Executive Summary of the Advanced Programming for Scientific Computing Project

Diverse Image Translation through Disentangled Gaussian Mixture Latent Spaces

Laurea Magistrale in Mathematical Engineering

Author: Fagnani Irene, Gorbani Greta

Advisor: Lara Cavinato

Co-advisor: Luca Caldera

Academic year: 2023/2024

Abstract

Image-to-Image (I2I) translation has recently gained significant attention due to its ability to learn mappings between different image domains. A key challenge in this task is generating diverse outputs for a single input image, known as the multimodality problem. To address this, we introduce a method that combines a disentangled representation with Generative Adversarial Networks (GANs). Our approach separates the image into two latent spaces: a content space that captures domain-invariant features and an attribute space that encodes domain-specific characteristics. By leveraging an attribute encoder and a multi-modal generator, our model can learn the joint distribution of images across different domains without paired training data. This disentangled representation allows us to generate a variety of outputs by manipulating the attribute code while preserving the image content. Experimental results demonstrate that our model shows promising progress in producing diverse image translations and addressing the multimodality issue. However, due to computational constraints, the full potential of our model could not be realized, as the training process was limited by the available GPU resources.

Keywords: GANs, unsupervised image-to-image translation, GMVAE, Disentangled Representation, Multimodality, Mixture of Gaussians.

1. Introduction

Image-to-Image Translation (I2I) refers to the task of learning a mapping between images from different domains without requiring paired training data. It's commonly used for tasks like turning

low-resolution images into high-resolution ones, adding color to black-and-white images, changing the style of an image (e.g. turning a photo into a painting), or even separating different elements in an image (e.g. object detection).
Recent approaches, such as Diverse Image-to-Image Translation (DRIT++) [1], achieve remarkable results by combining disentangled representations with generative adversarial networks. These methods allow for the generation of diverse outputs conditioned on a given input image, addressing the challenge of multimodality in translation tasks.
Despite these advantages, current models often rely on standard Variational Autoencoders (VAEs), that assume a uni-modal Gaussian latent space. This assumption represents a limit, when the data distribution is inherently multi-modal, as it restricts the model's ability to represent complex variations within the target domain. As a consequence, the expressiveness of the latent space may not fully capture the diversity of possible image translations.
In this work, we address this limitation by integrating a Gaussian Mixture Variational Autoencoder (GMVAE) into the I2I translation framework. Unlike a standard VAE, which models the latent space with a single Gaussian distribution, the GMVAE employs a mixture of Gaussians, providing a more flexible and structured representation. By adopting this approach, we study how the ability of the model to capture multimodal distributions changes.
The objective of this study is therefore to investigate whether replacing the standard VAE component with a GMVAE can lead to measurable improvements in image-to-image translation tasks. Specifically, we evaluate how this modification influences the quality and variability of translated images, and whether it better addresses the limitations identified in prior work.
The rest of this report is structured as follows: Section 2 provides Background information on Image-to-Image Translation and Gaussian Mixture Variational Autoencoder. Section 3 details our Contributions and the proposed model architecture. Section 4 describes the Implementation specifics. Section 5 presents our Experiments and the Results that we obtained, followed by a Discussion of the findings in Section 6. Finally, we conclude with an analysis of the model's Limitations and potential future work in Sections 7 and 8.

2. Related work

This section provides a concise overview of the foundational methods that form the basis of our work. Our project builds upon two key pieces of literature: the Diverse Image-to-Image Translation via Disentangled Representations (DRIT++) framework for image-to-image translation and the Gaussian Mixture Variational Autoencoder (GMVAE) model for unsupervised clustering. A brief introduction to these methods is essential to understand our specific contributions and the context of our experimental modifications.

2.1. DRIT++

The Diverse Image-to-Image Translation (DRIT++) framework is a powerful method for multimodal image-to-image translation without paired data. At its core, the model operates by disentangling an input image into two distinct latent representations: a domain-invariant content representation and a domain-specific attribute representation. The content representation captures the structural and geometric elements that are shared across all domains (e.g., the shape of an object or the layout of a scene). In contrast, the attribute representation encodes domain-specific characteristics such as color, texture, and style.
The model's success relies on a sophisticated architecture of encoders, a generator, and discriminators, along with several loss functions that enforce specific properties: a cross-cycle consistency loss for reconstruction, an adversarial loss for realism, and a mode-seeking regularization loss to promote diversity in the generated outputs. A key advancement of DRIT++ over its predecessor, DRIT, is its ability to generalize this disentanglement process to handle multi-domain translation tasks within a single unified framework.

2.1.1 MDMM

The DRIT++ method, as described above, is a powerful framework for multi-domain image-to-image translation. For this project, we leveraged a specific extension of this framework called Multi-Domain Multi-Modality (MDMM).
The MDMM method handles multi-domain translation tasks, generalizing the concepts of DRIT from dual-domain to multi-domain scenarios.
The process of image-to-image translation within this framework is achieved through the disentanglement of an image into a domain-invariant content representation (z^c) and a domain-specific attribute representation (z^a). The content encoder (E^c) and attribute encoder (E^a) are used to extract these representations from a given image x and its domain code z_x^d. As depicted in Figure 1, a single generator (G) then synthesizes a new image by combining a content representation from one image and an attribute representation from a different image and a target domain code. This allows for translations between multiple domains using the same generator.
To ensure the model learns to correctly handle multiple domains, the discriminator (D) is given an additional role as an auxiliary domain classifier (D_{cls}). This classifier is trained to correctly identify the domain of both real and generated images. The overall objective function is a combination of several loss terms, including adversarial losses for both the image and content spaces, a cross-cycle consistency loss for reconstruction, and a latent regression loss to ensure a consistent mapping. The introduction of the domain classification loss term is crucial for guiding the model to perform accurate multi-domain translation.

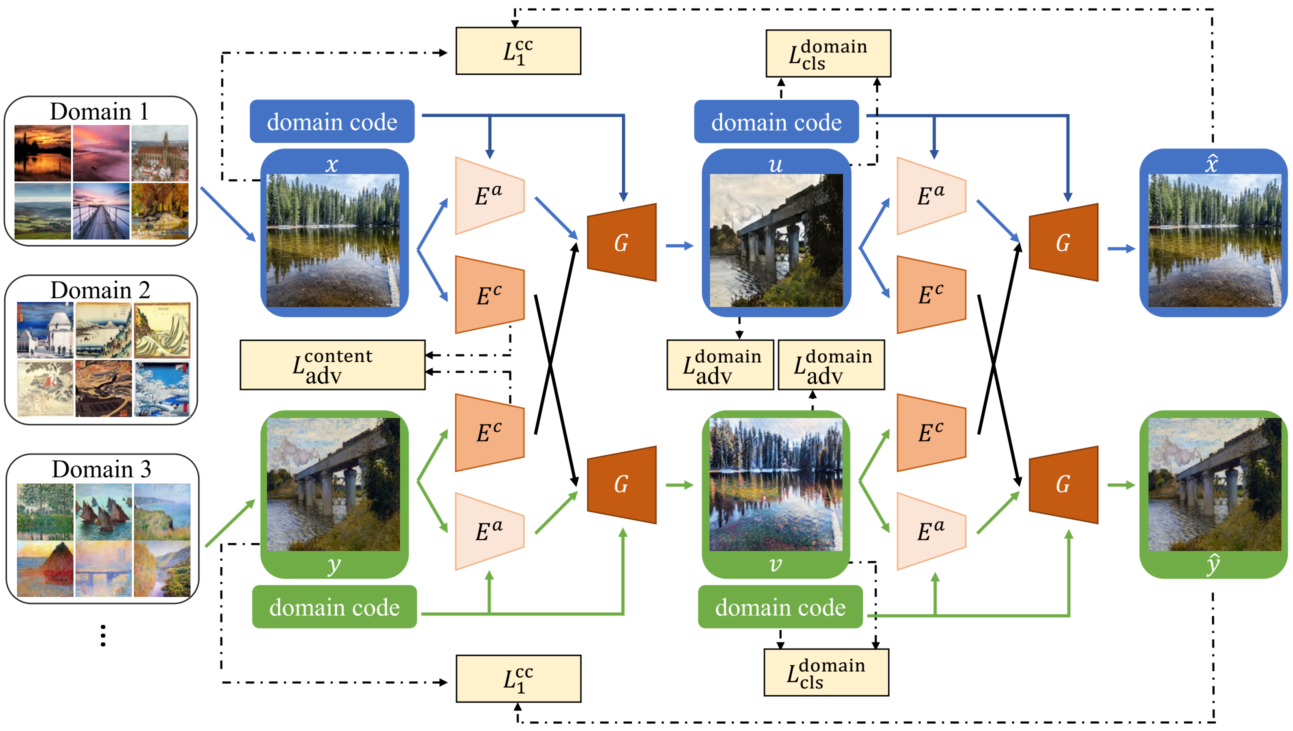

Figure 1: Method overview to learn the multi-modal mapping between several domains without paired data. We can thus generate images conditioned on attributes. Adapted from [1].

