Adaptive Augmentation in Latent Space using Autoencoders

Nikhila Ramisetti

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master of Science in Artificial Intelligence
of the
University of Aberdeen.

Department of Computing Science

2024

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed: Nikhila Ramisetti

Date: 2024

Abstract

This project focuses on the challenge of enhancing image generation through the integration of adaptive augmentations within the latent space of Autoencoders (AEs). While the scarcity of diverse data often hinders model performance and scalability, data augmentation techniques offer a promising solution by generating diverse training examples from limited datasets. However, conventional augmentation methods may not fully meet the complex requirements of image generation tasks.

Recent progress in latent space enhancements has inspired this study to explore the possibility of utilizing acquired feature representations for conducting transformations that can capture essential data attributes while staying unaffected by irrelevant variations. Through implementing augmentations within the latent space of autoencoders, the objective of the framework is to enhance model robustness, generalization, and computational efficiency.

Key objectives include investigating existing Autoencoder architectures and latent space techniques, implementing a prototype framework incorporating adaptive augmentations, experimenting with augmentation strategies, optimizing the framework for efficiency and scalability, and validating its performance through quantitative metrics and qualitative assessments.

The project utilizes a dataset containing 200,000 images from the celebA dataset, with computational constraints limiting the analysis to 70,000 images. Training is conducted over 500 epochs, resulting in the generation of 128x128 images with a Fréchet Inception Distance (FID) score of 34.5. Through this work, we aim to contribute to the advancement of adaptive latent space augmentation techniques in AI image generation, paving the way for more robust, generalizable, and interpretable image generation models.

Acknowledgements

I would like to express my gratitude to those who have supported and guided me throughout this project. Specifically, I want to acknowledge:

- My Academic Supervisor, Dr. Yin Lu, for his valuable insights and suggestions.
- My University friends, whose camaraderie and shared efforts made this journey memorable.

Their contributions have been instrumental in the successful completion of this project.

Contents

Chapter 1

Introduction

1.1 Motivation

The advent of artificial intelligence has been marked by significant progress in machine learning models, leading to revolutionary transformations in different domains. A significant challenge lies in the scarcity of diverse datasets, particularly for tasks such as image generation. This limitation often constrains the performance and scalability of models, hindering their ability to produce high-quality and diverse outputs. Traditional approaches to data augmentation, which are commonly employed in classification tasks, do not directly address the unique requirements of generative tasks (1).

However, the conventional approaches to data augmentation (2), although effective in many scenarios, may not fully meet the complex requirements of image generation tasks. Traditional methods apply transformations uniformly to entire images, including the relevant and irrelevant nuances present in the data. In recent years, there has been a growing recognition of the potential for data augmentation techniques to enhance the training of generative models (3). By synthesizing variations of existing data points and introducing novel examples, data augmentation holds promise in revitalizing learning algorithms for generative tasks, enabling the creation of more diverse and realistic outputs. Recent research has revealed a significant alternative: Augmentations in the latent space as shown in figure 1.1. These augmentations operate at a more abstract level, utilizing learned feature representations to shape data transformations that go beyond simple pixel manipulations (4). The latent space emerges with immense potential, enabling models to navigate with accuracy and create images that possess subtlety and depth.

Baseline investigations have underscored the superiority of latent space augmentations over their image-based counterparts, revealing three key benefits: Robustness, generalization, and computational efficiency (5). Integrating these augmentations during training process not only prevents over fitting but also expedites convergence and strengthens model regularization as shown in figure 1.2. Tailored enhancements, adjusted to align with the changing training environment, guarantee a smooth shift from learning to inference, promoting consistency and coherence in model performance. Furthermore, the utilization of latent space augmentations allows for the exploration of richer and more nuanced representations of the data manifold, facilitating the generation of novel and compelling images with greater fidelity and realism.

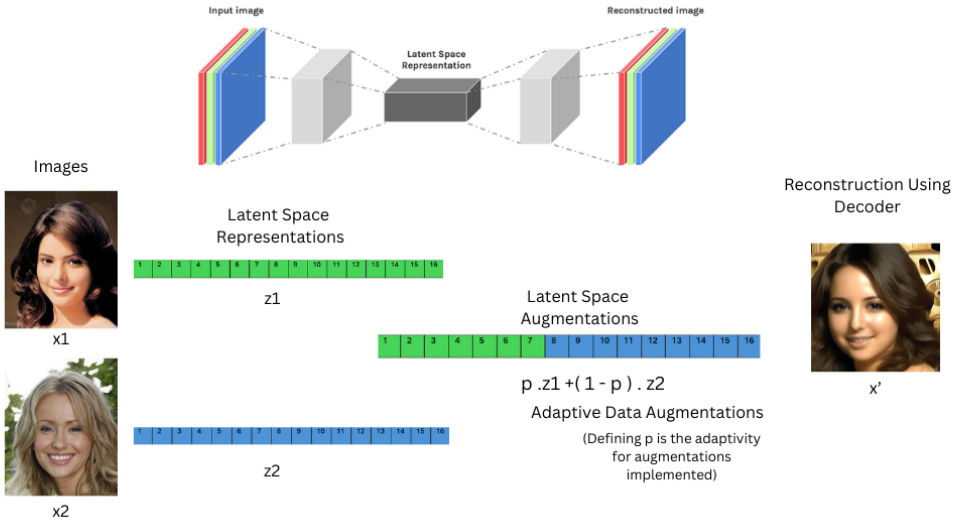

Figure 1.1: Latent Space Augmentations Objective

Leveraging adaptive data augmentation within the latent space of Auto encoders presents an opportunity to enhance the robustness, generalization and computational efficiency, By performing augmentations on learned feature representations, models can capture essential characteristics of the data while being invariant to variations irrelevant to the task. This not only improves model robustness and generalization but also allows for more interpret-able and controllable image generation.

1.2 Objectives

The primary objective of this project is to develop a composed framework using Auto encoders (AEs) to integrate adaptive augmentations within the latent space for AI image generation. Specific objectives include:

- Investigating existing Auto encoder architectures and latent space techniques: A thorough review of VAE architectures and latent space manipulation methodologies prevalent in literature is conducted. These methodologies are scrutinized for latent space manipulation within VAEs to ascertain their implications.
- Implementing a prototype framework incorporating adaptive augmentations: A prototype framework integrating VAEs with adaptive augmentation methodologies within latent space is designed and instantiated. The algorithms for dynamic application of augmentations during training, informed by real-time adjustments based on validation loss metrics are engineered.
- Experimenting with adaptive augmentation strategies and adjusting parameters based on data characteristics. The different augmentation strategies with VAE paradigm including cutout, mixup, and gaussian augmentation are empirically evaluated. The impact of dynamically adjusting augmentation parameters in response to validation loss dynamics on model performance is investigated.

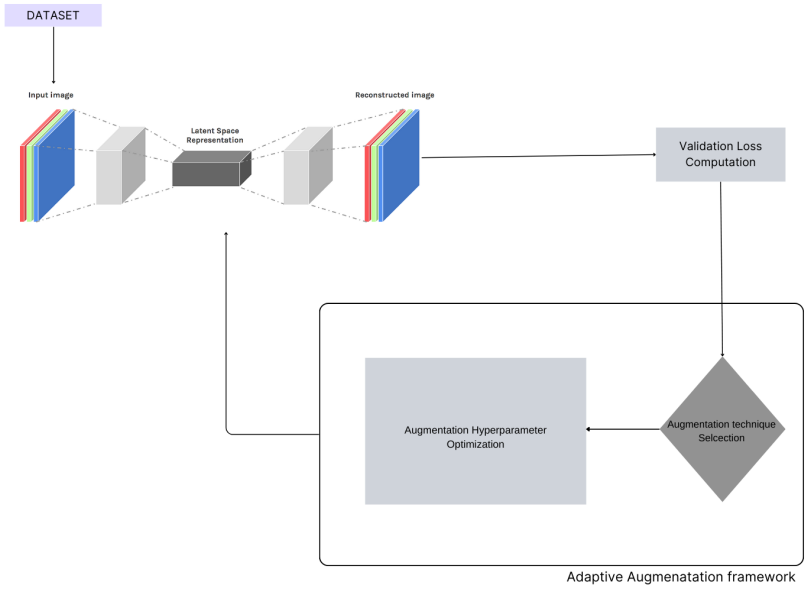

Figure 1.2: Incorporating Adaptivity to Latent Space Augmentation

- Optimizing the framework for efficiency and scalability. The MSE scores, Perceptual metrics are observed and model is optimized. Additionally, the computational efficiency of the VAE-augmentation framework usign parallelization and hardware accelaration is optimized. A scalable framework accommodating voluminous datasets and intricate augmentation pipelines without compromising computational efficacy is engineered.

- Validating the framework's performance through quantitative metrics and qualitative assessments The evaluation metrics encompassing quantitative metrics (e.g., MSE, IS, FID) and qualitative assessments (e.g., visual inspection) is developed to validate the robustness, generalization capacity, and image generation prowess of the framework across diverse datasets and tasks.

1.3 Outline(Thesis Structure)

The structure of this thesis reflects the order of the research process we followed in order to generate images using the framework proposed in the objective.

- Chapter 1 provides an overview of the project, including motivation, objectives, and outline.

- Chapter 2 exhibits the underlying research process which involves literature review on Auto encoders, adaptive data augmentation, and image generation.

- Chapter 3 refers to the details of the dataset description, processing pipeline, model architecture, training methodology, and timeline.

- Chapter 4 characterises the process of the actual development of the project, implementation of the proposed framework to testify the model's practicability.

- Chapter 5 describes how we monitored and evaluated the models. Additionally, it provides the findings, conclusions, and directions from a particular evaluation, including recommendations for how evaluation results were used to guide the model improvement(e.g. accuracy, precision) and the decision that was made.
- Chapter 6 summarizes the report as a whole, drawing inferences from the entire process of the project about the discoveries, or decision(Limitations, Future Work).

Chapter 2

Literature Review

2.1 Variational Autoencoders (VAEs)

Variational Autoencoders(VAEs) are an important development in the field of unsupervised learning and generative modelling. VAEs represent a specific class of neural network structures that incorporates probabilistic latent variable models with autoencoder components. They were first presented by Kingma and Welling in 2013 (6), and they completely changed the area of deep learning by providing a framework for unsupervised learning of hidden representations of data. VAEs are fundamentally made up of two parts: an encoder and a decoder. In the latent space, the encoder network translates the input data to a probabilistic distribution, usually Gaussian distribution. The latent variables or characteristics that capture the fundamental structure of the input data are represented by this distribution. After that, the decoder network reconstructs the original input data using samples from this latent space.

The capacity of VAEs to create fresh data samples by sampling from the learnt latent space in one of its primary characteristics. Through random selection of points from the latent space and their passage through the decoder network, VAEs can produce new data that has a striking resemblance to the distribution of training data. because of this, VAEs are strong generative models that can produce fresh text, pictures, or other kinds of data. The VAE model is trained using variational inference techniques, which is another significant feature of VAEs. The goal of VAE training is to maximise the evidence lower bound(ELBO), or a lower constraint on the data's log-likelihood. This entails optimising the KL divergence between the inferred latent distribution and the prior distribution as well as the reconstruction loss, which gauges how effectively the decoder can reconstruct the input data. VAEs may provide realistic data samples and develop meaningful latent representations by simultaneously optimizing these objectives.

Numerous research works have examined various aspects of Variational Autoencoders(VAEs), including training approaches, innovative architectures, and real-world uses (7).

2.1.1 Beta-VAE: Controlling Latent Disentanglement

The beta-VAE framework, introduced by Higgins, represents a significant advancement in Variational Autoencoder (VAE) technology (8). By incorporating a hyper parameter to regulate the disentanglement of latent representations, this framework enables VAEs

to learn more discernible and interpretable latent characteristics. This control over the disentanglement process enhances the model's ability to comprehend complex data structures and facilitates more effective manipulation of the underlying data. Through this innovative approach, researchers can gain deeper insights into the latent space and extract meaningful features, leading to improved performance in various machine learning tasks.

The beta-VAE framework introduces a novel hyperparameter known as beta, which plays a key role in controlling the disentanglement of latent representations. Disentanglement refers to the degree to which different dimensions of the latent space capture distinct and meaningful factors of variation within the data. Through the manipulation of the beta hyperparameter, researchers are able to prompt the VAE to acquire more separable and interpretable latent factors. This ability to regulate disentanglement is essential as it allows VAEs to better capture intricate data structures. In conventional VAEs, the latent space may be entangled, resulting in dimensions that are not easily interpretable and may contain redundant or overlapping information. By adjusting disentanglement using the beta parameter, the beta-VAE framework provides a more precise control over the process of representation learning. The capacity to learn disentangled representations has significant implications across various machine learning tasks. For instance, in image generation, disentangled representations can enhance the manipulation of attributes in generated images, such as shape, color, and orientation. In reinforcement learning, disentangled representations can aid in distinguishing between relevant and irrelevant features of the environment, thereby improving the efficiency of learning and decision-making.

