"Navigating Latent Spaces: Empowering Image Editing with Controllable GANs"

Sanket Potdar
Electrical Engineering Department
Indian Institute of Technology, Bombay
Mumbai, India
20d070071@iitb.ac.in

Sanika Padegoankar
Electrical Engineering Department
Indian Institute of Technology, Bombay
Mumbai, India
20d070069@iitb.ac.in

Simran Tanwar
Electrical Engineering Department
Indian Institute of Technology, Bombay
Mumbai, India
20d070078@iitb.ac.in

***Abstract*—This report presents the implementation and enhancement of a paper titled "Enjoy your editing: Controllable GANs for Image Editing Via Latent Space Navigation." The original work addresses challenges in attribute manipulation using Generative Adversarial Networks (GANs), aiming to achieve more effective and realistic image transformations. Our implementation builds upon the proposed methods, introducing refinements such as the incorporation of a learning rate scheduler and a shift from a ResNet50 regressor to a MobileNetV2 architecture, resulting in reduced training times.**

The paper addresses limitations in existing approaches to image editing, which often exhibit entangled attribute edits, global identity changes, and reduced photo-realism. To overcome these challenges, the model simultaneously learns multiple attribute transformations, integrates attribute regression into the training process, and incorporates content and adversarial losses to preserve image identity and photo-realism. It also proposes quantitative evaluation strategies for measuring controllable image editing performance and the model permits better control for both single and multiple-attribute editing while preserving image identity and realism.

***Index Terms*—Generative Adversarial Networks(GANs), regressor, latent space navigation, multiple attribute transformation, content loss, adversarial loss**

I. Introduction

Semantic image editing involves transforming a source image into a target image while adjusting desired semantic attributes, such as making a non-smiling person appear to be smiling. The primary goals of semantic image editing are twofold: (i) facilitating the simultaneous and continuous modification of multiple attributes and (ii) preserving the original image's identity as much as possible, all while maintaining a high level of photo-realism.

Current GAN-based approaches for semantic image editing can be broadly classified into two categories: (1) image-space editing methods directly convert one image into another across domains using various GAN variants. However, these approaches often come with a high computational cost and predominantly focus on binary attribute changes. (2) Latent-space editing methods concentrate on uncovering latent variable manipulations to enable continuous semantic image edits. Unsupervised latent-space editing methods are frequently less effective in providing semantically meaningful directions and often result in changes to the image identity during editing. Conversely, (self-)supervised methods are typically constrained to geometric edits.

The paper introduces a latent-space editing framework for semantic image manipulation. Specifically utilizing GANs, the authors employ a joint sampling strategy trained to concurrently edit multiple attributes. To unravel attribute transformations in the GAN's latent space, they integrate a regressor to predict the attributes exhibited by an image. This regressor not only allows precise control over the degree of manipulation but can also be easily extended to handle multiple attributes simultaneously. Additionally, the authors incorporate a perceptual loss and an adversarial loss to aid in preserving image identity and photo-realism during the manipulation process.

We introduced a learninig rate scheduler in the optimizer which can lead to improved accuracy and reduced overfitting of the model and changed the regressor pretrained model from resnet-50 to mobilenetv2, which has a simpler architecture compared to resnet-50 and thus requires less training time.

II. Literature Review

Generative Adversarial Netowrks (GANs) : Consisting of two neural network models, the Generator and Discriminator, engaged in a competitive learning process, GANs excel at capturing and replicating the intricate variations within datasets. The Generator generates synthetic samples to deceive the Discriminator, which, in turn, strives to distinguish between real and fake samples. This adversarial dynamic results in iterative improvements for both networks over successive training steps.

Latent Space : A key notion underlying most generative models is that of latent space. This refers to a lower-dimensional, abstract representation of data that captures the underlying structure and variations in the original high-dimensional data space. It can be considered a compressed, more organized space where different data points with similar characteristics are located closer to each other.

Semantic Image Editing: Semantic image editing provides users with a flexible tool to modify a given image guided by a corresponding segmentation map. In this task, the features of the foreground objects and the backgrounds are quite different. First, Semantic Image Editing by disentangling the objects and background disassembles the edited input into

background regions and instance-level objects. Then, we feed them into the dedicated generators. Finally, all synthesized parts are embedded in their original locations and utilize a fusion network to obtain a harmonized result.

