CIS 680: Advanced Machine Perception
Assignment 4: Generative Adversarial Network

Contents

1 Overview

A generative model has a huge difference from a discriminative model which you have encountered in earlier homework. The discriminative model outputs class label of the given input but it cannot imagine any instance of a specific class. However, the generative model can generate such instances. In this exercise, you will implement a family of generative models including a variational autoencoder (VAE) [5], a generative adversarial network (GAN)[3] and a CycleGAN [9].

2 Variational Auto-Encoders (VAEs)

Components to implement: a) Network Architecture, b) Loss function, c) Qualitative and Quantitative Evaluation. Training time for VAE $\sim$ 10 minutes for 10 epochs

2.1 Method

Variational Auto-Encoders (VAEs) is a generative model built on top of an auto-encoder. First, the encoder network encodes an input into a latent code (a compact Gaussian distribution), then perturb the latent encoding by adding a sampled noise, and finally the decoder decodes the latent code into the reconstructed input.

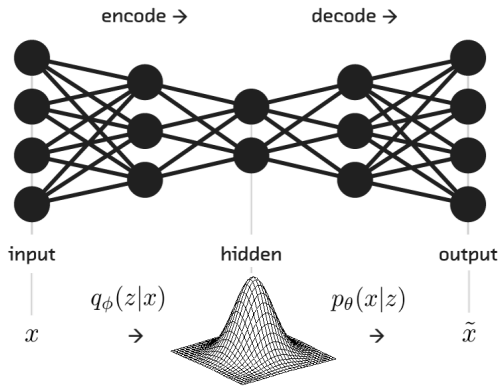


Figure 1: Variational Auto-Encoders (VAEs)

2.2 Training of VAE

2.2.1 Data set: Fashion MNIST

We use the Fashion-MNIST data set for VAE training - Fashion-MNIST. You could directly load the data set from "torchvision.datasets". The following image is a snapshot of Fashion-MNIST, each image in the grid is a 28x28 grayscale image, Fig.[2]:

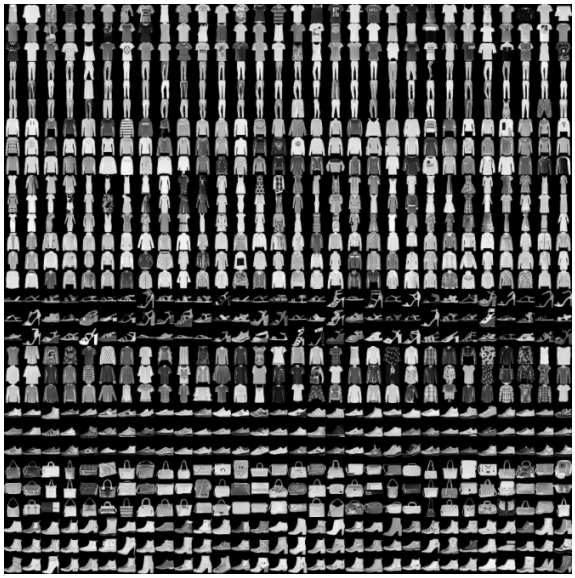


Figure 2: A snapshot of Fashion-MNIST data set

2.2.2 Network architecture

The VAE network consists of two parts, encoder and decoder, as we illustrated in Fig.[1]. Following, we provide the details of network architecture for encoder in Table.[1] and decoder in Table.[2] of VAE:

Layer	Hyperparameters
Fully-connected 1	input channel=784, output channel=400, followed by ReLU
Fully-connected 2 - $\mu(x)$	input channel=400, output channel=zdim, followed by ReLU
Fully-connected 2 - $log(\sigma^2(x))$	input channel=400, output channel=zdim, followed by ReLU

Table 1: Architecture for encoder

Layer	Hyperparameters
Fully-connected 3	input channel=zdim, output channel=400, followed by ReLU
Fully-connected 4	input channel=400, output channel=784, followed by Sigmoid

Table 2: Architecture for decoder

2.2.3 Objective function

The left-hand side of the objective function in Eq.[1] consists of two parts. The first part is log-likelihood over the training images. The second part is Kullback-Leibler (KL) divergence. KL term models the distance between the real underlying posterior $P(z|x)$ and the approximated posterior modeled by network $Q(z|x)$. The VAE objective function, we aim to maximize, is as follows,

$$\log P(x) - KL[Q(z \mid x) \| P(z \mid x)] = \mathbb{E}_{z\sim Q}[\log P(x \mid z)] - KL[Q(z \mid x) \| P(z)] \tag{1}$$

Rewriting the KL term in $KL[Q(z \mid x) \| P(z \mid x)] = E_{z\sim Q}[\log Q(z \mid x) - \log P(z \mid X)]$. Then apply Bayes rule to $P(z|x) = \frac{P(x|z)\cdot P(z)}{P(x)}$. We would get $KL[Q(z \mid x) \| P(z \mid X)] = E_{z\sim Q}[\log Q(z \mid x) - \log P(x \mid z) - \log P(z)] + \log P(x)$. Plug term $KL[Q(z \mid x) \| P(z \mid X)]$ back into left-hand side of Equ.[1], we get the right-hand side expression.

- First term on right-hand side: $E_{z\sim Q}[\log P(x \mid z)]$ is the log-likelihood modeled by the decoder. $P(x \mid z)$ is the output of the VAE decoder given a randomly sampled latent variable from latent space. In Fashion-MNIST, each pixel value is between 0 and 1, so we use binary cross entropy between decoder output $P(x \mid z)$ and the input image batch. Since minimizing the loss enforces the reconstructed outputs to be similar to the inputs, it is also called reconstruction loss.

- And the second term is Kullback-Leibler divergence between the posterior distribution $Q(z|x)$ modeled by the encoder and the prior distribution $P(z)$. In VAE, we assume $P(z)$ is a normal distribution $\mathcal{N}(0,1)$. The encoder estimates a mean $\mu(x)$ and log-variance $log(\sigma^2(x))$ (The diagonal values of covariance matrix, explanation of why only using diagonal values will be in Section.[2.2.4] point 4) to represent the posterior normal distribution by $Q(z|x)$. This term enforces the approximated latent space $Q(z|x)$ to be a prior Gaussian distribution $\mathcal{N}(0,1)$ which is a compact latent space that VAE can sample from during inference.