2.1.2 Conditional VAEs: Tailored Output Generation

In contrast, Zhao introduced conditional VAEs, which are specifically designed to generate structured outputs conditioned on particular labels or qualities (9). This pioneering development significantly enhances the utility of VAEs in diverse image generation tasks. By allowing the model to generate images with specified attributes such as style, appearance, or content, conditional VAEs empower researchers and practitioners to achieve more targeted and customized outputs. This feature creates opportunities for the development of new applications in fields like computer vision, image editing, and content creation, where precise control over generated outputs is crucial for achieving desired outcomes. Unlike traditional Variational Autoencoders (VAEs) that rely solely on randomly sampled latent variables to generate images, Conditional Variational Autoencoders (CVAEs) introduce an additional input: specific labels or attributes acting as conditioning factors.

CVAEs stand out due to their capability to create images that correspond to the provided labels or attributes. Instead of producing random images, CVAEs can generate images with distinct characteristics based on the conditioning factors. For instance, if the input includes attributes like style, appearance, or content, the CVAE can generate images that conform to these specifications. The impact of this advancement is profound and transformative. In the realm of computer vision, CVAEs can be utilized to generate images of objects with particular features or in specific conditions, aiding tasks such as object recognition and scene comprehension. Within image editing software, users can leverage CVAEs to manipulate images based on predefined attributes, enabling complex operations

like style transfer, image synthesis, and content morphing. Furthermore, CVAEs open up new avenues in content creation sectors such as graphic design and advertising. They empower designers and marketers to swiftly produce customized visuals tailored to precise requirements and preferences. Whether it involves crafting personalized product images, generating unique artistic designs, or developing targeted advertisements, CVAEs offer the flexibility and accuracy necessary to address diverse creative needs.

CVAEs present an opportunity for innovation in industries like fashion, architecture, and interior design. Designers can utilize CVAEs to explore and experiment with various design possibilities by generating images embodying different styles, aesthetics, and configurations. This not only expedites the design process but also fosters creativity and innovation. In essence, CVAEs signify a shift in generative modeling, providing a robust framework for generating structured outputs.

2.2 Complex Augmentation Techniques

Data Augmentation techniques are indeed invaluable for enhancing the performance and robustness of machine learning models, especially in image processing tasks. They enable models to learn from a more diverse set of data, leading to improved generalization and resistance to over-fitting. While traditional augmentation methods like rotation, flipping, and scaling are effective to some extent, more complex techniques such as Cutout, Mix up, and Gaussian Sampling offer additional benefits. During the training process, the technique of Cutout is implemented by randomly concealing rectangular portions of input images. This method compels the model to concentrate on different regions of the image, prompting it to acquire resilient features that remain unaffected by such obstructions. In contrast, Mixup combines pairs of images and their respective labels while training, generating novel training instances that fall along the interpolation trajectory between the original samples. This approach not only enhances the diversity of the training dataset but also regularizes the model by promoting the acquisition of linear combinations of features. Gaussian Sampling, on the other hand, entails introducing random noise sourced from a Gaussian distribution to the input images. This action mimics variations in lighting conditions, blurriness, or other elements that may arise in real-world situations.

The rationale for utilizing these intricate augmentation methods lies in their capacity to incorporate more authentic and varied modifications into the training dataset. By subjecting the model to a broader spectrum of situations, these methods empower it to acquire more resilient and adaptable representations. Additionally, they aid in reducing the likelihood of overfitting by imposing constraints on the model and deterring it from excessively depending on particular features or patterns found in the training data. Essentially, conventional augmentation methods establish the groundwork for introducing diversity into the training data. However, more intricate approaches such as Cutout, Mixup, and Gaussian Sampling provide further advantages by introducing sophisticated variations that improve the performance and resilience of the model in real-world scenarios. By effectively utilizing these techniques, both researchers and practitioners can create machine learning models that are better prepared to tackle the intricacies and difficulties

posed by diverse datasets.

2.2.1 Cutout: Enhancing Model Robustness Through Random Masking

DeVries and Taylor introduced Cutout, a data augmentation technique that involves randomly masking out regions of input images during training (4). By pushing the network to learn characteristics from different regions of the picture instead of depending just on localised information, this method seeks to improve the resilience of the model. Cutout successfully regularises the model and enhances its capacity to generalise to unknown data by routinely occluding sections of input pictures. The efficacy of this strategy in improving model performance is demonstrated by its broad acceptance in tasks such as semantic segmentation, object identification and picture classification.

Cutout is highly valued for its simplicity and efficiency. By introducing random occlusions to the training data, Cutout prompts the model to acquire more resilient and invariant features, ultimately resulting in enhanced performance on unseen data. Moreover, Cutout functions as a method of data augmentation, effectively broadening the diversity of the training set without the need for additional labeled data. The effectiveness of Cutout has been extensively proven in various computer vision tasks, such as semantic segmentation, object detection, and image classification. For instance, in semantic segmentation, Cutout aids the model in better comprehending object boundaries and spatial relationships by obscuring different sections of the input images. Similarly, in tasks like object detection and image classification, Cutout guides the model to concentrate on pertinent features while disregarding distracting details, thereby producing more precise and resilient predictions.

2.2.2 Mixup: Regularizing Models Through Data Mixing

Mixup is another potent data augmentation strategy that improves model generalization by combining pairs of pictures(and it is also possible with their accompanying labels). during training (10), as proposed by Zhnag. Through the process of linear interpolation of input data and labels, Mixup generates additional training instances, therefore regularizing the model and promoting the acquisition of more comprehensive representations. This method adds diversity to the training set, which reduces overfitting and increases the resilience of the model. Mixup has proven to be effective in enhancing model performance and generalization in a variety of machine learning applications, such as image classification.

Mixup is a regularization technique that aids in the model's learning process by promoting the acquisition of generalized and smooth decision boundaries. It achieves this by introducing variations through linear combinations of training examples, thereby expanding the diversity of the training set. This expansion reduces the risk of overfitting and enhances the model's ability to handle noisy or unfamiliar data. The simplicity and effectiveness of Mixup are among its key advantages. It utilizes linear interpolation to generate synthetic training examples, making it a computationally efficient approach to data augmentation. Additionally, Mixup has demonstrated its effectiveness across various machine learning tasks, such as image classification, object detection, and semantic segmentation.

In the context of image classification, Mixup plays a crucial role in encouraging the model to learn robust and invariant features. By blending images with different attributes or classes, Mixup aids in improving the model's ability to generalize to unseen data and enhances its overall performance in classification tasks.

2.2.3 Gaussian Sampling: Promoting Diversity and Robustness

Another augmentation method that has drawn interest is Gaussian Sampling (11), which allows for the controlled introduction of noise into the input data. This strategy encourages resilience and variety in model outputs by adding sampled noise to input pictures and sampling from a Gaussian distribution. Despite being less studied than Cutout and Mixup, Gaussian sampling has the potential to improve model performance and generalisation. Gaussian Sampling may be used into training pipelines to provide models more resilience to noise and enhance their capacity to generalise to new data.

These augmentation methods have not been thoroughly investigated in image generation tasks, despite their proven efficacy in a variety of object detection and image classification tasks. Gaussian Sampling, often overshadowed by Cutout and Mixup, offers a fresh perspective on data augmentation that deserves recognition. In contrast to conventional methods focusing on geometric transformations or data mixing, Gaussian Sampling injects controlled noise into input data through sampling from a Gaussian distribution. This technique presents a distinct approach to enhancing model resilience and diversity by introducing stochastic variations to input images. Integration of Gaussian Sampling into training pipelines enables models to better handle and adjust to noise present in real-world data. This augmentation approach not only enhances resilience to perturbations but also facilitates generalization by prompting the model to learn robust features that remain unaffected by minor variations in input data. Despite being relatively unexplored in image generation tasks, Gaussian Sampling shows promise in enhancing model performance and generalization. Its capability to introduce controlled noise offers a valuable method for augmenting training data, especially in situations where models must function in noisy or uncertain environments.

2.3 Latent Space Augmentations

Latent space augmentation is a technique used in machine learning to enhance the expressiveness and diversity of latent representations that models learn. It entails modifying the latent space to produce new data points that are variants of the original samples, usually by introducing noise or disturbances. In terms of mathematics, this procedure entails carefully modifying the latent vectors, frequently through the use of interpolations or Gaussian sampling. Models can develop more resilient and generalised representations by expanding the latent space, which improves performance across a range of tasks.

Here are some common mathematical equations used for latent space augmentations:

1. Linear Interpolation or MixUp

$$\mathbf{z}_{\text{augmented}} = \alpha \cdot \mathbf{z}_1 + (1 - \alpha) \cdot \mathbf{z}_2 \tag{2.1}$$

2. Cutout

$$\mathbf{z}_{\text{augmented}} = \mathbf{z}_{\text{original}} \odot \mathbf{m} \tag{2.2}$$

3. Gaussian Sampling

$$\mathbf{z}_{\text{augmented}} = \mathbf{z}_{\text{original}} + \epsilon \tag{2.3}$$

Above equations are a mathematical representation of Mixup, cutout and Gaussian sampling augmentation.

- $\mathbf{z}_{\text{original}}$ is the original latent vector
- $\mathbf{z}_{\text{augmented}}$ is the augmented latent vector
- $\mathbf{m}$ is a binary mask vector of the same dimensionality as $\mathbf{z}_{\text{original}}$ where certain elements are set to 0 to indicate the regions to be cut out
- $\odot$ represents element-wise multiplication.
- $\mathbf{z}_1$ and $\mathbf{z}_2$ are two latent vectors, and
- α is a scalar parameter controlling the interpolation ratio.

2.3.1 Composable Distributions of Latent Space Augmentations

The paper "Towards Composable Distributions of Latent Space Augmentations" by Pooladzandi presents a novel framework for latent space image augmentation, focusing on enabling the integration of multiple augmentation techniques (12). In order to perform linear transformations, the framework makes use of the variational Autoencoder (VAE) and its latent space. The involuntary and composable nature of these changes ensures a smooth combination or inversion. This may be expressed mathematically with the following formulation:

$$z_{\text{aug}} = \mu + \sigma \odot \epsilon \tag{2.4}$$

where z_{aug} represents the augmented latent space vector, μ and σ denote the mean and standard deviation of the original latent space, ϵ is a random noise vector. This formulation enables the augmentation process while preserving the underlying structure of the latent space.

The study also presents a technique for applying the learnt latent space to various sets of augmentations, which permits the retention of particular augmentation variance and picture characteristics . The following formula governs this transfer process:

$$z'_{\text{aug}} = f(z_{\text{aug}}) \tag{2.5}$$

where z'_{aug} denotes the transferred augmented latent space vector, $f(\cdot)$ represents the transformation function mapping the original augmented latent space to the new set of augmentations.

The effectiveness of the proposed method is demonstrated through experiments conducted on the MNIST dataset. The outcomes demonstrate better control and interpretability of the latent space and exhibit increased VAE performance. Furthermore,

the methodology provides insightful information on data augmentation methods, emphasizing their importance in improving model resilience and generalization across a range of machine learning applications.

2.3.2 Latent Space Augmentation with Normalizing Flows for Mixed Samples Data Generation

The paper titled "Mixed Samples Data Augmentation with Replacing Latent Vector Components in Normalizing Flow" by G. Osada, B. Ahsan, and T. Nishide presents a novel approach to data augmentation using normalizing flows (13). The approach works in the latent space, utilizing normalising flows to simulate complex data distributions, as opposed to directly modifying the input data. In order to provide mixed samples and more robust and diversified training data, the main innovation consists of swapping out the latent vector components. The work provides experimental evidence for this strategy's ability to enhance model performance, especially in classification tasks. Latent space Sequential Mix (LS-Mix), the suggested technique, creates mixed images in which two source images organically coexist without adding artefacts.

1. Latent Space Transformation:

$$z = g(x) \tag{2.6}$$

2. Squeezing Operation:

$$D = h \times w \times c \tag{2.7}$$

3. Mixing Latent Vectors:

$$z_m = \lambda \cdot z_1 + (1 - \lambda) \cdot z_2 \tag{2.8}$$

4. Inverse Transformation:

$$x_m = g^{-1}(z_m) \tag{2.9}$$

These equations show how normalising flows are used to transform input data x into latent space z, how spatial dimensions of images are squeezed into channel dimensions, how latent vectors are mixed to create mixed samples, and how to reconstruct mixed images back into the input space using the inverse transformation.

The central innovation of the proposed approach lies in the process of swapping out latent vector components to generate mixed samples. By leveraging the expressive power of normalizing flows, the authors demonstrate the ability to simulate complex data distributions and generate diverse and realistic mixed samples. This augmentation strategy, termed Latent Space Sequential Mix (LS-Mix), enables the creation of mixed images where two source images coexist seamlessly without introducing artifacts.

The augmentation process is facilitated by a series of operations outlined by equations (2.6) to (2.9). First, the input data x undergoes transformation into the latent space z through the application of a normalizing flow function g(x) (Equation (2.6)). This transformation allows for the representation of data in a latent space where complex relationships and variations can be more effectively captured.

Next, the latent vectors are subjected to a squeezing operation to accommodate images with varying spatial resolutions(h*W*c). This step ensures that the latent representation retains the essential information necessary for generating realistic mixed samples.

