EEE443 Neural Network Project Final Report
Text to Image Generation

Melih Berk Yılmaz, Fuat Arslan
Electrical and Electronics Engineering Department, Bilkent University, Bilkent, 06800, Ankara, Turkey
(Dated: May 27, 2022)

***Abstract* — The purpose of this project is to create a newly generated image corresponding to given text descriptions. The main idea behind this project is to understand the meaning of the input context so that the network can generate images representing these specifications. Therefore, 2 state of the art deep learning methods are stacked on top of each other, Natural Language Proceesin(NLP) and Generative Adversarial Network (GAN) model. NLP models are used to extract the feature vectors of given sentence(s) whereas GAN models are used to generate non existing images from extracted features of input sentence(s). For feature extraction of input sentences, Doc2Vec model is used, in which a unique vector is assigned for each input paragraphs. As in the n-gram model, the next word is predicted by utilizing the information gathered from contexts samples from the paragraph. [1] Then, the result of the Doc2Vec model is inputted to second stage of the general model, GAN. In this stage, output of text embedding, noise and images corresponding the given captions are combined to create a non existing image.**

1. Introduction

Visualization is one of the most fundamental human-like feature. Whenever a person read a text that describes somewhat a visual scene, that person can see the corresponding images in his/her mind. This text to image transition so natural that it is mostly done without conscious.Furthermore, in many cognitive processes such as memory, spatial navigation, and reasoning, in-mind visualization plays a key role [2] At this age of AI, machine are getting closer to human. For this purpose, text to image transition is significant that a machine should capable of. Therefore, these days text to image (T2I) became a active research area.

T2I task is not an easy task in to handle. It includes understanding a piece of text and extract the features which dictates the general frames of the visual material. However, most of the time text is not complete depiction thus, human brain completes rest of the features itself by using "imagination". These auto-complete generally works based on previous knowledge but somehow different from them. These are very natural for a human but requires complete study and attention to teach to a machine which works with just binary numbers.

In this paper it is aimed to create a deep learning based machine learning algorithm to generate corresponding images from given text. Feature extraction will be done on the text to learn from text. By using extracted knowledge, images will be generated.

1, A. Feature Extraction (Encoder)

Feature extraction is the method to reduce the dimensionality of the data, which greatly alleviates the burden stemming from the computational expenses. [3] Using feature extraction, all data is converted and combined into variables, which decreases the number of the data to be processed. [4] Especially, the dimensions of NLP related tasks can reach up to hundred thousands if a unique representation is assigned to each word. In such a case, training the model can be impossible or it may take lots of time, which doesn't an effective way of completing the project.

Since the dictionary size is around 6000 and captions ranges from 80 to 230 words in our case, word embedding methods are avoided not to increase the dimensionality of the vector representation of paragraphs. Moreover, extracting word by word meaning for such long captions are not effective for learning process. Thus, extracting features based on sentences and combining these features to obtain a vector representation for a paragraph yields better results for this task.

In this project, a special function of Doc2Vec of Gensim model is used to embed or extract features of captures. This model is trained from scratch by converting the captions for each training image into a list of list of words. Each caption corresponding to one image (each paragraph for one image) is represented by a 100 dimensional vector. Different numbers are tried but best results are obtained for this. Also, many papers uses the same text embedding size. [5]

1, B. Image Generation (Decoder)

The task requires generating image from text embedding. Since it is not just finding the image from data it should generate its own new created image. This task is very compatible with GAN. [6] There are plenty of different architecture for GANs. In this paper first Deep Convolutional GAN(DCGAN) is used. It is shown that DCGAN in unsupervised case generates better images from more basic GANs. [7] DCGAN was conditioned with text embedding in order to teach it description of the image and make it free and do the job which it is good at. Secondly, Wasserstein GAN (WGAN) was

used. It is shown that, WGAN can prevent mode collapse that can be occurred in DCGANs. Also, WGAN is more stable and robust architecture for training. [8] Therefore, WGAN tried as second architecture which provides a way different from traditional training methods. WGAN was conditioned just like DCGAN for make it text specific rather than it is basic form which work on unsupervised image generation. Lastly StackGAN architecture was used due to the its high performance in this specific task. [9] This architecture provides a new way of conditioning and gives better image resolutions. [10]

1, C. Techniques

There are many complicated models developed for T2I task. In this project, a Dov2Vec Encoder-Decoder Architecture is used for text embedding, DC-GAN, WGAN and Stack GAN Architectures are used for image generation. So, the models used in this project differ in the second stage, generative models. Basically, followings are the stacked state of art deep learning models used: 1) Doc2Vec Encoder - DC-GAN, 2) Doc2Vec Encoder - WGAN, 3) Doc2Vec Encoder - Stack GAN, 4) Inception Score and FID Score (evaluating all models)

1, D. Dataset Description

In this project, Oxford 102 Flowers Dataset is used, which consists of 8189 images with 102 different classes. Each class has at least 40, maximum 258 images, each image has 10 captions with different number of words ranging from 80 to 230. [11] The dictionary obtained from the captions includes 5661 words. Oxford 102 dataset is preprocessed to make it applicable for our model. Fortunately, there is no erroneous images. Dataset is splitted into train, validation and test set, each of which contains 6590, 815 and 814 images respectively. So, training data set is 80 per cent of the dataset; validation and test dataset is 10 per cent of the dataset each. Figure 1 shows a random image from dataset with its first three captions.

Caption: "prominent purple stigma petals are white in color this flower has bright purple spiky petals and greenish sepals below them this flower has a row of white petals with the multi colored stamens and a pistil at the centre this flower is white and blue in color with petals that are oval shaped this flower has petals that are green with stringy purple stamen"

1, E. Performance Metrics

In the GANs losses calculated for both generator and discriminator. Since generator tries to fool the discriminator it should make its loss higher. On the other hand discriminator tries to catch generator's fake images therefore generator loss increases. Overall they should show oscillatory behaviour. This trends should be observed in the loss metrics. However, for generated images there is no ground truth mathematical metric that can measure exactness of the model. Since images are newly created it cannot be exactly comparable. One metric is human eye which is not a good metric except instant measurements while training. A proposed idea in the literature was used that is Inception Score. [12] It acts similar human eye.It checks weather it generates meaningful images. IS score used from a source code belongs to Shane T. Barratt written at 2017. [13] At the final stage Fréchet Inception Distance(FID) was used as performance metric. [14] FID metric calculated by using a python script belong to Maximilian Seitzer. [15]

FIG. 1: Sample Image and Its Captions

2. Methodology

2, A. Data Split

Dataset is splitted into training, validation and test sets. Randomly chosen 80 per cent of the data is used for training, the rest of the data is splitted equally for validation and test data.

2, B. Data Preprocessing

As mentioned before, there were no erroneous images in the dataset, however, all images don't have the same dimensions. Therefore, each image is normalized and re-scaled before they are inputted to GAN model. Since

image operations are costly, each images converted into (3,64,64) matrices. Except in Stage GAN, Images with a shape of (3,64,64) are inputted at the first stage and images with a shape of (3, 256,256) is inputted at the second stage.

Captions are processed to be inputted for text embedding as well. The vocabulary dictionary supplied with the dataset is not used in this paper since it couldn't properly split the words in the captions. Firstly, all captions are splitted by not only space but also special characters so that each key in the dictionary is really a word. Then, all the words are randomly indexed. Sentences are converted to index row matrices to be encoded by Doc2Vec model.

2, C. Text (Caption) Encoding

Word embedding is done by trying to predict the next word in the sentences. Even though the word vectors are randomly initialized, the embedding matrix learns the semantics in the sentences indirectly by trying to predict the upcoming words. A similar approach is used in paragraph or text embedding as well. Each paragraph (captions corresponding to each image) is represented by a unique vector D. Also, each word is represented by a unique vector W. Then, paragraph vector and word vectors (the number of word vectors to be used for prediction is a parameter) are combined to predict the next word in the sentence. [1] This allows the model to both use and combine the information from the words at the current sentence and information from the past sentences.

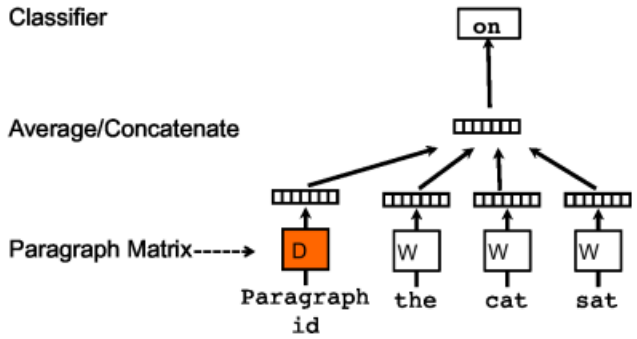


FIG. 2: Paragraph Vector and Word Vector Combination for Prediction [1]

As shown in figure 2, the paragraph vector can be regarded as an additional previous word. Thus, this model is called Distributed Memory Model of Paragraph Vectors (PV-DM). The advantage of using paragraph embedding rather than word embedding is that the number of dimensions decreased greatly compared to word embedding since we would need to concatenate dozens of words vector representation. Furthermore, it allows the sentiments and contexts in the previous sentences to be inherited to upcoming sentence(s).

Another method is used in Gensim Doc2Vec model is the reverse operation done in the figure 1. From paragraph vector, the model endeavours to find the next n words, which is very similar to Skip-gram model. This model is called Distributed Bag of Words version of Paragraph Vector (PV-DBOW). [1] This model doesn't yield effective results since this task is quite harder than the previous one and it uses the bag of words method. BoW method doesn't concentrate on the whole meaning of the sentences, rather it tries to capture and extract some significant words.

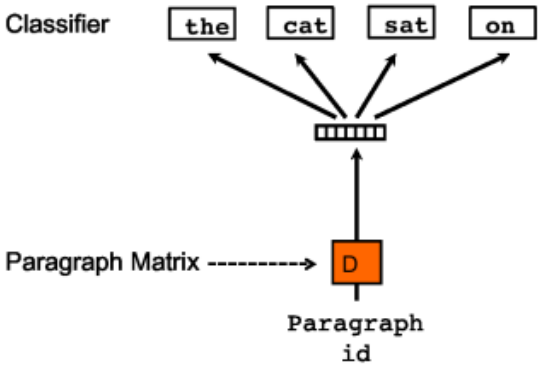


FIG. 3: Paragraph Vector for Word Vector Prediction [1]

2, D. Image Generation (Decoding)

3 different architecture was used.

2, D-1. DCGAN

Unlike generic GAN, DCGAN uses Convolutional Neural Networks (CNN) at both generator and discriminator. Fully connected layers are eliminated. While using DCGAN conditioning is added to firstly proposed version of it. [7] Text embedding is added as an addition channel on images for labeling on the discriminator side. In order to concatenate text embedding, resizing is required therefore it goes trough an multi layer perceptron. This process provides shape compatibility of image and text embedding. This fully connected layer is given up as learnable layer which is learnt to extract better embedding from given embedding.

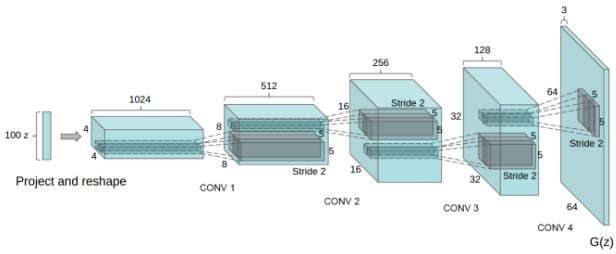


FIG. 4: DCGAN Generator Upsampling [7]

The most fundamental advancement of this net is its upsampling. In the upsampling (as seen above) convolution transpose is used. It starts with a Z latent whose size is a parameter. This latent is given as Gaussian noise for this paper's implementation. Again for dimensionality match text embedding passed trough FCNN and concated with Z latent. Therefore, generator network informed about text itself which it should generator the image. By this successive transpose convolution operations desired resolution image is attained.As suggested in DCGAN paper, between convolutions batch normalization layer and leaky ReLU actibation is used. It is stated that these makes training faster, stable and results better. Thus, batch normalization and leaky ReLU is used in this paper implementation. For generator output tanh is used to make it quicker to learn. For both discriminator and generator adam optimizer is used.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\big[logD(x)\big] + \mathbb{E}_{z\sim p_z(z)}\big[log(1-D(G(z)))\big]$$

Above equation is the DCGAN loss equation. D represents the discriminator and G represents the generator. The loss function is the same with first version of GAN. [6] The overall architecture given as figure 5.

