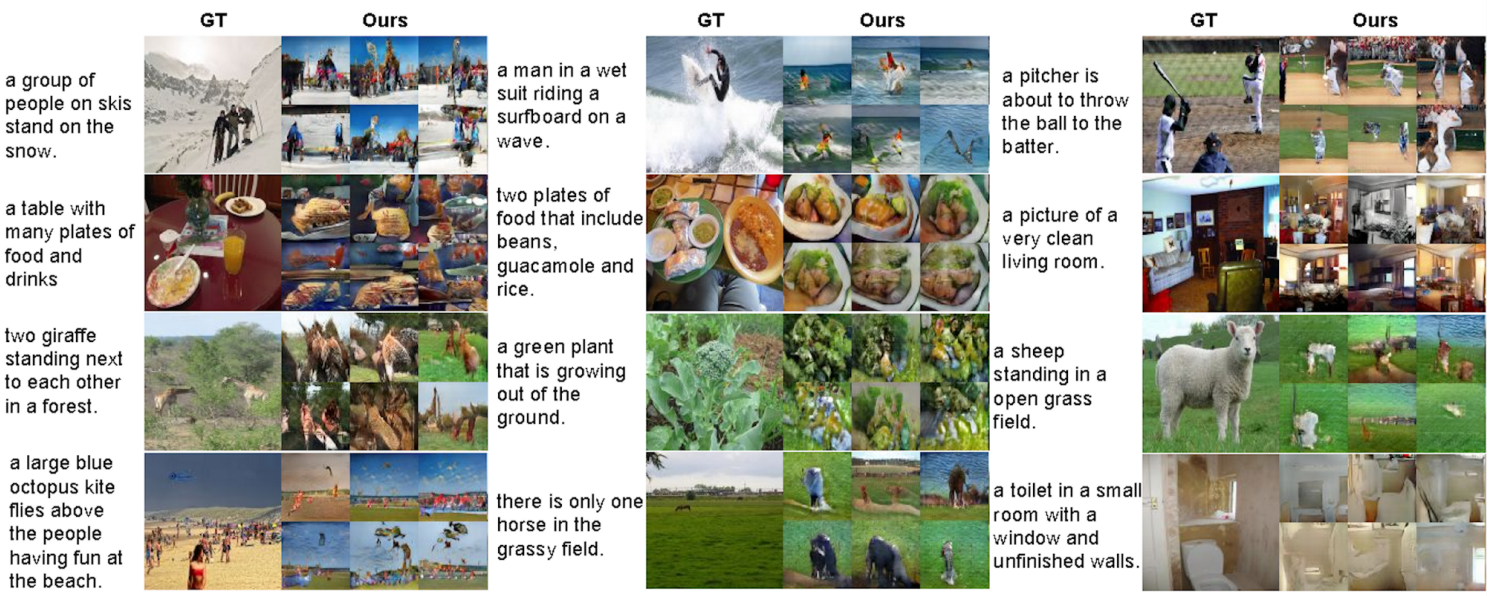


Summarizing popular Text-to-Image Synthesis methods with Python

Comparative Study of Different Adversarial Text to Image Methods

Sharmistha Chatterjee Follow
Jul 2, 2019 · 10 min read

Introduction

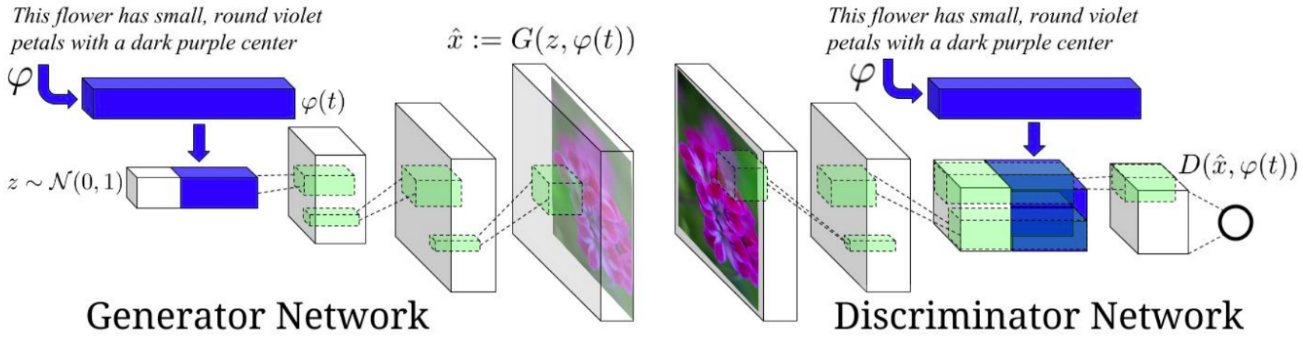
Automatic synthesis of realistic images from text have become popular with deep convolutional and recurrent neural network architectures to aid in learning discriminative text feature representations. Discriminative power and strong generalization properties of attribute representations even though attractive, its a complex process and requires domain-specific knowledge. In comparison, *natural language* offers a general and flexible interface for describing objects in any space of visual categories. *The best thing is to combine generality of text descriptions with the discriminative power of attributes.*

This blog addresses different text to image synthesis algorithms using *GAN (Generative Adversarial Network)* thats aims to directly map words and characters to image pixels with *natural language representation and image synthesis techniques*.

The featured algorithms *learn a text feature representation that captures the important visual details and then use these features to synthesize a compelling image that a human might mistake for real*.

