DATA SCIENCE TOOLBOX: PYTHON PROGRAMMING(INT-375) PROJECT REPORT

Image to image translation

Submitted by

ANKIT KUMAR

Registration No-12318930

Section - K23WA

Under the Guidance of

ANAND KUMAR (30561)

Discipline of CSE/IT
Lovely School of Computer Science

Lovely Professional University, Phagwara

<u>CERTIFICATE</u>

This is to certify that Ankit Kumar bearing Registration no. 12318930
has completed INT374 project titled, “Image to image translation”
under my guidance and supervision. To the best of my knowledge, the
present work is the result of his/her original development, effort and study.

Signature:
Name of the Supervisor: Mr Anand Kumar
Designation of the Supervisor
School of Computer Science

Lovely Professional University
Phagwara, Punjab.

Date: 12.04.2025

Acknowledgement

I am deeply grateful for the opportunity to work on this data analysis
project, which has greatly enhanced my understanding of data visualization,
statistical analysis, and Python programming.

I would like to sincerely thank my project guide and faculty members for
their constant guidance, valuable feedback, and encouragement throughout
the course of this work. Their insights and support were instrumental in
helping me complete the project successfully.

I would also like to express my appreciation to my classmates and friends for
their assistance and motivation at various stages of the project.

A special thanks to the developers and contributors of Python libraries such
as Pandas, NumPy, Seaborn, and Matplotlib, which played a key role in
performing data analysis and creating effective visualizations.

Lastly, I thank all those who directly or indirectly contributed to the
successful completion of this project.

DECLARATION

This is to declare that , I Ankit Kumar, a student of Bachelor of Technology under CSE discipline at Lovely Professional University, Punjab, hereby declare that all the information furnished in this project report is based on my own work and is authentic data.

Ankit Kumar (Reg. no: 12318930)
Department of Computer Science and Engineering Lovely Professional University, Punjab

TABLE OF CONTENTS

Image to Image Translation

Ankit Kumar Student
Computer Science Department
Department
Lovely Professional University,
Punjab
Kumarankit792005@gmail.com

Anand Kumar
Computer Science Department
Lovely Professional University
Punjab
anandk.cs20@gmail.com

Abstract

The aim of image-to-image translation algorithms is to tackle the challenges of learning a proper mapping function across different domains. Generative Adversarial Networks (GANs) have shown superior ability to handle this problem in both supervised and unsupervised ways. However, one critical problem of GAN in practice is that the discriminator is typically much stronger than the generator, which could lead to failures such as mode collapse, diminished gradient, etc. To address these shortcomings, we propose a novel framework, which incorporates a powerful spatial attention mechanism to guide the generator. Specifically, our designed discriminator estimates the probability of realness of a given image, and provides an attention map regarding this prediction. The generated attention map contains the informative regions to distinguish the real and fake images, from the perspective of the discriminator. Such information is particularly valuable for the translation because the generator is encouraged to focus on those areas and produce more realistic images. We conduct extensive experiments and evaluations, and show that our proposed method is both qualitatively and quantitatively better than other state-of-the-art image translation frameworks.

1. Introduction

Generative Adversarial Networks (GANs) [13] have drawn tremendous attention during the past few years, due to their proven ability to generate realistic and sharp looking images. Various computer vision problems are solved using this framework, such as colorization [5], super-resolution [23] and style transfer [44]. All these problems can be considered as an image-to-image translation problem: mapping an image from source domain to target domain. For instance, the super-resolution problem tries to convert a low-resolution image (source domain) to a corresponding high-resolution image (target domain). Existing literatures have show that variants of GAN achieve very impressive results under both supervised and unsupervised settings

Even with such great success, most existing GAN-based approaches are suffering from the imbalance issue between the generator and discriminator [1]. In practice, the discriminator is ordinarily too powerful compared to the generator. As a consequence, the generator may obtain limited gradients from discriminator and is hard to converge. Most state-of-the-art solutions are trying to either find an alternative objective function [2, 15, 27, 32] or plugin some new regularization terms [1, 18, 43]. However, such paradigms ignore the rich information inside the discriminator, which may lead to blurry and artificial regions.

On the other hand, the attention mechanism has been widely adopted in image translation algorithms. Recently, *Mejjati et al.* [29] concatenates an attention network before the generator and mask out the background of the output image, so merely the objects are translated into the target domain. They achieved superior performance on the object-only translation while cannot be easily generalized to scene translation. InstaGAN [31] achieves object deformation (e.g. sheep $\rightarrow$ giraffe) on the image by using the attention mask from an auxiliary network. A contemporary work proposed by *Emami et al.* [11] utilizes the internal activation from the discriminator to guide the translation. However, this approach can only be applied to the unsupervised setting. In our paper, we propose that attention mechanism should not be restricted to object translation and can be further applied to both supervised and unsupervised settings.

Inspired by the close-loop feedback control systems [4], we propose that the high intensity regions in the attention map are more significant during the translation, so that the generator should allocate more resource on these particular areas. Our framework focuses on this key idea, which aims to compute an attention map based on the discriminator's internal activation, and then feed it back to the generator. Imagine that a student is learning how to draw an apple. The standard discriminator, as a painting master, merely grades the student's painting and hopes that can help the student improve his work. On the other hand, another master point

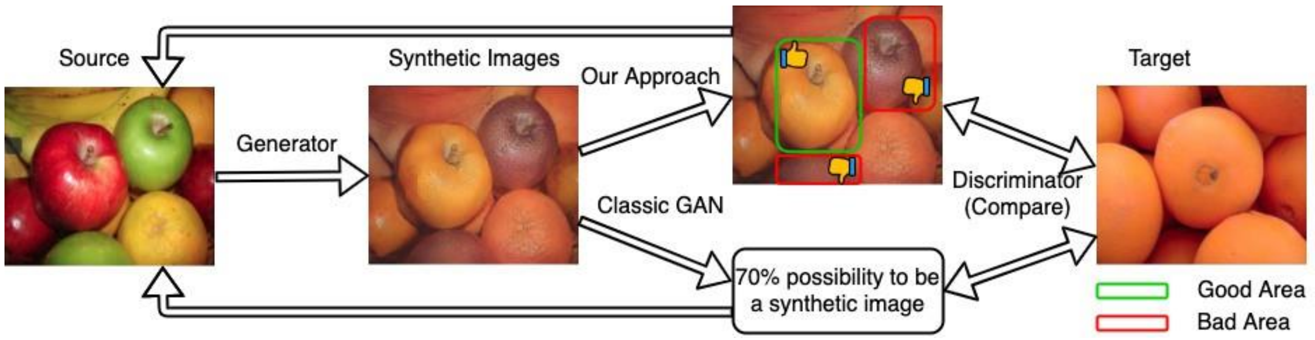


Figure 1. The discriminator distinguish real/fake image based on unrealistic regions. In this paper, we propose a novel framework that utilizes the internal information of the discriminator to enhance generator's capacity.

out areas for the student to improve for the next painting, such as incorrect regions (*e.g.* skin or stem). That is exactly our idea: we believe that the student (generator) would gain benefit from the second master (attention embedded discriminator), which provides better lead regarding spatial guidance. Our main contributions are threefold:

- A flexible attention-augmented discriminator: such discriminator provides not only the probability of realness, but also a valuable spatial attention map from its internal activation. We propose two types of attention mechanism in this paper.
- A unified GAN framework with spatial attention feedback: we propose two concatenation methods to combine the attention map with raw input 1) Adding an *alpha channel*; 2) compute the *Residual Hadamard Production* of the attention map and raw input. Noted that these methods naturally preserve the information of original input and amplify the signal of crucial regions.
- Extensive validation on different benchmarks: we provide extensive experimental validation of our proposed framework on different benchmarks. Both the qualitative results and quantitative comparisons against state-of-the-art methods demonstrate the effectiveness of our approach.

Different from previous approaches, our framework strengthens the communication and guidance between the generator and discriminator. At a high level, our work shed the light upon using auxiliary network attention information to improve the performance of image to image translation, which could be influential to other related research in the future as well.

2. Literature Review

Generative Adversarial Network GANs have achieved impressive results in image translation tasks [10, 21, 22, 23, 33]. Typically, GAN consists of two components: a generator and a discriminator. The generator is trained to fool the discriminator, which in turn tries to distinguish between real and synthetic samples. Various improvements to GANs have been proposed regarding different aspects, for instance, improved objective functions [2, 27] and advanced training strategies [14, 32, 39]. A recently proposed framework, *FAL* [20], iteratively improves synthetic images with the signal returned by a well designed spatial discriminative decoder. However, they either don't collect enough information from the discriminator, or are computational expensive because of multiple forward passes.