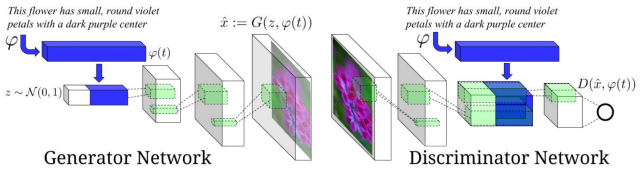


FIG. 5: DCGAN Overall Architecture [16]

2, D-2. WGAN

GAN model has some drawbacks such as unstability, divergence, and collapses in the model. [17] To eliminate these problem, more stable and convergent method WGAN is developed. Normal GAN models tries to minimize the Jensen-Shannon Divergence, whose formula as follows: [17], [18]

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2})$$

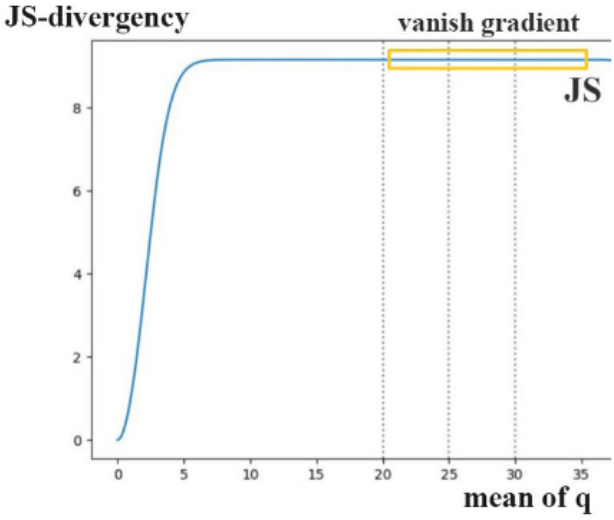


FIG. 6: Jensen-Shannon Divergence vs q Mean [18]

As seen figure 6, as the mean of q increases JS divergence starts to converge a fixed value, causing vanishing gradients. As a result, learning process is undermined.

On the other hand, a different distance metric is used called Earth-Mover (EM) distance. [18] Also, it includes the Kullback-Leibler (KL) Divergence term for cost calculation. Therefore, it minimizes a different loss function which has more penalty term compared to generic GAN.

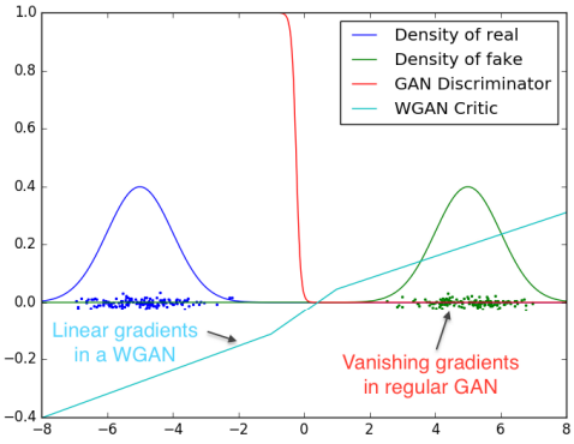


FIG. 7: WGAN and Generic GAN Gradients Comparison [18]

As seen from figure 7, the gradients of generic GAN

diminishes exponentially and vanishes at the end. On the other hand, gradients of WGAN are still linear, hence, improve the learning process.

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(\boldsymbol{x}^{(i)}\right)+\log\left(1-D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$	$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$
WGAN	$\nabla_{w}\frac{1}{m}\sum_{i=1}^{m}\left[f(x^{(i)})-f(G\left(z^{(i)}\right))\right]$	$\nabla_{\theta}\frac{1}{m}\sum_{i=1}^{m}f(G\left(z^{(i)}\right))$

FIG. 8: Lost Function Comparison [18]

Figure 8 represents the clear distinction on lost functions used in generic GAN and WGAN.

2, D-3. Stack GAN

Stack GAN is an architecture which designed specifically for text to image generation task. Its fundamental concept is stacking two different GAN in order to achieve high resolution image output correlated with the text.

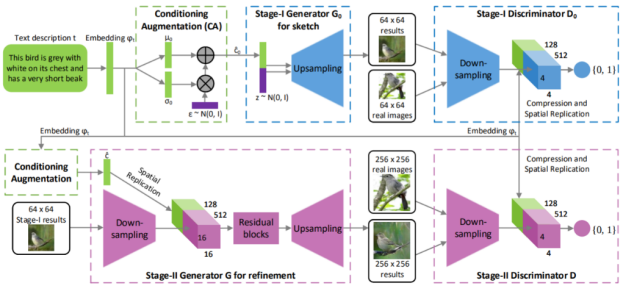


FIG. 9: Stack GAN Architecture [19]

As seen from the figure 9 that firstly text embedding and Gaussian noise is used and made conditional augmentation. According to Stack GAN paper this condition augmentation helps to increase diversity of the output image. [20] At the next step it concatenates and gives it to first stage generator. Then first stage generator generates an output which is fed to the second stage. Therefore first stage seem just an sketch of image and second stage makes is perfect. The GAN system is working similarly. Furthermore, the following loss functions are used at the discriminator and generator of stage 1 and stage 2.

$$\mathcal{L}_{D_0} = \mathbb{E}_{(I_0,t)\sim p_{data}}[\log D_0(I_0,\varphi_t)] + \mathbb{E}_{z\sim p_z, t\sim p_{data}}[\log(1 - D_0(G_0(z,\hat{c}_0),\varphi_t))],$$

$$\mathcal{L}_{G_0} = \mathbb{E}_{z\sim p_z, t\sim p_{data}}[\log(1 - D_0(G_0(z,\hat{c}_0),\varphi_t))] + \lambda D_{KL}(\mathcal{N}(\mu_0(\varphi_t),\Sigma_0(\varphi_t))\,||\,\mathcal{N}(0,I)),$$

FIG. 10: Stack GAN Stage 1 Loss Functions [20]

$$\mathcal{L}_{D} = \mathbb{E}_{(I,t)\sim p_{data}}[\log D(I,\varphi_t)] + \mathbb{E}_{s_0\sim p_{G_0}, t\sim p_{data}}[\log(1 - D(G(s_0,\hat{c}),\varphi_t))],$$

$$\mathcal{L}_{G} = \mathbb{E}_{s_0\sim p_{G_0}, t\sim p_{data}}[\log(1 - D(G(s_0,\hat{c}),\varphi_t))] + \lambda D_{KL}(\mathcal{N}(\mu(\varphi_t),\Sigma(\varphi_t))\,||\,\mathcal{N}(0,I)),$$

FIG. 11: Stack GAN Stage 2 Loss Functions [20]

In both cases stage 1 and stage 2, model tries to maximize discriminator loss whereas it tries to minimize the generator loss. KL loss is used at the generator part of both stage 1 and 2. This is used as a regularization term. Furthermore, gradient clipping is applied for gradient update to prevent gradients from vanishing or exploding, which contributes to improvement of learning process. Below figure demonstrates the effect of gradient clipping visually.

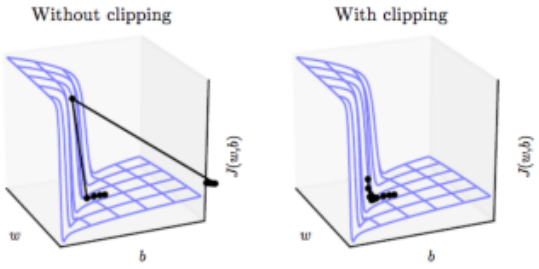


FIG. 12: Clipping Affect Gradients [18]

In this paper core ideas of Stack GAN is used as it is. Made proper arrangements to make it compatible with data and embedding that authors have. Since overall architecture is deep; first, stage 1 generator was trained with its own discriminator. Then Stage 1 parameters were frozen in order to not to ruin learnt ones. Thus, Stage 2 trained with its own by given input as output of the trained Stage1.

3. Results

3, -1. WGAN

For WGAN model, we couldn't achieve our expectations at the results. The figure below represents the IS mean score versus number of epochs. At first 20 epochs, Inception Score rapidly increases, the model learns very quickly how to generate an flower image with a low resolution. However, after 20th epoch, score doesn't increase, in fact, it has a decreasing trend.

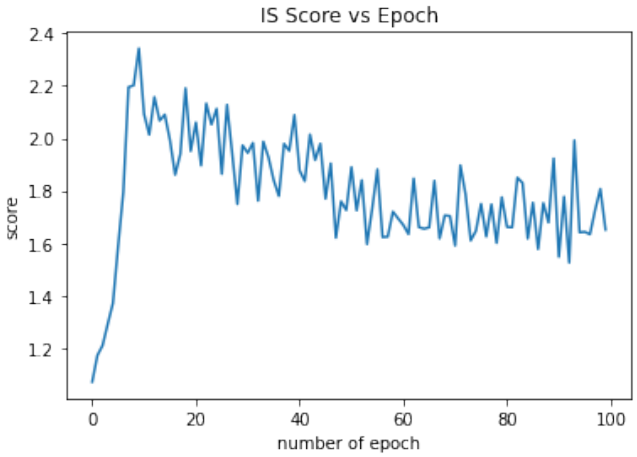


FIG. 13: Real Images]

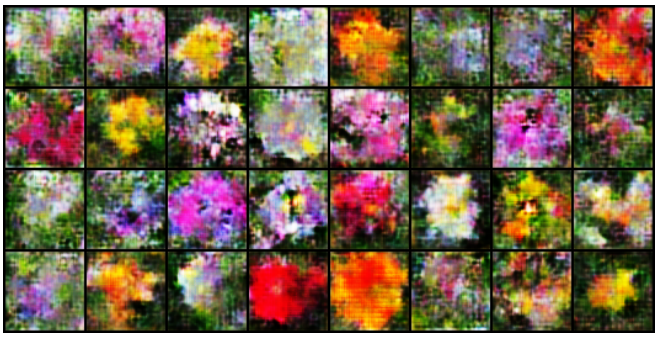


FIG. 14: Fake Images

As seen in figure 13 and 14, the model generally understands the label, the shape and the color of the generated flowers are similar to real images, however, their resolutions are quite low.

Caption : "this flower has petals that are purple with stringy stamen this flower is purple and white in color with petals that are oval shaped the petals of the flower are purple with green leaves and green stems a purple and yellow flower with a large stigma"

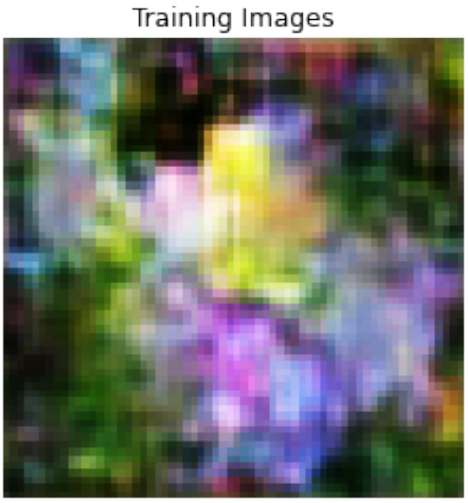


FIG. 15: Test Sample

In figure 15, a random test sample is drawn from test set. Again, the image represents the color, shape and their class but the resolution is low.

