labelling of the images and the predicted probabilities for these images to be real or fake. (Goodfellow I. B., 2016).

The implementation of this formula in Python is:

```
bce_loss = tf.keras.losses.BinaryCrossentropy()
def discriminator_loss(real, fake): #real->labeling of a real image,
                        #fake->labeling of a generated image
  real_loss = bce_loss(tf.ones_like(real), real)
                #error against real image
  fake_loss = bce_loss(tf.zeros_like(fake), fake)
                #error against generated image
  total_loss = real_loss + fake_loss #sum of both errors
  return total_loss

def generator_loss(preds):
  return bce_loss(tf.ones_like(preds), preds)
            #sum of differences (1-prediction)
```

This formula is based on this logic:

We have errors in discriminator if it labels with 0 the real images, and with 1 the generated images. Then the expression $p_i^{y_i} * (1 - p_i)^{1-y_i}$ represents how close or far are the predictions from the correct labeling.

Truly,

If an image is real, $y_i = 1$, the expression $p_i^{y_i} * (1 - p_i)^{1-y_i} = p_i$

If an image is generated, $y_i = 0$, the expression $p_i^{y_i} * (1 - p_i)^{1-y_i} = (1 - p_i)$

Instead of the above expression the logarithm of it with negative sign is considered:

$$-(y_i log(p_i) + (1 - y_i) log(1 - p_i))$$

Therefore:

If an image is real, $y_i = 1$, the expression $-(1 * log(p_i) + (1 - 1) log(1 - p_i)) = -log(p_i)$

If an image is generated, $y_i = 0$, the expression $-(0 * log(p_i) + (1 - 0) log(1 - p_i)) = -log(1 - p_i)$

In both cases if the prediction is far from the correct label (close to 0 for y=1 and close to 1 for y=0), the value of the expression is high and when the prediction is close to the correct label the value of the expression is low. The figure 5 represent the graph in both cases.

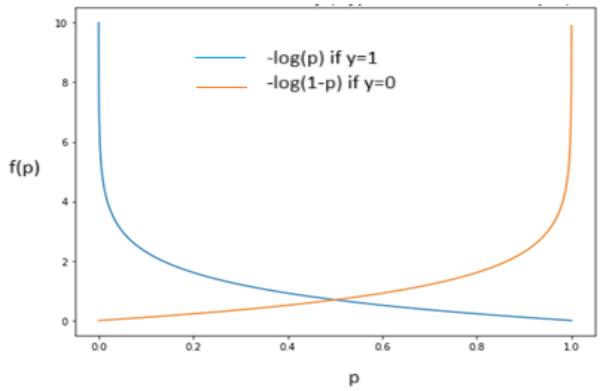


Figure 8. the Graph of $-(y_i log(p_i) + (1 - y_i) log(1 - p_i))$ for y=0 and y=1, the blue line represents the distribution of the real images and the orange line represents the distribution of the generated images, the optimal value is achieved when both distributions are equal. Source: Binary cross-entropy graph - Search Images

The value of the expression (4), which in fact represents the average of the errors for each image, dictates the backpropagation of the process, firstly for the discriminator and after for the generator to achieve a discriminator loss and generator loss as small as possible until the optimal value for the prediction is achieved, as shown in Figure 5.

Training process of the discriminator and generator code:

```
def train_step(dataset):
  real_images= dataset #real images
  # Sample random noise represented by a vector.
  random_vectors = tf.random.normal(shape=(batch_size, noise_dim))
  generated_images = g_model(random_vectors)

  # Train the discriminator.
  with tf.GradientTape() as tape:
    pred_fake = d_model(generated_images)
    pred_real = d_model(real_images)
    d_loss = discriminator_loss(pred_real, pred_fake)

  #gradient approach to modify the parameters(trainable_variables)
  grads = tape.gradient(d_loss, d_model.trainable_variables)
  d_optimizer.apply_gradients(zip(grads, d_model.trainable_variables))
```

```
# Sample random noise.
random_vectors = tf.random.normal(shape=(batch_size, noise_dim))

# Train the generator
with tf.GradientTape() as tape:
   fake_images = g_model(random_vectors)
   predictions = d_model(fake_images)
   g_loss = generator_loss(predictions)

#gradient approach to modify the parameters(trainable_variables)
grads = tape.gradient(g_loss, g_model.trainable_variables)
g_optimizer.apply_gradients(zip(grads, g_model.trainable_variables))

return d_loss, g_loss
```

III.4 Conditional GAN (CGAN) and the differences from the GAN model

The GAN model described above it is named as unconditioned GAN, because there is not control over the data generated. In conditional GANs, additional information is provided, for example in preliminarily the images provided could be associated by some labels.