Image Translation Image-to-image translation can be considered as a generative process conditioned on an input image. *pix2pix* [21] was the first unified framework for supervised image-to-image translation based on conditional GAN (cGAN) [30]. *TextureGAN* [41] solves the sketch-to-image problem using user defined texture patch, and *ContextualGAN* [25] addresses the same problem by learning a joint distribution of the sketch and its image. More recently, *Gonzalez et al.* [12] adopted disentanglement representation to improve the rendering process and *Tang et al.* [36] utilized the extra semantic information to guide the generation.

Despite the promising results they achieved, the above methods are generally not applicable in practice due to the lack of paired data. Several interesting frameworks have been proposed to solve the unsupervised image-to-image translation problem. Cycle consistency loss is first introduced in *CycleGAN* [46] and is then widely used by other unsupervised image translation frameworks. For example, *UNIT* [24] improves the translation with shared latent space assumption, and *MUNIT* [19] later uses it as backbone to handle multi-modal translation. In contrast, our flexible framework can be applied on both supervised and unsupervised settings.

Attention Mechanism Generally, the attention mechanism can be viewed as guidance to bias the allocation of available processing resources towards the most informative components of an input.It's divided into two categories: post hoc network analysis and trainable attention module. The former scheme has been predominantly employed to access network reasoning for the visual object recognition task [6, 34, 35, 45]. Trainable attention models fall into two main sub-categories, hard (stochastic) that requires reinforcement training and soft (deterministic) that can be trained end-to-end [18, 37, 40].

The attention mechanism is quite useful to solve the image-to-image translation problem. *Ma et al.* [26] use a deep attention encoder to discover the instance level correspondences. *AGGAN* [29] utilizes an auxiliary trainable attention network to separate the instance and background. *InstaGAN* [31] further incorporates the instance information to improve the multi-instance transfiguraiton. Noted that any attention mechanism producing an attention map can be integrated into our framework. Without loss of generality, we implement one representative attention model each category in this paper.

3. Method Implementation

3.1. Overview

Consider images from two different domains, source domain $\mathbb{X}$ and target domain $\mathbb{Y}$. Data instances in source domain $x \in \mathbb{X}$ follow the distribution P_x, whereas instances in target domain $y \in \mathbb{Y}$ follow the distribution P_y. Our goal, in the problem setting of image-to-image translation, aims to learn mapping functions across these two different image domains, $G : x \to y$ and/or $F : y \to x$, such that the differences between P_x and $F \circ P_y$ and the difference between P_y and $G \circ P_x$ are minimized.

The main idea of our approach is to incorporate a spatial attention map generated by the discriminator, *i.e.*, augment a space of attention map M to the original input space $\mathbb{X}$, to improve the image-to-image translation task. Formally, our approach can be described as a joint-mapping learning from attention-augmented space $\mathbb{X} \oplus M_{\mathbb{X}}$ to $\mathbb{Y}$, and $\mathbb{Y} \oplus M_{\mathbb{Y}}$ to $\mathbb{X}$ if cycle consistency is applied, where $\oplus$ is the concatenate operation. Our method explicitly forces the generator to allocate more processing resources to the attended areas so it can conduct a sharp and clear translation. Generally, our method can be applied to any conditional GAN-based translation.

3.2. Architecture

Our framework, as illustrated in Figure 2, is built upon GAN and attention mechanism. For the supervised learning setting, it consists of three components, a generator G, a discriminator $D_{\mathbb{Y}}$ and an attention transfer block T. It can be extended to unsupervised setting by simply enforcing cycle consistency, which now has five components, including:

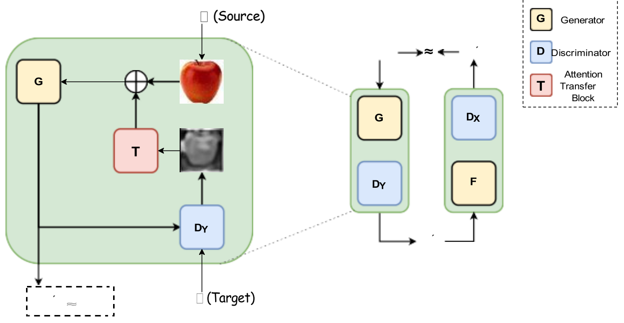


Figure 2. Overview of our framework. Left: standard GAN with an attention embedded discriminator. M_x is the attention map provided by the discriminator. The L1 loss between generated y_i' and corresponding ground truth y_i is computed. Right: the framework for unsupervised translation using cycle consistency. y_i is not available and the L1 loss between x and x' is calculated instead.

two generators G and F, two domain discriminators $D_{\mathbb{X}}$ and $D_{\mathbb{Y}}$, and one shared attention transfer component T.

The training is based on each generator-discriminator pair. Considering a standard GAN, the generator G translates an image x_i in $\mathbb{X}$ to an image in domain $\mathbb{Y}$, and the discriminator $D_{\mathbb{Y}}$ tries to distinguish whether its input is a real or fake image in domain $\mathbb{Y}$. Here, we denote $\hat{y}_i = G(x_i)$ as the output of generator, given x_i. Our attention embedded discriminator not only returns the probability of realness, $D_{\mathbb{Y}}(\hat{y}_i) \in [0, 1]$, but also an attention map A_{x_i} that highlights the attending areas of $D_{\mathbb{Y}}$. This attention map then will be transferred to a pixel-level weight map, M_{x_i} via the attention transfer block T and concatenated with the raw input. It's worth noting that the actual input of our generator G is the concatenation of x_i and M_{x_i}, formulated as $x_i' = x_i \oplus M_{x_i}$. At the start of the training, the attention map of each image is not available so we initialize it as an all-ones matrix $A_{x_i} \in \mathbb{R}^{m \times n}$, where $m \times n$ is the shape of the input image. Other initialization methods, like random noise, have also be examined but have limited impact on the final result. The translation process of generator G can be formulated as:

$$\hat{y}_i^{(k+1)} = G(x_i \oplus T(D(\hat{y}_i^{(k)})); \vartheta), k = 0, 1, 2, \dots \quad (1)$$

where k and $k+1$ denote the index of iteration and ϑ is the parameter of G. Please note that we use the attention feedback from previous iteration for the same input, which is more efficient comparing to *FAL* [20] that requires multiple forward passes per instance. Assume we only provide the raw input to the generator, G may waste its processing resources on some peripheral locations thus $D_{\mathbb{Y}}$ can beat it easily. As a consequence, the loss of the discriminator quickly converges to zero and the generator can no longer efficiently update its parameter. Alternatively, by concate-

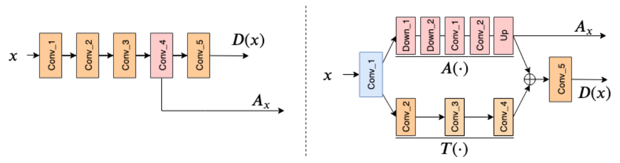


Figure 3. The architecture of different type of discriminator. Left: PHA that builds attention map from a specific layer. Right: TAM that builds attention map from an additional network branch.

nating the raw input with M_x, the generator knows exactly where the discriminator is noticing and can manage its resources appropriately. As illustrated in Figure 2, we can extend this framework to perform the unsupervised translation by adding another GAN component and enforcing cycle consistency.

3.3. Attention Map

Our discriminator provides an extra attention map A_{x_i} for each image generated from x_i. We consider both *post hoc attention* (PHA) that leaves the discriminator untouched, and *trainable attention module* (TAM) that leads to better distinguishing power.

Given input x, the PHA attention map can be constructed from the backward gradients, forward activation, or the mix of them [34]. We build our discriminator based on the classical PatchGAN [21]. The network is presented as $D = \{l_0, l_1, \dots, l_m\}$ where l_i denotes i-th convolution layer in the network, and $Act_D = \{a_1, a_2, \dots, a_m\}$ is the set of activation map of corresponding layer. The PHA attention is sensitive to layer selection, as different layer activation leads to different attention map [28]. Specifically, if l_t is the chosen layer, the attention map can be described as:

$$M = norm\left(\frac{1}{c}\sum_{i=1}^{c} |a_{t,i}|\right) \qquad (2)$$

where c is the number of channels in t-th layer and $norm(\cdot)$ applies the min-max normalization. As suggested by [28], we chose the 4-th convolution layer in our experiment. This attention map only requires minor computation and works surprisingly well in most cases, but it may not achieve promising results facing complex images (e.g. scene images). On the contrary, a TAM is suitable for such complex input since it simultaneously increases the capacity of generator and discriminator.