Code The code of MDMM method is taken from the github repository available at this link.
The MDMM framework is implemented in PyTorch.
The architecture of the content encoder E^c is composed of three convolutional layers followed by a series of four residual blocks.

The attribute encoder's architecture uses a CNN with four convolutional layers, which extract features that are then processed by fully connected layers.
The generator's architecture similarly includes four residual blocks, followed by three fractionally-strided convolutional layers to upsample the feature maps.
The MDMM framework employs two types of **discriminators**:

1. *Dis*: A multipurpose discriminator that evaluates image realism and performs domain classification. It uses convolutional layers to extract features and provides two outputs: a binary decision map for real/fake classification and a domain classification vector via a global pooling operation.
2. *Dis_ content*: This discriminator focuses on evaluating the consistency of the content representation across domains. It uses a network of convolutional layers with instance normalization to produce a final feature representation that ensures domain-invariant content consistency while preserving the unique characteristics of each domain.

For training, an Adam optimizer is used with the following parameters:

- Batch size = 1
- Learning rate = 0.0001
- Exponential decay rates: $(\beta_1, \beta_2) = (0.5, 0.999)$.

2.2. GMVAE

The Gaussian Mixture Variational Autoencoder (GMVAE) is a generative model that combines the capabilities of a Variational Autoencoder (VAE) with the clustering power of a Gaussian Mixture Model (GMM). This framework offers a more expressive and structured approach to learning latent representations by assuming a Gaussian mixture as the prior and latent space distributions.
The GMVAE's latent space is composed of two primary variables:

- A categorical variable y: This discrete variable represents the cluster assignment of a data point, following a categorical distribution.
- A continuous latent variable z: This variable captures fine-grained, continuous properties within each cluster and is assumed to follow a Gaussian distribution.

The framework operates in two distinct phases, as illustrated in Figure 2:

1. The Encoder Network ($q_\phi(z, y|x)$): The encoder processes the input data x to infer the posterior distributions of both latent variables. Its primary functions are:
   - *Categorical Inference*: It probabilistically assigns each data point to a cluster in the latent space. This assignment is represented by the categorical variable y, which expresses the probability that a data point belongs to each cluster. A hard cluster assignment can be performed by selecting the cluster with the highest membership probability.
   - *Continuous Inference*: It infers the latent variable z from the input x and the inferred categorical variable y. This process effectively performs dimensionality reduction while ensuring that the organization of data in the latent space is directly controlled.
2. The Decoder Network ($p_\theta(x|z, y)$): The decoder is responsible for reconstructing the original input data. It takes as input both the continuous latent variable z and the categorical variable y. The reconstruction process is therefore informed not only by the fine-grained latent features in z but also by the high-level cluster information in y. This dual input ensures that the reconstructed output aligns both with the continuous features and the discrete categorical properties of the original data.

A crucial component not explicitly detailed in the original VAE formulation is the likelihood model, represented as $q_\theta(\mathbf{y}|\mathbf{x})$. This model is used to predict the label of the cluster assignment. It represents the probability of a certain input x belonging to a specific cluster y, and its proper functioning is essential for accurate clustering.
As a result, the GMVAE simultaneously achieves dimensionality reduction and unsupervised clustering, while enabling explicit control over the organization and properties of the latent space configurations.

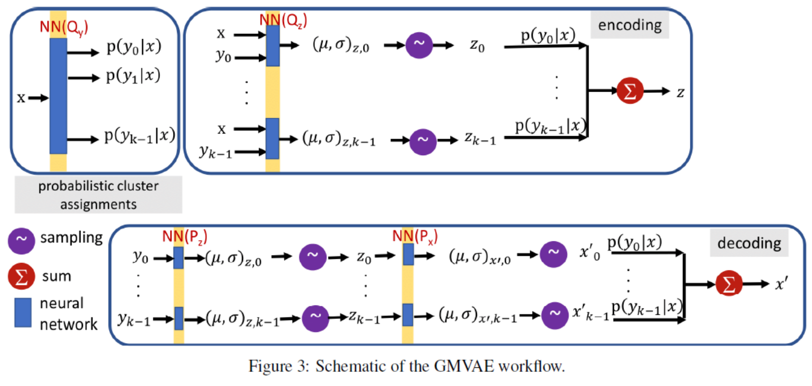

Figure 2: Scheme of a GMVAE. Adpted from [2]

2.2.1 Code

The core functions for the GMVAE, including the inference model $q_\phi(z, y|x)$ and the generative model $p_\theta(x|z, y)$, were adapted from an existing notebook and integrated into our general model. The code of these functions and a detailed discussion of this integration can be found in Section 4.

3. Contribution of this work

As stated in the previous sections, our main idea is to integrate a Gaussian Mixture Variational Autoencoder (GMVAE) into an image-to-image (I2I) translation framework, in particular within the Multi-Domain Multi-Modality (MDMM) model. The main contributions of this work can be summarized as follows:

- Integration of a GMVAE architecture within both the generator and the encoder modules.
- Design of novel loss functions tailored to the proposed framework.
- Incorporation of the generator output into the learning process as a form of pseudo-ground truth.
- Development of different training strategies to address the mismatch in learning dynamics between the original discriminator and our modified generator. In particular:
    - Investigation of the potential benefits of Adaptive Instance Normalization (AdaIN) layers.
    - Enhancement of the generator architecture by enriching the transposed convolution layers.
    - Exploration of different learning rate schedules for the encoder–generator pair and the discriminator.
- Acceleration of the training process by reducing the input image resolution from 216×216 to 108×108.

4. Implementation Details

This section provides an overview of the implementation aspects of our approach.

4.1. Proposed Architecture.

The proposed modifications target the encoder and the generator, which are the critical elements responsible for learning and sampling from the latent space. In the baseline model, these components rely on a standard VAE formulation with a unimodal Gaussian latent distribution. In contrast, our design incorporates a Gaussian Mixture Variational Autoencoder (GMVAE), enabling the use of a mixture of Gaussian components to better capture multimodal latent structures.
It is important to note that the original codebase contained two versions of the encoder and generator: one for use in an a-to-b domain translation setting, which concatenates inputs, and one for a-to-a translation without concatenation. Our architectural adjustments were applied exclusively to the

concatenated versions of both the encoder and generator, as these are the components responsible for disentangled attribute-based translation.
The following subsections describe in detail the architectural adjustments applied to both the encoder and the generator, highlighting the differences with respect to the baseline design.