1. **Generative Adversarial Text to Image Synthesis**

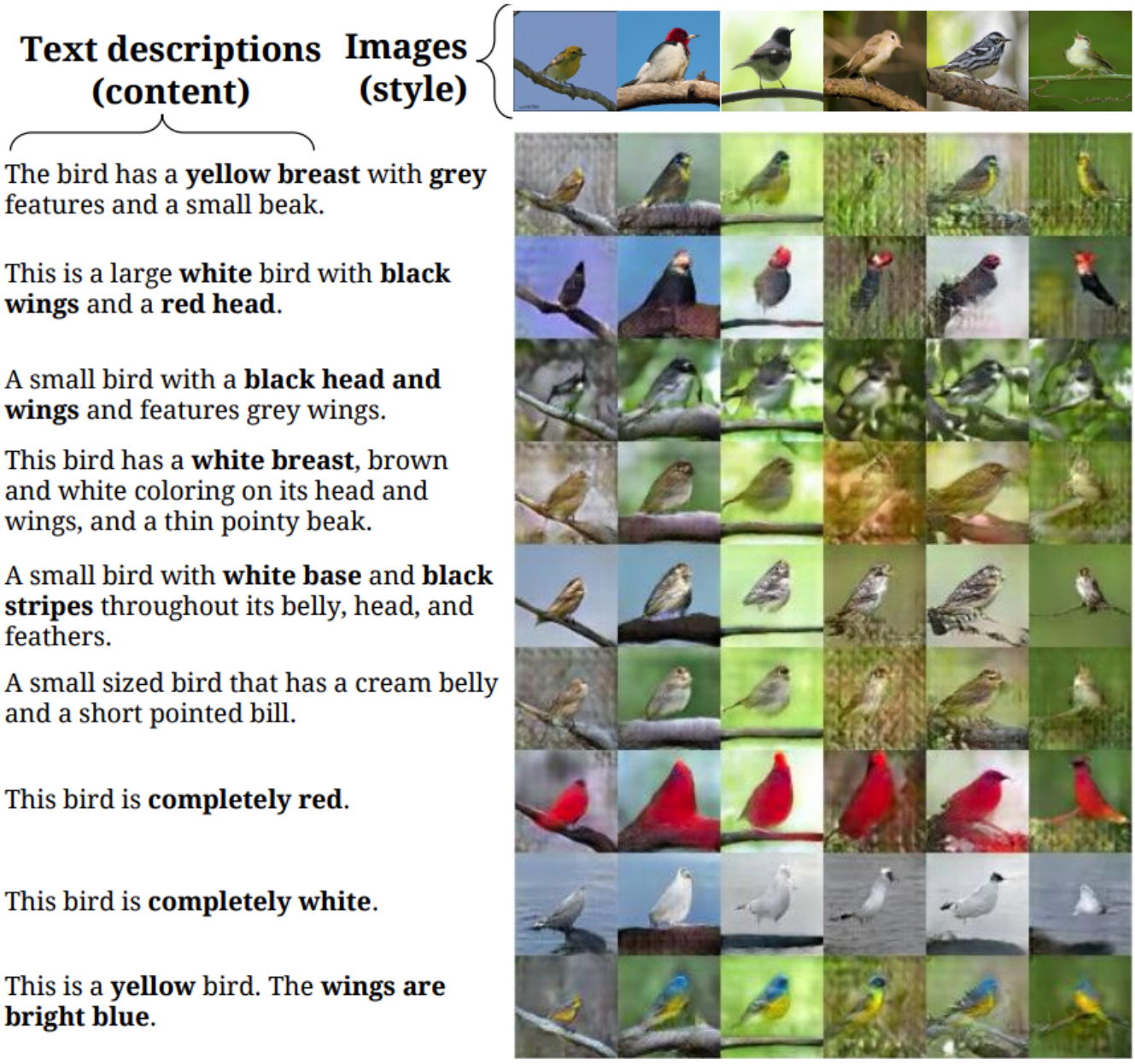
- This image synthesis mechanism uses *deep convolutional and recurrent text encoders* to learn a correspondence function with images by conditioning the model conditions on *text descriptions* instead of class labels.
- Effective approach for text-based image synthesis using a character-level text encoder and class-conditional GAN. The purpose of the GAN is to *view (text, image) pairs as joint observations* and *train the discriminator to judge pairs as real or fake*.
- Equipped with a manifold *interpolation regularizer*(regularization procedure which encourages interpolated outputs to appear more realistic) for the GAN generator that significantly improves the quality of generated samples.
- The objective of GAN is to view (text, image) *pairs as joint observations* and train the discriminator to judge pairs as real or fake.
- Both the generator network G and the discriminator network D perform *feed-forward inference* conditioned on the text feature.



Text-conditional convolutional GAN architecture, whereText encoding ϕ(t) is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

- Discriminator D, has several layers of *stride2 convolution* with *spatial batch normalization* followed by *leaky ReLU*.
- The GAN is trained in mini-batches with *SGD (Stochastic Gradient Descent)*.
- In addition to the real/fake inputs to the discriminator during training, it is also fed with a third type of input consisting of real images with mismatched text, that aids the discriminator to score it as fake.

The below figure illustrates text to image generation sample of different types of birds.



Transferring style from the top row (real) images to the content from the query text, with G acting as a deterministic decoder. The bottom three rows are captions made up by us

