TAIM-GAN: Fusing Image With Text For Advanced Face Filtering

Ruslan Mikhailov, Rohit Bharadwaj, Bhuvnesh Kumar
Mohamed bin Zayed University of Artificial Intelligence
Masdar City, Abu Dhabi, UAE
`{ruslan.mikhailov, rohit.bharadwaj, bhuvnesh.kumar}@mbzuai.ac.ae`
Project Report

Abstract

Our project aims to semantically modify the parts of an image according to a given text that contains desired attributes (e.g., color and background) while preserving text-irrelevant contents of the image. We employ a Lightweight generative adversarial network, i.e., Text Assisted Image Manipulation-Generative Adversarial Network (TAIM-GAN) based on [10]. The authors in [10] propose the word-level discriminator loss, which provides the Generator with fine-grained training feedback at word-level, to facilitate training a lightweight generator with a small number of parameters, but can still correctly focus on specific visual attributes of an image. Extensive experimental results show that our method can better disentangle different visual attributes, then correctly map them to corresponding textual words, and thus obtain a more accurate image modification using natural language descriptions. We train TAIM-GAN on CUB-Birds, COCO, and UTKFace Datasets. We also deploy our model as a web application that anyone can use and try the demo with their custom images and text. Github link: `https://github.com/Dmmc123/taim-gan`

1 Introduction

Text-assisted image manipulation involves editing the images according to given textual descriptions to meet the user's preferences. With recent development in deep learning techniques and generative models, text-assisted image manipulation has gained remarkable popularity, and various applications based on image modification have been developed [6, 2, 3].

Previously suggested methods [4, 14] fail to provide the desired results effectively and cannot modify the text-required attributes. On the other hand, a multi-stage network can produce more realistic images [11]. However, it uses a multi-stage framework with multiple pairs of generator and discriminator [11], which occupies a large memory and requires a significant amount of time for training and inference, which is inconvenient to memory-restricted devices, such as mobile phones.

The previous discussion motivates us to investigate a lightweight network architecture in which we use a single generator, and the word-level discriminator [10], which makes the architecture lightweight by reducing the number of stages [Our Generator, Discriminator, image encoder, and text encoder consist of 15.4M, 580K, 721K, and 516K trainable parameters, respectively, for the UTKFace dataset]. In this project, we propose the Text Assisted Image Manipulation Generative Adversarial Networks (TAIM-GAN) for generating high-quality new features corresponding to the given text and simultaneously preserving text-irrelevant contents of the original image. Thus it helps automatically edit an image using some natural language descriptions. To obtain this, we employ a word-level discriminator that provides the Generator with fine-grained training feedback at the word level to facilitate training a generator with a small number of parameters. However, the Generator can still accurately focus on particular visual attributes of an image and then modify them without affecting other features not mentioned in the text.

To this end, we implemented the baseline architecture given in [10] and trained our model on the CUB bird and COCO datasets.

2 Related Work

A Generative Adversarial Network (GAN) is a neural network generally used in computer vision tasks to synthesize images. Although there are many models, which have been developed to generate realistic images, like deterministic networks [5], Variational Autoencoders (VAE) [8], and autoregressive models [18]. GAN-based models have consistently shown superior results and obtained highly realistic image outputs. [7].

GANs can also be modified to take in a conditioning variable as an input. Introducing these conditions will constrain the GAN outputs, which is beneficial for image editing tasks. [23, 1]. The conditions can also be in text format, by which we can generate images to match the given text descriptions. Reed et al. [15] first proposed an end-to-end deep neural architecture based on conditional GAN framework, successfully generating realistic images (64×64) from natural language descriptions. Reed et al. [15] proposed a two-step method called "style transfer" to produce images using query images and text descriptions.

Our method can be seen as a variant of the conditional GAN framework but conditioned on both image and text information, including modifying images according to restrictions imposed by textual descriptions.

In this project, we used the method proposed in [10] as our baseline approach. [10] proposed a lightweight architecture of GAN consisting of word level discriminator to reduce the number of stages. The two networks in the paper are trained simultaneously. One aims to generate image samples from pure noise, called the generator network, and the other aims to classify them as either coming from the actual data distribution or the generated distribution, known as the Discriminator. The Generator consists of a text encoder, an image encoder, and a series of upsampling and residual blocks.