4.1.1 Encoder

In our model, the encoder is split into two parts: a content encoder and an attribute encoder. We focused our modifications exclusively on the attribute encoder (`MD_E_attr_concat`), as it's the critical component for learning a structured latent representation of domain-specific characteristics or "style." By integrating the Gaussian Mixture Variational Autoencoder (GMVAE) framework solely into this part, we can learn a richer, multimodal latent space without altering the content encoder, which captures domain-invariant features.
The baseline attribute encoder consists of a sequence of convolutional blocks followed by two fully connected layers that output the mean and variance of a latent Gaussian distribution. While this design is sufficient for modeling unimodal latent spaces, it is not adequate for capturing the multimodal structure that often characterizes image-to-image translation tasks.
Our proposed architecture introduces two major modifications. First, we replace the simple latent Gaussian parameterization with two inference branches: one for estimating the categorical distribution $q(y|x)$, implemented through a Gumbel-Softmax layer, and one for estimating the continuous latent distribution $q(z|x, y)$, parameterized by a Gaussian reparameterization module. The categorical inference branch enables the model to assign each input to one of multiple mixture components, while the continuous inference branch allows sampling of latent variables conditioned on both the input features and the categorical assignment.
Second, the forward computation is restructured to jointly output both the categorical variables and the Gaussian latent variables, returning the mean, variance, logits, and probabilities alongside the reparameterized latent vectors. This results in a richer and more flexible latent representation compared to the baseline.
The key modifications are highlighted in the following pseudocode and code snippets:

1. Integration of Dual Inference Branches in `__init__`: Instead of just `fc` and `fcVar` for a single Gaussian, we introduce two distinct `ModuleLists` to handle the categorical ($q(y|x)$) and continuous ($q(z|x, y)$) latent spaces.

```python
# Modified __init__ method snippet
def __init__(self, input_dim, z_dim, y_dim,
             output_nc=8, c_dim=3, norm_layer=None, nl_layer=None):
  # ... (previous code remains unchanged)

  self.fc = nn.Sequential(*[nn.Linear(output_ndf, output_nc)])

  # New: q(y|x) branch for categorical component
  self.inference_qyx = torch.nn.ModuleList([
      nn.Linear(output_nc, 512),
      nn.ReLU(),
      nn.Linear(512, 512),
      nn.ReLU(),
      GMVAE.GumbelSoftmax(512, c_dim) # Gumbel-Softmax for categorical variable
  ])

  # New: q(z|x,y) branch for continuous component
```

```
    self.inference_qzyx = torch.nn.ModuleList([
        nn.Linear(output_nc + y_dim, 512),
        nn.ReLU(),
        nn.Linear(512, 512),
        nn.ReLU(),
        GMVAE.Gaussian(512, z_dim) # Gaussian reparameterization
    for continuous variable
    ])
```

Listing 1: Modified __init__ method snippet for the encoder

2. Restructuring the `forward` Method: The `forward` method is changed to call the new inference branches and return a dictionary containing all latent space parameters.

```
# Modified forward method snippet
def forward(self, x, c, temperature=1.0, hard=0):
    # ... (initial conv layers and feature flattening remain
  unchanged)
    x_c = torch.cat([x, c], dim=1)
    x_conv = self.conv(x_c)
    conv_flat = x_conv.view(x.size(0), -1)

    # New: Apply base FC layer and pass through inference branches
    output_features = F.softplus(self.fc(conv_flat))

    # Estimate categorical variable y
    logits, prob, y = self.qyx(output_features, temperature, hard)

    # Estimate continuous variable z, conditioned on output
  features and y
    mu, var, z = self.qzxy(output_features, y)

    # Return a dictionary with all latent parameters
    return {'mean': mu, 'var': var, 'gaussian': z,
            'logits': logits, 'prob_cat': prob, 'categorical': y}
```

Listing 2: Modified forward method

This structural change allows the model to learn a mixture of distributions where the continuous latent variable z is conditioned on the categorical latent variable y. This design provides the flexibility needed to represent the multimodal nature of I2I translation tasks.

4.1.2 Generator

The generator architecture has been substantially extended with respect to the baseline design. In the original implementation, the generator is composed of a series of residual and transposed convolutional blocks that progressively decode the concatenation of the latent vector z, the condition c, and the intermediate feature maps, ultimately producing the output image.
The generator network is composed of four sequential modules, `dec1`, `dec2`, `dec3`, and `dec4`.
`dec1` consists of a sequence of three `INSResBlock` blocks, each with input and output channels of size `tch`, where each block implements a 3x3 convolution followed by instance normalization and ReLU activation, with a residual connection to preserve features.
After `dec1`, the number of channels is incremented by `nz` and fed into `dec2`, which is a

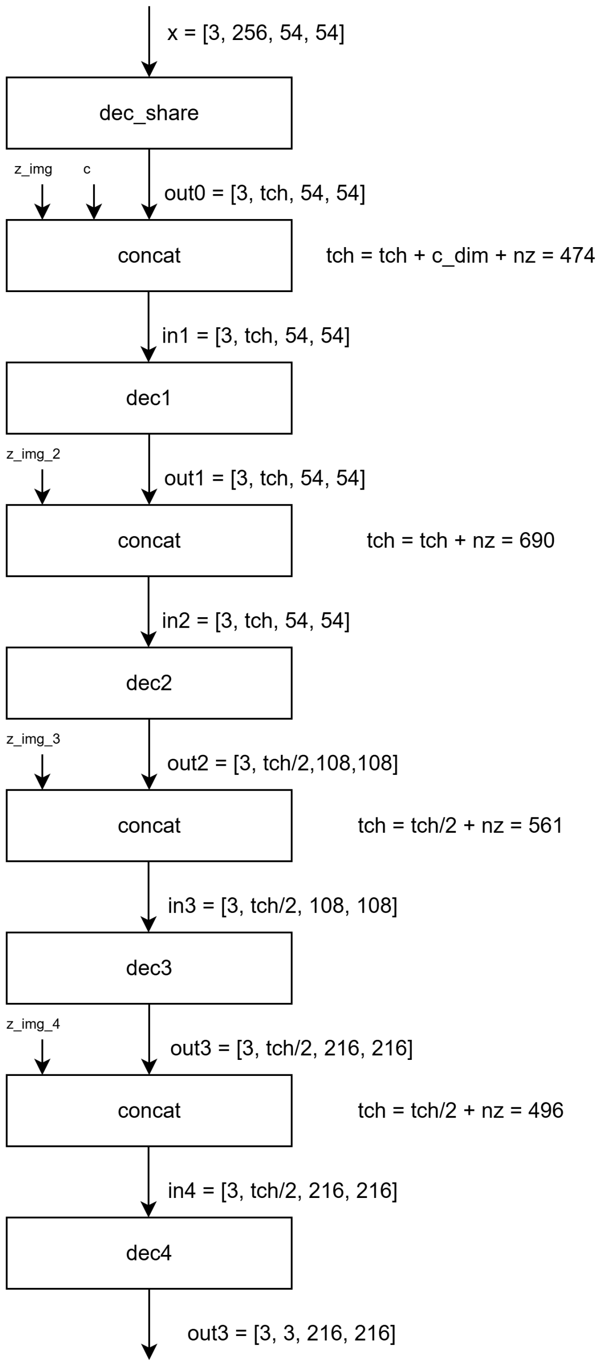

Figure 3: Schematic of the original Generator, using the concat option. Here, $nz = 216$, $c\_dim = 2$ and $tch = 256$.

