Research and Thesis Report

Name: Xu Ouyang

CWID: A20361390

Abstract

This is a report for research and thesis. I continued implement the model that generates images from natural language descriptions and combined with several neural networks such as DCGANS, Generative Neural Network and convolutional LSTM. DCGAN, including a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. LSTM, a recurrent neural network in order to learn the relevance among each word in sentence. We try on kinds of dataset such as Flickr30k, Caltech-UCSD Birds and Oxford Flowers, and we hope our model can input description annotation and output image which is related to this annotation finally.

1 Introduction

There exists some promising researches of computer vision and image understanding in natural image statistics. One of research direction is exploit the inference and generative capabilities of deep neural networks. The problem of generating images from visual descriptions gained interest in the research community, but it is far from being solved. Previously study on generating images using defined distributions which were conditioned on classification labels.

In this paper, we illustrate how recurrent neural network and generative adversarial network can be used to translate descriptions into real images. We use the flickr-30k dataset along with five text descriptions per image we collected as our evaluation setting at the beginning of experiment. And then we try on Caltech-UCSD Birds and Oxford Flowers datasets. We also use word2vec which is a group of related models that are used to produce word embeddings in each description. We demonstrate its performance both on the training set categories and on the testing set.

2. Related work

Many researchers have recently exploited the capability of deep convolutional decoder networks to generate realistic images. Dosovitskiy et al. (2015) trained a deconvolutional network (several layers of convolution and upsampling) to generate 3D chair renderings conditioned on a set of graphics codes indicating shape, position and lighting. Yang et al. (2015) added an encoder network as well as actions to this approach. They trained a recurrent convolutional encoder-decoder that rotated 3D chair models and human faces conditioned on action sequences of rotations. Reed et al. (2015) encode transformations from analogy pairs, and use a convolutional decoder to predict visual analogies on shapes, video game characters and 3D cars.

Generative adversarial networks (Goodfellow et al., 2014) have also benefited from convolutional decoder networks, for the generator network module. Denton et al. (2015) used a Laplacian pyramid of adversarial generator and discriminators to synthesize images at multiple resolutions. This work generated compelling high-resolution images and could also condition on class labels for controllable generation. Radford et al. (2016) used a standard convolutional decoder, but developed a highly effective and stable architecture incorporating batch normalization to achieve striking image synthesis results.

In contemporary work Mansimov et al. (2016) generated images from text captions, using a variational recurrent autoencoder with attention to paint the image in multiple steps, similar to DRAW (Gregor et al., 2015). Impressively, the model can perform reasonable synthesis of completely novel (unlikely for a human to write) text such as “a stop sign is flying in blue skies”, suggesting that it does not simply memorize. While the results are encouraging, the problem is highly challenging and the generated images are not yet realistic, i.e., mistakeable for real. Our model can in many cases generate visually-plausible 64×64 images conditioned on text, and is also distinct in that our entire model is a GAN, rather only using GAN for post-processing.

Building on ideas from these many previous works, we develop a simple and effective approach for text-based image synthesis using a character-level text encoder, DCGAN and LSTM. We propose a novel architecture and learning strategy that leads to compelling visual results. We focus on the case of fine-grained image datasets, for which we use the recently collected descriptions for flickr-30k images with 5 human-generated captions per image.

3. Background

3.1 Word2vec

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

Word2vec can utilize either of two model architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram. In the continuous bag-of-words architecture, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction (bag-of-words assumption). In the continuous skip-gram architecture, the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weights nearby context words more heavily than more distant context words. According to the authors' note, CBOW is faster while skip-gram is slower but does a better job for infrequent words.

After learning word2vec and glove, a natural way to think about them is training a related model on a larger corpus, and English wikipedia is an ideal choice for this task.

3.2 Generative adversarial networks

Generative adversarial networks (GANs) consist of a generator G and a discriminator D that compete in a two-player minimax game: The discriminator tries to distinguish real training data from synthetic images, and the generator tries to fool the discriminator. Concretely, D and G play the following game on V(D,G):

$$minGmaxD\ V(D, G) = Ex{\sim}pdata(x)[\log D(x)] + Ex{\sim}pz(z)[\log(1 - D(G(z)))]$$

Goodfellow et al. (2014) prove that this minimax game has a global optimium precisely when $p_g = p_{data}$, and that under mild conditions (e.g. G and D have enough capacity) p_g converges to p_{data}. In practice, in the start of training samples from D are extremely poor and rejected by D with high confidence. It has been found to work better in practice for the generator to maximize $\log(D(G(z)))$ instead of minimizing $\log(1 - D(G(z)))$.

3.3 Long short-term memory

Let's introduce RNN at first. A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented connected handwriting recognition or speech recognition.

Long short-term memory (LSTM) is a special form of recurrent neural networks (RNNs), which process sequence data. LSTM uses a few gate vectors at each position to control the passing of information along the sequence and thus improves the modeling of long-range dependencies. While there are different variations of LSTMs, here we present the one adopted by Rocktaschel et al. (2016). ¨Specifically, let us use $X = (x_1, x_2, \ldots, x_N)$ to denote an input sequence, where $x_k \in R (1 \leqslant k \leqslant N)$. At each position k, there is a set of internal vectors, including an input gate i_k, a forget gate f_k, an output gate o_k and a memory cell c_k. All these vectors are used together to generate a d-dimensional hidden state h_k as follows:

$$
\begin{aligned}
i_k &= \sigma(W^i x_k + V^i h_{k-1} + b^i),\\
f_k &= \sigma(W^f x_k + V^f h_{k-1} + b^f),\\
o_k &= \sigma(W^o x_k + V^o h_{k-1} + b^o),\\
c_k &= f_k \odot c_{k-1} + i_k \odot \tanh(W^c x_k + V^c h_{k-1} + b^c),\\
h_k &= o_k \odot \tanh(c_k),
\end{aligned}
$$

where σ is the sigmoid function, $\odot$ is the elementwise multiplication of two vectors, and all $W^* \in R^{d\times l}$, $V^* \in R^{d\times d}$ and $b^* \in R^d$ are weight matrices and vectors to be learned.

4. Implement details

Our approach is to train a deep convolutional generative adversarial network(DCGAN) conditioned on description features of n-1 words in each sentence, and only the generative part of DCGAN on description features of last word in each sentence using Euclidean distance as the loss function. What's more, we add a LSTM neural network above the generative model and under the discriminative model.

4.1 Network architecture

The architecture of network is as follows:

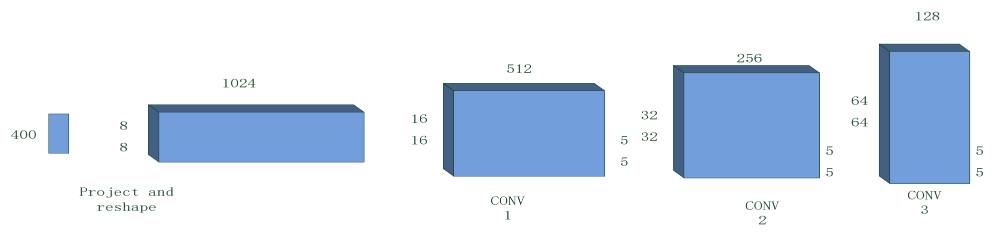


Figure 1: Generative model in DCGAN. A 400 dimensional word2vec Z is projected to a small spatial extent convolutional representation with many features maps. A series of three fractionally-strided convolutions then convert this high level representation into a 64x64 pixel image.