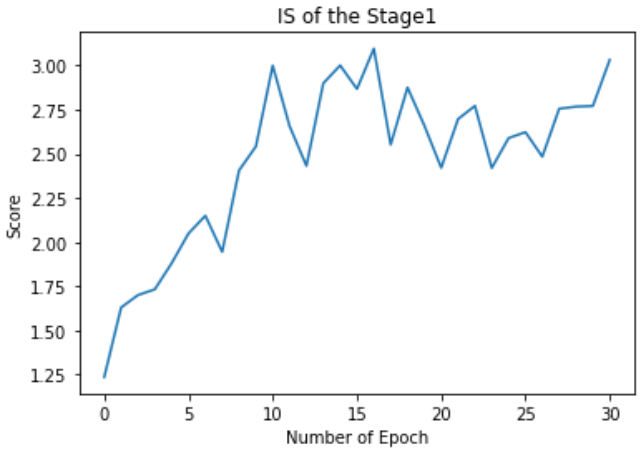
3, -2. Stack GAN

This was the most complex architecture of this paper. Stage1 is trained and tuned properly. Real images given as 64x64 and output of the generator was 64x64 as well. Results are okay.Its inception score is also okay. However Stage2 could not be tuned properly. It was built but due to the time constraints and resource constraints it did not trained and tuned therefore there is no result.

(a) Fake (b) Real

FIG. 16: Fake and Real Images for Stack GAN

As seen from the figure 16 that stage1 generator output images have high quality and flowers are clearly visible.This can be also verified with its increasing trend Inception Score graph that images have reasonable and meaningful. However in terms of labeling there are unwanted mismatches.



3, -3. DC GAN

The best results are obtained at model DC GAN. Actually, this model is more simplistic compared to other two models WGAN and Stack GAN, however, we couldn't tune the hyper parameters well in other models. At the training stage, the following images are generated for the given real images.

(a) Fake (b) Real

FIG. 17: Fake and Real Images for Training Batch at 19th Epoch

(a) Fake (b) Real

FIG. 18: Fake and Real Images for Training Batch at 42nd Epoch

(a) Fake (b) Real

FIG. 19: Fake and Real Images for Training Batch at The Last Epoch

Fake images generated by DC GAN starts to become more and more compatible with the corresponding real images as the number of epoch increases as seen in figures 16-18. Fake images are improved in terms of not only resolution but also label, it better detects the shape, color, class, type, etc. These results are attained after parameter tuning. This results reveals DCGAN's parameter sensitivity. Overall results at the high epochs are satisfactory.

Results for Test Data:

(a) Fake (b) Real

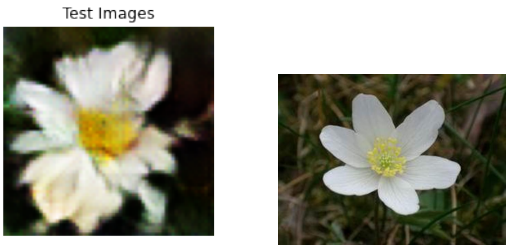
FIG. 20: Fake and Real Images for Sample Test Data

Caption: "this flower is pink in color with petals that are pointed on the ends the flower has petals that are bright pink and intertwined a hot pink sharp petaled flower "

(a) Fake (b) Real

FIG. 21: Fake and Real Images for Sample Test Data

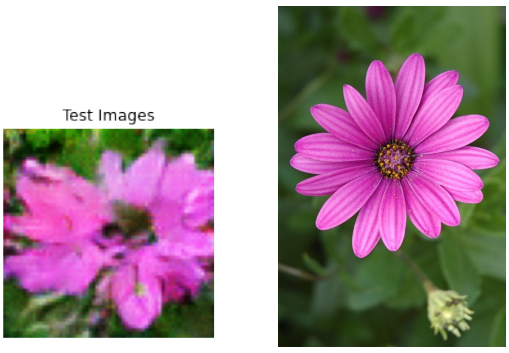
Caption: "this flower has multiple layers of yellow pointed petals with a large circular area of stamen in the center the petals of this flower are yellow with a short stigma "



(a) Fake (b) Real

FIG. 22: Fake and Real Images for Sample Test Data

Caption: "the petals of the flower are white in color and have a yellow center composed of anthers petals are oval in shape and are white in color stamens are many and are yellow in color"



(a) Fake (b) Real

FIG. 23: Fake and Real Images for Sample Test Data

Caption: "this vibrant flower has bright purple petals layered around the inner yellow stamen area the flower has petals that are oval and dark pink with yellow anthers"

As seen from the above figures 20-23, fake images generated by the output of discriminator of DC GAN are quite compatible in terms of color, shape, feature, class, etc. The resolution of the DC GAN is low, however, input images are generally in the shape of (500,500). Processing such images with their original sizes are extremely hard work to accomplish using regular computer CPU and GPU's. Since all images are downsampled, outputs have low resolution.

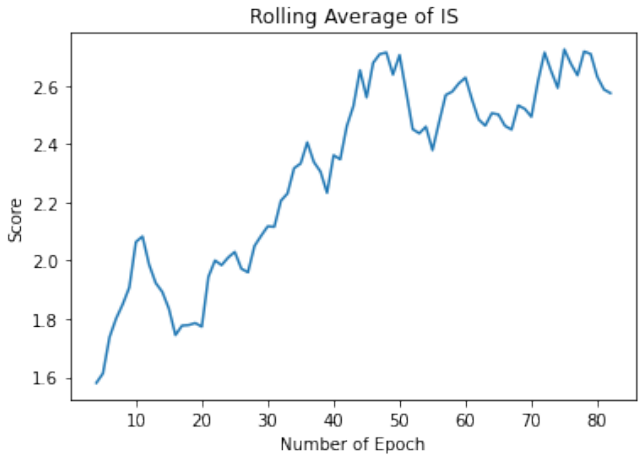


FIG. 24: IS Score of DC GAN vs Number of Epoch

Furthermore, IS mean score vs number of epochs plot demonstrates how the model is improved as the number of epochs increases.

4. Discussion and Comparison of Models

Firstly, text embedding is done using only Doc2Vec, which is not a high level model to extract features of such a high dimensional input (as mentioned captions corresponding to one image include 80 to 230 words). Since the usage of pretrained feature extraction vectors causes point deduction, we refrained from using them. If we were to train an RNN Encoder based text embedding matrix, the results definitely would be superior to current results.

In general real images scale to 64x64. Since CPU and GPU resource and time is considered it must be the way to go. The data set consists of images like 500x500 which is very high to process. The output images generated as 64x64. When its look as printed version seems not very good sometimes. However it mostly derives from its low resolution. Make the output higher resolution will again lead to high time and resource consumption thus, in this paper it was not prefered.

Our expectations about the results of WGAN were high since it has so many regularization term, and it uses different distance metric for minimization. Due to regularization, the results should have been more stable and convergent. Conversely, what we obtained from WGAN model is unstable and divergent results. Actually, output images are good at extracting the feature of captions

and being compatible with the specifications, however, images have really low resolutions. We think that this might stem from the fact that we couldn't properly tune the regularization terms so that they caused huge resolution loss.

Since Stack GAN is the one of the well known architecture for T2I task, expectation from the StackGAN was very very high. Building the architecture was harder from the others even though there is a very good reference of it. Controlling the architecture was hard. First, stage1 is trained and its results are excellent in terms of image quality. However, it did not well matched with label. It is suspected that conditional augmentation adds too much randomness to our labels. When this randomness added to relatively unstable embedding result became bad. Stage 2 could not trained. Because its training time was too much to handle in this papers time constraint. Also, it was required too much computational resources which authors urges to not train this stage2. However, as it is suggested on literature, it was seen that indeed Stack GAN promising architecture for advanced cases.

DC GAN has the best results. It is well tuned due to the its relatively simple architecture. Its training time was relatively low therefore more combinations are tried and attain better results and gone higher epoch numbers. CNNs are good way to go for image processing this is verified once more. Output qualities are also very good. Convolution Transpose layers seem done more effective job than others. Furthermore, its output well matched with its label. It extract correct information from labels.

References

[1] Le, Q. and Mikolov, T., 2022. Distributed Representations of Sentences and Documents. [online] arXiv.org. Available at: <https://doi.org/10.48550/arXiv.1405.4053> [Accessed 27 May 2022].

[2] Kosslyn, S., Ganis, G. and Thompson, W., 2022. Neural foundations of imagery.

[3] Medium. 2022. Feature Extraction Techniques. [online] Available at: <https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be> [Accessed 27 May 2022].

[4] DeepAI. 2022. Feature Extraction. [online] Available at: <https://deepai.org/machine-learning-glossary-and-terms/feature-extraction> [Accessed 27 May 2022].

[5] Medium. 2022. Use-cases of Google's Universal Sentence Encoder in Production. [online] Available at: <https://towardsdatascience.com> [Accessed 27 May 2022].

[6] Goodfellow, I., Pouget-Abadie, at all, 2022. Generative Adversarial Nets. [online] Proceedings.neurips.cc. Available at: <https://proceedings.neurips.cc/> [Accessed 27 May 2022].

[7] Radford, A., Metz, L. and Chintala, S., 2022. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. [online] arXiv.org. Available at: <https://arxiv.org/abs/1511.06434> [Accessed 27 May 2022].

[8] Arjovsky, M., Chintala, S. and Bottou, L., 2022. Wasserstein GAN. [online] arXiv.org. Available at: <https://arxiv.org/abs/1701.07875> [Accessed 27 May 2022].

[9] Paperswithcode.com. 2022. Papers with Code - Oxford 102 Flowers Benchmark (Text-to-Image Generation). [online] Available at: <https://paperswithcode.com/sota/text-to-image-generation-on-oxford-102> [Accessed 27 May 2022].

[10] Zhang, H., Xu, T., Li, H.,al all: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. [online] arXiv.org. Available at: <https://arxiv.org/abs/1612.03242>

[11] Robots.ox.ac.uk. 2022. Visual Geometry Group - University of Oxford. [online] Available at: <https://www.robots.ox.ac.uk/ vgg/data/flowers/102/> [Accessed 27 May 2022].

[12] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In NIPS, 2016. 2, 5

[13] GitHub. 2022. inception-score-pytorch/LICENSE.md at master · sbarratt/inception-score-pytorch. [online] Available at: <https://github.com/sbarratt/>

[14] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B. and Hochreiter, S., 2022. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. [online] arXiv.org. Available at: <https://arxiv.org/abs/1706.08500> [Accessed 27 May 2022].

[15] GitHub. 2022. GitHub - mseitzer/pytorch-fid: Compute FID scores with PyTorch.. [online] Available at: <https://github.com/mseitzer/pytorch-fid> [Accessed 27 May 2022].

[16] GitHub. 2022. GitHub - zsdonghao/text-to-image: Generative Adversarial Text to Image Synthesis / Please Star –>. [online] Available at: <https://github.com/zsdonghao/text-to-image> [Accessed 27 May 2022].

[17] Hui, J., 2022. GAN — Wasserstein GAN WGAN-GP. [online] Medium. Available at: <https://jonathan-hui.medium.com/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490> [Accessed 27 May 2022].

[18] Arjovsky, M., Chintala, S. and Bottou, L., 2022. Wasserstein GAN. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1701.07875v3.pdf> [Accessed 27 May 2022].

[19] GitHub. 2022. GitHub - savya08/StackGAN: A model for synthesising photo-realistic images given their textual descriptions. [online] Available at: <https://github.com/savya08/StackGAN> [Accessed 27 May 2022].

[20] Zhang, H., 2022. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1612.03242.pdf>.