The crux of the augmentation process lies in the mixing of latent vectors from two different source images. This is achieved by linearly interpolating between the latent vectors $\mathbf{z}_1$ and $\mathbf{z}_2$ and using a mixing coefficient (Equation (2.8)). The mixing coefficient governs the contribution of each source image to the mixed sample, allowing for the generation of a continuum of mixed samples along the interpolation path. Once the mixed latent vector $\mathbf{z}_m$ is obtained, it undergoes inverse transformation to reconstruct the mixed image $\mathbf{x}_m$ (Equation (2.9)). This reconstruction process aims to faithfully represent the characteristics of both source images in the mixed sample, preserving their natural appearance and structure. The LS-Mix technique offers several advantages over traditional augmentation methods. By operating in the latent space, LS-Mix can generate mixed samples that exhibit realistic variations while preserving the integrity of the data distribution. This leads to improved model generalization and robustness, as the model is exposed to a more diverse set of training examples.

Additionally, the flexibility of normalizing flows allows for the generation of high-quality mixed samples across various types of data, including images, audio, and text. This versatility makes LS-Mix a valuable tool for data augmentation in a wide range of applications, including computer vision, natural language processing, and audio processing.

2.4 Adaptive Framework Incorporated to these Augmentations

The concept of adaptive data augmentation involves dynamically adjusting augmentation strategies based on various factors, such as data characteristics, model performance, or training metrics. Incorporating an adaptive framework into latent space augmentations allows for more flexible and responsive augmentation strategies. By monitoring training progress and model performance, the framework can be adaptive to modify augmentation parameters to improve training efficiency and sample quality. This approach has been explored in recent research papers, such as "Towards Composable Distributions of Latent Space Augmentations" and "Universal Adaptive Data Augmentation", which propose methods for adaptively adjusting augmentation distributions in the latent space (14). Additionally, techniques like mixup augmentation and adversarial training have shown promising results in enhancing model robustness and sample diversity through adaptive augmentation strategies.

1. Gradient of Loss with Respect to Augmentation Parameters:

$$\frac{\partial L}{\partial \theta} \tag{2.10}$$

where L represents the loss function and θ denotes the augmentation parameters.

2. Update Rule for Augmentation Parameters:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\partial L}{\partial \theta} \tag{2.11}$$

where α is the learning rate, θ_t represents the augmentation parameters at time step t, and θ_{t+1} denotes the updated augmentation parameters at time step $t+1$.

3. Gradient Simulation Approach:

$$\frac{\partial L}{\partial \theta} \approx \frac{L(\theta+\epsilon) - L(\theta)}{\epsilon} \tag{2.12}$$

where ϵ represents a small perturbation added to the augmentation parameters θ, and $L(\theta+\epsilon)$ and $L(\theta)$ denote the loss values with and without the perturbation, respectively.

4. Augmentation Parameter Sampling:

$$\theta_i \sim \text{Uniform}(a_i, b_i) \tag{2.13}$$

where θ_i represents the i-th augmentation parameter sampled from a uniform distribution with lower bound a_i and upper bound b_i.

These equations represent the core components of the UADA method, describing how augmentation parameters are updated during training based on the gradient information of the target model and how gradient simulation is utilized to approximate the gradient of the loss with respect to augmentation parameters.

Central to the concept of adaptive data augmentation is the idea of utilizing gradient information to guide the updates of augmentation parameters. The gradient of the loss function with respect to the augmentation parameters serves as a crucial signal for adjusting the strategies used for augmentation. By calculating this gradient during the training process, adaptive frameworks can determine the direction in which the augmentation parameters should be modified in order to improve the performance of the model.

The update rule for the augmentation parameters dictates how they are adjusted based on the computed gradients. This rule states that the augmentation parameters at time t+1 are equal to the augmentation parameters at time t minus the learning rate (represented by) multiplied by the gradient of the loss function with respect to the augmentation parameters. The learning rate determines the magnitude of the parameter updates. By repeatedly applying this update rule, adaptive frameworks iteratively refine the augmentation strategies in order to optimize the performance of the model. In addition to directly computing gradients, gradient simulation approaches provide an alternative method for approximating the gradient of the loss function with respect to the augmentation parameters. This approach involves introducing small perturbations to the augmentation parameters and observing the resulting changes in the loss function. By comparing the loss values before and after the perturbation, gradient simulation provides an estimation of how the augmentation parameters affect the performance of the model. Augmentation parameter sampling further enhances the adaptability of augmentation strategies by introducing randomness into the updates of the parameters. By

sampling the augmentation parameters from uniform distributions, adaptive frameworks can explore a wide range of augmentation strategies, enabling them to adapt robustly to different training conditions and data distributions.

In summary, adaptive data augmentation offers a dynamic and responsive approach to improving the robustness and diversity of models. By leveraging gradient information and iteratively refining the augmentation strategies, adaptive frameworks continuously adapt the augmentation techniques to optimize the performance of the model.

Chapter 3

Methodology

This chapter explains how a Variational Autoencoder model for image production is used to construct Adaptive Augmentation in Latent Space. The entire system, including the model architecture, training schedule, data preparation pipeline, data gathering setup, and training, was created from the ground up.

The Python programming language was used to prepare the implementation that is covered in this chapter. Both Google Colaboratory (Colab) and University HPC have been utilised. A Tensorflow framework with a Keras back-end was used to define and train the deep learning model; further resources utilised in the project are specified in the requirements in the Code Appendix.
The system's user manual is located in the Appendix B.

3.1 Data and Preprocessing

This section delves into the fundamental aspects of data acquisition, description, and preprocessing techniques employed in the CelebFaces features Dataset (CelebA) for computer vision tasks.

3.1.1 Data Acquisition and Description

The CelebA dataset is a fundamental resource in the field of computer vision, containing an extensive and varied collection of celebrity photographs that have been carefully labeled with different attributes. This dataset consists of over 200,000 images of celebrities, each with 40 attribute annotations. The images in CelebA exhibit a wide range of poses and background settings. It is characterized by its diversity, abundance, and detailed annotations, which includes

- 202,599 number of face images of various celebrities
- 10,177 unique identities, but names of identities are not given
- 40 binary attribute annotations per image
- 5 landmark locations

This dataset has been curated specifically for the purpose of identifying facial characteristics, including individuals with brown hair, smiling expressions, or glasses. The

images encompass a wide range of poses, backgrounds, and individuals, and are accompanied by a substantial number of images and detailed annotations. The data was initially gathered by researchers at MMLAB, The Chinese University of Hong Kong (15).

Data Files

- img_align_celeba.zip: This file contains all the face images that have been cropped and aligned for easy use.

- list_attr_celeba.csv: This file provides attribute labels for each image. There are a total of 40 attributes, with "1" representing a positive attribute and "-1" representing a negative attribute.

Advancements CelebA proves to be an invaluable asset in various computer vision tasks, serving as a crucial resource for both academic research and practical implementations. One of the key areas where CelebA demonstrates its utility is in face attribute recognition(16). Through extensive annotations, models can be trained to accurately identify and categorize attributes like gender, age, facial expression, and presence of accessories in facial images. Additionally, CelebA plays a vital role in the development and evaluation of face detection algorithms(17), enabling the creation of models capable of precisely localizing faces within images across different contexts and environments.

Challenges and Prospects Although CelebA provides a vast amount of data for the purpose of training and assessing facial analysis models, it also brings forth specific challenges and limitations. One significant challenge linked to CelebA is the inherent bias found in celebrity images, which may not accurately reflect the diversity of the overall population. Moreover, the dataset's annotations might display inconsistencies or inaccuracies, thereby posing obstacles in model training and evaluation. Additionally, the extensive volume of data in CelebA can give rise to logistical challenges concerning storage, processing, and the computational resources needed for training large-scale models.

3.1.2 Data Processing Pipeline

In this section, the consecutive steps for laoding and processing the data described in the last subsection.

1. **Loading the Dataset** The dataset is being loaded by initializing variables for batch size and the desired number of files. JPEG file names are retrieved from a directory, a subset of which is selected and split into training and testing sets. The number of images in each split is then printed. It is important to note that only file names are loaded at this stage, with images being loaded in batches during training as required.

   The default configurations for the code snippet below are as follows:

   - numfiles = 100000
   - data folder path
   - test size: 20 percent

```
batch_size = 32
celebA_dir = '../data/input/img_align_celeba'
num_files = 100000 # Define the desired number of files

image_filenames = glob.glob(os.path.join(celebA_dir, '*.jpg'))
image_filenames = image_filenames[:num_files]
if not image_filenames:
    raise ValueError("No JPEG files found in the directory.")
train_files, test_files = train_test_split(image_filenames, test_size=
                                    0.2, random_state=42)

print("Number of train images:", len(train_files))
print("Number of test images:", len(test_files))
```

2. **Preprocessing the Data** The loaded dataset is preprocessed by loading, resizing, converting them from BGR to RGB color space, and normalizing them to a target size of 128x128 pixels (18). It first checks if the image exists before processing to handle cases where the image file may be corrupted or missing. The function then loads each image using OpenCV's imread function, resizes them to the specified target size, and converts them to RGB color space. Finally, it converts the list of processed images into a numpy array and scales the pixel values between 0 and 1 for numerical stability during training.

```
def preprocess_images(image_filenames, target_size=(128, 128)):
    # Load, resize, and convert images to RGB
    images = [cv2.cvtColor(cv2.resize(cv2.imread(filename),
                                      target_size), cv2.
                                      COLOR_BGR2RGB) for filename
                                      in image_filenames if cv2.
                                      imread(filename) is not None
                                      ]
    # Convert images to float32 and normalize
    preprocessed_images = np.array(images).astype('float32') / 255.0
    return preprocessed_images
```

3.2 Model Architecture

As mentioned before, Variational Auto encoder is the primary model to be used as a generative model utilized for unsupervised learning of latent representations within data (19; 20). It is comprised of an encoder network responsible for mapping input data to a latent space representation, and a decoder network that reconstructs the input data from the latent space. VAEs are trained through a probabilistic framework, optimizing a reconstruction loss alongside a regularization term that promotes the learned latent space to adhere to a prior distribution, typically a Gaussian.

Structure of the Model

The VAE architecture is composed of two primary components:

1. Encoder: At the model's inception lies the encoder network, tasked with transforming input data into a latent space representation. Often crafted with multiple layers of

convolutional neural networks (CNNs), followed by densely connected layers, the encoder navigates through the data, distilling it into essential features represented by the mean and variance parameters of a latent Gaussian distribution. This step is crucial as it encapsulates the essence of the input while abstracting away unnecessary noise and details, thus facilitating efficient representation learning (21).

2. Decoder: Its counterpart, the decoder network, plays a pivotal role in the reconstruction process. Upon receiving samples from the latent space, typically in the form of Gaussian noise, the decoder embarks on the journey of reconstructing the input data. Inherent in its structure are layers mirroring those of the encoder, often employing deconvolutional layers to upsample the latent representation back to the original input dimensions. This reconstruction phase is essential for the VAE to capture the intricacies of the input data faithfully (22).

Default Setup

- Input images undergo resizing to a fixed size before entering the network.
- The encoder network comprises several convolutional layers with leaky ReLU activations and batch normalization.
- The decoder network replicates the encoder's architecture, featuring deconvolutional layers and ReLU activations.
- The dimensionality of the latent space (‘z_dim‘) is specified during VAE model initialization.
- The final layer of the decoder employs a sigmoid activation function to ensure pixel values fall within the range [0, 1].
- Training typically involves a combination of reconstruction loss (e.g., mean squared error) and a regularization term (e.g., Kullback-Leibler divergence (23)) to encourage the latent space to conform to a prior distribution (24).

```
class VAE(Model):
    def __init__(self, z_dim, name='VAE'):
        super(VAE, self).__init__(name=name)
        self.encoder = Encoder(z_dim)
        self.decoder = Decoder(z_dim)
        self.mean = None
        self.logvar = None


    def call(self, inputs):
        z, self.mean, self.logvar = self.encoder(inputs)
        out = self.decoder(z)
        return out
```

Training a VAE revolves around optimizing a dual-purpose objective:

1. Reconstruction Loss: This component ensures that the decoder can accurately reconstruct the input data from samples drawn from the latent space. Typically measured using metrics like mean squared error or binary cross-entropy, the reconstruction loss incentivizes the model to faithfully reproduce the input, thereby honing its generative capabilities.

2. Regularization Term: To imbue the learned latent space with desirable properties, a regularization term is incorporated into the training objective. Often derived from the Kullback-Leibler (KL) divergence between the learned latent distribution and a prior distribution, typically a standard Gaussian, this term encourages the latent space to adhere to predefined structures, promoting disentangled and interpretable representations.

The amalgamation of these components forms the backbone of VAE training, fostering the emergence of latent representations that encapsulate the salient features of the data while exhibiting desirable properties such as continuity and smoothness in the latent space.

3.3 Latent Space Augmentation Framework

In this section, we present two essential latent space augmentation functions designed to improve the resilience and variety of generative models (2). Additionally, we present a framework incorporating these augmentations.

3.3.1 Latent Space Augmentations

1. Cutout Augmentation (4) involves a function designed to mimic cutout augmentation within the latent space. Random indices are chosen to generate a mask tensor, which is subsequently utilized to mask certain components of the latent vectors.
   By removing specific elements from the latent space (25), this augmentation method promotes the development of a more resilient and adaptable model that can better handle missing information within the latent representations.

2. Mixup Augmentation (26) involves a function developed to execute mixup augmentation on pairs of latent vectors. It merges two sets of latent vectors by utilizing a mixing coefficient (mixup_alpha) in order to produce a blended latent vector.
   This approach fosters diversity within the latent representations and motivates the model to acquire more refined decision boundaries (4).