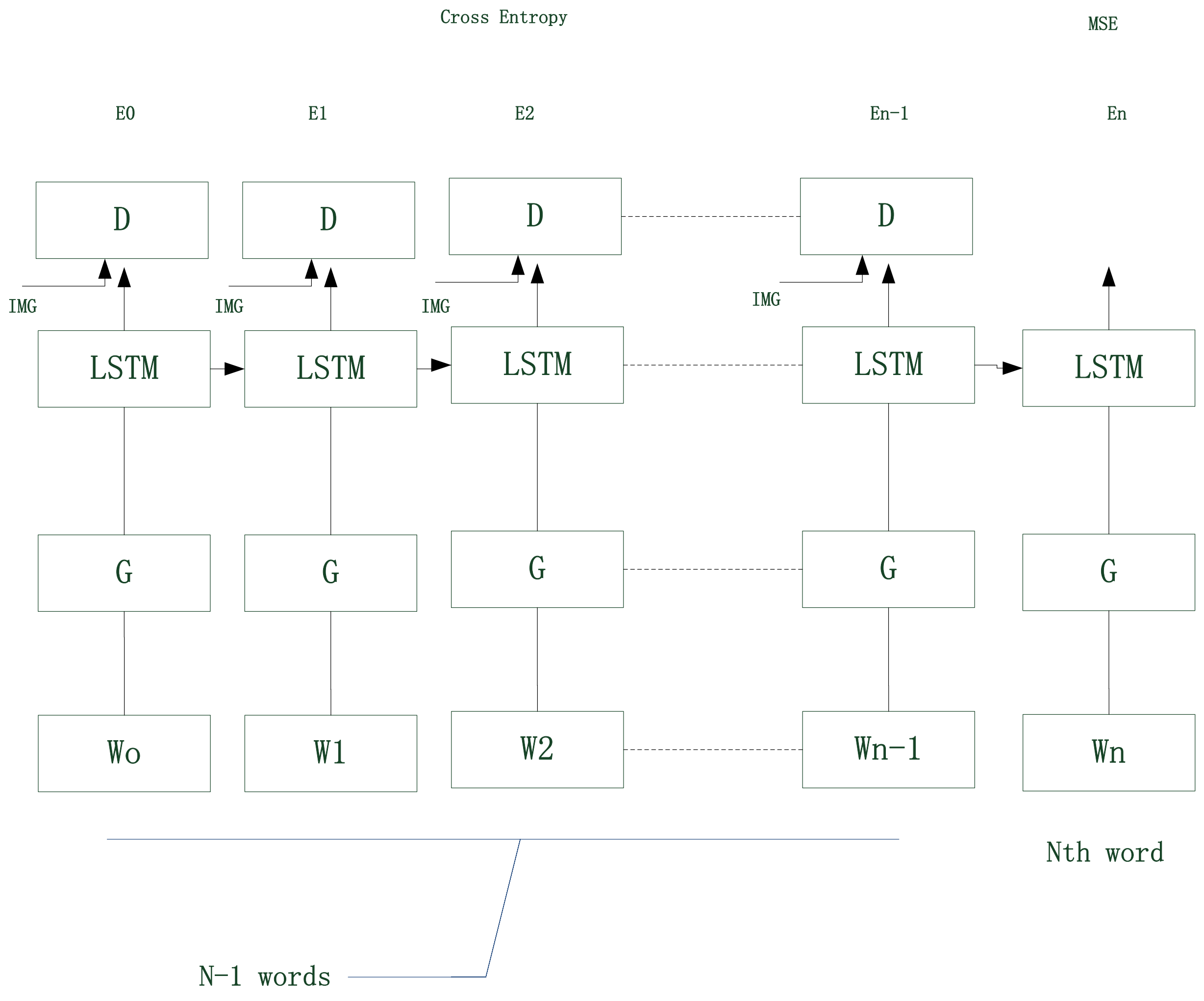


Figure 2: The whole network including DCGAN, LSTM.

There are two parts in the network, the first part we input 24 words which each of them is 400 dimensional word2vec features into the generator. The generator would output 24 number of 64x64 dimensional features via three deconvolutional layers. Meanwhile, the second part we input the last word of sentence which is 400 dimensional word2vec features into the generator. The generator output 64x64 dimensional features via three deconvolotional layers. And then we concatenated these outputs of two parts as the input of LSTM. We use the 1~24th output of LSTM and the real images as the inputs of discriminator, As a result, we hope the output of the last LSTM would be similar to the real image of this description as much as possible.

4.2 Algorithm

Input: minibatch n-1 word of 400 dimensional word2vec features, minibatch n-1 real images, minibatch nth word of 400 dimensional word2vec features, minibatch 1 real images.

```
For number of training iterations do
    For k steps do
    ● Sample mninibatch of m number of n-1 word of 400 dimensional word2vec features from pg(Z)
    ● Sample minibatch of m number of n-1 real images from data generating distribution pdata(x)
    ● Update the discriminator by ascending its stochastic gradient:
          $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$
    End for
    ● Sample minibatch of m number of nth word of 400 dimensional word2vec features from pg(z)
    ● Update the generator by descending its stochastic gradient:
          $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$
End for
```

The discriminator would label the real images as one, and the generated images as zero. We use binary cross entropy as the loss function. We compute the Euclidean distance between 25th output of LSTM and the real image which is the same image of this description. And then we combine the binary cross entropy of generator with Euclidean distance as the total loss function to update the gradient of generator part. So the stochastic gradient of updating generator formula becomes:

$$\nabla_{\theta_d \frac{1}{m}} \sum_{i=1}^{m} (\log\left(1 - D\left(G(z^{(i)})\right)\right) + ||Og - R||)$$

Where the Og represents the output of generator model, and the R represents the real images.

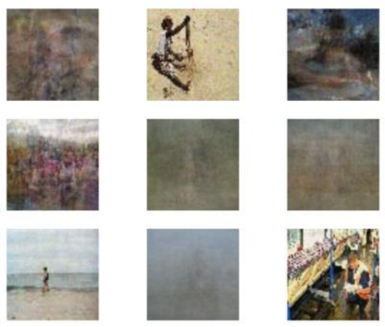
5. Experiment and discussion

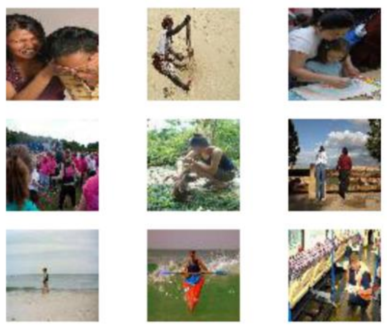
In order to verify the generative neural network (GNN), DCGAN, LSTM, I did plenty of experiments.