2.2.4 Implementation detail

1. The details of VAE model architecture are described in Table.[1][2]. As the architecture of VAE is fully connected, input to VAE model is a 784 dimensional vector, obtained after flattening the 28×28 image.

2. The overall logic of VAE would be the encoder estimates posterior distribution $Q(z|x)$ by estimating mean and log-variance using Fully-connected layer 2 - $\mu(x)$ and Fully-connected layer 2 - $log(\sigma^2(x))$ given input image. Once $Q(z|x)$ is computed, reparameterization technique is applied to get a new latent code. Finally, decoder decodes the reparameterized latent code into reconstructed input. The detail of reparameterization will be discuss in point 5.

3. For the reconstruction loss, please use element-wise binary cross entropy loss between reconstructed image and input image. As mentioned in Section.[2.2.3].

4. In the original VAE paper [5], author assumes the approximated latent posterior distribution $Q(z|x)$ as a isotropic multivariate Gaussian. Isotropic multivariate Gaussian means that each dimension of multivariate Gaussian is independent to each other. Thus, the covariance matrix of $Q(z|x)$ is a diagonal matrix with elements on the diagonal being the variance $\sigma(x)$, which could be computed from output of decoder $log(\sigma(x))$. In the Section.[2.2.2], the dimension of encoded log-variance is $zdim$ dimensional vector rather than $zdim \times zdim$ matrix that is because the diagonal of the matrix is represented as a vector. KL divergence $D_{KL}[\mathcal{N}(\mu(X), \Sigma(X)) \| \mathcal{N}(0,1)]$ in loss function could be derived in the following steps:

$$
\begin{aligned}
&D_{KL}[\mathcal{N}(\mu_1(x), \Sigma_1(x)) \| \mathcal{N}(\mu_2(x), \Sigma_2(x))] \\
&= \int \left[\frac{1}{2} \log \frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2} (x-\mu_1)^T \Sigma_1^{-1} (x-\mu_1) + \frac{1}{2} (x-\mu_2)^T \Sigma_2^{-1} (x-\mu_2) \right] \times p(x) dx \\
&= \frac{1}{2} \log \frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2} \operatorname{tr} \left\{ E\left[(x-\mu_1)(x-\mu_1)^T \right] \Sigma_1^{-1} \right\} + \frac{1}{2} E\left[(x-\mu_2)^T \Sigma_2^{-1} (x-\mu_2) \right] \\
&= \frac{1}{2} \log \frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2} \operatorname{tr} \{I_d\} + \frac{1}{2} (\mu_1-\mu_2)^T \Sigma_2^{-1} (\mu_1-\mu_2) + \frac{1}{2} \operatorname{tr} \{\Sigma_2^{-1} \Sigma_1\} \\
&= \frac{1}{2} \left[\log \frac{|\Sigma_2|}{|\Sigma_1|} - d + \operatorname{tr} \{\Sigma_2^{-1} \Sigma_1\} + (\mu_2-\mu_1)^T \Sigma_2^{-1} (\mu_2-\mu_1) \right] \\
&D_{KL}[\mathcal{N}(\mu(x), \Sigma(x)) \| \mathcal{N}(0,1)] \\
&= \frac{1}{2} \left(\operatorname{tr}(\Sigma(x)) + \mu(x)^T \mu(x) - k - \log \det(\Sigma(x)) \right) \\
&= \frac{1}{2} \left(\sum_{zdim} diag(\Sigma(x)) + \sum_{zdim} \mu(x)^2 - \sum_{zdim} 1 - \sum_{zdim} log(diag(\Sigma(x))) \right) \\
&= \frac{1}{2} \sum_{zdim} \left(diag(\Sigma(x)) + \mu(x)^2 - 1 - log(diag(\Sigma(x))) \right) \\
&= \frac{1}{2} \sum_{zdim} \left(exp(\log(\sigma^2(x))) + \mu(x)^2 - 1 - \log(\sigma^2(x)) \right)
\end{aligned}
$$

   In last step, $\log(\sigma^2(x))$ is log-variance output from VAE encoder.

5. Reparametrization procedure is important for VAE network to generate different reconstruction images. VAE utilizes this step to sample a neighbor code centered around encoded latent code. As we discussed in previous point, the approximated posterior $Q(z|x)$ are independent in each dimension. In reparameterization, we also treat different dimension independently. To elaborate on this point,

- Obtain $\mu(x)$ and $log(\sigma^2(x))$ from decoder output. Apply function $exp(\frac{1}{2}log(\sigma^2(x)))$ to log-variance to get the standard deviation $\sigma(x)$ for each dimension.
- Use expression $\mu(x)+\sigma(x)\cdot\epsilon$ to sample a reparameterized latent code for each latent dimension, where ϵ is sampled from standard Gaussian distribution in dimension $zdim$.

In this way, reparameterization takes $\mu(x)$ and $log(\sigma^2(x))$ as input and samples a new point from the Gaussian distribution $\mathcal{N}(\mu(x), \sigma^2(x))$ in $zdim$ dimension.

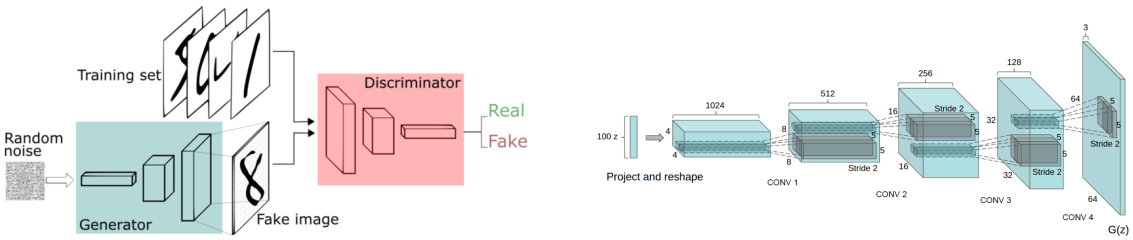
6. The default latent space dimension(zdim) is equal to 5 in Table.[1][2]. You could play around with this value and include your observation in the report. The default weight between the reconstruction loss and KL divergence loss is 1:1. The default optimizer is Adam with a learning rate 1e-3. And the default number of training epochs is 10. You could change the hyper-parameters and optimizer settings in order to get better performance.