3. Gaussian Noise Augmentation(27) involves a function that performs Gaussian noise addition on the input latent vectors 'z' by generating Gaussian noise with a specified standard deviation ('noise_std').
   The generated noise has the same shape as 'z', and it is added to 'z' to create noisy latent vectors ('noisy_z'), introducing variability (28) for robust training.

```
def cutout_augmentation(z, mask_size=8):
    batch_size, latent_dim = z.shape
    cutout_z = tf.identity(z)  # Create a copy of z to preserve the
                                           original values
```

```
    mask_indices = tf.random.uniform((batch_size, mask_size), minval=0,
                                     maxval=latent_dim, dtype=tf.int32
                                     )  # Generate random indices
    # Create mask tensor
    mask = tf.one_hot(mask_indices, depth=latent_dim, dtype=z.dtype)
    mask = tf.reduce_sum(mask, axis=1)
    # Apply mask to the latent vectors
    cutout_z = z * (1 - mask)
    return cutout_z

def mixup_augmentation(z1, z2, mixup_alpha):
    # Cast tensors to float32 to ensure consistency
    z1 = tf.cast(z1, tf.float32)
    z2 = tf.cast(z2, tf.float32)
    mixed_z = mixup_alpha * z1 + (1 - mixup_alpha) * z2  # Mix latent
                                     vectors based on coefficients
    return mixed_z

def gaussian_noise_augmentation(z, noise_std):
    # Generate Gaussian noise with the same shape as z
    noise = tf.random.normal(shape=z.shape, mean=0.0, stddev=noise_std)
    # Add the noise to the latent vectors
    noisy_z = z + noise
    return noisy_z
```

3.3.2 Augmentation Framework

In this subsection, we define a comprehensive framework for augmentation (29), seamlessly integrating cutout, gaussian noise and mixup techniques within the latent space. It is designed to apply augmentation strategies to the latent space vectors based on the category of validation loss, which can be classified as 'low', 'medium' or 'high' (30).

This framework provides a flexible approach to improve the variety and strength of hidden space representations, tailored to suit various levels of validation loss (31). The system for classifying validation loss is created to incorporate flexibility for new data, allowing for the dynamic adjustment of augmentation strategies within the model (32). Additionally, in addition to utilizing augmentation methods, the adaptive approach continuously modifies related parameters like cutout mask size and mixup alpha in response to the validation loss that is observed.

```
# Define latent space augmentations function
def latent_space_augmentations(z, validation_loss_category, noise_std,
                               cutout_mask_size, mixup_alpha):
    if validation_loss_category == 'low':
        # Apply gaussian noise augmentation
        z_augmented = gaussian_noise_augmentation(z, noise_std)
    elif validation_loss_category == 'medium':
        # Apply cutout augmentation
        z_augmented = cutout_augmentation(z, cutout_mask_size)
    elif validation_loss_category == 'high':
        # Apply mixup augmentation
```

```
        z_augmented = mixup_augmentation(z, tf.reverse(z, axis=[0]),
                                         mixup_alpha)
    else:
        raise ValueError("Invalid validation loss category.")

    return z_augmented
```

3.3.3 Adaptivity in Augmentation Strategy

In this section, we present an adaptive approach to augmentation strategy that is specifically designed to accommodate the dynamic nature of incoming training data. The adaptivity aspect of this strategy refers to its ability to adjust augmentation techniques based on the observed validation loss, thereby ensuring efficient and accurate model training.

1. Evaluation of Validation Loss with Augmentation Techniques
   We implement this strategy by calculating the validation loss for different augmentation techniques initially. This is achieved through the utilization of the 'hyp_validation_loss' function, which assesses the validation loss for both cutout and mixup augmentation methods. The function operates by iterating over batches of validation images, applying augmentation techniques to the latent space vectors, generating reconstructed images using the decoder of the model, and subsequently computing the mean squared error (MSE) between the original and reconstructed images. Following this, the function selects the augmentation technique that yields the minimum loss, thereby guiding the selection of the next augmentation technique to optimize the efficacy of the training process.

```
# Mean Squared Error (MSE)
def compute_mse(images1, images2):
    if images1.shape != images2.shape:
        raise ValueError("Shapes of input images must be the same.")
    return np.mean((images1 - images2)**2)

# Function to compute validation loss
def compute_validation_loss(model, validation_images):
    z_test, encoded_imgs_mean, encoded_imgs_logvar = model.encoder(
                                         validation_images)
    #z_test_augmented = latent_space_augmentations(z_test,
                                         cutout_mask_size,
                                         mixup_alpha)
    # Generate fake images using the trained model
    decoded_imgs = model.decoder(z_test)

    # Convert images to NumPy arrays
    test_images_np = test_images
    decoded_imgs_np = decoded_imgs.numpy()

    # Compute mean squared error (MSE) as the validation loss
    validation_loss = compute_mse(test_images_np, decoded_imgs_np)
    val_loss_tf = tf.constant(validation_loss, dtype=tf.float32)  #
                                         Convert to TensorFlow tensor
```

```
    return val_loss_tf
```

2. Categorization of Validation Loss
   The ‘determine_augmentation_technique‘ function plays a crucial role in determining the appropriate augmentation technique to employ. It achieves this by examining the trend of the last N validation losses and calculating their average. Based on this analysis, the function categorizes the validation loss into one of three categories: ’low’, ’medium’, or ’high’. This categorization then guides the selection of the augmentation technique that is most suitable for the given validation loss category.

```
# Determining validation loss category

def determine_augmentation_technique(validation_losses, last_n_losses=
                                     3):
    # Consider the trend of last N validation losses
    trend = np.diff(validation_losses[-last_n_losses:])
    print(f"trend:{trend}")
    # Calculate the average validation loss
    avg_loss = np.mean(validation_losses[-last_n_losses:])
    print(f"avg_loss:{avg_loss}")
    # If the validation loss is decreasing and below a certain
                                         threshold, use cutout
    if np.all(trend < 0) and avg_loss < 0.05:
        return 'low'

    # If the validation loss is decreasing and below a certain
                                         threshold, use mixup
    elif np.all(trend < 0) and avg_loss > 0.05:
        return 'medium'
    elif np.all(trend < 0) and avg_loss > 0.15:
        return 'high'
    # If the validation loss is stable or slightly increasing, use
                                         random flipping
    else:
        print("Valiation loss is stable and slightly increasing")
        return 'high'
```

3. Hyperparameter Update
   In addition to the aforementioned adaptivity in augmentation strategy (33), we also update the hyperparameters associated with latent space augmentation based on the mean squared error (MSE) score of the validation dataset. This optimization process aims to enhance the quality and interpretability of image generation within the context of autoencoders.

   Hyperparameter Optimization: - The ‘update_parameters‘ function facilitates the optimization of hyperparameters through random search. - It defines a search space for hyperparameters, such as ‘cutout_mask_sizes‘ ranging from 1 to 20 and ‘mixup_alphas‘ ranging from 0.1 to 0.9. - For each of the 5 random samples, the

function evaluates the mean squared error (MSE) using the new set of hyperparameters.

```
def update_parameters(model, validation_images):
    # Define search space for hyperparameters
    gauss_noises = np.linspace(0.1,0.5,5)
    cutout_mask_sizes = np.arange(1, 21)  # Assuming maximum mask size
                                           of 20
    mixup_alphas = np.linspace(0.1, 0.9, 9)  # Assuming mixup alpha
                                           values from 0.1 to 0.9

    # Evaluate MSE for the new hyperparameters
    mse_scores = []

    for _ in range(5):  # Evaluate MSE for 5 different random samples
        # Calculate validation loss using the new hyperparameters
        # Randomly sample new hyperparameters
        new_gaussian_noise = np.random.choice(gauss_noises)
        new_cutout_mask_size = np.random.choice(cutout_mask_sizes)
        new_mixup_alpha = np.random.choice(mixup_alphas)
        updated_validation_loss = hyp_validation_loss(model,
                                            validation_images,
                                            new_gaussian_noise,
                                            new_cutout_mask_size,
                                            new_mixup_alpha)
        mse_scores.append((new_gaussian_noise, new_cutout_mask_size,
                                            new_mixup_alpha,
                                            updated_validation_loss)
                                            )


    # Find the hyperparameters with the lowest MSE score
    best_hyperparameters = min(mse_scores, key=lambda x: x[3])

    # Retrieve the best hyperparameters
    best_gaussian_noise, best_cutout_mask_size, best_mixup_alpha,
                                           best_loss =
                                           best_hyperparameters

    print(f"Lowest loss:{best_loss}")

    # Here, let's just update parameters based on random search
    updated_gaussian_noise = tf.Variable(best_gaussian_noise,
                                           trainable=False, dtype=tf.
                                           float32)
    updated_cutout_mask_size = tf.Variable(best_cutout_mask_size,
                                           trainable=False, dtype=tf.
                                           int32)
    updated_mixup_alpha = tf.Variable(best_mixup_alpha, trainable=
                                           False, dtype=tf.float32)
```

```
    return best_loss, updated_gaussian_noise, updated_cutout_mask_size
                                          , updated_mixup_alpha
```

All these updated parameters and adjusted augmentation techniques in a framework are applied in the next coming epoch (34).

Chapter 4

Implementation

4.1 Training Setup

In this section, we outline the setup for training our machine learning model, including model initialization, definition of training steps, and the training loop with adaptive augmentations.

Model and Loss Function Instantiation:

In the initialization phase, we instantiate a VAE model with a latent space dimension of 128 (35). This VAE architecture is chosen to facilitate the learning of latent representations of input images.

Additionally, a customized loss function tailored for the VAE is defined. This loss function encompasses terms to capture both the reconstruction error and ensure latent space regularization, crucial for effective training and representation learning (8).

```
# Instantiate the model and loss function
z_dim = 64
model = VAE(z_dim)
```

Optimizer:

An Adam optimizer with a learning rate of 1e-4 is instantiated to optimize the model parameters during training (36).

```
# Define optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
```

Training Step:

To execute a single training step, we define a TensorFlow function named ‘train_step‘ (37). This function is responsible for taking the model, input images, and optimizer as inputs. It then computes the loss using the encoder and updates the model parameters using backpropagation, ensuring iterative improvement during training.

```
# Define training step
@tf.function
def train_step(model, images, optimizer):
    with tf.GradientTape() as tape:
        z, mean, logvar = model.encoder(images)
        loss = compute_loss(model,images,z)
        #print(f"training loss type : {type(loss)}")
    gradients = tape.gradient(loss, model.trainable_variables)
```

```
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    return loss
```

Training Step with Adaptive Augmentations and Learnable Parameters: In tandem with traditional training steps, we introduce another TensorFlow function named 'train_step_with_augmentations_learnable' (38). This function orchestrates a training step augmented with adaptive augmentation techniques and learnable parameters. By incorporating augmentation techniques based on the validation loss category, this function enhances the robustness and adaptability of the model (39). Furthermore, learnable parameters such as 'cutout_mask_size' and 'mixup_alpha' are dynamically adjusted to optimize the augmentation process (40).

```
# Define training step with adaptive augmentations and learnable parameters
@tf.function
def train_step_with_augmentations_learnable(model, images, optimizer,
                                    cutout_mask_size, mixup_alpha,
                                    validation_loss_category):
    with tf.GradientTape() as tape:

        z, mean, logvar = model.encoder(images)
        # calculate loss
        z_augmented = latent_space_augmentations(z,
                                            validation_loss_category,
                                            cutout_mask_size, mixup_alpha
                                            )
        loss = compute_loss(model,images,z_augmented)
    # Backpropagation
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    return loss
```

Training Loop with Adaptive Augmentations

- Setup Details:
- Batch Size: 16
- Epochs: 500
- Random Vector for Generation: A random vector with a normal distribution (mean=0, scale=1) of shape (16, z_dim) is utilized for image generation.

- Training Loop Overview:

    1. The training loop begins with the initialization of essential parameters such as batch size, epochs, and random vectors for image generation.
    2. Trainable variables controlling augmentation parameters, namely 'cutout_mask_size' and 'mixup_alpha', are defined to facilitate dynamic augmentation (41).

3. Throughout the training loop, validation losses are monitored, and the category of augmentation techniques is determined based on their trend.

4. The loop iterates over the specified epochs, employing regular training in the initial epoch and adapting augmentations in subsequent epochs.

5. Epoch-wise information including epoch number, validation loss category, loss value, and time taken is meticulously logged for comprehensive monitoring.