Our TAM follows the same 2-branch architecture of the attention block in RAM [37]. Noted that the discriminator is ordinarily powerful than the generator, the enhancement over discriminator must be chosen wisely. Thus, we replaced the *Resblock* [16] by a simple convolution layer. As presented in the right part of Figure 3, first few layers of the discriminator extract the low-level information of the input, and passes it through following branches. Given the trunk branch output $T(x)$ with the input x, the attention branch learns an attention map $A(x)$ that softly weights the output of trunk branch. The output of such module is:

$$E_C = (A_C(x) + 1) \times T_C(x) \qquad (3)$$

where C is the set of channels. Finally, a convolutional layer computes the probability of realness based on E and an attention map from the attention branch output, $M = \text{Avg}(A_C(x))$, will be returned.

3.4. Concatenation

In this section, we propose two methods to blend the attention map M_x with its corresponding input x. The first one is based on the aforementioned TAM. We compute the Residual Hadamard Production (RHP) of the attention map and original input. Such operation is superior comparing to dot production because dot production with the weight factor range from zero to one will degrade the pixel value and cause fractional pixel problem [29]. RHP can be formulated as:

$$x' = x \oplus M_x = (g(M_x; \vartheta) + 1) \times x \qquad (4)$$

where $g(;\vartheta)$ is the attention transfer block T that transfer the attention map to corresponding pixel weight map. It's implemented as a small 3-layers convolution network.

Another intuitive concatenation is called *Alpha* concatenation and is inspired by RGBA and Depth image, which contains three-channel RGB color model supplemented with a 4-th channel that provides additional information, like the opaque level of each pixel. By using this method, the importance of each pixel is observed by the generator explicitly. Formally, it is described as:

$$x' = x \oplus M_x = \{x_r, x_g, x_b, g(M_x; \vartheta)\} \qquad (5)$$

where $g(;\vartheta)$ is the same transfer function in RHP. Remeber that a gray scale image can be transformed into RGB image by repeating its intensity for each RGB channel. It's worth noting that these two concatenate methods only allow the attention map to amplify the pixel signal, and the generator can always receive the original input. It is crucial for the trick we used during the test since the generator won't completely rely on the attention map.

3.5. Training loss

Let's start with the supervised translation setting. The adversarial loss of a vanilla GAN consists of one generator G and one discriminator D can be expressed as:

$$\begin{aligned} L_{GAN}(G, D) = \ & \mathbb{E}_{y\sim Y}[\log D(y)] + \\ & \mathbb{E}_{x\sim X}[\log(1 - D(G(x')))] \end{aligned} \qquad (6)$$

where x' is computed from Eq 4 or 5. This cost function is well known for its training difficulty [1]. We adopt the modified least-squares loss [27] to further stabilize the training process and improve the quality of generated images:

$$L_{GAN}(G, D) = \mathbb{E}_{y\sim Y}[(D(y) - 1)^2] + \mathbb{E}_{x\sim X}[(G(x'))^2] \tag{7}$$

Noted that adversarial loss alone does not guarantee a sound translation. It is beneficial to mix traditional loss like L1 or L2 distance between synthesized image and ground truth. Based on the suggestion from *pix2pix* [21] that L1 loss encourages less blurry, L1 loss has be chosen as part of our supervised training objective:

$$L_{L1}(G) = \mathbb{E}_{x,y}[\|y - G(x')\|_1] \tag{8}$$

The final objective function in this setting is:

$$\arg\min_{G}\max_{D} L_{GAN}(G, D) + \lambda L_{L1}(G) \tag{9}$$

We can extend this framework to further conduct the unsupervised translation task by adding another pair of generator and discriminator, and enforcing cycle consistency. Assume the generator G simulates the map function $G: X \rightarrow Y$ and discriminator D_Y are trying to distinguish between $G(x)$ and y, the objective of this GAN component is $L_{GAN}(G, D_Y)$. The generator F and discriminator D_X is doing the same task in the opposite direction, its loss function is $L_{GAN}(F, D_X)$. Cycle consistency is employed in such unsupervised setting because it alleviate the shortness of paired data. It assumes that if a image x from domain X has be translated to a fake image in domain Y, we should get the same image x by applying $F: Y \rightarrow X$. This behavior is formally presented as:

$$L_{cyc}(G, F) = \mathbb{E}_{x\sim X}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y\sim Y}[\|G(F(y)) - y\|_1] \tag{10}$$

The final objective in the unsupervised setting is:

$$\arg\min_{G,F}\max_{D_X,D_Y} L_{GAN}(G, D_Y) + L_{GAN}(F, D_X) + \lambda L_{cyc}(G, F) \tag{11}$$

4. Analysis

To verify the effectiveness of our proposed framework, we evaluate it on both unsupervised setting and supervised setting in this paper. The source code is available at `https://github.com/voidstrike/ASGIT`

A crucial point of our framework is how can we perform the inference in test phase. The attention map of each image is not available beforehand, and some placeholders are

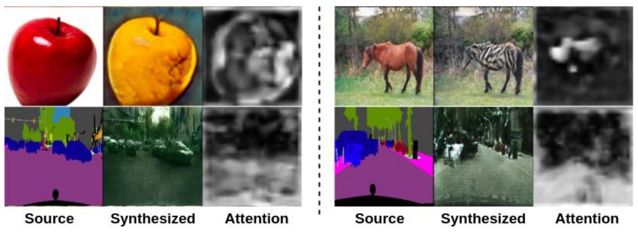


Figure 4. Examples of attention maps. Left: Attention maps generated by PHA; Right: Attention maps generated by TAM.

required. Based on the training phase and the concatenation in Sec 3.4, an all-one attention is used as the placeholder as we assume the whole image is important by default.

4.1. Settings

4.1.1 Datasets

We evaluate our method on four benchmarks for unsupervised translation. *orange2apple*, *horse2zebra* [9] are for object transfer; *summer2winter* [46] and *day2night* are two challenging scenery tasks. *day2night* contains 7870 daytime street images and 8592 night street images cropped from BDD110k [42]. Furthermore, we evaluate on *Cityscape* [8] for both supervised and unsupervised translations. All data are randomly split into train and test (80/20 split).

4.1.2 Baselines

For the unsupervised translation setting, we compare our framework to CycleGAN [46] that enforces cycle consistency, and UNIT [24] that leverages the latent space assumption between source/target images. Also, we compare with StarGAN [7], which is capable for multiple domains translation. Additionally, we include AGGAN algorithm [29] in the comparison, which separates the foreground and background via an attention network.

For the supervised translation setting, we compare to GAN [13] and cGAN [30]. The only difference between them is that cGAN is conditioned on the input. We also consider pix2pix [21] in the comparison, which extends cGAN by adding a reconstruction loss. Moreover, we compare with FAL [20], which iteratively modifies the hidden feature according to the discriminator's feedback.

4.1.3 Metrics

To be comparable with previous approaches [21, 29, 46], FCN score is computed to evaluate *Cityscape* tasks and Kernel Inception Distance (KID) [3] is for unsupervised translation. KID computes the squared MMD (Maximum Mean Discrepancy) between feature representations of real and generated images. Different from the Fre´chet Inception Distance [17], KID is more reliable because of the unbiased estimator. While KID is unbounded, the lower its value, the more shared visual similarities there are between real and generated images.

Two types of KID are reported based on the task. Target-KID measures the distance between generated images and target domain, while fused-KID denotes the distance between synthesized images and both domains. Generally, target-KID is suitable for object translation since we only care about the target object rather than the background. On the other hand, fused-KID is good for scenery task because both foreground and background matter [29].

4.1.4 implementation

To be comparable with previous methods [20, 21, 46], we use 256×256 images for the *unsupervised Cityscape translation* and all object and scenery tasks, and 128×128 images for the *supervised Cityscape translation*. In the preprocessing step, we resized the input image to 286×286 (143×143) then randomly cropping back to 256×256 (128×128). For all the unsupervised experiments, we set the weight factor of the GAN loss to 1, $\lambda_{GAN} = 1$, and the weight factor of cycle consistency to 10, $\lambda_{Cyc} = 10$. On the other hands, we set $\lambda_{GAN} = 1$ and $\lambda_{L1} = 100$. for the supervised setting.

We used Adam optimizer with batch size 1, training on a Quadro 8000 GPU. All networks were trained from scratch, with learning rate of 0.0002 for both the generator and discriminator, and $\beta_1 = 0.5, \beta_2 = 0.999$ for the optimizer. We kept learning rate for first 100 epochs and linearly decayed to 0 for next 100 epochs.