3 Generative Adversarial Networks (GANs)

Components to implement: a) Network Architecture, b) Loss function, c) Qualitative and Quantitative Evaluation. Training time for VAE $\sim$ 1 hour for 100 epochs.

3.1 Method

A generative adversarial network (GAN) is a generative model composed of two neural networks: a generator and a discriminator. These two networks are trained in an unsupervised way via competition. The generator creates "realistic" synthetic images given random noise to fool the discriminator, while the discriminator evaluates the given image (real image or synthetic image) for authenticity Fig.[3a].



(a) Generative Adversarial Network [7]

(b) DCGAN[6]

Figure 3: GAN model and DCGAN Architecture

3.2 Training of GAN

3.2.1 Data set: STL-10

GANs have the capability to generate realistic images. In this part, you would be asked to train the GAN on STL-10 data set. The data set could be directly loaded from "torchvision.datasets". The following is a snapshot of the data set, Fig.[4]:

The data pre-processing as follows:

1. resize the whole image to 64×64.
2. normalize the image with mean=$[0.5, 0.5, 0.5]$ and std=$[0.5, 0.5, 0.5]$. That's is normalizing the image in range [-1, 1].

3.2.2 Network Architecture

The GAN consists of two parts, generator and discriminator, as illustrated in Fig.[3a]. Following are the details of network architecture for generator Table.[3] and discriminator Table.[4] of GAN. Please also refer to implementation details in Section.[3.2.4 2c] for more details:

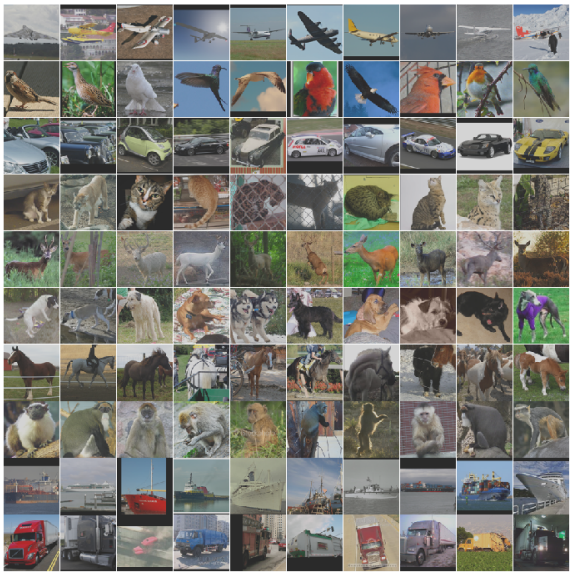


Figure 4: A snapshot of STL-10 data set

Layer	Hyperparameters
TransposedConv 1	Kernel = $(4, 4, 1024)$, Stride$= 1$, Padding $= 0$, BatchNorm, ReLU
TransposedConv 2	Kernel = $(4, 4, 512)$, Stride $= 2$, Padding $= 1$, BatchNorm, ReLU
TransposedConv 3	Kernel $= (4, 4, 256)$, Stride $= 2$, Padding $= 1$, BatchNorm, ReLU
TransposedConv 4	Kernel $= (4, 4, 128)$, Stride $= 2$, Padding $= 1$, BatchNorm, ReLU
TransposedConv 5	Kernel $= (4, 4, 3)$, Stride $= 2$, Padding $= 1$, tanh

Table 3: Network details of DCGAN generator

Layer	Hyperparameters
Conv 1	Kernel = $(4, 4, 128)$, Stride=2, Padding$= 1$, LeakyReLU, negative slop=0.2, inplace=True
Conv 2	Kernel = $(4, 4, 256)$, Stride =2, Padding $= 1$, BatchNorm, LeakyReLU, negative slop=0.2, inplace=True
Conv 3	Kernel =$(4, 4, 512)$, Stride =2, Padding $= 1$, BatchNorm, , LeakyReLU, negative slop=0.2, inplace=True
Conv 4	Kernel =$(4, 4, 1024)$, Stride =2, Padding $= 1$, BatchNorm, , LeakyReLU, negative slop=0.2, inplace=True
Conv 5	Kernel =$(4, 4, 1)$, Stride =1, Padding $= 0$, Sigmoid

Table 4: Network details of DCGAN discriminator

3.2.3 Objective function

The objective function is a min-max game. The generator wants to minimize the loss, such that the term $D(G(z))$ goes to 1. The discriminator wants to maximize the loss such that $D(x)$ term goes to 1 and $D(G(z))$ term goes to 0. The overall optimization problem is as follows,

$$\min_G \max_D \mathcal{L}(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2}$$

Here, G and D are the generator and the discriminator. For training, the overall optimization task is split into two sub-tasks.

- First task: Maximize the objective w.r.t. discriminator D:

$$\max_D \mathbb{E}_{x \sim p_{\text{dun}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{3}$$

  In Eq.[3], the parameters of network G are fixed, and only the parameters in network D are updated.

- Second task: Minimize the objective w.r.t. generator G:

$$\min_G \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{4}$$

and In Eq.[4], the parameters of network D are fixed, and only the parameters in network G are updated.

3.2.4 Implementation Details

1. Training GANs can get a little tricky sometimes and mode collapse could be an issue. Mode collapse is the problem when different inputs mapped into the same generator output. To address this potential problem, the following tricks are used in the GAN implementation (some of them are mentioned in previous sections),

   - Normalize the images using the statistics in Section.[3.2.1].
   - Use Tanh activation function layer as the last layer of the generator, as discussed in architecture section.[3.2.2]].
   - Using a spherical latent code z as input to generator network will be helpful. This means using Gaussian distribution for z.
   - Modified loss function. For second sub-task in Section.[3.2.3], instead of minimizing $\log(1 - D(G(z)))$ it is recommended to maximize $\log(D(G(z)))$. In practice, the element-wise binary cross entropy between $\log(D(G(z)))$ and 1 are minimized.
   - In the discriminator training, at each iteration, the loss of $logD(x)$ is computed from a batch of real images, and the loss of $log(1 - D(G(z))$ is computed from a set of generated images with the same batch size as the real images. For example, if batch size of real images is 128, then we should sample 128 latent variables and generate 128 fake images. Computation of log-likelihood $\log D(x)$ could be done by computing binary cross entropy between $D(x)$ and 1, and $\log(1 - D(G(z)))$ could be computed as the binary cross entropy between $D(G(z))$ and 0.
   - Training GAN essentially optimizes two major networks - generator (G) and discriminator (D). Specifically, both G and D are optimized at every iteration. While people have different preferences/reasons to choose first optimize G or D, we follow the convention to optimize D first and then G at every iteration.