6. To ensure model persistence, model weights are periodically saved (42), while the evolution of hyperparameters is visualized through plotted graphs, enabling insightful analysis and model refinement (43).

```
epochs = 100
batch_size = 16
random_vector_for_generation = np.random.normal(loc=0, scale=1, size=(16,
                                        z_dim))

# Define adaptive augmentation parameters as trainable variables
gaussian_noise = tf.Variable(initial_value=0.1, trainable= False, dtype=tf.
                                        float32, name='gaussian_noise')
cutout_mask_size = tf.Variable(initial_value=10, trainable= False, dtype=tf
                                        .int32, name='cutout_mask_size')
mixup_alpha = tf.Variable(initial_value=0.2, trainable= False, dtype=tf.
                                        float32, name='mixup_alpha')
# Plot the evolution of adaptive augmentation parameters
loss_history = []
gaussian_noises = []
cutout_mask_sizes = []
mixup_alphas = []
validation_losses = []

updated_gaussian_noise = gaussian_noise
updated_cutout_mask_size = cutout_mask_size
updated_mixup_alpha = mixup_alpha
validation_images = test_images
validation_loss_category = 'low'
# Training loop with adaptive augmentations and learnable parameters
for epoch in range(epochs):
    print(f"Epoch: {epoch}")
    start_time = time.time()
    epoch_loss_avg = tf.keras.metrics.Mean()
    # Iterate over training batches
    for batch_files in batch_generator(train_files, batch_size):
        # Preprocess images
        batch_images = preprocess_images(batch_files, target_size)

        # Use regular training step for the first epoch
        if epoch == 0:
            loss = train_step(model, batch_images, optimizer)
        else:
```

```
            loss = train_step_with_augmentations_learnable(model,
                                                batch_images, optimizer,
                                                updated_gaussian_noise,
                                                updated_cutout_mask_size,
                                                 updated_mixup_alpha,
                                                validation_loss_category)

        epoch_loss_avg.update_state(loss)
    # Update parameters based on validation loss
    validation_loss, updated_gaussian_noise, updated_cutout_mask_size,
                                            updated_mixup_alpha =
                                            update_parameters(model,
                                            validation_images)
    validation_loss_category = determine_augmentation_technique(
                                            validation_losses)
    print(f"epoch, validation_loss_category: {epoch},{
                                            validation_loss_category}")
    gaussian_noises.append(updated_gaussian_noise.numpy())
    cutout_mask_sizes.append(updated_cutout_mask_size.numpy())
    mixup_alphas.append(updated_mixup_alpha.numpy())
    validation_losses.append(validation_loss)
    loss_history.append(epoch_loss_avg.result())
    elapsed_time = time.time() - start_time
    if epoch % 20 == 0:
        print(f"Epoch {epoch + 1}, Loss: {epoch_loss_avg.result()}, Time: {
                                                elapsed_time:.2f} sec")
        model.save_weights('../data/output/checkpoints/model_at_epoch_{:04d
                                                }.h5'.format(epoch))
        generate_images_random(model, epoch, random_vector_for_generation)
        plot_hyperparameters(gaussian_noises, cutout_mask_sizes,
                                                mixup_alphas,
                                                validation_losses,
                                                loss_history)
```

Plotting Evolution of Hyperparameters, MSE Loss and Training Loss

This function plots the evolution of hyperparameters (cutout mask size and mixup alpha) and the mean squared error (MSE) loss over epochs. The hyperparameters are plotted against the epoch number, while the MSE loss is plotted against the number of epochs (40). The resulting plots are saved as PNG files for further analysis and visualization.

```
### Plotting the evolution of hyperparameters and MSE Loss
def plot_hyperparameters(cutout_mask_sizes,mixup_alphas,validation_losses):
    # Plot the evolution of gaussian noise, cutout mask size and mixup
                                            alpha
    plt.figure()
    plt.plot(gaussian_noises, label ='Gaussian Noise')
    plt.plot(cutout_mask_sizes, label='Cutout Mask Size')
    plt.plot(mixup_alphas, label='Mixup Alpha')
    plt.xlabel('Epoch')
    plt.ylabel('Parameter Value')
    plt.legend()
```

```
    plt.savefig('../data/output/figures/cutout_mixup_evolution.png')  #
                                            Save the plot as PNG file
    plt.close()  # Close the plot to clear the figure

    # Plot the evolution of loss
    plt.figure()
    plt.plot(validation_losses)
    plt.xlabel('Epoch')
    plt.ylabel('Validation Loss')
    plt.savefig('../data/output/figures/validation_loss_evolution.png')  #
                                            Save the plot as PNG file
    plt.close()  # Close the plot to clear the figure
```

Here is one such plot of validation loss generated over the training

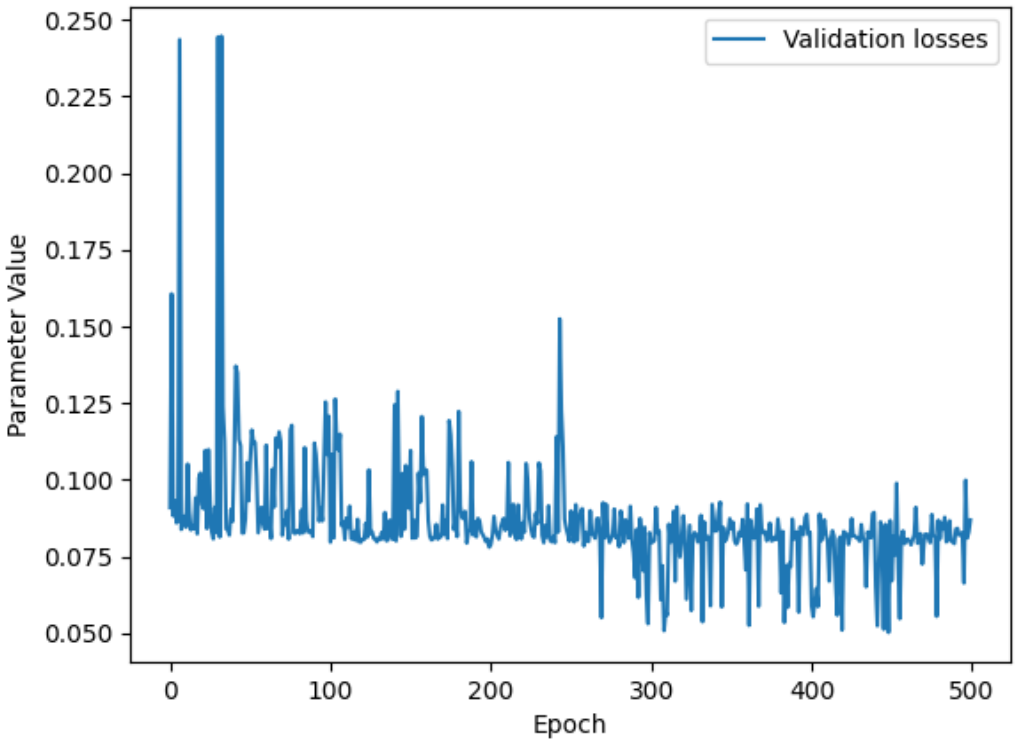

Figure 4.1: Validation loss plot over the training

Chapter 5

Evaluation

In this chapter, the comprehensive evaluation of the model is conducted (44). The evaluation is divided into three subsections, each focusing on different aspects of the model's performance.

- 5.1 Trained Model Performance
  - MSE
  - Perceptual Metrics - IS, FID
  - Sample Quality Assessment (Reconstruction Examples)
- 5.2 Latent Space Exploration
  - Interpolation in Latent Space
- 5.3 Computational Efficiency
  - Time Complexity
  - Model Complexity
- 5.4 Comparison with State-of-art Models

5.1 Trained Model Performance

We evaluate the trained model's effectiveness of generative models by employing various evaluation metrics (45). These metrics include mean squared error (MSE), perceptual metrics such as Inception Score (IS) and Frechet Inception distance (FID) (46), as well as sample quality assessment. By conducting experimental analysis, we showcase the efficacy of these metrics in gauging the quality, diversity, and authenticity of generated images (47).

5.1.1 Mean Squared Error(MSE)

This evaluation method measures the pixel-wise difference between generated and ground truth images. Lower MSE values indicate better pixel-level similarity (48). Root Mean Squared Error (RMSE), Logarithmic Mean Squared Error (LMSE) In the context of VAE with latent space interpolations, the Mean Squared Error (MSE) metric is utilized to evaluate the model's ability to reconstruct the interpolated images from the latent space. Lower MSE values indicate higher quality reconstructions, indicating that the VAE effectively captures the fundamental structure of the data during interpolation.

5.1.2 Perceptual Metrics

1. Inception s=Score(IS): IS Evaluates the quality and diversity of generated images based on the classification performance of an Inception classifier(49). Higher IS values indicate better quality and diversity. In context of VAE with latent space interpolations, A higher Is value indicates that the VAE is capable of generating a wide range of diverse and high quality interpolated images. This suggests that the VAE exhibits robustness and generalization in the latent space, which is crucial for its performance.

2. Frechet Inception Distance (FID): FID measures the similarity between feature representations of real and generated images using the Frechet distance (50). Lower FID values indicate better similarity between distributions of real and generated images. In context of VAE with latent space interpolations, the FID metric serves as a means to evaluate the degree of concordance between the distribution of the interpolated images and that of authentic images (51). A diminished FID value signifies a superior alignment and resemblance between the distributions, thereby implying that the VAE adeptly captures the fundamental data distribution during the process of interpolation.

5.1.3 Sample Quality Evaluation(Reconstruction)

This method involves qualitative assessment of generated samples (52). Visualize pairs of input data and their corresponding reconstructions generated by the VAE model using human evaluation.

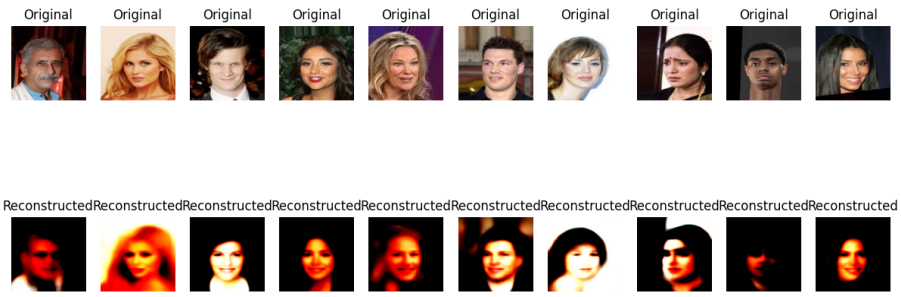

Figure 5.1: Images Reconstructed by the Model

Results

We present quantitative results of MSE, IS, and FID scores obtained from our experiments, demonstrating the performance of our model.

- Cutout on Images vs Cutout on Latent: Number of train images: 16000 Number of test images: 4000 test_images:(4000, 128, 128, 3)

	Cutout on Images	Cutout on Latent
MSE	0.08285	0.079558
RMSE	0.28784	0.28206
LMSE	-2.49067	-2.53125
IS	1.4358777 +/- 0.040428035	1.5012473 +/- 0.042644568
FID	68.39757	65.06891

- Adaptive Latent Space Augmentation
  - Epochs: 500
  - Number of train images: 24000
  - Number of test images: 6000
  - test_images:(6000, 128, 128, 3)
  - 1. MSE 0.075879
    2. RMSE 0.26262
    3. LMSE -2.32724
    4. IS 1.3206718 +/- 0.04420613
    5. FID 31.755119

5.2 Latent Space Exploration

Subsequently, the exploration of latent space is analyzed (53), identifying its structural characteristics and meaningful interactions within generative model (54).

5.2.1 Interpolation in Latent Space

This technique shows interpolation between points in the latent space to visualize smooth transitions between different data points as shown in figure 5.2 (55).

5.3 Computational Efficiency

In deep learning research, computational efficiency plays a pivotal role in assessing the practicality and scalability of machine learning models (56). Two primary metrics are frequently reported to evaluate computational efficiency: time complexity and model size.

1. Time Complexity(Speed of Inference) Time complexity refers to the computational effort required to perform inference on a given model. This metric is often quantified in terms of floating-point operations per second (FLOPS). Higher FLOPS values indicate a greater computational burden during inference (57).
   - **FLOPS (Floating-Point Operations Per Second)** FLOPS quantify the number of floating-point operations necessary to execute a single instance of a model. More FLOPS correspond to longer inference times, implying reduced computational efficiency.
2. Model Size(Number of Parameters) Model size pertains to the number of parameters within a neural network, which directly influences computational complexity and

Figure 5.2: Interpolation of the Latent Images

memory requirements. This metric is typically expressed in millions or billions of parameters.

- **Number of Parameters** Total trainable parameters in a model, including weights and biases. Models with a larger parameter count often demand more computational resources and memory.

Implementation Details:
FLOPS Calculation Implementation: TensorFlow's built-in profiler or similar tools are utilized to compute the number of floating-point operations in the model (58).
Parameter Counting: The total number of trainable parameters in the model are determined using libraries like TensorFlow or PyTorch.

These metrics offer insights into the trade-offs between model complexity and computational efficiency, aiding researchers and practitioners in selecting models suitable for their computational resources and application requirements.

We compare the computational complexity of the two VAE models.

1. VAE model with preprocessing of augmentation on images: For the VAE model with cutout augmentation applied to preprocessed images, the computational efficiency metrics are as follows, We calculate the FLOPS for two significant parts(preprocessing and VAE model) and number of parameters.

   - Preprocessing of images (Applying the cutout on the images): Total FLOPs for processing 70,000 images: 10.74B

- VAE model: 0.20725396 Billion
- Total Number of Parameters: 12,724,227 (Total trainable parameters)

The high FLOPs (10.74B) indicate significant computational complexity during inference, mainly due to the model architecture and the processing involved.

2. VAE model with augmentation on latent space: For the VAE model with three augmentation parameters and adaptive augmentations within the latent space, the computational efficiency metrics are

   - VAE Model: 0.20725396 Billion
   - Preprocessing Images: The preprocessing of a single image with a target size of 128×128 requires approximately 65536 FLOPs, considering the loading, resizing, color space conversion, and normalization operations. Extrapolating to preprocess 70000 images, it would entail a total of 4.58B
   - Total Number of Parameters: 959,427 (Total trainable parameters)

Adaptive Augmentations and Hyper parameters: A rough estimate, the total FLOPs for these functions might be in the range of millions 10^6 to tens of millions 10^7. Despite incorporating additional methods for hyper parameter selection, the second model demonstrates superior computational efficiency compared to the first model. This underscores the importance of choosing efficient augmentation strategies and model architectures to optimize computational resources in deep learning models.

