People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Science and Technology Houari Boumediene

Faculty of Computer Science
Department of Artificial Intelligence

Project Report No. 1

Module

Atelier Créatif

Specialty

Visual Computing

Report prepared by:

- Lynda SAYOUD.
- Fatima BENAZZOU.
- Lina MALLEK.
- Rayan Ibrahim BENATALLAH.

Sommaire

Table des figures

Liste des tableaux

Introduction

In the realm of image synthesis, Image Analogies emerges as a pivotal and innovative approach, offering a unique perspective on transforming images. Analogies, in essence, involve comparing structures by leveraging properties and relationships among objects in a '**source**' structure to infer those in a '**target**' structure **[1]**. The ability to transfer the style and structure of one image to another is a captivating prospect.

This report delves into the algorithm's focus on generating a new filtered image (B') from a pair of source images (A and A') and an unfiltered target image (B), aiming to find an 'analogous' image B' similar to the transformation between A' and A. Image Analogies not only enables the creation of new images by transferring features from a source to a target but also presents an innovative method for enhancing and applying diverse image filters. This approach simplifies the creation of high-quality textures with improved consistency, allowing for the transfer of textures and artistic styles between images. Its application extends to the realm of digital art, emphasizing a shift from selecting filters explicitly to learning and applying them using analogy.

This report is inspired by the research paper 'Image Analogies' authored by Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin.

For this, we divide our work as follows:

- **Part I -Image Analogy:** In the first step, we thoroughly study the algorithm. After that, we put it into action using existing open-source code. We then conduct tests to see how well it works and evaluate its performance.

- **Part II - Motion Sequence Transformation:** In this phase, the objective is to find the transformation from one moving image sequence to another, with a focus on the analogy within the context of the person depicted. Two approaches are considered: a straightforward application of image analogies and the use of Deep Learning techniques. Our choice aligns with leveraging Deep Learning.

Part I- Image Analogie

I.1 Introduction

In this part, we'll discuss the algorithm presented in the research paper **[1]**. We began by studying it, then implemented it by choosing to fix and enhance an open-source version. The process involved careful understanding, adjustments, and ultimately, testing to evaluate its effectiveness.

I.2 Study of the Image Analogy Algorithm

The paper provides two sets of instructions (pseudo-code) that we need to follow to create a filtered target image, B', using input images A, A', and B. Here's a closer look at what we're dealing with: **A** is **the original unfiltered source** image, **A'** is **the filtered source** image, and **B** is **the target unfiltered** image. Our goal is to synthesize the **filtered target** image, **B'**, Following the two sets of instructions depicted in Figures 1 and 2 below:

function CREATEIMAGEANALOGY(A, A', B):
    Compute Gaussian pyramids for A, A', and B
    Compute features for A, A', and B
    Initialize the search structures (e.g., for ANN)
    for each level ℓ, from coarsest to finest, **do**:
        for each pixel $q \in B'_\ell$, in scan-line order, **do**:
            $p \leftarrow$ BESTMATCH(A, A', B, B', s, ℓ, q)
            $B'_\ell(q) \leftarrow A'_\ell(p)$
            $s_\ell(q) \leftarrow p$
    return B'_L

Figure I.2-1:Algorithm 'Create Image Analogy' ***[1].***

function BESTMATCH(A, A', B, B', s, ℓ, q):
    $p_{app} \leftarrow$ BESTAPPROXIMATEMATCH(A, A', B, B', ℓ, q)
    $p_{coh} \leftarrow$ BESTCOHERENCEMATCH(A, A', B, B', s, ℓ, q)
    $d_{app} \leftarrow \|F_\ell(p_{app}) - F_\ell(q)\|^2$
    $d_{coh} \leftarrow \|F_\ell(p_{coh}) - F_\ell(q)\|^2$
    if $d_{coh} \leq d_{app}(1 + 2^{\ell - L}\kappa)$ **then**
        return p_{coh}
    else
        return p_{app}

Figure I.2-2:function 'BestMatch' used in Creating Image Analogy ***[1].***

I.2.1 First Step: Gaussian Pyramid – Multi Scale

The first step of the algorithm is computing the gaussian pyramid of our three inputs: the original one (A), the filtered version (A'), and the target one (B).

Gaussian Pyramid is the representative method to build a multi-scale representation of an image. It consists of two steps:

1- Applying Gaussian filter: smoothing to remove high-frequency components that could result aliasing, and

2-Down-sampling: reduce the image size by half. The complete picture of Gaussian Pyramid is illustrated in Fig 3.

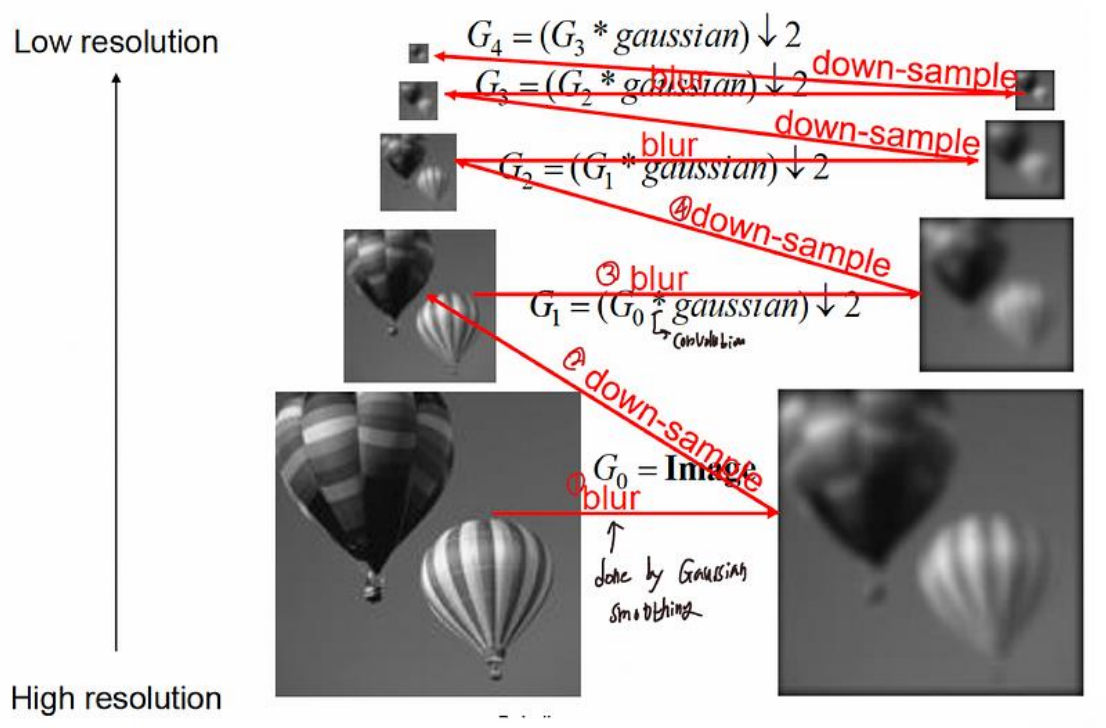

Figure I.2-3: Illustration of building the Gaussian Pyramid ***[2]***

I.2.2 Second Step: Features Extraction

This step involves extracting features from images A, A', and B. In this study, we focus on the luminance channel (Y in the YIQ color space) instead of the RGB space as a characteristic for feature matching. Therefore, a feature vector for a pixel refers to the luminance value of that pixel.

I.2.3 Third Step: Initializing an ANN structure

This step involves initializing an 'ANN' (Approximate Nearest Neighbors) search structure. An ANN search structure is employed to speed up the search for the best pixel matches between source and target images at different scales. It enables a swift comparison of pixel features and facilitates the selection of optimal matches. We also initialize a matrix s, that stores the mapping of each pixel in the target pair (B and B') to the corresponding pixel in the source pair (A and A').

I.2.4 Crucial Step: BEST MATCH sub-routine

The heart of the image analogies algorithm is the BEST M ATCH subroutine. This routine takes as input the three complete images A, A′ and B, along with the partially synthesized B′, the source information s, the level ℓ, and the pixel q being synthesized in B′**[1].**

for each level ℓ , moving from low-resolution to high-resolution:
For each pixel q in the synthesized image B':

p ← BESTMATCH(A, A′, B, B′,s, ℓ, q);

Best Match function in the following steps:

1. **Approximation phase:** we leverage the efficiency of the ANN method. To expedite the search process, we integrate Principal Components Analysis (PCA), a technique that reduces the dimensionality of feature vectors. In this method, we pinpoint the nearest neighbor by considering the luminance values in the 5x5 neighborhood (or 3x3 in the smallest and coarsest level) around pixel q in image B. The similarity between the feature vectors of pixels in image B and those in image A is then measured using the L2 norm, also known as the Euclidean norm. This approach allows us to efficiently identify the most approximate pixel, termed "$\mathbf{p_{app}}$".