Often the input data are images that can be preliminary labelled, and this helps in the generation process in the speed of it and in the quality of the image generated, when it is needed the generated images to keep their categorization.

The figure 9 represent such a model. The images are labelled.

The formula (1) based on the first definition of Conditional GANs by Mirza and Osindero (Osindero, 2014) can be presented as:

$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x|y)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z|y)))] \quad (5)$$

As this research describes this model, in GAN model the data are generated randomly from a distribution chosen a priory, but in CGAN the generator creates images by considering *specific conditions.*

In the example considered in this paper, as dataset is considered a collection of 400 facades and for the GAN model they are used without any label. For the CGAN model the facades are classified in two classes: facades with roofs and facades without roofs.

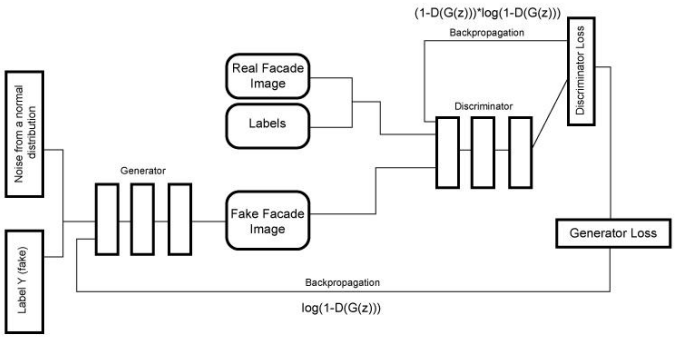


Figure 9. The CGAN model
Source: Authors processing

This difference takes place discriminator model and generator model, where ae add some lines to include labels. To keep everything the same lake for the GAN model:

- *In discriminator:*

```
# label input
label = tf.keras.layers.Input(shape=(1,))
#create a vector of size 50
l_output = tf.keras.layers.Embedding(n_class, 50)(label) #n_class=2
n_nodes = img_size * img_size
l_output = tf.keras.layers.Dense(n_nodes)( l_output)
l_output = tf.keras.layers.Reshape((img_size, img_size, 1))( l_output)
#image input
image = tf.keras.layers.Input(shape=(img_size, img_size, 3))
                                #this line was before
i_output = tf.keras.layers.Concatenate()([image, l_output])
 . . .
model = Model([image, label], i_output)
```

- *In generator:*

```
 # label input
 label = tf.keras.layers.Input(shape=(1,))
 # create an embedding layer for 2 classes in the form of a vector
 # of size 50
 l_output = tf.keras.layers.Embedding(n_class, 50)(label)
 n_nodes = 8 * 8
 l_output = tf.keras.layers.Dense(n_nodes)( l_output)
 # reshape the layer
 l_output = tf.keras.layers.Reshape((8, 8, 1))( l_output)
 # image generator input
 noise = tf.keras.layers.Input(shape=(noise_dim,)) #this line was before
```

```
. . .
g_output = tf.keras.layers.Concatenate()([noise, l_output])
. . .
model = Model([noise, label], g_output)
```

III.5 PixelToPixel Model

The Image generation model Pix2Pix is as well a conditional GAN (cGAN). Phillip Isola developed it 2017. The characteristic of this model is it uses a pair of images as input, for example a sketch and the corresponding facade and a new image is generated. The architect is interested to generate a facade, then the sketch is the input of the generator. If the architect wants a generated sketch than the facade will be the input of the generator.
These specifics of the pixel-to-pixel model are presented below and illustrated with figures and code presented in this paper (Isola, 2017).

III.5.1 Input data

A pair of images: a sketch or a blueprint of an object, and a real image of this object are the **input images** for the discriminator, for example the sketch of a facade and a real photo of the facade

For the generator the sketch of the object is the **Input image**, like we see in the figure. It is this sketch that is transformed to generate a new image and it is the discriminator that controls this process until it decides to accept the generated image as a real one.