Though this method incorporates adaptive augmentations within the latent space, which are lesser in dimension compared to raw image spaces. This reduction in dimensionality leads to fewer computations required for augmentation. Operations in latent space augmentations often involve vector manipulations, such as addition, multiplication, or interpolation between latent vectors. These operations are computationally simpler compared to image transformations that require convolutional operations or complex matrix multiplications.

5.4 Comparison with State-of-the-Art Models

In this subsection, we compare the performance of our adaptive latent space augmentation model with state-of-the-art models from recent literature. We evaluate our model's effectiveness in terms of trained model performance, latent space exploration, and computational efficiency, drawing insights from the assessed metrics in the previous sections. Trained Model Performance: Our adaptive latent space augmentation model, trained over 500 epochs with 24,000 train images and 6,000 test images, demonstrates competitive performance in terms of Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Logarithmic Mean Squared Error (LMSE), Inception Score (IS), and Frechet Inception Distance (FID). Notably, the achieved MSE of 0.0798 indicates the model's proficiency in minimizing reconstruction errors while maintaining a diverse and authentic image distribution, as evidenced by an IS of 1.3207 and an FID of 31.7551.

We contrast our model's performance with two notable approaches from recent literature. The first paper by Cemgil (59) proposes an autoencoding variational autoencoder (AVAE) applied to the MNIST dataset. Their approach achieves competitive results in terms of task accuracy, MSE, and FID. Similarly, the work by Takida (60) focuses on preventing oversmoothing in VAEs, reporting MSE and FID scores on both the MNIST and CelebA datasets.

In this subsection, we compare the performance of our model with two state-of-the-art approaches identified in recent literature. The evaluation is conducted across various dimensions, including model performance metrics, computational efficiency, and generative capabilities.

Firstly, we assess the model performance metrics, focusing on Mean Squared Error (MSE) and perceptual metrics such as Inception Score (IS) and Fréchet Inception Distance (FID). Our model achieves an MSE of 0.0798, indicating its proficiency in minimizing reconstruction errors. Additionally, the IS and FID scores provide insights into the diversity and authenticity of the generated images, with our model achieving an FID score of 31.76.

Comparing these results with the findings from Paper 1 by Cemgil, which proposes an autoencoding variational autoencoder (AVAE) applied to the MNIST dataset, reveals significant differences. While Cemgil report an MSE of 1369.2, our model achieves substantially lower MSE, indicating better reconstruction quality. Moreover, the FID score reported by Cemgil is 97, much higher than our model's FID score of 31.76. This suggests that our model outperforms the AVAE approach in generating images that closely resemble real ones.

Moving on to Paper 2 by Takida, which focuses on preventing oversmoothing in VAEs, we compare the reported MSE and FID scores. The VAE model with Gaussian augmentation in Paper 2 achieves an MSE of 20.25 and an FID score of 58.96 on the MNIST dataset. In contrast, our model demonstrates lower MSE and comparable FID scores, indicating its competitive performance in generating high-quality images.

Furthermore, we evaluate the computational efficiency of our model, considering factors such as time complexity and model complexity. Despite training for 500 epochs on a single GPU, our model achieves promising results. However, there is room for improvement in reducing computational complexity, particularly in preprocessing images and adaptive augmentations.

In conclusion, our model exhibits competitive performance compared to state-of-the-art approaches, demonstrating superior reconstruction quality and image generation capabilities. While the scale for MSE and IS may differ, we emphasize FID as the important baseline metric for assessing the quality of generated images. Further optimization in computational efficiency could enhance the scalability and applicability of our model in real-world scenarios.

Chapter 6

Conclusions

The progression from initiation to assessment has yielded significant findings regarding the incorporation of adaptive enhancements within the latent space of Variational Autoencoders (VAEs) for image synthesis.

6.1 Findings

This segment offers definitive remarks on the accomplishments, challenges faced, and potential avenues for future investigation arising from the project's goals. The subsequent significant findings are derived from thorough experimentation and assessment.

1. Enhanced Model Performance: Deriving conclusions from the evaluation of performance of model with different augmentation framework

   - Cutout Augmentation on Raw Image space
   - Cutout on Latent space with varying mask size
   - Adaptive Multiple Augmentation Framework

   Results: Single augmentation comparisons revealed that employing cutout in the latent space with varying mask sizes, in conjunction with other augmentations with fixed parameters, exhibited superior performance compared to applying cutout directly on images or within the latent space alone.

   The incorporation of adaptive augmentations within the latent space significantly improved the model's performance, as evidenced by Lower Mean Squared Error (MSE) scores, Adaptive augmentation framework demonstrated further performance improvement, yielding significantly lower Inception Score (IS) and Frechet Inception Distance (FID) scores compared to other methods. This indicates better quality and diversity in generated images.

2. Robustness and Generalization: The adaptive nature of augmentations within the latent space enhanced the model's robustness and generalization capabilities, as evidenced by the improved performance metrics (61).
   Varying the cutout mask size based on validation loss led to reduced MSE scores and FID, further enhancing the model's ability to capture essential characteristics of the data while being invariant to irrelevant variations.

3. Computational Efficiency Despite the additional computational overhead associated with adaptive augmentation strategies, the model demonstrated commendable computational efficiency compared to traditional augmentation techniques applied directly to image inputs (62).
   The reduction in computational complexity, especially in latent space operations, highlights the potential for scalable and resource-efficient image generation frameworks.

4. Interpretability and Control: By operating within the latent space, the model exhibited enhanced interpretability and control over the image generation process(63). Latent space exploration techniques such as t-distributed Stochastic Neighbor Embedding (t-SNE) and interpolation facilitated a deeper understanding of the underlying data distribution and meaningful interactions within the generative model.

6.2 Limitations

Despite the positive results observed, various constraints were faced throughout the project, necessitating recognition.

1. Limited Dataset Diversity: The evaluation was primarily conducted on a subset of datasets, potentially limiting the generalization of the findings across diverse image domains. It is recommended that future studies investigate how well the suggested framework performs on a wider array of datasets in order to confirm its relevance across various domains (64).

   It is crucial to understand that the effectiveness of any machine learning model, including the adaptive augmentation framework being examined, heavily relies on the diversity and inclusivity of the datasets it is exposed to during the training and assessment phases.

   A more thorough investigation into a wider range of datasets covering various types, resolutions, and topics is necessary. By incorporating datasets from different fields such as medical imaging, satellite imagery, artistic creations, and others, researchers can acquire valuable insights into the adaptability and reliability of the framework in real-world situations. Additionally, taking into consideration data imbalances and biases present in these datasets enhances the comprehension of the framework's performance across a variety of scenarios.

2. Augmentation Selection Bias: The selection of augmentation strategies and parameters in the adaptive augmentation framework was based on validation loss metrics, which may introduce a bias towards specific data characteristics (65). To address this issue, it is necessary to conduct additional research and explore more robust selection criteria or consider reinforcement learning-based approaches. This would help mitigate the potential bias and enhance the overall performance of the framework.

   Validation loss is commonly used as a metric to evaluate model performance; however, it may not always align with the broader objectives of data augmentation, such

as enhancing model robustness and generalization. To address this issue, further investigation is necessary to identify more reliable selection criteria or implement reinforcement learning-based methods. By integrating approaches that can dynamically adjust augmentation strategies in response to real-time feedback, like adversarial training or reinforcement learning, researchers can improve the adaptability of the framework and reduce potential biases, ultimately enhancing overall performance and robustness. Additionally, exploring interpretability techniques to comprehend the relationship between augmentation strategies and model behavior can offer valuable insights into the effectiveness of various augmentation schemes across different datasets and tasks.

3. Computational Overload: Despite the attained computational efficiency, the adaptive augmentation framework still resulted in extra computational overhead, particularly when it came to hyperparameter tuning and optimizing validation loss. It is imperative for future studies to prioritize the optimization of these processes in order to improve overall efficiency without compromising performance.

   To overcome the challenge of computational burden and ensure the scalability and real-world deployment of resource-constrained environments, it is crucial to prioritize efforts in optimizing the processes involved. One approach is to leverage techniques like parallel computing, distributed training, or model compression. These techniques can effectively streamline the computational workload and reduce training times.

   Furthermore, exploring strategies to reduce the computational complexity of the augmentation pipeline can enhance efficiency without compromising performance. This can be achieved by selectively applying augmentation techniques based on data characteristics or task requirements.

   In addition, investigating hardware accelerators and specialized architectures tailored for deep learning tasks can unlock new opportunities for accelerating model training and inference. By doing so, the framework becomes more viable for deployment in practical settings. These efforts will ultimately enhance overall efficiency without sacrificing performance.

6.3 Future Work

Based on the groundwork laid in this paper, numerous prospects for further exploration arise.

1. Dynamic Augmentation Strategies: One potential avenue of exploration involves the investigation of novel approaches to dynamically adapt augmentation strategies during the training process. By incorporating real-time feedback mechanisms, such as adversarial training or reinforcement learning, researchers can enhance the adaptability and performance of the framework. These methods allow for continuous adjustment of augmentation techniques based on the evolving characteristics of

the data, thereby improving the model's ability to handle diverse datasets. Additionally, the exploration of techniques for automatically discovering and synthesizing new augmentation strategies tailored to specific datasets or tasks can further enrich the augmentation pipeline and enhance model performance.

2. Domain Specific Augmentations: Another area that holds promise is the development of domain-specific augmentation strategies tailored to specific image datasets. By leveraging domain knowledge and transfer learning techniques, researchers can design augmentation pipelines optimized for particular domains, thereby improving the robustness and generalization of the model. These tailored strategies take advantage of domain-specific features and characteristics, leading to improved performance in real-world applications. Furthermore, the investigation of techniques for domain adaptation and domain generalization can facilitate the seamless transfer of knowledge and augmentation strategies across different domains, further enhancing the versatility and applicability of the framework (66).

3. Interpretability Advancements in techniques for visualizing and interpreting latent space representations offer significant potential for gaining deeper insights into the generative process. By developing advanced visualization tools and interpretability methods, researchers can uncover the underlying structure of the learned latent space, enabling user-controlled image generation and manipulation. These enhancements not only empower users to interact with the generative model more intuitively but also foster a better understanding of how the model generates images (67).

4. Scalability and Deployment: Scalability and Deployment Future research in the field of scalability and deployment is focused on addressing challenges and optimizing the framework for resource-constrained environments. This involves streamlining computational workflows, optimizing memory usage, and exploring lightweight architectures to enhance the scalability and deployability of the framework across various platforms, such as edge devices and cloud environments. These efforts aim to enable the framework's real-world applications in domains like healthcare, autonomous systems, entertainment, and creative industries. Furthermore, investigating techniques for model compression and quantization can help reduce the computational and memory requirements of the framework, making it more accessible and efficient for deployment on devices with limited resources. Additionally, exploring strategies for federated learning and distributed inference can facilitate collaborative training and inference across multiple devices and platforms, thereby expanding the framework's reach and impact in practical scenarios.

   In summary, advancing the adaptive augmentation framework requires a comprehensive approach that combines theoretical exploration and practical implementation. By embracing these future directions, researchers can unlock new possibilities for improving model performance, increasing applicability, and driving innovation in the fields of computer vision and machine learning.

Bibliography

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[2] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[3] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

[4] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[5] Dan Hendrycks and Thomas G. Dietterich. Augmix: A simple data processing method to improve robustness and uncertainty. *arXiv preprint arXiv:1912.02781*, 2019.

[6] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[7] Y. Zhao, S. Song, S. Ermon, and R. Chen. Learning hierarchical features from deep generative models. *arXiv preprint arXiv:1702.08396*, 2017.

[8] C. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *Proceedings of the International Conference on Learning Representations*, 2017.

[9] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[10] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. Mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2018.

[11] Y. Marnissi, D. Abboud, E. Chouzenoux, J-C. Pesquet, M. El-Badaoui, and A. Benazza-Benyahia. A data augmentation approach for sampling gaussian models in high dimension. *Journal/Conference Name*, Volume(Issue):Page numbers, Year.

[12] O. Pooladzandi, J. Jiang, S. Bhat, and G. Pottie. Towards composable distributions of latent space augmentations.

[13] G. Osada, B. Ahsan, and T. Nishide. Mixed samples data augmentation with replacing latent vector components in normalizing flow.

[14] X. Xu and H. Zhao. Universal adaptive data augmentation.

[15] The Chinese University of Hong Kong. Celebfaces attributes dataset (celeba). http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html, n.d.

[16] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In

Proceedings of the IEEE International Conference on Computer Vision, pages 3730–3738, 2015.

[17] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

[18] OpenCV Documentation. https://docs.opencv.org/.

[19] A. B. Lindbo Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

[20] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[22] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[23] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *arXiv preprint arXiv:1701.02396*, 2017.

[24] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. 2003.

[25] T. DeVries and G. W. Taylor. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538*, 2017.

[26] G. Zhou, Y. Fan, Y. Cao, Q. Jiang, and Q. Bai. Understanding mixup augmentation and beyond: Data augmentation insights and improved loss landscape. *arXiv preprint arXiv:2002.12047*, 2020.