2. **Coherence Phase:** we search for a pixel, denoted as r*, within the neighborhood of a pixel q in the partially synthesized image B'. This pixel, when mapped to its corresponding location in A, minimizes the feature difference. The procedure returns s(r*) + (q - r*), providing the best pixel coherent with the already synthesized portion of B', where:

$$r^\star = \arg\min_{r \in N(q)} \|F_\ell(s(r) + (q - r)) - F_\ell(q)\|^2$$

*Figure I.2-4: Formula of pixel r**[1].***

- N(q) represents the neighborhood of already synthesized pixels adjacent to q in B'. This formula essentially returns the best pixel coherent with some previously synthesized portion of B' adjacent to q, aligning with the fundamental insight of Ashikhmin's method **[1]**.

This phase allows us to efficiently identify the most coherent pixel, termed "$\mathbf{p_{coh}}$".

3. Computing "d_{app}" and "d_{coh}":

According the paper **[1]** , FL(p) represents the concatenation of all feature vectors within the neighborhood of pixels around pixel p from images A and A', considering both the current level and the previous, smaller level. Similarly, FL(q) denotes the concatenation of feature vectors around pixel q from images B and B', where B' is only partially synthesized. To compute both distances, d_{app} and d_{coh}, we calculate the norm $||FL(p)-FL(q)||^2$ of the difference between these two concatenations of feature vectors for both images, using both candidate pixels p_{app} and p_{coh} respectively.

[1] The approximation approach, while not ensuring the closest match in terms of feature vectors, becomes significant due to the imperfect nature of the L2-norm in measuring perceptual similarity. To enhance visual coherence, we rescale the distance obtained from the approximate search using a coherence parameter κ. A higher κ value emphasizes coherence over accuracy in the synthesized image. To maintain coherence consistency across scales, we attenuate the coherence term by a factor of $2^{\ell-L}$, considering that pixel locations at coarser scales are spaced further apart than at finer scales. (L represents Lmax).

4.Comparing between the 2 distances:

if $d_{coh} \leq d_{app}(1 + 2^{\ell-L}\kappa)$ **then**
    return p_{coh}
else
    return p_{app}

Figure I.2-5: Comparison between the 2 distances ***[1]***

Afterward, the function returns either pixel p_{app} or p_{coh} in a variable p. The feature vector $B'_{\ell}(q)$ is then assigned the feature vector $A'_{\ell}(p)$ from the closest-matching pixel p, and the pixel that achieved the best match is recorded in s(q). This process involves retrieving the feature vector of the best-matching pixel in A' and copying it to our target image B' for the current level, at the current pixel q. Finally, we get our synthesized image B'_L.

I.3 Implementation of the algorithm

To see the result, we implemented an open-source code available on GitHub. The code repository can be found: Open source Image Analogie

1. Reading our Input images : A, Ap and B .

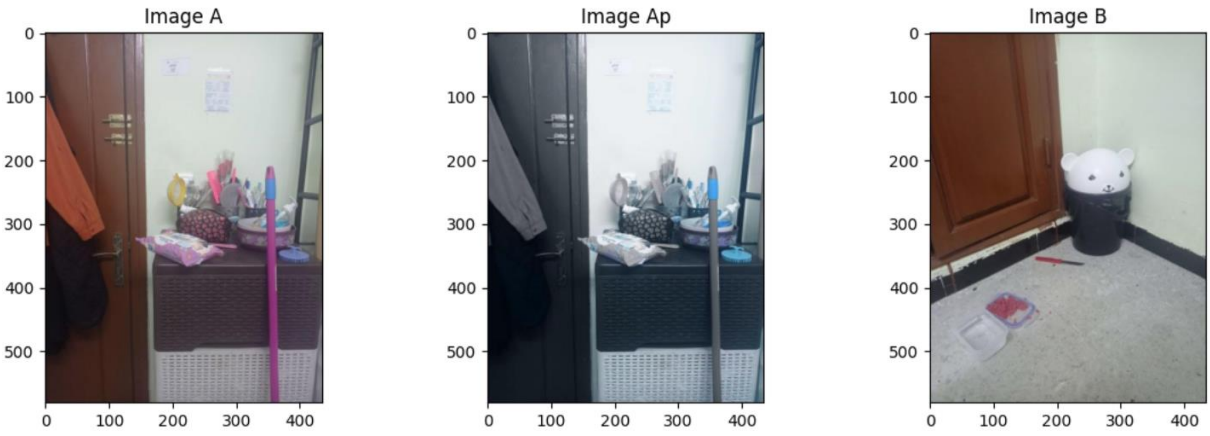

Figure I.3-1:Inputs of our implementation

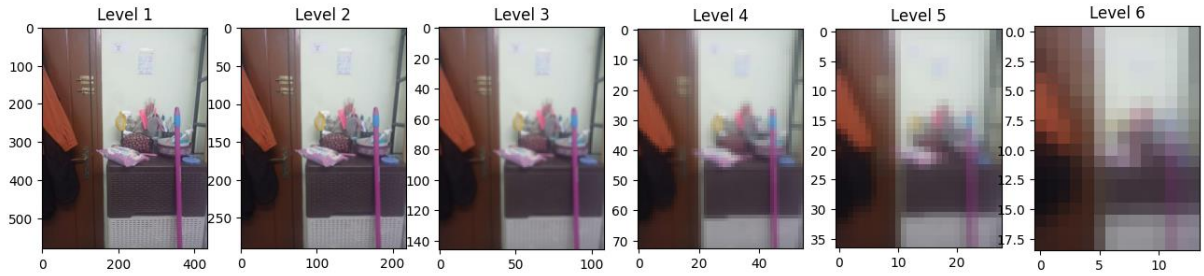

Figure I.3-2: Gaussian pyramid of input A

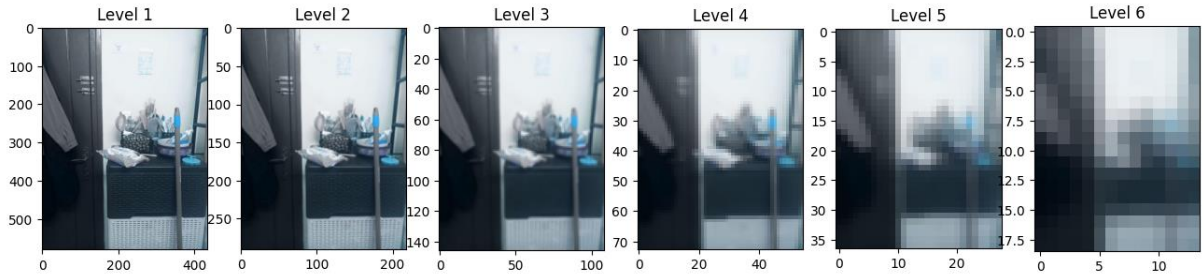

Figure I.3-3: Gaussian pyramid of input A`

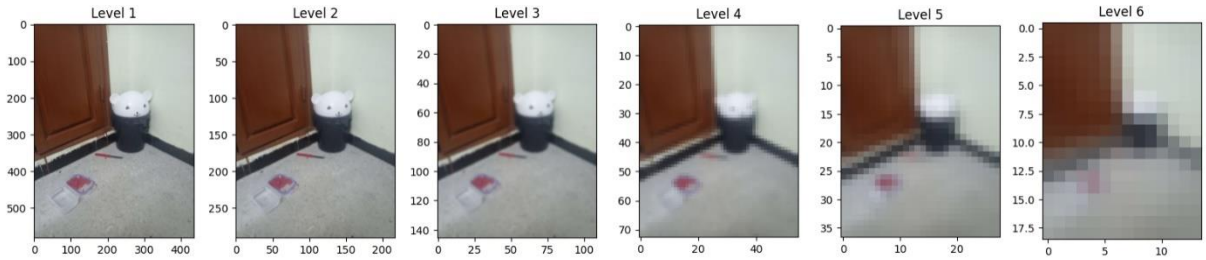

Figure I.3-4: Gaussian pyramid of input B

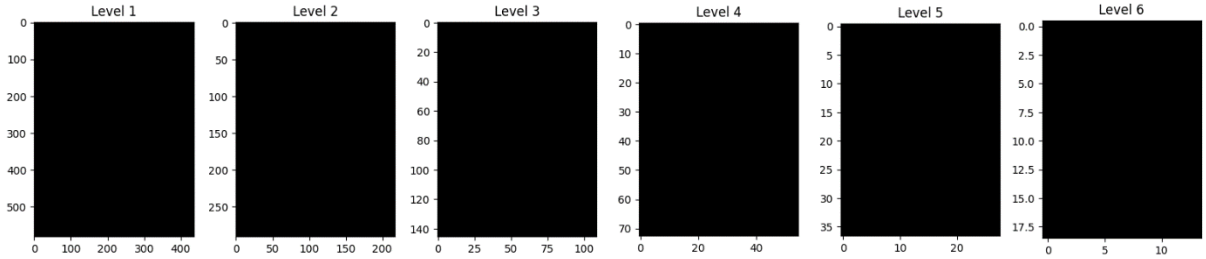

Figure I.3-5: Intial state of B`

- The search in the bestmatch function is implemented thanks to the **pyflann library** .