`ReLUINSConvTranspose2d` block that upsamples the feature maps by a factor of 2 using a transposed convolution with kernel size 3, stride 2, and padding 1, followed by layer normalization and ReLU activation.
An optional second convolutional layer can be added in `dec2` if `double_layer_ConvT` is enabled. Similarly, `dec3` is another `ReLUINSConvTranspose2d` block that further upsamples the feature maps and reduces the number of channels by half, again optionally applying a double convolutional layer.
Finally, `dec4` consists of a single `ConvTranspose2d` layer with kernel size 1, stride 1, and padding 0, projecting the features to the desired `output_dim` and followed by a `Tanh` activation to constrain the

output values between -1 and 1.
The generator schematic is shown at Figure 3.
Our modified version introduces several key enhancements.
First, an optional Adaptive Instance Normalization (AdaIN) layer is incorporated in the shared residual block, allowing style-dependent modulation of feature statistics. The AdaIN layer performs instance normalization and scales it based on the mean and variance of a style feature map. Given content and style feature maps x and y, respectively, AdaIN performs the following transformation:

$$AdaIN(x, y) = \sigma(y) \cdot \frac{(x - \mu(x))}{\sigma(x)} + \mu(y) \tag{1}$$

Second, we extend the transposed convolutional layers by enabling a double-layer configuration, which increases the representational capacity of the upsampling process.
Third, we enrich the probabilistic modeling of the latent space by introducing two additional generative distributions: the conditional prior $p(z|y)$, parameterized by a mean and variance predicted from the categorical variable y, and the likelihood $p(x|z)$, implemented as a multi-layer perceptron that reconstructs the image from the latent representation.
Finally, the forward pass of the modified generator not only outputs the translated image but also returns the parameters of $p(z|y)$ and a reconstruction x_{rec} sampled from $p(x|z)$, thereby fully embedding the GMVAE formulation into the decoding stage. These modifications significantly increase the flexibility of the generator, enabling multimodal generation while providing a tighter integration between the latent structure and the image space.
The key modifications to the generator architecture, `MD_G_multi_concat`, are highlighted in the following code snippets. We've primarily focused on integrating the GMVAE's generative process, including the prior distributions for both the categorical and continuous latent variables, as well as an optional AdaIN layer.
In the `__init__` method, we introduce two new generative branches: `self.y_mu` and `self.y_var` to model the prior $p(z|y)$, and `self.generative_pxz` to model the reconstruction likelihood $p(x|z)$.

```python
# Modified __init__ method snippet
def __init__(self, output_dim, x_dim, z_dim, crop_size,
             c_dim=3, nz=8, use_adain=False, double_ConvT=False):
    super(MD_G_multi_concat, self).__init__()
    self.nz = nz
    self.c_dim = c_dim
    self.use_adain = use_adain
    self.double_ConvT = double_ConvT

    # Optional AdaIN layer for disentanglement
    if self.use_adain:
        self.style_dim = 2
        self.adain1 = AdaIN(tch, self.style_dim)
        tch = 256 + self.nz + self.c_dim
    else:
        tch = 256

    # ... (rest of the original network layers)

    # New: p(z|y) branch for continuous latent variable prior
    self.y_mu = nn.Linear(c_dim, z_dim)
    self.y_var = nn.Linear(c_dim, z_dim)

    # New: p(x|z) branch for reconstruction
    self.generative_pxz = torch.nn.ModuleList([
        nn.Linear(z_dim, 512),
        nn.ReLU(),
        nn.Linear(512, 512),
        nn.ReLU(),
        nn.Linear(512, 108),
        torch.nn.Sigmoid()
    ])
```

Listing 3: Modified __init__ method snippet

The `forward` method is updated to handle these new components. It now computes the prior distribution parameters and the reconstruction from the `generative_pxz` branch before returning a dictionary that contains all the relevant outputs for the training loop.

```
# Modified forward method snippet
def forward(self, x, c, temperature=1.0, hard=0):
    # ... (initial conv layers and feature flattening remain unchanged)
    x_c = torch.cat([x, c], dim=1)
    x_conv = self.conv(x_c)
    conv_flat = x_conv.view(x.size(0), -1)

    # New: Apply base FC layer and pass through inference branches
    output_features = F.softplus(self.fc(conv_flat))

    # Estimate categorical variable y
    logits, prob, y = self.qyx(output_features, temperature, hard)

    # Estimate continuous variable z, conditioned on output features
   and y
    mu, var, z = self.qzxy(output_features, y)

    # Return a dictionary with all latent parameters
    return {'mean': mu, 'var': var, 'gaussian': z,
            'logits': logits, 'prob_cat': prob, 'categorical': y}
```

Listing 4: Modified forward method

4.2. Losses implementations.

The definition of a proper loss function exerts a substantial influence on the model and the manner in which parameters are updated. It is an essential component of the model that must be modified when the model or code changes.
Given our modification of the generator model, it is necessary to introduce and account for losses in the generator.
The initial loss that was modified pertains to the training of the encoder and the generator in unison, designated as backward_EG. This loss is responsible for the training of the generator using the encoded inputs and not random latents. Specifically, following the encoding of an image, the generator produces a subsequent image based on the encoded image, with the objective of achieving maximum similarity with the original image that was provided as the input.
The contribution was the development of a KL-divergence loss function in the attribute space. This loss function is designed to be consistent with the new generator integrated with the GMVAE. In standard variational autoencoders, the prior is usually a standard normal distribution. In contrast,

our prior is defined by a mixture of Gaussians contingent on the discrete latent variable y representing a class or domain.
In the original implementation, the KL divergence between the approximate posterior $q(z|x)$ and the standard Gaussian prior $p(z) = \mathcal{N}(0, I)$ was computed using a closed-form expression:

$$\mathcal{L}_{\mathrm{KL}}^{\mathrm{standard}} = -\frac{1}{2}\sum_{i=1}^{d}\left(1 + \log\sigma_i^2 - \mu_i^2 - \sigma_i^2\right),$$

where μ and $\log\sigma^2$ represent the mean and log-variance of the encoder's output. This formulation, although efficient, assumes a fixed, isotropic Gaussian prior, limiting its capacity to model complex, multi-modal latent distributions.
To accommodate the integration of a Gaussian Mixture Variational Autoencoder (GMVAE) within the generator, we replace the above with a more general form of the KL divergence that allows class-conditional priors. Specifically, we compute the divergence between $q(z|x) = \mathcal{N}(\mu_q, \sigma_q^2)$ and a learned, domain-dependent prior $p(z|y) = \mathcal{N}(\mu_y, \sigma_y^2)$:

$$\mathcal{L}_{\mathrm{KL}}^{\mathrm{GMVAE}} = \mathbb{E}_{q(z|x)}\left[\log\frac{q(z|x)}{p(z|y)}\right] =$$
$$= \log\mathcal{N}(z; \mu_q, \sigma_q^2) - \log\mathcal{N}(z; \mu_y, \sigma_y^2),$$

where μ_y and σ_y^2 are specific to each domain or class y, as learned by the generative prior. This formulation enables the latent space to capture semantically meaningful and disentangled variations aligned with the class labels, making it more suitable for our generator, which conditions sampling and reconstruction on structured latent priors. Consequently, the substitution enhances the model's expressiveness and alignment with the GMVAE architecture.
The second loss we implemented is related to the training of the generator alone. Here, the adversarial loss term loss_G_GAN2 measures how much the generator can fool the discriminator, therefore is important to adapt it to our new model.
The original approach involves iterating over multiple discriminator outputs, where each output $D_i(\tilde{x})$ corresponding to the fake image $\tilde{x}$ is passed through a sigmoid activation to produce a probability of being classified as real. This is expressed as

$$\text{loss\_G\_GAN2} = \sum_{i=1}^{N}\mathrm{BCE}\big(\sigma(D_i(\tilde{x})), \mathbf{1}\big),$$

where BCE denotes binary cross-entropy, σ is the sigmoid function, and the target label is a tensor of ones, reflecting the generator's objective to fool the discriminator.
For the model that uses the GMVAE in the generator, the loss can be formulated using a label similarity loss, which directly compares predicted categorical distributions $\hat{y}$ from the discriminator to the true class labels y using a cross-entropy or soft cross-entropy function:

$$\text{loss\_G\_GAN2} = \mathrm{CE}(\hat{y}, y),$$

where CE is the cross-entropy loss appropriate for either integer class labels or one-hot encodings. This method shifts focus from binary real/fake discrimination to encouraging semantic consistency in the generated images.
In our model, which integrates a Gaussian Mixture Variational Autoencoder (GMVAE) within the generator, the label similarity loss approach is adopted for loss_G_GAN2. This choice aligns with the GMVAE's probabilistic latent structure, enabling more effective supervision of the generator's output through categorical latent variables and improving the semantic alignment between generated images and target classes.