2. Default network and training settings -

   (a) The default batch size is 128.

   (b) The initial random noise input into the generator network is sampled from standard Gaussian distribution with latent space dimension 100.

   (c) The bias term for Convolution and Transpose Convolution layers in Table.[3][4] are set to 'False'. Because those layers are followed by the batch normalization which already contains bias term.

   (d) The default initialization of network follows normal distribution for the weights in convolution layers and weights in batch normalization, and set bias in batch normalization to 0.

   (e) The default optimizer is Adam with learning rate of 2e-4. The default training epochs is 100.

   (f) You could change above hyper-parameters and optimizer settings for better performance.

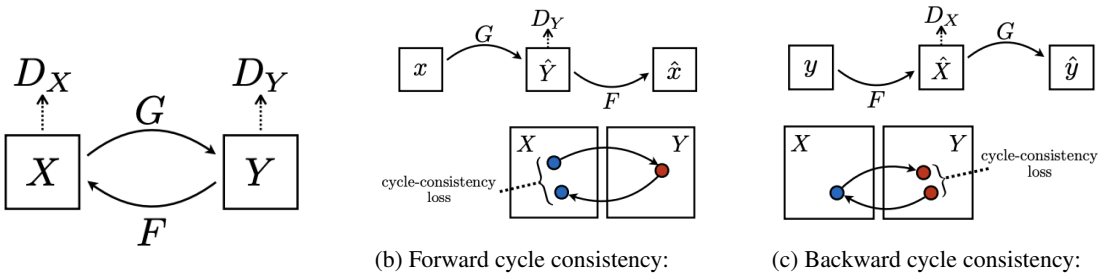
4 CycleGAN

Components to implement: a) Objective function of CycleGAN, b) Qualitative and Quantitative Evaluation. Training time for CycleGAN $\sim$ 2 hour for 1 epoch.

4.1 Method

Image-to-image translation problem describes translating an image from domain A to domain B. The main goal of CycleGAN is to solve the image-to-image translation problem without paired training images. To be more specific, the goal is to learn a mapping $G : X \rightarrow Y$. Because this mapping are highly unconstrained, CycleGAN couples G with a inverse mapping $F : Y \rightarrow X$ and proposes a cycle consistency loss to enforce $F(G(X)) \approx X$ (and vice versa). The following would be an illustration Fig.[5]:

Different from the GAN you have implemented in Section.[3], CycleGAN model contains two generators $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated discriminators D_Y and D_X, where D_Y encourages the



(a) CycleGAN [9]

(b) Forward cycle consistency: X →G(X)→F(G(X)) ≈ X [9]

(c) Backward cycle consistency: Y →F(Y)→G(F(Y))≈ Y [9]

Figure 5: An overview of CycleGAN

generated images $G(x)$ to be distinguishable from images in domain Y. And vice versa for D_X and F. As illustrated in Fig.[5], Cycle consistency consists of two parts. One is the forward cycle consistency, which encourages X →G(X)→F(G(X)) ≈ X. Another is the backward cycle consistency, which encourages Y →F(Y)→G(F(Y)) ≈ Y. We will elaborate on CycleGAN's adversarial objective and cycle consistency loss in Section.[4.2.3]

4.2 Training of CycleGAN

4.2.1 Data set

Edges2shoes data set is used for CycleGAN training. This data set is widely used in many variations of GAN network. You would be able to download this data set by running a notebook code block in colab as we did in previous projects. A snapshot of edges2shoes data set is as follows. This data set is actually paired (paired means a matched pair of images in edges domain and shoes domain), but CycleGAN is capable to train on an unpaired data set.

Figure 6: A snapshot of edge2shoe data set

For this part, a data pre-processing as follows would be applied:

1. resize image size to 128 × 128.
2. normalize the pixel value in the range [-1,1]

4.2.2 Network Architecture

The network is already implemented in the code template for you. For your knowledge, the details of the network are presented as below.

Generator Architecture: Naming convention is as follows, c7s1-k : a 7×7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1, dk : a 3 × 3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2, Rk : a residual block that contains two 3 × 3 convolutional layers with the same k number of filters on both layer, uk : an Upsampling(scale factor=2) followed by a 3 × 3 Convolution-InstanceNorm-ReLU layer with k filters.

Components of the generators as per the paper are as follows, a high-level illustration is provide in Fig.[8]:

A. Encoder:

   (a) Initial Convolution Block : c7s1-64

   (b) Downsampling : d128, d256

B. Residual blocks: R256, R256, R256, R256, R256, R256

C. Decoder:

   (a) Upsampling : u128, u64

   (b) Output Convolution Block: c7s1-3

In the implementation, reflection padding is performed before every convolution layers in the A(a)-initial convolution block, B-Residual blocks and C(b)-output convolution block. Reflection padding pads the input tensor using the reflection of the input boundary. An example of this is given in Fig.[7]. This is done for reducing artifacts/edge effects that padding with zero might introduce.

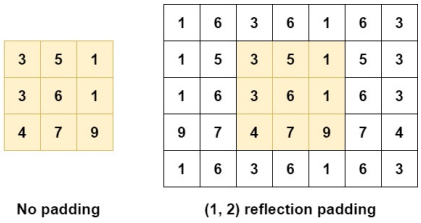


Figure 7: Schematic of no padding vs reflection padding

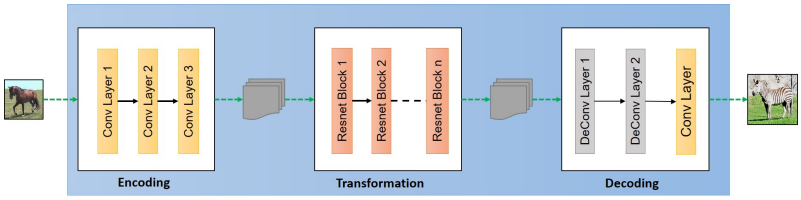


Figure 8: A high level diagram of the generator architecture. Note that A(a)-initial convolution block is Convolution Layer 1 in the Encoding block and the C(b)-output convolution block is Convolution Layer in Decoding block in this diagram.