The **pyflann library** is the python bindings for FLANN - Fast Library for Approximate Nearest Neighbors **[3]** . It offers a variety of algorithms for different types of data and distance metrics, making it a valuable tool for tasks such as machine learning, computer vision, and data mining. We used "k_means" and "kd_tree" as algorithms from this library for nearest neighbor search.

```
""" Begin Neighbor Search Methods """
if method == 'pyflann_kmeans':
    flann = pyflann.FLANN()

    print("Building FLANN kmeans index for size:", A_f.size, "for A size", Ap_L[lvl].size)
    flann_p = flann.build_index(A_f_2d, algorithm="kmeans", branching=32, iterations=-1, checks=16)
    print("FLANN kmeans index done...")

elif method == 'pyflann_kdtree':
    flann = pyflann.FLANN()

    print("Building FLANN kdtree index for size:", A_f.size, "for A size", Ap_L[lvl].size)
    flann_p = flann.build_index(A_f_2d, algorithm="kdtree")
    print("FLANN kdtree index done...")
```

Figure I.3-6: K_means and kd_tree.

And as we mentioned in section I.2 , the YIQ color space is used for our images, while the original data is in RGB format. To bridge this gap, we employ the rgb2yiq function, as illustrated in the figure I-12 below. This function facilitates the conversion from RGB to YIQ.

```
def rgb2yiq(image, remap=False, remap_target=None, feature='y'):
    yiq_xform = np.array([[0.299, 0.587, 0.114], #ROUGE
                          [0.596, -0.275, -0.321], #Vert
                         [0.212, -0.523, 0.311]]) #bleu
    yiq = np.dot(image, yiq_xform.T.copy())

    if remap:
        remap_y(image, remap_target)
    if feature == 'y':
        return yiq[:,:,0]
    elif feature == 'yiq':
        return yiq
```

Figure I.3-7:transformation from RGB to YIQ.

So when we choosed :
search_method = 'pyflann_kdtree'

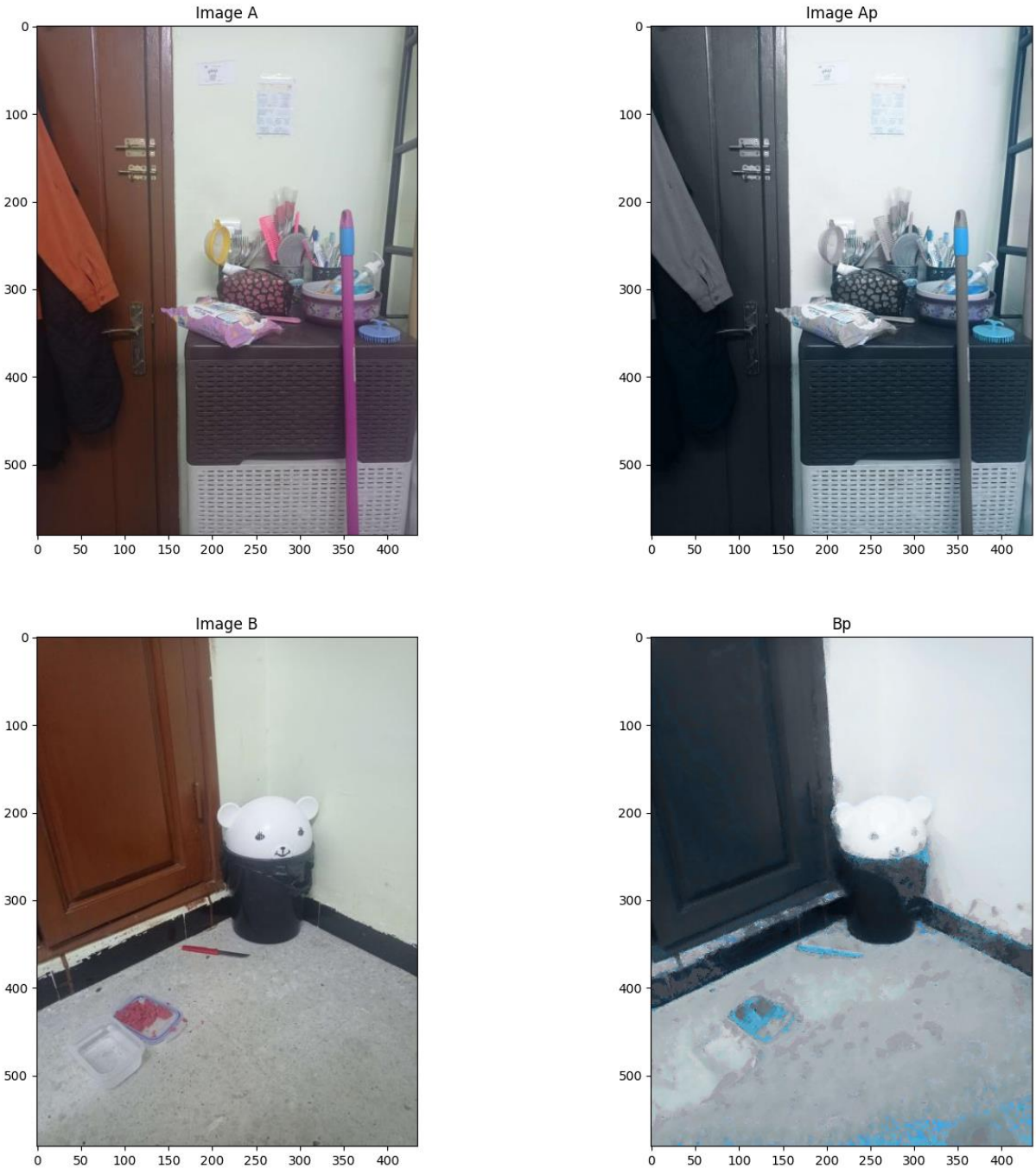

Figure I.3-8:Result of the implementation of Image Analogy Algorithm using Kd_Tree

```
Total execution time: 46.83909869194031 seconds
```

- search_method = 'pyflann_kmeans'

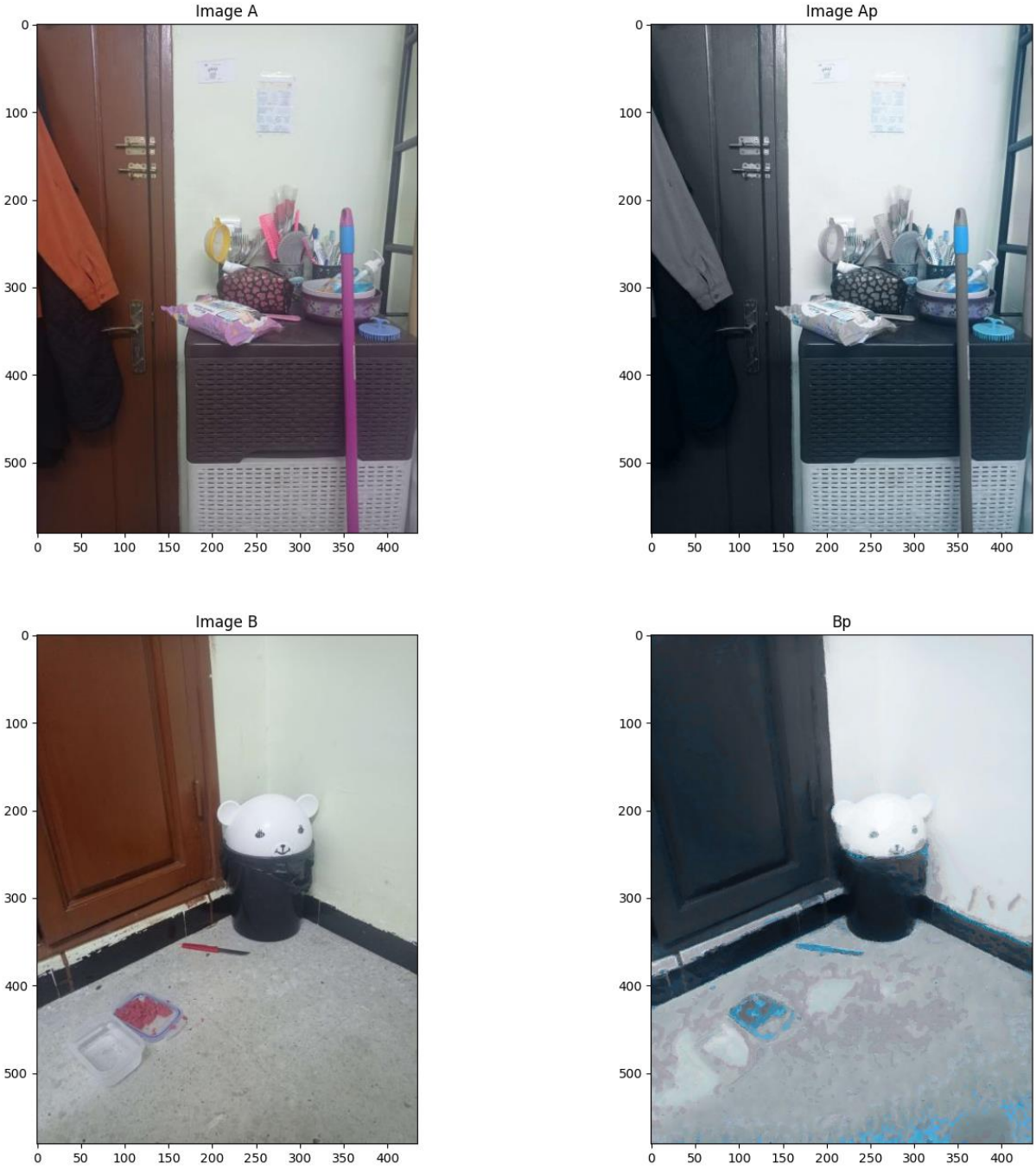

Figure I.3-9: Result of Image analogy Algorithme using K_means

```
Total execution time: 306.34448075294495 seconds
```

