Assignment 2

Generative Artificial Intelligence: CS F437

Submitted To

Prof. NL Bhanu Murthy

BITS Pilani Hyderabad Campus

Submitted By:

Name	**BITS ID**
Tarimala Vignesh Reddy	2021A7PS0234H
Aashish Chandra K	2021A7PS0467H
Harshit Agarwal	2021A7PS0247H

Birla Institute of Technology and Science, Hyderabad Campus, India

Contents

Part-A-I: Implementing VAE

Methodology

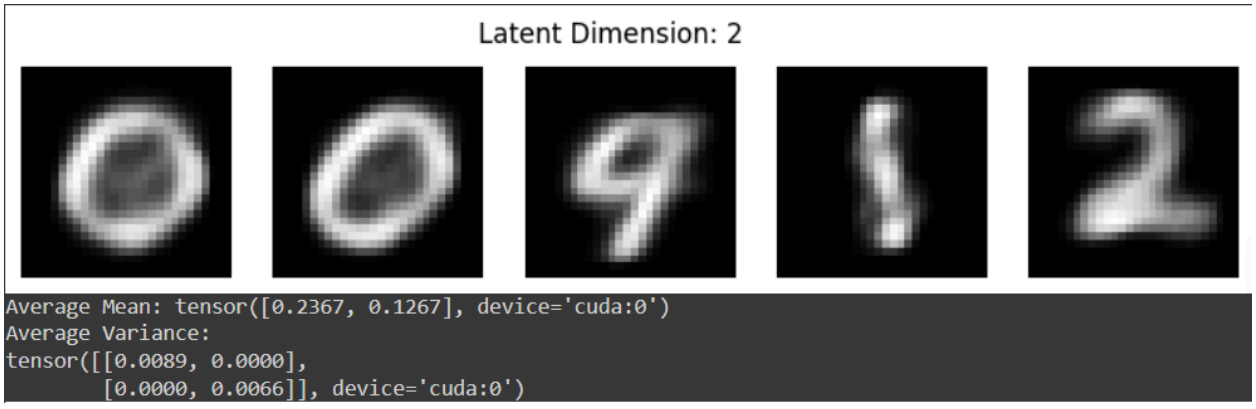
Variational Autoencoders (VAEs) are a type of artificial neural network used for image reconstruction, among other applications. They belong to the family of generative models, which means they can generate new data instances that resemble the training data

A VAE consists of two main components: the encoder and the decoder. The Encoder takes an input (e.g., an image) and compresses it into a lower-dimensional representation. This representation is often called the latent space or latent variables. Instead of encoding an input as a single point, VAEs encode inputs as distributions over the latent space.
The Decoder then takes this lower-dimensional representation and reconstructs the input data from it.

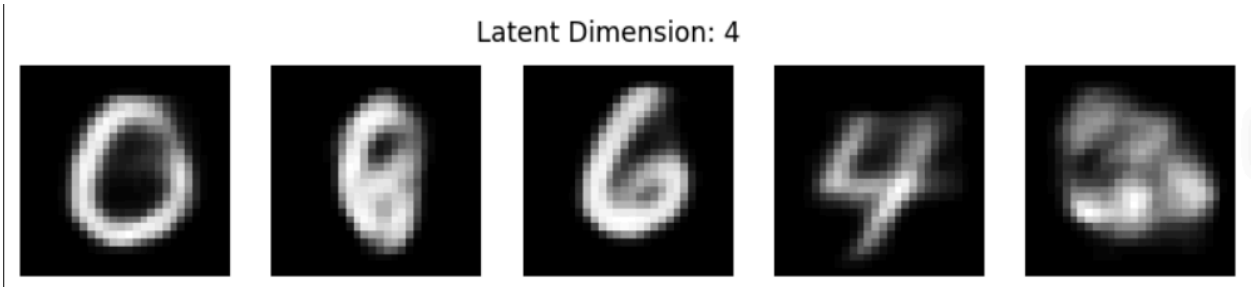
During training, VAEs optimize two things:
The accuracy of the reconstruction (how closely the output of the decoder matches the original input).
The regularity of the encoded representations (using a term in the loss function called the Kullback-Leibler divergence, which measures how much the learned distribution deviates from a prior distribution, typically chosen to be a normal distribution).

Given Fig 1.1 shows the reconstructed image for the latent dimensions 2,4,8,16,32,64.

