Wasserstein Auto-Encoder using Residual Neural Network

By Md Azad Ansari

WORD COUNT	7398	TIME SUBMITTED	19-JUL-2022 05:22PM
		PAPER ID	88575536

Wasserstein Auto-Encoder using Residual Neural Network

Md Azad Ansari

Wasserstein Auto-Encoder using Residual Neural Network

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Md Azad Ansari

[Roll No: CS2039]

Under the Guidance of

Dr. Rajat K De

Professor

Machine Intelligence Unit

Indian Statistical Institute

Kolkata-700108, India

July 2022

To my family and my guide

CERTIFICATE

This is to certify that the dissertation entitled **"Wasserstein Auto-Encoder using Residual Neural Network"** submitted by **Md Azad Ansari** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this Institute in my opinion, has reached the standard needed for submission.

Rajat Kumar De

Dr. Rajat K De
Professor,
Machine Intelligence Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgments

I would like to show my highest gratitude to my advisor, *Prof. Dr. Rajat K De*, Machine Intelligence Unit, Indian Statistical Institute, Kolkata, for his guidance and continuous support and encouragement. He has literally taught me how to do good research, and motivated me with great insights and innovative ideas.

My deepest thanks to all the teachers of Indian Statistical Institute, for their valuable suggestions and discussions which added an important dimension to my research work.

I would also like to thank Anish, Ayush, Soham, and Sachin for his valuable suggestions and discussions.

Finally, I am very much thankful to my parents and family for their everlasting supports.

Last but not the least, I would like to thank all of my friends for their help and support. I thank all those, whom I have missed out from the above list.

Md. Azad Ansari

Md Azad Ansari
Indian Statistical Institute
Kolkata - 700108 , India.

Abstract

We propose the Wasserstein Auto-Encoder using Residual Neural Network (WAE-Resnet)— an improved version of Wasserstein Auto-Encoder [7] for building a generative model of the data distribution. WAE is an alternative to Variational Auto-encoders (VAE) as a method of getting the encoded data distribution to match the prior distribution, which can be used for generative modeling.

WAE-ResNet minimizes a penalized form of the Wasserstein distance between the latent model distribution and the target distribution, which used a different regularizer from the one used by the Variational Auto-Encoder (VAE) [6]. Wasserstein distance metric is used in WAE-ResNet to penalize the distance between the encoded distribution and the prior, as opposed to the KL-divergence term used in VAEs.

We used ResNet architecture to improve the preservation of the features at the latent space in WAE-ResNet. We implemented VAE, VAE-ResNet, WAE, and WAE-ResNet. WAE-ResNet gives better results compared to the others. Generated image quality is measured by Frechet Inception Distance (FID) score and Sharpness score. WAE-ResNet has outperformed the other models with respect to FID and Sharpness scores.

Contents

Chapter 1

Introduction

Deep learning-based generative models have been increasingly popular over the past few years as a result of some incredible advancements in the area. Deep generative models have demonstrated an amazing capacity to develop highly realistic contents of many kinds, such as images, texts, and sounds, by relying on well-designed network architectures, a vast quantity of data, and smart training techniques.

Initially, supervised learning approaches were used to drive the field of representation learning, with impressive results using huge labeled data-sets. Unsupervised generative modeling, in contrast, is used by probabilistic approaches focusing on low-dimensional data-sets. Variational Auto-encoders (VAEs), while theoretically a well-established method, have the limitation that when used on real-world images, they frequently provide samples that are blurry [4.4]. Generative Adversarial Networks (GANs) [1] generate more impressive visual image quality when applied to natural images, but come without an encoder. GANs have also drawbacks of being harder to train and "mode collapse" problem.

Wasserstein Auto-Encoders (WAEs) is a generative model which generate fake images. WAEs use wasserstein distance to minimize the loss of auto-encoder instead of KL divergence and JS divergence used in VAEs and GANs respectively. WAE does not generate blurry image and also it handled the "mode collapse" problem.

If we make a deeper neural network using a more convolution layer, the "vanishing gradient problem" occurs. This happens because of the gradient-based learning process and back-propagation in neural networks. Due to the "vanishing gradient problem", the network stops learning. To solve this problem ResNets are used. In the Residual neural network, input from the previous layer is added directly to the output of the other layer. Based on this observation, we combined the architectures of VAE/WAE and ResNet to build a new model for the generative modeling task.

Using the ResNet architecture, we build an Encoder that will encode better information at latent space variable specified by prior Gaussian distribution so that generator model (decoder) can generate better image samples. We have used three 2D convolutional layer to build the identity block and three 2D convolutional layer to build the convolutional block. Using these identity and convolutional blocks, we build the Encoder. Encoder takes input as image of size 128×128 and outputs 1D tensor of size 64 and Decoder takes 1D tensor of size 64 and outputs image of size 128×128

We trained VAE, VAE-ResNet, WAE, and WAE-ResNet on a real data-set called "CelebA" which has roughly 203k images. After evaluation, the FID score and sharpness score of the generated test sample of WAE-ResNet model are better than other implemented models.

Chapter 2

Preliminaries and Previous Works

Before introducing to my proposed framework, let us have some idea about the pre-requisite concepts which have been used in my work.

2.1 Variational Auto-encoders (VAEs)

According to the definition of a variational auto-encoder [6], it is an auto-encoder whose training is regulated to prevent over-fitting and to guarantee that the latent space distribution has favourable qualities that support the generating process. Just as a basic non-stochastic auto-encoder, a variational auto-encoder is an architecture composed of both an "encoder" which encodes the data in latent space and a "decoder" that decodes the latent space to the original dimension. While training we minimize the reconstruction loss between the initial input data and encoded-decoded data. In VAEs, we encode input data as distribution at latent space instead of encoding as a single point.

A variational autoencoder contains a specific probability model of data x and latent variables z. Now, joint probability of the model is given as

$$p(x, z) = p(x|z)p(z).$$

The following is a description of the generative process..

For every data point i:

- draw latent variable z_i from latent space $p(z)$
- draw data-point x_i from $p(x|z)$

The latent space variables are taken from prior distribution $p(z)$. The likelihood of data x is $p(x|z)$ conditioned on latent variable z. $p(x, z)$ is the joint probability distribution over the data and latent space variables.

Now, decompose this into likelihood and prior:

$$p(x, z) = p(x|z)p(z).$$

Now, our goal is to infer a good latent space given the original data.

By Bayes theorem:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

$p(x)$ is called the evidence and it is calculated by marginalizing the latent space variables:

$$p(x) = \int p(x|z)p(z)dz$$

Unfortunately, the above equation takes exponential time to compute. Therefore, we need to approximate this posterior distribution.

A family of distributions $q_\lambda(z|x)$ is used to approximate the posterior. λ is the variational parameter which indexes the family of distributions. Here, we are taking q as Gaussian distribution. So, it will be the mean and variance of the latent space variables for every data-point $\lambda_{x_i} = (\mu_{x_i}, \sigma^2_{x_i})$.

Kullback-Leibler divergence is used to measure the information loss when using q posterior to approximate p posterior.

$$\mathcal{KL}(q_\lambda(z|x)||p(z|x)) = \mathbf{E}_q[\log q_\lambda(z|x)] - \mathbf{E}_q[\log p(x,z)] + \log p(x)$$

To minimize the divergence, we have to find optimal variational parameter λ.