(1). GNN on flickr30k

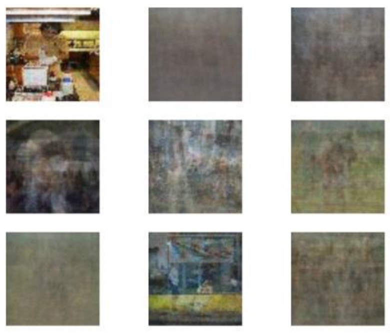
I ran the GNN on the flickr30k dataset. I used word2vec(400 dimensions vector) as the input of network, the results after 100 epoch are as follows:

Training result:


generative image


real image

Validation result:

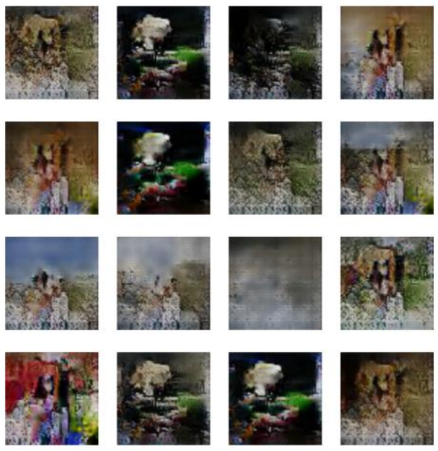

generative image

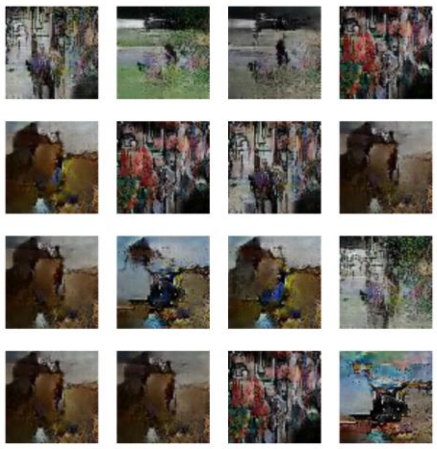
real image

As we can see, some training results are pretty good even same to the real images. However, in the validation result, even though we can see some meaningful images, there are no corresponding relationship between generative images and real images. The reason is that one word might correspond to several images.

(2) DCGAN on flickr30k

First, I used 400 dimension random uniform distribution noise as the input of DCGAN which I get clear and meaningful images in flickr30k datasets. The results are as the follows:



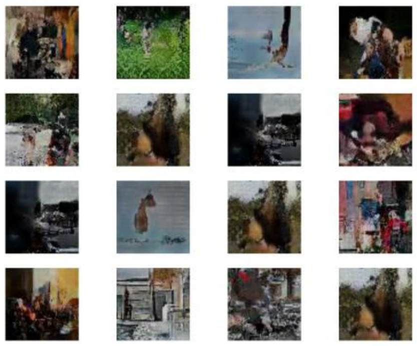


epoch = 250th   epoch = 500th

epoch = 750th   epoch = 1000th

As we can see, the generated images are not clear and meaningless.

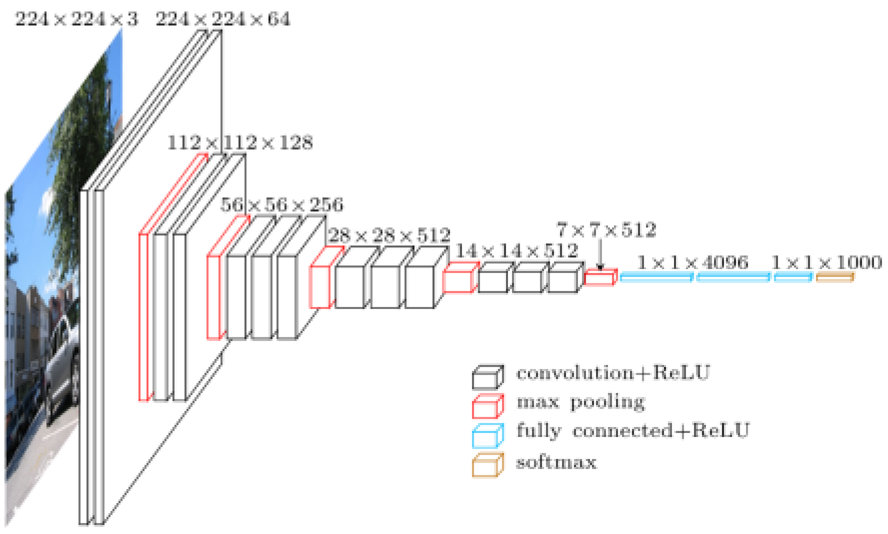
Second, I used 400 dimension word2vec features as the input of DCGAN. The results are as the follows:



As we can see, they are not clear and meaningless.
I tried to increase the network layer, fun tune the learning rate, change the number of dataset, however, the results are still not clear. I thought the reason is that the content of images of flickr30k are too complicated to be learned by neural network.

To make sure the project continued, I decided to make some changes on the dataset: I'm trying to use the VGG16 model (the input is the images of flickr30k), and extract the output of one of full connect layer. And then I will use Gaussian Mixture Model to cluster these outputs. Actually, I hope to label the flickr30k images and extract one class of them as the input of DCGAN which can decrease the complexity of neural network learning.

(3) VGG16 on flickr30k
In VGG16 model, the input is a fixed-size 224 × 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where we use filters with a very small receptive field: 3 × 3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations we also utilise 1 × 1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3 × 3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2 × 2 pixel window, with stride 2.



A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU (Krizhevsky et al., 2012)) nonlinearity.

The first thing is I tried to use the VGG16 model (the input is the images of flickr30k), and extract the output of one of full connect layer. And then I will use Gaussian Mixture Model to cluster these outputs. Actually, I hope to label the flickr30k images and extract one class of them as the input of DCGAN which can decrease the complexity of neural network learning. Second, I used the extracted one class images of GMM as the input of DCGAN and ran it.

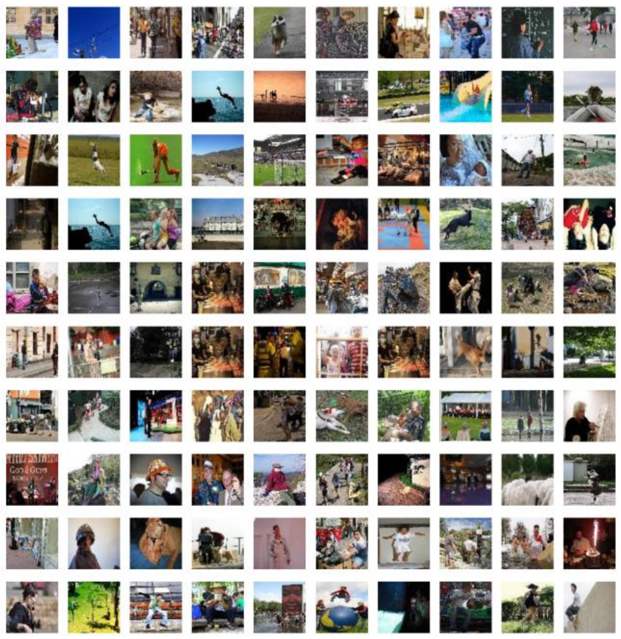
First, the components=10, and the result of GMM is as follows(there are 158915 images):

class0	15030
class1	19330
class2	12880
class3	22550
class4	12885
class5	20400
class6	6440
class7	19345
class8	16105
class9	13950