4.2. Attention and Concatenation

Recall that we implement two attention mechanisms and two concatenations for our experiment. The remaining problem is how to combine them properly. We first present qualitative results in Figure 5. As discussed in Section 3, TAM is not good at handling simple datasets, e.g. *apple2orange*, while the results are more attractive for more complex *summer2winter* dataset. By comparing alpha concatenation with RHP under post hoc attention, we find that the contrast ratio of the synthesized image is usually too high and leads to unrealistic images.

We also present attention map examples in Figure 4 and a quantitative evaluation for each combination in Table 1. Numerical results in the table justify our previous observations. Based on the overall performance across different tasks, most experiments use PHA and RHP in the following sections.

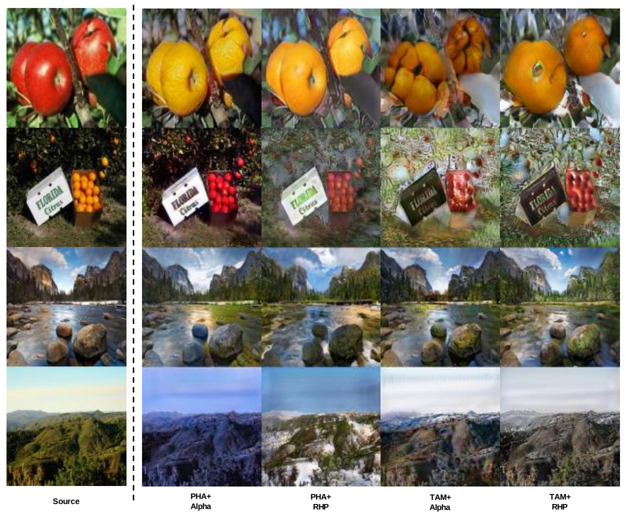


Figure 5. Different combination of attention and concatenation on *apple2orange* and *summer2winter*. First column is the real input. From second column to the right: PHA and alpha, PHA and RHP, TAM and alpha, TAM and RHP

4.3. Object and Scenery Translation

We present target-KID in Table 2 and fused-KID in Table 3. Our proposed framework outperforms all baselines in all tasks except *day→night*. Nevertheless, our result is very close to the winner. This observation is consistent with our qualitative evaluation in Figure 6, where our fake horse (zebra) is much more realistic than the counterparts produced by baselines. However, our method dramatically changes the background comparing to other methods, which means it is a better choice if the background doesn't play a important role in the translation.

Scenery translation results are presented in Figure 6. It's surprising to see the simplest CycleGAN model got first place in *day2night*, which is harder than two aforementioned object transfer datasets. Notwithstanding, CycleGAN got 2nd place on night→day, which is commonly considered easier. Another interesting observation is AGGAN does not perform any translation these cases. Based on the idea of AGGAN, it will decompose the image into foreground and background. But a proper 'foreground' cannot be found in scenery translation, thus no translation can be conduct. To sum up, our method produces more realistic scenery images comparing to baselines.

4.4. Cityscape translation

We evaluate our method on *Cityscape* [8] for both supervised and unsupervised settings. We train *photo→label* and *label→photo* on the *Cityscape*, and compare the output images with the ground truth.

As shown in Table 4, our method significantly outper-

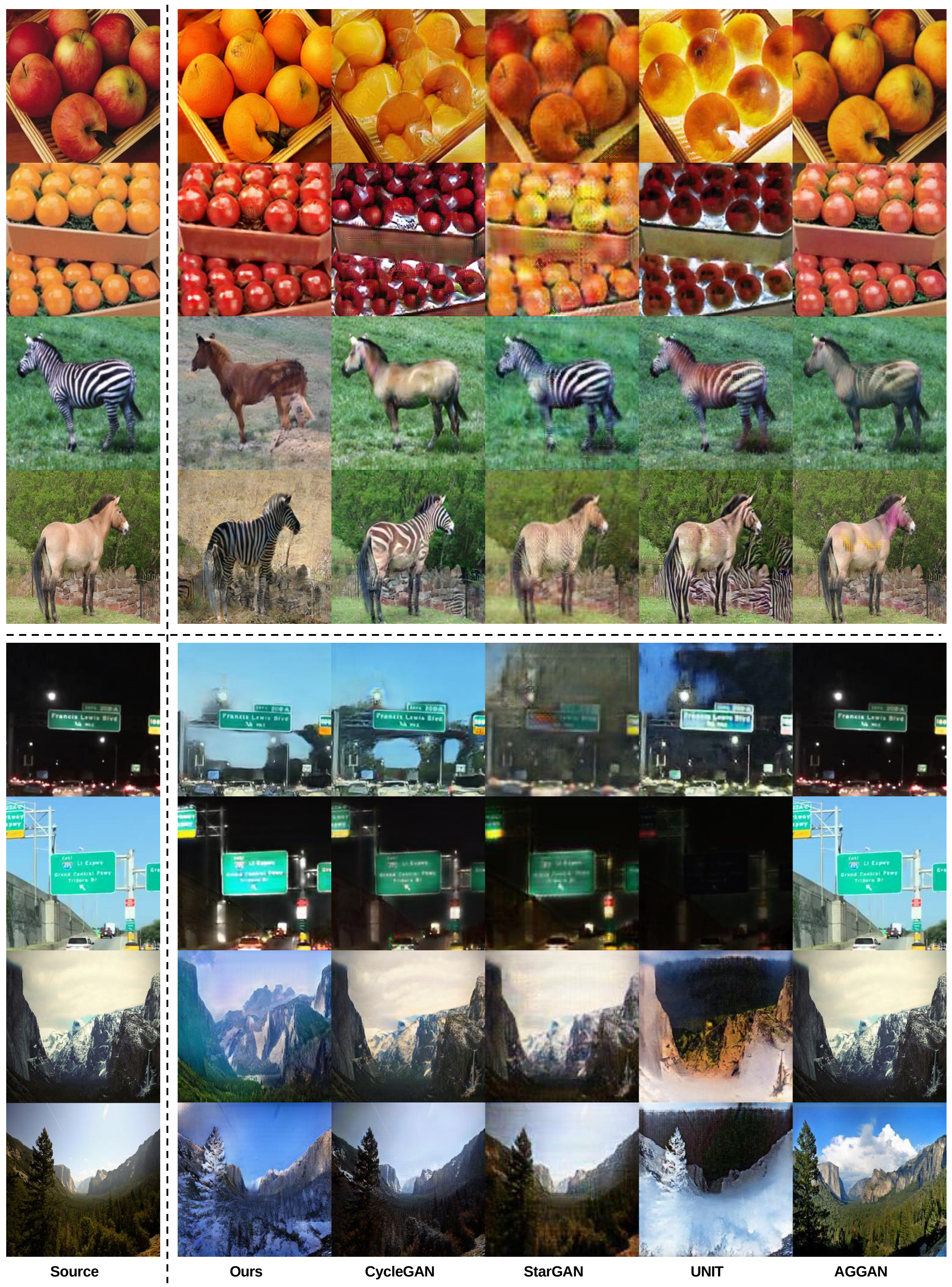


Figure 6. Image-to-Image translation results generated by different approaches on object translation and scenery translation. Every two rows from top: *apple*$\longleftrightarrow$*orange*, *zebra*$\longleftrightarrow$*horse*, *night*$\longleftrightarrow$*day* and *winter*$\longleftrightarrow$*summer*. More result is available in the supplementary

Method	(A)pple⟷(O)range		(S)ummer⟷(W)inter		(A)pple⟷(O)range		(S)ummer⟷(W)inter	
	A→O	O→A	S→W	W→S	A→O	O→A	S→W	W→S
PHA+Alpha	7.25 ± 0.83	3.69 ± 0.41	1.86 ± 0.24	1.01 ± 0.23	4.02 ± 0.37	4.11 ± 0.31	1.04 ± 0.12	1.23 ± 0.12
PHA+RHP	6.31 ± 0.60	2.99 ± 0.38	1.98 ± 0.33	1.03 ± 0.26	3.69 ± 0.27	4.30 ± 0.31	1.18 ± 0.16	1.55 ± 0.13
TAM+Alpha	10.80 ± 0.71	7.26 ± 0.47	2.37 ± 0.35	1.76 ± 0.37	5.93 ± 0.31	6.70 ± 0.36	1.45 ± 0.18	1.71 ± 0.15
TAM+RHP	10.06 ± 0.64	6.81 ± 0.45	1.34 ± 0.29	1.73 ± 0.30	5.54 ± 0.31	6.47 ± 0.37	0.82 ± 0.14	1.72 ± 0.15

Table 1. KID ± std. (scaled by 100) computed for different combination on *apple2orange* and *summer2winter*. Left 4 columns shown the target-KID and the rest 4 columns show the fused-KID (Lower the better).