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 21 20:09:54 2022

@author: melih
"""

# -*- coding: utf-8 -*-
"""StackGAN.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1PQLQZaaGx10NcFZHH4BfvMRfw5JEmehj

This code is inspired by code implementation of original paper
its GitHub Repo given below:
https://github.com/hanzhanggit/StackGAN-Pytorch
"""

from google.colab import drive
drive.mount('/content/drive')

# Commented out IPython magic to ensure Python compatibility.
from __future__ import print_function
import numpy as np
import time
import h5py
import matplotlib.pyplot as plt
import glob
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torchvision.utils as vutils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML
import os
from skimage import io
import re
from gensim.models import Word2Vec
from gensim.models import word2vec
from gensim.models import doc2vec
from gensim.models.doc2vec import TaggedDocument
import argparse
from PIL import Image
import random
import torch
import torch.nn as nn
from torch.nn import functional as F
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torchvision
import torch.utils.data
import torchvision.datasets as dset
```

```
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
import torchvision.utils as vutils
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML
from torch.utils.tensorboard import SummaryWriter
from torch.autograd import Variable
from torchvision.models.inception import inception_v3
from scipy.stats import entropy



# %matplotlib inline
"""# PREPROCESSING AND TEXT EMBEDDING STARTS"""

train_imid

train_url[25928]

train_url[[54015,54016,54018,54022,54024,82698,82705,82713,82732]]

list_index = []

for i in range(len(train_url)-82700):
    if i % 5000 == 0:
        print(i, len(list_index), len(list_index)/82273)

    st="/content/drive/MyDrive/img/pic"+str(i+1)+".jpg"

    try:
        with Image.open(st) as im:
            print(i)
            pass
    except:
        list_index.append(i)

print(len(list_index), i)

file_name = open("/content/drive/MyDrive/list_index.txt", "w")
for i in list_index:
    file_name.write(str(i) + "\n")
file_name.close()
print("max", max(list_index), "min", min(list_index))

count = 0
def image_reconstruct(error_list, train_ims, count):
    for i in range(1, train_ims.shape[0]+1):
        if (i-1) in error_list:
            count += 1
        else:
            string_os = "/content/drive/MyDrive/img/pic" + str(i) + ".jpg"
            string_dest =   "/content/drive/MyDrive/img/q" + str(i - count) + ".jpg"
            os.rename(string_os, string_dest)

    train_ims_new = np.delete(train_ims, error_list, axis = 0)
    return train_ims_new

error_list = np.loadtxt("/content/drive/MyDrive/list_index.txt", delimiter = "\n")
error_list = error_list - 1
```

```
train_ims_new = image_reconstruct(error_list, train_ims, count)

train_ims_new = np.delete(train_ims, error_list.astype(int), axis = 0)
a, b = train_ims_new.shape
print(a,b)
print(a+11689)

error_list = np.loadtxt("/content/drive/MyDrive/list_index.txt", delimiter = "\n")
error_list = error_list.astype(int) -1
error_list

# 8 dk sürüyor ortalama
def zero_based_name(num_exmple, error_list):
    count = 0
    for i in range(num_exmple):
        if i in error_list:
            count += 1
        else:
            zeros = 5 - len(str(i - count))
            new_file ="/content/drive/MyDrive/images/" + "0" * zeros + str(i - count)
            string_os = "/content/drive/MyDrive/images/pic" + str(i) + ".jpg"
            os.mkdir(new_file)
            shutil.move(string_os, new_file)

    return
zero_based_name(train_ims.shape[0]+1, error_list)

def reverse_image(error_list):
    count = 0
    for i in range(1, 54019):
        name = "/content/drive/MyDrive/img/p" + str(i - count) + ".jpg"
        if (i-1) in error_list:
            count += 1

        else:
            string_os = "/content/drive/MyDrive/img/p" + str(i - count) + ".jpg"
            string_dest =   "/content/drive/MyDrive/img/pic" + str(i) + ".jpg"
            os.rename(string_os, string_dest)

        if i % 1000 == 0 :
            print(i)

    return "baş"

error_list = np.loadtxt("/content/drive/MyDrive/list_index.txt", delimiter = "\n")
a = reverse_image(error_list)
print(a)

str_url = train_url.astype(str)
error_list = np.loadtxt("/content/drive/MyDrive/list_index.txt", delimiter = "\n")
error_list = error_list -1

t1 = time.time()
for i in range(82543, len(str_url)):

  if i not in error_list:
    with open('/content/drive/MyDrive/images/pic'+str(i)+'.jpg', 'wb') as handle:
        response = requests.get(str_url[i], stream=True)
        if not response.ok:
            pass
        for block in response.iter_content(1024):
```

```
            if not block:
                break
            handle.write(block)
  if i % 1000 == 0:
      t2 = time.time()
      print(i)
      print("elapsed time", t2-t1)

count = 0
def image_reconstruct(error_list, train_ims, count):
    for i in range(1, train_ims.shape[0]+1):
        if (i-1) in error_list:
            count += 1
        else:
            string_os = "/content/drive/MyDrive/images/pic" + str(i) + ".jpg"
            string_dest =   "/content/drive/MyDrive/images/p" + str(i - count) + ".jpg"
            os.rename(string_os, string_dest)

    train_ims_new = np.delete(train_ims, error_list, axis = 0)
    return train_ims_new

train_ims_new = image_reconstruct(error_list, train_ims, count)

a, b = train_ims_new.shape

def zero_based_name(num_exmple):
    for i in range(num_exmple):
        zeros = 5 - len(str(i))
        new_file ="/content/drive/MyDrive/images/" + "0" * zeros + str(i)
        string_os = "/content/drive/MyDrive/images/p" + str(i+1) + ".jpg"
        os.mkdir(new_file)
        shutil.move(string_os, new_file)

    return

zero_based_name(a)

vector = np.load("/content/drive/MyDrive/MoodleFiles/custom_train_ims2.npy")

from PIL import Image
with Image.open("/content/drive/MyDrive/images/08106/pic9484.jpg") as im:
    im.show()


all_text = glob.glob("D:\\Flowers\\Image-to-synthesis-flowers\\cvpr2016_flowers\\text_c10\\class
all_text.sort(key = lambda x: x.split("\\")[-1])


# In[15]:


image_names = glob.glob("D:\\Flowers\\Image-to-synthesis-flowers\\102flowers\\jpg\\*.jpg")
image_names.sort()


# In[16]:


with open('D:\\neural_project\\all_text', 'at') as f:
    for parag in all_text:
        f.write(open(parag).read().replace('\n', '') + '\n')
```

```
# In[17]:


parag4img = []
for i in range(len(all_text)):
    with open(all_text[i]) as f:
        lines = f.read()
    sentences = re.sub("[.,!?\\-]", ' ', lines.lower()).split('\n')
    parag4img.append(list((" ".join(sentences).split())))


# In[18]:


liste = []
for i in range(len(parag4img)):
    x = len(parag4img[i])
    liste.append(x)
print(max(liste))


# In[19]:


vocab_dict = {}
counter = 0
for i in range(len(parag4img)):
    for j in range(len(parag4img[i])):
        word = parag4img[i][j]
        if word not in vocab_dict:
            counter += 1
            vocab_dict[word] = counter

print(len(vocab_dict))


# In[29]:


string = ""
for i in range(len(parag4img[190])):
    string += parag4img[190][i]
    string += " "
string


# In[34]:


string = ""
for i in range(len(parag4img[5559])):
    string += parag4img[5559][i]
    string += " "
string


# # PREPROCESS

# In[266]:
```

```
data = np.zeros((len(parag4img), max(liste)))
for i in range(data.shape[0]):
    for j in range(len(parag4img[i])):
        word = parag4img[i][j]
        data[i][j] = vocab_dict[word]
data = data.astype(float)


# In[283]:


np.save("D:\\neural_project\\data.npy", data)



text_embed = np.load("/content/drive/MyDrive/DCGAN/text_embed.npy")
!cp -r '/content/drive/MyDrive/102flowers/jpg' photo
image_names = glob.glob("/content/photo/*.jpg")
image_names.sort()



def inception_score(imgs, cuda=True, batch_size=32, resize=False, splits=1):
    """Computes the inception score of the generated images imgs
    imgs -- Torch dataset of (3xHxW) numpy images normalized in the range [-1, 1]
    cuda -- whether or not to run on GPU
    batch_size -- batch size for feeding into Inception v3
    splits -- number of splits
    """
    N = len(imgs)

    assert batch_size > 0
    assert N > batch_size

    # Set up dtype
    if cuda:
        dtype = torch.cuda.FloatTensor
    else:
        if torch.cuda.is_available():
            print("WARNING: You have a CUDA device, so you should probably set cuda=True")
        dtype = torch.FloatTensor

    # Set up dataloader
    dataloader = torch.utils.data.DataLoader(imgs, batch_size=batch_size)

    # Load inception model
    inception_model = inception_v3(pretrained=True, transform_input=False).type(dtype)
    inception_model.eval();
    up = nn.Upsample(size=(299, 299), mode='bilinear').type(dtype)
    def get_pred(x):
        if resize:
            x = up(x)
        x = inception_model(x)
        return F.softmax(x).data.cpu().numpy()

    # Get predictions
    preds = np.zeros((N, 1000))

    for i, batch in enumerate(dataloader, 0):
```

```
        batch = batch.type(dtype)
        batchv = Variable(batch)
        batch_size_i = batch.size()[0]

        preds[i*batch_size:i*batch_size + batch_size_i] = get_pred(batchv)

    # Now compute the mean kl-div
    split_scores = []

    for k in range(splits):
        part = preds[k * (N // splits): (k+1) * (N // splits), :]
        py = np.mean(part, axis=0)
        scores = []
        for i in range(part.shape[0]):
            pyx = part[i, :]
            scores.append(entropy(pyx, py))
        split_scores.append(np.exp(np.mean(scores)))

    return np.mean(split_scores), np.std(split_scores)

class FlowersDataset(Dataset):

    def __init__(self, im_dir, text_embed, transform=None):

        self.im_dir = im_dir
        self.text_embed = text_embed
        self.transform = transform

    def __len__(self):
        return len(self.im_dir)

    def __getitem__(self, idx):

        if torch.is_tensor(idx):
            idx = idx.tolist()

        image = io.imread(self.im_dir[idx])
        text_vec = self.text_embed[idx]

        sample = {'image': image, 'text_vec': text_vec}

        if self.transform:
            image = self.transform(Image.open(self.im_dir[idx]))
            sample["image"] = image

        return image, text_vec

image_size = 64
batch_size = 32
ngpu = 1

dataset = FlowersDataset(im_dir = image_names,
                         text_embed = text_embed,
                         transform=transforms.Compose([
                            transforms.Resize(image_size),
                            transforms.CenterCrop(image_size),
                            transforms.ToTensor(),
                            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                        ]))
```

```
# %80 train %10 val %10 test split
train_size = int(0.8 * len(dataset))
double = len(dataset) - train_size
val_size = int(double/2)
test_size = double - val_size

train_set, double_set = torch.utils.data.random_split(dataset, [6560, 1629])
#val_set, test_set = torch.utils.data.random_split(double_set, [val_size, test_size])

#print(len(val_set))
print(train_size + val_size + test_size)

# Create the dataloader
dataloader = DataLoader(train_set, batch_size = batch_size,
                                   shuffle = True,
                                   num_workers = 2)

# Decide which device we want to run on
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cpu")
# Plot some training images
real_batch,label = next(iter(dataloader))
plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch.to(device)[:64], padding=2, normalize=True).