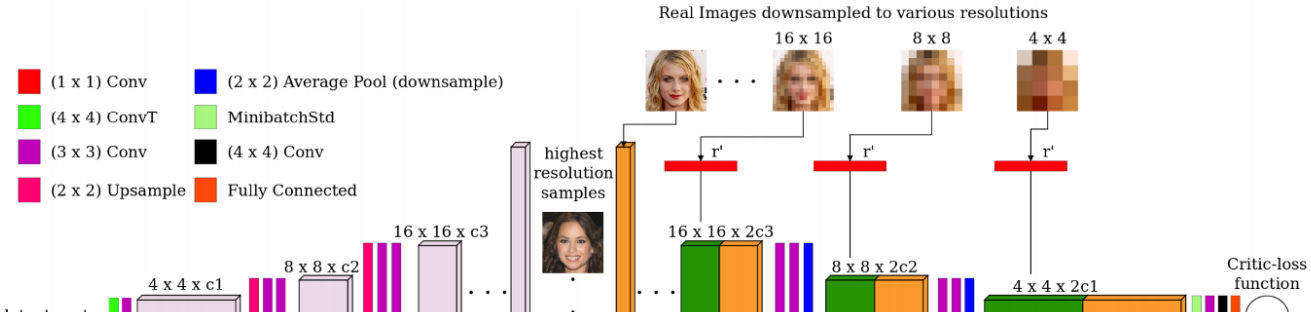
Library and Usage

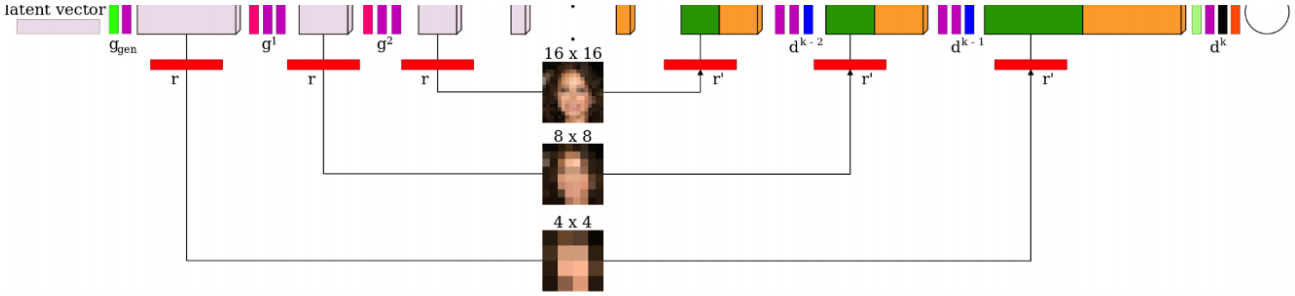
```
git clone https://github.com/zsdonghao/text-to-image.git [TensorFlow 1.0+, TensorLayer 1.4+, NLTK : for tokenizer]

python downloads.py [download Oxford-102 flower dataset and caption files(run this first)]

python data_loader.py [load data for further processing]

python train_txt2im.py [train a text to image model]

python utils.py  [helper functions]

python models.py [models]
```

2. Multi-Scale Gradient GAN for Stable Image Synthesis

Multi-Scale Gradient Generative Adversarial Network (MSG-GAN) is responsible for handling instability in gradients passing from the discriminator to the generator that become uninformative, due to a learning imbalance during training. It uses an effective technique that allows flow of gradients from the discriminator to the generator at multiple scales helping to generate synchronized multi-scale images.

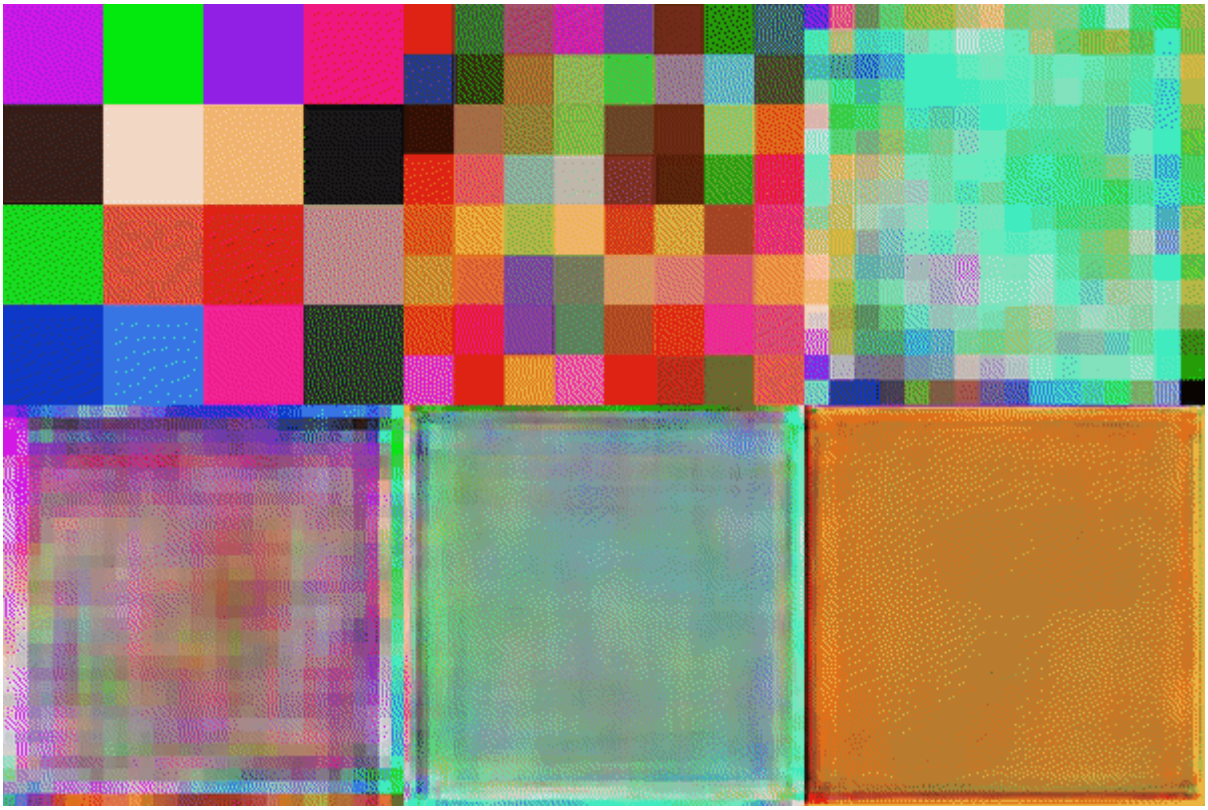
- The discriminator not only looks at the final output (highest resolution) of the generator, but also at the outputs of the intermediate layers as illustrated in the below figure. As a result, the discriminator becomes *a function of multiple scale outputs of the generator* (by using concatenation operations) and importantly, passes gradients to all the scales simultaneously.





Architecture of MSG-GAN for generating synchronized multi-scale images. Architecture proposed includes connections from the intermediate layers of the generator to the intermediate layers of the discriminator. The multi-scale images input to the discriminator are converted into spatial volumes which are concatenated with the corresponding activation volumes obtained from the main path of convolutional layers.

- MSG-GAN is robust to changes in learning rate and has a more consistent increase in image quality when compared to progressive growing (Pro-GAN).
- MSG-GAN shows the same convergence trait and consistency for all the resolutions and images generated at higher resolution maintain symmetry of certain features such as same color for both eyes, or earrings in both ears. Moreover the training phase allows better understanding of image properties (e.g., quality and diversity).



The higher resolution layers initially display plain color blocks but eventually the training penetrates all layers and then they all work in unison to produce better samples. In the first few secs of the training, the

face like blobs appear in a sequential order from the lowest resolution to the highest resolution.