Part II- Motion Sequence Transformation

II.1 Introduction

In this part, we delve into our specific contributions, aiming to discover the transformation between two sequences of moving images, with a particular emphasis on the analogy within the depicted individuals. Our approach centers on the application of deep learning techniques to yield meaningful results in our pursuit.

II.2 Data Acquisition

We selected two videos featuring different individuals in motion. Using a dedicated algorithm, we processed the videos to extract frames (you can find the algorithm in our GitHub repository of the project). Subsequently, we performed data cleaning to refine the frames, ensuring that the movement exhibited by the person was accurately captured. This involved selecting a specific frame and discarding others to obtain a clear representation of the person's motion. Additionally, we standardized the resolution of all frames, resizing them to a common dimension. This resolution standardization is beneficial for model training, ensuring consistency in the input data.

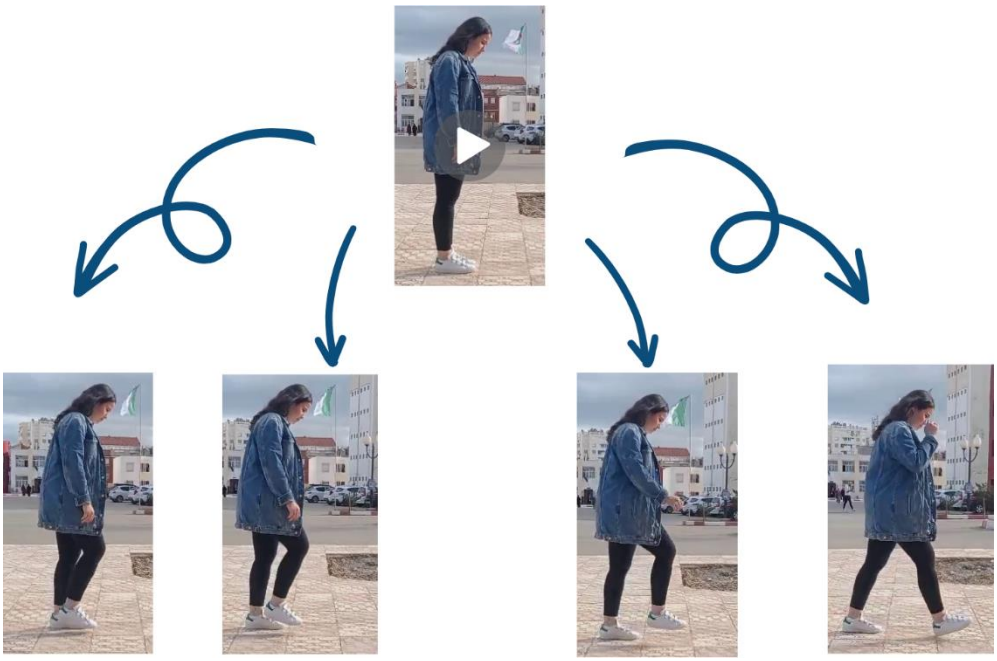

Figure II.2-1: Data Acquizition

To achieve our goal and respond to the problematic given, we employed several deep learning techniques, each contributing to the overall analysis. Specifically, we utilized two autoencoders, a single autoencoder, and a Variational Autoencoder (VAE). We will delve into the details of each technique, elucidating their roles and contributions to our study.

II.3 2-AutoEncoder (AE)

What's an AE ?

Autoencoder is used to learn efficient embeddings of unlabeled data for a given network configuration. The autoencoder consists of two parts, an encoder, and a decoder. The encoder compresses the data from a higher-dimensional space to a lower-dimensional space (also called the latent space), while the decoder does the opposite i.e., convert the latent space back to higher-dimensional space. The decoder is used to ensure that latent space can capture most of the information from the dataset space, by forcing it to output what was fed as input to the decoder **[4].**

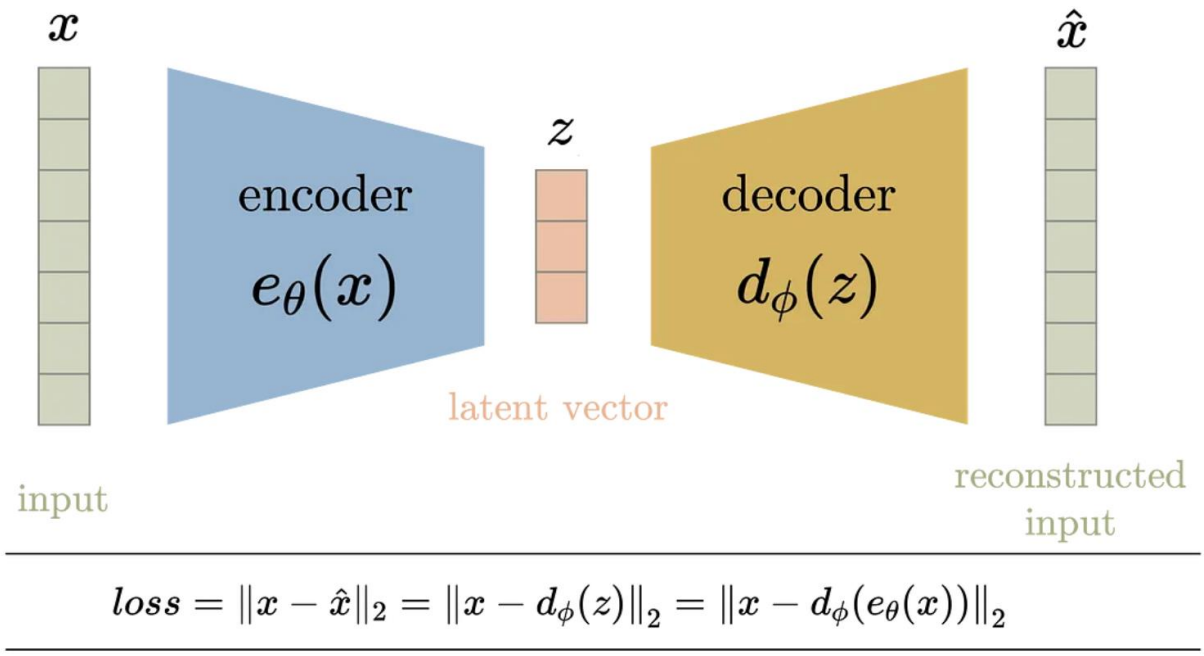

*Figure II.3-1: AutoEncoder **[4]**.*

Our initial idea was to use two **autoencoders** – one for an input frame captured at time t1 and the other after a specific movement at time t2. The concept involved combining the encoder of the first autoencoder with the decoder of the second. Unfortunately, this approach was difficult to be implemented, after many attempts, did not give any results.