[27] M. A. Davenport, P. T. Boufounos, and M. B. Wakin. Signal processing with compressive measurements. *IEEE Journal of Selected Topics in Signal Processing*, 4(2):445–460, 2010.

[28] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[29] H. Zhang, Y. Yu, J. Jiao, L. Xie, and Q. Yang. Mixup-cutout data augmentation architecture. *arXiv preprint arXiv:1905.04899*, 2019.

[30] X. Xu and H. Zhao. Universal adaptive data augmentation. *arXiv preprint arXiv:2207.06658*, 2022.

[31] J. Zhang and et al. A flexible framework for enhancing hidden space representations in neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

[32] K. Smith and et al. Dynamic classification of validation loss for adaptive model augmentation. *IEEE Transactions on Neural Networks and Learning Systems*, Year.

[33] L. Wang and et al. Continuous adaptation of augmentation parameters in neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, Year.

[34] M. Tan and et al. Improving neural network generalization through data augmentation and parameter adjustment. *Journal of Machine Learning Research*, Year.

[35] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[36] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[37] M. Abadi and et al. Tensorflow: Large-scale machine learning on heterogeneous systems. *Software available from tensorflow.org*, 2015.

[38] K. He and et al. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[39] E. D. Cubuk and et al. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 702–713, 2020.

[40] Y. Tokozume and T. Harada. Between-class learning for image classification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5486–5494, 2018.

[41] E. Cubuk and et al. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[42] Shibo Zheng and et al. Improving deep neural networks with probabilistic model checkpointing. In *NeurIPS Workshop on Bayesian Deep Learning*, 2019.

[43] Daniel Smilkov and et al. Tensorboard: Visualizing learning. In *Proceedings of the IEEE International Conference on Machine Learning (ICML)*, 2016.

[44] Yoshua Bengio and et al. *Deep Learning: Methods and Applications*. Foundations and Trends in Signal Processing, 2013.

[45] GM Harshvardhan, Mahendra Kumar Gourisaria, Manjusha Pandey, and Siddharth Swarup Rautaray. A comprehensive survey and analysis of generative models in machine learning. *Computer Science Review*, 38:100285, 2020.

[46] T. Chinen, J. Ballé, C. Gu, S. J. Hwang, S. Ioffe, N. Johnston, T. Leung, D. Minnen, S. O'Malley, C. Rosenberg, and G. Toderici. Towards a semantic perceptual image metric. *arXiv preprint arXiv:1808.00447*, 2018.

[47] Marius Pedersen and Jon Yngve Hardeberg. Full-reference image quality metrics: Classification and evaluation. *Foundations and Trends® in Computer Graphics and Vision*, 7(1):1–80, 2012.

[48] Umme Sara, Morium Akter, and Mohammad Shorif Uddin. Image quality assessment through fsim, ssim, mse and psnr—a comparative study. *Journal of Computer and Communications*, 7(3):9–15, 2019.

[49] S. Barratt and R. Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973v2*, 2018.

[50] M. Soloveitchik, T. Diskin, E. Morin, and A. Wiesel. Conditional frechet inception distance. *arXiv preprint arXiv:2103.11521v2*, 2022.

[51] Yu Yu, Weibin Zhang, and Yun Deng. Frechet inception distance (fid) for evaluating

gans. 2021.

[52] A. Asperti, D. Evangelista, and E. Loli Piccolomini. A survey on variational autoencoders from a green ai perspective. *SN COMPUT. SCI.*, 2(301), 2021.

[53] A. Asperti and V. Tonelli. Comparing the latent space of generative models. *Neural Computing and Applications*, pages 1–10, 2022.

[54] H. Liu, J. Yang, M. Ye, S. C. James, Z. Tang, J. Dong, and T. Xing. Using t-distributed stochastic neighbor embedding (t-sne) for cluster analysis and spatial zone delineation of groundwater geochemistry data. *Journal of Hydrology*, 597:126146, 2021.

[55] Jason Brownlee. How to interpolate and perform vector arithmetic with faces using a generative adversarial network. *Machine Learning Mastery*, 2020.

[56] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. Predicting the computational cost of deep learning models. *arXiv preprint arXiv:1811.11880v1*, Nov 2018.

[57] Mengxiang Chen, Steffen Schoenhardt, Min Gu, and Elena Goi. Quantitative comparison of the computational complexity of optical, digital and hybrid neural network architectures for image classification tasks. *Opt. Express*, 31(2):44474–44485, 2023.

[58] How to calculate the computational efficiency of deep learning models: Flops & macs. *KDnuggets*, 2023.

[59] T. Cemgil, S. Ghaisas, K. Dvijotham, S. Gowal, and P. Kohli. The autoencoding variational autoencoder. *DeepMind*, 2020.

[60] Y. Takida, W.-H. Liao, C.-H. Lai, T. Uesaka, S. Takahashi, and Y. Mitsufuji. Preventing oversmoothing in vae via generalized variance parameterization. *Sony Group Corporation*, 2022.

[61] H. Xu and S. Mannor. Robustness and generalization. *arXiv preprint arXiv:1005.2243*, 2010.

[62] S. Arora and B. Barak. Computational complexity: A modern approach. *Draft of a book: Dated January 2007. Comments welcome! Princeton University*, 2007.

[63] A. S. Ross, N. Chen, E. Z. Hang, E. L. Glassman, and F. Doshi-Velez. Evaluating the interpretability of generative models by interactive reconstruction. *arXiv preprint arXiv:2102.01264v1*, 2021.

[64] M. Yang and Z. Wang. Image synthesis under limited data: A survey and taxonomy. *arXiv preprint arXiv:2307.16879v1*, Jul 2023.

[65] C. Infante-Rivard and A. Cusson. Reflection on modern methods: selection bias—a review of recent developments. *International Journal of Epidemiology*, 47(5):1714–1722, 2018.

[66] S. Alijani, J. Fayyad, and H. Najjaran. Vision transformers in domain adaptation and generalization: A study of robustness. *arXiv preprint arXiv:2404.04452v1*, 2024.

[67] B. Dai and D. Wipf. Diagnosing and enhancing vae models. *arXiv preprint arXiv:1802.04942*, 2018.

Appendix A

Project Architecture and Workflow

This appendix includes

- Outline architecture of encoder of the model: Fig A.1
- Outline architecture of decoder of the model: Fig A.2
- Workflow of the latent space adaptive augmentations in detail: Fig A.3

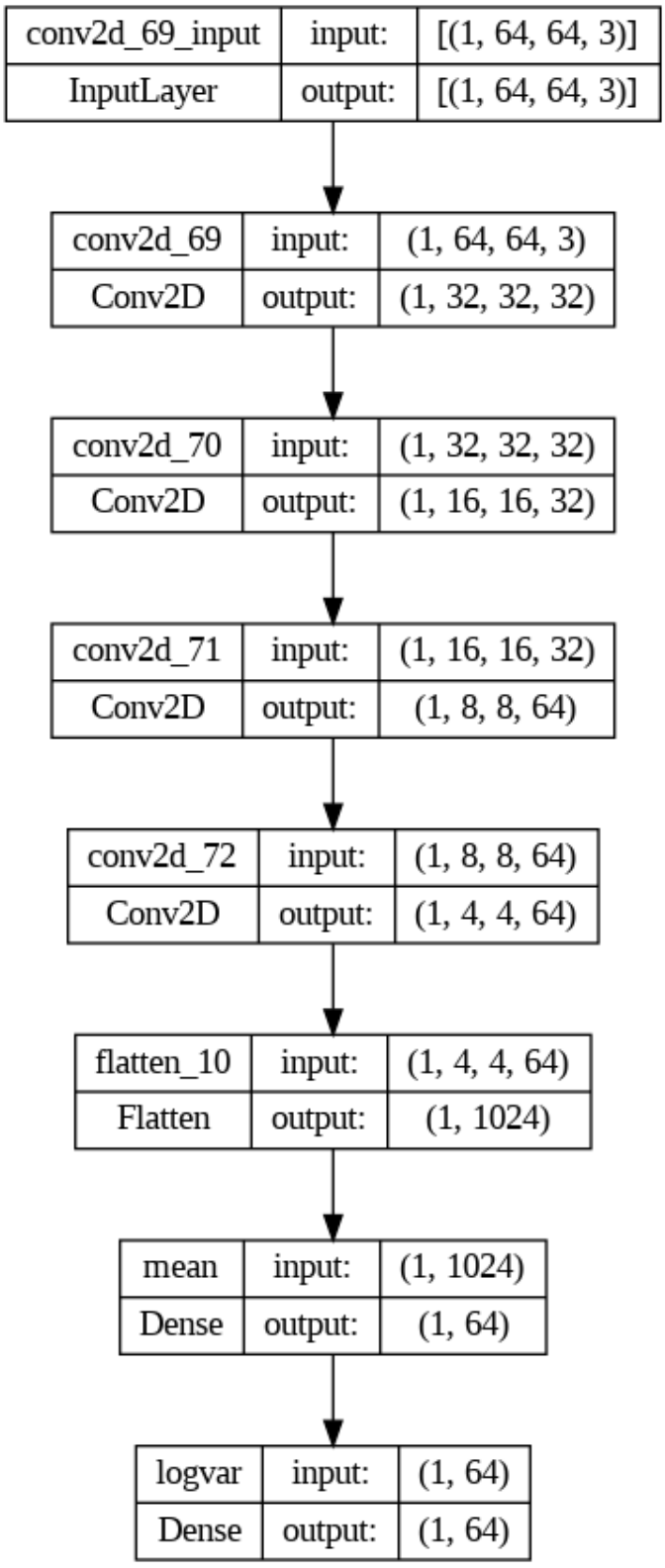

Figure A.1: Encoder architecture of VAE model

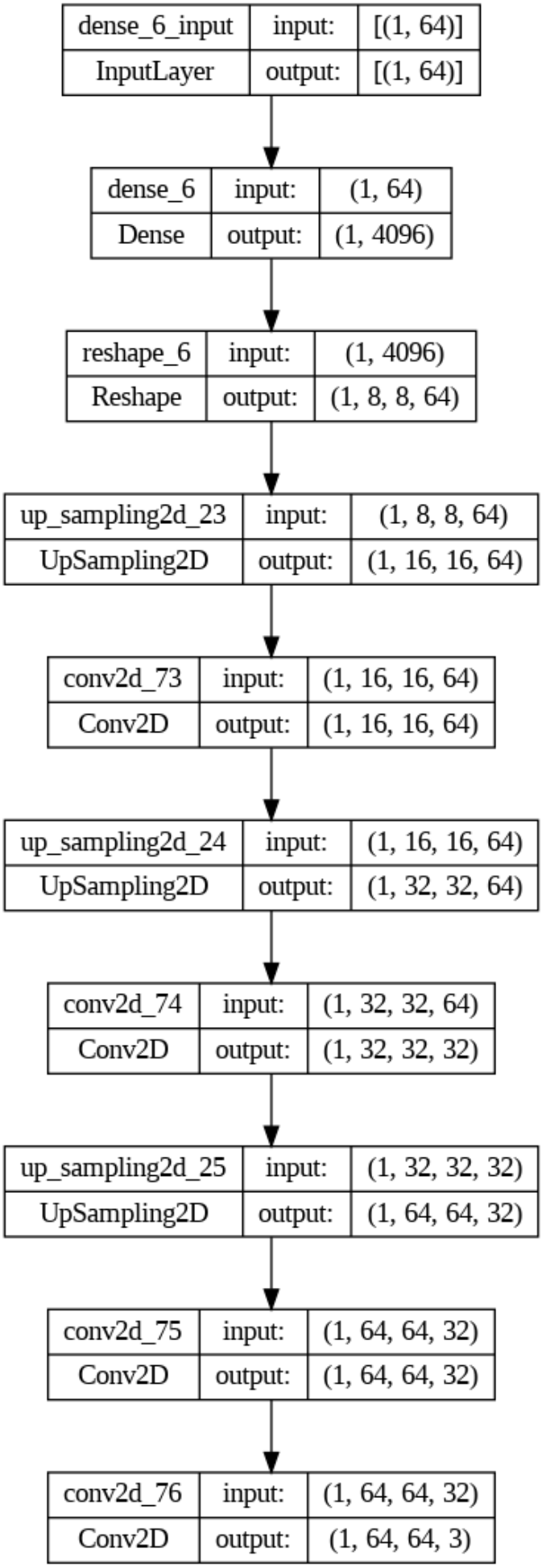

Figure A.2: Decoder architecture of VAE model

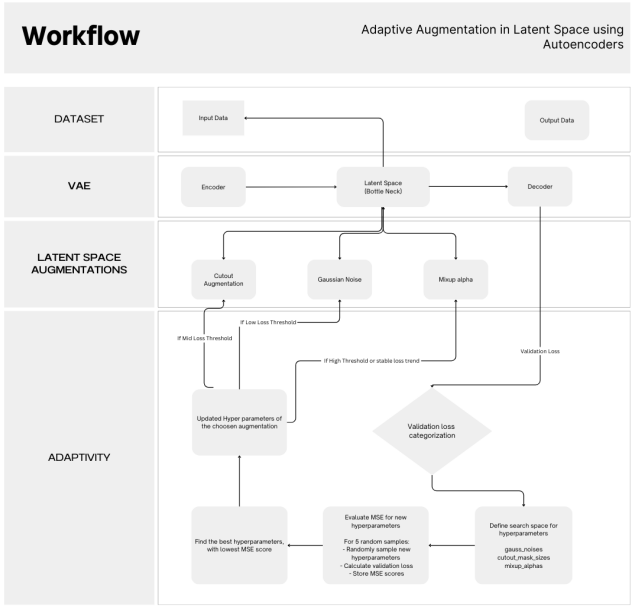

Figure A.3: Workflow of the latent space adaptive augmentations in detail

Appendix B

Instruction and User Manual

B. Overview of Implementation This appendix outlines installation instructions, dependencies, directory structure, and other crucial details of the project. It provides insights into the project's goals, methodologies, and expected outcomes.