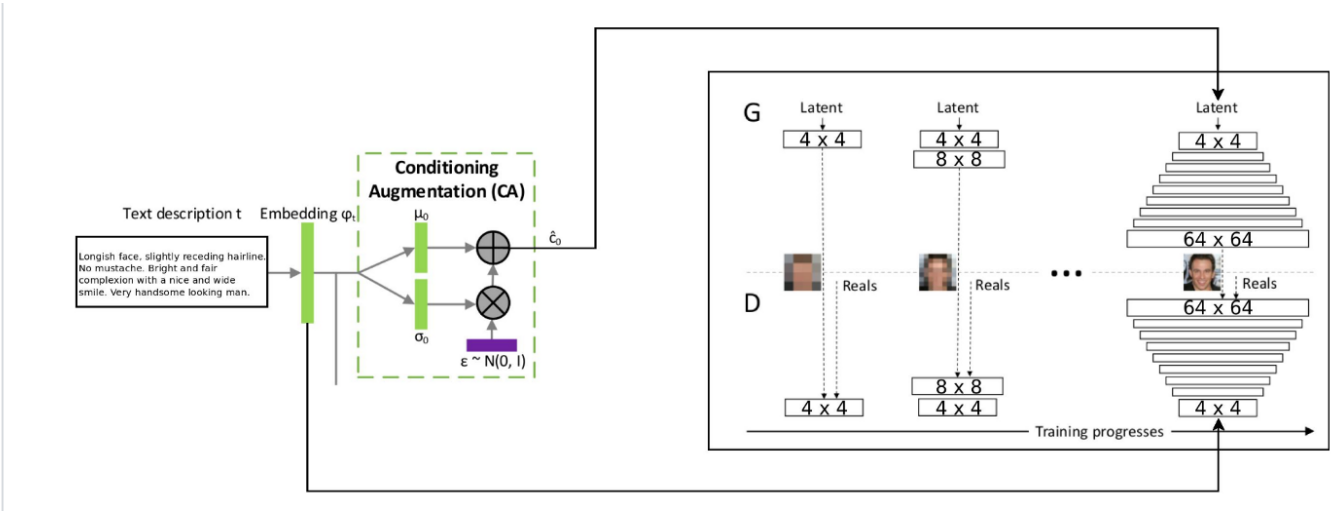
Library and Usage

```
git clone https://github.com/akanimax/BMSG-GAN.git [PyTorch]

python train.py --depth=7 \
                --latent_size=512 \
                --images_dir=<path to images> \
                --sample_dir=samples/exp_1 \
                --model_dir=models/exp_1
```

3. **T2F-text-to-face-generation-using-deep-learning (StackGAN++ and ProGAN)**

- In the ProGAN architecture, both the generator and discriminator are grown progressively starting from a low resolution, by adding new layers that model increasingly fine details as training progresses. It helps in a more stable and faster training process.
- StackGAN architecture consists of *multiple generators* and *discriminators in a tree-like structure*; images at multiple scales corresponding to the same scene are generated from different branches of the tree. StackGAN approximates multiple related distributions like multi-scale image distributions and joint conditional and unconditional image distributions.
- T2F uses a combined architecture of *ProGAN-> for synthesis of facial images* and *StackGAN -> for text encoding with conditioning augmentation*. The textual description is encoded into a summary vector using an *LSTM network*. The summary vector i.e. Embedding as illustrated in the below diagram is passed through the *Conditioning Augmentation block (a single linear layer)* to obtain the textual part of the latent vector (uses VAE like re-parameterization technique) for the GAN as input.
- The second part of the latent vector is *random gaussian noise.* The latent vector so produced is fed to the generator part of the GAN, while the embedding is fed to the final layer of the discriminator for *conditional distribution matching*. The training of the GAN progresses layer by layer at increasing spatial resolutions.

- The new layer is introduced using the fade-in technique to avoid destroying previous learning.



T2F architecture for generating face from textual descriptions

The below figure illustrates mechanism of facial image generation from textual captions for each of them.

Library and Usage

```
git clone https://github.com/akanimax/T2F.git

pip install -r requirements.txt

mkdir training_runs

mkdir training_runs/generated_samples training_runs/losses
training_runs/saved_models

train_network.py --config=configs/11.comf
```

4. **Object-driven Text-to-Image Synthesis via Adversarial Training**

Top: AttnGAN and its grid attention visualization. Middle: Modified implementation of two-step (layout-image) generation. Bottom: Obj-GAN and its object-driven attention visualization. The middle and bottom generations use the same generated semantic layout, and the only difference is the object-driven attention.

