POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

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 present a method that unites disentangled representations with GANs, integrating a mixture of gaussian to model the latent space. 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. While our current implementation demonstrates promising results, further improvements are needed to fully unlock the model's potential. We believe that with access to greater computational resources, the training process could be extended to produce more refined and robust image translations.

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), and inpainting/image completion, which involves filling in missing or corrupted parts of an image to make it whole.
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) [2] into the Disentangled Representation Image to Image Translation (DRIT) 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 to guide the reader through our work in a logical progression.
First, Section 2 provides the necessary background on Image-to-Image Translation and Gaussian Mixture Variational Autoencoders. Following this, Section 3 details our specific contributions and the proposed model architecture. The implementation details are then described in Section 4.
We present our experiments in Section 5 and show the corresponding results in Section 6. Finally, Section 7 presents our conclusions, including an analysis of the model's limitations and potential future work.

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: DRIT++ framework for image-to-image translation and 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.
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) is used to extract domain-invariant features from a given image x, while the attribute encoder (E^a) extracts domain-specific characteristics from the image and its corresponding 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.

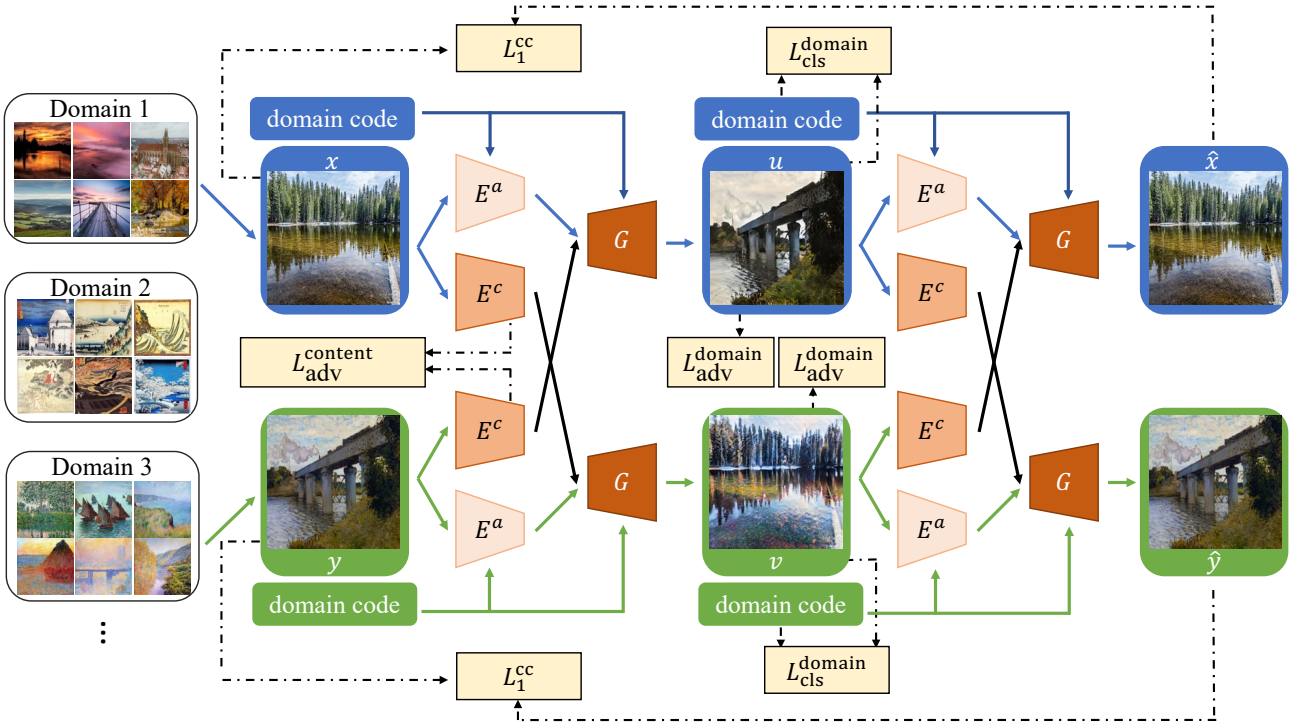

Figure 1: Method overview to learn the multi-modal mapping between several domains without paired data. We can thus generate images conditioned on attributes. Adapted from [1].

Code The code of Multi-Domain Multi-Modality (MDMM) method is taken from the official GitHub repository [3], which 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, developed in two different classes:

1. 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. A Discriminator that 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.

2.2. GMVAE

The Gaussian Mixture Variational Autoencoder (GMVAE) is a generative model that extends the standard Variational Autoencoder (VAE) by replacing its single Gaussian prior with a Gaussian Mixture Model (GMM) prior, which makes the approximation of data distribution more flexible.
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.

   These components don't appear in the original VAE formulation.
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.

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.

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 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.

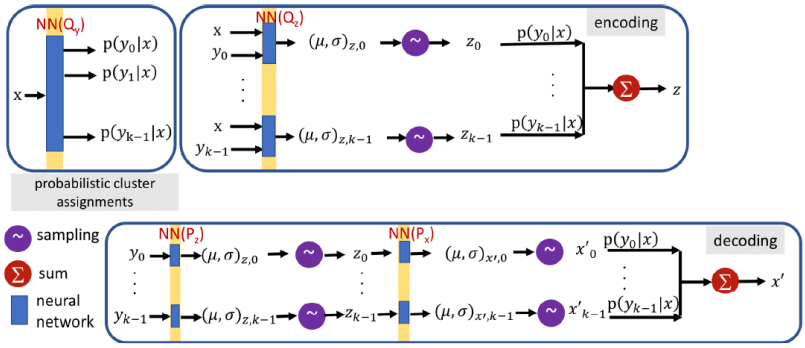

Figure 3: Schematic of the GMVAE workflow.