#Parameters
EMBED_DIM = 100
#EMBED_DIM = text_embed.shape[1]
CONDITIONAL_DIM = 128
NGF = 192 #Size of feature maps in generator
NZ = 100 # Size of z latent vector (i.e. size of generator input)
NDF = 96 # NUmber of discriminator feature Map
RES_NUM = 3 #Resnet Number


generator_lr = 2e-4
discriminator_lr = 2e-4
lr_decay_step = 20
batch_size = 32

""" STACK GAN STARTS"""

def conv3x3(in_planes, out_planes, stride=1):
    "3x3 convolution with padding"
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
                     padding=1, bias=False)

class D_GET_LOGITS(nn.Module):
    def __init__(self, ndf, nef, bcondition=True):
        super(D_GET_LOGITS, self).__init__()
        self.df_dim = ndf
        self.ef_dim = nef
        self.bcondition = bcondition
        if bcondition:
            self.outlogits = nn.Sequential(
                conv3x3(ndf * 8 + nef, ndf * 8),
                nn.BatchNorm2d(ndf * 8),
                nn.LeakyReLU(0.2, inplace=True),
                nn.Conv2d(ndf * 8, 1, kernel_size=4, stride=4),
                nn.Sigmoid())
```

```
        else:
            self.outlogits = nn.Sequential(
                nn.Conv2d(ndf * 8, 1, kernel_size=4, stride=4),
                nn.Sigmoid())

    def forward(self, h_code, c_code=None):
        # conditioning output
        if self.bcondition and c_code is not None:
            c_code = c_code.view(-1, self.ef_dim, 1, 1)
            c_code = c_code.repeat(1, 1, 4, 4)
            # state size (ngf+egf) x 4 x 4
            h_c_code = torch.cat((h_code, c_code), 1)
        else:
            h_c_code = h_code

        output = self.outlogits(h_c_code)
        return output.view(-1)

class ResBlock(nn.Module):
    def __init__(self, channel_num):
        super(ResBlock, self).__init__()
        self.block = nn.Sequential(
            nn.Conv2d(channel_num, channel_num, kernel_size=3, stride=1,
                      padding=1, bias=False),
            nn.BatchNorm2d(channel_num),
            nn.ReLU(True),
            nn.Conv2d(channel_num, channel_num, kernel_size=3, stride=1,
                      padding=1, bias=False),
            nn.BatchNorm2d(channel_num))
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        residual = x
        out = self.block(x)
        out += residual
        out = self.relu(out)
        return out


class CA(nn.Module):
    """
    Conditional Agumentation

    """

    def __init__(self):
        super().__init__()

        self.emd = EMBED_DIM
        self.gcd = CONDITIONAL_DIM
        self.FNN = nn.Sequential(
            nn.Linear(self.emd,self.gcd*2),
            nn.ReLU()


            )

    def forward(self,embed):
        post_embed = self.FNN(embed)
        mu = post_embed[:, self.gcd:]
        logvar = post_embed[:, :self.gcd]
```

```
        std = logvar.mul(0.5).exp_()
        eps = torch.cuda.FloatTensor(std.size()).normal_()
        eps = Variable(eps)
        cond = eps.mul(std).add_(mu)
        return cond, mu, logvar


class Stage1_Gen(nn.Module):
    """
    First stage generator

    """
    def __init__(self):
        super().__init__()
        self.ngf = NGF * 8
        self.gcd = CONDITIONAL_DIM
        self.nz = NZ
        self.cond_ag = CA()
        self.FNN = nn.Sequential(
            nn.Linear(self.nz + self.gcd, self.ngf * 4 * 4, bias=False),
            nn.BatchNorm1d(self.ngf * 4 * 4),
            nn.ReLU(True))

        self.conv3 = nn.Conv2d(self.ngf // 16, 3, kernel_size=3, stride=1,
                     padding=1, bias=False)

        # ngf x 4 x 4 -> ngf/2 x 8 x 8
        self.upsample1 = self.upblock(self.ngf, self.ngf // 2)
        # -> ngf/4 x 16 x 16
        self.upsample2 = self.upblock(self.ngf // 2, self.ngf // 4)
        # -> ngf/8 x 32 x 32
        self.upsample3 = self.upblock(self.ngf // 4, self.ngf // 8)
        # -> ngf/16 x 64 x 64
        self.upsample4 = self.upblock(self.ngf // 8, self.ngf // 16)
        # -> 3 x 64 x 64


    def upblock(self,in_feature,output_feature,stride = 1):
        net =  nn.Sequential(

            nn.Upsample(scale_factor=2, mode='nearest'),
            nn.Conv2d(in_feature, output_feature, kernel_size=3, stride=stride,
                     padding=1, bias=False),
            nn.BatchNorm2d(output_feature),
            nn.ReLU(True)
            )
        return net

    def forward( self , embed , noise):
        cond , mu, logvar = self.cond_ag(embed)
        concat =  torch.cat((noise, cond), 1)
        g_out = self.FNN(concat)

        #Turn to a 4D tensor like a 4 X 4 image
        g_out = g_out.view(-1, self.ngf, 4, 4)
        #Upsampling
        g_out = self.upsample1(g_out)
        g_out = self.upsample2(g_out)
        g_out = self.upsample3(g_out)
        g_out = self.upsample4(g_out)
        g_out = self.conv3(g_out)
```

```
        fake_img =  nn.Tanh()(g_out)
        return fake_img, mu, logvar



class Stage1_Dis(nn.Module):
    """
    First stage Discriminator

    """
    def __init__(self):
        super().__init__()
        self.ndf = NDF
        self.gcd = CONDITIONAL_DIM

        self.img2tensor = nn.Sequential(
            nn.Conv2d(3, self.ndf, 4 , 2,1,bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(self.ndf, self.ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(self.ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size (ndf*2) x 16 x 16
            nn.Conv2d(self.ndf*2, self.ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(self.ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size (ndf*4) x 8 x 8
            nn.Conv2d(self.ndf*4, self.ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(self.ndf * 8),
            # state size (ndf * 8) x 4 x 4)
            nn.LeakyReLU(0.2, inplace=True)
            )

        self.get_cond_logits = D_GET_LOGITS(self.ndf, self.gcd)
        self.get_uncond_logits = None

    def forward(self,image):
        img_tensor = self.img2tensor(image)
        return img_tensor

class Stage2_Gen(nn.Module):
    """
    Second stage Generator

    """
    def __init__(self,STAGE1_G):
        super().__init__()
        self.ngf = NGF
        self.gcd = CONDITIONAL_DIM
        self.nz = NZ
        self.Stage1_Gen = STAGE1_G
        self.cond_agu = CA()
        #fix parameters
        for param in self.Stage1_Gen.parameters():
            param.requires_grad = False


        self.encoder = nn.Sequential(
            nn.Conv2d(3, self.ngf, kernel_size=3, stride=1,
```

```
                padding=1, bias=False),

        nn.ReLU(True),
        nn.Conv2d(self.ngf, self.ngf * 2, 4, 2, 1, bias=False),
        nn.BatchNorm2d(self.ngf * 2),
        nn.ReLU(True),
        nn.Conv2d(self.ngf * 2, self.ngf * 4, 4, 2, 1, bias=False),
        nn.BatchNorm2d(self.ngf * 4),
        nn.ReLU(True))

    self.conv3 = nn.Conv2d(self.ngf//4, 3, kernel_size=3, stride=1,
                padding=1, bias=False)

    self.hr_joint = nn.Sequential(
        nn.Conv2d(self.gcd + self.ngf * 4, self.ngf * 4,kernel_size=3, stride=1,
                padding=1, bias=False),
        nn.BatchNorm2d(self.ngf * 4),
        nn.ReLU(True))


    self.residual = self.resnet(ResBlock, self.ngf * 4)

    # --> 2ngf x 32 x 32
    self.upsample1 = self.upBlock(self.ngf * 4, self.ngf * 2)
    # --> ngf x 64 x 64
    self.upsample2 = self.upBlock(self.ngf * 2, self.ngf)
    # --> ngf // 2 x 128 x 128
    self.upsample3 = self.upBlock(self.ngf, self.ngf // 2)
    # --> ngf // 4 x 256 x 256
    self.upsample4 = self.upBlock(self.ngf // 2, self.ngf // 4)
    # --> 3 x 256 x 256


def upBlock(self,in_feature,output_feature,stride = 1):
    net =  nn.Sequential(

        nn.Upsample(scale_factor=2, mode='nearest'),
        nn.Conv2d(in_feature, output_feature, kernel_size=3, stride=stride,
                padding=1, bias=False),
        nn.BatchNorm2d(output_feature),
        nn.ReLU(True)
        )
    return net

def resnet (self, Res, channel) :
    layers = []
    for i in range(RES_NUM):
        layers.append(Res(channel))
    return nn.Sequential(*layers)

def forward(self, embed, noise):

       with torch.no_grad():
            stage1_img, _, _ = self.Stage1_Gen(embed, noise)

       stage1_img = stage1_img.detach()
       encoded_img = self.encoder(stage1_img)

       c_code, mu, logvar = self.cond_agu(embed)
       c_code = c_code.view(-1, self.gcd, 1, 1)
       c_code = c_code.repeat(1, 1, 16, 16)
```

```
            i_c_code = torch.cat([encoded_img, c_code], 1)
            h_code = self.hr_joint(i_c_code)
            h_code = self.residual(h_code)


            h_code = self.upsample1(h_code)
            h_code = self.upsample2(h_code)
            h_code = self.upsample3(h_code)
            h_code = self.upsample4(h_code)
            h_code =self.conv3(h_code)
            fake_img = nn.Tanh()(h_code)

            return stage1_img, fake_img, mu, logvar


class Stage2_Dis(nn.Module):
    """
    Second stage Discriminator

    """
    def __init__(self):
        super().__init__()
        self.ndf = NDF
        self.gcd = CONDITIONAL_DIM
        self.define_module()

    def define_module(self):
        ndf, nef = self.ndf, self.gcd
        self.encode_img = nn.Sequential(
            nn.Conv2d(3, ndf, 4, 2, 1, bias=False),  # 128 * 128 * ndf
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),  # 64 * 64 * ndf * 2
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),  # 32 * 32 * ndf * 4
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),  # 16 * 16 * ndf * 8
            nn.Conv2d(ndf * 8, ndf * 16, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 16),
            nn.LeakyReLU(0.2, inplace=True),  # 8 * 8 * ndf * 16
            nn.Conv2d(ndf * 16, ndf * 32, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 32),
            nn.LeakyReLU(0.2, inplace=True),  # 4 * 4 * ndf * 32
            nn.Conv2d(ndf *32, ndf*16, kernel_size=3, stride=1,
                      padding=1, bias=False),
            nn.BatchNorm2d(ndf * 16),
            nn.LeakyReLU(0.2, inplace=True),   # 4 * 4 * ndf * 16
            nn.Conv2d(ndf *16, ndf*8, kernel_size=3, stride=1,
                      padding=1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True)   # 4 * 4 * ndf * 8
        )

        self.get_cond_logits = D_GET_LOGITS(ndf, nef, bcondition=True)
        self.get_uncond_logits = D_GET_LOGITS(ndf, nef, bcondition=False)
```

```
    def forward(self, image):
        img_embedding = self.encode_img(image)
        return img_embedding

def KL_loss(mu, logvar):
    # -0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD_element = mu.pow(2).add_(logvar.exp()).mul_(-1).add_(1).add_(logvar)
    KLD = torch.mean(KLD_element).mul_(-0.5)
    return KLD


def compute_discriminator_loss(netD, real_imgs, fake_imgs,
                               real_labels, fake_labels,
                               conditions):
    criterion = nn.BCELoss().to(device)
    batch_size = real_imgs.size(0)
    cond = conditions.detach()
    fake = fake_imgs.detach()
    real_features = netD(real_imgs)
    fake_features = netD(fake)
    # real pairs
    #inputs = (real_features, cond)

    real_logits = netD.get_cond_logits(real_features,cond)
    errD_real = criterion(real_logits, real_labels)
    # wrong pairs
    #inputs = (real_features[:(batch_size-1)], cond[1:])
    wrong_logits = \
        netD.get_cond_logits(real_features[:(batch_size-1)],cond[1:])
    errD_wrong = criterion(wrong_logits, fake_labels[1:])
    # fake pairs
    #inputs = (fake_features, cond)
    fake_logits = netD.get_cond_logits(fake_features,cond)
    errD_fake = criterion(fake_logits, fake_labels)

    if netD.get_uncond_logits is not None:
        real_logits = \
            netD.get_uncond_logits(real_features)
        fake_logits = \
           netD.get_uncond_logits(fake_features)
        uncond_errD_real = criterion(real_logits, real_labels)
        uncond_errD_fake = criterion(fake_logits, fake_labels)
        #
        errD = ((errD_real + uncond_errD_real) / 2. +
                (errD_fake + errD_wrong + uncond_errD_fake) / 3.)
        errD_real = (errD_real + uncond_errD_real) / 2.
        errD_fake = (errD_fake + uncond_errD_fake) / 2.
    else:
        errD = errD_real + (errD_fake + errD_wrong) * 0.5
    return errD, errD_real.data, errD_wrong.data, errD_fake.data


def compute_generator_loss(netD, fake_imgs, real_labels, conditions):
    criterion = nn.BCELoss().to(device)
    cond = conditions.detach()
    fake_features = netD(fake_imgs)
    inputs = (fake_features, cond)
    fake_logits = netD.get_cond_logits(fake_features,cond)
    errD_fake = criterion(fake_logits, real_labels)
    if netD.get_uncond_logits is not None:
        fake_logits = \
```