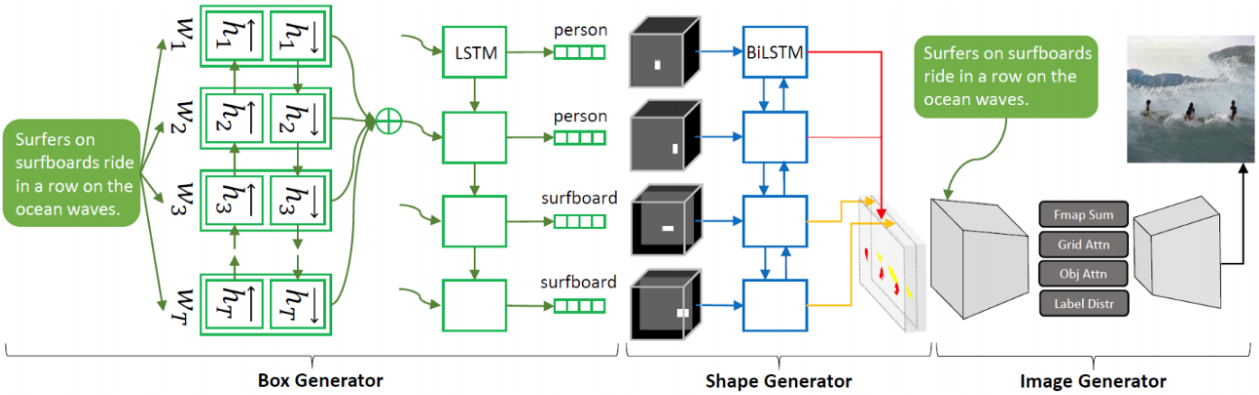


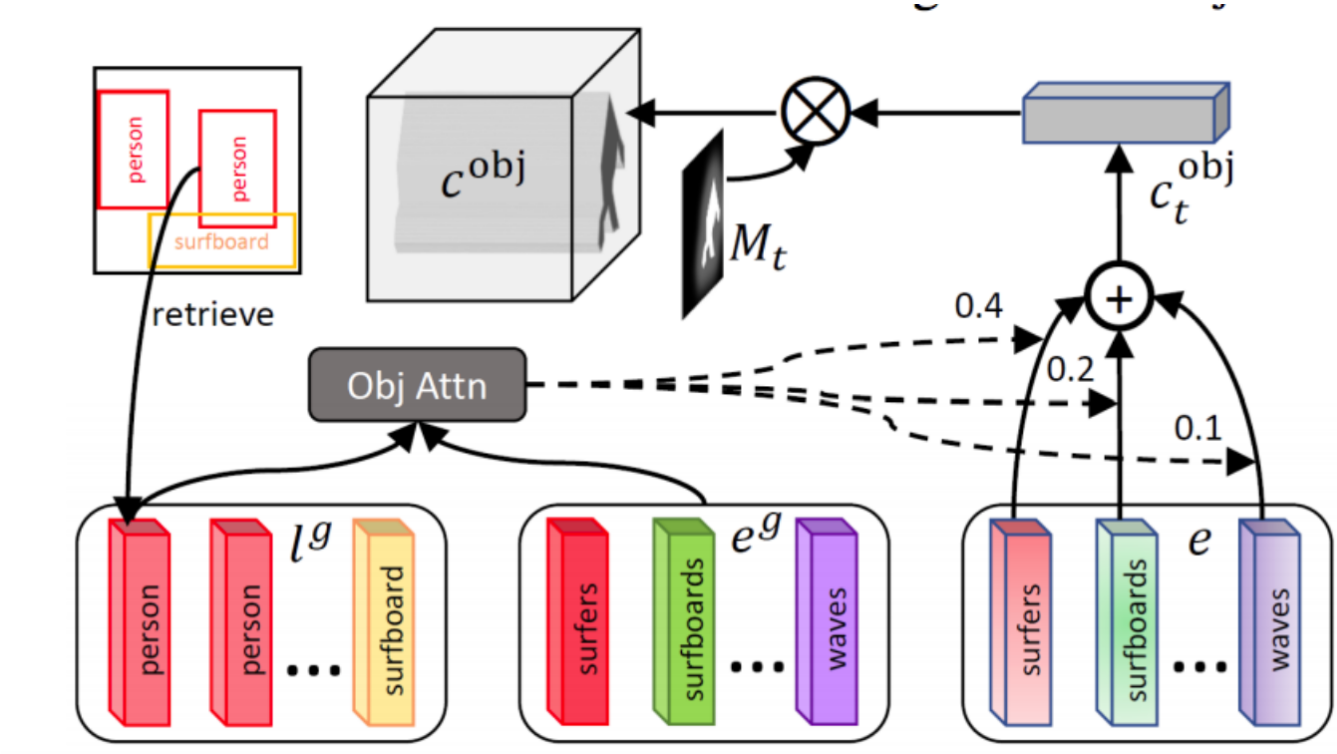
Figure 2: Obj-GAN completes the text-to-image synthesis in two steps: the layout generation and the image generation. The layout generation contains a bounding box generator and a shape generator. The image generation uses the object-driven attentive image generator.

Obj-GAN2 step text-image synthesis: layout generation and the image generation. The layout generation contains a bounding box generator and a shape generator. The image generation uses the object-driven attentive image generator

- **Object-driven Attentive GAN (Obj-GAN)** performs fine-grained text-to-image synthesis in two steps: generating a semantic layout (class labels, bounding boxes, shapes of salient objects), and then generating the image by synthesizing the image by a de-convolutional image generator.
- Semantic layout generation is accomplished when the Obj-GAN takes the sentence as input and generates a semantic layout, a sequence of objects specified by their bounding boxes (with class labels) and shapes.
- The box generator is trained as an *attentive seq2seq model* to generate a sequence of bounding boxes, followed by a shape generator to *predict and generate the shape of each object in its bounding box*.
- In the image generation step, the *object-driven attentive generator* and *object-wise discriminator* are designed to enable image generation conditioned on the semantic layout generated in the first step. The generator concentrates on synthesizing the image region within a bounding box by focusing on words that are most relevant to the object in that bounding box.

- Attention-driven context vectors use both patch-wise and object-wise context vectors for specific image regions to encode information from the words that are most relevant to that image region.
- A *Fast R-CNN* based *object-wise discriminator* is used to provide rich object-wise discrimination signals on whether the synthesized object matches the text description and the pre-generated layout.
- *Object-driven attention* (paying attention to most relevant words and pre-generated class labels) performs better than traditional grid attention, capable of *generates complex scenes in high quality*.

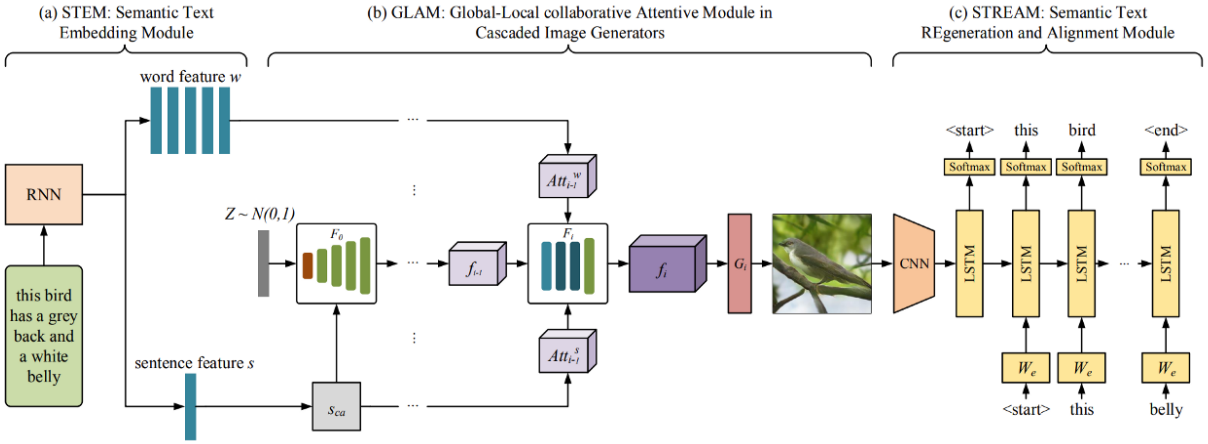
```
The open source code for Obj-GAN from Microsoft is not available
yet.
```



5. **MirrorGan**

- MirrorGAN is a global-local attentive and semantic-preserving text-to-image-to-text framework.