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

- 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 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, we use two separate encoders: a content encoder to learn domain-invariant features and an attribute encoder to capture domain-specific characteristics. 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 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.

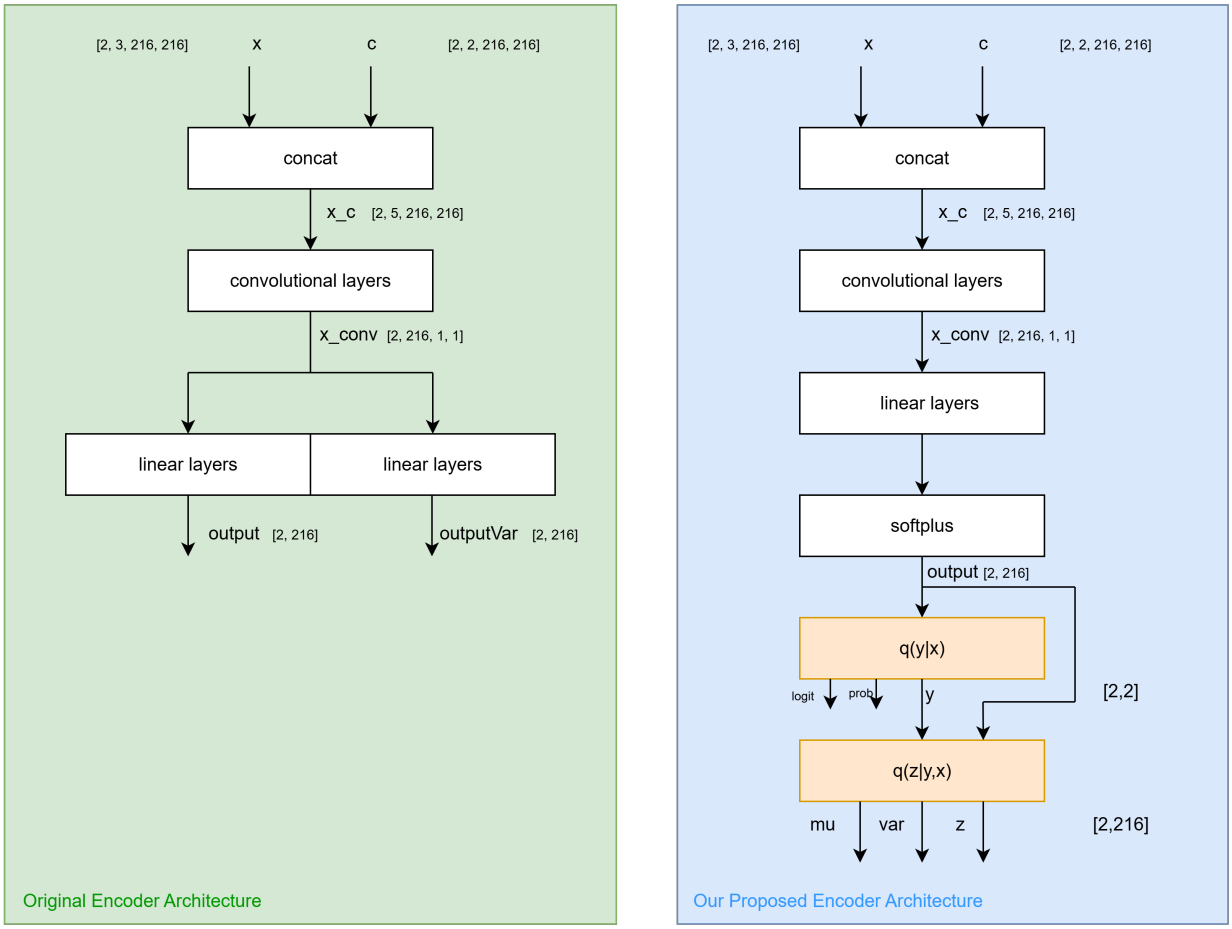

Figure 3: Comparison between the original attributed encoder and the one proposed by us. Further details about the implementation of $q(y|x)$ and $q(z|x,y)$ at paragraph [4.1.1].

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. To clarify the changes, we first provide a simplified pseudocode of the encoder structure *before* and *after* our modifications, so that the role of each added branch becomes evident.

Integration of Dual Inference Branches in `_init_`: In the original code, only two linear layers (`fc` and `fcVar`) were used to parameterize a single Gaussian latent variable. In contrast, the

new design introduces two distinct inference branches: one for the categorical latent variable $q(y|x)$ and one for the continuous latent variable $q(z|x,y)$.

Original pseudocode:

```
    class Encoder:
        def _init_():
            conv_layers =
   build_conv()
            self.fc = Linear(...)
            self.fcVar = Linear
   (...)

```

Modified pseudocode:

```
    class Encoder:
        def _init_():
            conv_layers =
   build_conv()
            self.fc = Linear(...)
            # New categorical
   branch q(y|x)
            self.inference_qyx = [
   Linear, ReLU, ...,
   GumbelSoftmax]
            # New continuous
   branch q(z|x,y)
            self.inference_qzyx =
   [Linear, ReLU, ..., Gaussian]

```

```
    # 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):

        # ... (convolutional backbone 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)
        ])

        # 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)
        ])

```

Listing 1: Modified _init_ method snippet for the encoder

Restructuring the `forward` Method: In the original implementation, the forward pass directly produced the parameters of a single Gaussian distribution. In the modified version, the forward pass first infers the categorical latent variable y via the $q(y|x)$ branch, then conditions the continuous latent

variable z on both x and y via the $q(z|x, y)$ branch.