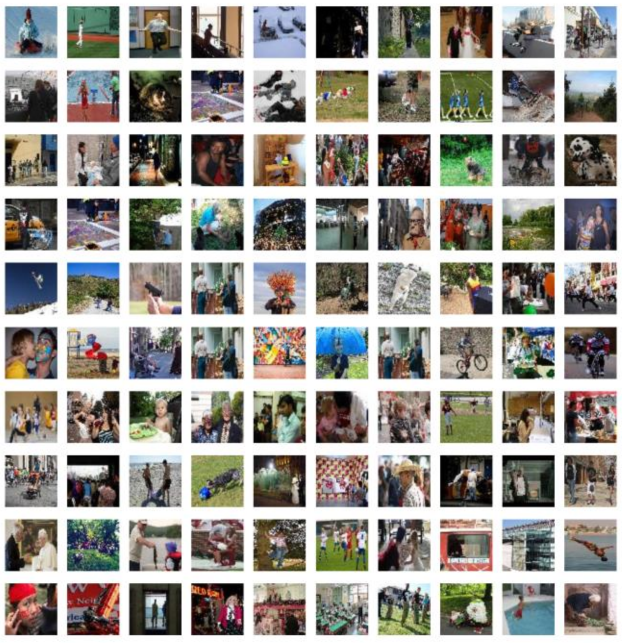
I choose class 3 images as the input of DCGAN, the results after 25 epoch are as the follows:

As we can see, the content of images are not clear at all. The most serious problem is that after 60 epochs the network seems like collapsed.

I think the reason is that the GMM can’t classify the flickr30k dataset well. In order to verify my supposition, I looked at the clustered images from Gaussian mixture model. The images are as the follows:



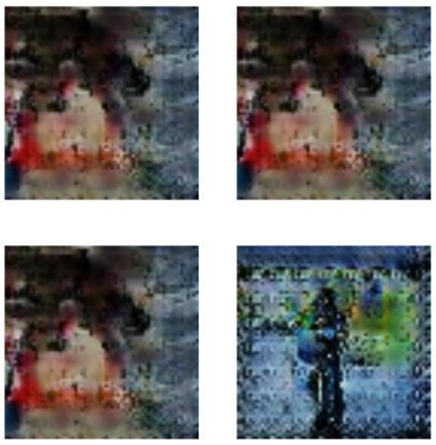
Cluster 1

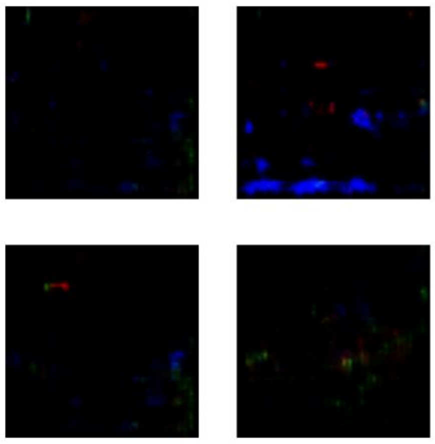


Cluster 2

As we can see, there is almost no relationship among each images in same cluster. What's more, some kind of images among different clusters look like similar to each other. I think the reason is that the content of images in flickr30k are too complicated, which leads to cluster by GMM or generated by DCGAN difficult.

For the next step, I manually classify the images of flickr30k, I think this is the most directed way to cluster the data, even though it costs some time. After classified, I used preprocessed flickr dataset which I just manually chose the person and dog images as the inputs of DCGAN. The results are as the follows:





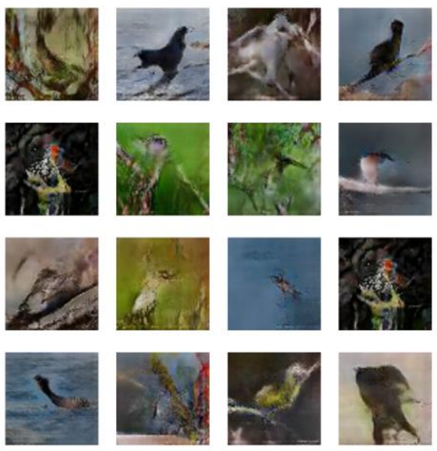
Epoch = 35th   epoch = 45th

As we can see, the performance is not good, even collapse at the end. In conclusion, the DCGAN can't learn the flickr dataset.

(4) DCGAN on Caltech-UCSD Birds 200

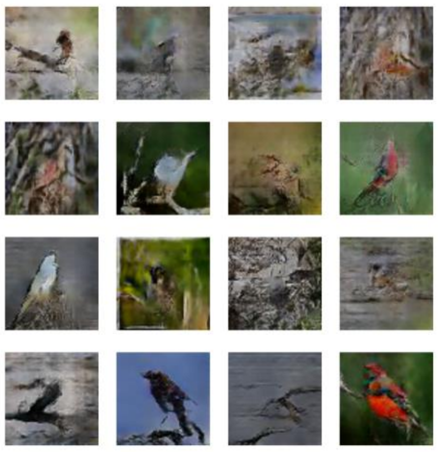
I used another dataset named Caltech-UCSD Birds 200 to train DCGAN. I used 400 dimension uniform random noise as the input, which has gotten some not bad results on CIFAR10 dataset, the results are as the follows:

Real image

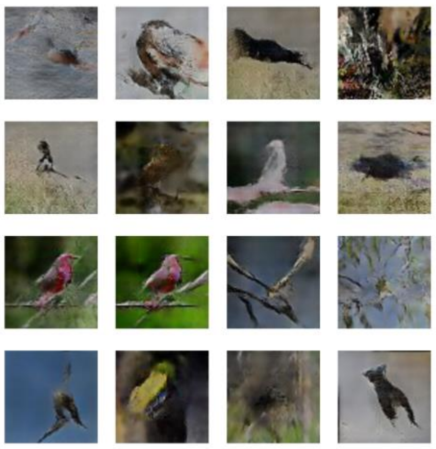


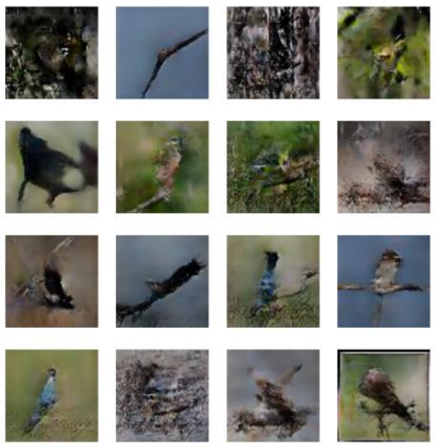
As we can see, there are some objects which look like bird, however, part of others still look like meaningless.