The optimal approximation of posterior is

$$q^*_\lambda(z|x) = \arg\min_\lambda \mathcal{KL}(q_\lambda(z|x)||p(z|x)).$$

Above equation is impossible to be computed directly. So consider the following function:

$$ELBO(\lambda) = \mathbf{E}_q[\log p(x,z)] - \mathbf{E}_q[\log q_\lambda(z|x)]$$

After rewriting the evidence becomes

$$\log p(x) = ELBO(\lambda) + \mathcal{KL}(q_\lambda(z|x)||p(z|x))$$

The Jensen inequality states that the KL divergence is always larger than or equal to zero. This suggests that in order to minimise the KL divergence, we must maximise the ELBO. Now, we can approximate posterior inference using Evidence Lower BOund (ELBO). Instead of minimizing KL divergence between the approximate and true posterior, we can maximize the Evidence Lower BOund which is equivalent.

In VAEs, there are only local latent space variables. As a result, we can divide the Evidence Lower BOund into terms, each of which depends on a single data point. Now, we can apply stochastic gradient descent with respect to the parameters λ. ELBO for single data-point is given by

$$ELBO_i(\lambda) = \mathbf{E}_{q_\lambda}(z|x_i)[\log p(x_i|z)] - \mathcal{KL}(q_\lambda(z|x_i)||p(z)).$$

Let us make the connection to neural network architecture. Using inference network (Encoder), we will parameterize the approximate posterior $q_\theta(z|x,\lambda)$ that takes x as input and parameter λ as output. Using generative network (Decoder), we will parameterize the likelihood $p(x|z)$ that takes latent variables as input and output parameters to the data distribution $p_\phi(x|z)$. The Encoder and Decoder have parameters θ and ϕ respectively. These parameters are the neural network's weights and biases. We will use stochastic gradient descent to maximise the ELBO in order to optimise these. We can write the ELBO and include the Encoder and Decoder parameters as

$$ELBO_i(\theta, \phi) = \mathbf{E}_{q_\theta}(z|x_i)[\log p_\phi(x_i|z)] - \mathcal{KL}(q_\theta(z|x_i)||p(z))$$

This ELBO is the negative of the loss

$$ELBO_i(\theta, \phi) = -l_i(\theta, \phi)$$

where,

$$l_i(\theta, \phi) = -\mathbf{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + \mathcal{KL}(q_\theta(z|x_i)||p(z))$$

Algorithm 1 Algorithm for Variational Auto-encoder

```
Ensure: Initialize the parameters for the encoder $q_\theta$, decoder $p_\phi$
  while $(\theta, \phi)$ not converged do
    Sample $\{x_1, x_2, \ldots, x_n\}$ from the training set
    generate $\mu_i$ and $\sigma_i$ from $q_\theta(x_i)$ for $i = 1, \ldots, n$
    generate $\tilde{z}_i = \mu_i + \sigma_i\epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, 1)$ for $i = 1, \ldots, n$
    Update $q_\theta$ and $p_\phi$ by descending:
        $-\mathbf{E}_{\tilde{z}_i\sim q_\theta(z|x_i)}[\log p_\phi(x_i|\tilde{z}_i)] + \mathcal{KL}(q_\theta(\tilde{z}_i|x_i)||p(z))$
  end while=0
```

2.2 Wasserstein Auto-Encoder

In Wasserstein Auto-Encoder [7], we try to minimize the optimal transport cost [8] $W_c(P_X, P_G)$ between the true data distribution P_X and the latent variable model P_G specified by prior distribution P_Z of latent codes $Z \in \mathcal{Z}$ and the generative model $P_G(X|Z)$ of the data points $X \in \mathcal{X}$ given Z. The decoder tries to precisely reconstruct the encoded training data as measured by the cost c in the resulting optimization problem. The encoder aims to accomplish two opposing objectives: It first attempts to match the training data's encoded distribution $Q_Z := \mathbf{E}_{P_X}[Q(Z|X)]$ to the prior P_Z which is determined by any divergence $\mathcal{D}_Z(Q_Z, P_Z)$, and second, it has to be ensured that the latent space codes which are provided to the decoder are enough information so that it reconstructs the encoded training data.

2.2.1 Optimal transport and its dual formulation

Kantorovich's [1] formulation of the problem is defined as:

$$W_c(P_X, P_G) := \inf_{\Gamma \in \mathcal{P}(X \sim P_X, Y \sim P_G)} \mathbf{E}_{(X,Y)\sim\Gamma}[c(X,Y)] \tag{2.1}$$

where $\mathcal{P}(X \sim P_X, Y \sim P_G)$ is a set of all joint distributions of (X, Y) with marginal P_X and P_G respectively, and $c(x, y) : \mathcal{X} \times \mathcal{X} \to \mathcal{R}_+$ is any measurable cost function. A specific interesting case is when $(\mathcal{X}, d)$ is metric space and $c(x, y) = d^p(x, y)$ for $p \geq 1$. The p-Wasserstein distance is defined here as W_p, or the p-th root of W_c.. The following Kantorovich-Rubinstein duality holds, if $c(x, y) = d(x, y)$:

$$W_1(P_X, P_G) = \sup_{f \in \mathcal{F}_L} \mathbf{E}_{X\sim P_X}[f(X)] - \mathbf{E}_{Y\sim P_G}[f(Y)] \tag{2.2}$$

where $\mathcal{F}_L$ is the class of all functions that are 1-Lipschitz bounded on $(\mathcal{X}, d)$.

Now, latent variable model P_G is defined as:

- first, on a latent space $\mathcal{Z}$, a code z is taken as a sample from a fixed distribution P_Z.
- second, with a possible random transformation, z is mapped to the image $x \in \mathcal{X} = \mathcal{R}^d$

This results in

$$p_G(x) := \int_{\mathcal{Z}} p_G(x|z)p_z(z)dz, \qquad \forall x \in \mathcal{X} \tag{2.3}$$

For the shake of simplicity, we will take non-random decoders, i.e. generative models

$P_G(X|Z)$, which deterministically map Z to $X = G(Z)$ for a given map $G : \mathcal{Z} \to \mathcal{X}$.

Now, under this model, through the map G, the optimal transport cost takes a simpler form as the transportation plan factors: Instead of finding a coupling Γ in equation(2.1) between two random variables belonging to $\mathcal{X}$ space, it is sufficient to identify a conditional distribution $Q(Z|X)$ such that its Z marginal $Q_Z(Z) := \mathbf{E}_{X\sim P_X}[Q(Z|X)]$ is similar to the prior distribution P_Z, .

The objective of WAE is

$$D_{WAE}(P_X, P_G) := \inf_{Q(Z|X)\in\mathcal{Q}} \mathbf{E}_{P_X}\mathbf{E}_{Q(Z|X)}[c(X, G(Z))] + \lambda.\mathcal{D}_Z(Q_Z, P_Z) \qquad (2.4)$$

where $\mathcal{D}_Z$ is an arbitrary divergence between Q_Z and P_Z, $\mathcal{Q}$ is any non-parametric set of probabilistic encoders, and $\lambda > 0$ is a hyper-parameter. Similar to VAE, WAE also uses deep neural networks to parameterize the encoders Q and decoders G.