Original pseudocode:

```
    def forward(x, c):
        features = conv(x, c)
        mu = self.fc(features)
        var = self.fcVar(features)
        z = sample_gaussian(mu, var)
        return {mu, var, z}

```

Modified pseudocode:

```
    def forward(x, c):
        features = conv(x, c)
        output = self.fc(features)
        logits, prob, y = self.qyx(output)
        mu, var, z = self.qzxy(output, y)
        return {mu, var, z, logits, prob, y}

```

```
    # 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 explicitly conditioned on the categorical latent variable y. Such a design provides the flexibility needed to represent the multimodal nature of I2I translation tasks.
To highlight the architectural differences, we present a summary comparison of the original encoder implementation and our modified version in Table 1.

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

Table 1: Comparison between the original and modified encoder implementations.

Original Implementation	**Modified Implementation**
Single Gaussian latent space with parameters μ and σ^2, obtained from two linear layers (`fc`, `fcVar`).	Two distinct latent branches: • $q(y\|x)$ categorical branch with Gumbel-Softmax. • $q(z\|x,y)$ continuous branch with Gaussian reparametrization.
Forward pass computes: • $\mu = \texttt{fc}(features)$ • $\sigma^2 = \texttt{fcVar}(features)$ • $z \sim \mathcal{N}(\mu, \sigma^2)$	Forward pass computes: • logits, prob, $y \sim q(y\|x)$ • $\mu, \sigma^2, z \sim q(z\|x,y)$ • Returns full dictionary: $\{\mu, \sigma^2, z, logits, prob, y\}$
Latent representation limited to a unimodal Gaussian distribution.	Latent representation modeled as a mixture: categorical y captures multimodality, continuous z refines within each mode.

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 `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 4.
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.
The following code snippets highlight the key architectural changes to the GM-DRIT generator, `MD_G_multi_concat`. Our primary focus was on extending the original model with the GMVAE's

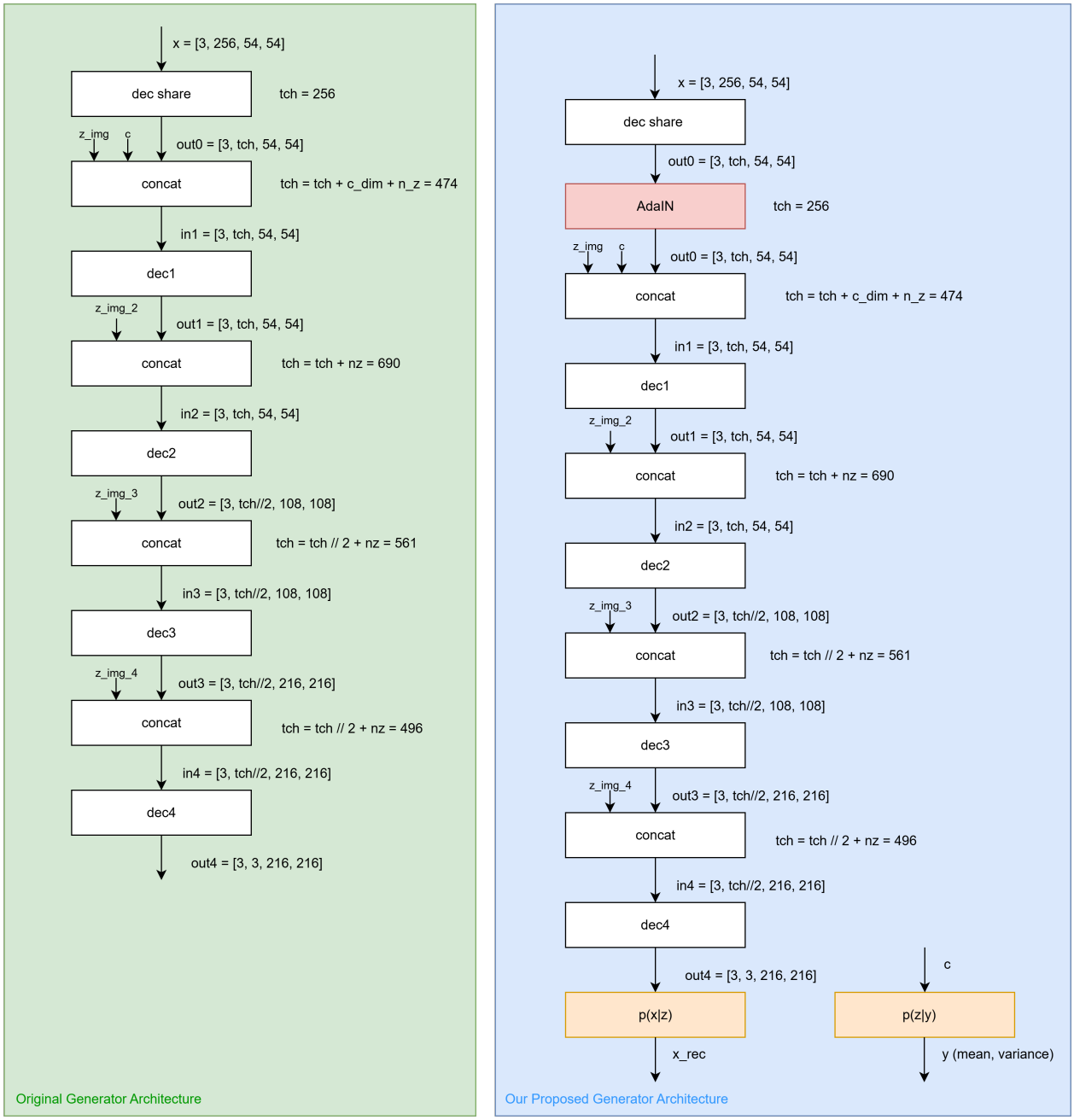

Figure 4: Architecture comparison, between the original generator and the one proposed in this work, using the concat option. Here, $n_z = 216$, $tch = 256$ and $c_{dim} = 2$. Detailed description about the implementation of $p(x|z)$ and $p(z|y)$ can be found at paragraph [4.1.2].

generative process, which now explicitly models the prior distributions, $p(z|y)$ and $p(x|z)$. The integration also includes an optional AdaIN layer, as detailed in the comparison below.

Modified `_init_` method. We add two new linear layers (`self.y_mu`, `self.y_var`) to parameterize the Gaussian prior $p(z|y)$, and a multilayer perceptron (`self.generative_pxz`) to model the likelihood $p(x|z)$.

Original pseudocode:

```
class MD_G_multi_concat:
    def _init_():
        self.dec_share = ResBlocks
    (...)
        self.dec1, self.dec2, self
    .dec3, self.dec4 = ...
```

Modified pseudocode:

```
class MD_G_multi_concat:
    def _init_():
        self.dec_share = ResBlocks
    (...)
        self.dec1, self.dec2, self
    .dec3, self.dec4 = ...
        # New priors
        self.y_mu  = Linear(c_dim,
     z_dim)
        self.y_var = Linear(c_dim,
     z_dim)
        # New likelihood branch
        self.generative_pxz = [
    Linear -> ReLU -> ... ->
    Sigmoid]
```

```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

    # shared and decoding layers (unchanged)
    self.dec_share = nn.Sequential(INSResBlock(256, 256))
    # ... (dec1, dec2, dec3, dec4 as in original)

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

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

Listing 3: Modified _init_ method snippet for the generator

Modified `forward` method. The forward pass is extended: besides decoding the image as in the original model, it now also computes the prior distribution $p(z|y)$ and the reconstruction likelihood $p(x|z)$.

Original pseudocode:

```
    def forward(x, z, c):
        out0 = self.dec_share(x)
        out1 = self.dec1(concat(out0, z, c))
        out2 = self.dec2(concat(out1, z))
        out3 = self.dec3(concat(out2, z))
        out4 = self.dec4(concat(out3, z))
        return out4
```

Modified pseudocode:

```
    def forward(x, z, c, y):
        c0 = c
        out0 = self.dec_share(x)
        if use_adain:
            out0 = adain(out0, c)
        # decode with skip concatenations
        out1 = self.dec1(concat(out0, z, c))
        out2 = self.dec2(concat(out1, z))
        out3 = self.dec3(concat(out2, z))
        out4 = self.dec4(concat(out3, z))
        # compute prior p(z|y)
        y_mu, y_var = self.pzy(c0)
        # compute likelihood p(x|z)
        x_rec = self.pxz(out4)
        return { 'y_mean': y_mu, 'y_var': y_var, 'x_rec': x_rec }
```

```
# Modified forward method snippet
def forward(self, x, z, c, y):
    c0 = c
    out0 = self.dec_share(x)
    if self.use_adain:
        out0 = self.adain1(out0, c)

    z_img = z.view(z.size(0), z.size(1), 1, 1)
    z_img = z_img.expand(z.size(0), z.size(1), x.size(2), x.size(3))
    c = c.view(c.size(0), c.size(1), 1, 1)
    c = c.repeat(1, 1, out0.size(2), out0.size(3))
    x_c_z = torch.cat([out0, c, z_img], 1)
    out1 = self.dec1(x_c_z)
    z_img2 = z.view(z.size(0), z.size(1), 1, 1).expand(z.size(0), z.size(1), out1.size(2), out1.size(3))
    x_and_z2 = torch.cat([out1, z_img2], 1)
    out2 = self.dec2(x_and_z2)
    z_img3 = z.view(z.size(0), z.size(1), 1, 1).expand(z.size(0), z.size(1), out2.size(2), out2.size(3))
    x_and_z3 = torch.cat([out2, z_img3], 1)
    out3 = self.dec3(x_and_z3)
    z_img4 = z.view(z.size(0), z.size(1), 1, 1).expand(z.size(0), z.size(1), out3.size(2), out3.size(3))
    x_and_z4 = torch.cat([out3, z_img4], 1)
    out4 = self.dec4(x_and_z4)

    # New: compute prior p(z|y)
```

```
    y_mu, y_var = self.pzy(c0)

    # New: compute likelihood p(x|z)
    x_rec = self.pxz(out4)

    output = {'y_mean': y_mu, 'y_var': y_var, 'x_rec': x_rec}
    return output
```

Listing 4: Modified forward method for the generator

This modification allows the generator not only to decode x from (z, c), but also to explicitly parameterize the conditional prior $p(z|y)$ and the reconstruction likelihood $p(x|z)$, thereby aligning with the variational formulation of the model.
Table 2 provides an overview of our main architectural contributions by comparing the original generator with our modified implementation.

Table 2: Comparison between the original and modified generator implementations.