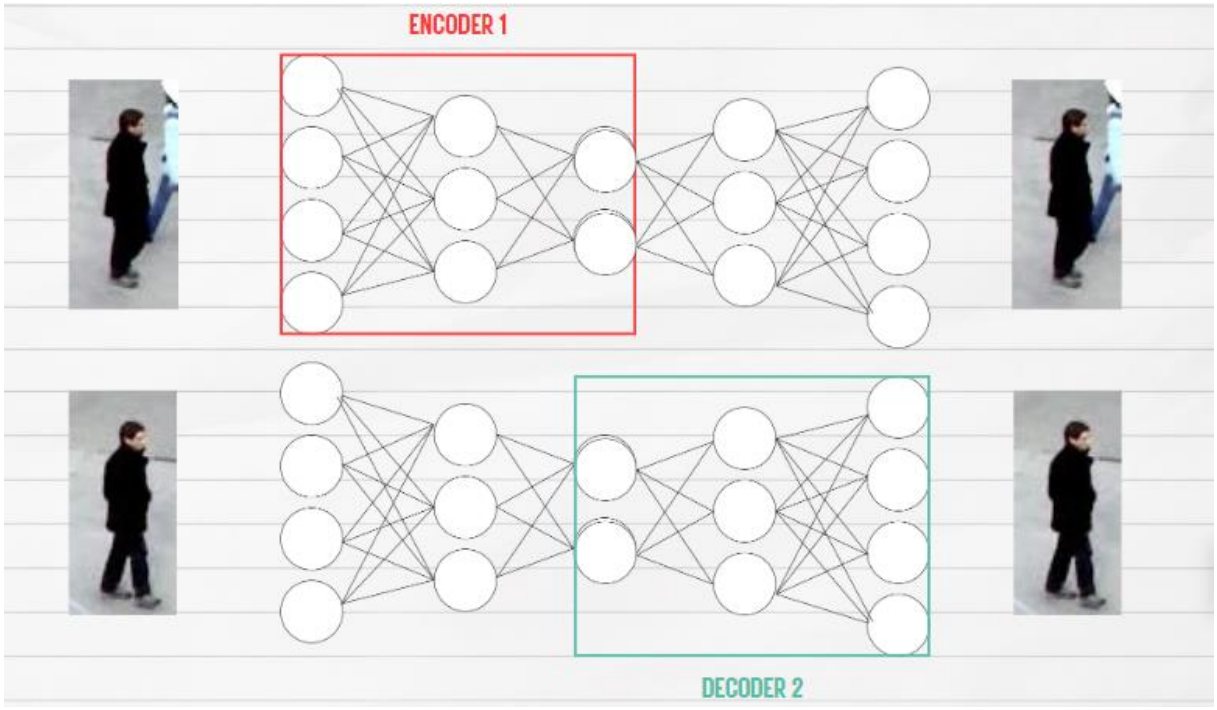

Figure II.3-2: Overview of step 1 of 2 AE.

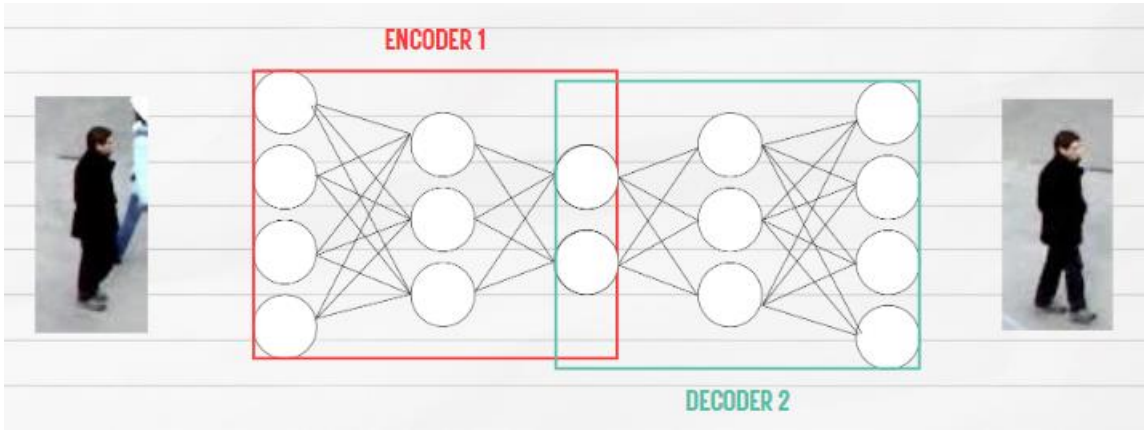

Figure II.3-3: Overview of implementation of 2 AE

Obstacles of this approach:

- The approach is challenging to implement, particularly in the manipulation of combining layers between the first and second autoencoders.
- It requires a long training period to achieve optimal performance.
- This technique demands a colossal amount of data to may be generalize effectively.

For more details about the code, you can find it in our GitHub via the following link:

Project 1- Image Analogie

II.4 1-AutoEncoder

After some research, we discovered that one of the applications of autoencoders is in the domain adaptation context **[5]**. This involves providing the autoencoder with input data x and expecting it to generate different output data y, deviating from the standard

reconstruction of the same input.

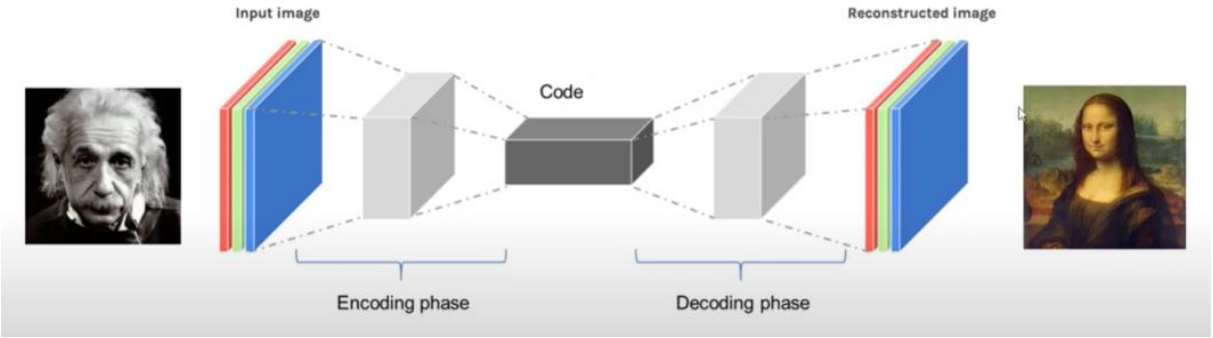

Figure II.4-1: Domain Adaptation of AE [5]

II.4.1 Overview of the architecture of our autoencoder:

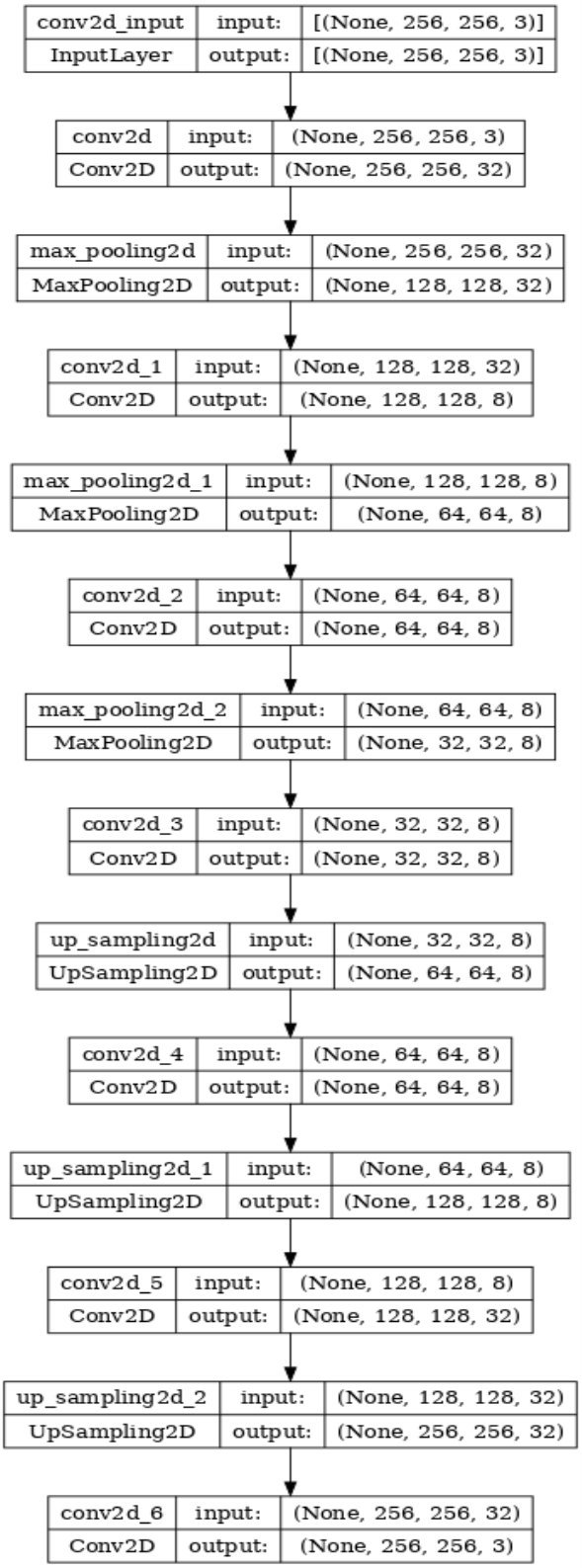

Figure II.4-2: Architecture of our autoencoder.

To implement it, we used our own dataset, where each input is present in both the input and target image folders. This setup ensures that each input is associated with its corresponding target, forming a sequence of images.