4.3. Refinement of Attribute and Label Handling in the Generator

A significant refinement in our model's data flow involved a crucial change in how the generator was conditioned during the training process. Initially, the generator was directly provided with `c_org`, representing the attribute label associated with the input image. While this approach served as a starting point, it bypassed the very inference capabilities that our integrated encoder, based on the GMVAE's $q(y|x)$ component, was designed to learn.
We realized that for the model to truly learn to disentangle and re-synthesize images based on inferred attributes, the generator should be conditioned not on the ground truth 'c_org', but rather on the categorical label 'y' that is produced by the encoder's inference mechanism. This 'y' represents the model's own understanding of the input image's attributes, inferred through the $q(y|x)$ pathway.
By passing the encoder's output 'y' to the generator as its conditioning label, we establish a closed-loop learning process. The encoder is trained to infer meaningful attributes, and the generator is simultaneously trained to generate images based on these inferred attributes. This ensures that the entire system learns a more cohesive and disentangled representation, as the generator is forced to become robust to the variations and nuances present in the encoder's learned attribute space, rather than relying solely on perfect ground truth information. This modification is critical for developing a truly generative model capable of attribute manipulation.
Following this modification, together with the new loss integration, we observed a notable improvement in the quality of the generated images.

4.4. Two-Time Scale Update Rule

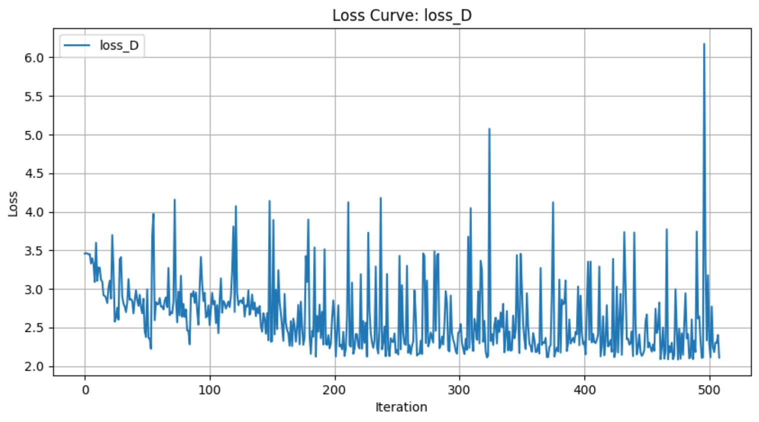

Figure 4: Discriminator loss over 5000 iterations during the first epoch, using cropped images 108×108. The values fluctuate around 2–2.5, with several noticeable peaks.

As shown in Figures 4 and 5, the learning dynamics between the Discriminator and the Generator are unbalanced. Specifically, although the Discriminator loss exhibits occasional peaks, its overall values remain very small compared to the Generator loss, which, while decreasing, exhibits much larger magnitudes. Overall, the two losses do not reach equilibrium, resulting in an imbalance during the training process. In other words, the Generator is unable to fully fool the Discriminator, and consequently, the training of the Generative Adversarial Network is not fully effective. To mitigate imbalance between losses without altering the architectures themselves, we employed the two-timescale update rule [3], which allows the use of distinct learning rates for the discriminator and the generator/encoder. Specifically, we explored two strategies: reducing the discriminator learning rate by half and doubling the learning rate of the generator and encoder, thereby accelerating or decelerating the learning dynamics in different parts of the network. The baseline learning rate for both the discriminator and generator/encoder was set to $lr = 0.0001$.

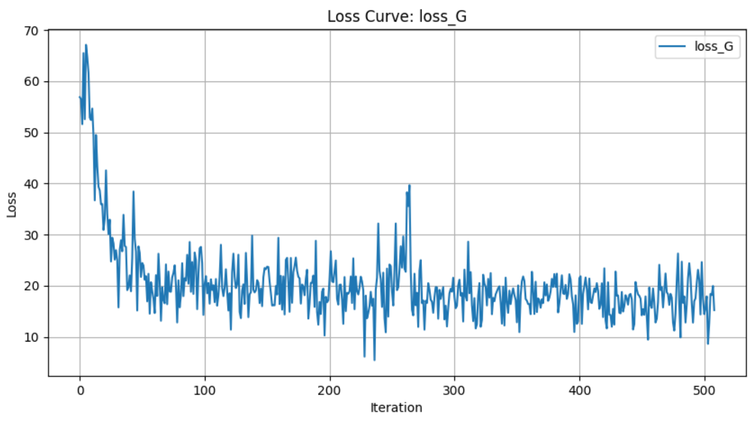

Figure 5: Generator loss over 5000 iterations during the first epoch, using cropped images 108×108. The loss initially starts at very high values and then decays, stabilizing around 20, which is still relatively high. A generator loss of this magnitude suggests that a more complex architecture may be required to match the performance of the original work (without the autoencoder with a Gaussian mixture) and to compensate for the introduction of this additional network component.

4.5. Challenges faced.

Since our approach integrates the GMVAE into the generator without modifying the discriminator, it is necessary to adopt strategies that ensure balanced learning between these two components. In Generative Adversarial Networks (GANs), a key factor for stable training is maintaining equilibrium between the discriminator loss ($\mathcal{L}_D$) and the generator loss ($\mathcal{L}_G$). In our experiments, we observed that the generator loss remained nearly constant, indicating that the generator was not learning sufficiently to reduce its error, whereas the discriminator loss consistently decreased over iterations. This behavior indicates an imbalance between the generator and the discriminator, with the discriminator becoming overly dominant due to its more structured and complex architecture compared to the modified generator.
To address this imbalance, we employed two complementary strategies. The first focuses on enhancing the capacity of the encoder-generator pair to effectively challenge the discriminator. Specifically, we increased the complexity of the transposed convolutional layers by doubling their number and incorporated Adaptive Instance Normalization (AdaIN) into the generator, as discussed in Section 4.1.2. The second strategy targets the training dynamics: we applied the Two-Time Scale Update rule, described in Section 4.4, which allows the learning rates of the discriminator and generator/encoder to be adjusted independently, thereby regulating their respective learning velocities.
Another challenge encountered during training was the computational cost associated with high-resolution images. To mitigate this limitation, we reduced the total number of training iterations and decreased the image resolution from 216×216 to 108×108, ensuring feasible training times without compromising model convergence.

5. Experiments and Results

In this section, we discuss the results of the various experiments that we have conducted, detailing both our initial findings and the subsequent improvements. It is important to note that all our experiments on the modified model were incomplete. Due to the computational demands of our architecture, the training process consistently exceeded the 4-hour GPU usage limit on the Google Colab environment, preventing us from completing the training for any of our full experimental runs. Therefore, quantitative evaluation metrics such as FID, LPIPS, JSD, and NDB could not be obtained due to the

insufficient number of generated images. All computational tasks were performed on an NVIDIA Tesla T4 GPU within the Google Colab environment.
The visual output of our model is systematically presented in a structured format to facilitate analysis. The program's output consists of a table with two rows and five columns. Each row displays the results for an image randomly selected from the test dataset. From left to right, the columns show the following: the original image, an intermediate images representing the transition phase of the transformation process, the transformed image in the central column, an intermediate images representing the transition phase of the reconstruction process and finally, the reconstructed image. This layout provides a clear qualitative overview of the model's generative and reconstructive capabilities.

5.1. Encoder and Generator Integration

Our initial experiment utilize the dataset mini containing 10 images each of horses and zebras.
A key modification for this experiment is the integration of our custom GMVAE-based encoder and generator into the original framework.
The model is configured with the following parameters:

- Learning rate for both discriminator and generator/encoder: 0.0001
- Batch size: 2
- Number of iterations: 5000
- Image resolution: 216x216 pixels

The initial outputs, as illustrated in Figure 6, demonstrate significant visual deficiencies. The generated images exhibit pronounced vertical artifacts, a general lack of coherent structure, and poor detail. This indicates that while the model is technically operational following architectural integration, its learning objective is not adequately guiding the generative process toward producing high-fidelity images.