Latent Variable: 2

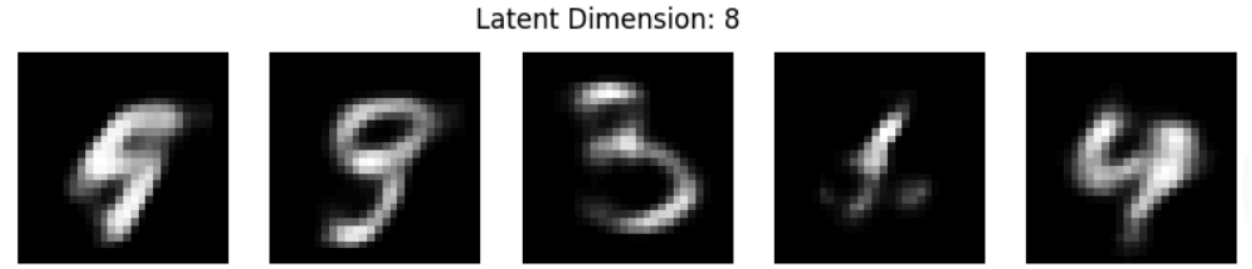


Latent Variable: 4



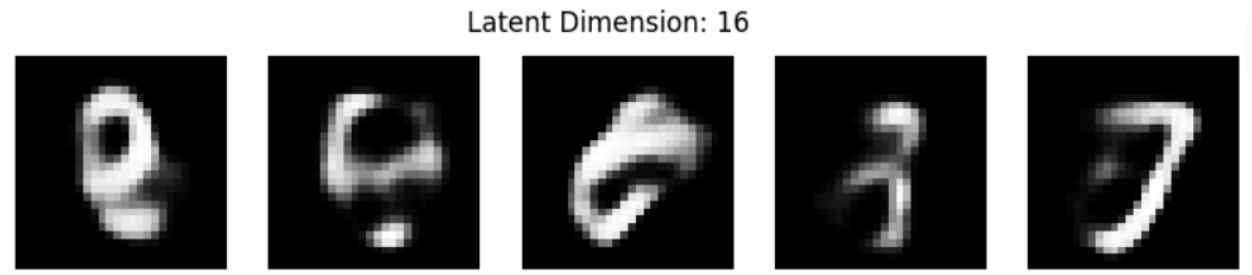

```
Average Mean: tensor([-0.1119,  0.0393, -0.1118, -0.1937], device='cuda:0')
Average Variance:
tensor([[0.0137, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0116, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0166, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0162]], device='cuda:0')
```

Latent Variable: 8



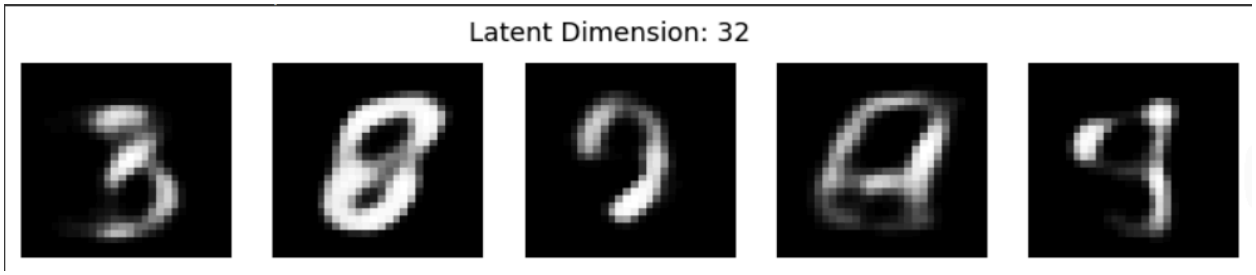

```
Average Mean: tensor([-0.0512, -0.0519, -0.1023, -0.0720, -0.1323, -0.0383,  0.0717,  0.0117],
       device='cuda:0')
Average Variance:
tensor([[0.0250, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0285, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0283, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0423, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0284, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0220, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0259, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0195]],
       device='cuda:0')
```

Latent Variable: 16

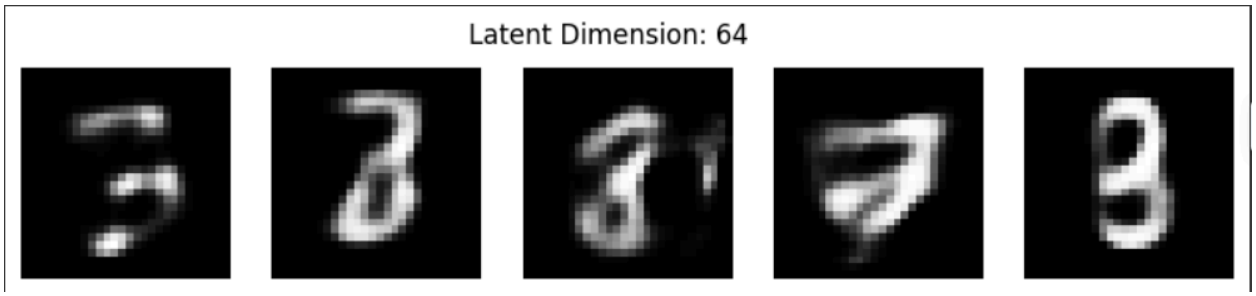


```
Average Mean: tensor([ 0.0553,  0.0546,  0.0284,  0.0363, -0.0267, -0.0358, -0.0387, -0.0456,
         0.0178,  0.0450,  0.1357,  0.0075, -0.0089,  0.0026, -0.1011,  0.0739],
       device='cuda:0')
Average Variance:
tensor([[0.0294, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0520, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0290, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0679, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.1123, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0524, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0711, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0615, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1207,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0234, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.1367, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.2370, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0807, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0869, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0458, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0631, 0.0000]],
       device='cuda:0')
```

Latent Variable: 32



Latent Variable: 64



Loss Function:

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}}$$

Where:

- $\mathcal{L}_{\text{recon}}$ is the reconstruction loss, which measures the difference between the input data and the output of the decoder.
- $\mathcal{L}_{\text{KL}}$ is the KL divergence term, which measures the divergence between the distribution of latent variables produced by the encoder and a chosen prior distribution (typically a standard Gaussian).

Analysis:

From manual analysis of images from different images we can understand that the performance of VAE model increases with increase in latent dimension size from 2-8 and then decreases drastically. This is most likely due to the data requiring only 2 or 4 dimensions and remaining are irrelevant features.

Another reason for the poor performance of the higher latent dimensional models is that they require more time and GPU resources to train and therefore do not perform on the same level as lower dimensional models when trained for the same number of epochs. Thus we can conclude that 4 dimensional VAE model is reasonably good at reconstructing digits.

Part-A-II: Generative Adversarial Networks

Methodology

Generative Adversarial Networks (GANs) are a class of deep learning models utilized for generating new data instances that closely resemble the training data. Like Variational Autoencoders (VAEs), GANs belong to the family of generative models, enabling them to produce novel data instances resembling the input data distribution.