- MirrorGAN is responsible of learning text-to-image generation by re-description and consists of three modules: *a semantic text embedding module (STEM), a global-local collaborative attentive module for cascaded image generation (GLAM),* and *a semantic text regeneration and alignment module (STREAM)*. The following figure illustrates the individual components.



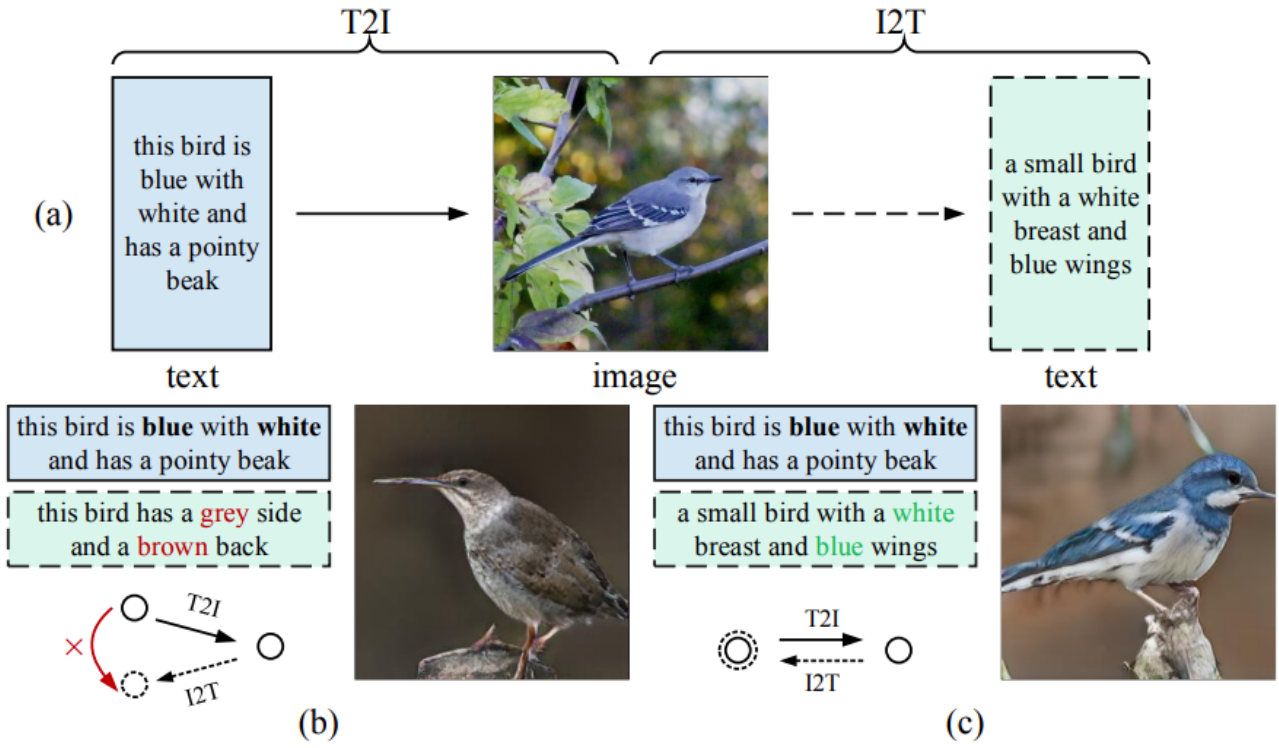
Semantic text embedding module (STEM), Global-local collaborative attentive module for cascaded image generation (GLAM), Semantic text regeneration and alignment module (STREAM)

- *STEM* generates word-and sentence-level embeddings using recurrent neural network (RNN) to embed the given text description into local word-level features and global sentence-level features.
- *GLAM* has a multi-stage cascaded generator by stacking three image generation networks sequentially for generating target images from coarse to fine scales, leveraging both local word attention and global sentence attention to progressively enhance the diversity and semantic consistency of the generated images.
- *STREAM* seeks to regenerate the text description from the generated image, which semantically aligns with the given text description.
- Word-level attention model is used to generate an attentive word-context feature. Word embedding and the visual feature is taken as the input in each stage. The word embedding is first converted into an underlying common semantic space of visual features by a perception layer and multiplied with the visual feature to obtain the attention score. Finally, the attentive word-context feature is obtained by calculating

the inner product between the attention score and perception layer along with word embedding.

- MirrorGAN includes a *semantic text regeneration* and *alignment module* to regenerate the text description from the generated image, which semantically aligns with the given text description. An encoder decoder-based image caption framework is used in the architecture. The *encoder is* a *convolutional neural network (CNN)* and the *decoder is* a *RNN*.

- *MirrorGAN* performs better than *AttnGAN* at all settings by a large margin, demonstrating the superiority of the proposed text-to-image-to-text framework and the global-local collaborative attentive module, since MirrorGAN generated high-quality images with semantics consistent with the input text descriptions.