Algorithm 2 Algorithm for Wasserstein Auto-encoder

```
Require: Regularization coefficient $\lambda > 0$.
  Initialize the parameters of the encoder $Q_\theta$, decoder $G_\phi$ and latent discriminator $D_\gamma$.
  while $(\theta, \phi)$ not converged do
    Sample $\{x_1, x_2, \ldots, x_n\}$ from the training set
    Sample $\{z_1, z_2, \ldots, z_n\}$ from the prior $P_Z$
    Sample $\tilde{Z}_i$ from $Q_\theta(Z|x_i)$ for $i = 1, \ldots, n$
    Update $D_\gamma$ by ascending:
        $\frac{\lambda}{n}\sum_{i=1}^{n}[\log D_\gamma(z_i) + \log(1 - D_\gamma(\tilde{z}_i))]$
    Update $Q_\theta$ and $G_\phi$ by descending:
        $\frac{1}{n}\sum_{i=1}^{n}[c(x_i, G_\phi(\tilde{z}_i)) - \lambda.\log D_\gamma(\tilde{z}_i)]$
  end while=0
```

2.2.2 GAN-based penalty

As a penalty, Choose $D_Z(Q_Z, P_Z) = D_{JS}(Q_Z, P_Z)$ and apply adversarial training to estimate it. Real data points sampled from P_Z are distinguished from fake data points taken from Q_Z using an adversary, or discriminator, in the latent space Z. This result is the WAE, GAN based penalty which is described in above Algorithm 2.

2.3 ResNet

More layers are added to a deep neural network in every successful architecture to reduce the error rate after the first CNN-based architecture (AlexNet) that won the ImageNet 2012 competition. This is effective for smaller numbers of layers, but when we add more layers, a typical deep learning issue known as the Vanishing/Exploding gradient arises. This results in the gradient becoming zero or being overly large. Therefore, the training and test error rate similarly increases as the number of layers is increased.

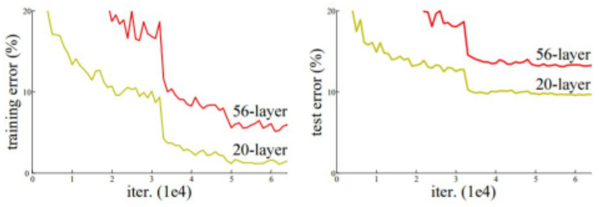


Figure 2.1: Training and Test error

We can see from the following figure that a 20-layer CNN architecture performs better on training and testing datasets than a 56-layer CNN architecture. The authors came to the conclusion that error rate is caused by vanishing/exploding gradient after further analysis of error rate.

This architecture introduces the idea of Residual Blocks to address the issue of the vanishing/exploding gradient. They employ the skip connections method in this network. By skipping some layers in between, the skip connection connects activations of a layer to subsequent layers. This forms a residual block. By stacking these residual blocks, ResNets [2] are created.

2.4 Fréchet Inception Distance

Fréchet Inception Distance (FID) [3] is a metric for evaluating the performance of generative networks. The FID score compares the collections of generated images from generator and collections of real images from the given data-set.

Inception v3 model is used to capture the features of input images. Then the mean and the variance of the features are calculated. These statistics are calculated for both real and fake generated collection of images. Then, the distance between the real and the fake images are calculate by Fréchet distance.

$$d^2 = \|\mu_1 - \mu_2\|_2^2 + \mathrm{Tr}\{C_1 + C_2 - 2(C_1C_2)^{1/2}\}$$

where, d^2: distance with square unit, μ_1: feature-wise mean of real images, μ_2: feature-wise mean of fake images, C_1: feature-wise covariance of real images, C_2: feature-wise covariance of fake images, and $\|\mu_1 - \mu_2\|_2^2$: the sum of the squares of the means of the two vectors.

2.5 Sharpness

It is a metric for evaluating the sharpness of images. First, convert 'rgb' image into 'gray' scale. Second, calculate its gradient. Third, add the square of the gradients. Fourth, calculate square-root. Sixth, calculate average.

2.6 Activation Functions

2.6.1 ReLU

The rectified linear activation function, or (ReLU), is a linear function that, if the input is positive, outputs the input directly; if not, it outputs zero.

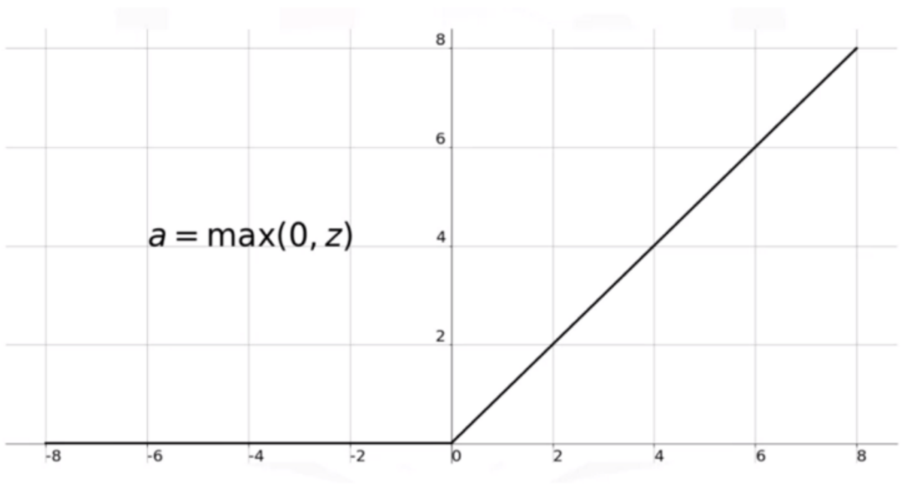


Figure 2.2: ReLU Activation Function

2.6.2 SELU

Scaled Exponential Linear Units (SELU), are activation functions that induce self-normalization. SELU activation functions automatically converge to a zero mean and unit variance.

$$SELU(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases}$$

where,

$$\alpha \approx 1.6732632423543772848170429916717$$

$$\lambda \approx 1.0507009873554804934193349852946$$

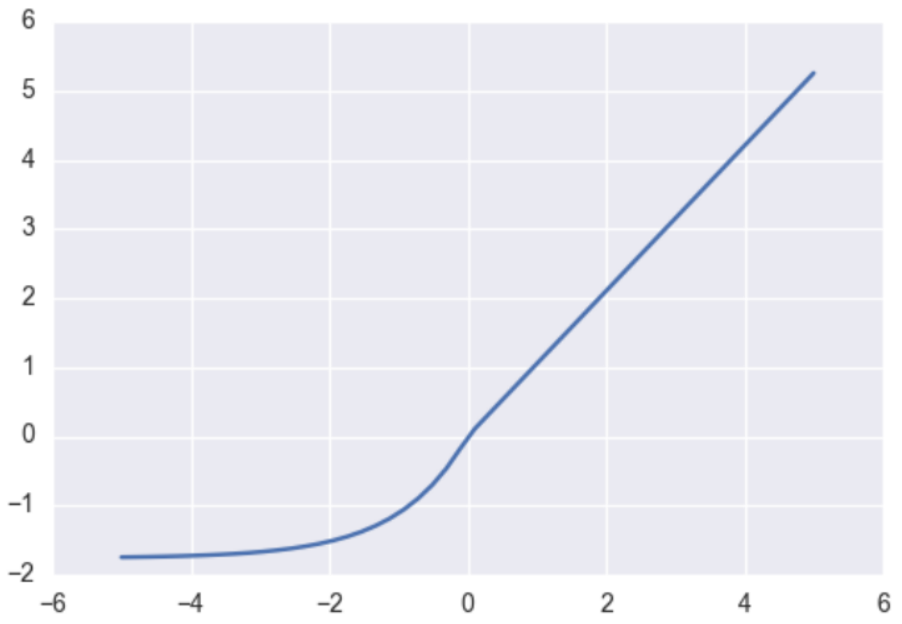


Figure 2.3: SELU Activation Function

Chapter 3

Proposed Works

3.1 Introduction

In this chapter, we describe the proposed work which improves the WAE FID and Sharpness score. WAE uses the simple convolution layer to build the architecture of Encoder, Decoder, and Discriminator. After experiment on building the architecture of auto-encoder, we conclude that to improve the FID score of WAE, we have to build the deeper (increase number of layers) architecture of encoder.