A GAN comprises two primary components: the generator and the discriminator. The generator is akin to the decoder in VAEs, as it generates synthetic data samples. Conversely, the discriminator serves a role similar to the encoder, distinguishing between real and synthetic data.

During training, GANs optimize two primary objectives:

1. The accuracy of the generated samples, ensuring that they closely resemble real data instances.
2. The training of the discriminator to discern between real and synthetic samples accurately.

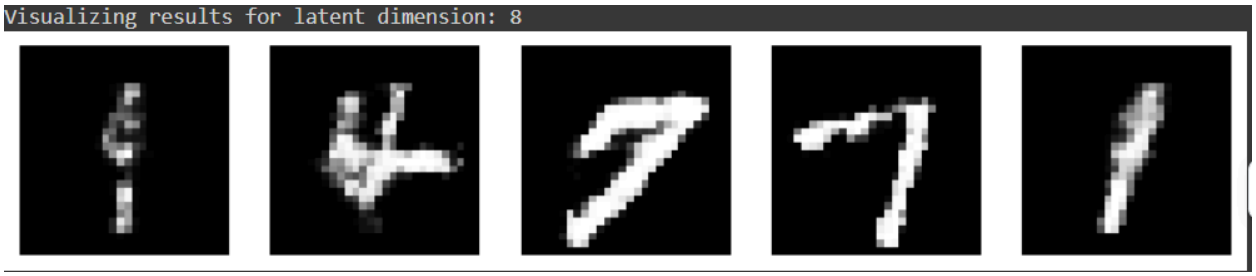
Much like VAEs, GANs utilize a loss function to guide the training process. However, instead of the Kullback-Leibler divergence used in VAEs, GANs employ a unique adversarial loss function. This loss function fosters a competitive dynamic between the generator and the discriminator, leading to the generation of realistic data samples.

Given Fig 1.2 shows the reconstructed image for the latent dimensions 2,4,8,16,32,64.

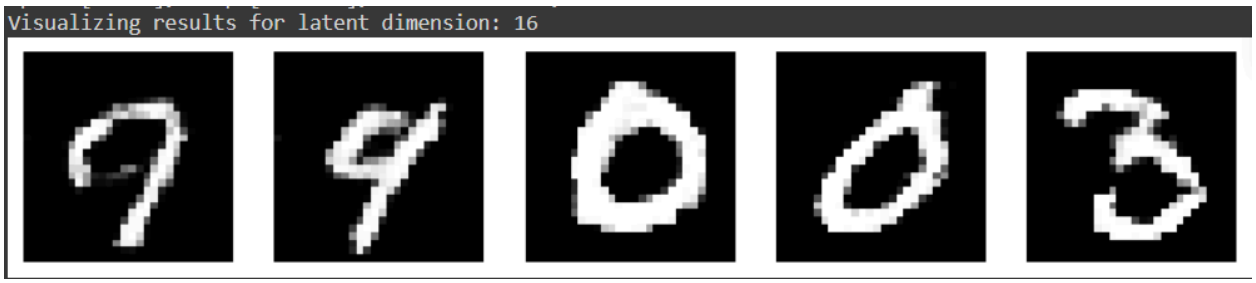
Latent Variable: 2

Latent Variable: 4

Latent Variable: 8



Latent Variable: 16



Latent Variable: 32

Latent Variable: 64

Formulation:

$$V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

Where,

- $p_{\text{data}}(x)$ is the true data distribution.
- $p_z(z)$ is the prior distribution of the input noise z (often a simple distribution like uniform or Gaussian).
- $D(x)$ represents the discriminator's output probability for a real data sample x .
- $D(G(z))$ represents the discriminator's output probability for a generated data sample $G(z)$.

Analysis

We can observe that with an increase in latent variable dimension the precision of the reconstructed digits also increases. Hence we can say that on training for the same number of epochs the GAN with the most number of latent variables is working the best.

This could be because higher latent dimension results in increased generalization. So the model can learn the generative distribution of the data much better. Higher dimensions also make the model more robust to the noise in the images.

Part-A-III: Implementing Diffusion Model.

Methodology

Diffusion models consist of two main components: the diffusion process and the learnable diffusion model.

The diffusion process involves a series of iterative steps where noise is injected into an input image, and transformations are applied to produce new samples. At each step of the diffusion process, noise is diffused across the image, gradually transforming it into a realistic sample. This iterative process allows diffusion models to generate high-quality images by simulating the gradual evolution of noise across the image.

The learnable diffusion model comprises neural networks that parameterize the transformations applied during the diffusion process. These neural networks learn to model the conditional distributions of image transformations given the previous state of the image. By learning these conditional distributions, the diffusion model can generate realistic samples that closely resemble the training data distribution.

During training, the diffusion model is trained to minimize a loss function that measures the discrepancy between the generated samples and the real data. This loss function incentivizes the generated samples to resemble the real data distribution while penalizing deviations from the input image. Optimization techniques such as stochastic gradient descent are employed to train the diffusion model, ensuring that it learns to generate high-quality samples.

Trained diffusion models can generate new samples by iteratively applying learned transformations to noise-initialized images. This sampling process yields realistic samples that capture the characteristics of the training data distribution. Evaluation of diffusion models typically revolves around the quality and fidelity of generated samples, with metrics such as Inception Score (IS) and Frechet Inception Distance (FID) used to quantify the quality and diversity of generated samples.

Given Fig 1.3 shows the reconstructed image for the diffusion steps 10.