Original Implementation	**Modified Implementation**
Decoder architecture composed of shared residual blocks and four decoding stages (`dec_share`, `dec1-4`).	Same backbone (`dec_share`, `dec1-4`) is preserved. Additional probabilistic branches are introduced.
Forward pass: decodes x directly from (z, c) through successive skip-concatenations.	Forward pass: • Decodes intermediate output $out4$ as in original. • Computes prior $p(z\|y)$ via `y_mu`, `y_var`. • Computes likelihood $p(x\|z)$ using `pxz(out4)`.
No explicit probabilistic modeling of latent variables.	Probabilistic branches added: • $p(z\|y)$ models latent prior (conditioned on y). • $p(x\|z)$ models reconstruction likelihood from generator output.
Returns only the decoded image.	Returns a dictionary with: • y_mean, y_var (parameters of $p(z\|y)$), • x_rec (from $p(x\|z)$ branch).

4.2. Losses implementations

Modifying a deep learning model, especially its architecture, often requires a careful re-evaluation of its loss functions to ensure they align with the new design. In our case, integrating a GMVAE into the generator necessitates two key changes to the loss framework. First, we need to adapt the Kullback-Leibler (KL) divergence loss to accommodate the GMVAE's structured latent space, which moves beyond the simple Gaussian prior. Second, we must revise the adversarial loss to not only encourage realistic image generation but also promote semantic consistency and proper class representation within

the outputs. These modifications are crucial for ensuring the model effectively learns and leverages the GMVAE's disentangled representation.
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:

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

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 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 $p(z|y) = \mathcal{N}(\mu_y, \sigma_y^2)$:

$$\mathcal{L}_{\text{KL}}^{\text{GMVAE}} = \mathbb{E}_{q(z|x)} \left[\log \frac{q(z|x)}{p(z|y)}\right] =$$

$$= \log \mathcal{N}(z; \mu_q, \sigma_q^2) - \log \mathcal{N}(z; \mu_y, \sigma_y^2),$$

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 $\mathcal{L}_G^{GAN2}$ 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

$$\mathcal{L}_G^{GAN2} = \sum_{i=1}^{N} \text{BCE}\big(\sigma(D_i(\tilde{x})), \mathbf{1}\big),$$

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 our GM-DRIT model 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:

$$\mathcal{L}_G^{GAN2} = \text{CE}(\hat{y}, y),$$

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.
The selection of the loss function for the Generator is driven by the following rationale: in the initial version, the loss function quantifies the classification of the generated image as real (low loss) or fake (high loss). These values are employed to train the Generator, thereby enhancing its capacity to deceive the Discriminator. In the case of transitioning to a generator that utilizes a mixture of Gaussians, however, the situation is distinct. Given the probabilistic nature of the new generator, the primary interest lies in training a generator that is aligned with the baseline. This loss penalizes the generator for producing images that are visually realistic but semantically incorrect.

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 the ground-truth attribute label of 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, 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

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 [4], 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 $lr = 0.0001$.

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

In this section, we detail the setup for all experiments conducted to validate our proposed model. Due to the computational demands of our architecture, the training process consistently exceeded the 4-hour GPU usage limit on the Google Colab environment. All computational tasks were performed on an NVIDIA Tesla T4 GPU.

5.1. Phase 1: Encoder and Generator Integration

Our initial experiment utilized the mini dataset, containing 10 images each of horses and zebras. The primary goal was to validate the architectural integration of our custom GM-DRIT model into the original framework.
The model was 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

Following this first experiment, we transitioned to the more robust apple2orange dataset which comprises 266 images each of apples and oranges. This change was motivated by the need for a larger dataset, which would provide a stronger signal for the model's complex learning objectives and mitigate the risk of overfitting observed with the limited horse-to-zebra dataset.

5.2. Phase 2: Loss Function Integration

This phase focused on a crucial modification to the optimization objective, including the refinement of the method to perform the backward of the Encoder and Generator loss loss (using a mixture of gaussians rather than the standard univariate gaussian distribution), KL divergence loss, and the implementation of a novel label similarity loss. The model was 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

5.3. Phase 3: Addressing the Discriminator-Generator Imbalance

To address the observed imbalance between the discriminator's and generator's training signals, we conducted three distinct experiments. All of these experiments utilized the same dataset as in Phase 2.

5.3.1 AdaIN

The first approach involved integrating an Adaptive Instance Normalization (AdaIN) layer within the generator. The training parameters were as follows:

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

5.3.2 Transposed Convolutional Layer

The second approach involved adding transposed convolutional layers to each block of the generator to enhance the feature upsampling process. The training parameters were:

- Learning rate for both discriminator and generator/encoder: 0.0001
- Batch size: 2

- Number of iterations: 4000
- Image resolution: 216x216 pixels

5.3.3 Two-Time Scale Update Rule (TT-SUR)

The third approach implemented the Two-Time Scale Update Rule (TT-SUR) [4]. This was tested under two different configurations:

Configuration A: Halving the Discriminator's Learning Rate

- Learning rate for discriminator: 0.00005
- Learning rate for generator/encoder: 0.0001
- Batch size: 2
- Number of iterations: 5500
- Image resolution: 216x216 pixels

Configuration B: 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

5.4. Phase 4: Reduced Resolution Training

In our final experiment, we adopted a reduced image resolution to overcome the computational limitations. The objective was to enable a significantly extended training duration.

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

5.5. Phase 5: Comparison with Baseline

To benchmark our proposed GM-DRIT model, we conducted a comparison with the original baseline architecture, using the apple2orange dataset.