The major benefit of the very deep neural networks are that they can represent very complex functions. However, one problem is encountered while training i.e. "vanishing gradient problem". This happens because of the gradient-based learning process and back-propagation in neural networks. We know that to update the weights and biases of the networks, we calculate gradients and back-propagate the gradients through the networks. But sometimes gradient becomes vanishingly small and it prevents changing the value of weights and biases. Therefore, the Network stops learning because the same values are propagated again and again.

To solve this problem, we have used the idea of identity mapping as described in ResNet to build our Encoder to deal with the problem of vanishing gradient. Following such approach, we are essentially creating an ensemble of ResNet which helps in the variance reduction of the model.

3.2 Architecture of Encoder

We build the encoder which will encode better information at latent space variable Q_Z specified by prior distribution P_Z so that generator model $P_G(X|Z)$ can generate better image samples.

Now, the latent variable model P_G is defined as:

- first, on a latent space $\mathcal{Z}$, a code z is taken as a sample from a fixed distribution P_Z.
- second, with a possible random transformation, z is mapped to the image $x \in \mathcal{X} = \mathcal{R}^d$

This results in

$$p_G(x) := \int_{\mathcal{Z}} p_G(x|z)p_z(z)dz, \qquad \forall x \in \mathcal{X} \tag{3.1}$$

For the shake of simplicity, we will take non-random decoders, i.e. generative models $P_G(X|Z)$, which deterministically map Z to $X = G(Z)$ for a given map $G : \mathcal{Z} \to \mathcal{X}$.

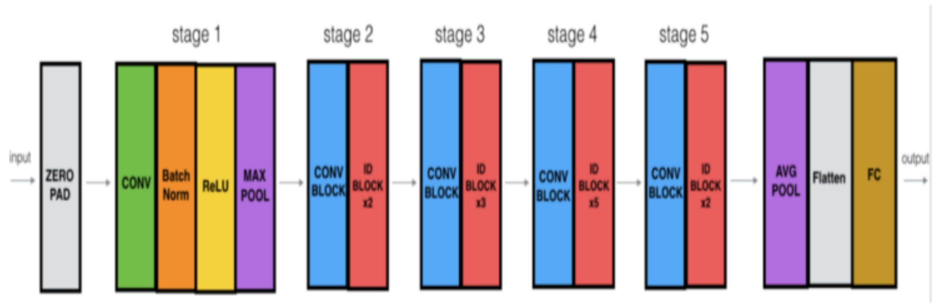


Figure 3.1: Encoder

The details of the Encoder architecture is:

- **Zero-padding:** pads the input with a pad of (2,2)

- **Stage 1:** The 2D Convolution layer has 64 filters of kernel size (7,7) with a stride of (2,2). BatchNormalization is applied to the channels axis=3 of the input. MaxPooling uses a (3,3) window with a (1,1) stride.

- **Stage 2:** The convolutional block uses three sets of filters [64, 64, 256] of kernel size f=3 with stride s=1. The two identity blocks use 3 sets of filters [64, 64, 256] of kernel size f=3.

- **Stage 3:** The convolutional block uses three sets of filters [128, 128, 512] of kernel size f=3 with stride s=2. The three identity blocks use 3 sets of filters [128, 128, 512] of kernel size f=3.

- **Stage 4:** The convolutional block uses three sets of filters [256, 256, 1024] of kernel size f=3 with stride s=2. The three identity blocks use 3 sets of filters [256, 256, 1024] of kernel size f=3.

- **Stage 5:** The convolutional block uses three sets of filters [512, 512, 2048] of kernel size f=3 with stride s=2. The three identity blocks use 3 sets of filters [512, 512, 2048] of kernel size f=3.

- **Average Pooling:** The 2D AveragePooling uses a window of shape (2,2).

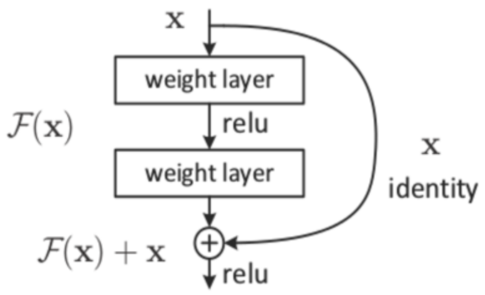
- **Flatten:** Flatten layer is used to make 1D tensor.

- **Dense:** The Fully Connected Dense layer use uints=64.

Let's look at the fundamental residual block: The shortcut connection, also known as a skip connection, is the most crucial idea in play here, as can be seen in the image below. We can summarise the fundamental residual block using the illustration below:

The output of the residual block is defined as follows if X is the input and $\mathcal{F}(X)$ is the layer's output

$$\mathcal{H}(X) = \mathcal{F}(X) + X$$


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

From residual block, by using two weight layers to process the input X, $\mathcal{F}(X)$ is produced. After that, it adds the original input data X and $\mathcal{F}(X)$ to get $\mathcal{H}(X)$. Let's now assume that $\mathcal{H}(X)$ represents our ideal predicted output, matching X as our initial input data. Obtaining the ideal $\mathcal{F}(X)$ is necessary to acquire the desired result of $\mathcal{H}(X) = \mathcal{F}(X)+X$. This implies that in order to obtain the optimum $\mathcal{H}(X)$, the two weight layers of the residual block must be able to provide the desired $\mathcal{F}(X)$.

5

$\mathcal{F}(X)$ is obtained from X as follows:

$$\mathcal{F}(X) : X \to W_1 \to ReLU \to W_2$$

$\mathcal{H}(X)$ is obtained from $\mathcal{F}(X)$ as follows:

$$\mathcal{H}(X) : \mathcal{F}(X) + X$$

The Hypothesis is that $\mathcal{F}(X)$ is easier to optimize than $\mathcal{H}(X)$. To understand, let us take an example. Assume that the ideal $\mathcal{H}(X)$ equals X. Following is a stack of non-linear layers for a direct mapping, making learning an identity mapping challenging.

$$X \to W_1 \to ReLU \to W_2 \to ReLU \dots X$$

5

Therefore, it would be challenging to approximate the identity mapping with all of these weights $\{W_1, W_2, \dots\}$ and ReLUs in the middle.