```
            netD.get_uncond_logits(fake_features)
        uncond_errD_fake = criterion(fake_logits, real_labels)
        errD_fake += uncond_errD_fake
    return errD_fake


##############################
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        m.weight.data.normal_(0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)
    elif classname.find('Linear') != -1:
        m.weight.data.normal_(0.0, 0.02)
        if m.bias is not None:
            m.bias.data.fill_(0.0)

device = torch.device("cuda:0" if (torch.cuda.is_available() and 1 > 0) else "cpu")

netG = Stage1_Gen()
netG.apply(weights_init)
netG = netG.cuda()
print(netG)
netD = Stage1_Dis().to(device)
netD.apply(weights_init)
netD = netD.cuda()
print(netD)

# Commented out IPython magic to ensure Python compatibility.
IS_score_mean=[]
IS_score_std=[]
IS_mean= 0
IS_std = 0

err_gen = []
err_dis = []


fixed_noise = \
            Variable(torch.FloatTensor(batch_size, NZ).normal_(0, 1),
                     volatile=True).to(device)


optimizerD = optim.Adam(netD.parameters(),
                        lr= discriminator_lr, betas=(0.5, 0.999))
netG_para = []
for p in netG.parameters():
    if p.requires_grad:
        netG_para.append(p)

optimizerG = optim.Adam(netG_para,
                        lr=generator_lr,
                        betas=(0.5, 0.999))
count = 0

max_epoch = 60
KL_COEFF = 10

writer_real = SummaryWriter(f"/content/sample_data/logs99/real")
```

```
writer_fake = SummaryWriter(f"/content/sample_data/logs99/fake")
writer_fake_stage1 = SummaryWriter(f"/content/sample_data/logs99/fake_stage1")
t1=time.time()

for epoch in range(max_epoch):
            start_t = time.time()
            if epoch % lr_decay_step == 0 and epoch > 0:
                generator_lr *= 0.5
                for param_group in optimizerG.param_groups:
                    param_group['lr'] = generator_lr
                discriminator_lr *= 0.5
                for param_group in optimizerD.param_groups:
                    param_group['lr'] = discriminator_lr

            for i, data in enumerate(dataloader, 0):

                # (1) Prepare training data

                real_img_cpu, txt_embedding = data
                real_imgs = Variable(real_img_cpu)
                txt_embedding = Variable(txt_embedding)

                real_imgs = real_imgs.cuda()
                txt_embedding = txt_embedding.float()
                txt_embedding = txt_embedding.cuda()

                bsize = real_imgs.shape[0]
                real_labels = Variable(torch.FloatTensor(bsize).fill_(1)).to(device)
                fake_labels = Variable(torch.FloatTensor(bsize).fill_(0)).to(device)
                noise = Variable(torch.FloatTensor(bsize, NZ))

                # (2) Generate fake images

                noise = noise.data.normal_(0, 1).to(device)


                fake_imgs, mu, logvar = netG(noise, txt_embedding)
                #stack_img, fake_imgs, mu, logvar = netG(txt_embedding,noise)


                # (3) Update D network

                netD.zero_grad()
                errD, errD_real, errD_wrong, errD_fake = \
                    compute_discriminator_loss(netD, real_imgs, fake_imgs,
                                               real_labels, fake_labels,
                                               mu)
                errD.backward()
                optimizerD.step()
                               # (2) Update G network

                netG.zero_grad()
                errG = compute_generator_loss(netD, fake_imgs,
                                              real_labels, mu)
                kl_loss = KL_loss(mu, logvar)
                errG_total = errG + kl_loss * KL_COEFF
                errG_total.backward()
                optimizerG.step()
                if (i % 200 == 0) or ((epoch == max_epoch-1) and (i == len(dataloader)-1)):
                    print('[%d/%d][%d/%d]\tLoss_D: %.4f\tLoss_G: %.4f'
#                           % (epoch, max_epoch, i, len(dataloader),
```

```
                    errD.item(), errG.item()))
                t2 = time.time()
                hours = (t2 -  t1) // 60 // 60
                mins = (t2 -t1) // 60 - hours * 60
                secs = (t2 -t1) - mins * 60 - 3600 * hours

                err_gen.append(errG)
                err_dis.append(errD)

                print("Elapsed time : {} hours {} mins {} secs ".format(int(hours), int(mins
                with torch.no_grad():
                    fake,_,__ = netG(fixed_noise, txt_embedding)
                    #stage1_image, fake,_,__ = netG(fixed_noise, txt_embedding)
                    fake = fake.detach().cpu()
                    img_grid_real = torchvision.utils.make_grid(real_imgs[:32], normalize=Tr
                    img_grid_fake = torchvision.utils.make_grid(fake[:32], normalize=True)
                    #img_grid_fake_stage1 = torchvision.utils.make_grid(stage1_image[:32], n

                    IS_mean,IS_std=inception_score(fake,batch_size=int(batch_size/2),resize=
                    IS_score_mean.append(IS_mean)
                    IS_score_std.append(IS_std)

                    writer_real.add_image("Real", img_grid_real, global_step=count)
                    writer_fake.add_image("Fake", img_grid_fake, global_step=count)
                    #writer_fake.add_image("Stage1", img_grid_fake_stage1, global_step=count

        # Save Losses for plotting later

                count = count + 1

            print(
            f"Epoch [{epoch}/{max_epoch}]  \
                Total Error: {errG_total:.4f},IS mean: {IS_mean:.4f},IS std: {IS_std:.4f}"
        )

plt.plot(IS_score_mean[::2])

IS_score_np = np.array(IS_score_mean)
errd_np = np.array(err_dis)
errg_np = np.array(err_gen.cpu())
np.save("/content/drive/MyDrive/stackgan_LastDance/errdisc_stag1.npy", errd_np)
np.save("/content/drive/MyDrive/stackgan_LastDance/errgen_stae1.npy", errg_np)
np.save("/content/drive/MyDrive/stackgan_LastDance/IS_score_np_stage1.npy", IS_score_np)

!cp -r '/content/sample_data/logs99' '/content/drive/MyDrive/stackgan_LastDance'

torch.save(netG, "/content/drive/MyDrive/stackgan_LastDance/netGmodel_gece3")

""" STAGE 1 TRANINIG HAS ENDED """

# Commented out IPython magic to ensure Python compatibility.
# %load_ext tensorboard
# %tensorboard --logdir="/content/sample_data/logs99"

"""**STAGE 2 TRANININ HAS STARTED**"""

image_size = 256
batch_size = 32
ngpu = 1

dataset = FlowersDataset(im_dir = image_names,
```

```
                            text_embed = text_embed,
                            transform=transforms.Compose([
                               transforms.Resize(image_size),
                               transforms.CenterCrop(image_size),
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                           ]))


# %80 train %10 val %10 test split
train_size = int(0.8 * len(dataset))
double = len(dataset) - train_size
val_size = int(double/2)
test_size = double - val_size

train_set, double_set = torch.utils.data.random_split(dataset, [6560, 1629])
#val_set, test_set = torch.utils.data.random_split(double_set, [val_size, test_size])

#print(len(val_set))
print(train_size + val_size + test_size)

# Create the dataloader
dataloader = DataLoader(train_set, batch_size = batch_size,
                                    shuffle = True,
                                    num_workers = 2)

# Decide which device we want to run on
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cpu")
# Plot some training images
real_batch,label = next(iter(dataloader))
plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch.to(device)[:64], padding=2, normalize=True).

foo = Stage1_Gen()
foo1 = Stage2_Gen(netG).to(device)

netG2 = Stage2_Gen(netG).to(device)
#netG2.apply(weights_init)
#netG2 = Stage2_Gen(netG).to(device)
print(netG2)
netD2 = Stage2_Dis()
netD2.apply(weights_init)
print(netD2)
netG2.cuda()
netD2.cuda()

# Commented out IPython magic to ensure Python compatibility.
IS_score_mean=[]
IS_score_std=[]
IS_mean= 0
IS_std = 0

disc_err = []
gen_err = []

fixed_noise = \
            Variable(torch.FloatTensor(batch_size, NZ).normal_(0, 1),
                     volatile=True).to(device)
```

```
optimizerD = optim.Adam(netD2.parameters(),
                        lr= discriminator_lr, betas=(0.5, 0.999))
netG_para = []
for p in netG2.parameters():
    if p.requires_grad:
        netG_para.append(p)

optimizerG = optim.Adam(netG_para,
                        lr=generator_lr,
                        betas=(0.5, 0.999))
count = 0

max_epoch = 25
KL_COEFF = 10

writer_real = SummaryWriter(f"/content/sample_data/logs100/real")
writer_fake = SummaryWriter(f"/content/sample_data/logs100/fake")
writer_fake_stage1 = SummaryWriter(f"/content/sample_data/logs100/fake_stage1")
t1=time.time()

for epoch in range(max_epoch):
            start_t = time.time()
            if epoch % lr_decay_step == 0 and epoch > 0:
                generator_lr *= 0.5
                for param_group in optimizerG.param_groups:
                    param_group['lr'] = generator_lr
                discriminator_lr *= 0.5
                for param_group in optimizerD.param_groups:
                    param_group['lr'] = discriminator_lr

            for i, data in enumerate(dataloader, 0):

                real_img_cpu, txt_embedding = data
                real_imgs = Variable(real_img_cpu)
                txt_embedding = Variable(txt_embedding)

                real_imgs = real_imgs.cuda()
                txt_embedding = txt_embedding.float()
                txt_embedding = txt_embedding.cuda()

                bsize = real_imgs.shape[0]
                real_labels = Variable(torch.FloatTensor(bsize).fill_(1)).to(device)
                fake_labels = Variable(torch.FloatTensor(bsize).fill_(0)).to(device)
                noise = Variable(torch.FloatTensor(bsize, NZ))

                noise = noise.data.normal_(0, 1).to(device)
                #inputs = (txt_embedding, noise)
                stack_img, fake_imgs, mu, logvar = netG2(txt_embedding,noise)


                netD2.zero_grad()
                errD, errD_real, errD_wrong, errD_fake = \
                    compute_discriminator_loss(netD2, real_imgs, fake_imgs,
                                               real_labels, fake_labels,
                                               mu)
                errD.backward()
                optimizerD.step()

                netG2.zero_grad()
                errG = compute_generator_loss(netD2, fake_imgs,
```

```
                                            real_labels, mu)
                kl_loss = KL_loss(mu, logvar)
                errG_total = errG + kl_loss * KL_COEFF
                errG_total.backward()
                optimizerG.step()
                if (i % 200 == 0) or ((epoch == max_epoch-1) and (i == len(dataloader)-1)):
                    print('[%d/%d][%d/%d]\tLoss_D: %.4f\tLoss_G: %.4f'
#                           % (epoch, max_epoch, i, len(dataloader),
                              errD.item(), errG.item()))
                    t2 = time.time()
                    hours = (t2 -  t1) // 60 // 60
                    mins = (t2 -t1) // 60 - hours * 60
                    secs = (t2 -t1) - mins * 60 - 3600 * hours

                    disc_err.append(errD.item())
                    gen_err.append(errG.item())

                    print("Elapsed time : {} hours {} mins {} secs ".format(int(hours), int(mins
                    with torch.no_grad():
                        stage1_image, fake,_,__ = netG2(fixed_noise, txt_embedding)
                        fake = fake.detach().cpu()
                        img_grid_real = torchvision.utils.make_grid(real_imgs[:32], normalize=Tr
                        img_grid_fake = torchvision.utils.make_grid(fake[:32], normalize=True)
                        img_grid_fake_stage1 = torchvision.utils.make_grid(stage1_image[:32], no