Discriminator Architecture: Naming convention is as follows, Ck is a 4×4 Convolution-InstanceNorm-LeakyReLU layer with k filters and stride 2. At the last layer, a 4×4 Convolution layer is applied to produce a 1-channel output. The discriminator architecture is thus: C64-C128-C256-C512, followed by a last convolution layer. As illustrated in Fig.[9]
This convolution architecture is the same as the discriminator from PatchGAN [4]. Given an input image of size $C \times H \times W$, the discriminator output has shape $(1, H/2^4, W/2^4)$. The number of layers and kernel parameters of the discriminator architecture are configured such that every spatial unit in this output has a receptive field of a 70×70 patch in the image. Thus for every patch in the image, the discriminator output is a scalar value indicating whether the patch is real or fake. Conventionally, a discriminator would classify an entire image as a real or fake, but dividing it into patches and classifying each patch instead captures high frequency components (For example, edges in the image) and forces sharper looking pictures in generation process.

4.2.3 Objective function

The standard CycleGAN objective consists of three terms, namely the GAN objective from domain X to domain Y, GAN objective from domain Y to domain X, and cycle consistency loss. The overall cycleGAN objective could be expressed in the following form:

$$\begin{aligned} \mathcal{L}\left(G, F, D_X, D_Y\right) = & \mathcal{L}_{\mathrm{GAN}}\left(G, D_Y, X, Y\right) \\ & + \mathcal{L}_{\mathrm{GAN}}\left(F, D_X, Y, X\right) \\ & + \lambda \mathcal{L}_{\mathrm{cyc}}(G, F) \end{aligned} \tag{5}$$

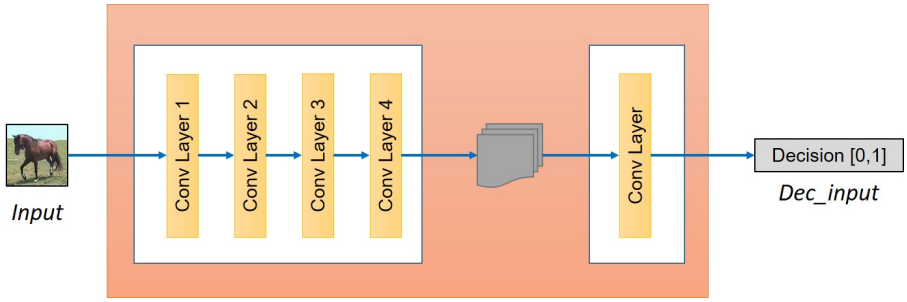


Figure 9: A high level diagram of the discriminator architecture.

The high-level functionality of G, F, D_X and D_Y are illustrated in Section.[4.1]. The detailed definition of each terms are as follows:
The **first** and **second term** in the objective function are both GAN objectives which have been introduced in Section.[3.2.3]. Notice here, however, the input to the generator G and F in CycleGAN is not random sample z from the latent space as introduced in the previous GAN part. In both GAN objectives, we want to optimize a min-max game as mentioned in Section.[3.2.3].

$$\mathcal{L}_{\text{GAN}}\left(G, D_Y, X, Y\right) = \mathbb{E}_{y \sim p_{\text{data}}(y)}\left[\log D_Y(y)\right] + \mathbb{E}_{x \sim p_{\text{data}}(x)}\left[\log\left(1 - D_Y(G(x))\right)\right] \tag{6}$$

Similarly, we would have,

$$\mathcal{L}_{\text{GAN}}\left(F, D_X, Y, X\right) = \mathbb{E}_{x \sim p_{\text{data}}(x)}\left[\log D_X(x)\right] + \mathbb{E}_{y \sim p_{\text{data}}(y)}\left[\log\left(1 - D_X(G(y))\right)\right] \tag{7}$$

The **third term** $\mathcal{L}_{\text{cyc}}(G, F)$ is cycle consistency loss. With first and second term, GAN objectives enforces the generated image distribution approaching the real image distribution. One problem with such GAN objectives alone would be that the mapped image would be any permutation of the input image with a large capacity generator network. Cycle consistency term is introduced as a regularization to reduce the space of possible mapping of generators. As illustrated in Fig.[5b] and [5c], cycle consistency loss tries to minimize the difference between the original image and reconstructed image. Cycle consistency loss is defined as follows Equ.[8]. Here, we want to minimize this loss.

$$\begin{aligned}\mathcal{L}_{\text{cyc}}\left(G, F\right) = &\mathbb{E}_{x \sim p_{\text{data}}(x)}\left[\|F(G(x)) - x\|_1\right] \\ &+ \mathbb{E}_{y \sim p_{\text{data}}(y)}\left[\|G(F(y)) - y\|_1\right]\end{aligned} \tag{8}$$

4.2.4 Implementation Details

1. The implementation detail of Equ.[6][7] are similar as mentioned in Section.[3.2.4] with a different criterion function, MSE (mean square error) instead of BCE (binary cross entropy). As before, treat maximizing $log(D(x))$ and minimizing $D(G(x))$ in discriminator as computing MSE between $log(D(x))$ and 1 and between $D(G(x))$ and 0. maximizing $log(D(G(x))$ in generator as computing MSE between it and 1. Cycle consistency loss is computed based on L1 norm.

2. The λ in Equ.[5] is set to be 10.

3. The default batch size is set to be 1. The default training epoch is 2 due to long training time for one epoch. Feel free to change the training iterations to a larger or smaller number. This is not a fixed requirement.

5 Report

5.1 Code submission

In this project assignment, please fill in the code blocks in the two provided notebooks. You will implement the VAE, GAN and CycleGAN. Please submit two .ipynb files to the Gradescope.

5.2 PDF report submission

5.2.1 Training curves

- For VAE part, plot the reconstruction loss curve and KL divergence loss for VAE network.
- For GAN part, plot the generator loss and discriminator loss curve in the same figure.
- For CycleGAN part, plot the loss curve for generators G, F and discriminators D_Y, D_X. You can plot all these 4 losses in the same figure. Please also plot the cycle consistency loss in a separate figure.

5.2.2 Qualitative Evaluation

- For VAE part, you need to generate 2 visualization images. One during the testing, generate 36 images (in 6-by-6 grid as the example below Fig.[10]) based on sampling 36 random noise vectors from the standard Gaussian distribution latent space. Secondly, plot the reconstruction output of the decoder given any 36 input images in Fashion-MNIST.