Now, if we define the ideal mapping $\mathcal{H}(X) = \mathcal{F}(X) + X$, then we just need to get $\mathcal{F}(X) = 0$ as follows:

$$X \rightarrow W_1 \rightarrow ReLU \rightarrow W_2 \rightarrow ReLU \ldots 0$$

It is easy to achieve the above. To get zero output, just set any weight to zero, and then add X to get mapping $\mathcal{H}(X)$.

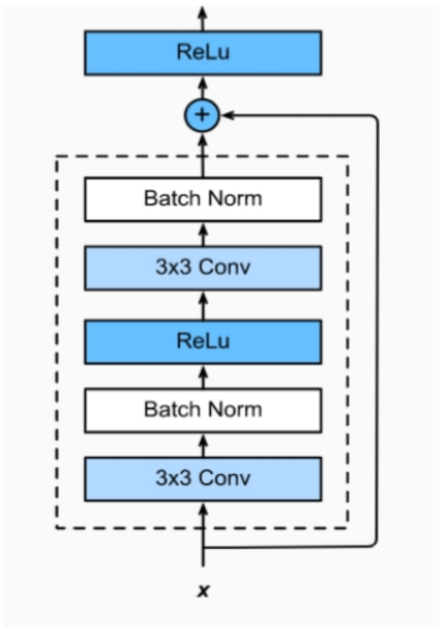


Figure 3.2: Identity Block

3.2.1 Identity Block

It is a residual block, which is used when input and output dimensions are the same i.e. when the input X and output $\mathcal{F}(X)$ are of the same dimensions, the identity shortcut can be directly used.

$$\mathcal{H}(X) = \mathcal{F}(X, W_i) + X$$

3.2.2 Convolutional Block

It is a residual block, which is used when input and output dimensions are different i.e. one convolution layer of kernel size (1,1) used with X to match the dimension of of $\mathcal{F}(X)$.

$$\mathcal{H}(X) = \mathcal{F}(X, W_i) + W_s X$$

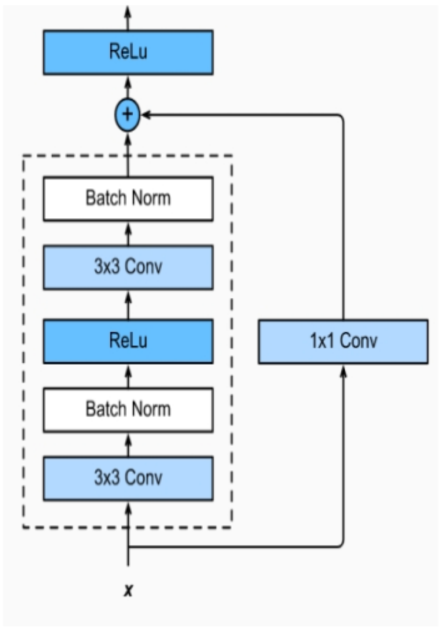


Figure 3.3: Convolutional Block

Chapter 4

Experiments and Results

In this chapter, we evaluate performance of the proposed WAE ResNet model. We would like to test if WAE ResNet can simultaneously achieve (i) reasonable geometry of the latent manifold, (ii) random samples of good visual quality, and (iii) accurate reconstructions of data points. We have trained VAE, VAE ResNet, WAE and WAE ResNet on real data-set, called "CelebA" which has 202599 images.

4.1 Hardware Requirement

All the experiments have been performed on Indian Statistical Institute Kolkata server which has the following hardware

Table 4.1: Hardware Requirement

	Model Name	**Memory**
CPU	Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz	197568076 kB (197.56 GB)
GPU	Tesla P100-PCIE	16384 MB

4.2 Data-sets

We have used **CelebFaces Attributes Dataset (CelebA)** [9]. It is a large-scale face attributes data-sets. There are **202599** images in this data-sets. Statistics of data-sets are given in Table 4.1.

Table 4.2: Data-set Statistics

Data-set	Number of Images	Training Images	Test Images
CelebA	202599	162079	40520

4.3 Baseline and Experimental setup

For baseline we have used Variational Auto-encoder architecture and Wasserstein Auto-encoder architecture.

4.3.1 Variational Auto-encoder Experimental setup

In VAEs, we have used Euclidian latent space $\mathcal{Z} = \mathcal{R}^{d_z}$ for various d_z depending on the complexity of the data-set and standard Gaussian prior $P_Z = \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$. We have used Gaussian Encoder $Q(Z|X) = \mathcal{N}(Z; \mu_\theta(X), \Sigma(X))$ with mean μ_θ and diagonal covariance Σ and Gaussian decoder $P_G(X|Z) = \mathcal{N}(X; G_\phi(Z), \sigma_G^2.\mathbf{I}_d)$. Functions μ_θ, Σ and G_ϕ are parameterized by the below neural networks.

Encoder:

- **Layer 1:** The 2D Convolution layer has 32 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization[4].
- **Layer 2:** The 2D Convolution layer has 64 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.
- **Layer 3:** The 2D Convolution layer has 128 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.
- **Layer 4:** The 2D Convolution layer has 256 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.

- **Layer 5:** The 2D Convolution layer has 512 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization .
- **Flatten:** Flatten to make 1D tensor.
- **Dense:** Dense layer is applied with units=2048 and activation='selu'

Decoder:

- **Layer 1:** Dense layer has output units=2048 with activation='selu and BatchNormalization.
- **Layer 2:** Dense layer has output units=8192 with activation='selu and then reshape with dimension [4, 4, 512]
- **Layer 3:** The 2D Transpose Convolution layer has 256 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.
- **Layer 4:** The 2D Transpose Convolution layer has 128 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.
- **Layer 5:** The 2D Transpose Convolution layer has 64 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.
- **Layer 6:** The 2D Transpose Convolution layer has 32 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.
- **Layer 7:** The 2D Transpose Convolution layer has 16 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.
- **Layer 8:** The 2D Transpose Convolution layer has 3 filters of kernel size (3,3) with a stride of (1,1) and padding='same'. Sigmoid used as activation function and BatchNormalization.

Latent space dimension: 512

Optimizer: Adam[5] with **learning rate** = 0.001 and **exponential decay rate** = 0.5

Epochs: 100

4.3.2 Wasserstein Auto-encoder Experimental setup

In WAEs, depending on the complexity of the data set, we employed Euclidian latent space $\mathcal{Z} = \mathcal{R}^{d_z}$ for various d_z, isotropic Gaussian prior $P_Z = \mathcal{N}(Z; \mathbf{0}, \sigma_z^2.\mathbf{I}_d)$ over $\mathcal{Z}$ and a squared cost function $c(x, y) = \|x - y\|_2^2$ for data-points $x, y \in \mathcal{X} = \mathcal{R}^{d_x}$. We have used neural network architecture for encoder mapping $\mu_\theta : \mathcal{X} \to \mathcal{Z}$ and decoder mapping $G_\phi : \mathcal{Z} \to \mathcal{X}$ with batch normalization. We have used discriminator of fully connected neural networks with ReLU.

Encoder:

- **Layer 1:** The 2D Convolution layer has 128 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and ReLU as activation function.
- **Layer 2:** The 2D Convolution layer has 256 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and ReLU as activation function.
- **Layer 3:** The 2D Convolution layer has 512 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and ReLU as activation function.
- **Layer 4:** The 2D Convolution layer has 1024 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and ReLU as activation function.
- **Layer 5:** The 2D Convolution layer has 2048 filters of kernel size (5,5) with a stride of (2,2)vand padding='same'. Use BatchNormalization and ReLU as

activation function.