                        IS_mean,IS_std=inception_score(fake,batch_size=int(batch_size/2),resize=
                        IS_score_mean.append(IS_mean)
                        IS_score_std.append(IS_std)

                        writer_real.add_image("Real", img_grid_real, global_step=count)
                        writer_fake.add_image("Fake", img_grid_fake, global_step=count)
                        writer_fake.add_image("Stage1", img_grid_fake_stage1, global_step=count)


            # Save Losses for plotting later

                    count = count + 1

                print(
                f"Epoch [{epoch}/{max_epoch}]  \
                    Total Error: {errG_total:.4f},IS mean: {IS_mean:.4f},IS std: {IS_std:.4f}"
            )

    IS_score_np = np.array(IS_score_mean)
    errd_np = np.array(disc_err)
    errg_np = np.array(gen_err)
    np.save("/content/drive/MyDrive/stackgan_LastDance/errdisc_stag2.npy", errd_np)
    np.save("/content/drive/MyDrive/stackgan_LastDance/errgen_stae2.npy", errg_np)
    np.save("/content/drive/MyDrive/stackgan_LastDance/IS_score_np_stage2.npy", IS_score_np)
    !cp -r '/content/sample_data/logs100' '/content/drive/MyDrive/stackgan_LastDance'

    # Commented out IPython magic to ensure Python compatibility.
    # %tensorboard --logdir="/content/sample_data/logs100"

    plt.plot(IS_score_mean)


    """ WGAN HAS STARTED """

    """WGAN.ipynb
```

```
Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1J8dDYcbwj0VaeiMY20U5FNKsCLdc1w6q

This code inspired from a GitHub repo belong to Aladdin Persson:
https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/GANs/4.%20W
"""

class Discriminator(nn.Module):

    def __init__(self, channels_img, features_d):
        super(Discriminator, self).__init__()
        self.disc = nn.Sequential(
            # input: N x channels_img x 64 x 64
            nn.Conv2d(channels_img+1, features_d, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2),
            # _block(in_channels, out_channels, kernel_size, stride, padding)
            self._block(features_d, features_d * 2, 4, 2, 1),
            self._block(features_d * 2, features_d * 4, 4, 2, 1),
            self._block(features_d * 4, features_d * 8, 4, 2, 1),
            # After all _block img output is 4x4 (Conv2d below makes into 1x1)
            nn.Conv2d(features_d * 8, 1, kernel_size=4, stride=2, padding=0),
        )

        self.text_FNN = nn.Sequential(
            nn.Linear(EMBED_SHAPE,64*64),
            nn.BatchNorm1d(64*64),


        )



    def _block(self, in_channels, out_channels, kernel_size, stride, padding):
        return nn.Sequential(
            nn.Conv2d(
                in_channels, out_channels, kernel_size, stride, padding, bias=False,
            ),
            nn.InstanceNorm2d(out_channels, affine=True),
            nn.LeakyReLU(0.2),
        )

    def forward(self, x,embed):
        dense_embed = self.text_FNN(embed)
        dense_embed = dense_embed.view(-1 , 1, 64, 64)
        disc_input = torch.cat([x,dense_embed],axis = 1)
        return self.disc(disc_input)


class Generator(nn.Module):
    def __init__(self, channels_noise, channels_img, features_g,embed_shape):
        super(Generator, self).__init__()
        self.net = nn.Sequential(
            # Input: N x channels_noise x 1 x 1
            self._block(channels_noise + embed_shape , features_g * 16, 4, 1, 0),  # img: 4x4
            self._block(features_g * 16, features_g * 8, 4, 2, 1),  # img: 8x8
            self._block(features_g * 8, features_g * 4, 4, 2, 1),  # img: 16x16
            self._block(features_g * 4, features_g * 2, 4, 2, 1),  # img: 32x32
            nn.ConvTranspose2d(
                features_g * 2, channels_img, kernel_size=4, stride=2, padding=1
```

```
            ),
            # Output: N x channels_img x 64 x 64
            nn.Tanh(),
        )



    def _block(self, in_channels, out_channels, kernel_size, stride, padding):
        return nn.Sequential(
            nn.ConvTranspose2d(
                in_channels, out_channels, kernel_size, stride, padding, bias=False,
            ),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
        )

    def forward(self, x,embed):
        embed = embed.view(-1 , embed.shape[1],1,1)
        gen_input = torch.cat([x,embed],axis= 1)

        return self.net(gen_input)


def initialize_weights(model):
    # Initializes weights according to the DCGAN paper
    for m in model.modules():
        if isinstance(m, (nn.Conv2d, nn.ConvTranspose2d, nn.BatchNorm2d)):
            nn.init.normal_(m.weight.data, 0.0, 0.02)


def test():
    N, in_channels, H, W = 8, 3, 64, 64
    noise_dim = 100
    x = torch.randn((N, in_channels, H, W))
    disc = Discriminator(in_channels, 8)
    assert disc(x).shape == (N, 1, 1, 1), "Discriminator test failed"
    gen = Generator(noise_dim, in_channels, 8)
    z = torch.randn((N, noise_dim, 1, 1))
    assert gen(z).shape == (N, in_channels, H, W), "Generator test failed"

def gradient_penalty(critic, train_ims,real,fake, device="cpu"):
    BATCH_SIZE, C, H, W = real.shape
    alpha = torch.rand((BATCH_SIZE, 1, 1, 1)).repeat(1, C, H, W).to(device)
    interpolated_images = real * alpha + fake * (1 - alpha)

    # Calculate critic scores
    mixed_scores = critic(interpolated_images,train_ims)

    # Take the gradient of the scores with respect to the images
    gradient = torch.autograd.grad(
        inputs=interpolated_images,
        outputs=mixed_scores,
        grad_outputs=torch.ones_like(mixed_scores),
        create_graph=True,
        retain_graph=True,
    )[0]
    gradient = gradient.view(gradient.shape[0], -1)
    gradient_norm = gradient.norm(2, dim=1)
    gradient_penalty = torch.mean((gradient_norm - 1) ** 2)
    return gradient_penalty
```

```
def save_checkpoint(state, filename="celeba_wgan_gp.pth.tar"):
    print("=> Saving checkpoint")
    torch.save(state, filename)


def load_checkpoint(checkpoint, gen, disc):
    print("=> Loading checkpoint")
    gen.load_state_dict(checkpoint['gen'])
    disc.load_state_dict(checkpoint['disc'])

device = "cuda" if torch.cuda.is_available() else "cpu"

LEARNING_RATE = 1e-4
BATCH_SIZE = 64
IMAGE_SIZE = 64
CHANNELS_IMG = 3
Z_DIM = 100
NUM_EPOCHS = 100
FEATURES_CRITIC = 16
FEATURES_GEN = 16
CRITIC_ITERATIONS = 3
LAMBDA_GP = 7



workers = 2
ngpu = 1

text_embed = np.load("/content/drive/MyDrive/DCGAN/text_embed.npy")
!cp -r '/content/drive/MyDrive/102flowers/jpg' photo
image_names = glob.glob("/content/photo/*.jpg")
image_names.sort()
EMBED_SHAPE = text_embed.shape[1]

class FlowersDataset(Dataset):

    def __init__(self, im_dir, text_embed, transform=None):

        self.im_dir = im_dir
        self.text_embed = text_embed
        self.transform = transform

    def __len__(self):
        return len(self.im_dir)

    def __getitem__(self, idx):

        if torch.is_tensor(idx):
            idx = idx.tolist()

        image = io.imread(self.im_dir[idx])
        text_vec = self.text_embed[idx]

        sample = {'image': image, 'text_vec': text_vec}

        if self.transform:
            image = self.transform(Image.open(self.im_dir[idx]))
            sample["image"] = image

        return image, text_vec
```

```
dataset = FlowersDataset(im_dir = image_names,
                            text_embed = text_embed,
                            transform=transforms.Compose([
                               transforms.Resize(IMAGE_SIZE),
                               transforms.CenterCrop(IMAGE_SIZE),
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                           ]))


# %80 train %10 val %10 test split
train_size = int(0.8 * len(dataset))
double = len(dataset) - train_size
val_size = int(double/2)
test_size = double - val_size

train_set, double_set = torch.utils.data.random_split(dataset, [train_size, double])
val_set, test_set = torch.utils.data.random_split(double_set, [val_size, test_size])

print(len(val_set))
print(train_size + val_size + test_size)

# Create the dataloader
dataloader = DataLoader(train_set, batch_size = BATCH_SIZE,
                                   shuffle = True,
                                   num_workers = 2)

real_batch,label = next(iter(dataloader))
plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch.to(device)[:64], padding=2, normalize=True).

from torchvision.models.inception import inception_v3
from scipy.stats import entropy


gen = Generator(Z_DIM, CHANNELS_IMG, FEATURES_GEN,EMBED_SHAPE).to(device)
critic = Discriminator(CHANNELS_IMG, FEATURES_CRITIC).to(device)
initialize_weights(gen)
initialize_weights(critic)

# initializate optimizer
opt_gen = optim.Adam(gen.parameters(), lr=LEARNING_RATE, betas=(0.0, 0.9))
opt_critic = optim.Adam(critic.parameters(), lr=LEARNING_RATE, betas=(0.0, 0.9))

# for tensorboard plotting
fixed_noise = torch.randn(BATCH_SIZE, Z_DIM, 1, 1).to(device)
writer_real = SummaryWriter(f"logs5/GAN_MNIST/real")
writer_fake = SummaryWriter(f"logs5/GAN_MNIST/fake")
step = 0

gen.train()
critic.train()

IS_score_mean=[]
IS_score_std=[]
IS_mean= 0
IS_std = 0
```

```
t1=time.time()

for epoch in range(NUM_EPOCHS):



    for batch_idx, data_pair in enumerate(dataloader):

        img, embed = data_pair
        embed = embed.float()
        embed = embed.to(device)


        real = img.to(device)

        cur_batch_size = real.shape[0]

        # Train Critic: max E[critic(real)] - E[critic(fake)]
        # equivalent to minimizing the negative of that
        for _ in range(CRITIC_ITERATIONS):
            noise = torch.randn(cur_batch_size, Z_DIM, 1, 1).to(device)
            fake = gen(noise,embed)
            critic_real = critic(real,embed).reshape(-1)
            critic_fake = critic(fake,embed).reshape(-1)
            gp = gradient_penalty(critic, embed,real, fake, device=device)
            loss_critic = (
                -(torch.mean(critic_real) - torch.mean(critic_fake)) + LAMBDA_GP * gp
            )
            critic.zero_grad()
            loss_critic.backward(retain_graph=True)
            opt_critic.step()

        # Train Generator: max E[critic(gen_fake)] <-> min -E[critic(gen_fake)]
        gen_fake = critic(fake,embed).reshape(-1)
        loss_gen = -torch.mean(gen_fake)
        gen.zero_grad()
        loss_gen.backward()
        opt_gen.step()

        # Print losses occasionally and print to tensorboard

        if batch_idx % 100==0 and batch_idx > 0:

            with torch.no_grad():
                fake = gen(fixed_noise,embed)
                # take out (up to) 32 examples
                img_grid_real = torchvision.utils.make_grid(real[:32], normalize=True)
                img_grid_fake = torchvision.utils.make_grid(fake[:32], normalize=True)

                writer_real.add_image("Real", img_grid_real, global_step=step)
                writer_fake.add_image("Fake", img_grid_fake, global_step=step)

                step += 1

                IS_mean,IS_std=inception_score(fake,batch_size=int(BATCH_SIZE/2),resize=True)
                IS_score_mean.append(IS_mean)
                IS_score_std.append(IS_std)
```