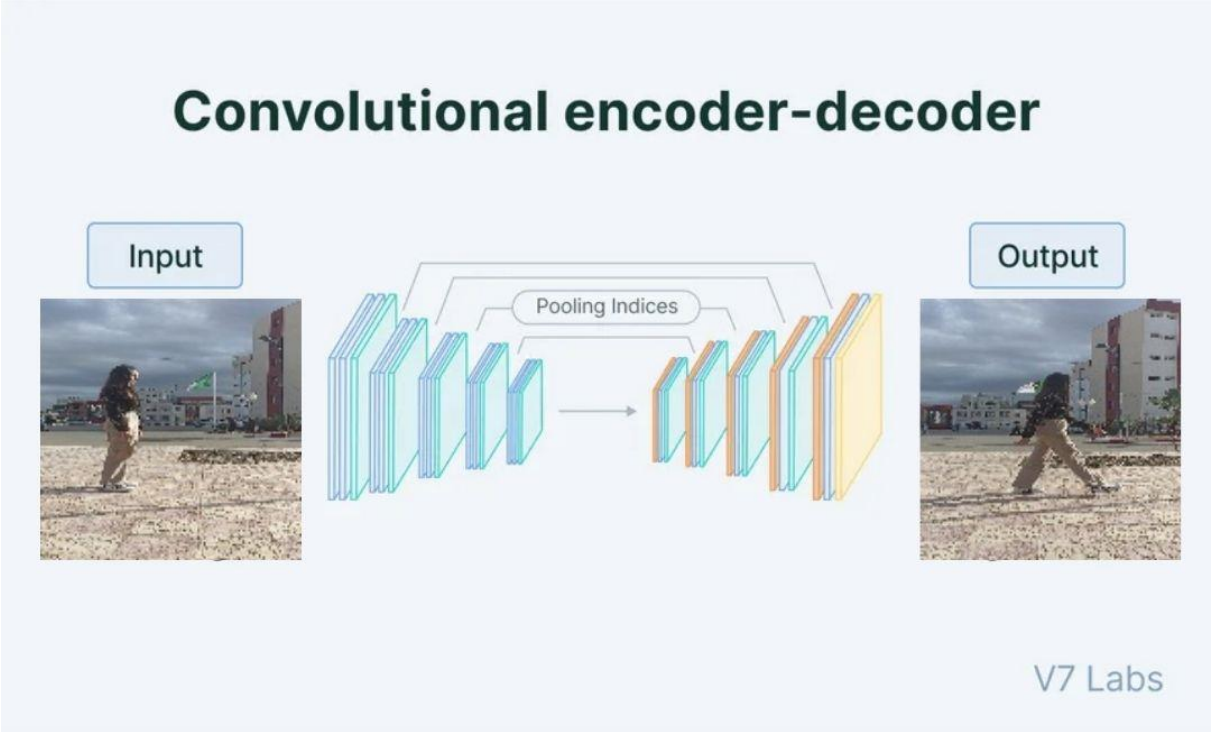

Figure II.4-3: Our implementation with our dataset.

II.4.2 Results with an existing dataset:

After 1000 Epochs

Figure II.4-4: Result of Implementing 1AE for 1000 epochs.

After 5000 Epochs

Figure II.4-5: Result of Implementing 1AE for 5000 epochs

II.4.3 Results with our dataset

After 5000 epochs

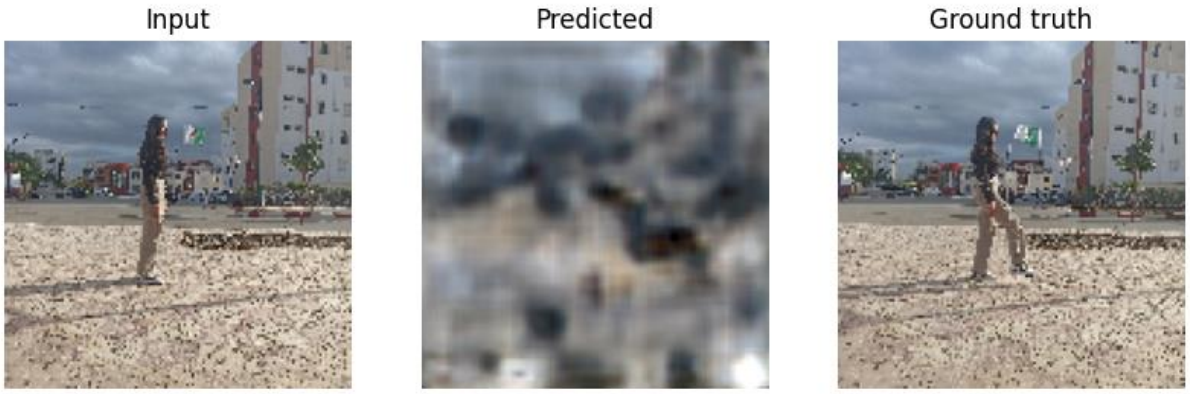

Figure II.4-6: Result with our dataset after 5000 Epochs

After 10000 Epochs

Figure II.4-7: Result with our dataset after 10000 Epochs

Conclusion

We inferred that there is a linear relationship between the size of the dataset and the number of epochs.

Following these results, we conducted further research to enhance our outcomes. Two methods emerged: the use of Variational Autoencoders (VAEs) and Conditional Generative Adversarial Networks (cGANs). The cGANs introduce a conditional mechanism, allowing the model to generate outputs based on specific conditions or labels, providing more control over the synthesis process. On the other hand, VAEs that we adopted in our following approach to improve the performance of our model.

II.5 Image Analogy (Extraction of Key points)

In our pursuit of refining image synthesis through analogy, we explored a method centered around pose alignment for enhanced predictions. The process involves:

1. **Movement Extraction with OpenPose**:

Leveraging OpenPose, intricate movements are extracted from source images (A and B).

2. **Key Point Extraction for Movements**:

Identifying crucial key points of movements for both images (A and B).

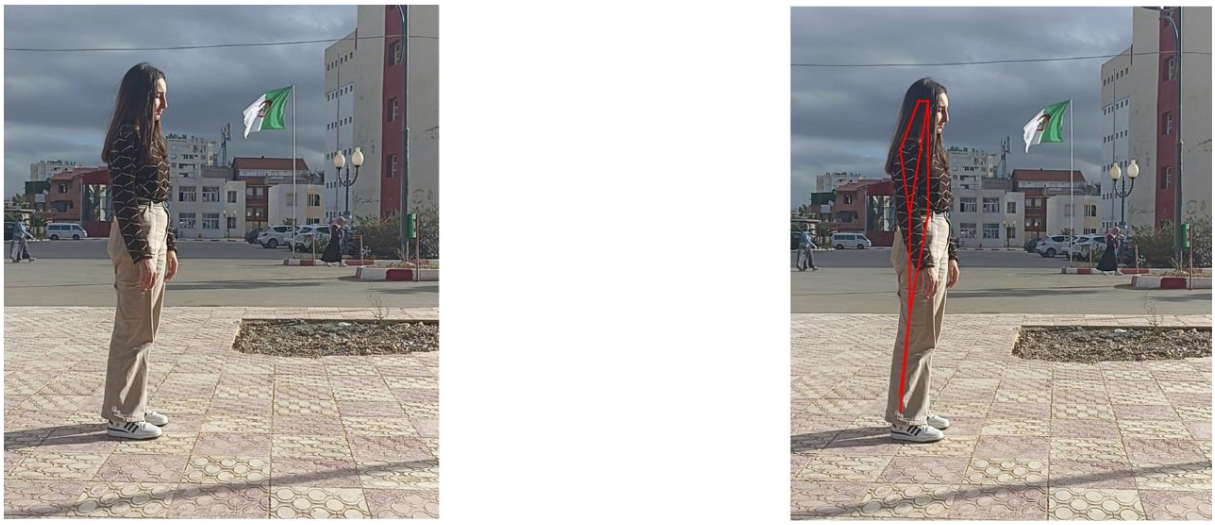

Figure II.5-1:Extracting Key Movement Points for ***A.***

*Figure II.5-2: Extracting Key Movement Points for **B***

3. Fusion of Movement Key Points:

fusion of key points from both images to create a complete representation.