(a) Illustration of the mirror structure that embodies the idea of learning text-to-image generation by re-description. (b), (c) Semantically inconsistent and consistent images/re-descriptions generated by attentional Generative adversarial networks and the MirrorGAN, respectively

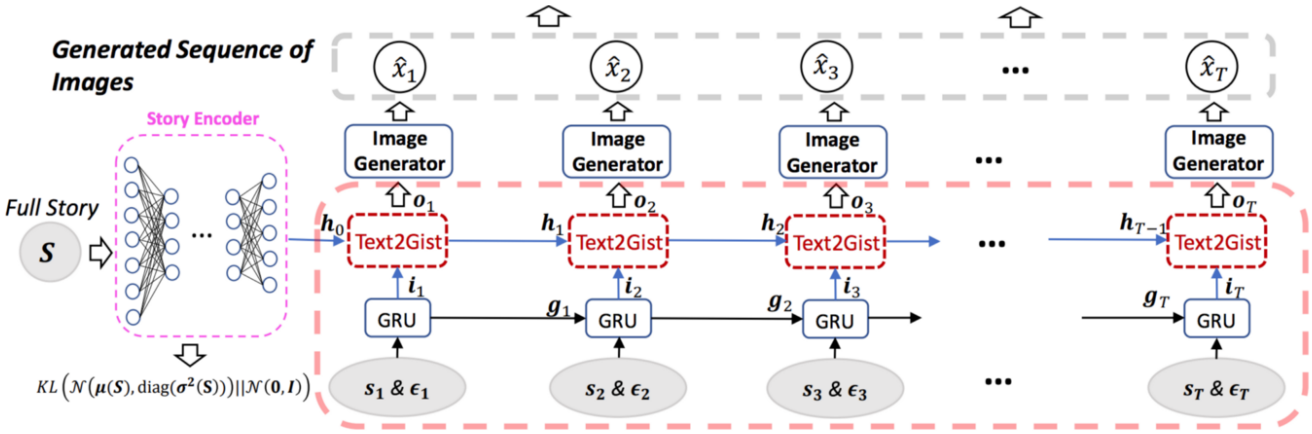
Library and Usage

```
git clone git@github.com:komiya-m/MirrorGAN.git [python 3.6.8, keras
2.2.4, tensor-flow 1.12.0]

Dependencies : easydict, pandas, tqdm

python main_clevr.py

cd MirrorGAN

python pretrain_STREAM.py

python train.py
```

6. **StoryGAN**

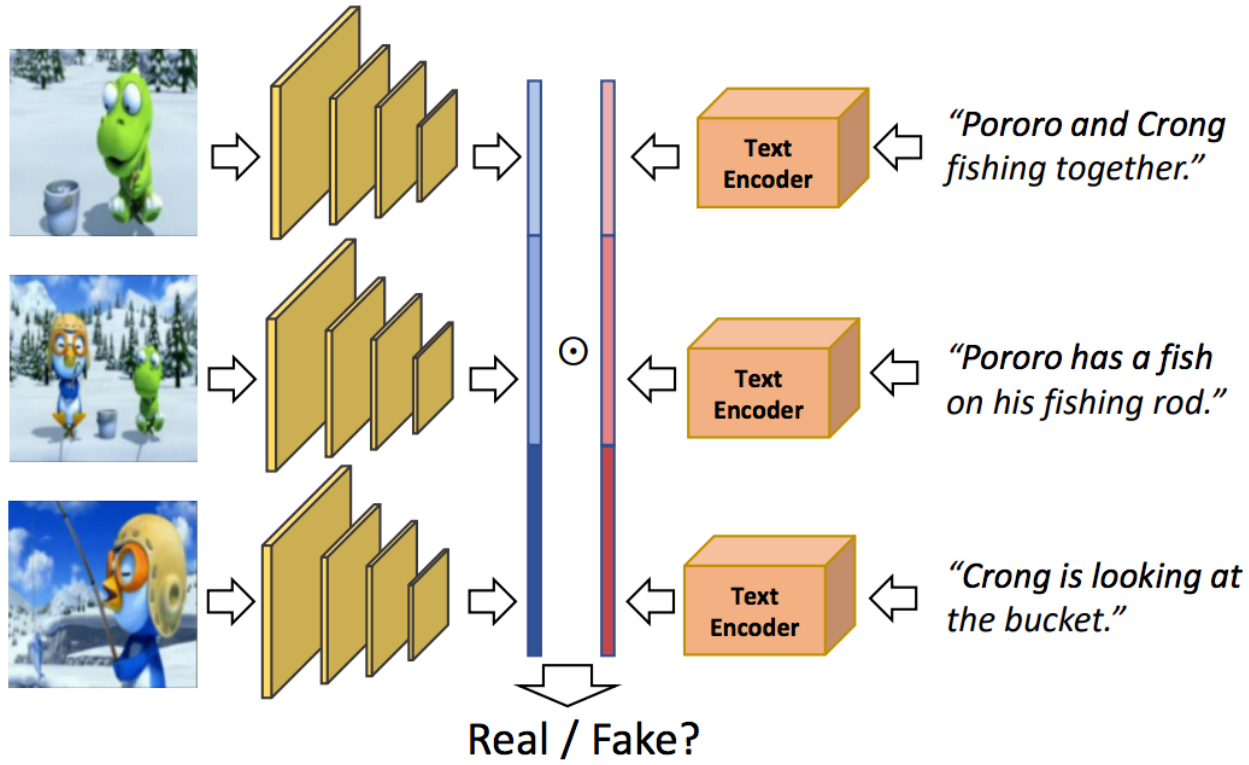
- Story visualization takes as input a multi-sentence paragraph and generates at its output sequence of images, one for each sentence.
- Story visualization task is a sequential conditional generation problem where it jointly considers the current input sentence with the contextual information.
- Story GAN gives less focus on the continuity in generated images (frames), but more on the global consistency across dynamic scenes and characters.
- Relies on Text2Gist component in the Context Encoder, where the Context Encoder dynamically tracks the story flow in addition to providing the image generator with both local and global conditional information.
- Two-level discriminator and the recurrent structure on the inputs helps to enhance the image quality and ensure the consistency across the generated images and the story to be visualized.

The below figure illustrates a StoryGAN architecture, where the variables in gray solid circles are the input story S and individual sentences *s1, . . . , sT* with random noise *1, . . . , T* . The generator network contains the *Story Encoder, Context Encoder and Image Generator*. There are two discriminators on top, which discriminate whether each image sentence pair and each image-sequence-story pair are real or fake

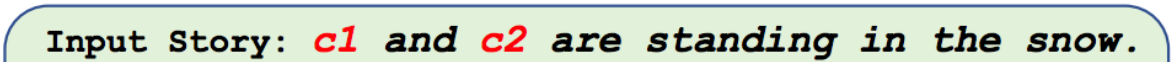


The framework of StoryGAN

The Story GAN architecture is capable of distinguishing real/fake stories wit the feature vectors of the images/sentences in the story when they are concatenated. The product of image and text features are input to a fully connected layer with sigmoid non-linearity to predict whether it is a fake or real story pair.



Structure of the story discriminator.