Method	(A)pple⟷(O)range		(H)orse⟷(Z)ebra		(D)ay⟷(N)ight		(S)ummer⟷(W)inter	
	A→O	O→A	H→Z	Z→H	D→N	N→D	S→W	W→S
CycleGAN	8.48 ± 0.53	5.94 ± 0.65	3.94 ± 0.41	4.87 ± 0.52	2.63 ± 0.20	7.68 ± 0.35	2.78 ± 0.22	1.86 ± 0.26
StarGAN	13.32 ± 0.52	11.19 ± 0.51	12.42 ± 0.74	12.21 ± 0.89	5.37 ± 0.43	8.49 ± 0.34	8.05 ± 0.37	8.72 ± 0.47
AGGAN	10.61 ± 0.79	4.57 ± 0.30	4.12 ± 0.80	4.46 ± 0.40	8.09 ± 0.37	7.85 ± 0.29	3.45 ± 0.43	2.75 ± 0.20
UNIT	17.41 ± 1.13	7.26 ± 0.57	12.25 ± 0.74	12.37 ± 0.84	2.83 ± 0.30	11.00 ± 0.53	6.20 ± 0.25	5.99 ± 0.28
Ours (PHA+RHP)	6.31 ± 0.60	2.99 ± 0.38	1.03 ± 0.35	3.42 ± 0.51	2.76 ± 0.32	6.96 ± 0.38	1.98 ± 0.33	1.03 ± 0.26

Table 2. Target KID ± std. (scaled by 100) computed for different methods and on different datasets. Best results are bolded.

Method	(A)pple⟷(O)range		(H)orse⟷(Z)ebra		(D)ay⟷(N)ight		(S)ummer⟷(W)inter	
	A→O	O→A	H→Z	Z→H	D→N	N→D	S→W	W→S
CycleGAN	11.02 ± 0.60	9.82 ± 0.51	10.25 ± 0.25	11.44 ± 0.38	1.95 ± 0.13	3.63 ± 0.20	2.05 ± 0.12	3.34 ± 0.12
StarGAN	9.15 ± 0.43	8.31 ± 0.48	7.14 ± 0.48	4.50 ± 0.36	3.43 ± 0.20	5.18 ± 0.23	3.95 ± 0.17	4.14 ± 0.21
AGGAN	6.44 ± 0.69	5.32 ± 0.48	6.93 ± 0.27	6.71 ± 0.27	4.14 ± 0.14	4.97 ± 0.18	3.15 ± 0.19	2.45 ± 0.13
UNIT	11.68 ± 0.43	10.48 ± 0.67	4.91 ± 0.36	4.39 ± 0.33	2.48 ± 0.16	6.12 ± 0.29	3.51 ± 0.15	2.83 ± 0.12
Ours (PHA+RHP)	3.69 ± 0.27	4.30 ± 0.31	8.42 ± 0.47	8.46 ± 0.41	2.48 ± 0.15	4.58 ± 0.23	1.18 ± 0.16	1.55 ± 0.13

Table 3. Fused KID ± std. (scaled by 100) computed for different methods and on different datasets. Best results are bolded.

Method	Label→Photo			Photo→Label		
	Per-pixel acc.	Per-class acc.	IoU	Per-pixel acc.	Per-class acc.	IoU
CycleGAN	0.42	0.15	0.10	0.56	0.21	0.17
UNIT	0.48	0.17	0.11	0.58	0.18	0.14
AGGAN	0.37	0.11	0.09	0.49	0.14	0.10
StarGAN	0.47	0.16	0.11	0.61	0.21	0.17
Ours (PHA)	0.52	0.20	0.12	0.60	0.24	0.19
Ours (TAM)	0.49	0.19	0.10	0.59	0.23	0.19

Table 4. FCN-scores (Higher is better) for different methods, evaluated on *Cityscape* label⟷photos in unsupervised setting.

Method	Label→Photo			Photo→Label		
	Per-pixel acc.	Per-class acc.	IoU	Per-pixel acc.	Per-class acc.	IoU
GAN	0.22	0.05	0.01	0.32	0.08	0.02
cGAN	0.57	0.20	0.14	0.71	0.26	0.21
FAL	0.57	0.18	0.13	0.77	0.25	0.21
pix2pix	0.61	0.22	0.16	0.80	0.43	0.32
Ours(PHA)	0.63	0.23	0.16	0.81	0.42	0.32
Ours(TAM)	0.63	0.22	0.16	0.75	0.40	0.30

Table 5. FCN-scores (Higher is better) for different methods, evaluated on *Cityscape* label⟷photos in supervised setting.

forms the baselines in the unsupervised experiments. The compelling improvement in the pixel-level accuracy comes from the guidance of the attention map, which aligns with our expectations. However, the improvement of other metrics is somehow limited. We suggest that it's because only few domain specific classes are highlighted in the attention map, and the generator works too hard on these objects and ignores others. Another possible cause would be the number of classes per image, which is small in this task and we cannot increase the score for nonexistent classes. Since it's not our major contribution, we leave the justification in the supplementary.

Meanwhile, the improvement of the supervised translation is not as sharp as the unsupervised translation according to Table 5, yet it still shows that further improvement can be achieved with little extra computation. We believe that it's majorly due to the strong regularization enforced by the L1 norm. Note that *pix2pix* and our framework share $\lambda = 100$ in Eq. 9, but FAL has $\lambda = 10$ in their implementation. This may explain why FAL, as a recurrent modification of *pix2pix*, got worse performance. It also further justified that L1 loss may sufficient for the supervised case already.

5. Conclusion

This work argues for spatial attention, which unveils the regions of an image for the discriminator to determine whether that image is real or fake, can significantly improve the performance of GANs on image-to-image translation tasks. It is noteworthy that no additional supervision is needed to generate this attention map. Our method not only shows compelling improvement on both unsupervised and supervised learning tasks compared to state-of-the-art algorithms, but also demonstrates an insightful investigation to the behaviors of GANs. We further remark that our idea can apply on any GAN-based model with little modification. According to our experiment, we observe that the performance of our proposed framework is sensitive to the selection of attention module and concatenation method. Investigating the impact of different attention mechanisms and new tasks could be an interesting research direction in the future.

6. Future Scope

The field of image-to-image translation holds immense potential for growth, with several promising directions for future research and application:

1. **Reduction in Dependency on Paired Data**
   Current supervised models rely heavily on large, high-quality paired datasets, which are expensive and time-consuming to collect. Future advancements may focus on minimizing this dependency using semi-supervised, few-shot, or self-supervised approaches.
2. **Real-Time Translation and Efficiency**
   As the demand for real-time image translation in applications like AR/VR and autonomous driving increases, future work may aim to develop lightweight models that can run efficiently on edge devices with limited computational resources.
3. **Improved Generalization Across Domains**
   Enhancing the generalization capability of models across unseen domains without retraining remains a key challenge. Techniques such as domain adaptation, meta-learning, and better normalization strategies are likely to be explored.
4. **Multimodal and High-Resolution Translation**
   There is a growing interest in multimodal translation—generating diverse outputs for the same input—and translating high-resolution images with greater detail and consistency, which remains an open research problem.
5. **Integration with Transformers and Diffusion Models**
   The use of Vision Transformers (ViTs) and Diffusion Models in generative tasks has shown promising results. Combining them with GAN-based methods could lead to superior image synthesis and stability.
6. **Ethical and Fair Use of Translation Models**
   As image translation becomes more powerful, ensuring ethical use and avoiding bias or misinformation becomes critical. Future research may focus on fairness, transparency, and responsible AI practices in model design and deployment.

7. References

1. **Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A.**
   Image-to-Image Translation with Conditional Adversarial Networks.
   Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.
   https://arxiv.org/abs/1611.07004
2. **Wang, T. C., Liu, M. Y., Zhu, J. Y., Tao, A., Kautz, J., & Catanzaro, B.**
   High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs.
   Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.
   https://arxiv.org/abs/1711.11585
3. **Park, T., Liu, M. Y., Wang, T. C., & Zhu, J. Y.**
   Semantic Image Synthesis with Spatially-Adaptive Normalization.
   Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.
   https://arxiv.org/abs/1903.07291
4. **Zhu, J. Y., Park, T., Isola, P., & Efros, A. A.**
   Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks.
   Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017.
   https://arxiv.org/abs/1703.10593
5. **Pang, Y., Lin, J., Qin, T., & Chen, Z.**
   Image-to-Image Translation: Methods and Applications.
   arXiv preprint arXiv:2101.08629, 2021.
   https://arxiv.org/abs/2101.08629

Overview

This repository contains a comprehensive analysis of the **"Sketch Image to Image Translation Data.csv"** dataset, focusing on translating sketch images into realistic images using supervised learning techniques. The aim is to understand how sketch features can be effectively mapped to real image representations through data-driven approaches, particularly using image-to-image translation models like Pix2Pix.