We also use the Affine Combination Module (ACM) from [11]. ACM maps the image features relevant to the text with corresponding semantic words for effective manipulation.

The architecture of the Generator is a modified version of StackGAN++. [21]

3 Methodology of TAIM-GAN

Figure 1 describes the overall architecture of our GAN-based model. The usage of transfer learning has shown remarkable success in many of the deep learning based applications and benchmarks. Transfer learning allows us to perform better from our models at lower compute costs. Motivated by this, we use Inception-v3 [17] and VGG-16 [16], which are pre-trained on ImageNet, as our image encoders. Since the features obtained from Inception are extracted from deeper layers, they contain only rough information about the image (basic shape, outline, and layout). Thus, inception features can effectively interact with textual features, and we implement this interaction in the form of different attention modules described in the subsequent subsections. Features from VGG are extracted from shallower layers; thus, they contain more semantic information about the input image. Thus, we utilize VGG features in the perceptual loss objective of the Generator. We extract two types of features from the Inception image encoder. The feature v_l corresponds to the local features, and the feature v_g corresponds to the global features of the image. Following [20], v_l is extracted from the *'mixed_6e'* layer of the inception network. In contrast, v_g is extracted from the last pooling layer of the Inception network. Both v_l and v_g are then converted into the same semantic space of text embeddings. Thus,

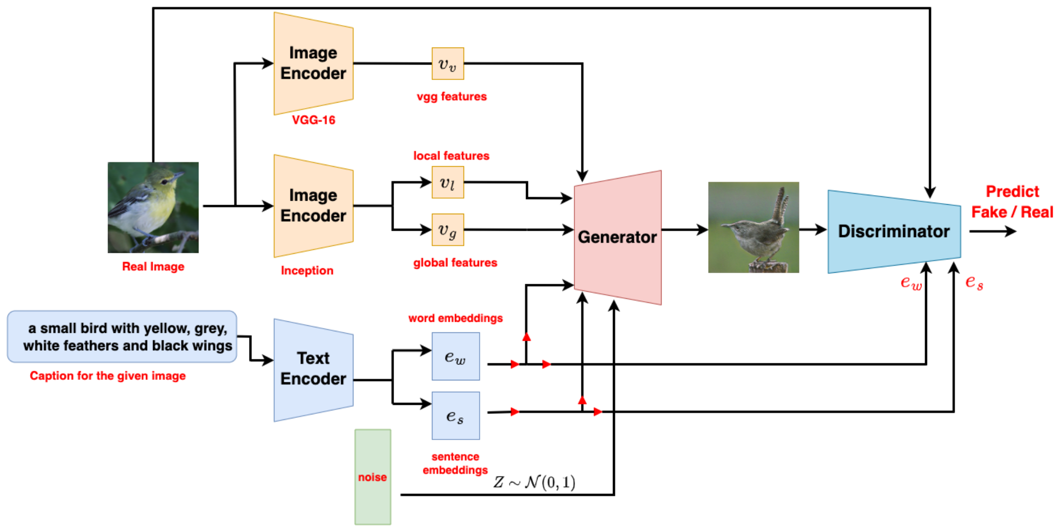

Figure 1: The Architecture of our model.

$v_l \in \mathbb{R}^{D \times 17 \times 17}$, and $v_g \in \mathbb{R}^D$, where D is the dimension of text embeddings. Using our text encoder, we obtain both the sentence embeddings, $e_s \in \mathbb{R}^{\mathbb{D}}$, and word level embeddings, $e_w \in \mathbb{R}^{\mathbb{D} \times \mathbb{L}}$, where L is the number of words in the sentence. We use a bi-directional LSTM network as our text encoder. To get the word embeddings, we concatenate the forward and the reverse hidden states of the LSTM cell corresponding to each word. For sentence embedding, we concatenate the forward and the reverse hidden state of the last word. These sentence and word embeddings are then used by the Generator and the Discriminator networks for generating the edited image and providing strong training feedback to the Generator, respectively.
We describe the architecture of the Generator, Discriminator, and the training objectives used for both networks in the subsequent subsections.