Formulation:

$$x_t = x_{t-1} + \sqrt{\beta_t} \cdot \epsilon_t$$
$$x_t = T_t(x_{t-1}; \theta)$$

Where,

- x_t is the image at step t.
- x_{t-1} is the previous image.
- ϵ_t is the injected noise at step t.
- T_t represents the transformation applied at step t parameterized by θ.
- β_t controls the magnitude of the noise injection.

Analysis

From the images it is clear that the diffusion model is better than GAN and VAE in reconstruction of images. It predicts the images with lesser noise and better accuracy than both the models.

The better performance is mainly because Diffusion models follows probabilistic training and its loss is likelihood based.

Part-B-I: Deep Convolutional GAN

Methodology

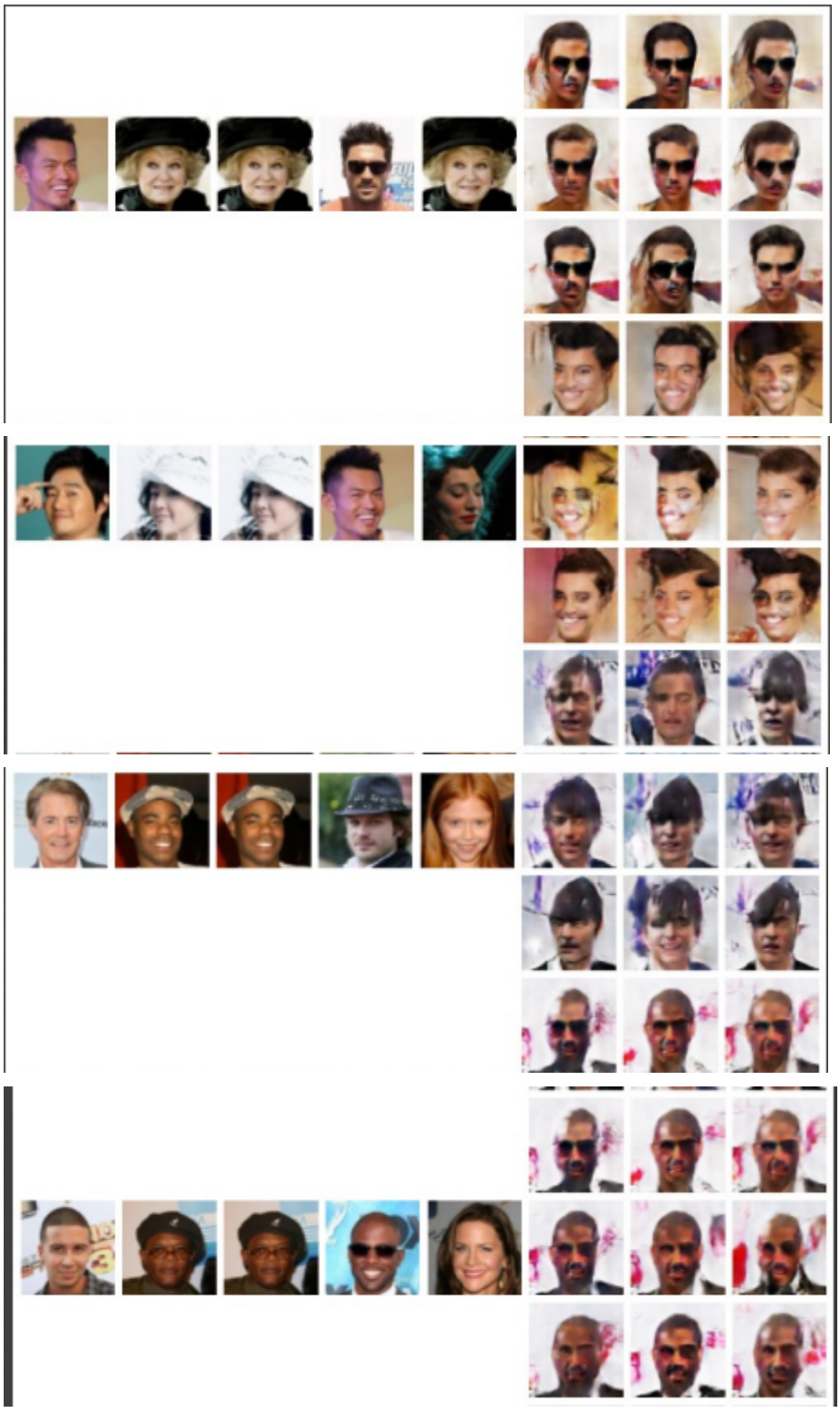
The architecture of DCGANs consists of two main components: the generator and the discriminator.

The generator in DCGANs comprises a deep convolutional neural network designed to transform random noise vectors into synthetic images. It typically consists of several convolutional layers followed by upsampling layers, allowing the generator to gradually generate images of increasing resolution. The final output of the generator is a synthetic image that closely resembles the real data distribution.

On the other hand, the discriminator is also a deep convolutional neural network tasked with distinguishing between real and synthetic images. Similar to the generator, the discriminator consists of convolutional layers followed by downsampling layers, enabling it to effectively classify images at different resolutions. The discriminator is trained to accurately distinguish between real and fake images, providing feedback to the generator to improve its ability to generate realistic samples.

During training, DCGANs optimize a minimax objective function that encourages the generator to produce images that are indistinguishable from real images while simultaneously training the discriminator to accurately classify between real and fake images. This adversarial training process leads to the mutual improvement of both the generator and discriminator, resulting in the generation of high-quality images.

Given Fig 1.4 shows the reconstructed image.