- **Flatten:** Flatten to make 1D tensor.
- **Dense:** Dense layer is applied with units=64.

Decoder:

- **Layer 1:** Dense layer has output units=4*4*2048 with activation='ReLU' and BatchNormalization and then reshape with dimension [4, 4, 2048].
- **Layer 2:** The 2D Transpose Convolution layer has 1024 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and activation='ReLU'.
- **Layer 3:** The 2D Transpose Convolution layer has 512 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and activation='ReLU'.
- **Layer 4:** The 2D Transpose Convolution layer has 256 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and activation='ReLU'.
- **Layer 5:** The 2D Transpose Convolution layer has 128 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and activation='ReLU'.
- **Layer 6:** The 2D Transpose Convolution layer has 3 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and activation='sigmoid'.

Discriminator:

- **Layer 1:** Dense layer is applied with units=512 and activation='relu'.
- **Layer 2:** Dense layer is applied with units=512 and activation='relu'.

- **Layer 3:** Dense layer is applied with units=512 and activation='relu'.
- **Layer 4:** Dense layer is applied with units=512 and activation='relu'.
- **Layer 5:** Dense layer is applied with units=1 and activation='sigmoid'.

Latent space dimension: 64

Auto-encoder Optimizer: Adam with **learning rate** $= 3e^{-4}$ and **exponential decay rate** $=$ 0.5

Discriminator Optimizer: Adam with **learning rate** $= 1e^{-3}$ and **exponential decay rate** $=$ 0.5

Gaussian Distribution's standard deviation: 1.414

Epochs: 100

4.3.3 Variational Auto-encoder with ResNet

In VAE ResNet, we have used Euclidian latent space $\mathcal{Z} = \mathcal{R}^{d_z}$ for various d_z depending on the complexity of the data-set and standard Gaussian prior $P_Z = \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$. We have used Gaussian Encoder $Q(Z|X) = \mathcal{N}(Z; \mu_\theta(X), \Sigma(X))$ with mean μ_θ and diagonal covariance Σ and Gaussian decoder $P_G(X|Z) = \mathcal{N}(X; G_\phi(Z), \sigma_G^2.\mathbf{I}_d)$. Functions μ_θ, Σ and G_ϕ are parameterized by below neural networks.

Encoder:

- **Zero-padding:** pads the input with a pad of (2,2)
- **Stage 1:** The 2D Convolution layer has 64 filters of kernel size (7,7) with a stride of (2,2). BatchNormalization is applied to the channels axis=3 of the input. MaxPooling uses a (3,3) window with a (1,1) stride.

- **Stage 2:** The convolutional block uses three sets of filters [64, 64, 256] of kernel size f=3 with stride s=1. The two identity blocks use 3 sets of filters [64, 64, 256] of kernel size f=3.
- **Stage 3:** The convolutional block uses three sets of filters [128, 128, 512] of kernel size f=3 with stride s=2. The three identity blocks use 3 sets of filters [128, 128, 512] of kernel size f=3.
- **Stage 4:** The convolutional block uses three sets of filters [256, 256, 1024] of kernel size f=3 with stride s=2. The three identity blocks use 3 sets of filters [256, 256, 1024] of kernel size f=3.
- **Stage 5:** The convolutional block uses three sets of filters [512, 512, 2048] of kernel size f=3 with stride s=2. The three identity blocks use 3 sets of filters [512, 512, 2048] of kernel size f=3.
- **Average Pooling:** The 2D AveragePooling uses a window of shape (2,2).
- **Flatten:** Flatten layer is used to make 1D tensor.
- **Dnese:** The Fully Connected Dense layer use uints=64.

Decoder:

- **Layer 1:** Dense layer has output units=2048 with activation='selu and BatchNormalization.
- **Layer 2:** Dense layer has output units=8192 with activation='selu and then reshape with dimension [4, 4, 512]
- **Layer 3:** The 2D Transpose Convolution layer has 256 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.
- **Layer 4:** The 2D Transpose Convolution layer has 128 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.

- **Layer 5:** The 2D Transpose Convolution layer has 64 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation funtion with value (0.02) and BatchNormalization.

- **Layer 6:** The 2D Transpose Convolution layer has 32 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.

- **Layer 7:** The 2D Transpose Convolution layer has 16 filters of kernel size (3,3) with a stride of (2,2) and padding='same'. LeakyReLU used as activation function with value (0.02) and BatchNormalization.

- **Layer 8:** The 2D Transpose Convolution layer has 3 filters of kernel size (3,3) with a stride of (1,1) and padding='same'. Sigmoid used as activation function and BatchNormalization.

Latent space dimension: 512

Optimizer: Adam with **lerning rate** = 0.001 and **exponential decay rate** = 0.5

Epochs: 100

4.3.4 Wasserstein Auto-encoder with ResNet

In WAEs, depending on the complexity of the data set, we employed Euclidian latent space $\mathcal{Z} = \mathcal{R}^{d_z}$ for various d_z, isotropic Gaussian prior $P_Z = \mathcal{N}(Z; \mathbf{0}, \sigma_z^2.\mathbf{I}_d)$ over $\mathcal{Z}$ and a squared cost function $c(x, y) = \|x - y\|_2^2$ for data-points $x, y \in \mathcal{X} = \mathcal{R}^{d_x}$. We have used neural network architecture for encoder mapping $\mu_\theta : \mathcal{X} \to \mathcal{Z}$ and decoder mapping $G_\phi : \mathcal{Z} \to \mathcal{X}$ with batch normalization. We have used discriminator of fully connected neural networks with ReLU.

Encoder:

- **Zero-padding:** pads the input with a pad of (2,2)

- **Stage 1:** The 2D Convolution layer has 64 filters of kernel size (7,7) with a stride of (2,2). BatchNormalization is applied to the channels axis=3 of the input. MaxPooling uses a (3,3) window with a (1,1) stride.
- **Stage 2:** The convolutional block uses three sets of filters [64, 64, 256] of kernel size f=3 with stride s=1. The two identity blocks use 3 sets of filters [64, 64, 256] of kernel size f=3.
- **Stage 3:** The convolutional block uses three sets of filters [128, 128, 512] of kernel size f=3 with stride s=2. The three identity blocks use 3 sets of filters [128, 128, 512] of kernel size f=3.
- **Stage 4:** The convolutional block uses three sets of filters [256, 256, 1024] of kernel size f=3 with stride s=2. The three identity blocks use 3 sets of filters [256, 256, 1024] of kernel size f=3.
- **Stage 5:** The convolutional block uses three sets of filters [512, 512, 2048] of kernel size f=3 with stride s=2. The three identity blocks use 3 sets of filters [512, 512, 2048] of kernel size f=3.
- **Average Pooling:** The 2D AveragePooling uses a window of shape (2,2).
- **Flatten:** Flatten layer is used to make 1D tensor.
- **Dnese:** The Fully Connected Dense layer use uints=64.

Decoder:

- **Layer 1:** Dense layer has output units=4*4*2048 with activation='ReLU' and BatchNormalization and then reshape with dimension [4, 4, 2048].
- **Layer 2:** The 2D Transpose Convolution layer has 1024 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and activation='ReLU'.
- **Layer 3:** The 2D Transpose Convolution layer has 512 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and activation='ReLU'.