Dataset

The primary dataset used is **"Sketch Image to Image Translation Data.csv"**, which includes the following key features:

- **Sketch_ID**: Unique identifier for each sketch image.
- **Real_Image_ID**: Corresponding real image identifier mapped to the sketch.
- **Sketch_Features**: Encoded or flattened representation of visual features extracted from sketch images.
- **Real_Image_Features**: Encoded or flattened representation of features extracted from real images.
- **Category**: The object class or domain (e.g., face, shoe, handbag).
- **Pixel_Similarity**: A numeric score reflecting similarity between sketch and real image at the pixel level.
- **Edge_Density**: A measure of how dense the edge lines are in the sketch.
- **Transformation_Loss**: Loss metric obtained during image translation training (e.g., L1 or adversarial loss).
- **Label**: A binary or categorical label indicating if the transformation was successful or for classification purposes.

Procedures and Analysis

Data Exploration and Visualization:

- Loaded the dataset into a Pandas DataFrame for structured analysis.
- Performed a preliminary scan of the dataset structure, including types, missing values, and statistical summaries.
- Visualized distribution of sketch complexity (edge density) and transformation success using **Matplotlib** and **Seaborn**.
- Created correlation plots to assess the relationship between sketch features, transformation loss, and pixel similarity.

Data Preprocessing:

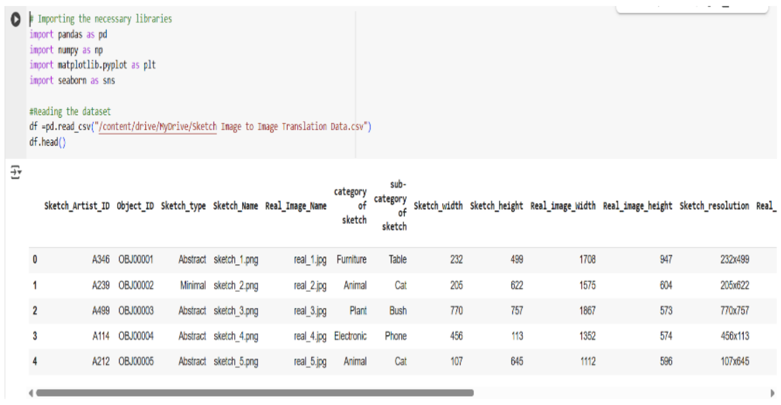
- Applied **MinMaxScaler** to normalize continuous features like pixel similarity and transformation loss.
- Encoded categorical variables like Category using one-hot encoding for model compatibility.
- Split the dataset into training and testing sets (typically 80/20) to train supervised models such as **Random Forests**, **Support Vector Machines**, and **Neural Networks**.

Image Pair Visualization:

- Sampled several sketch-to-real image pairs to visually inspect the quality of mapping and reconstruction.
- Included edge-detection overlays and difference heatmaps to analyze areas where the model performs well or poorly.

IMPLEMENTATION

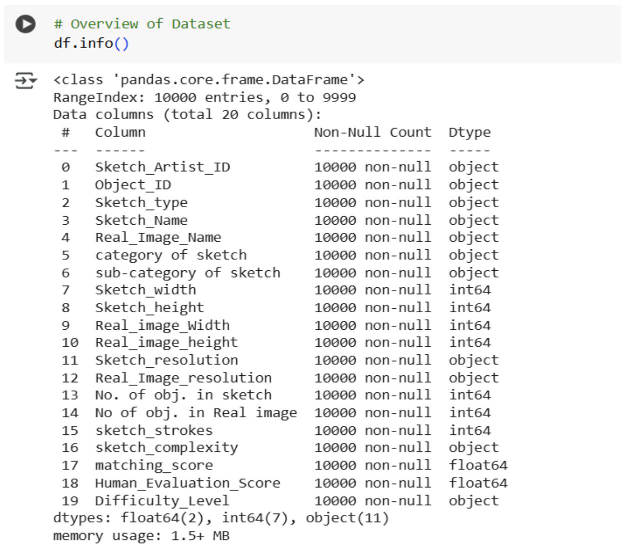
```
# Importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Reading the dataset
df =pd.read_csv("/content/drive/MyDrive/Sketch Image to Image Translation Data.csv")
df.head()
```



```
# Importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Reading the dataset
df =pd.read_csv("/content/drive/MyDrive/Sketch Image to Image Translation Data.csv")
df.head()
```

	Sketch_Artist_ID	Object_ID	Sketch_type	Sketch_Name	Real_Image_Name	category of sketch	sub-category of sketch	Sketch_width	Sketch_height	Real_image_Width	Real_image_height	Sketch_resolution	Real_
0	A346	OBJ00001	Abstract	sketch_1.png	real_1.jpg	Furniture	Table	232	499	1708	947	232x499	
1	A239	OBJ00002	Minimal	sketch_2.png	real_2.jpg	Animal	Cat	205	622	1575	604	205x622	
2	A499	OBJ00003	Abstract	sketch_3.png	real_3.jpg	Plant	Bush	770	757	1867	573	770x757	
3	A114	OBJ00004	Abstract	sketch_4.png	real_4.jpg	Electronic	Phone	456	113	1352	574	456x113	
4	A212	OBJ00005	Abstract	sketch_5.png	real_5.jpg	Animal	Cat	107	645	1112	596	107x645	

Overview of Dataset
df.info()

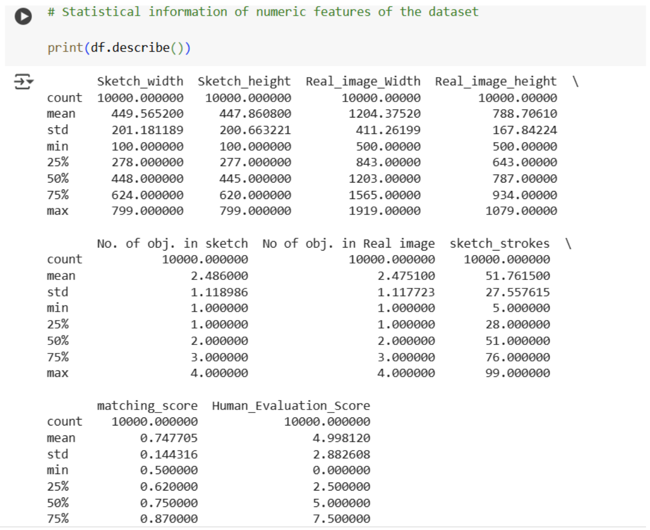


```
# Overview of Dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Sketch_Artist_ID          10000 non-null  object
 1   Object_ID                 10000 non-null  object
 2   Sketch_type               10000 non-null  object
 3   Sketch_Name               10000 non-null  object
 4   Real_Image_Name           10000 non-null  object
 5   category of sketch        10000 non-null  object
 6   sub-category of sketch    10000 non-null  object
 7   Sketch_width              10000 non-null  int64
 8   Sketch_height             10000 non-null  int64
 9   Real_image_Width          10000 non-null  int64
 10  Real_image_height         10000 non-null  int64
 11  Sketch_resolution         10000 non-null  object
 12  Real_Image_resolution     10000 non-null  object
 13  No. of obj. in sketch     10000 non-null  int64
 14  No of obj. in Real image  10000 non-null  int64
 15  sketch_strokes            10000 non-null  int64
 16  sketch_complexity         10000 non-null  object
 17  matching_score            10000 non-null  float64
 18  Human_Evaluation_Score    10000 non-null  float64
 19  Difficulty_Level          10000 non-null  object
dtypes: float64(2), int64(7), object(11)
memory usage: 1.5+ MB
```

Statistical information of numeric features of the dataset

print(df.describe())