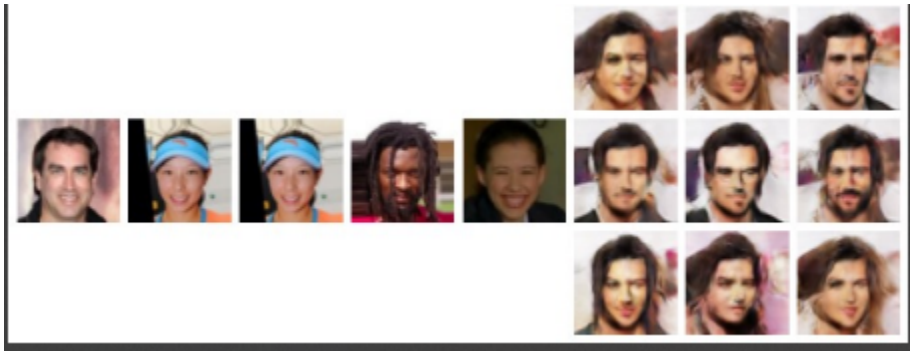


Image = Smiling men + hat - hat + mustache - without mustache

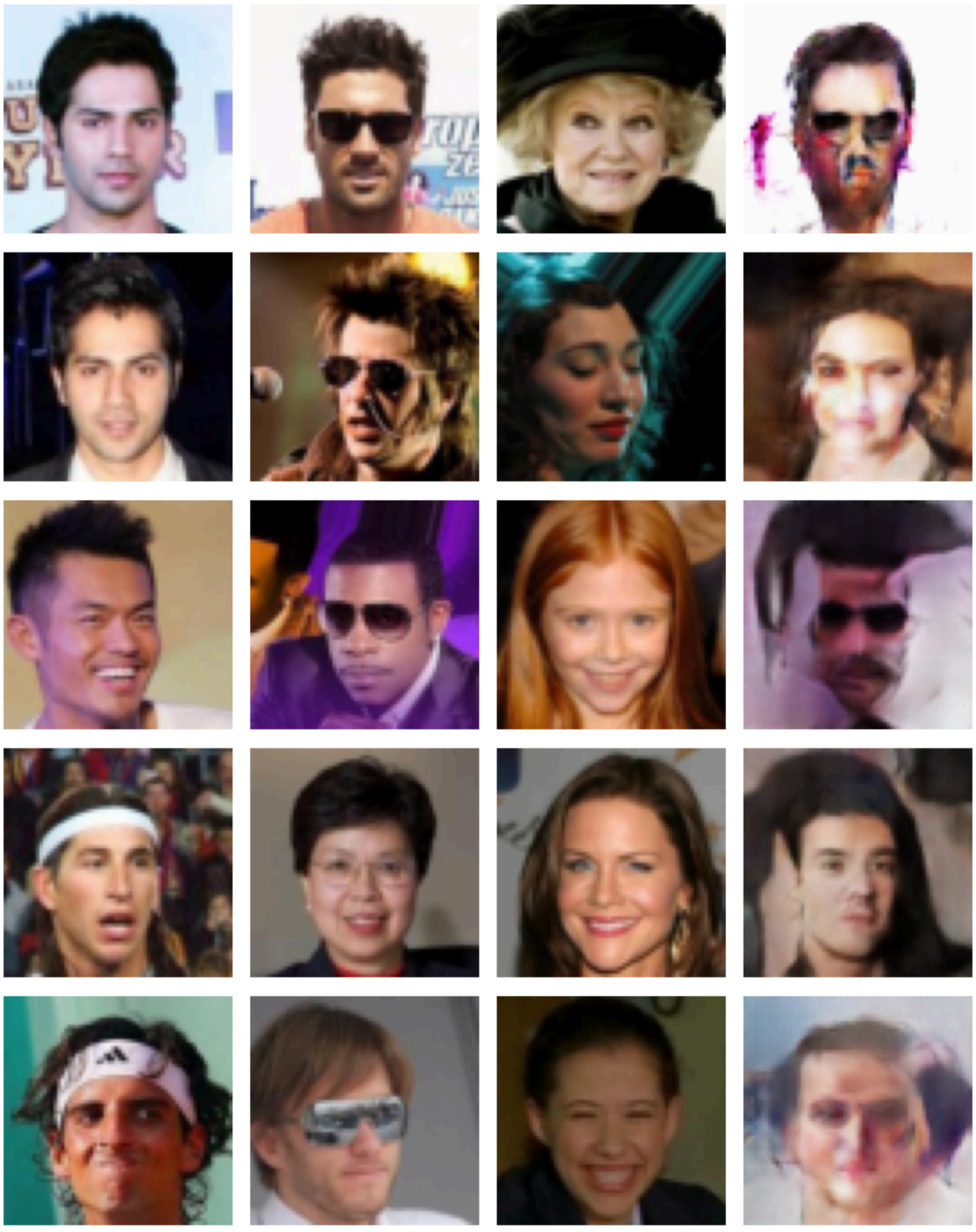
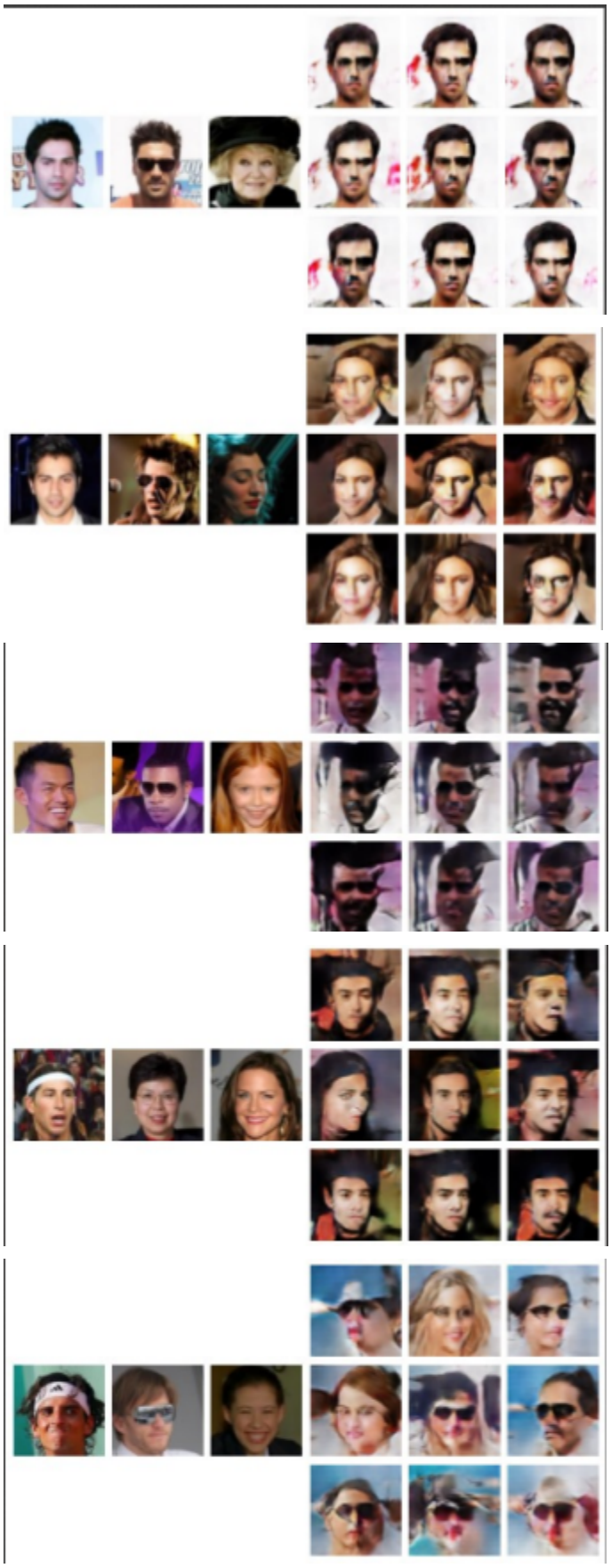