```
    print(
            f"Epoch [{epoch}/{NUM_EPOCHS}]  \
                Loss D: {loss_critic:.4f}, loss G: {loss_gen:.4f}, IS mean: {IS_mean:.4f},IS st
        )
    t2 = time.time()
    hours = (t2 -  t1) // 60 // 60
    mins = (t2 -t1) // 60 - hours * 60
    secs = (t2 -t1) - mins * 60 - 3600 * hours
    print("Elapsed time : {} hours {} mins {} secs ".format(int(hours), int(mins), int(secs)))

torch.save(gen, '/content/drive/MyDrive/WGANmodel/gen')



# Commented out IPython magic to ensure Python compatibility.
# %load_ext tensorboard

# Commented out IPython magic to ensure Python compatibility.
# %tensorboard --logdir="/content/logs5"

plt.plot(IS_score_mean)
plt.xlabel("number of epoch")
plt.ylabel("score")
plt.title("IS Score vs Epoch")

type(text_embed[128:192])

noisee =  torch.randn(64, Z_DIM, 1, 1).to(device).float()

gen.eval()
with torch.no_grad():
    x = gen(noisee,torch.from_numpy(text_embed[8000:8064]).to(device).float())

x.shape

plt.imshow(x[1].T.cpu())

plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(x.to(device)[:64], padding=2, normalize=True).cpu(),(1,

plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(x.to(device)[48], padding=2, normalize=True).cpu(),(1,2



"""Untitled11.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1PcQCXVNBpjuRiYEQqUCmFCoKVx2I8T16

This code is inspired from Pytorch official websites DCGAN tutorial
https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
"""
```

```
text_embed = np.load("/content/drive/MyDrive/DCGAN/text_embed.npy")

!cp -r '/content/drive/MyDrive/102flowers/jpg' photo

image_names = glob.glob("/content/photo/*.jpg")
image_names.sort()


workers = 2

batch_size = 32
image_size = 64
nc = 3
nz = 100
ngf = 64
ndf = 64
num_epochs = 100
lr = 0.0002
beta1 = 0.5
ngpu = 1

class FlowersDataset(Dataset):

    def __init__(self, im_dir, text_embed, transform=None):

        self.im_dir = im_dir
        self.text_embed = text_embed
        self.transform = transform

    def __len__(self):
        return len(self.im_dir)

    def __getitem__(self, idx):

        if torch.is_tensor(idx):
            idx = idx.tolist()

        image = io.imread(self.im_dir[idx])
        text_vec = self.text_embed[idx]

        sample = {'image': image, 'text_vec': text_vec}

        if self.transform:
            image = self.transform(Image.open(self.im_dir[idx]))
            sample["image"] = image

        return image, text_vec

dataset = FlowersDataset(im_dir = image_names,
                         text_embed = text_embed,
                         transform=transforms.Compose([
                            transforms.Resize(image_size),
                            transforms.CenterCrop(image_size),
                            transforms.ToTensor(),
                            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                        ]))


# %80 train %10 val %10 test split
train_size = int(0.8 * len(dataset))
double = len(dataset) - train_size
```

```
val_size = int(double/2)
test_size = double - val_size

train_set, double_set = torch.utils.data.random_split(dataset, [train_size, double])
val_set, test_set = torch.utils.data.random_split(double_set, [val_size, test_size])

print(len(val_set))
print(train_size + val_size + test_size)

# Create the dataloader
dataloader = DataLoader(train_set, batch_size = batch_size,
                                    shuffle = True,
                                    num_workers = 2)

# Decide which device we want to run on
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cpu")
# Plot some training images
real_batch,label = next(iter(dataloader))
plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch.to(device)[:64], padding=2, normalize=True).

def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)

class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.main=nn.Sequential(
            nn.ConvTranspose2d(2*nz,ngf*8,4,1,0,bias=False),
            nn.BatchNorm2d(ngf*8),
            nn.ReLU(True),

            nn.ConvTranspose2d(ngf*8,ngf*4,4,2,1,bias=False),
            nn.BatchNorm2d(ngf*4),
            nn.ReLU(True),

            nn.ConvTranspose2d(ngf*4,ngf*2,4,2,1,bias=False),
            nn.BatchNorm2d(ngf*2),
            nn.ReLU(True),

            nn.ConvTranspose2d(ngf*2,ngf,4,2,1,bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),

            nn.ConvTranspose2d(ngf,nc,4,2,1,bias=False),
            nn.Tanh()

        )


    def forward(self,x,embed):
        embed = embed.view(-1 , embed.shape[1],1,1)
        gen_input = torch.cat([x,embed],axis= 1)
        return self.main(gen_input)
```

```
netG=Generator().to(device)
netG.apply(weights_init)

# Print the model
print(netG)

class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()

        self.main =nn.Sequential(

            nn.Conv2d(nc+1,ndf,4,2,1,bias=False),
            nn.LeakyReLU(0.2,inplace=True),

            nn.Conv2d(ndf,ndf*2,4,2,1,bias=False),
            nn.BatchNorm2d(ndf*2),
            nn.LeakyReLU(0.2,inplace=True),

            nn.Conv2d(ndf*2,ndf*4,4,2,1,bias=False),
            nn.BatchNorm2d(ndf*4),
            nn.LeakyReLU(0.2,inplace=True),

            nn.Conv2d(ndf*4,ndf*8,4,2,1,bias=False),
            nn.BatchNorm2d(ndf*8),
            nn.LeakyReLU(0.2,inplace=True),

            nn.Conv2d(ndf*8,1,4,1,0,bias=False),
            nn.Sigmoid()

        )

        self.text_FNN = nn.Sequential(
            nn.Linear(nz,64*64),
            nn.BatchNorm1d(64*64),


        )

    def forward(self,x,embed):

        dense_embed = self.text_FNN(embed)
        dense_embed = dense_embed.view(-1 , 1, 64, 64)
        disc_input = torch.cat([x,dense_embed],axis = 1)

        return self.main(disc_input)

netD = Discriminator().to(device)

netD.apply(weights_init)
print(netD)

criterion=nn.BCELoss()

optimizerD=torch.optim.Adam(netD.parameters(),lr=lr, betas=(0.5, 0.999))
optimizerG=torch.optim.Adam(netG.parameters(),lr=lr, betas=(0.5, 0.999))



fixed_noise = torch.randn(batch_size, nz, 1, 1,device=device)
```

```
# Establish convention for real and fake labels during training
real_label = 1.
fake_label = 0.

# Commented out IPython magic to ensure Python compatibility.
img_list = []
G_losses = []
D_losses = []
iters = 0



IS_score_mean=[]
IS_score_std=[]
IS_mean= 0
IS_std = 0

print("Starting Training Loop...")

writer_real = SummaryWriter(f"/content/sample_data/logs109/real")
writer_fake = SummaryWriter(f"/content/sample_data/logs109/fake")

t1=time.time()

for epoch in range(num_epochs):
    for i, data_pair in enumerate(dataloader, 0):

        img, embed = data_pair
        embed = embed.float()
        embed = embed.to(device)
        #embed = torch.from_numpy(embed).to(device)
        netD.zero_grad()
        real_cpu = img.to(device)
        batch_size = real_cpu.size(0)

        label = torch.full((batch_size,), real_label, dtype=torch.float, device=device)
        # Forward pass real batch through D
        output = netD(real_cpu, embed).view(-1)
        # Calculate loss on all-real batch
        errD_real = criterion(output, label)
        # Calculate gradients for D in backward pass
        errD_real.backward()
        D_x = output.mean().item()

        ## Train with all-fake batch
        # Generate batch of latent vectors
        noise = torch.randn(batch_size, nz, 1, 1, device=device)



        # Generate fake image batch with G
        fake = netG(noise,embed)
        label.fill_(fake_label)
        # Classify all fake batch with D

        output = netD(fake.detach(), embed).view(-1)
        # Calculate D's loss on the all-fake batch
        errD_fake = criterion(output, label)
        # Calculate the gradients for this batch, accumulated (summed) with previous gradients
        errD_fake.backward()
```

```
        D_G_z1 = output.mean().item()
        # Compute error of D as sum over the fake and the real batches
        errD = errD_real + errD_fake
        # Update D
        optimizerD.step()


        netG.zero_grad()
        label.fill_(real_label)  # fake labels are real for generator cost
        # Since we just updated D, perform another forward pass of all-fake batch through D
        wrong_label = text_embed[np.random.randint(0,8189, size = (batch_size,))]
        wrong_label = torch.from_numpy(wrong_label).to(device)
        wrong_label = wrong_label.float()

        #output = netD(fake,wrong_label).view(-1)
        output = netD(fake,embed).view(-1)
        # Calculate G's loss based on this output
        errG = criterion(output, label)
        # Calculate gradients for G
        errG.backward()
        D_G_z2 = output.mean().item()
        # Update G
        optimizerG.step()

        # Output training stats
        if i % 200 == 0:
            print('[%d/%d][%d/%d]\tLoss_D: %.4f\tLoss_G: %.4f\tD(x): %.4f\tD(G(z)): %.4f / %.4f'
#                   % (epoch, num_epochs, i, len(dataloader),
                     errD.item(), errG.item(), D_x, D_G_z1, D_G_z2))
            t2 = time.time()
            hours = (t2 -  t1) // 60 // 60
            mins = (t2 -t1) // 60 - hours * 60
            secs = (t2 -t1) - mins * 60 - 3600 * hours
            print("Elapsed time : {} hours {} mins {} secs ".format(int(hours), int(mins), int(s
        # Save Losses for plotting later
        G_losses.append(errG.item())
        D_losses.append(errD.item())

        # Check how the generator is doing by saving G's output on fixed_noise
        if (iters % 200 == 0) or ((epoch == num_epochs-1) and (i == len(dataloader)-1)):
            with torch.no_grad():
                fake = netG(fixed_noise, embed).detach().cpu()
                img_grid_real = torchvision.utils.make_grid(img[:32], normalize=True)
                img_grid_fake = torchvision.utils.make_grid(fake[:32], normalize=True)

                writer_real.add_image("Real", img_grid_real, global_step=iters)
                writer_fake.add_image("Fake", img_grid_fake, global_step=iters)

                IS_mean,IS_std=inception_score(fake,batch_size=int(batch_size/2),resize=True)
                IS_score_mean.append(IS_mean)
                IS_score_std.append(IS_std)


            img_list.append(vutils.make_grid(fake, padding=2, normalize=True))

        iters += 1

    if epoch % 1 == 0:
        torch.save(netD, "/content/drive/MyDrive/DCGAN/netDmodel5")
        torch.save(netG, "/content/drive/MyDrive/DCGAN/netGmodel5")
```

```
!pip install tensorboard

# Commented out IPython magic to ensure Python compatibility.
# %load_ext tensorboard

# Commented out IPython magic to ensure Python compatibility.
# %tensorboard --logdir="/content/sample_data/logs109"

import pandas as pd

q = pd.DataFrame(IS_score_mean)

y = pd.DataFrame()

y['y'] = q.rolling(5).mean()

plt.plot(ssss[::2])
plt.xlabel('Number of Epoch')
plt.ylabel('Score')
plt.title('IS of the Stage1')

ssss = np.load('/content/drive/MyDrive/best_IS_score_stg1.npy')

plt.plot(IS_score_mean[:])

plt.plot(IS_score_mean)
plt.xlabel("number of epoch")
plt.ylabel("score")
plt.title("IS Score vs Epoch")

noisee =  torch.randn(64, Z_DIM, 1, 1).to(device).float()

gen.eval()
with torch.no_grad():
    x = gen(noisee,torch.from_numpy(text_embed[8000:8064]).to(device).float())

plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(x.to(device)[:64], padding=2, normalize=True).cpu(),(1,

plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(x.to(device)[48], padding=2, normalize=True).cpu(),(1,2

#!cp -r '/content/sample_data/logs69' '/content/drive/MyDrive'
```