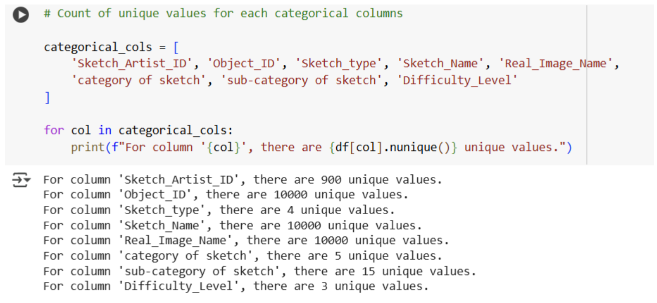


```
# Statistical information of numeric features of the dataset

print(df.describe())
```

```
       Sketch_width  Sketch_height  Real_image_Width  Real_image_height  \
count  10000.000000   10000.000000       10000.00000        10000.00000
mean     449.565200     447.860800        1204.37520          788.70610
std      201.181189     200.663221         411.26199          167.84224
min      100.000000     100.000000         500.00000          500.00000
25%      278.000000     277.000000         843.00000          643.00000
50%      448.000000     445.000000        1203.00000          787.00000
75%      624.000000     620.000000        1565.00000          934.00000
max      799.000000     799.000000        1919.00000         1079.00000

       No. of obj. in sketch  No of obj. in Real image  sketch_strokes  \
count           10000.000000              10000.000000    10000.000000
mean                2.486000                  2.475100       51.761500
std                 1.118986                  1.117723       27.557615
min                 1.000000                  1.000000        5.000000
25%                 1.000000                  1.000000       28.000000
50%                 2.000000                  2.000000       51.000000
75%                 3.000000                  3.000000       76.000000
max                 4.000000                  4.000000       99.000000

       matching_score  Human_Evaluation_Score
count    10000.000000            10000.000000
mean         0.747705                4.998120
std          0.144316                2.882608
min          0.500000                0.000000
25%          0.620000                2.500000
50%          0.750000                5.000000
75%          0.870000                7.500000
```

Count of unique values for each categorical columns

```
categorical_cols = [
        'Sketch_Artist_ID', 'Object_ID', 'Sketch_type',
    'Sketch_Name', 'Real_Image_Name',
        'category of sketch', 'sub-category of sketch',
    'Difficulty_Level'
]

for col in categorical_cols:
        print(f"For column '{col}', there are
    {df[col].nunique()} unique values.")
```

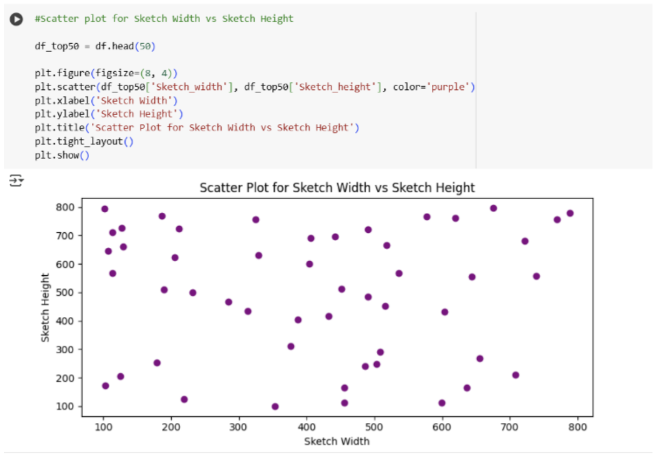


```
# Count of unique values for each categorical columns

categorical_cols = [
    'Sketch_Artist_ID', 'Object_ID', 'Sketch_type', 'Sketch_Name', 'Real_Image_Name',
    'category of sketch', 'sub-category of sketch', 'Difficulty_Level'
]

for col in categorical_cols:
    print(f"For column '{col}', there are {df[col].nunique()} unique values.")
```

```
For column 'Sketch_Artist_ID', there are 900 unique values.
For column 'Object_ID', there are 10000 unique values.
For column 'Sketch_type', there are 4 unique values.
For column 'Sketch_Name', there are 10000 unique values.
For column 'Real_Image_Name', there are 10000 unique values.
For column 'category of sketch', there are 5 unique values.
For column 'sub-category of sketch', there are 15 unique values.
For column 'Difficulty_Level', there are 3 unique values.
```

#Scatter plot for Sketch Width vs Sketch Height

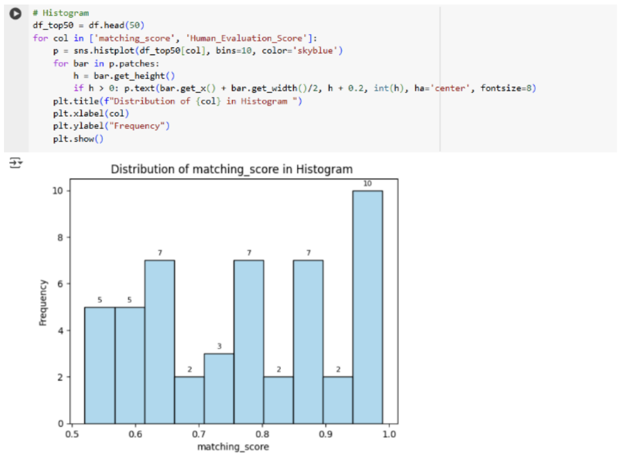
```
df_top50 = df.head(50)

plt.figure(figsize=(8, 4))
plt.scatter(df_top50['Sketch_width'],
        df_top50['Sketch_height'], color='purple')
plt.xlabel('Sketch Width')
plt.ylabel('Sketch Height')
plt.title('Scatter Plot for Sketch Width vs Sketch Height')
plt.tight_layout()
plt.show()
```



```
#Scatter plot for Sketch Width vs Sketch Height

df_top50 = df.head(50)

plt.figure(figsize=(8, 4))
plt.scatter(df_top50['Sketch_width'], df_top50['Sketch_height'], color='purple')
plt.xlabel('Sketch Width')
plt.ylabel('Sketch Height')
plt.title('Scatter Plot for Sketch Width vs Sketch Height')
plt.tight_layout()
plt.show()
```

# Histogram

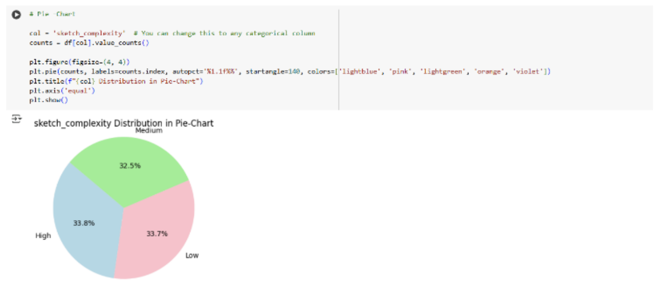
```
df_top50 = df.head(50)

for col in ['matching_score', 'Human_Evaluation_Score']:
    p = sns.histplot(df_top50[col], bins=10, color='skyblue')
    for bar in p.patches:
        h = bar.get_height()
         if h > 0: p.text(bar.get_x() + bar.get_width()/2, h + 0.2,
        int(h), ha='center', fontsize=8)
    plt.title(f"Distribution of {col} in Histogram ")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```



```
# Histogram
df_top50 = df.head(50)
for col in ['matching_score', 'Human_Evaluation_Score']:
    p = sns.histplot(df_top50[col], bins=10, color='skyblue')
    for bar in p.patches:
        h = bar.get_height()
        if h > 0: p.text(bar.get_x() + bar.get_width()/2, h + 0.2, int(h), ha='center', fontsize=8)
    plt.title(f"Distribution of {col} in Histogram ")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```

Pie -Chart

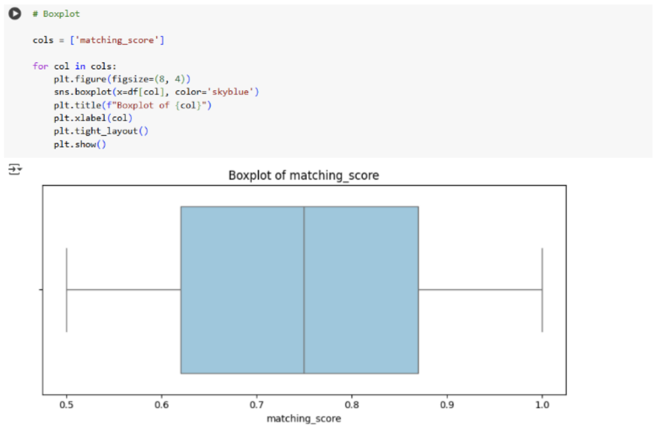
```
col = 'sketch_complexity'  # You can change this to any categorical column
counts = df[col].value_counts()

plt.figure(figsize=(4, 4))
plt.pie(counts, labels=counts.index, autopct='%1.1f%%', startangle=140, colors=['lightblue', 'pink', 'lightgreen', 'orange', 'violet'])
plt.title(f"{col} Distribution in pie Chart")
plt.axis('equal')
plt.show()
```



```
# Pie -Chart

col = 'sketch_complexity'  # You can change this to any categorical column
counts = df[col].value_counts()

plt.figure(figsize=(4, 4))
plt.pie(counts, labels=counts.index, autopct='%1.1f%%', startangle=140, colors=['lightblue', 'pink', 'lightgreen', 'orange', 'violet'])
plt.title(f"{col} Distribution in Pie-Chart")
plt.axis('equal')
plt.show()
```