III. Methodlogy

The paper employs GAN model consisting of generator, discriminator and regressor, all of which are pre-trained. It is using style_v2_GAN as pretrained GAN model. **Generator (G)** is responsible for transforming latent vectors z $\in \mathbb{R}^m$ from latent space Z (here m is dimentionality of latent space) into synthetic images G(z). These synthetic images are generated such that it captures the inherent attributes and features of the learned data distribution. During training, G takes a latent vector z and an edited latent vector $z^{'}$, where the latter is obtained by applying a transformation T to z with a spcified transformation degree ϵ .

$$z^{'} = z + T * \epsilon = z + \sum_{i=1}^{N} \epsilon_i \mathbf{d}_i \tag{1}$$

where
$\mathbf{T} = \{\mathbf{d}_1, ..., \mathbf{d}_N\}$,
$\epsilon = \{\epsilon_1, ..., \epsilon_N\}$,
$\mathbf{d}_i \in \mathbb{R}^m$ for all i$\in \{1, ..., N\}$
The goal is to learn latent directions T that enable attribute-specific image edits. **Discriminator (D)** evaluates the quality of the generated images produced by the generator and do its usual work which is distinguish between real and fake images i.e. images generated by generator. It computes a loss L_{disc} by assessing how well it can distinguish between real and generated images. This adversarial training process helps improve the quality and realism of the generated images over time. **Regressor(R)** is responsible for predicting attribute values of the generated images. It plays a crucial role in assessing the effectiveness of attribute transformations applied during the editing process. It takes the original attributes α
$\alpha = \{\alpha_1, ..., \alpha_N\}$
$\alpha = R(G(z))$ and $\alpha^{'} = R(G(z^{'})$
,of the generated image G(z) and predicts these attributes $\alpha^{'}$ for an edited image G($z^{'}$). The original attributes α are utilized in calculating the regression loss L_{reg} to ensure that the transformations indicated by ϵ are actually reflected in the edited images.
Intuitively, the goal of **T** is to transform z by adding semantically meaningful information such that the corresponding output image G($z\prime$) exhibits attribute changes ϵ from α. In practice, we normalize the range of attribute values to [0, 1], i.e., both α and $\alpha' \in [0, 1]$. During training, we maintain the unit range by controlling the given ϵ. Formally, ϵ is drawn from a distribution D_ϵ uniform in $[-1, 1]^N$ while considering the constraint that $0 \leq \alpha + \epsilon \leq 1$.
The objective function aims to find the latent directions **T** by minimizing the weighted objective:

$$\min_{\mathbf{T}} L,\ \lambda_1 L_{\text{reg}} + \lambda_2 L_{\text{disc}} + \lambda_3 L_{\text{content}}.$$

This objective is tailored for optimizing **T**, while keeping other modules fixed. The hyperparameters λ_1, λ_2, and λ_3 fine-tune the loss terms. The regression loss (L_{reg}) evaluates if **T** successfully performs the transformations indicated by ε using binary cross entropy:

$$L_{\text{reg}} = \mathbb{E}_{z\sim Z,\, \varepsilon\sim D_\varepsilon} \left[-\hat{\alpha}' \log \alpha' - (1 - \hat{\alpha}') \log(1 - \alpha')\right].$$

Here, α' is sampled from the distribution generated by z and ε. The second loss term (L_{disc}) measures the quality of generated images using the discriminator D:

$$L_{\text{disc}} = \mathbb{E}_{z'\sim Z'|z} \left[\log(1 - D(G(z')))\right].$$

Finally, the content loss (L_{content}), also known as perceptual loss, estimates the distance between two images to maintain image identity during transformation:

$$L_{content} = \mathbb{E}_{\prime\prime X} \left\| F(G(z')) - F(G(z)) \right\|_2, \tag{4}$$

where $F_i(\cdot)$ is a feature function, D_{content} indicates pre-trained model layers used as features, and i ranges over D_{content}. They chose 1 = 10, 2 = 0.05, 3 = 0.05 and compute the perceptual loss using the conv1 2, conv2 2, conv3 2, conv4 2 activations in a VGG-19 network pretrained on the ImageNet dataset. For the attribute regressor R, we adopt a ResNet-50 for attribute prediction on the Transient Attribute and the CelebA data. The last fully connected layer in the ResNet-50 is replaced by a linear layer with an output dimension of 40. We train the regressors for 500 epochs and use the weights with the best validation mean squared error (MSE) on CelebA and the best test MSE on the Transient Scene Database. The GAN follows the StyleGAN2 architecture and is pretrained with 200k iterations on a union of the two natural scene datasets using a resolution of 256 × 256. The training batch size is 32 and an Adam optimizer is used with a learning rate of 2e-3.