Image = Men no glass + ppl with glass - ppl without glass



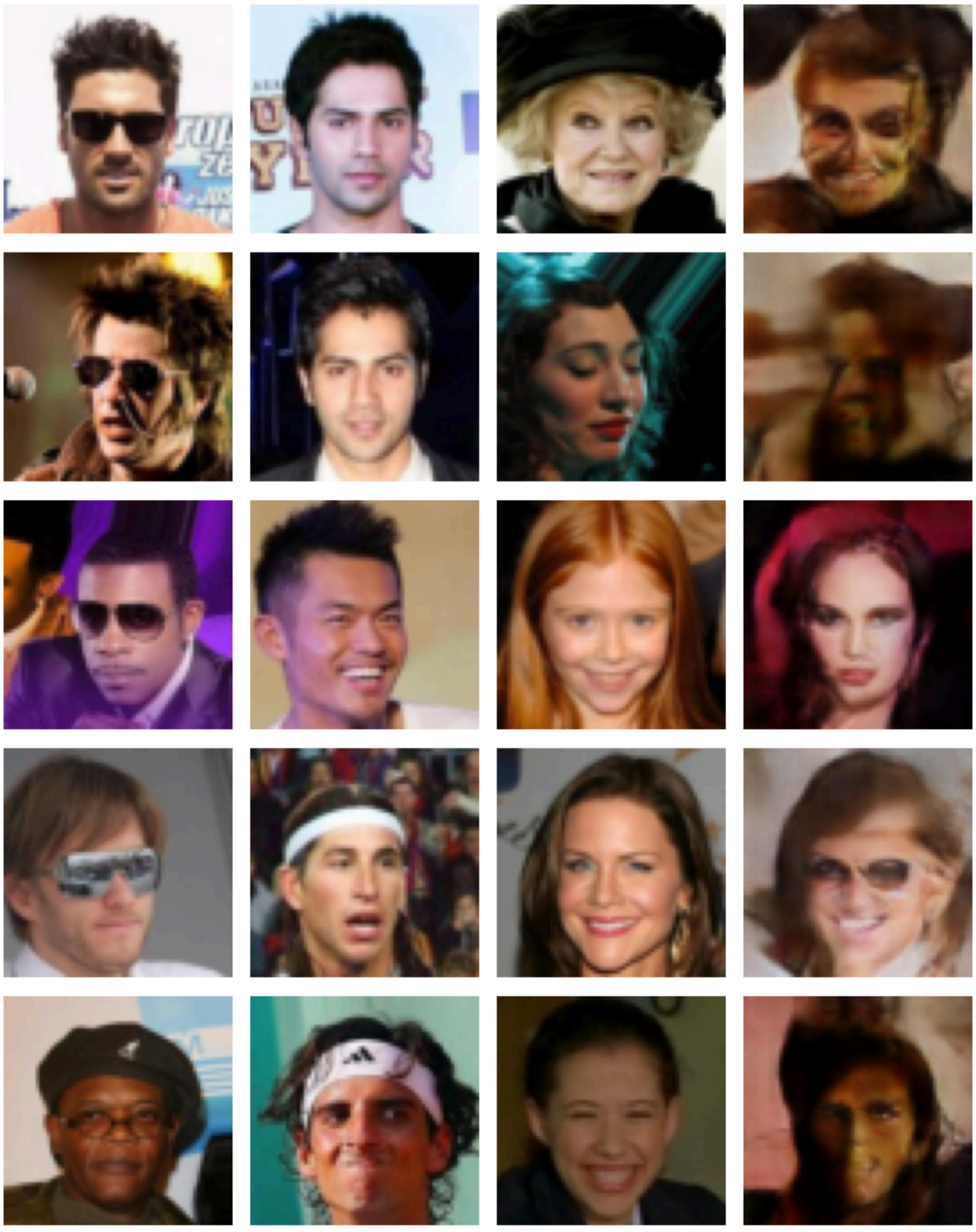


Image = Men with glass - men no glass + women without glass

$\min_{\theta_G} \max_{\theta_D} V(D, G)$

Where:

- θ_G represents the parameters of the generator G.
- θ_D represents the parameters of the discriminator D.
- $V(D, G)$ is the value function, defined as the sum of two terms:

$$V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

- $p_{\text{data}}(x)$ represents the true data distribution.
- $p_z(z)$ represents the prior distribution of the noise vector z.
- $D(x)$ is the discriminator's output, representing the probability that x is a real image.
- $G(z)$ is the generator's output, representing the synthetic image generated from noise z.

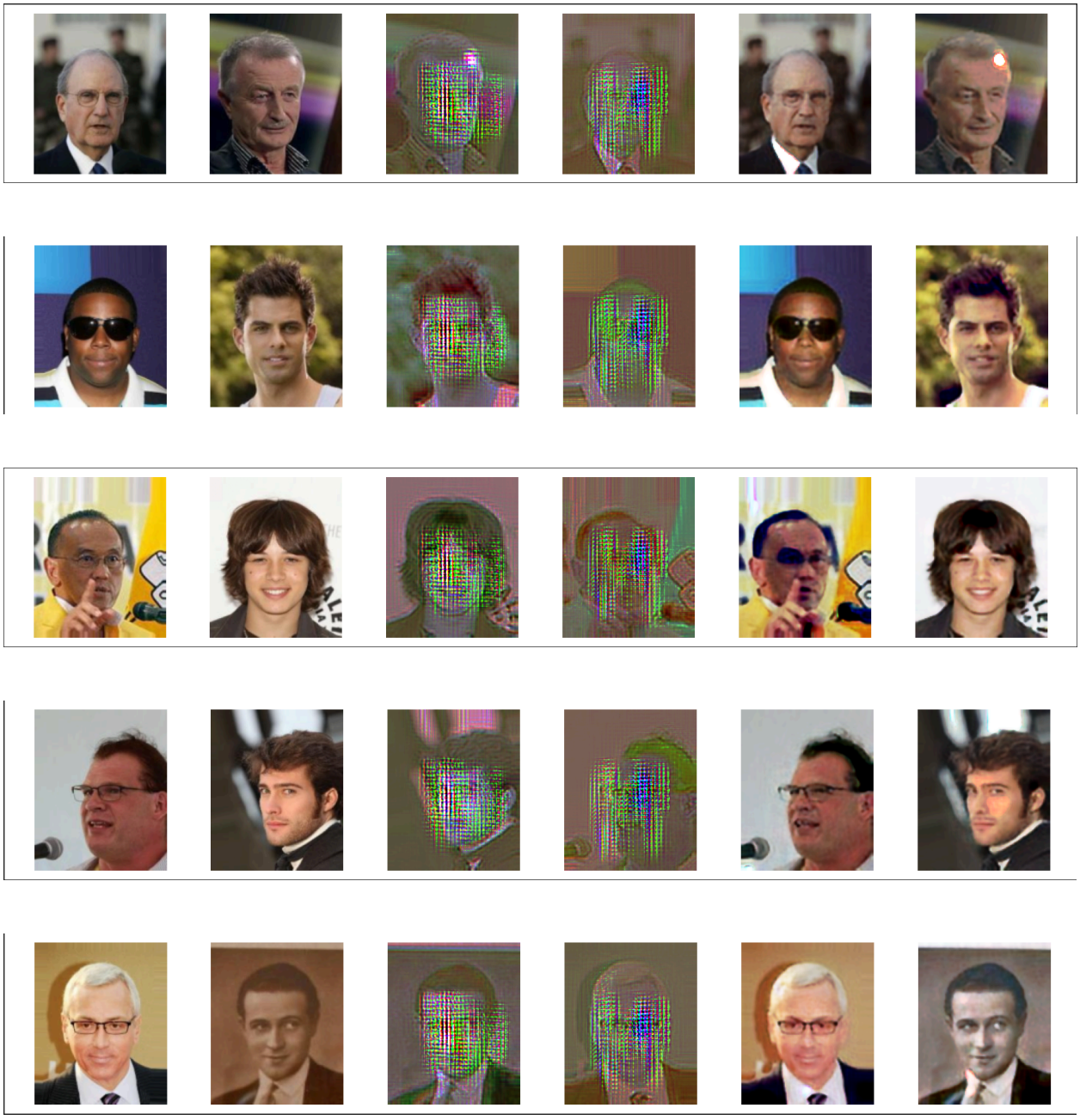
Part-B-II: CycleGAN

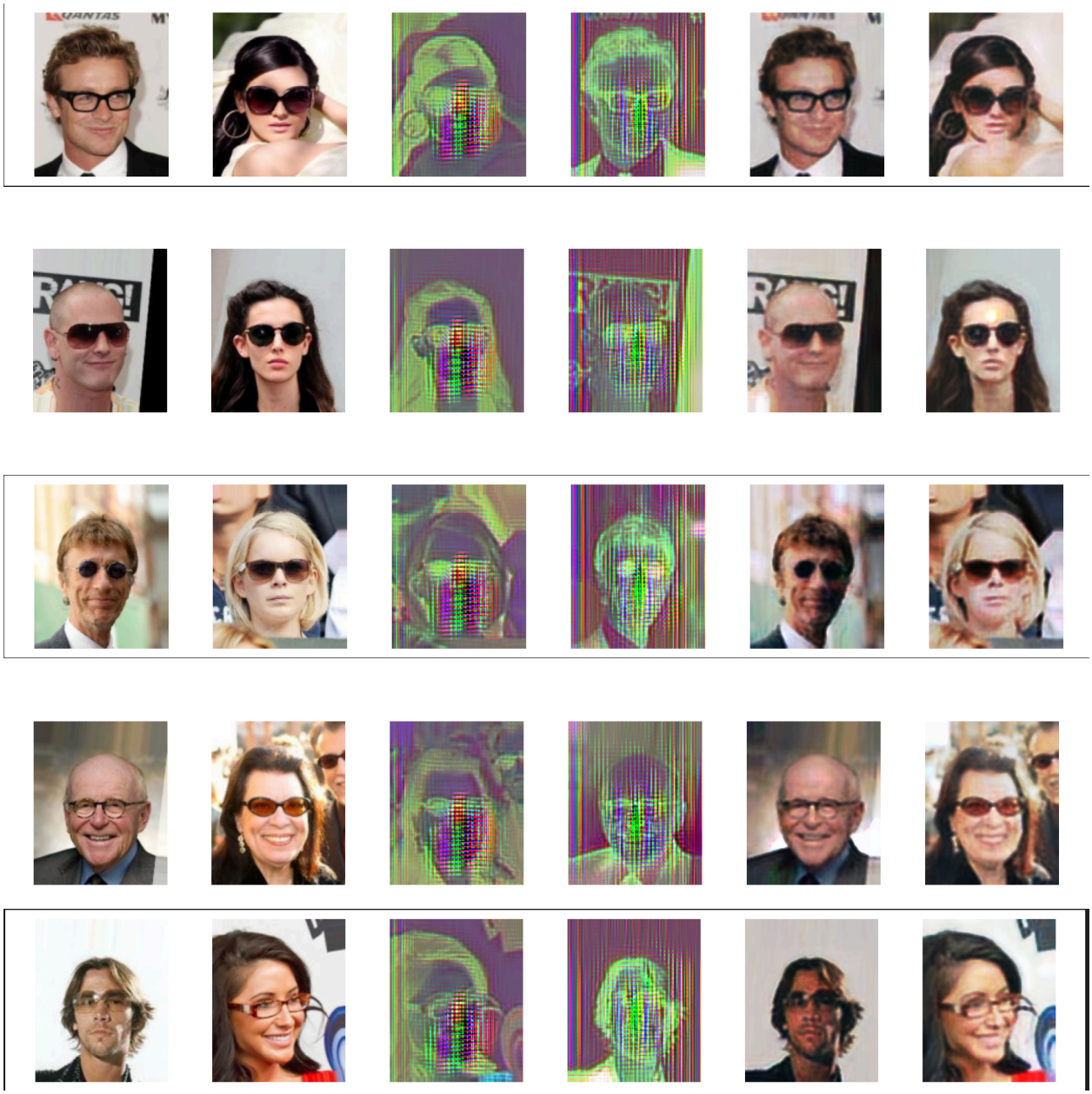
The CycleGAN architecture comprises two primary components: the generator and the discriminator. For generators, CycleGAN employs two networks: Gab and Gba, where Gba translates images from domain a to b, and Gba performs the reverse translation. These generators typically adopt deep convolutional neural networks (CNNs) with encoder-decoder architectures, where the depth and capacity of the networks are controlled by the number of layers and filters, respectively. Similarly, CycleGAN utilizes two discriminators, Da and Db, which are also implemented using CNNs with PatchGAN architectures. The architecture parameters, such as patch size and the number of layers and filters, affect the discriminators' capacity and discriminative ability.