Boxplot

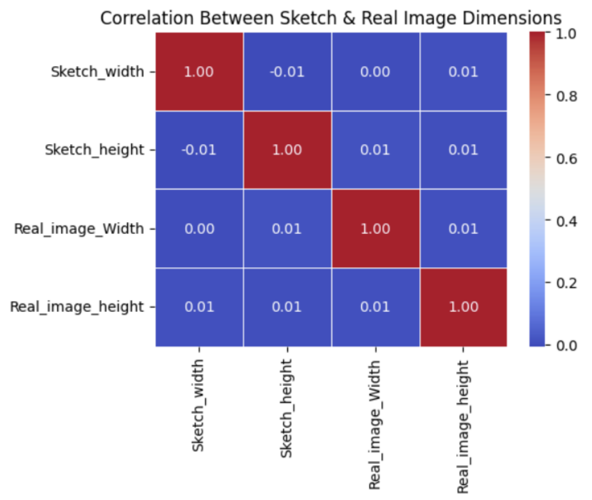
```
cols = ['matching_score']

for col in cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[col], color='skyblue')
    plt.title(f"Boxplot of {col}")
    plt.xlabel(col)
    plt.tight_layout()
    plt.show()
```



```
# Boxplot

cols = ['matching_score']

for col in cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[col], color='skyblue')
    plt.title(f"Boxplot of {col}")
    plt.xlabel(col)
    plt.tight_layout()
    plt.show()
```

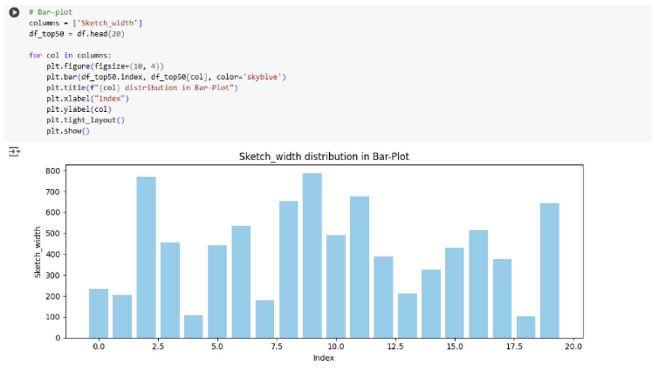
# Correlation bewtween Sketch_width, Sketch_height, Real_image_Width & Real_image_height.

```
cols = ['Sketch_width', 'Sketch_height', 'Real_image_Width', 'Real_image_height']

correlation_matrix = df[cols].corr()

plt.figure(figsize=(6, 5))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Between Sketch & Real Image Dimensions")
plt.tight_layout()
plt.show()
```



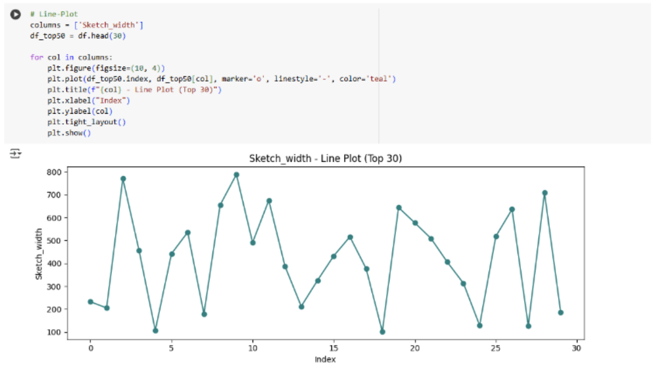
# Bar-plot

```
columns = ['Sketch_width']
df_top50 = df.head(20)

for col in columns:
    plt.figure(figsize=(10, 4))
    plt.bar(df_top50.index, df_top50[col], color='skyblue')
    plt.title(f"{col} distribution in Bar-Plot")
    plt.xlabel("Index")
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()
```



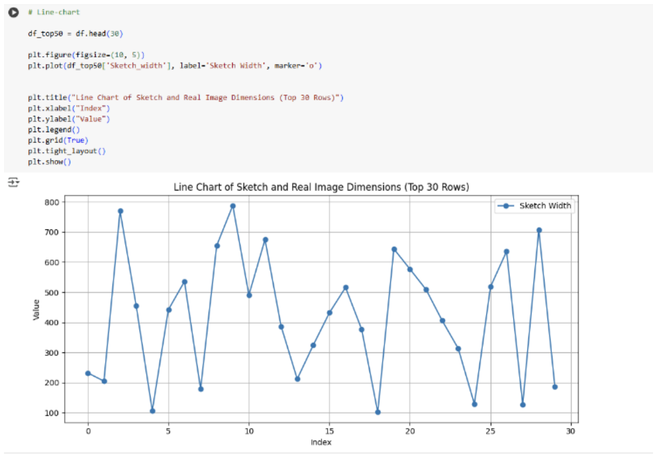
```
# Bar-plot
columns = ['Sketch_width']
df_top50 = df.head(20)

for col in columns:
    plt.figure(figsize=(10, 4))
    plt.bar(df_top50.index, df_top50[col], color='skyblue')
    plt.title(f"{col} distribution in Bar-Plot")
    plt.xlabel("Index")
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()
```

```
# Line-Plot
columns = ['Sketch_width']
df_top50 = df.head(30)

for col in columns:
    plt.figure(figsize=(10, 4))
    plt.plot(df_top50.index, df_top50[col], marker='o', linestyle='-', color='teal')
    plt.title(f"{col} - Line Plot (Top 30)")
    plt.xlabel("Index")
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()
```



```
# Line-Plot
columns = ['Sketch_width']
df_top50 = df.head(30)

for col in columns:
    plt.figure(figsize=(10, 4))
    plt.plot(df_top50.index, df_top50[col], marker='o', linestyle='-', color='teal')
    plt.title(f"{col} - Line Plot (Top 30)")
    plt.xlabel("Index")
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()
```

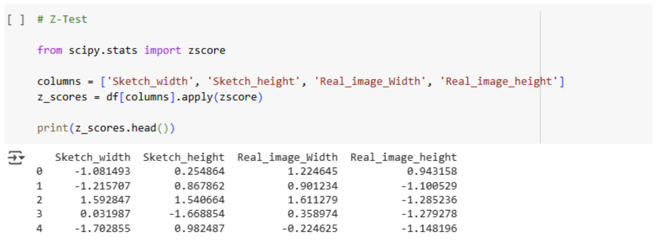
```
# Line-chart

df_top50 = df.head(30)

plt.figure(figsize=(12, 6))
plt.plot(df_top50['Sketch_width'], label='Sketch Width', marker='o')

plt.title("Line Chart of Sketch and Real Image Dimensions (Top 30 Rows)")
plt.xlabel("Index")
plt.ylabel("Value")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
# Line-chart

df_top50 = df.head(30)

plt.figure(figsize=(10, 5))
plt.plot(df_top50['Sketch_width'], label='Sketch Width', marker='o')


plt.title("Line Chart of Sketch and Real Image Dimensions (Top 30 Rows)")
plt.xlabel("Index")
plt.ylabel("Value")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Line Chart of Sketch and Real Image Dimensions (Top 30 Rows)
Sketch Width
Value
Index

```
# Z-Test

from scipy.stats import zscore

columns = ['Sketch_width', 'Sketch_height', 'Real_image_Width', 'Real_image_height']
z_scores = df[columns].apply(zscore)

print(z_scores.head())
```



```
[ ] # Z-Test

    from scipy.stats import zscore

    columns = ['Sketch_width', 'Sketch_height', 'Real_image_Width', 'Real_image_height']
    z_scores = df[columns].apply(zscore)

    print(z_scores.head())
```

```
   Sketch_width  Sketch_height  Real_image_Width  Real_image_height
0     -1.081493       0.254864          1.224645           0.943158
1     -1.215707       0.867862          0.901234          -1.100529
2      1.592847       1.540664          1.611279          -1.285236
3      0.031987      -1.668854          0.358974          -1.279278
4     -1.702855       0.982487         -0.224625          -1.148196
```

Linkedin Link:

https://www.linkedin.com/posts/ankitkumar57_machinelearning-deeplearning-sketchtoimage-activity-7316449567485448192-Bldd?utm_source=share&utm_medium=member_desktop&rcm=ACoAAEga1JgB2G0gy3GeA0kJ0GEOmhBH9dWV_bU

Github Link:

https://github.com/kumarankit79/Project-on-Image-to-image-translation