Algorithm 1 Training Procedure for Local Transformation T

Require: A pre-trained GAN with G, D, and input noise distribution $z \sim Z$; a pre-trained regressor R for predictions on N attributes; an initialized T; maximum iteration number M.

for $m = 1, \ldots, M$ **do**
  Sample random noise $z \sim Z$, and ε
  Compute internal attributes for a synthetic image, i.e., $\alpha = R(G(z))$
  Compute the actual shift value $\delta = \text{CLIP}(\alpha + \varepsilon, (0, 1)) - \alpha$
  Compute the transformed latent vector $z' = z + T(z)\delta$
  Compute $I' = G(z')$, $\alpha' = \alpha + \delta$
  Compute attribute predictions $\hat{\alpha}' = R(I')$
  Compute the loss L
  Update T
end for
Return: T
=0

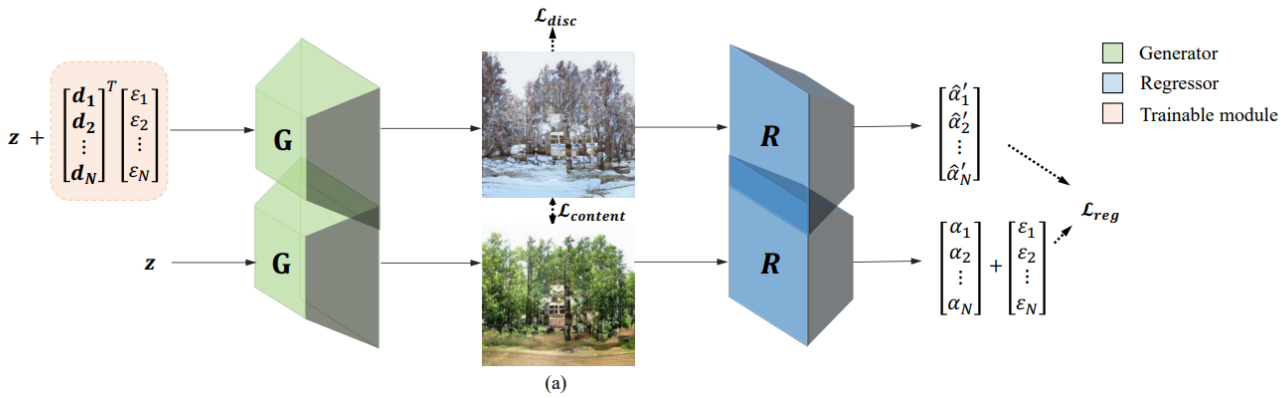


Fig. 1: Our overall framework. G takes z and an edited latent vector separately to synthesize images. $T = d_1, ..., d_N$ are the trainable latent-space directions and represents transformation extent. Original and edited image attributes, and ˆ 0, are predicted by the pre-trained regressor R. The discriminator loss L_{disc}, the regression loss L_{reg}, and the perceptual loss $L_{content}$ are used to update T .

IV. Implementation Details

A. Dataset

We evaluate the proposed approach on two types of datasets: face datasets – FFHQ (Karras et al., 2019a), CelebA. We implemented the above algorithm and made some refinements like:

B. Introducing LR scheduler in Optimizer

Initially, we incorporated a step-based learning rate (LR) scheduler with a step size of 5 and a decay factor () of 0.1 in conjunction with the Adam optimizer of LR = 1e-4. Despite these settings, the model's performance on the Smile attribute did not meet expectations. As it decayed too much by the end while we were not yet near global minima due to less number of samples. Subsequently, we opted for an exponential LR scheduler with an initial learning rate of 1e-3 and set to 0.9. However, Smile attributes were still inadequately represented in most generated images. Recognizing the need for a more precise LR adjustment, we calculated an optimal of 0.25. This calculation was based on the requirement for the LR to reach approximately 1e-4 by the 10th epoch, starting from LR of adam to be 0.001. By taking the 10th root of 0.1, which is the decay factor by 10th epoch, we determined that, $(0.1)^{1/10}$ = 0.79, thus a decay of 0.2 in every epoch, but we need to go from LR = 1e-3 to something less than 1e-4 in 10 epoch, thus we chose = 0.25. This adjustment aimed to strike a balance, ensuring a gradual LR reduction while preventing excessive diminution within the first 10 epochs of training and the results were also good with this .