A crucial aspect of CycleGANs is the incorporation of cycle-consistency loss, enforcing that the translation from one domain to another and back results in the original image. This loss encourages generators to preserve semantic information during translation. Additionally, adversarial losses are employed to guide training, with generators aiming to deceive discriminators into classifying translated images as real, while discriminators aim to distinguish between real and fake images. The importance of cycle-consistency and adversarial losses relative to other losses can be controlled through hyperparameters, such as the weight of the cycle-consistency loss and the adversarial loss.

During optimization, CycleGANs are trained using stochastic gradient descent (SGD) or variants like the Adam optimizer. The learning rate, determining the rate of parameter updates during optimization, influences the convergence speed and stability of training. By carefully adjusting these hyperparameters, CycleGANs can achieve high-quality image translations that preserve semantic content while transforming images across different domains.

Given Fig 1.4 shows the reconstructed image





FORMULATION:

$$\min_{G_{AB},G_{BA}} \max_{D_A,D_B} \mathcal{L}_{\text{GAN}}(G_{AB}, D_B, A, B) + \mathcal{L}_{\text{GAN}}(G_{BA}, D_A, B, A) + \lambda \mathcal{L}_{\text{cycle}}(G_{AB}, G_{BA})$$

Where:

- $\mathcal{L}_{\text{GAN}}$ is the adversarial loss function for each generator and discriminator pair.
- $\mathcal{L}_{\text{cycle}}$ is the cycle-consistency loss function.
- λ is the weighting factor controlling the influence of the cycle-consistency loss.
- G_{AB} and G_{BA} are the generators translating between domains A and B.
- D_A and D_B are the discriminators distinguishing real from translated images in domains A and B, respectively.