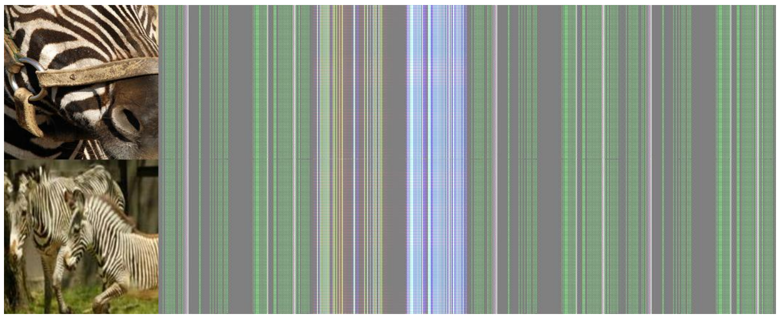

Figure 6: Example of an initial low-quality image generated by the model after architectural integration.

This critical analysis lead us to conclude that the primary bottleneck is not in the architectural connectivity or dimensional consistency, but rather in the optimization objective itself. Therefore, the subsequent phase of our project focuses on a thorough re-evaluation and modification of the loss functions. We identify the necessity to introduce and refine specific loss terms that would more effectively penalize undesirable image characteristics and promote the generation of visually realistic and contextually appropriate outputs. This fundamental shift in strategy is deemed essential to improve the generative capabilities of our integrated model.

5.2. Loss integration

Our initial experiments with the mini dataset reveal that the horse-to-zebra translation task is not optimally suited for our model's initial development, as the inherent morphological differences between the domains proved difficult to manage. Consequently, we transition to the more semantically constrained apple2orange dataset, comprising 266 images each of apples and oranges.
This second experiment is defined by a crucial modification to the optimization objective. Specifically, we refine the backward_EG loss and KL divergence loss, we implement a novel label similarity loss.

These modifications are designed to more effectively guide the generator by leveraging the encoder's learned attributes.
The model is configured with the following parameters:

- Learning rate for both discriminator and generator/encoder: 0.0001
- Batch size: 2
- Number of iterations: 5000
- Image resolution: 216x216 pixels

An example of the generated results after this change is presented in Figure 7. The images, particularly the initial stages of generation, begin to exhibit more discernible shapes and color patterns that resemble the target objects (oranges and apples in this instance), as opposed to the severe artifacts seen previously. While the generated images still appear blurry and lack fine details, and a checkerboard pattern remains visible, the overall structure and color coherence are considerably improved. This indicates that conditioning the generator on the encoder's learned attributes has guided the model towards generating more semantically meaningful and visually interpretable outputs, affirming the importance of this architectural refinement.

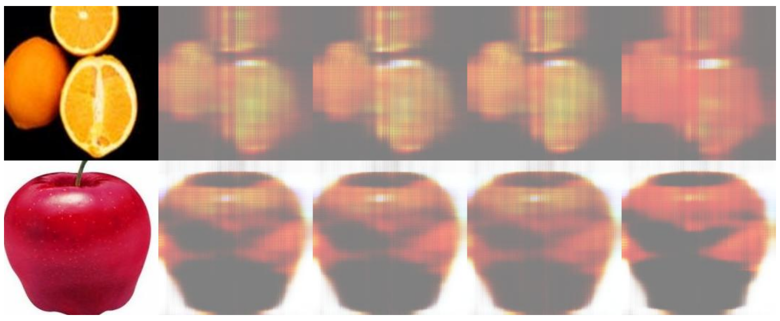

Figure 7: Generated images after conditioning the generator on encoder's inferred labels (5000 iterations). Top row: oranges. Bottom row: apples. Note the improved structural and color coherence compared to previous results.

5.3. Addressing the Discriminator-Generator Imbalance

To address the imbalance between the discriminator and the generator, where the discriminator's more complex architecture led to a dominant training signal, we explored three distinct methodological adjustments.

5.3.1 AdaIN

Our first approach involves the integration of an Adaptive Instance Normalization (AdaIN) layer within the generator. This layer is intended to normalize the feature maps of the generator based on the mean and variance of a style feature, thereby providing finer control over the output style.
The model is configured with the following parameters:

- Learning rate for both discriminator and generator/encoder: 0.0001
- Batch size: 2
- Number of iterations: 4500
- Image resolution: 216x216 pixels

As shown in Figure 8, the generated images exhibit pronounced contrast. The model appeared to successfully transform oranges into apples, as visible in the bottom row of the grid. However, the inverse transformation from apples to oranges does not appear to be successful.

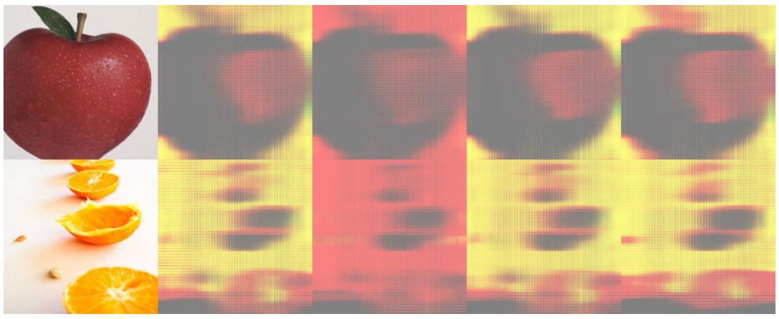

Figure 8: Images generated after integrating an AdaIN layer within the generator.

5.3.2 Transposed Convolutional Layer

Our second approach involves adding transposed convolutional layers to each block of the generator, aiming to enhance the feature upsampling process.
The model is trained with the following parameters:

- Learning rate for both discriminator and generator/encoder: 0.0001
- Batch size: 2
- Number of iterations: 4000
- Image resolution: 216x216 pixels

The results are inconsistent. In Figure 9, the generated images show increase contrast and a darker appearance. The bottom row, representing a transformation from oranges, shows only a slight hint of red in the central transition images, and the overall transformation is limited. Conversely, as depicted in Figure 10, the model struggles with the inverse transformation, with only a slight hint of yellow/orange visible in the central images of the apple transformation. The initial apple image appears to be entirely lost in the process.

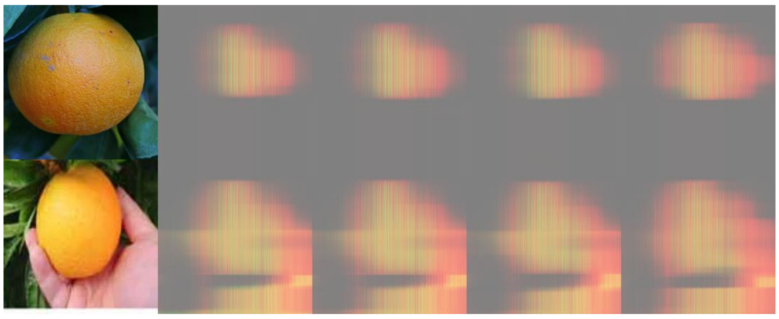

Figure 9: Images generated with 4,000 iterations after adding a transposed convolutional layer to the generator.

5.3.3 Two-Time Scale Update Rule

Finally, we implement the Two-Time Scale Update Rule (TT-SUR) [3], using two distinct learning rates to balance the adversarial training process.

Halving the Discriminator's Learning Rate In this configuration, we reduce the discriminator's learning rate to half that of the generator.