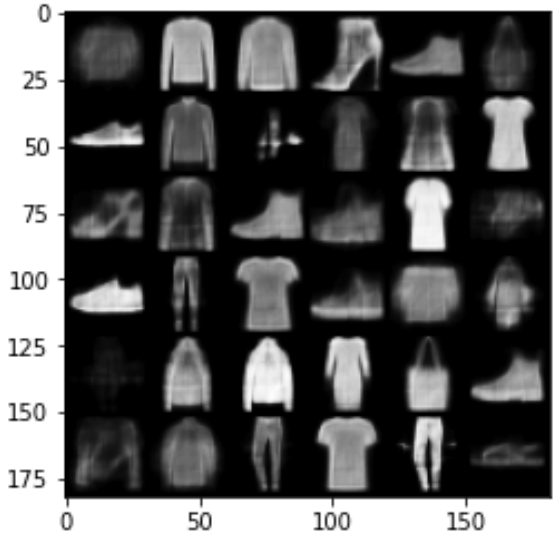


Figure 10: example of VAE generated image in testing

- For GAN part, generate 2 sets of visualization images. For the first set, visualize the generated 36 images (in 6-by-6 grid) at different epochs to observe how generator improves. You could choose to visualize every 10 epochs or any sequence of epochs for this visualization. Please include at least 5 visualization images from different training epochs. To further clarify, each visualization image is a 6×6 grid of generated images. For the second set, when the network is trained, use the generator network to generate 36 images (in 6-by-6 grid) based on sampling 36 random noise vectors from standard Gaussian distribution latent space.
- For CycleGAN part, visualize real_edge, fake_shoe, real_shoe, fake_edge in 2-by-2 grid figure, where real_edge and real_shoe are coming from test set. Use trained Generator network G and F to generate fake_shoe and fake_edge based on real images. The following Fig.[11] could be an example. Please include 4 of such 2-by-2 grid figures in the report.

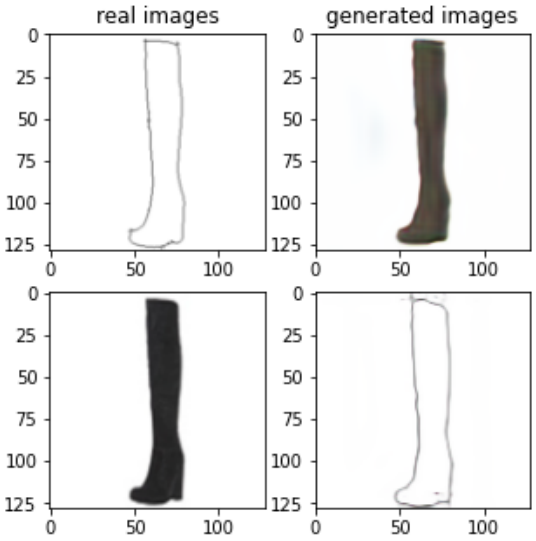


Figure 11: example of CycleGAN generated image in testing

5.2.3 Quantitative Evaluations

1. For VAE, compute two quantitative evaluations as follows:

   Reconstruction Error: Here we compute $MSE(x, \hat{x})$ over test set images and their reconstructed output from VAE as our reconstruction error. $\hat{x}$ is the output of VAE network and X is the input image. Flatten the input image into vectors in order to compute $MSE(x, \hat{x})$. Please compute this score every 100 iterations in training. And plot the reconstruction error curve along training. This is the metric monitoring the potential over-fitting of our network. The module of MSE computation is to be implemented in the code block.

   Inception Score (IS): IS score measures the quality and diversity of the input image set. It is one of the most widely adopted score for generative model evaluation[1]. It uses Inception network V3 pre-trained on ImageNet to predict the class labels represented as a likelihood $P(y|x_i)$ for generated image x_i. IS measures the average KL divergence between the conditional label distribution $p(y|x)$ of images (expected to have low entropy for easily classifiable images since their class is highly predictable. This means good image quality) and the marginal distribution $p(y)$ obtained from all the samples (expected to have high entropy if all classes are equally represented in the set of images that means high diversity). So ideally, a good generator will give us the following two distributions Fig.[12]. In general, a high IS score indicates good performance in photo-realistic and diversity.

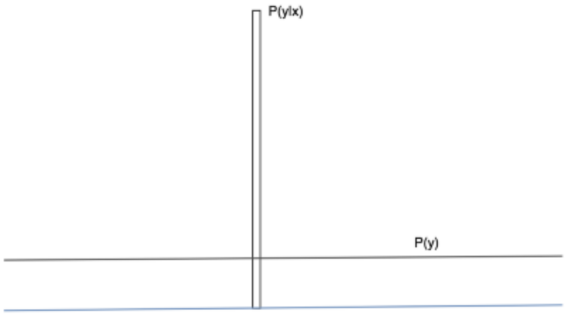


Figure 12: ideal distribution for $P(y|x_i)$ and $P(y)$ Image Credit.

IS score computation equation is as follows Equ.[9], the detailed explanation, interpretation and implementation steps will be in Appendix Section.[6].

$$IS(x) = \exp\left(\mathbb{E}_{\mathbf{x}}[\mathbb{KL}(p(\mathbf{y} \mid \mathbf{x}) \| p(\mathbf{y}))]\right) = \exp\left(H(y) - \mathbb{E}_{\mathbf{x}}[H(y \mid \mathbf{x})]\right) \tag{9}$$

You could use existing API that is downloaded at the beginning of code template to complete the IS score. If you are interested in the detail of IS implementation, refer to the detailed implementation steps in Appendix Section.[6.3] and corresponding code template in notebook. Please fill IS score in the Table.[5].