- Learning rate for both discriminator and generator/encoder: 0.0001
- Batch size: 2
- Number of iterations: 6000
- Image resolution: 216x216 pixels

6. Results

In this section, we present the results from the experiments detailed above. Due to the computational limitations, quantitative evaluation metrics such as FID, LPIPS, JSD, and NDB could not be obtained. All results are presented through qualitative visual analysis.
The visual output of our model is systematically presented in a structured format for analysis. The 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 original image, an intermediate image representing the transition, the transformed image, a second intermediate image, and the reconstructed image.

6.1. Phase 1: Architectural Integration Results

The initial outputs, as illustrated in Figure 5, 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, its learning objective is not adequately guiding the generative process toward producing high-fidelity images. This critical analysis led us to conclude that the primary bottleneck was not in architectural connectivity or dimensional consistency, but rather in the optimization objective itself.

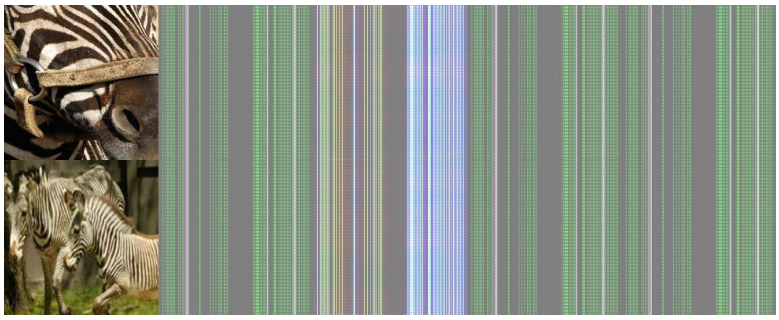

Figure 5: Example of an initial low-quality image generated by the GM-DRIT model after architectural integration.

6.2. Phase 2: Loss Function Integration Results

An example of the generated results after the modification of the loss functions is presented in Figure 6. The images begin to exhibit more discernible shapes and color patterns that resemble the target objects. 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.

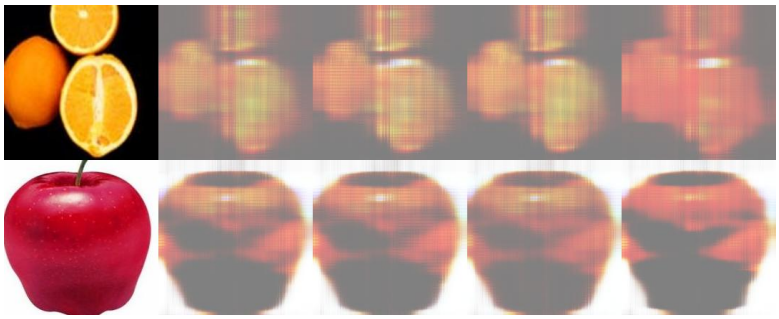

Figure 6: Generated images after conditioning the generator on encoder's inferred labels (5000 iterations).

6.3. Phase 3: Imbalance Mitigation Results

As shown in Figures 7 and 8, 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.

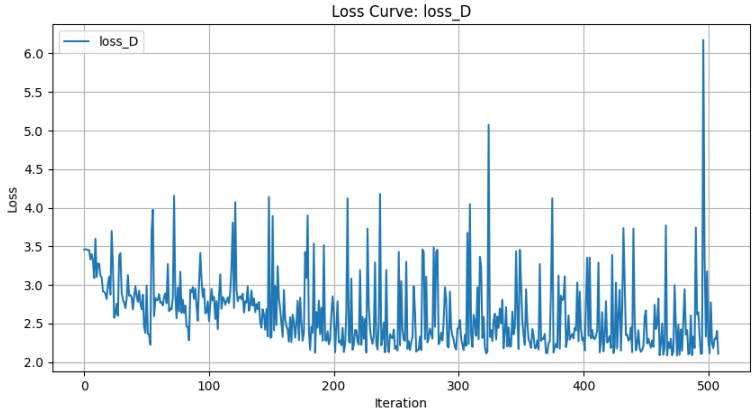

Figure 7: Discriminator loss over 5000 iterations during the first epoch, using cropped images 108×108. The values fluctuate around 2–2.5, with several noticeable peaks.

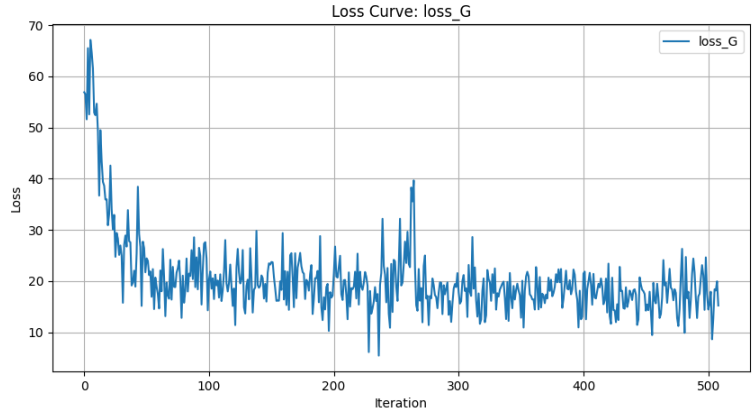

Figure 8: Generator loss over 5000 iterations during the first epoch, using cropped images 108×108. The loss initially starts at very high values and then decays, stabilizing around 20, which is still relatively high. A generator loss of this magnitude suggests that a more complex architecture may be 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.

6.3.1 AdaIN Results

As shown in Figure 9, the generated images exhibit pronounced contrast. The model appeared to successfully transform oranges into apples, as visible in the bottom row. However, the inverse transformation from apples to oranges does not appear to be successful.