- Learning rate for discriminator: 0.00005
- Learning rate for generator/encoder: 0.0001
- Batch size: 2
- Number of iterations: 5500

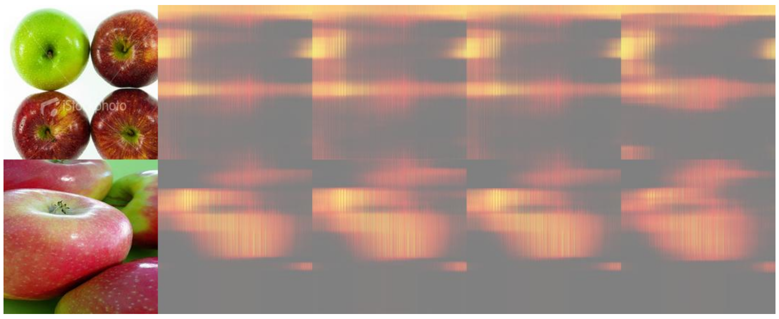

Figure 10: Another example of an image generated after adding a transposed convolutional layer.

- Image resolution: 216x216 pixels

Figure 11 shows that the model successfully transforms oranges, with a noticeable increase in redness in the central images. In Figure 12, the model demonstrates a clearer transformation of an apple to a more pronounced orange hue in the central images.

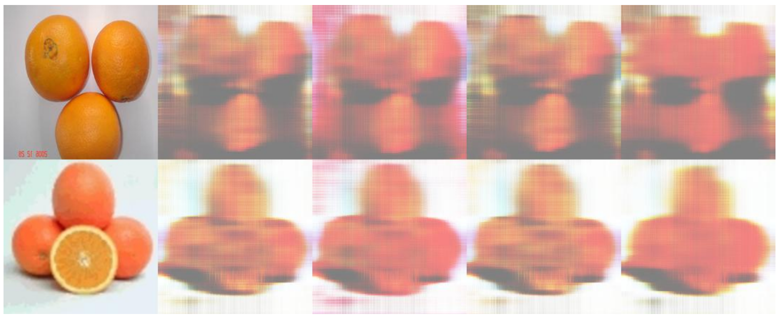

Figure 11: Generated images with a discriminator learning rate of 0.00005 and a generator/encoder learning rate of 0.0001.

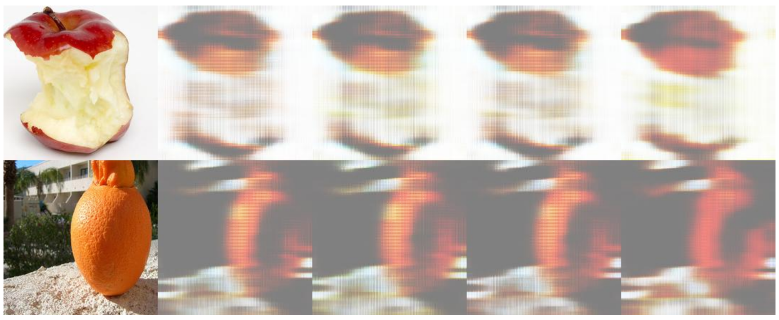

Figure 12: Another example of an image generated with a halved discriminator learning rate.

Doubling the Generator's Learning Rate We then experimented by doubling the generator's learning rate.

- Learning rate for discriminator: 0.0001
- Learning rate for generator/encoder: 0.0002
- Batch size: 2
- Number of iterations: 5500
- Image resolution: 216x216 pixels

As illustrated in Figure 13, this adjustment improved the transformation of oranges into apples. However, the model still struggled with the inverse transformation, suggesting a persistent asymmetry in the learning process.

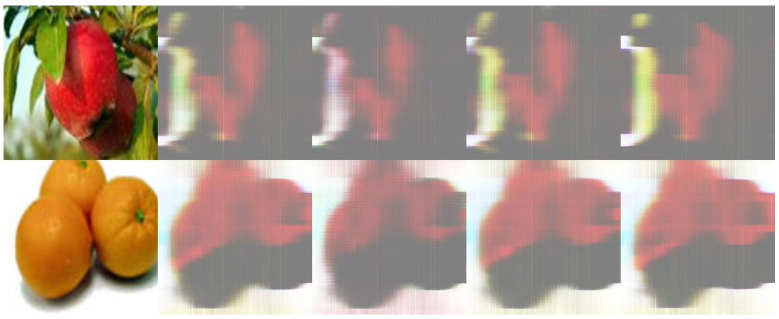

Figure 13: Image generated with a discriminator learning rate of 0.0001 and a generator/encoder learning rate of 0.0002.

5.3.4 The best strategy

Based on the experimental results, the most effective strategy for mitigating the observed imbalance between the discriminator and generator was the implementation of the Two-Time Scale Update Rule (TT-SUR) with a halved learning rate for the discriminator.
While other approaches, such as AdaIN and the addition of transposed convolutional layers, yielded inconsistent or asymmetric results, reducing the discriminator's learning rate provided the most stable and balanced training dynamics. This adjustment allowed the generator to achieve more meaningful progress, leading to visually improved and more symmetric transformations between the two domains. This finding confirms that a careful calibration of the adversarial learning rates is a critical factor in achieving effective and robust image-to-image translation within this framework.

5.4. Reduced resolution

In our final experiment, we adopt a reduced image resolution to overcome the computational limitations imposed by the available GPU memory. This methodological trade-off allows for a significantly extended training duration, enabling the model to learn more complex relationships.

- Learning rate for discriminator: 0.00005
- Learning rate for generator/encoder: 0.0001
- Batch size: 2
- Number of iterations: 10000
- Image resolution: 108x108 pixels

As shown in Figure 14, the results from this extended training period demonstrate a successful manipulation of core attributes. We observed a clear progression in the color transformation of the apple, where its hue shifts toward orange before being progressively darkened back to a distinct red. This demonstrates that the model is successfully learning to manipulate and control fundamental attributes, a crucial step towards effective image-to-image translation. This compromise, sacrificing spatial resolution for a higher number of training iterations, proved essential for achieving a meaningful learning signal and validating our approach within the given computational constraints.

5.5. Comparison with the original model

To benchmark our proposed GMVAE-based model, we conducted a direct comparison with the original baseline architecture under identical training conditions. Both models were trained with the following hyperparameters:

- Learning rate for both discriminator and generator/encoder: 0.0001

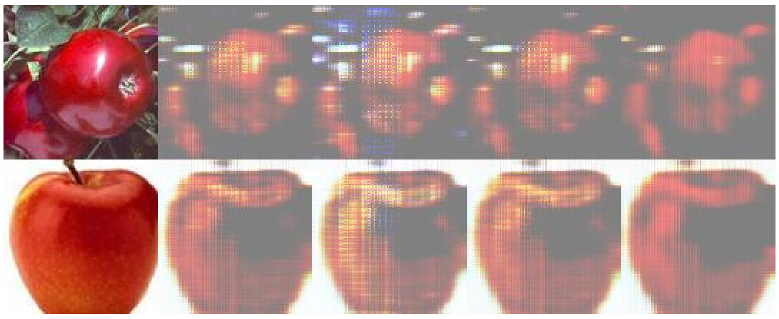

Figure 14: Generated images after 10,000 iterations with reduced resolution.

- Batch size: 2
- Number of iterations: 6000
- Image resolution: 216x216 pixels

During this initial evaluation, the original model demonstrated superior performance in terms of image quality and visual fidelity at 6000 iterations. The images generated by the baseline model, as shown in Figure 15, exhibit cleaner, more realistic translations. Specifically, the model effectively applies the desired attribute transformations (e.g., darkening the orange and lightening the apple) with a high degree of clarity and detail.

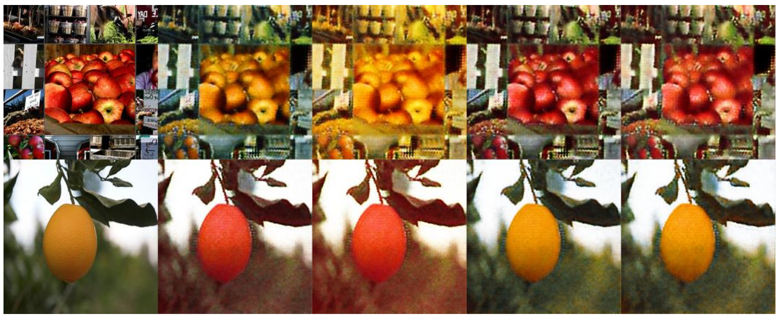

Figure 15: Image-to-image translation results of the original model after 6000 iterations. The translation applies a filter to darken the oranges and another to lighten the apples.