3.1 Generator

The basic architecture of the Generator is adapted from [21]. However, there are various modifications done to the base architecture. StackGAN++ uses multiple multi-stage generators and discriminators to generate outputs at different image scales. In our work, we use only a pair of one Discriminator and one Generator, and our output is generated at only a single image scale of 256×256. However, the Generator still follows the multi-stage approach of [21]. The Generator also incorporates Spatial[20] and Channel Attention[9] modules, along with the Affine Combination Module[11], which helps in generating edited images according to given text description during the inference stage.

3.1.1 Conditioning Augmentation

The first step of the generation process is to obtain the conditioning variable, $\hat{c}$, from the sentence embeddings. The conditioning variable is randomly sampled from $\hat{c} \sim \mathcal{N}(\mu(e_s), \Sigma(e_s))$, i.e., a gaussian distribution whose mean and diagonal covariance matrix is a function of the sentence embeddings. This sampling process allows us to create more training data, even for a limited set of text-image pairs. Further, the conditioning variable allows the generated images to be conditioned on the text descriptions. The randomness in generating $\hat{c}$ is beneficial to our model, as a single text description can generate different images with varying poses and appearances.

3.1.2 Network Components

The Generator takes in the local inception features, v_l, and upsamples the 17×17 feature map into 32×32 spatial size, generating h_0. Figure 2 shows the architecture of the first stage of the Generator.

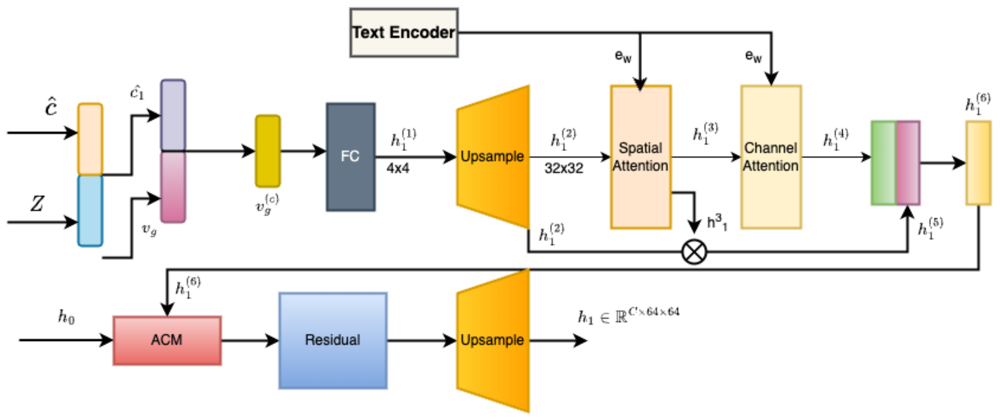

Figure 2: The Architecture of the first stage of Generator

Spatial Attention allows the model to focus on the essential words while generating a sub-region. To account for channel information, where different channels may encode different types of information, we then employ the Channel Attention module [Fig. 5]. The features $h_1^{(6)}$ and h_0 are passed as inputs to the ACM Module 3, which selects image regions from h_0 relevant to the given text, and correlates the regions with the respective semantic word tokens, to get the output feature $h_1^{(7)}$. Finally, we apply residual layers and upsample the hidden feature to return h_1 of spatial size 64×64.

The second stage follows a similar process of applying attention modules to the features and the word embeddings. However, in the ACM module, we pass in VGG features as inputs instead of h_0, which contrasts with the first stage. Since these VGG features contain more semantic details than Inception, they can help the Generator by correcting wrong attributes and filling in the missing details. Finally, after up-sampling, the output h_2 from the second generator stage is of spatial size 256×256.