6.3.2 Transposed Convolutional Layer Results

The results from adding transposed convolutional layers were inconsistent. In Figure 10, the generated images show increased contrast and a darker appearance. The bottom row, representing a transformation from oranges, shows only a slight hint of red, and the overall transformation is limited. Conversely, as depicted in Figure 11, the model struggles with the inverse transformation, with only a slight hint of yellow/orange visible. The initial apple image appears to be entirely lost.

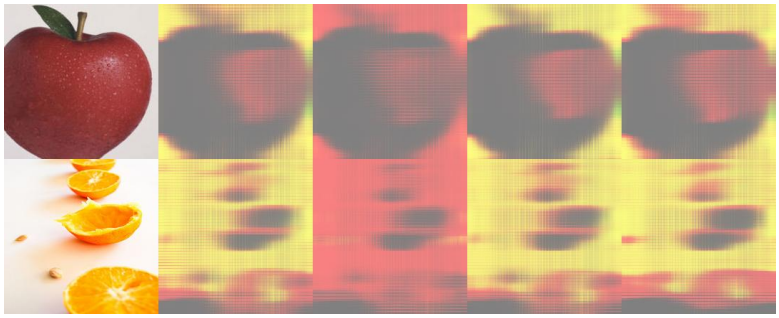

Figure 9: Images generated after integrating an AdaIN layer within the generator.

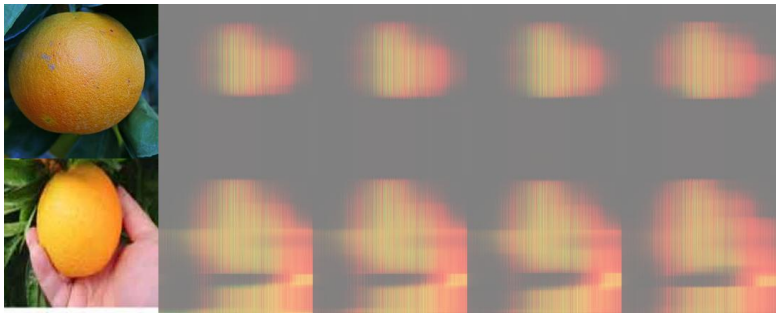

Figure 10: Images generated with 4,000 iterations after adding a transposed convolutional layer to the generator.

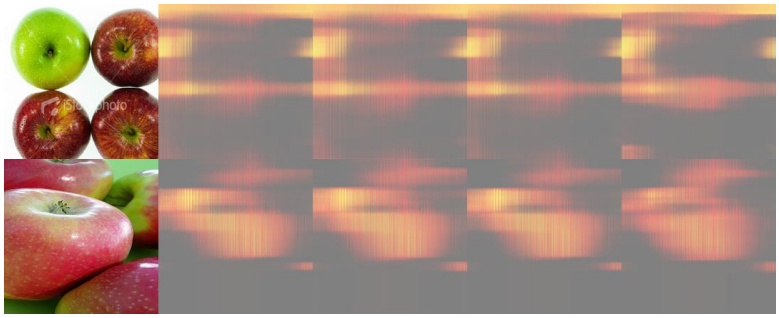

Figure 11: Another example of an image generated after adding a transposed convolutional layer.

6.3.3 TT-SUR Results

Halving the Discriminator's Learning Rate Figure 12 shows that the model successfully transforms oranges, with a noticeable increase in redness. In Figure 13, the model demonstrates a clearer transformation of an apple to a more pronounced orange hue.

Doubling the Generator's Learning Rate As illustrated in Figure 14, 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.

6.4. Phase 4: Reduced Resolution Results

As shown in Figure 15, 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,

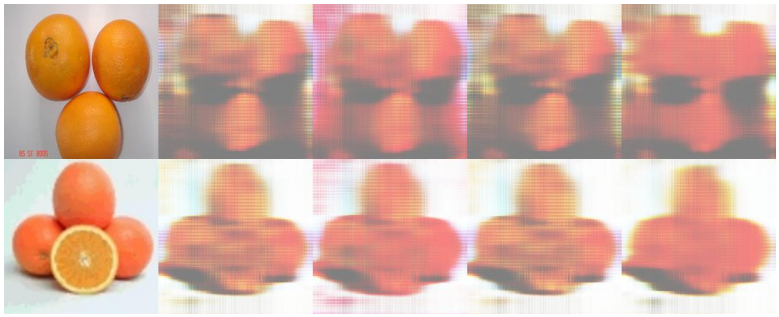

Figure 12: Generated images with a discriminator learning rate of 0.00005 and a generator/encoder learning rate of 0.0001.

Figure 13: Another example of an image generated with a halved discriminator learning rate.

Figure 14: Image generated with a discriminator learning rate of 0.0001 and a generator/encoder learning rate of 0.0002.

a crucial step towards effective image-to-image translation.

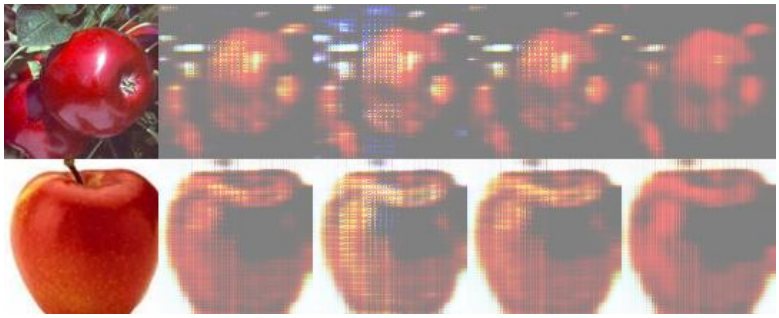

Figure 15: Generated images by GM-DRIT after 10,000 iterations with reduced resolution.