- **Layer 4:** The 2D Transpose Convolution layer has 256 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and activation='ReLU'.
- **Layer 5:** The 2D Transpose Convolution layer has 128 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and activation='ReLU'.
- **Layer 6:** The 2D Transpose Convolution layer has 3 filters of kernel size (5,5) with a stride of (2,2) and padding='same'. Use BatchNormalization and activation='sigmoid'.

Discriminator:

- **Layer 1:** Dense layer is applied with units=512 and activation='relu'.
- **Layer 2:** Dense layer is applied with units=512 and activation='relu'.
- **Layer 3:** Dense layer is applied with units=512 and activation='relu'.
- **Layer 4:** Dense layer is applied with units=512 and activation='relu'.
- **Layer 5:** Dense layer is applied with units=1 and activation='sigmoid'.

Latent space dimension: 64

Autoencoder Optimizer: Adam with **learning rate** $= 3e^{-4}$ and **exponential decay rate** $=$ 0.5

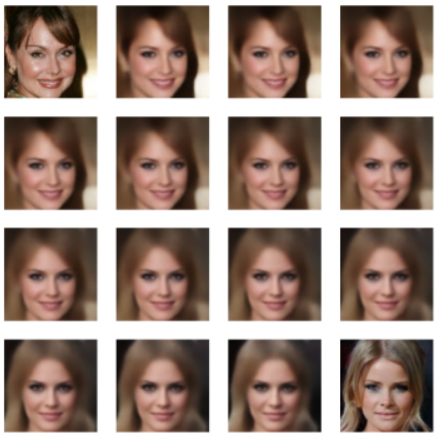
Discriminator Optimizer: Adam with **learning rate** $= 1e^{-3}$ and **exponential decay rate** $=$ 0.5

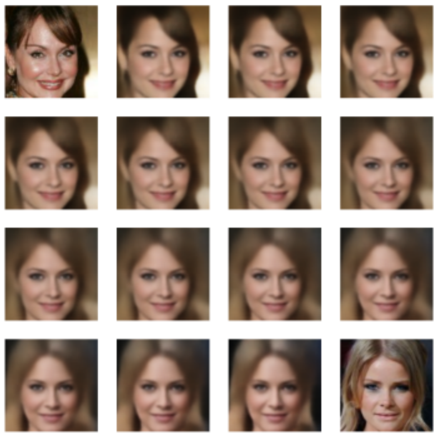
Gaussian Distribution's standard deviation: 1.414

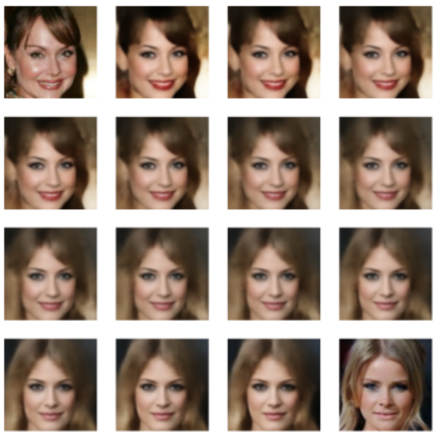
Epochs: 100

4.4 Results

4.4.1 Interpolated Images


(a) Interpolated images of VAE


(b) Interpolated images of VAE ResNet


(c) Interpolated images of WAE

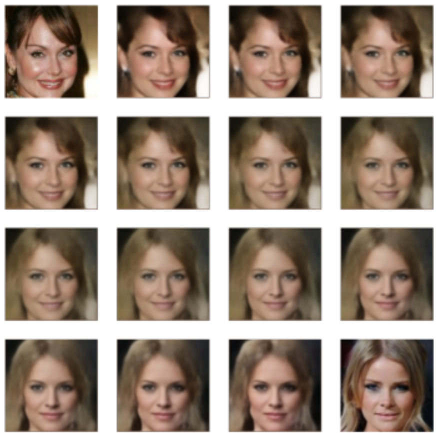

(d) Interpolated images of WAE ResNet

Figure 4.1: Interpolated images

Interpolated images of VAE, VAE-ResNet, WAE and WAE-ResNet are given in Figure 4.1. For each model in Figure 4.1, we selected the first and final two original images, while the rest images are created interpolated images. Interpolated images i.e. how one image is transforming to the other image. Sharpness score of the interpolated images are given in Table 4.3. WAE-ResNet interpolated images are better than other models.

Table 4.3: Sharpness (larger is better) scores of Interpolated Images

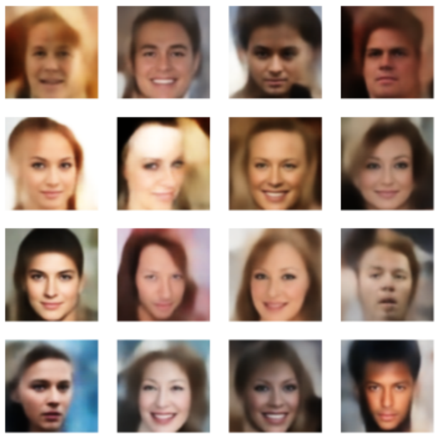
Models	**Sharpness score**
VAE	6.541
VAE ResNet	6.580
WAE	6.565
WAE ResNet	**7.952**

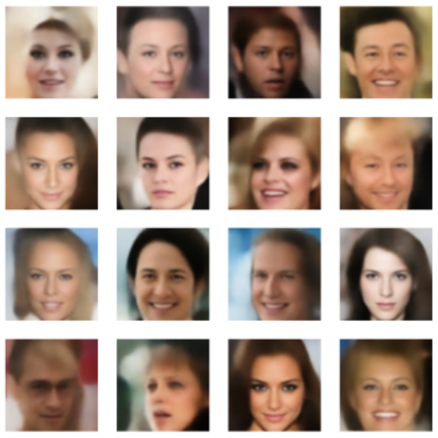
4.4.2 Random Images

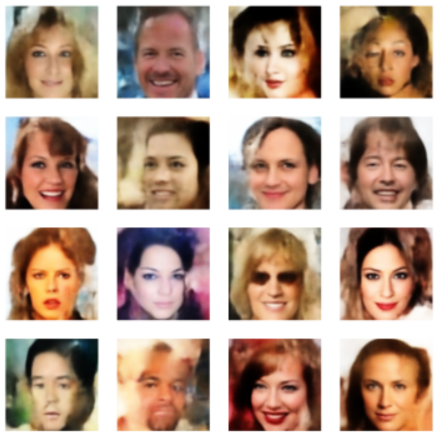
Randomly generated images are given in Figure 4.2. We randomly select the value from Gaussian distribution of size 64 and then decoder takes it as input and then produce the image of size 128×128. FID and Sharpness scores of 10000 randomly generated images are given in Table 4.4. WAE-ResNet FID and Sharpness scores are better compare to other models.