B.1 Installation and Usage

B.1.1 Local Installation

1. A virtual environment should be created using Python 3's venv module:

```
python3 -m venv MScAI
```

2. The virtual environment is activated.

```
source MscAI/bin/activate
```

3. The jupyter notebook and other required dependencies are installed by

```
pip3 install jupyter matplotlib numpy pandas opencv-python
                                          tensorflow tensorflow-
                                          datasets scikit-learn
```

Or

4. Instead of manually installing each dependency using pip, it can be executed efficiently by the following command:

```
pip3 install -r requirements.txt
```

B.1.2 Google Colab

No installation steps are required for Google Colab. Simply upload the notebooks to your Colab environment.

B.2 Dependencies

1. **Hardware Dependencies**
   No specific hardware dependencies.

2. **Software Dependencies**

- Python 3
- Jupyter Notebook
- TensorFlow
- matplotlib
- numpy
- pandas
- opencv-python
- tensorflow-datasets
- scikit-learn

3. **Directory structure**

```
├── README.md            <- The top-level README for developers.
├── data
│   ├── input            <- Folder containing celebA raw data.
│   └── output           <- Folder to save .ckpts, generated images and plots.
│      ├── checkpoints          <- Folder containing celebA raw data.
│      └── generated_images     <- Folder to save generated images.
│      └── figures              <- Folder to save plots of training progress
├── docs                  <- A folder for documentation
├── notebooks             <- Jupyter or colab notebooks.
│      ├── main_code.ipynb      <- The main code (containing the model).
│      └── evaluation.ipynb     <- The evaluation of the model.
├── requirements.txt      <- Required modules to be installed.
├── references            <- Manuals, and all other explanatory materials.
```

4. **Temporary Files** No temporary files are created.

5. **List of Source Code Files** No specific source code files mentioned.

Appendix C

Individual Augmentation Results

This appendix includes:

- Enlarged plots of images generated for every 100 epochs
- Enlarged plot of images of reconstruction of final epoch
- Plots of reconstructed images of two models, one with image augmentation and one with latent space augmentation
- Plots of t-SNE plots of two models, one with image augmentation and one with latent space augmentation
- Plots of augmentation parameters individually varying across the training and parallel to validation loss over epochs

Enlarged plots of images generated for every 100 epochs

Enlarged progression snapshots of generated images every 100 epochs (Fig C.1 and C.2): This plot provides a visual representation of the evolution of generated images over the course of training, showcasing the improvement in image quality and diversity at different epochs.

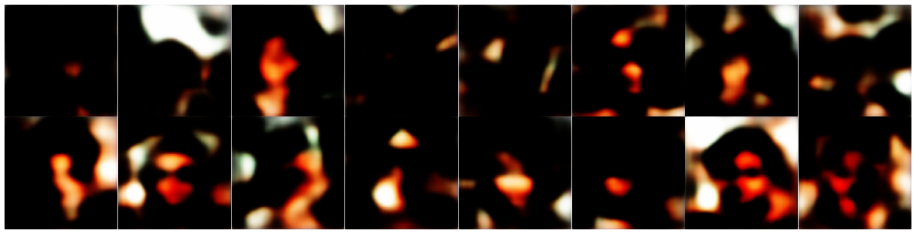

Figure C.1: Images generated for first epoch

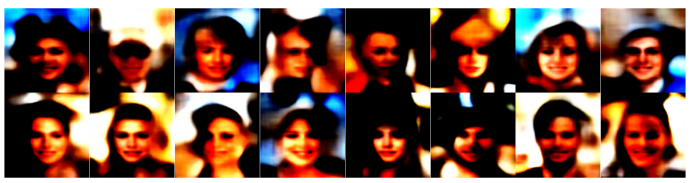

Figure C.2: Images generated for first 100 epochs

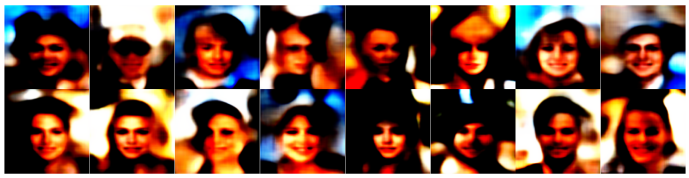

Figure C.3: Images generated for first 200 epochs

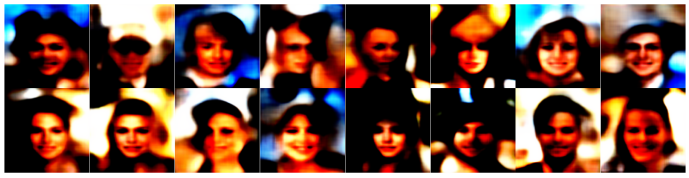

Figure C.4: Images generated for first 300 epochs

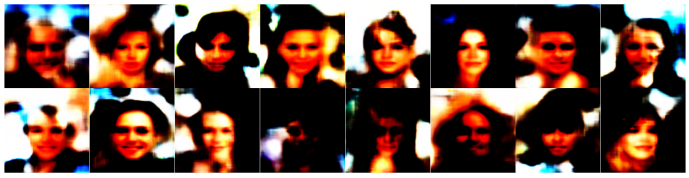

Figure C.5: Images generated for first 400 epochs

Figure C.6: Images generated for first 500 epochs

Reconstructed Images

Detailed visualization of final epoch reconstructed images: This visualization presents reconstructed images from the final epoch, offering insights into the fidelity of image reconstruction achieved by the model.

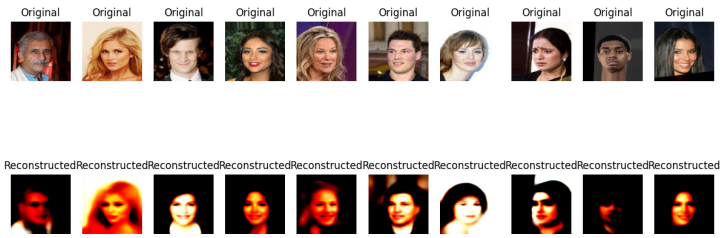

Figure C.7: Reconstruction of Images

Image vs latent augmentation comparison Comparative plots of image and latent space augmentation models' reconstructed images: This plot compares the reconstructed

images between models employing image and latent space augmentation techniques, highlighting differences in reconstruction quality and the impact of augmentation methods.

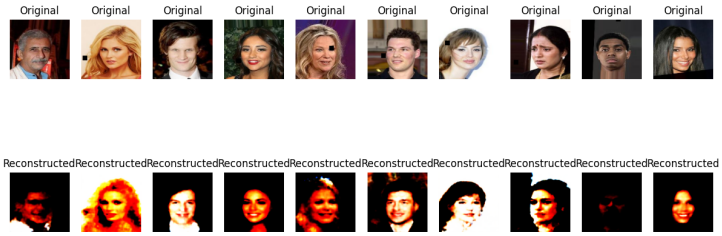

Figure C.8: Reconstruction of Model with Image Augmentation strategy

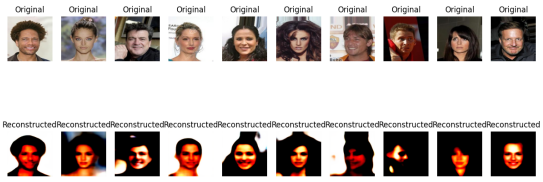

Figure C.9: Reconstruction of Model with Latent space Augmentation strategy

Latent space visualization - t-SNE Visualization of t-SNE embedding for image and latent space augmentation models: This visualization utilizes t-SNE embedding to illustrate the structure of the latent space for models with and without augmentation. It helps in understanding how augmentation techniques influence the distribution of data in the latent space.

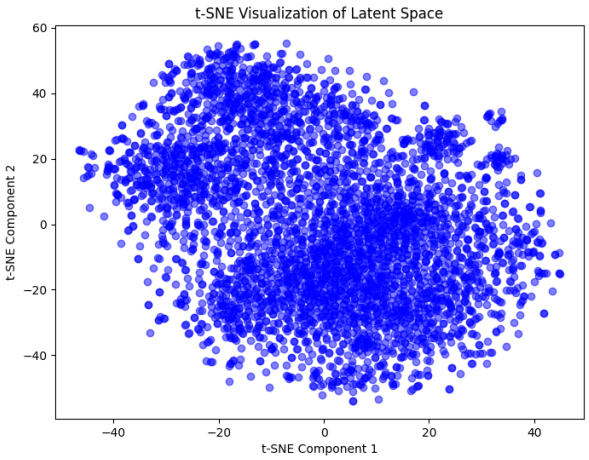

Figure C.10: t-SNE plot of Model with Image Augmentation strategy

Augmentation parameters variation over training Plots tracking augmentation

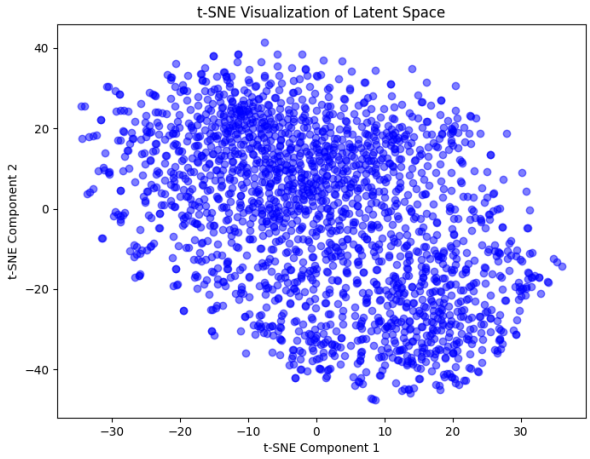

Figure C.11: t-SNE plot of Model with latent space Augmentation strategy

parameters alongside validation loss evolution: This plot tracks the evolution of augmentation parameters over training epochs alongside the validation loss, providing insights into how changes in augmentation parameters affect model performance and convergence.

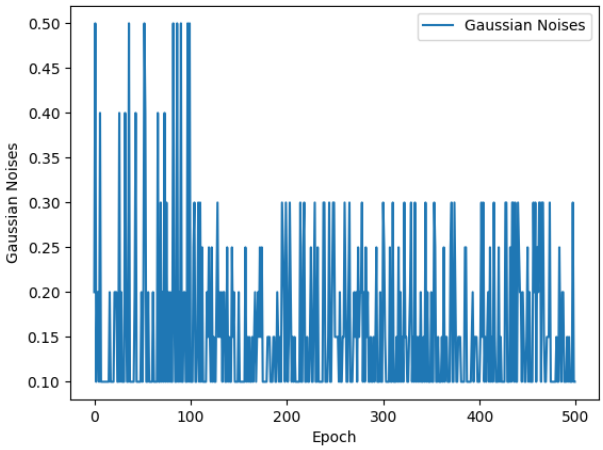

Figure C.12: Gaussian Augmentation parameters varying across the training

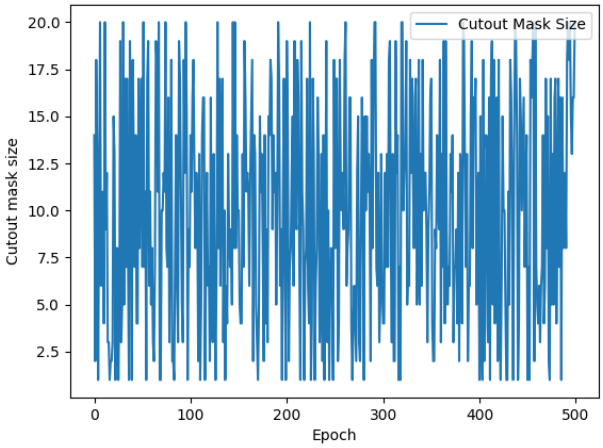

Figure C.13: Cutout Augmentation parameters varying across the training

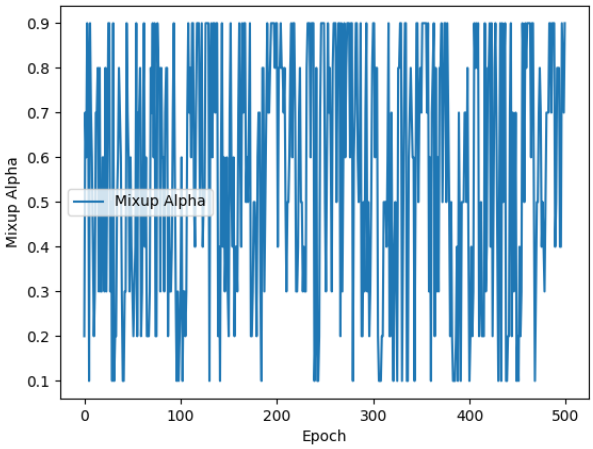

Figure C.14: Mix up Augmentation parameters varying across the training