6.5. Phase 5: Comparison with Baseline Results

To benchmark our proposed GM-DRIT model, we conducted a direct comparison with the original baseline architecture. 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 16, 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 16: Image-to-image translation results of the original model after 6000 iterations.

In contrast, our GM-DRIT 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.

6.6. Time Results

The table 3 and the bar plot in Figure 17 clearly illustrate the computational trade-offs associated with different architectural modifications. The performance metric is defined as the average time per iteration, calculated over the initial 50 iterations for each model. 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 GM-DRIT models experience a similar scale-up in computational time.
A key observation is the higher average running time of all GM-DRIT 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 (*GM-DRIT 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 GM-DRIT variants, its performance is very close to that of the baseline *GM-DRIT* 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 3: Average running time per iteration over the first 50 iterations for different experiments

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

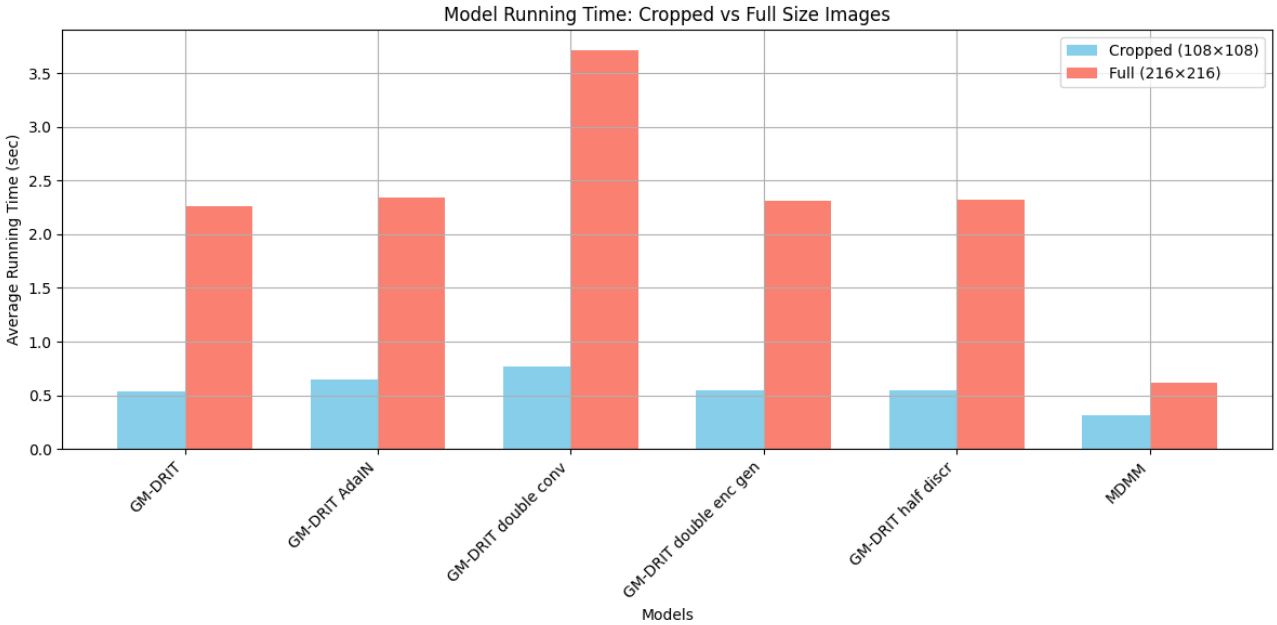

Figure 17: Bar plot illustrating the average time required by each model to perform a single iteration.

7. Conclusion

Our primary architectural modification consisted of integrating a Gaussian Mixture Variational Autoencoder (GMVAE) module into the generator, requiring substantial adjustments to both the encoder and generator in order to accommodate the mixture of Gaussians prior. This integration, together with the generator's conditioning on the encoder-predicted categorical variable y, enhanced the semantic alignment between latent representations and generated outputs, enabling the model to better capture multimodal distributions. However, the added complexity of the VAE components introduced challenges: training dynamics between the generator–encoder and the discriminator became imbalanced, and the increased computational cost constrained us to lower-resolution training, which likely affected visual fidelity. To address these issues, we explored multiple strategies. Increasing architectural complexity, for example, by incorporating Adaptive Instance Normalization (AdaIN) layers or additional transposed convolutions, did not yield consistent improvements and sometimes degraded image quality, producing darker reconstructions or collapsed transformations. In contrast, carefully adjusting

learning rates according to the Two-Time Scale Update Rule provided modest but tangible benefits: halving the discriminator's learning rate helped preserve semantic content, while doubling the generator's rate improved certain domain translations, such as orange-to-apple. In addition, we introduced two new loss functions: a KL-divergence loss tailored to the GMVAE's mixture-of-Gaussians prior, and a label similarity loss to enforce domain alignment. Overall, our experiments demonstrate that balancing adversarial dynamics and tailoring loss functions are more effective than increasing architectural depth alone. The proposed framework shows strong potential for producing diverse and semantically consistent image translations, though its greater complexity results in slower training. Due to computational constraints, we were unable to fully explore its long-term performance, suggesting that extended training and further optimization could unlock the full advantages of the GMVAE-based architecture, achieving results that surpass the baseline.

7.1. 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 and the fully exploration of the potential of our approach, 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.

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] Hsin-Ying Lee. *Multi-Domain Multi-Modality I2I translation*. GitHub repository. 2019. URL: https://github.com/HsinYingLee/MDMM.

[4] 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.