Table 4.4: FID (smaller is better) and Sharpness (larger is better) scores of 10000 Random Images

Models	**FID score**	**Sharpness score**
VAE	38.439	6.233
VAE ResNet	34.942	5.985
WAE	5.756	7.848
WAE ResNet	**5.693**	**10.434**


(a) Random images of VAE


(b) Random images of VAE ResNet


(c) Random images of WAE

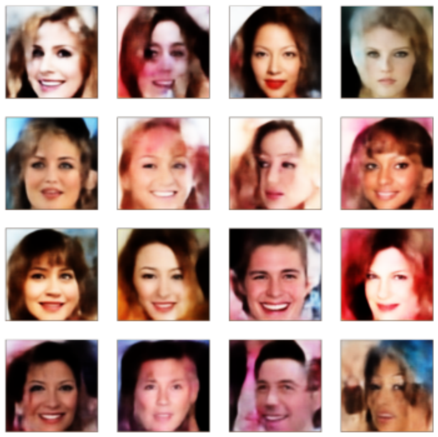

(d) Random images of WAE ResNet

Figure 4.2: Random images

4.4.3 Reconstructed Images

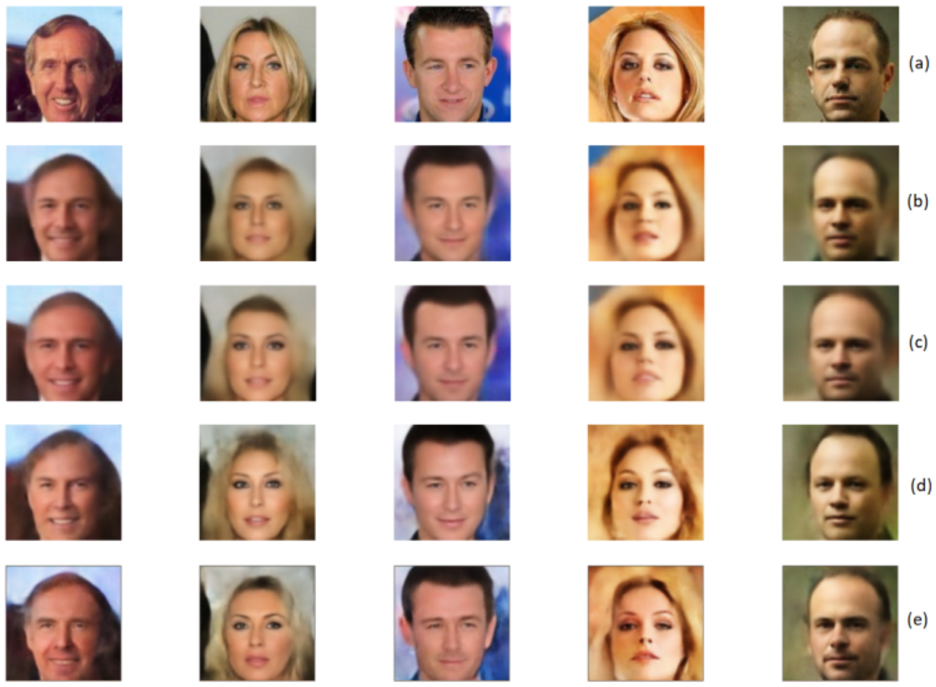


Figure 4.3: (a) Original Images, (b) VAEs Reconstructed Images, (c) VAE-ResNet Reconstructed Images, (d) WAEs Resconstructed Images and (e) WAE-ResNet Reconstructed Images

Table 4.5: FID (smaller is better) and Sharpness (larger is better) scores of 40000 Reconstructed Images

Models	FID score	Sharpness score
VAE	13.008	6.013
VAE ResNet	10.901	7.607
WAE	5.449	7.187
WAE ResNet	**4.155**	**9.172**

Reconstructed images are given in Figure 4.3, and WAE-ResNet gives better results compared to VAE, VAE-ResNet, and WAE. To compare the quality of original image samples and generated images samples, we use FID (Fréchet Inception Distance) [3] and report the results on CelebA dataset based on 40000 test samples. We also evaluate the sharpness of generated image samples using the gradient square method.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this work, we have developed different architecture of Encoder for WAEs using ResNet, which uses the identity mapping, i.e. we have added the output of previous layer directly to output of next three layers. This has resulted in better feature preservation at latent space.

We have used ResNet architecture with VAEs and WAEs to improve the performance of base line VAEs and WAEs. The FID scores that we have got from WAE-ResNet proves that gives better result for the generative modeling task, i.e. interpolation of images, reconstruction of test images and also random generartion. In addition, average sharpness of the generated images are better than generated images obtained by the VAEs, VAE ResNet and WAE models. Interpolated images of WAE-ResNet also looks sharper compared to the others.

5.2 Future Work

After implementing ResNet architecture to encoder, we have observed that the generative capacity of the model improved significantly. Therefore, we conclude that the generative capacity of the model is strongly dependent on the type of architecture used for encoder/decoder. Future works could be in the direction of development of novel architectures for encoder and decoder.

Bibliography

[1] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks (2014), https://arxiv.org/abs/1406.2661

[2] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015), https://arxiv.org/abs/1512.03385

[3] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium (2017), https://arxiv.org/abs/1706.08500

[4] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015)

[5] Kingma, D.P., Lei, J.: Adam: A method for stochastic optimization (2014)

[6] Kingma, D.P., Welling, M.: Auto-encoding variational bayes (2013), https://arxiv.org/abs/1312.6114

[7] Tolstikhin, I., Bousquet, O., Gelly, S., Schoelkopf, B.: Wasserstein auto-encoders (2017), https://arxiv.org/abs/1711.01558

[8] Villani, C.: Topics in optimal transportation (2003)

[9] Ziwei Liu, Ping Luo, X.W., Tang, X.: Deep learning face attributes in the wild (2005)

Wasserstein Auto-Encoder using Residual Neural Network

ORIGINALITY REPORT

15%
SIMILARITY INDEX

PRIMARY SOURCES

1	arxiv.org Internet	398 words — 5%
2	library.isical.ac.in:8080 Internet	170 words — 2%
3	github.com Internet	132 words — 2%
4	jaan.io Internet	110 words — 1%
5	stackoverflow.com Internet	60 words — 1%
6	Xingyu Chen, Jin Li, Xuguang Lan, Nanning Zheng. "Generalized Zero-shot Learning via Multi-modal Aggregated Posterior Aligning Neural Network", IEEE Transactions on Multimedia, 2020 Crossref	42 words — 1%
7	escholarship.org Internet	38 words — 1%
8	tojqi.net Internet	24 words — < 1%
9	researchsystem.canberra.edu.au Internet	

19 words — < 1%

10	Long Xu, Ying Wei, Chenhe Dong, Chuaqiao Xu, Zhaofu Diao. "Wasserstein Distance-Based Auto-Encoder Tracking", Neural Processing Letters, 2021 Crossref	18 words — < 1%
11	core.ac.uk Internet	18 words — < 1%
12	pt.slideshare.net Internet	17 words — < 1%
13	d.lib.msu.edu Internet	16 words — < 1%
14	Li, Yao. "On Robustness and Efficiency of Machine Learning Systems.", University of California, Davis, 2020 ProQuest	15 words — < 1%
15	machinelearningmastery.com Internet	14 words — < 1%
16	openreview.net Internet	14 words — < 1%

EXCLUDE QUOTES	ON	EXCLUDE SOURCES	< 14 WORDS
EXCLUDE BIBLIOGRAPHY	ON	EXCLUDE MATCHES	< 14 WORDS