Generation result by changing character names in the same story. The story template is given at the top, with the character names c1, c2 and c3 in the two instances of the story, each one shown in a column

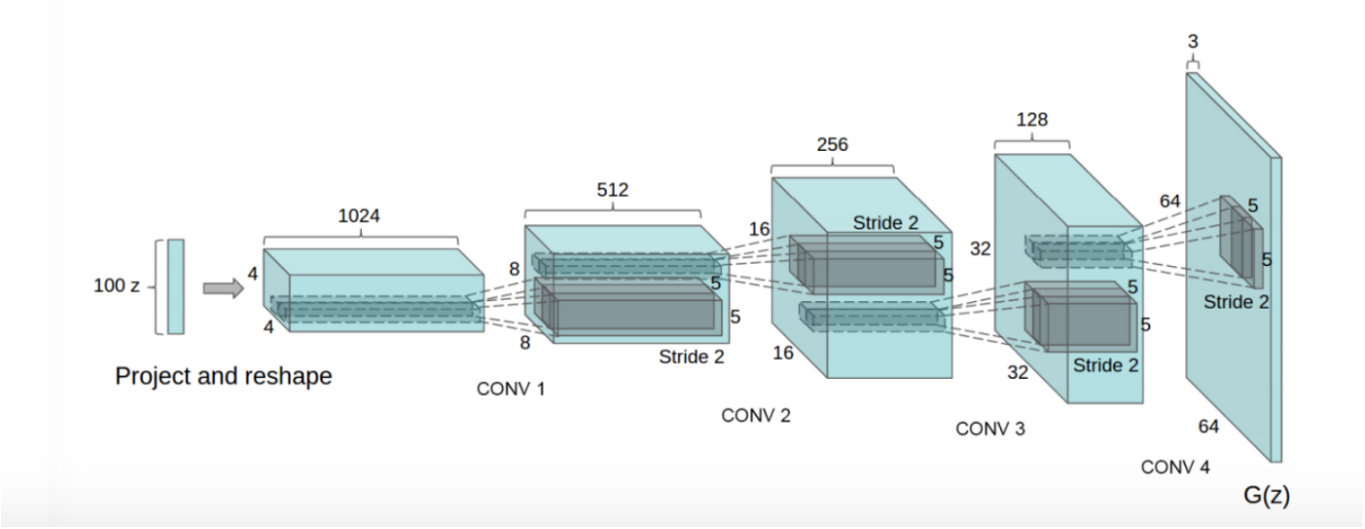
Library and Usage

```
git clone https://github.com/yitong91/StoryGAN.git   [Python 2.7, PyTorch, cv2]

python main_clevr.py
```

7. Keras-text-to-image :

In Keras text to image translation is achieved using GAN and Word2Vec as well as recurrent neural networks.

It uses *DCGan*(*Deep Convolutional Generative Adversarial Network*) which have been a breakthrough in GAN research as it introduces major architectural changes to tackle problems like training instability, mode collapse, and internal covariate shift.



Sample DCGAN Architecture to generate 64x64 RGB pixel images from the LSUN dataset

Library and Usage

```
git clone https://github.com/chen0040/keras-text-to-image.git

import os
import sys
import numpy as np
from random import shuffle


def train_DCGan_text_image():
    seed = 42

    np.random.seed(seed)

    current_dir = os.path.dirname(__file__)
    # add the keras_text_to_image module to the system path
    sys.path.append(os.path.join(current_dir, '..'))
    current_dir = current_dir if current_dir is not '' else '.'

    img_dir_path = current_dir + '/data/pokemon/img'
    txt_dir_path = current_dir + '/data/pokemon/txt'
    model_dir_path = current_dir + '/models'

    img_width = 32
    img_height = 32
    img_channels = 3

    from keras_text_to_image.library.dcgan import DCGan
    from keras_text_to_image.library.utility.img_cap_loader import
load_normalized_img_and_its_text

    image_label_pairs =
load_normalized_img_and_its_text(img_dir_path, txt_dir_path,
```

```
img_width=img_width, img_height=img_height)

    shuffle(image_label_pairs)

    gan = DCGan()
    gan.img_width = img_width
    gan.img_height = img_height
    gan.img_channels = img_channels
    gan.random_input_dim = 200
    gan.glove_source_dir_path = './very_large_data'

    batch_size = 16
    epochs = 1000
    gan.fit(model_dir_path=model_dir_path,
image_label_pairs=image_label_pairs,
            snapshot_dir_path=current_dir + '/data/snapshots',
            snapshot_interval=100,
            batch_size=batch_size,
            epochs=epochs)

def load_generate_image_DCGaN():
    seed = 42
    np.random.seed(seed)

    current_dir = os.path.dirname(__file__)
    sys.path.append(os.path.join(current_dir, '..'))
    current_dir = current_dir if current_dir is not '' else '.'

    img_dir_path = current_dir + '/data/pokemon/img'
    txt_dir_path = current_dir + '/data/pokemon/txt'
    model_dir_path = current_dir + '/models'

    img_width = 32
    img_height = 32

    from keras_text_to_image.library.dcgan import DCGan
    from keras_text_to_image.library.utility.image_utils import
img_from_normalized_img
    from keras_text_to_image.library.utility.img_cap_loader import
load_normalized_img_and_its_text

    image_label_pairs =
load_normalized_img_and_its_text(img_dir_path, txt_dir_path,
img_width=img_width, img_height=img_height)

    shuffle(image_label_pairs)

    gan = DCGan()
    gan.load_model(model_dir_path)

    for i in range(3):
        image_label_pair = image_label_pairs[i]
        normalized_image = image_label_pair[0]
        text = image_label_pair[1]
```

```
        image = img_from_normalized_img(normalized_image)
        image.save(current_dir + '/data/outputs/' + DCGan.model_name
+ '-generated-' + str(i) + '-0.png')
        for j in range(3):
            generated_image = gan.generate_image_from_text(text)
            generated_image.save(current_dir + '/data/outputs/' +
DCGan.model_name + '-generated-' + str(i) + '-' + str(j) + '.png')
```

Conclusion

Here I have presented some of the popular techniques for generating images from text. You can explore more on some more techniques at https://github.com/topics/text-to-image. Happy Coding!!

Machine Learning Artificial Intelligence Data Science Python Deep Learning