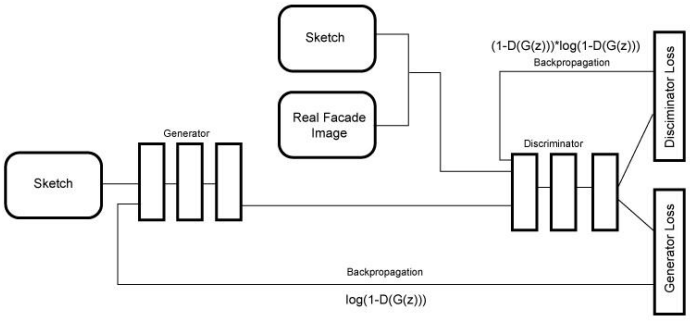


Figure 10. PixelToPixel Model
Source: Authors processing

III.5.2 Discriminator and Generator Structures

- *Generator*

To generate something new from a sketch, so to translate an image to another one, some transformations are performed to the input image by the generator using some layers.

The neural network architecture called UNET is used for the generator to generate new images from input images. The generator with this architecture uses some layers (encoder layers) to reduce the image into a smaller one and some layers (decoder layers) to enlarge it again but during the second part of transformation the image is mixed with the image of the parallel step of the first part of the process, like it is shown in the figure:

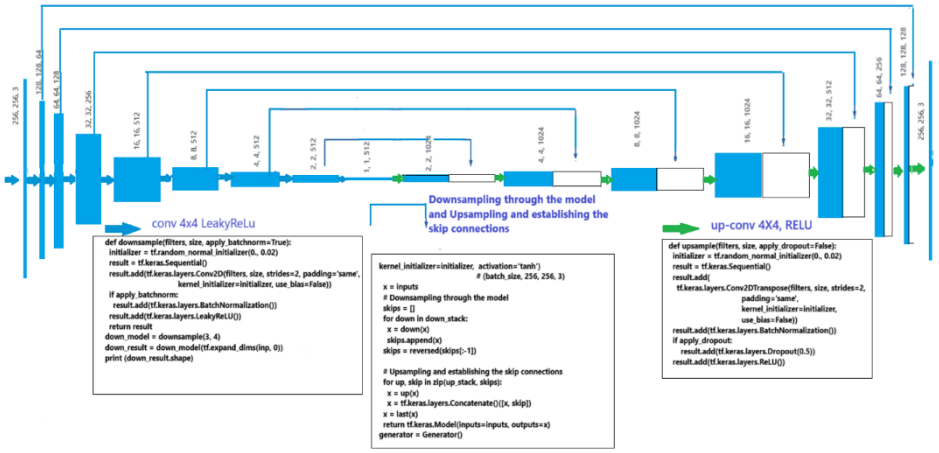


Figure 11. Generator Structure
Source: Authors processing

- *Discriminator*

As well as in the above models GAN and CGAN, the task of the discriminator is to distinguish between a real image and a generated one. The Discriminator specific in Pixel-to-Pixel is that this comparison is done for patches of the images and not for the entirely image. Therefore, it is called a patchGAN classifier. The inputs are the sketch (real or a generated one) image of dimensions 256x256x3, and the facade photo image, the two images (one sketch and facade photo) are concatenate before going in the other layers.

```
inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_image')
target = tf.keras.layers.Input(shape=[256, 256, 3], name='target_image')
x = tf.keras.layers.concatenate([inp, target])
```

These layers are the usual layers that a cGAN discriminator like Conv2D layer, BatchNormalization, LeakyReLU, etc. to achieve to a patch of a shape 30, 30, 1. This provide a better analysis of the image.

- *Training Process*

Regarding the training process, in this model the discriminator and generator are trained simultaneously, and, as in the other models, they consider different object as goal of training. Discriminator training aims to distinguish as much as possible the errors in the classification of

generated images as real ones and of real images as generated ones, meanwhile the generator training aims to diminish the difference between the generated and real images. The simultaneously training is presented with a loop like below:

```
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)
```

IV. RESEARCH RESULTS

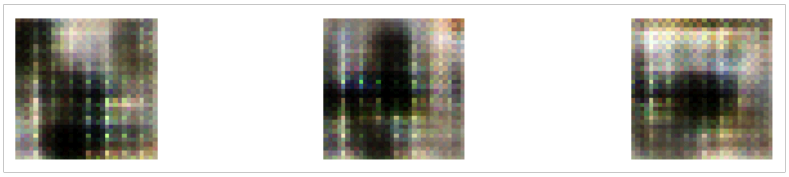
The dataset used for all the cases is dataset of 400 facades and sketches taken from www.kaggle.com/datasets. They are considered as two separated datasets. A series of experiments are done, considering different values for the parameters batch size (number of samples processed together in one forward and backward pass) and epoch number (The number of times the entire training dataset processed through the model).