Next, I added some blurry images as the faker dataset into the discriminator of DCGAN. The results are as the follows:



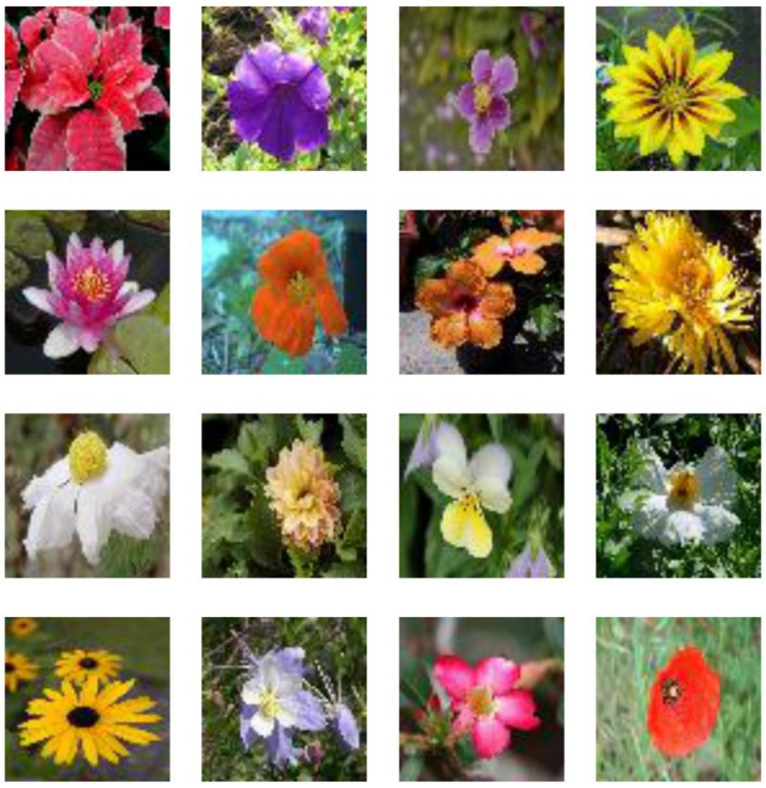
Next, I used the word2vec of description of each images as the input of DCGAN, each image have 10 sentences. To simplify the experiment at the beginning, I sum this ten sentences and mean them, so the input shape becomes 1x400(former: 10x25x400), the results are as the follows:





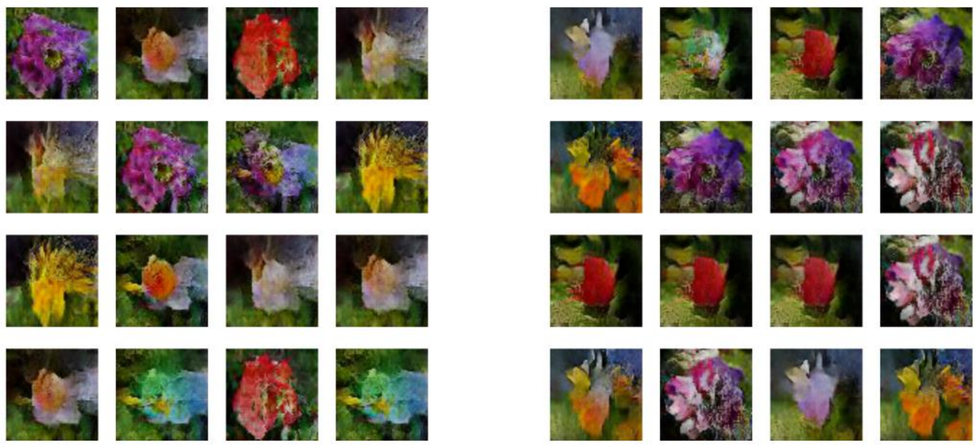
As we can see, there are some meaningful results. However, some other images look like meaningless. I think the reason is that the bird dataset is still difficult for DCGAN learning. So I will try on 102 Category Flower Dataset later.

(5) DCGAN on 102 Category Flower Dataset



Real image

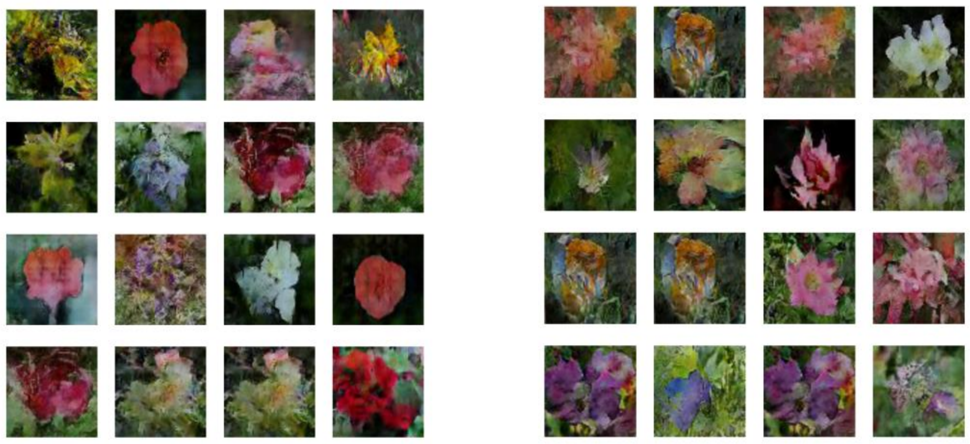
First I use 400 dimension random uniform distribution noise as input of network, the results are as the follows:



Generative image

As we can see, we get some great results which we never saw before.

Second, I sum the word2vec of all words in each sentence(original input size: # x 25 x 400, new size: # x 1 x 400) as the input of networks, the results are as the follows:



Generative images

In the next stage, I will implement the whole network on this dataset.

(6) Convolutional LSTM

First, when I add the LSTM on the network, there are around 15 billion parameters in traditional LSTM due to it¡¯s full connect structure, which leads the network impassible to train. So I search for the substitutable method to make sure it can work. Here is a paper named ¡°Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting which introduce the convolutional LSTM. The main idea is as the follows:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i)\\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f)\\
c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)\\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o)\\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

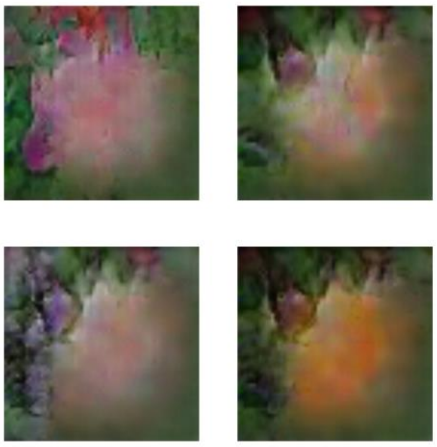
Traditional LSTM

$$
\begin{aligned}
i_t &= \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i)\\
f_t &= \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f)\\
\mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c)\\
o_t &= \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o)\\
\mathcal{H}_t &= o_t \circ \tanh(\mathcal{C}_t)
\end{aligned}
$$

Convolutional LSTM

As a result, the network can work now.