TransformGraph() Class in transform_base.py is the class in which optimizer function is defined and we need to add LR scheduler in that optimizer so we defined a new function of LR scheduler to use it in optimizer. After this we call that in

```
461  def return_lrs(self):
462      scheduler = ExponentialLR(self.optimizers, gamma=0.25)
463      return scheduler
```

Fig. 2: Declaring a function to return our LR Scheduler in TransformGraph() Class using our declared optimizer

train function in train.py file.

```
39      scheduler = graphs.return_lrs()
40      for epoch in range(n_epoch):
```

Fig. 3: Storing LR Scheduler in train function

```
125         graphs.save_multi_models('{}/model_w_{}'.format(output_dir, epoch),
126                                  '{}/model_gan_{}.ckpt'.format(output_dir, epoch),
127                                  trainEmbed=trainEmbed,
128                                  updateGAN=updateGAN)
129         scheduler.step()
```

Fig. 4: Calling the step of LR Scheduler in train function at the end of each epoch

C. Using different model for Regressor

In our experimentation, we diverged from the original approach, replacing the pretrained ResNet-50 model with a pretrained MobileNetV2 as the regressor. This shift involved modifying the classification head to accommodate the desired output from 1000 to 40 classes. By adopting MobileNet as our regressor, we aimed to explore alternative model architectures and assess their impact on the overall performance, offering a fresh perspective on the role of the regressor in our specific context. We chose MobileNet because it is a small, low-latency, low-power model parameterized to meet the resource

constraints of a variety of use-cases. MobileNetV2 sometimes performs better than Resnet50 due to its less complex nature. It does not learn the noises as much as Resnet50 and thus, ends up with a better generalization. There is a new python file named MobileNet_Regressor.py which is used to fine-tune the pretrained mobilenetv2 regressor on the CelebA dataset and save the model with the best validation Mean Squared Error (MSE).The code given below is usage of that regressor in transform_base.py file which includes essential functions for modelling the architecture.

```
def get_reg_module(self):
        # Scene/Face, hard code resnet50 here
        #model = torch.hub.load('pytorch/vision:v0.5.0', 'resnet50', pretrained=False)
        #model.fc = torch.nn.Linear(2048, 40)
        model = torch.hub.load('pytorch/vision:v0.5.0', 'mobilenet_v2', pretrained=True)
        #print(model)
        model.classifier = torch.nn.Sequential(
    torch.nn.Dropout(p=0.2, inplace=False),
    torch.nn.Linear(in_features=1280, out_features=40, bias=True)
)
        model = model.cuda()
        ckpt = torch.load(constants.reg_path)
        model.load_state_dict(ckpt['model'])
        """
        If fine-tune or jointly train the classifier
        """
        # optimizer.load_state_dict(ckpt['optm'])
        # return model, optimizer
        return model, None
```

Fig. 5: Regressor Module (MobileNetV2 Model) in transform_base.py

D. Batch Size and Number of Samples

We conducted experiments to analyze the influence of batch size and the number of samples on the model's performance. We took number of samples = 1000 and batch size 4 and 8 with it and similarly we took 2000 number of samples and batch size of 4 and 8 with it and compared the results as shown in next section.

V. Results

A. Introducing LR scheduler in Optimizer

Our exponential LR scheduler gave visually good results compared to our constant LR of 1e-4 and Step LR scheduler. Also LR scheduler slightly reduced smile on images which already had a smile, but added a smile to faces with no smile.

(a) Input (b) Output

Fig. 6: With LR Scheduler

(a) Input (b) Output

Fig. 7: Without LR Scheduler

B. Different Model for Regressor

As you can see from the figures 8 and 9, MobilenetV2 performs slighlty better than Resnet50. (Images at (1,2), (2,1), (3,1) and (4,1)). Resnet50 removes the smile entirely from the image at (2,1) whereas, MobilenetV2 diminishes it but doesn't remove it entirely. Just like in the case of Resnet50, there are instances when the attribute (Smiling in this case) is diminished but the overall performance is either slightly better or at-par. This comes with the added advantage of a smaller time required for fine-tuning.

(a) Input (b) Output

Fig. 8: With MobileNet

C. With Blonde Hair Attribute

Our Blond Hair attribute model could perform better on the cases where our original/synthetic hair was already brown and our model turned it to blonde. This however does not give visually good results for persons with black hair or white hair, where only a small visual portion of hair turns blonde. Same is the case for white hair. This is due to the fact that dark brown colour being close to blonde is easier to turn, even

(a) Input (b) Output

Fig. 9: With ResNet-50

with a small number of training samples, while the black or white hair colour might need some rigorous training.