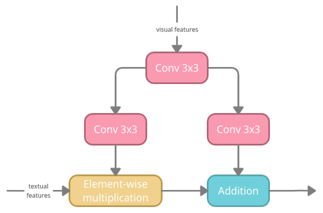

Figure 3: Architecture of Affine Combination Module

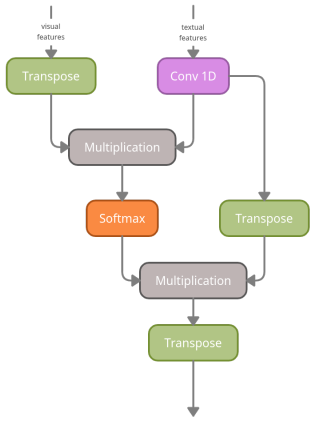

Figure 4: Architecture of Spatial Attention Module

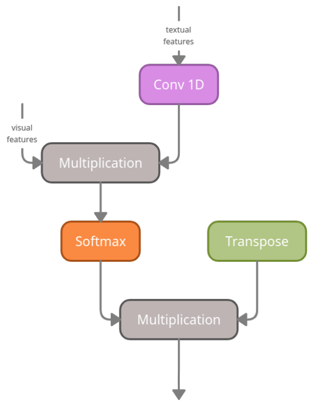

Figure 5: Architecture of Channel Attention Module

3.2 Discriminator

In GAN architectures, the Discriminator part is responsible for telling if the image is actual (comes from a raw data distribution) or fake (is generated by the Generator). Our Discriminator has a broader scale of responsibilities. Besides providing the image-level feedback (either true or false), our Discriminator also provides level-word feedback to ensure that each word is present on an image.

Our Discriminator in TAIM-GAN consists of two parts:

1. **Feature extractor** which we chose to be Inception-v3 network. The key purpose of a feature extractor is to convolve the images and retrieve meaningful visual information about them.
2. **Logit extractors** which provide image- or word-level feedback given features of real/fake images. In total, there are three types of logit extractors:
   (a) **Word-level logits extractor** which takes image features and word-level embeddings as an input and outputs a number from 0 to 1 for each word in a caption of every image to signify the presence of that word in an image.
   (b) **Unconditional logits extractor** takes just image features and tries to guess whether they correspond to real or fake images.
   (c) **Conditional logits extractor** which takes not only visual features but also sentence-level embeddings and tries to *condition* the real/fake prediction on the text of a caption.

3.3 Training

To train TAIM-GAN, we first pre-train the image and text encoders by using only the DAMSM Loss [20]. This aligns the textual embeddings with the corresponding semantic visual attributes. After this pre-training, we freeze both the image and text encoders and train the generator and discriminator modules on the given dataset.

The loss function used to train the Generator is given by Eq 1

$$\mathcal{L}_{\text{G}} = \mathcal{L}_{\text{uncond}} + \mathcal{L}_{\text{cond}} + \mathcal{L}_{\text{percept}}(I, I') + \mathcal{L}_{\text{DAMSM}} \tag{1}$$

here,

$$\begin{aligned}
\mathcal{L}_{\text{uncond}} &= -\frac{1}{2}\mathbb{E}_{I' \sim P_{\text{G}}}[\log(D(I'))] \quad \text{[Unconditional Adversarial Loss]} \\
\mathcal{L}_{\text{cond}} &= -\frac{1}{2}\mathbb{E}_{I' \sim P_{\text{G}}}[\log(D(I', S))] \quad \text{[Conditional Adversarial Loss]} \\
\mathcal{L}_{\text{percept}} &= MSE(\phi(I'), \phi(I)) \quad \text{[Perceptual Loss]}
\end{aligned}$$

The unconditional adversarial loss is calculated by using unconditional logits obtained from the Discriminator. We pass the fake images from the Generator to the Discriminator to get the image features. These image features are then converted to unconditional logits using convolution and flatten layers. Similarly, the conditional adversarial loss is calculated using the conditional logits obtained from the Discriminator. The Discriminator takes in the visual features and sentence embeddings which are then projected to the same dimensional space, concatenated, and flattened to get the conditional logits. These fake logits are then passed to a binary cross-entropy loss function with labels corresponding to the true/real images to train the generator network. $\mathcal{L}_{\text{DAMSM}}$ is the fine-grained image-text matching loss adapted from [20]. We also use perceptual loss, $\mathcal{L}_{\text{percept}}$, which prevents modifying image contents that are not described in the text and reduces randomness in the generation process. We compute the mean-squared-error loss of the VGG features [$\phi(.)$] obtained by encoding the generated and real image to calculate $\mathcal{L}_{\text{percept}}$. The following equation gives the loss for the Discriminator:

$$\begin{aligned}
\mathcal{L}_D = &-\frac{1}{2}\Big[\overbrace{\mathbb{E}_{I \sim P_{\text{data}}}[\log(D(I))]}^{\text{unconditional adversarial loss}} + \overbrace{\mathbb{E}_{I \sim P_{\text{data}}}[\log(D(I, S))]}^{\text{conditional adversarial loss}}\Big] \\
&- \frac{1}{3}\Big[\mathbb{E}_{I' \sim P_{\text{G}}}[\log(1 - D(I'))] + \mathbb{E}_{I' \sim P_{\text{G}}}[\log(1 - D(I', S))]\Big] \\
&+ \mathcal{L}_{\text{word}}(I, W)
\end{aligned}$$

where $\mathcal{L}_{\text{word}}$ [10] is the word-level loss provided to the Generator by the Discriminator, I is the real images, I' is the images generated by the Generator, S is the sentence-level embeddings from a caption, and W is word-level embeddings from each word of each caption. The unconditional and conditional adversarial loss is computed using a similar method as described earlier. However, to train the Discriminator, we pass labels corresponding to real/fake images based on whether the image is obtained from the dataset or generated by the Generator, respectively.

4 Experiments & Results

4.1 Datasets

In our project, we first train and validate our approach by performing experiments on CUB Bird, COCO, and UTKFace datasets.

- **CUB Bird Dataset[19]:** This dataset is the most widely-used dataset for fine-grained visual categorization tasks. It contains 11,788 images of 200 subcategories belonging to birds. Each image has detailed annotations: 1 subcategory label, 15 part locations, 312 binary attributes, and one bounding box. Each image has ten captions. The natural language descriptions are collected through the Amazon Mechanical Turk (AMT) platform.
- **COCO Dataset[13]:** COCO dataset is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of roughly 41k images in the test set. There are five text captions for each image in COCO. In contrast with CUB Bird, which is limited to bird images and captions, COCO contains a diverse set of objects occurring in the real world.
- **UTKFace [22]:** Dataset with 20k+ facial images for the tasks of face recongition, age & gender prediction.

We took around 10k images from UTKFace and generated captions for each image by using the BLIP [12] model since UTKFace did not provide us with text captions which are required for our work. As we are using captions generated from an AI model for UTKFace, the quality of the captions is not very good, which leads to slightly poor performance compared to the model trained on the other two datasets.

Fig 6, Fig 7, and Fig 8 show the output from the Generator during training on Birds, COCO, and UTKFace data, respectively.

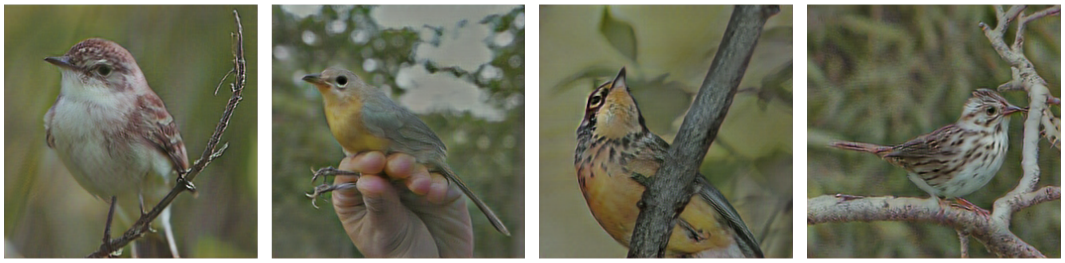

Figure 6: Images from Generator after 200 epochs of training on the CUB Bird dataset

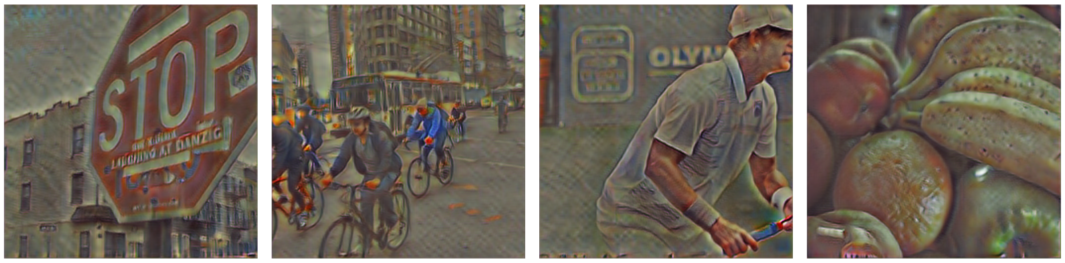

Figure 7: Images from Generator after 20 epochs of training on the COCO dataset

4.2 Evaluation

Table 4.2 shows the Inception Score (IS) metrics of different GAN architectures across different datasets. IS is based on the idea that if a GAN can generate realistic images, then a classifier trained