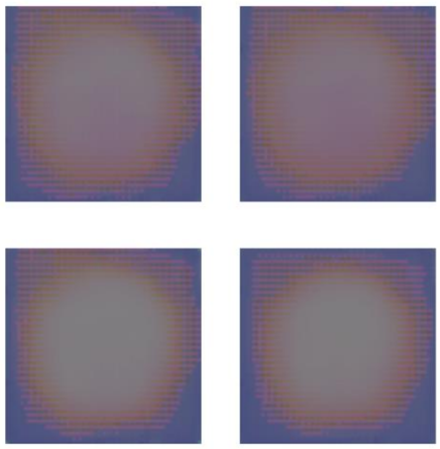
So next, I ran the whole neural network at first time. Now, the size of input of DCGAN is 81890x24x3x64x64, the size of input of generative neural network is 81890x3x64x64. The results after 5 epoch (I only trained 5 epoch, because it would cost 18 hours each epoch)are as the follows:


Generative image

real image

As we can see, the images are not clear.

Because the training part costs too much time, I decided to reduce the number of samples from 81890 to 24567. What's more, I found that the original batch normalization is not suit for new inputs that I increase a new dimension the length of sentence, so I need to change the batch normalization code. However, when I try to use the new batch normalization method, the performance of network become pool after 9 epochs as the follows:

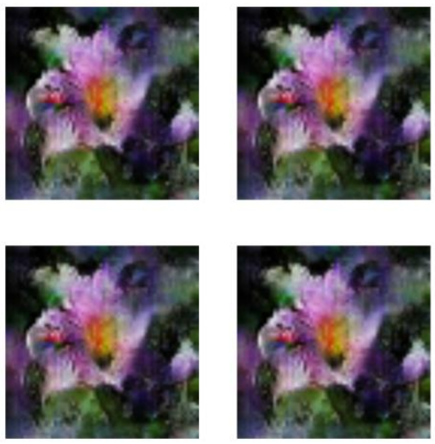


Generative image

real image

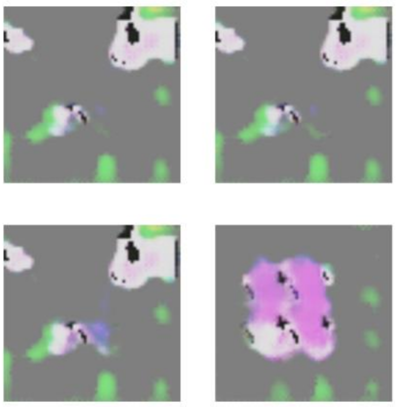
I think my new method should be worked on the 5D inputs, however the result is not good. So I analyze my network and I suppose the problem might come from three places: batch normalization, convolutional LSTM, the loss function.

First, the batch normalization, when the inputs are 4D, the DCGAN works using batch normalization. To verify whether the new method is right, I try to only run DCGAN with 5D inputs now. The results after 10 epochs are as the follows:



As we can see, the results look like meaningful. So I think the new batch normalization work now.

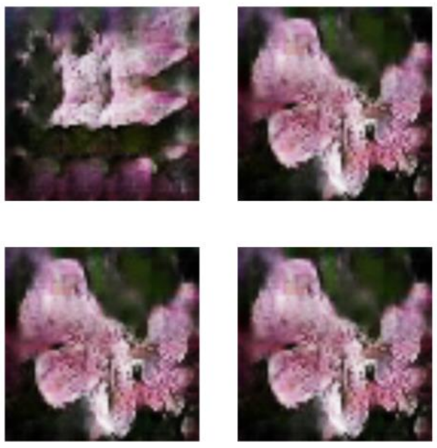
Second, the convolutional LSTM. I'm not quit sure whether my convolutional LSTM work on due to my first time to implement it. To verify my method, I run the DCGAN adding convolutional LSTM except generative neural network. The results after 5 epoch are as the follows:



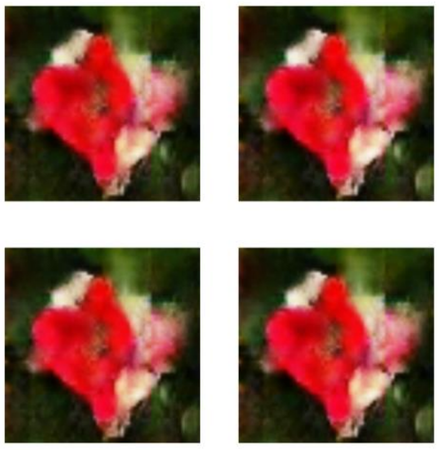
As we can see, the results look like unclear, I’m still analyzing the reason.

Third, the loss function. The generative neural network and the generator of DCGAN share the same parameter and use the combination loss as the loss function(cost = generator cost + GNN cost), I and Mr. Xi are not sure whether it can work. Once I finish the first and second experiment, and the network still not work, at this time, we can say that the problem comes from the loss function part. At that time, I will create parameter for generator of DCGAN and GNN respectively.

As we discussed before, there were two ways to implement batch normalization. The first way is using the shape of batchsize*length*channel*height*width as the input of network. The second way is using the shape of length*batchsize*channel*height*width as the input of network. To verify which way is better, I did two experiments, and the results after 20 epochs are as the follows:



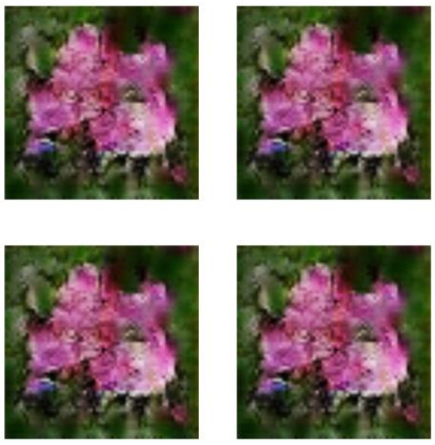
1st input



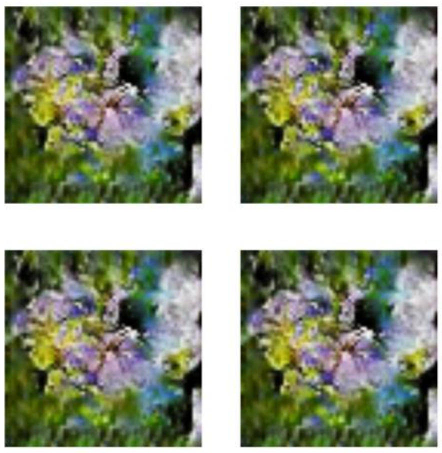
2nd input

As we can see, both methods work. In order to simplify the network, I choose the second input.

Next, I found the result that add convolutional LSTM on network is failed. I think the reason is that I use the convLSTM as the last layer of generator in network which lead to the convLSTM layer not only need to transmit the information among each word in each sentence, but also need to generate images. In other words, the convLSTM layer affords to too many functions which might have been out of its capacity. So there are two ways to figure out this problem: the first is adding two deconvolution layers without channel and size changing on the convLSTM; the second is changing the position of convLSTM that put it down to the last layer but one. I think these two ways can both share responsiblitiy for image generating. To verify my supposition and find which method is better, I did two experiments and the results of 18th epoch are as follows:



1st method



2nd method

As we can see, the first method performs well, however the second result looks like unclear. I think the reason is that the parameters in second method are not enough, as a result, the network can’t learn the flower data well. Of course, these meaningful images prove my supposition. So next step, I can add the generative model to my network.