In our practice there are used different structures of the generator and discriminator.

1. The first five results, that means, the result for batch size=16 and epoch=50,100,200,500,1000, are taken using a simple generator and discriminator, where:
   - generator composed by layers: Input, Dense, Reshape, LeakyReLU, Conv2DTranspose, LeakyReLU Conv2DTranspose, LeakyReLU , Conv2D
   - discriminator composed by layers: Input, Conv2D, LeakyReLU, Conv2D, LeakyReLU, Flatten, Dropout, Dense

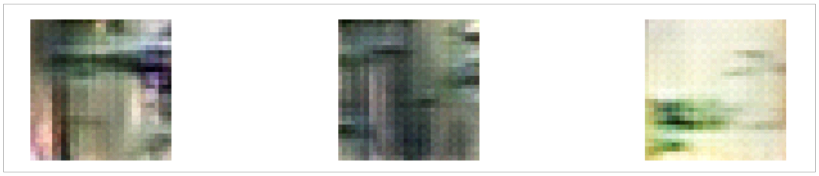
It is shown that the improvement of the quality needs a big epoch number, which is 1000. The graphics of generator and discriminator loss also show that in the case of 1000 epoch there is a closer relation between them.

DCGAN batch=16, epochs=50, kernel initialization by uniform distribution

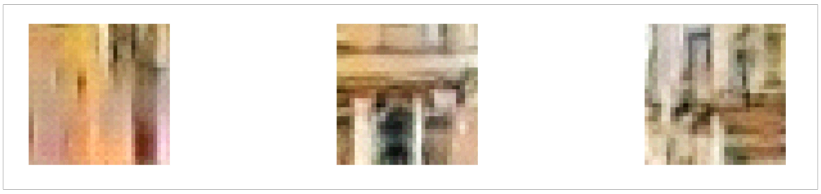


DCGAN batch=16, epochs=100,

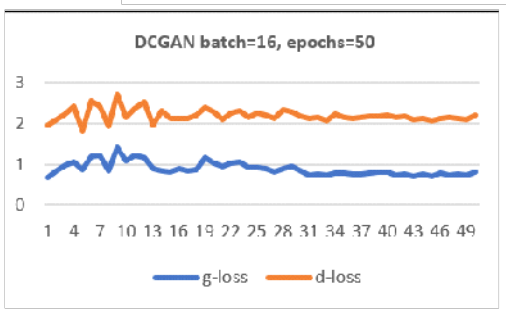
DCGAN batch=16, epochs=200,

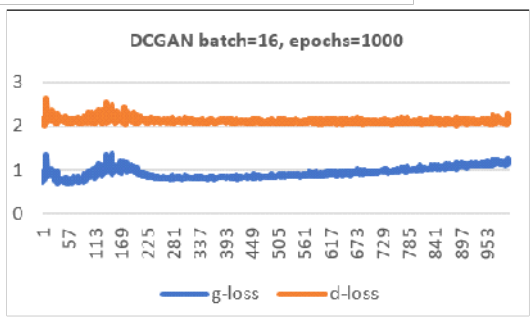


DCGAN batch=16, epochs=500,



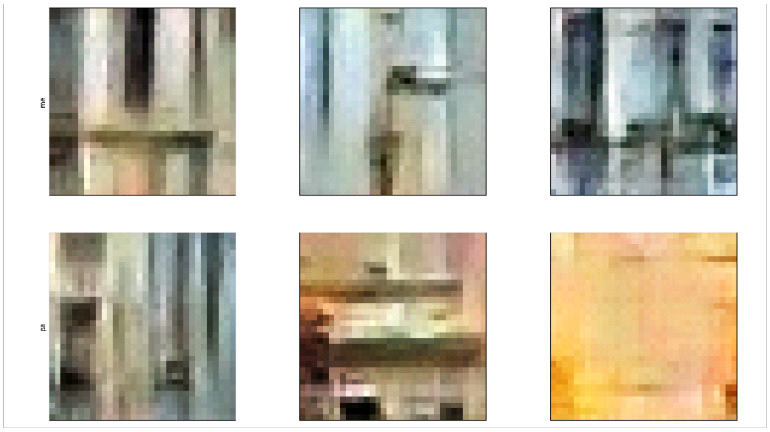
DCGAN batch=16, epochs=1000, kernel initialization by uniform distribution