(a) Input (b) Output

Fig. 10: Blonde Hair Attribute 1

(a) Input (b) Output

Fig. 11: Blonde Hair Attribute 2

D. Batch Size and Number of Samples

For n_samples = 1000, batch size of 4 performed better than batch size of 8 as number of iterations = n_samples/batch_size, which gives 250 iterations for each epoch in batch size 4 and 125 iterations for a batch size of 8, thus larger number of iterations leads to better results. We compared the results for the Smiling attribute.

(a) Input (b) Output

Fig. 12: With Batch_size - 4 and No. of Samples - 1000

(a) Input (b) Output

Fig. 13: With Batch_size - 4 and No. of Samples - 2000

(a) Input (b) Output

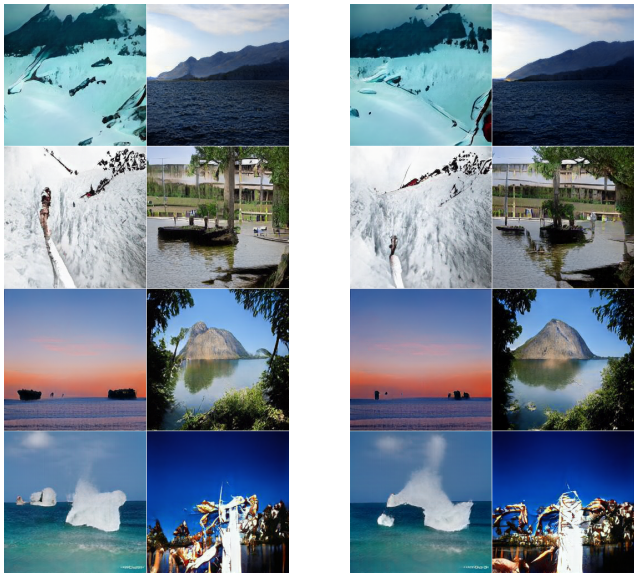
Fig. 14: With Batch_size - 8 and No. of Samples - 1000

(a) Input (b) Output

Fig. 15: With Batch_size - 8 and No. of Samples - 2000

E. Natural Images

The architecture doesn't perform very well for natural images. When the attribute to be modified was night, we can see some darkening of the scene images but no other significant changes are observed.



(a) Input (b) Output

Fig. 16: Attribute: night

VI. Challenges

- We could only train our model on 2000 or 1000 samples for 10 epochs instead of the original 20,000 samples, which took 1.5 hour for 1 epoch, due to the lack of computation power and GPU resources. We tried to purchase colab pro but it international transactions weren't enabled on any of our team members' card. Training on 2000 samples also gave quite visually good results on certain attributes on faces but did not give good results on natural/scene images. Training on 2000 samples with 4 batch size took around 100 mins for 10 epochs.
- We could not exceed our batch size greater than 8 as using a batch size of 8 took up gpu ram of 13.9 GB constantly, while colab offers only 15 GB RAM.
- The results on nature/scene pictures are not as expected due to less rigorous training process.

VII. Conclusion

This report details the implementation and refinement of a paper on "Controllable GANs for Image Editing Via Latent Space Navigation." By introducing a learning rate scheduler and transitioning from a ResNet-50 to a MobileNet regressor, the implemented enhancements aim to address challenges in attribute manipulation using Generative Adversarial Networks (GANs). The methodology involves latent space navigation to discover meaningful attribute transformations, with an objective function combining regression, adversarial, and content losses. The training procedure incorporates random noise sampling and attribute computation. Experimentation includes the evaluation of batch size and sample numbers, revealing their impact on model performance. Despite resource limitations, the implemented improvements exhibit promising results, particularly in facial attribute manipulation. A learning rate scheduler was also incorporated in the optimization process and the regressor module was changed for faster convergence. In conclusion, the refined approach enhances controllable image editing, emphasizing improved realism and identity preservation.

VIII. References

References

[1] Peiye Zhuang, Oluwasanmi Koyejo, Alexander G. Schwing, Enjoy Your Editing: Controllable GANs for Image Editing via Latent Space Navigation, Submitted on 1 Feb 2021 (v1), last revised 28 Mar 2021 (this version, v3), https://arxiv.org/abs/2102.01187

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, Generative Adversarial Networks, Submitted on 10 Jun 2014, https://arxiv.org/abs/1406.2661

[3] Wuyang Luo, Su Yang, Xinjian Zhang, Weishan Zhang, SIEDOB: Semantic Image Editing by Disentangling Object and Background, Submitted on 23 Mar 2023

[4] Latent Space in Generative Models Article