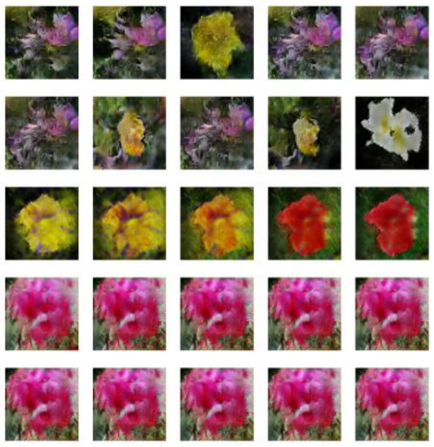
Here is one thing noticing me that each image look like the same in each epoch. There are two explanations: first, due to the property of GAN itself that the results converge to same point together easily; second, I choose the same situation words of different sentences as the printing images.

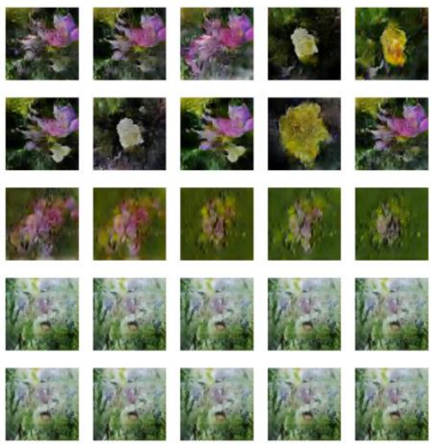
Description samples:
this is a flower with white petals and purple and white anthers.
this festive looking flower has beautiful pedals of white and pink.
this flower has petals that are pink with purple and yellow lines.
I will verify this via several experiments, I will print the image of each word in one sentence, and the image of different position word in different sentence in the future.

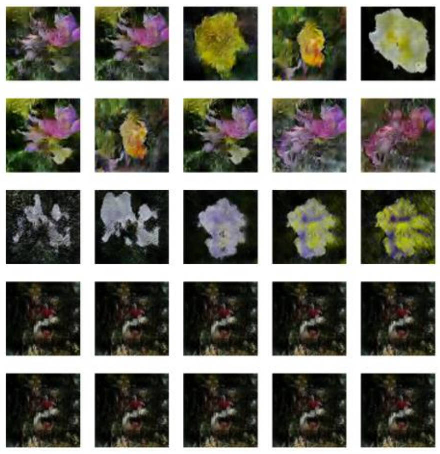
Next, I figured out the problem of same output of DCGAN.

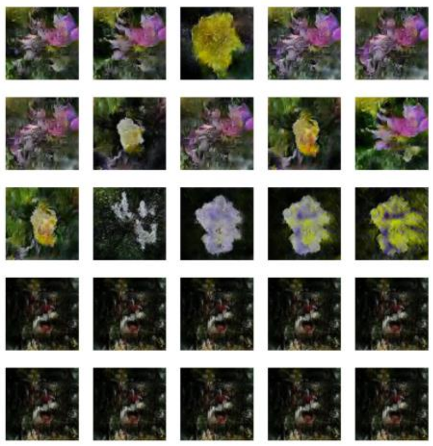
I use random real images as the input of discriminator instead of corresponding real images. The reason is that, when you use same images for each word generating of one sentence, there is short of representativeness of samples. The results after 7 epoch are as the follows:

this flower has petals that are red with white shading. 10
this flower is light pink in color and has petals that are long and skinny. 15
this flower has yellow pistil and white petals as its main features. 12
this flower has petals that are pink with yellow and purple lines. 12

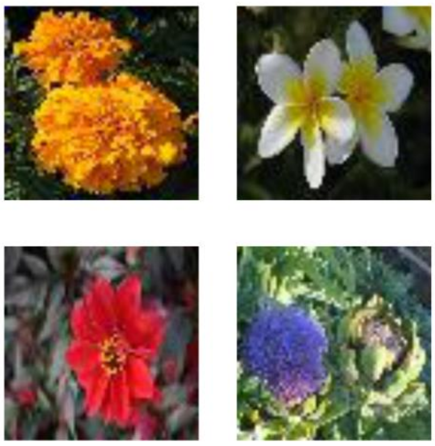








Next, I run my final program continully, which includes DCGAN, convolutional LSTM and Generative Neural Network. The goal is inputing one sentence into this network and then outputting an image which corresponds with this sentence. And in my program, I hope the last word of each sentence can output the corresponding images. The results after 16 epochs (each epoch cost around 6 hours) are as the follows:



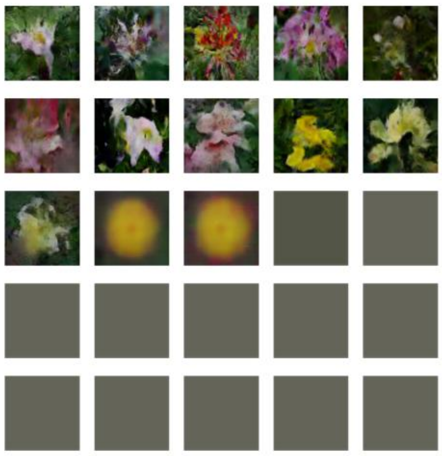
Description:

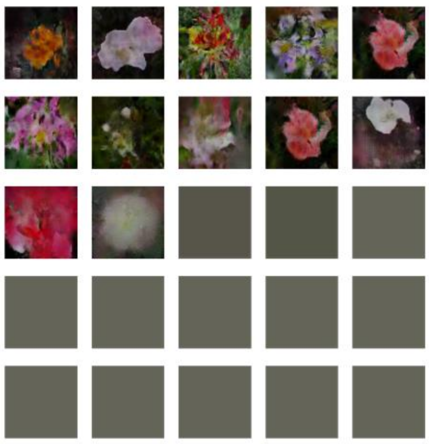
this flower is yellow in color, with petals that are bunched together closely. 13

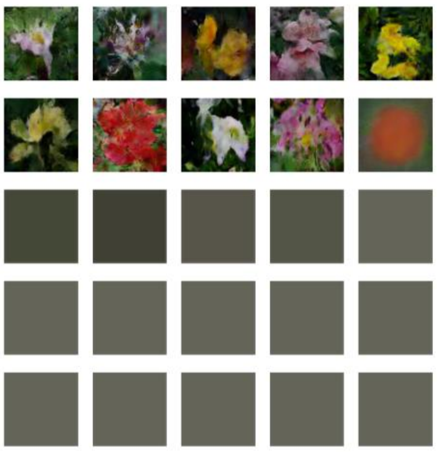
these flowers is white and yellow in color and looks like rubber. 12