2. The following is the result of the conditional GAN model but with the structure of the generator and discriminator. The images are labelled preliminary as facades with roof and without roof.

CGAN batch=16, epochs=1000, kernel initialization by normal distribution



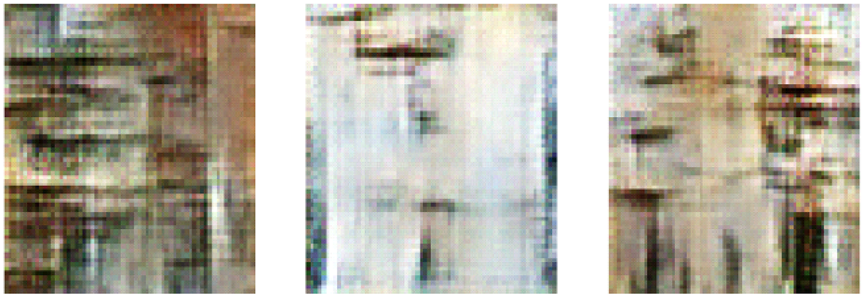
3. The two following results shows correspond to some richer structures of generator and discriminator where:
   - generator composed by layers: Input, Dense, Reshape, LeakyReLU , Conv2DTranspose, LeakyReLU Conv2DTranspose, LeakyReLU , Conv2DTranspose, LeakyReLU , Conv2D
   - discriminator composed by layers: Input, Conv2D, LeakyReLU, Conv2D, LeakyReLU, Dropout, Conv2D, LeakyReLU, Dropout, Conv2D, LeakyReLU, Dropout, Conv2D, Flatten, Dense

It is shown that the clarity of the images needs a smaller number of epochs. For the initialization of the network parameters (kernel initialization), there are used two ways: by uniform distribution, and normal distribution.

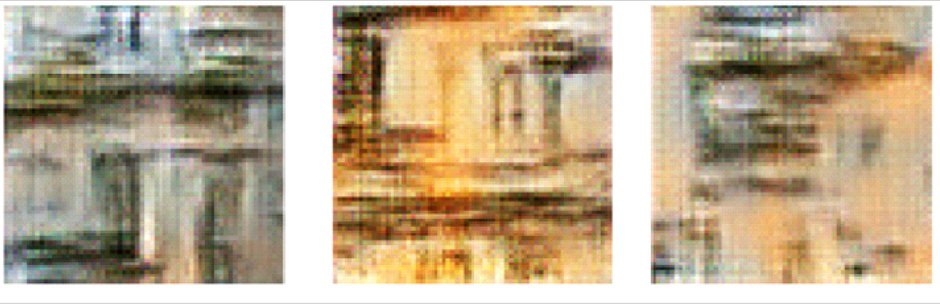
DCGAN batch=16, epochs=100, kernel initialization by uniform distribution

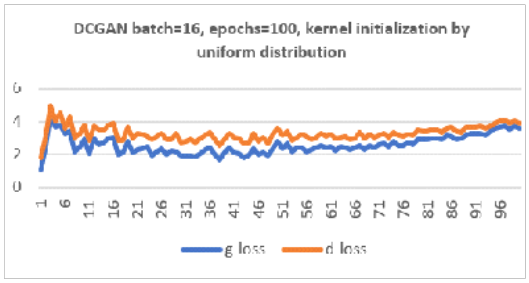
DCGAN batch=16, epochs=100, kernel initialization by normal distribution