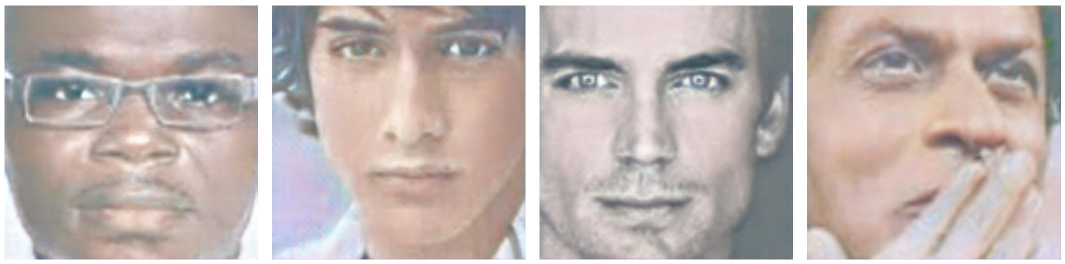

Figure 8: Images from Generator after 125 epochs of training on the UTKFace dataset

on real images should be able to classify the generated images accurately. The Inception Score is calculated as the average of the classifier's accuracy on the generated images, multiplied by the exponential of the classifier's entropy. The Inception Score has been shown to correlate well with human image quality judgments, and it has become a popular metric for evaluating GANs.

	COCO	**Birds**	**UTKFace**
StackGAN++	8.30	4.04	-
AttnGAN	25.89	4.36	-
ControlGAN	24.06	4.58	-
ManiGAN	14.96	8.47	-
TAIM-GAN (ours)	11.94	3.18	2.52

Table 1: Evaluation results

4.3 Deployment to Hugging Face

Our trained model has been deployed as a web application on hugging face (`https://huggingface.co/spaces/ML701G7/taim-gan`). Our model has three variants having three different datasets of Bird, COCO, and UTK Face. Fig. 9 shows the generated image from our bird model with text input as **"black bird with white wings"** while Fig 10 shows the generated image with textual input as **"green bird with white wings"**. similarly figure 11 is the manipulated output image according to textual input **"white bird with orange wings"**.

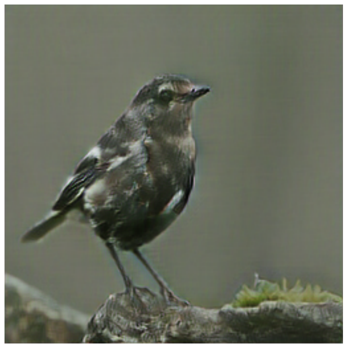

Figure 9: Black bird with white wings

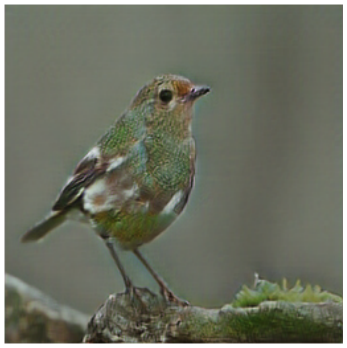

Figure 10: Green bird with white wings

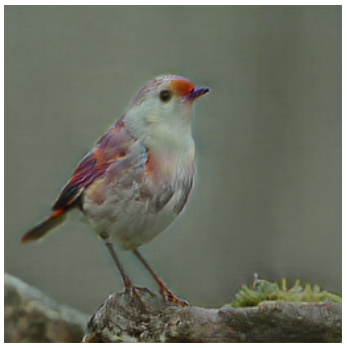

Figure 11: White bird with orange wings

4.4 Progress

Our project progress can be summarized in the following checklist:

- ☑ Research the primary references of LWGAN (AttnGAN, ManiGAN, ControlGAN, StackGAN, StackGAN++)

- ☑ Rewrite all LWGAN code from scratch:
  - ☑ Generator
  - ☑ Discriminator
  - ☑ Textual and visual encoders
- ☑ Setup for training and conducting experiments
  - ☑ Configure remote training system on HPC
  - ☑ Develop loss functions to meet objectives of D and G
  - ☑ Implement the code for training TAIM-GAN
- ☑ Test the hypothesis and train our first version of TAIM-GAN
- ☑ With TAIM-GAN, report values for Inception Score on COCO, Bird, and UTKFace datasets
- ☑ Deploy model on the internet