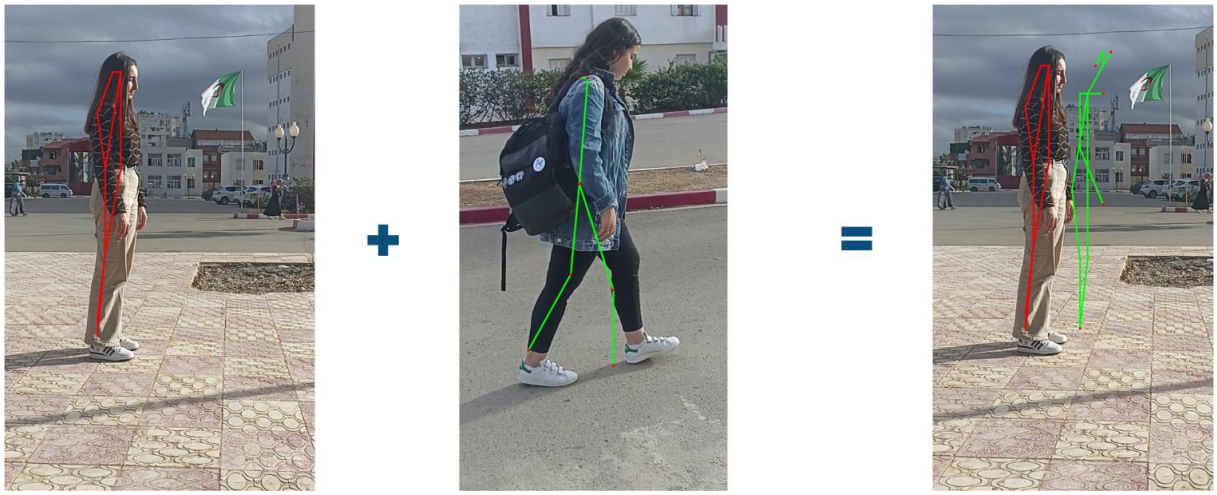

Figure II.5-3: Merge the Movements of A and B

4. Recreating Image B with Altered Pose:

The algorithm generates a new rendition of Image B, incorporating fused key points for a transformed pose.

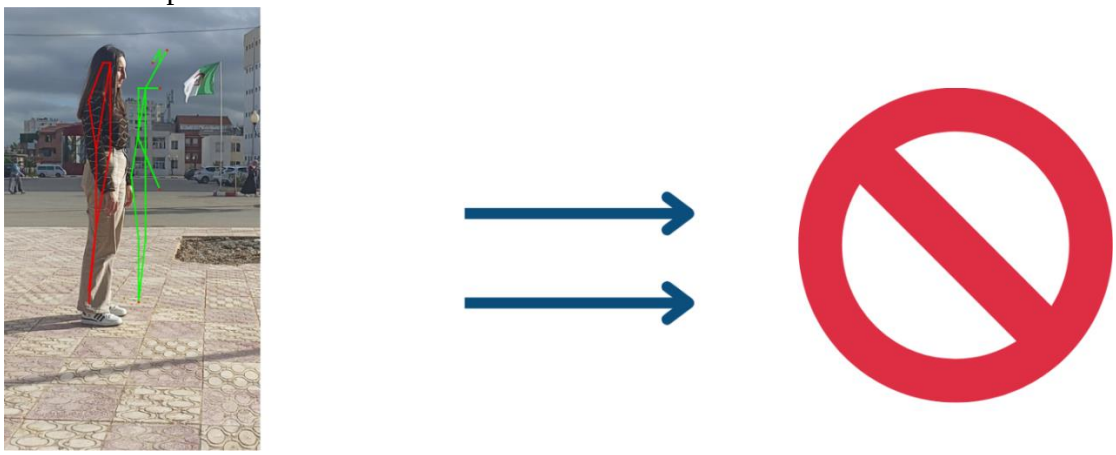

Figure II.5-4: Movements fusion: No results.

While these methods were explored, the results did not meet expectations, prompting further investigation for improved outcomes.

II.6 Variational Autoencoders (VAEs)

II.6.1 What's it?

Variational autoencoder addresses the issue of non-regularized latent space in autoencoder and provides the generative capability to the entire space. The encoder in the AE outputs latent vectors. Instead of outputting the vectors in the latent space, the encoder of VAE outputs parameters of a pre-defined distribution in the latent space for every input. The VAE then imposes a constraint on this latent distribution forcing it to be a normal distribution. This constraint makes sure that the latent space is regularized **[4].**

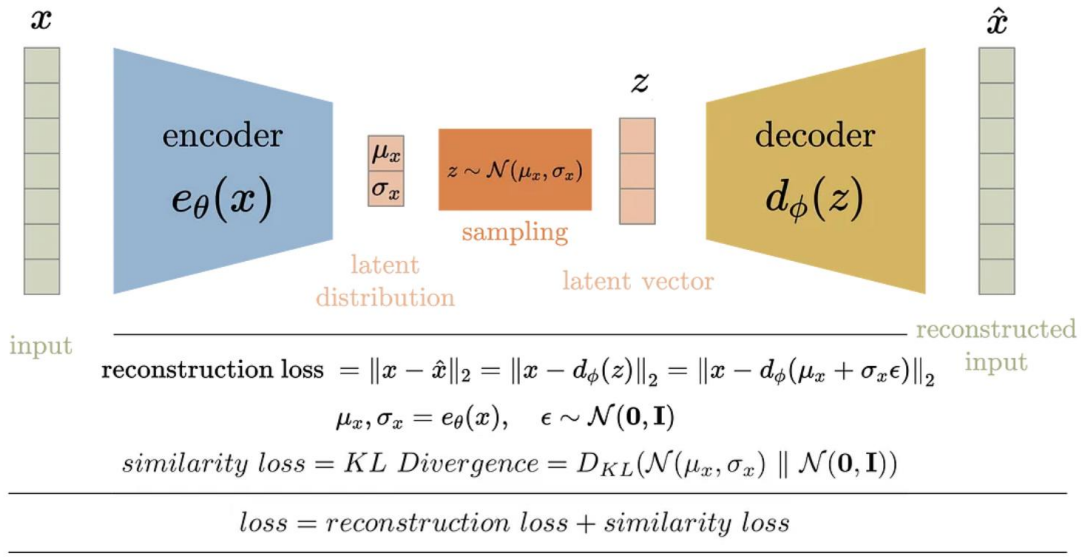

Figure II.6-1: Variational Autoencode VAE

II.6.2 Loss function [4]

The loss function is defined by the VAE objectives. VAE has two objectives

1. Reconstruct the input
2. Latent space should be normally distributed.

Hence the training loss of VAE is defined as the sum of these the ***reconstruction loss*** and the ***similarity loss***. The reconstruction error, just like in AE, is the mean squared loss of the input and reconstructed output. The similarity loss is the KL divergence between the latent space distribution and standard gaussian (zero mean and unit variance). The loss function is then the sum of these two losses.

II.6.3 Brief Comparison between AE and VAE

Why we moved to VAE?

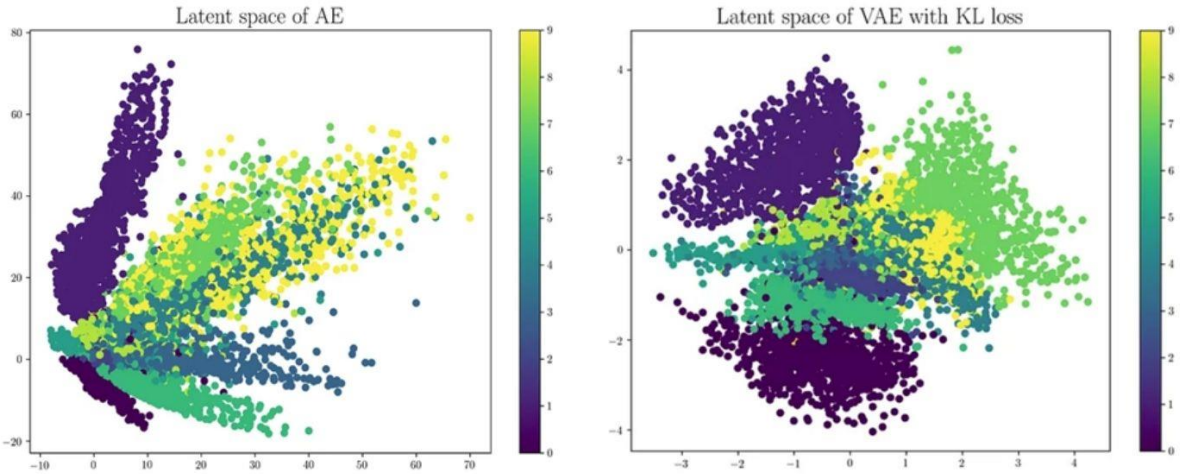

Figure II.6-2: Latent space of AE and VAE [4]

Autoencoder (AE)	**Variational Autoencoder (VAE)**
Used to generate a compressed transformation of input in a latent space	Enforces conditions on the latent variable to be the unit norm
The latent variable is not regularized	The latent variable in the compressed form is mean and variance
Picking a random latent variable will generate garbage output	The latent variable is smooth and continuous
The latent variable has a discontinuity	A random value of latent variable generates meaningful output at the decoder
Latent variable is deterministic values	The input of the decoder is stochastic and is

	sampled from a gaussian with mean and variance of the output of the encoder.
The latent space lacks the generative capability	Regularized latent space
	The latent space has generative capabilities.