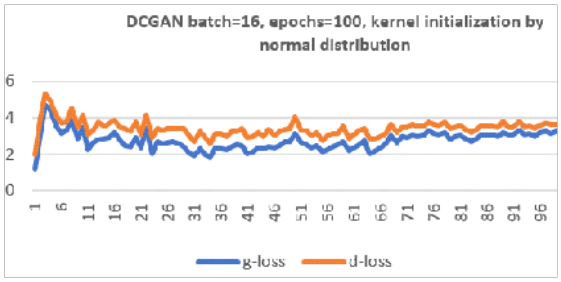
DCGAN batch=16, epochs=100, kernel initialization by uniform distribution



DCGAN batch=16, epochs=100, kernel initialization by normal distribution

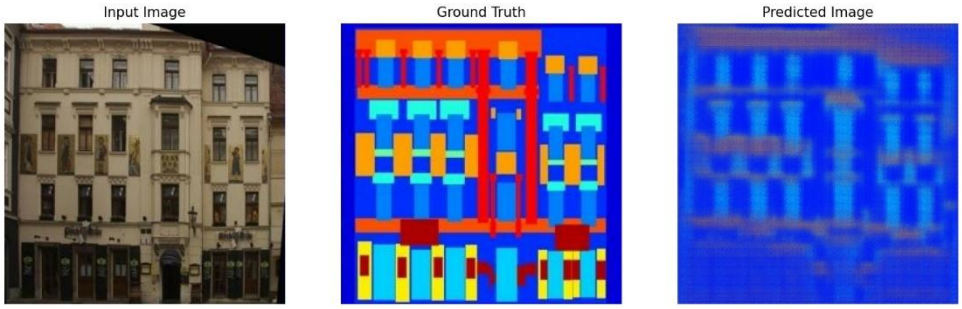




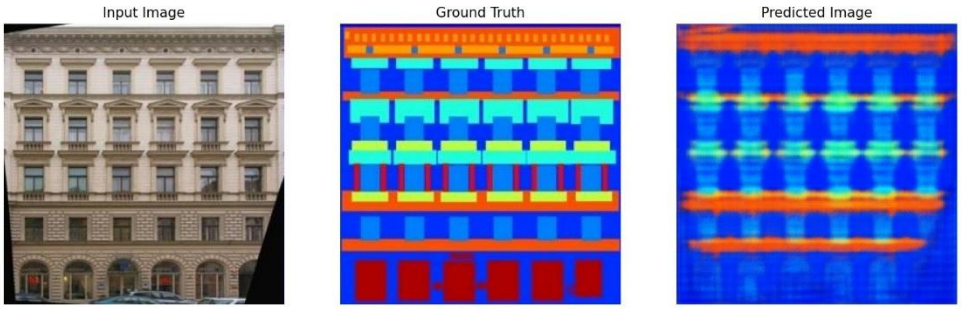


4. The following are the result of model PixelToPixel used in two ways:
   - The input for the generator is the sketch, and it is generated the fasade
   - The input of the generator is the façade and the sketch is generated

Input: the facade, generated: the sketch number of epochs=100

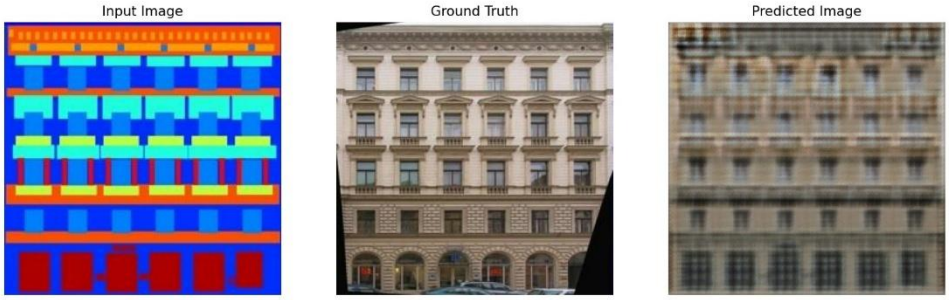


Input: the facade, generated: the sketch number of epochs=800



Input: the sketch, generated: the facade number of epochs=100

Input: the sketch, generated: the facade number of epochs=800



V. CONCLUSIONS

This paper discusses Machine Learning methodologies of image generation. These computational methodologies can be utilized in the architectural design process by furnishing a wealth of images in a short amount of time. At the same time the paper attempts to use these methodologies in such a way that the distinction between the geometrical syntax and architectural form is maintained.