For more detailed information about the development process please visit the GitHub page for our project: `https://github.com/Dmmc123/taim-gan`

5 Conclusion

In this work, we implemented a text-conditioned image generation model - TAIM-GAN. TAIM-GAN is based on the LWGAN architecture and builds upon it by refactoring the entire codebase and upgrading the underlying software that the primary implementation of LWGAN uses. From this work, we have also tried to align the code with the theory described in the paper; thus, researchers can easily make a one-to-one correlation between the code and the report.

We have also collected a new dataset on top of UTKFace by propagating facial images through the BLIP network to get captions for them and obtain over 10k pairs of images and corresponding text in the domain of facial data.

In the end, we evaluated our model on three datasets - the COCO dataset, the modified version of the UTKFace dataset, and the CUB dataset and in the latter two datasets, our model was able to generate realistic images of birds or people's faces that matched the given text description. The generated images were of decent quality, and the text descriptions were often reflected in the generated images. For example, when the text description contained the word "red," the generated image often showed a bird with red plumage.

Our results suggest that our model can generate good-quality images matching the text description. In future work, we would like to evaluate our model on other datasets, such as the ImageNet dataset. We would also like to investigate other ways to improve the quality of the generated images.

References

[1] Andrew Brock et al. “Neural photo editing with introspective adversarial networks”. In: *arXiv preprint arXiv:1609.07093* (2016).

[2] Guillaume Charpiat, Matthias Hofmann, and Bernhard Schölkopf. “Automatic image colorization via multimodal predictions”. In: *European conference on computer vision*. Springer. 2008, pp. 126–139.

[3] Qifeng Chen and Vladlen Koltun. “Photographic image synthesis with cascaded refinement networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1511–1520.

[4] Hao Dong et al. “Semantic image synthesis via adversarial learning”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5706–5714.

[5] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. “Learning to generate chairs with convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1538–1546.

[6] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. “Image style transfer using convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2414–2423.

[7] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[8] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[9] Bowen Li et al. “Controllable Text-to-Image Generation”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: `https://proceedings.neurips.cc/paper/2019/file/1d72310edc006dadf2190caad5802983-Paper.pdf`.

[10] Bowen Li et al. “Lightweight generative adversarial networks for text-guided image manipulation”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 22020–22031.

[11] Bowen Li et al. “ManiGAN: Text-Guided Image Manipulation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[12] Junnan Li et al. “Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation”. In: *arXiv preprint arXiv:2201.12086* (2022).

[13] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[14] Seonghyeon Nam, Yunji Kim, and Seon Joo Kim. “Text-adaptive generative adversarial networks: manipulating images with natural language”. In: *Advances in neural information processing systems* 31 (2018).

[15] Scott Reed et al. “Generative adversarial text to image synthesis”. In: *International conference on machine learning*. PMLR. 2016, pp. 1060–1069.

[16] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: `10.48550/ARXIV.1409.1556`. URL: `https://arxiv.org/abs/1409.1556`.

[17] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2818–2826. DOI: `10.1109/CVPR.2016.308`.

[18] Aaron Van den Oord et al. “Conditional image generation with pixelcnn decoders”. In: *Advances in neural information processing systems* 29 (2016).

[19] C. Wah et al. *CUB Bird Dataset*. Tech. rep. CNS-TR-2011-001. California Institute of Technology, 2011.

[20] Tao Xu et al. “AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks”. In: *CoRR* abs/1711.10485 (2017). arXiv: `1711.10485`. URL: `http://arxiv.org/abs/1711.10485`.

[21] Han Zhang et al. “Stackgan++: Realistic image synthesis with stacked generative adversarial networks”. In: *IEEE transactions on pattern analysis and machine intelligence* 41.8 (2018), pp. 1947–1962.

[22] Song Zhang Zhifei. “Age Progression/Regression by Conditional Adversarial Autoencoder”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017.

[23] Jun-Yan Zhu et al. “Generative visual manipulation on the natural image manifold”. In: *European conference on computer vision*. Springer. 2016, pp. 597–613.