Tableau 1: table of Comparison between AE and VAE .[4]

II.6.4 Building our VAE model

```
#VAE model
def build_vae(input_dim, latent_dim):
    # Encoder
    inputs = Input(shape=(input_dim,))
    h = Dense(256, activation='relu')(inputs)
    z_mean = Dense(latent_dim)(h)
    z_log_var = Dense(latent_dim)(h)

    # Sampling function
    def sampling(args):
        z_mean, z_log_var = args
        batch = K.shape(z_mean)[0]
        dim = K.int_shape(z_mean)[1]
        epsilon = K.random_normal(shape=(batch, dim))
        return z_mean + K.exp(0.5 * z_log_var) * epsilon

    # Use Lambda layer to create a layer from the sampling function
    z = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_var])

    # Decoder
    decoder_h = Dense(256, activation='relu')
    decoder_mean = Dense(input_dim, activation='sigmoid')
    h_decoded = decoder_h(z)
    x_decoded_mean = decoder_mean(h_decoded)
```

```python
#custom loss function
def vae_loss(x, x_decoded_mean):
    xent_loss = input_dim * binary_crossentropy(x, x_decoded_mean)
    kl_loss = -0.5 * K.sum(1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
    return xent_loss + kl_loss

# build + compile
vae = Model(inputs, x_decoded_mean)
vae.add_loss(vae_loss(inputs, x_decoded_mean))
vae.compile(optimizer='adam')

encoder = Model(inputs, z)
decoder_input = Input(shape=(latent_dim,))

_h_decoded = decoder_h(decoder_input)
_x_decoded_mean = decoder_mean(_h_decoded)
decoder = Model(decoder_input, _x_decoded_mean)

return vae, encoder, decoder
```

Figure II.6-3: Necessary functions to build VAE.

```python
latent_dim = 2
#  A VAE is a type of generative model that learns
# a probabilistic mapping between input data and a latent space, allowing for the generation of new data points.
# Build the VAE model
vae, encoder, decoder = build_vae(SIZE*SIZE*3, latent_dim)
```

Figure II.6-4: building VAE

II.6.5 Overview of the architecture of the VAE encoder:

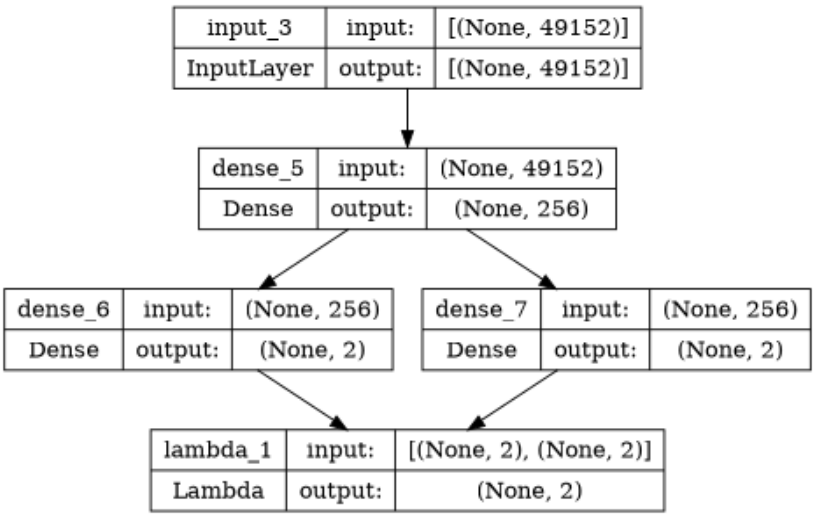

Figure II.6-5: Overview of the architecture of VAE encoder

II.6.6 Overview of the architecture of the VAE decoder:

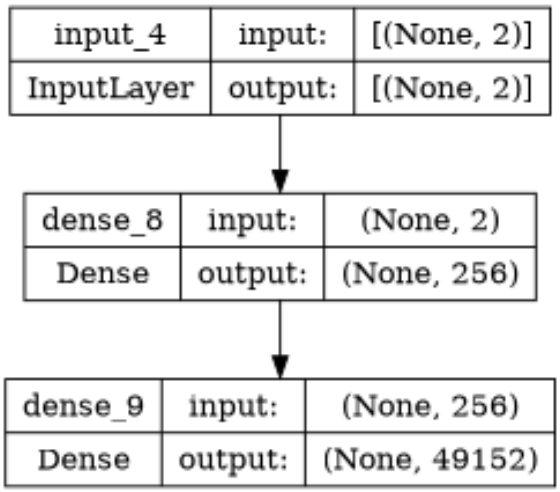

Figure II.6-6: Overview of the architecture of VAE decoder

Why using dimension 2 for the latent space?

Setting the latent space dimension to 2 in a VAE serves to create a compact and easily visualized representation of the data. The resulting 2D latent space facilitates a clear understanding of the model's learned features and supports intuitive exploration of the encoded information.

Our input image :

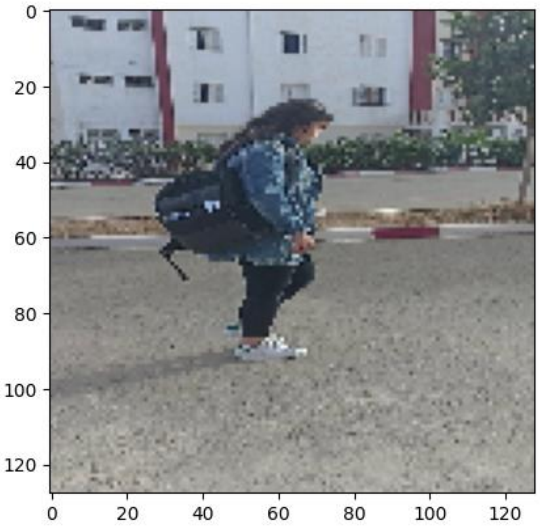

Figure II.6-7: One frame of our input data.

```python
# Generate new samples by sampling from the latent space
num_generated_samples = 100
random_latent_samples = np.random.normal(size=(num_generated_samples, latent_dim))
generated_images = loaded_decoder.predict(random_latent_samples)
frames = [generated_images[i].reshape((SIZE, SIZE, 3)).astype(np.uint8) for i in range(num_generated_samples)]

# Plot the generated images
plt.figure(figsize=(15, 3))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(generated_images[i].reshape((SIZE, SIZE, 3)))
    plt.title(f'Generated {i + 1}')
    #plt.savefig(f'generation{i + 1}.png')
    plt.axis('off')

#plt.savefig('generation.png')
plt.show()
```

Figure II.6-8: Code to generate new samples by sampling the latent space

In this code snippet, we aimed to generate new image samples using a pre-trained decoder from our VAE model. We specified the number of samples to be 100 and created random latent vectors by sampling from a normal distribution. These latent vectors were then fed into the loaded decoder, resulting in a set of generated images. Each generated image underwent reshaping to the specified dimensions and was converted to uint8 format for visualization. The matplotlib library was employed to create a compact display of five randomly selected generated images, showcasing a subset of the model's capacity to produce diverse and novel samples from the learned latent space. And The figure below displays the results of the generated images.

Figure II.6-9: Result of the generated images using our VAE model.

After seeing improvements with the VAE model, we wanted to dig deeper and find motion details in these images using a method called optical flow. Unfortunately, it didn't work well for us due to the fact that optical flow performs better with a static camera setup, but our camera moves around, so we couldn't extract the motion details we were hoping for.

II.7 Future work

As part of our future work, we plan to reconstruct our dataset under static camera conditions and explore the possibility of conditioning our VAE with motion vectors in the latent space. This adjustment may offer more promising results aligned with our objectives.

Bibliography

[1] Image analogies. A. Hertzmann, C. Jacobs, N. Oliver, B. Curless, and D. Salesin. SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, page 327--340. New York, NY, USA, ACM Press, (2001)

URL: https://mrl.cs.nyu.edu/publications/image-analogies/analogies-72dpi.pdf

[2] jun94, published in jun-devpBlog , (November 17,2020), '[CV] 4. Multi-Scale Representation (Gaussian and Laplacian Pyramid)',
URL : https://medium.com/jun94-devpblog/cv-4-multi-scale-representation-gaussian-and-laplacian-pyramid-527ca4c4831c [consulted : October 21, 2023]

[3]URL:https://pypi.org/project/pyflann/#:~:text=pyflann%20is%20the%20python%20bindings,Library%20for%20Approximate%20Nearest%20Neighbors.

[4] Aqeel Anwar (November 3, 2021), 'Difference between AutoEncoder (AE) and Variational AutoEncoder (VAE)' , URL:

https://towardsdatascience.com/difference-between-autoencoder-ae-and-variational-autoencoder-vae-ed7be1c038f2 [consulted: November 10, 2023]

[5] DigitalSreeni, (january 22,2020) , '89 - Applications of Autoencoders - Domain Adaptation', URL :
https://youtu.be/Te3YieMUYd8?si=UTtcdHO0FdLSL1c6 [Consulted: november 04, 2023].