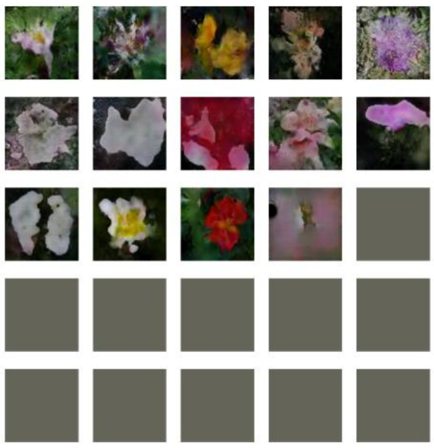
this flower has petals that are red with yellow stamen. 10

this flower has the lavender colored string like petals arranged clumsy as a bunch. 14

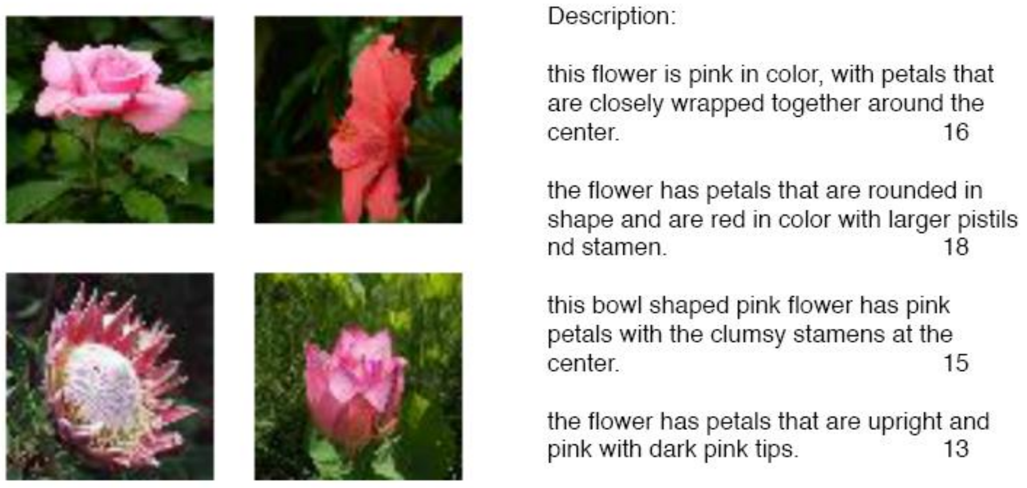








As we can see, we just get rough outline of flower with similar color. There are two things we should notice: First, the shape of each image look like circle; second, there are no relationship among each word in each sentence, however there should be because I use convolutional LSTM. I think the reason is that the proportion of euclidean distance in loss function is too small. What’s more, there exists information transferring lose and gradient transferring lose in LSTM. There are two solutions: first, adding a penalty on euclidean distance in loss function; second, meaning the image of output of each word as the input of euclidean distance. Both of them in order to reserve information as much as possible. So now, let’s look at the results using three different methods at same epoch(the 4th epoch):



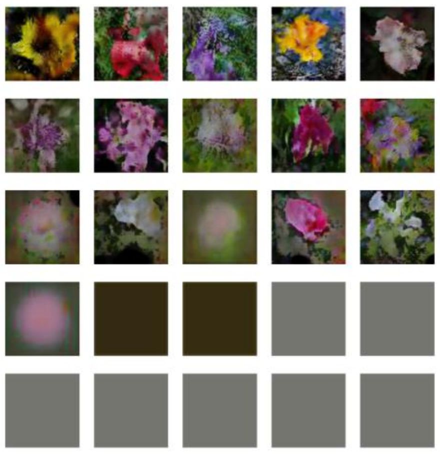
Description:

this flower is pink in color, with petals that are closely wrapped together around the center. 16

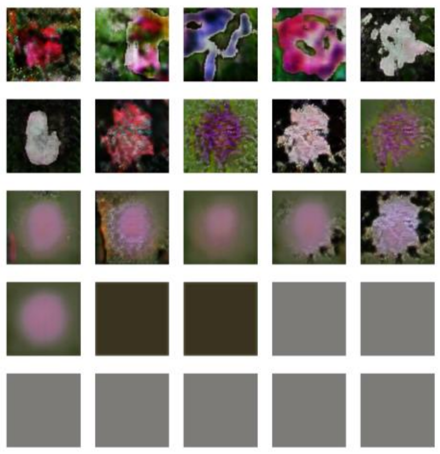
the flower has petals that are rounded in shape and are red in color with larger pistils nd stamen. 18

this bowl shaped pink flower has pink petals with the clumsy stamens at the center. 15

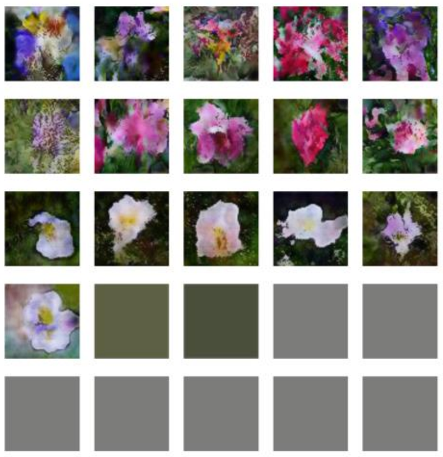
the flower has petals that are upright and pink with dark pink tips. 13



Original method



penalty method



Mean method

As we can see, both the second and the third method build relationships among each word in each sentence. What's more, the output of last word looks like real image in third method. Of course, we still need to observe the following results in the future.

6. Conclusion and future work

In conclusion, we proved that the GNN (generative adversarial neural network), DCGAN, LSTM works on flower dataset well. And we have combined these three neural network together to see the output of network.

In the next step, we will optimize the performance of final neural network.

7. Reference

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B.,Warde-Farley, D., Ozair, S., Courville, A., and Bengio,Y. Generative adversarial nets. In NIPS, 2014.

[2] Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. 2016.

[3] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee. Generative Adversarial Text to Image Synthesis, 2016

[4] Dosovitskiy, A., Tobias Springenberg, J., and Brox, T. Learning to generate chairs with convolutional neural networks. In CVPR, 2015.

[5] Mansimov, E., Parisotto, E., Ba, J. L., and Salakhutdinov,R. Generating images from captions with attention. ICLR, 2016.

[6] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen. Improved Techniques for Training GANs, 2016

[7] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015

[8] Xingjian S H I, Chen Z, Wang H, et al. Convolutional LSTM network: A machine learning approach for precipitation nowcasting[C]. Advances in Neural Information Processing Systems. 2015: 802-810.

[9] Gemici M, Hung C C, Santoro A, et al. Generative Temporal Models with Memory[J]. arXiv preprint arXiv:1702.04649, 2017.

[10] Reed S, Akata Z, Yan X, et al. Generative adversarial text to image synthesis[C]. Proceedings of The 33rd International Conference on Machine Learning. 2016, 3.

[11] Arjovsky M, Bottou L. Towards principled methods for training generative adversarial networks[C]. NIPS 2016 Workshop on Adversarial Training. In review for ICLR. 2017, 2016.

[12] Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.