The GAN model and some extension of it, like DCGAN, cGAN and PixToPix model are discussed. The paper is focused in some key parts of these models, like their structures and the training process of them, pointing out the differences between them. The experiments have used some facades and sketches datasets provided by the site www.kaggle.com.

The experiments are based on two elements batch size and number of epochs and also, in different structures of the models like generator and discriminator. There are not definitive values for the batch size and epochs, so an experience is given to avoid underfitting and overfitting phenomena for a dataset of the size around 400. This experience shows that for a such small database, to get a clear view of a generated facade are needed at least 1000 epochs, but with a richer structure of

layers of the two models Generator and Discriminator, the clarity comes faster, starting from 100 epochs. Regarding PixToPix model, it is used in two ways. The architect has a sketch, and needs a generated facade, or he has a facade and needs to generate a sketch. The results are better for a bigger number of epochs. The metric for the error calculation used in the tests, mainly was binary cross-entropy approach, because it was shown not a big change in results when we used JS-Divergence metric.

From both perspective as an architect or data scientist, the comparative analysis of the three GAN-based models — DCGAN, CGAN, and Pix2Pix — reveals notable differences in structural design and output capabilities. Given these architectural advancements and the empirical results observed, Pix2Pix emerges as the most promising model for further exploration, particularly in domains such as urban design or architectural visualization. Its ability to translate different geometric inputs or sketches into realistic façade images highlights its potential for practical applications where structure-to-image generation is essential.

REFERENCES

Arjovsky, M., & Bottou, L. (2017). *Towards principled methods for training generative adversarial networks*. In Proceedings of the 5th International Conference on Learning Representations (ICLR).

Bader, B. W., & Kolda, T. G. (2009). Tensor decompositions and applications. *SIAM Review, 51*(3), 455–500. https://doi.org/10.1137/07070111X

Chaillou, S. (2022). *Artificial intelligence and architecture: From research to practice*. Birkhäuser.

Del Campo, M. (2022). *Neural architecture: Design and artificial intelligence*. ORO Editions.

Goodfellow, I. (2014). Generative adversarial nets. In Advances in neural information processing systems (NeurIPS) (pp. 2672–2680).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Hossain, S., & Lee, D. (2023). NG-GAN: A robust noise-generation generative adversarial network for generating old image noise. *Sensors, 23*(2), Article 251. https://doi.org/10.3390/s23020251

IBM. (2025). *Semantic segmentation*. IBM Think Topics. https://www.ibm.com/think/topics/semantic-segmentation

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 5967–5976). IEEE. https://doi.org/10.1109/CVPR.2017.632

Kaneko, T., & Harada, T. (2020). Noise-robust generative adversarial networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 9994–10003). IEEE.

Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2018). Progressive growing of GANs for improved quality, stability, and variation. In Proceedings of the International Conference on Learning Representations (ICLR).

Li, C., Zhang, Y., & Wang, J. (2024). Generative AI models for different steps in architectural design: A literature review. *Frontiers of Architectural Research*. https://doi.org/10.1016/j.foar.2024.xx.xxx

Liu, G., Reda, F. A., Shih, K. J., Wang, T.-C., Tao, A., & Catanzaro, B. (2018). Image inpainting for irregular holes using partial convolutions. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 85–100). Springer.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Odena, A., Dumoulin, V., & Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*. https://doi.org/10.23915/distill.00003

Parente, J., et al. (2023). Integration of convolutional and adversarial networks into building design: A review. *Journal of Building Engineering, 76*, 107155. https://doi.org/10.1016/j.jobe.2023.107155

Radford, A., Metz, L., & Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In Proceedings of the International Conference on Learning Representations (ICLR).

Sadiku, M. N. O., Musa, S. M., & Ashaolu, T. J. (2025). Generative design in architecture. *International Journal of Trend in Scientific Research and Development*.

Shiv Ram Dubey, S., Singh, S. K., & Chaudhuri, B. B. (2021). Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing, 450*, 92–120. https://doi.org/10.1016/j.neucom.2021.03.061

X. Li, Y. Zhang, & Z. Wang. (2021). Research on multi-layer perceptron based on TensorFlow. In *Proceedings of SPIE – The International Society for Optics and Photonics*. https://doi.org/10.1117/12.258xxxx

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE International Conference on Computer Vision (ICCV) (pp. 2223–2232). IEEE. https://doi.org/10.1109/ICCV.2017.244