2. For GAN and CycleGAN, you would be asked to compute two quantitative evaluations as follows:

Fréchet Inception Distance (FID):

FID scores captures the photo-realistic quality of generated images by measuring the distance between real image set and generated image set. We embed a set of generated images into a feature space given by the output of final avgpool layer of Inception Net v3. Viewing the embedding as a continuous multivariate Gaussian, the mean and covariance matrix are estimated for both the generated data and the real data. The Fréchet distance between these two Gaussian distributions is used to quantify the quality of generated samples.

$$FID(r, g) = \|\mu_1 - \mu_2\|_2^2 + Tr\left(\Sigma_1 + \Sigma_2 - 2\left(\Sigma_1\Sigma_2\right)^{\frac{1}{2}}\right) \tag{10}$$

According to above equation, a lower FID value means the compared image sets have a closer distribution in the feature space.
Please calculate the FID score pair-wise between 3 image sets, namely, real image set 1, real image set 2 and generated image set provided in Table.[6]. For GAN, real image set contains images in STL-10 test data set and generated image set contains generated images given random noise z in latent space. For CycleGAN, there are two real image domains, you would need to compute pair-wise FID scores between image sets: real_edge 1, real_edge 2, generated_edge, real_shoe 1, real_shoe 2, generated_shoe provided in Table.[7]. Noted here two real data sets in the same domain should not have overlaps for both GAN and CycleGAN. For each data set created for GAN evaluation should have 1000 images. And 100 images for each data set created for CycleGAN evaluation.

You can use the existing API provided in colab to compute FID score and fill in the Table.[6]. Implementation details for FID computation are provided in Appendix Section.[6.2], you are encouraged to follow this instruction and implement FID. The FID score implementation is based on this github page FID-Github.

Inception Score (IS): Please follow the same instruction in Section.[5.2.3] to compute the IS score for the data set in Table.[6][7]

Image Set	IS VAE
Real image set	
Generated image set	

Table 5: Fill in the IS score in this Quantitative Evaluation Table

Image Set Pair	FID GAN	Image set	IS GAN
STL-10 real set 1 vs STL-10 real set 1		STL-10 real set 1	
STL-10 real set 1 vs STL-10 real set 2		STL-10 real set 2	
STL-10 real set 1 vs STL-10 generated set		STL-10 generated set	

Table 6: Fill in the FID and IS score in this Quantitative Evaluation Table

Image Set Pair	FID CycleGAN	Image set	IS CycleGAN
real_edge 1 vs real_edge 2		real_edge1	
real_edge 1 vs generated_edge		generated_edge	
real_shoe 1 vs real_shoe 2		realshoe 1	
real_shoe 1 vs generated_shoe		generatedshoe	

Table 7: Fill in the FID and IS score in this Quantitative Evaluation Table

6 Appendix

6.1 Inception network v3

The **Inception Network v3** [8] is a deep convolution architecture designed for classification tasks on ImageNet [2], a dataset consisting of 1.2 million RGB images from 1000 classes. Given an image x, the task of the network is to output a class label y in the form of a vector of probabilities $p(y|x) \in [0,1]^{1000}$, indicating the probability the network assigns to each of the class labels. The Inception network v3 is one of the most widely used networks for transfer learning and pre-trained models are available in most deep learning software libraries.

It has the following network architecture Fig.[13]:

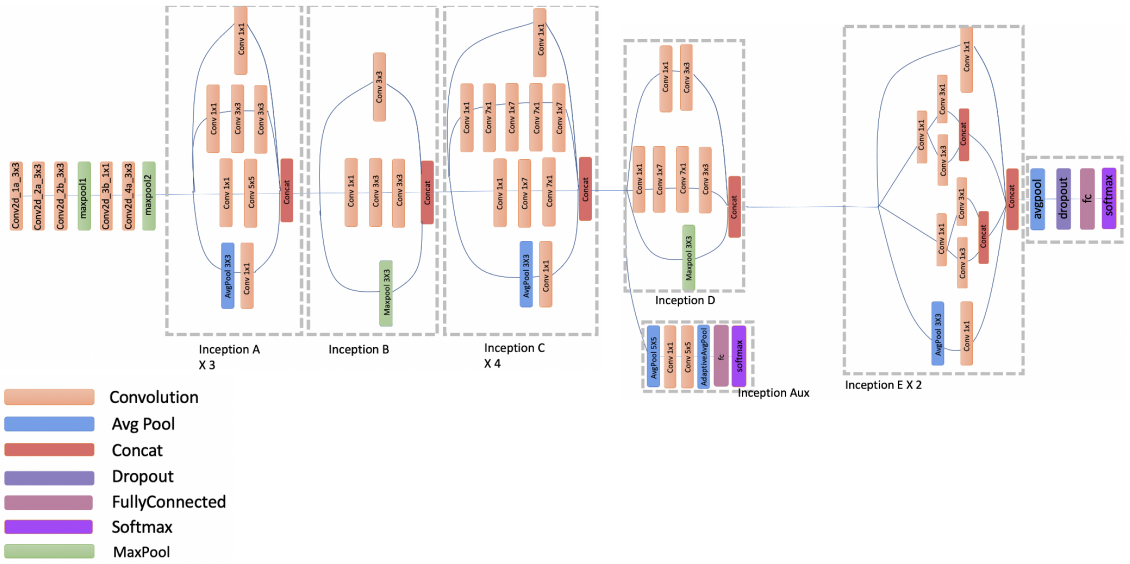


Figure 13: The architecture of Inception Network V3

For the quantitative evaluation of the generated images, we shall feed training or synthesized images to the Inception network and calculate statistics like mean and covariance on the network outputs. **Inception Score(IS)** and **Frechet Inception Distance(FID)** are two evaluation methods, that are based on these statistics of feature vectors.

6.2 Implementation detail of FID

This section provides implementation details of FID score which has been introduced in Section.[5.2.2].

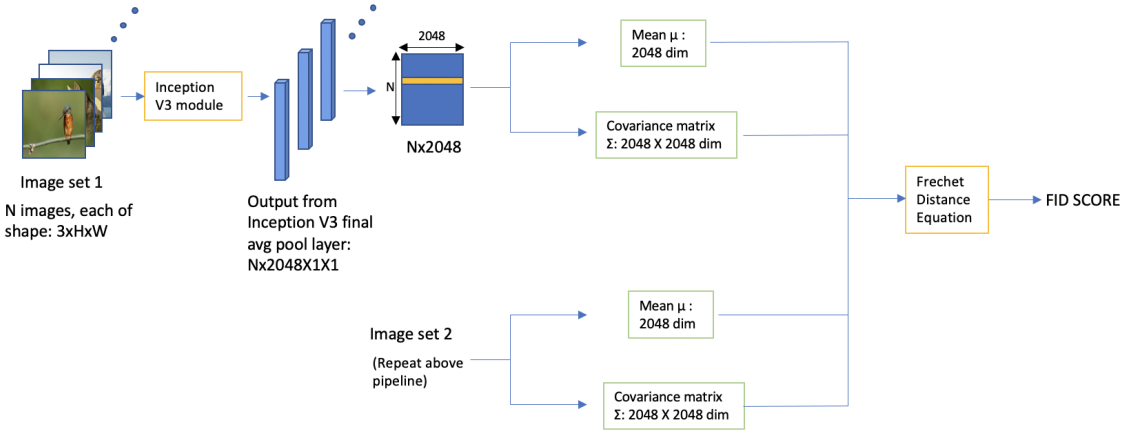


Figure 14: FID pipeline flow-chart

Given two sets of images, image set 1 and image set 2. FID score is computed in the following steps. Template code is also provided:

1. We start with a torch dataset of dimension $(N, 3, H, W)$, which contains RGB image set 1. Here N is the number of images in the data set. And (H, W) is the resolution of each image.

2. In function "build_feature_table". First, create a torch dataloader based on dataset. Then, forward image batch through the provided Inception V3 network to obtain the feature representation: In FID computation the last layer output of "avgpool" after Inception E module in Fig.[13] is used as features. Each feature is a 2048 dimension vector. Each of the batch output feature would be stacked into a matrix shown in the flow-chat .[14]. This is what we called as feature table. Note, here N may not be a multiple of the batch size, please use a index counter to keep track of the image index. For example, N is 1000 and batch size is 64. We would have a batch size of 40 in the last batch.

3. From feature table, compute the mean and covariance, namely μ_1 and Σ_1 in function "compute_stat". Our feature table is of dimension $(N, 2048)$. So the mean would be a dimension vector with dimension 2048 and covariance matrix would have dimension $(2048, 2048)$.

4. We repeat the above steps for data set 2 to get μ_2 and Σ_2.

5. After obtaining μ_1, Σ_1, μ_2 and Σ_2, implement the Equ.[10] in the code block for FID score computation.

6.3 Implementation detail of IS

In this section, we will elaborate on implementation details of IS score which has been introduced in Section.[5.2.3].

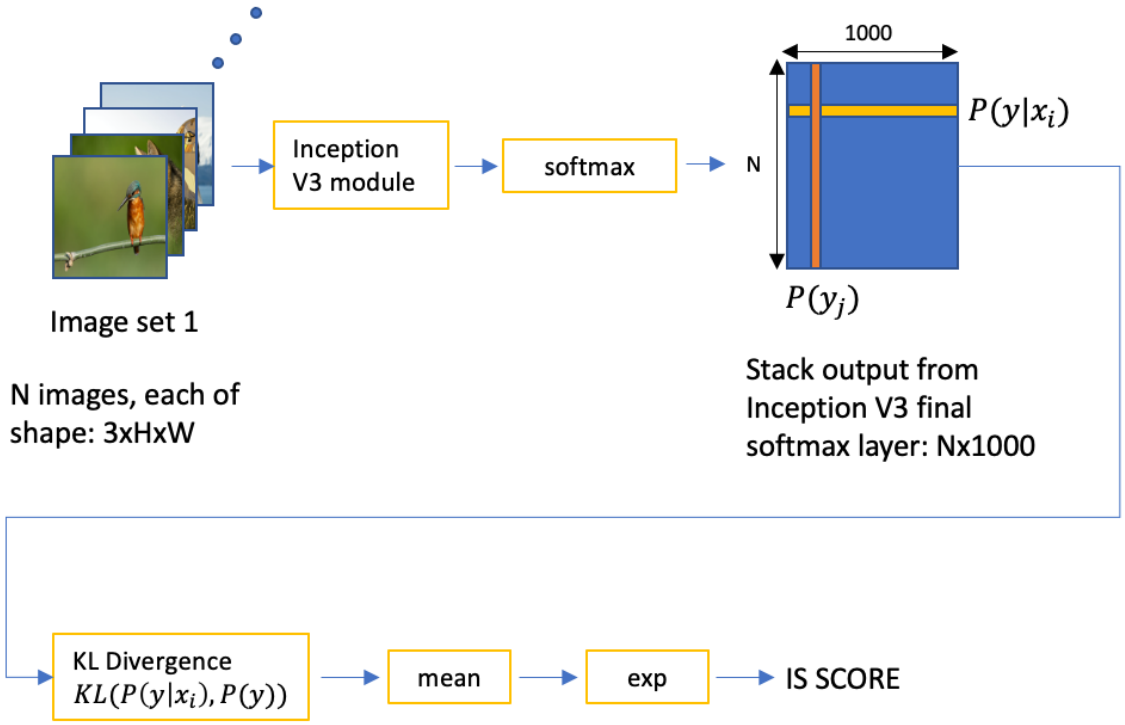


Figure 15: IS pipeline flow-chart

IS score is computed in the following steps Fig.[15]:

1. Start with a torch dataset of dimension $(N, 3, H, W)$, which contains RGB images. Note, the Fashion-MNIST data set has grayscale images, so before sending it to IS computation pipeline, we need to repeat its gray color channel 3 times to have the shape $(N, 3, H, W)$. Here N is the number of images in the data set. And (H, W) is the resolution of each image. Inception network v3 only accepts resolution 299×299. Thus, up-sample the $H \times W$ image into 299×299.

2. In the template code, function "build_feature_table", is similar to point.[2] in FID computation in Section.[6.2]. Construct a matrix which contains stacked feature vectors. The difference here, however, is we get the final softmax classification output of the Inception network v3, likelihood over 1000 classes, as the feature vectors. In the Fig.[13], that is the output of last softmax layer. Stacked feature matrix is of shape $(N, 1000)$.

3. After obtaining the feature matrix, each row represents distribution $P(y|x_i)$ and the mean of each column represents $P(y_j)$. Because, $P(y_j) = \sum_{x_i} P(y_j|x_i)P(x_i)$ where $P(x_i) = \frac{1}{N}$. Then, compute the KL divergence $KL(P(y|x_i), P(y))$ for the input x_i.

4. Finally, average over these KL divergences and apply an exponential function to the resulting mean value in order to get the IS score.

References

[1] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[5] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[7] Thalles Silva. An intuitive introduction to generative adversarial networks (gans). *freeCodeCamp. org*, 2018.

[8] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[9] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.