In contrast, our GMVAE-based model produced less refined results at a similar number of iterations. This observation suggests that while our architecture is theoretically more expressive due to its ability to model multimodal distributions, its increased complexity leads to a slower convergence rate. The additional complexity of the GMVAE's latent space, which involves training both categorical and continuous variables, appears to require a longer training schedule to fully leverage its expressive capabilities.
This was confirmed in our final experiment, where our model's performance significantly improved at 10,000 iterations with a reduced image resolution, approaching the visual quality of the baseline model. This indicates that our GMVAE-based architecture is inherently slower to train but is capable of achieving comparable visual quality given a longer training schedule.

5.6. Time Results

The table 1 clearly illustrates the computational trade-offs associated with different architectural modifications. As expected, training time is highly dependent on image resolution, with all full-size image experiments taking significantly longer than their cropped counterparts. For instance, the original MDMM model's average iteration time increases from 0.313 seconds for cropped images to 0.615 seconds for full-size images. Similarly, the more complex GMVAE models experience a similar scale-up in computational time.

A key observation is the higher average running time of all GMVAE models compared to the baseline MDMM model, indicating that the added complexity of the GMVAE's encoder-generator architectures, designed to capture more complex latent distributions, results in a slower performance.
In this context, our primary model choice was the model with halved discriminator's learning rate (*GMVAE half discr* [5.3.3]), which shows an average running time of 0.551 seconds for cropped images and 2.326 seconds for full-size images. While this is not the fastest among the GMVAE variants, its performance is very close to that of the baseline *GMVAE* model (0.538 seconds and 2.261 seconds). We selected this variant because, for a minimal increase in training time, it proved capable of producing visually superior images. This is attributed to the optimization of the discriminator's learning rate, which facilitates more stable training and leads to higher-quality outputs.

Table 1: Average running time per iteration over the first 50 iterations for different experiments

Image Resolution	**Model**	**Average Running Time (sec)**
Cropped Images (108 × 108)	GMVAE [5.2]	0.538
Cropped Images (108 × 108)	GMVAE double enc gen [5.3.3]	0.544
Cropped Images (108 × 108)	GMVAE half discr [5.3.3]	0.551
Cropped Images (108 × 108)	GMVAE AdaIN [5.3.1]	0.647
Cropped Images (108 × 108)	GMVAE double conv [5.3.2]	0.771
Cropped Images (108 × 108)	MDMM [5.5]	0.313
Full Size Images (216 × 216)	GMVAE [5.2]	2.261
Full Size Images (216 × 216)	GMVAE double enc gen [5.3.3]	2.307
Full Size Images (216 × 216)	GMVAE half discr [5.3.3]	2.326
Full Size Images (216 × 216)	GMVAE AdaIN [5.3.1]	2.339
Full Size Images (216 × 216)	GMVAE double conv [5.3.2]	3.716
Full Size Images (216 × 216)	MDMM [5.5]	0.615

6. Discussion

The integration of a GMVAE into an image-to-image translation framework introduces a significant computational overhead compared to the original implementation with a classical VAE, resulting in longer training times. Despite this, the generator is still occasionally fooled by the discriminator, and the adversarial learning remains partially unbalanced. When training with a reduced number of iterations, our GMVAE integration does not yield noticeable improvements in image quality, suggesting that a higher number of iterations may be required to fully realize the potential benefits of the approach. The introduction of a task-specific loss function, combined with conditioning the generator on the categorical variable y predicted by the encoder rather than the ground truth c_{org}, proved to be a meaningful enhancement, improving the alignment between latent representations and generated outputs.
To address the imbalance inherent in adversarial training, we evaluated multiple strategies. Increasing the architectural complexity of the generator—for instance, through the incorporation of Adaptive Instance Normalization (AdaIN) layers or additional transposed convolutional layers—did not lead to consistent performance gains and in some cases degraded image quality, producing darker images or collapsed transformations. In contrast, adjusting the learning rates according to the Two-Time Scale Update rule produced modest but tangible improvements. Specifically, halving the discriminator's learning rate allowed the generator to better preserve semantic content, while doubling the generator's learning rate improved certain domain translations, such as orange-to-apple transformations.
Overall, these results indicate that balancing adversarial dynamics is critical for successful training, and that careful control of learning rates can be more effective than increasing architectural complexity alone.

7. Limitations and Future Work

As previously mentioned, this project was developed under significant computational constraints. Lacking access to a dedicated machine with a powerful GPU, our work was carried out entirely on the Google Colab platform. While this service provides free access to GPU acceleration, it comes with inherent memory and runtime limitations that directly impacted our training process.

The primary bottleneck in our experimental setup is the GPU's memory and runtime limits, which prevent us from executing the full 50,000 iterations specified in the original codebase. This constraint means that we were never able to complete the entire training process, as we could only perform a limited number of iterations per epoch. This limitation significantly impacted the model's ability to reach optimal convergence and fully exploit its learning potential.

To mitigate this challenge and allow for more extensive training, we implemented a strategy of image resolution reduction. This compromise, while sacrificing some visual fidelity, significantly lowered the computational cost per iteration. As a result, we were able to increase the number of training cycles to a maximum of 10,000 iterations, a process which took approximately 4 hours to complete. This duration represents the maximum continuous GPU usage time allowed by the Google Colab environment.

Future work will focus on addressing the limitations identified in this study. The most critical area for improvement is scaling the model to higher resolutions. This can be achieved through a combination of the following strategies:

- Accessing More Powerful Hardware: Utilizing a dedicated machine with a high-memory GPU (e.g., NVIDIA A100) would remove the current runtime and memory constraints. This would allow for training at the full resolution (216x216 pixels or higher) and for the full 50,000 iterations specified in the original codebase.
- Architectural Optimizations: Implementing more memory-efficient architectures, such as progressive growing of GANs or using techniques like gradient checkpointing, could significantly reduce memory usage without compromising resolution.

8. Conclusion

In this work, we successfully integrated a Gaussian Mixture Variational Autoencoder (GMVAE) into a disentangled image-to-image translation framework. This involved significant modifications to the original encoder and generator architectures to accommodate the GMVAE's mixture of Gaussians, which is designed to capture more complex, multimodal latent distributions.

Our integration, however, introduced several challenges. The increased complexity of the VAE components led to differing training dynamics between the discriminator and the generator-encoder. Furthermore, the higher computational cost forced us to train at lower image resolutions, which likely impacted the overall visual quality of the final output. To mitigate these issues, we introduced two new loss functions: a modified KL-divergence loss that aligns with the GMVAE's mixture of Gaussians prior and a label similarity loss for the generator to enforce better domain alignment.

The results of our experiments are promising and show that our model has the potential to produce diverse and high-quality image translations. However, the increased complexity of the GMVAE makes our model inherently slower to train compared to the original baseline. Due to the significant computational demands and the time constraints of our experimental setup, we were unable to fully explore the model's capabilities over a prolonged training period. This suggests that while the proposed modifications are a step in the right direction, a more thorough exploration is needed to fully realize the benefits of the GMVAE architecture and achieve results that surpass the baseline.

References

[1] Hsin-Ying Lee et al. *Diverse Image-to-Image Translation via Disentangled Representations*. 2018. arXiv: 1808.00948 [cs.CV]. URL: https://arxiv.org/abs/1808.00948.

[2] Nat Dilokthanakul et al. *Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders*. 2017. arXiv: `1611.02648 [cs.LG]`. URL: `https://arxiv.org/abs/1611.02648`.

[3] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: `1706.08500 [cs.LG]`. URL: `https://arxiv.org/abs/1706.08500`.