Caption to Image generation using Deep Residual Generative Adversarial Networks [DR-GAN]

Github : [Github Repository]

Abhishek Kumar	**Saj Bhavesh Maru**	**Siddhi Thakur**
Dept.of CSE	Dept. of CSE	Dept.of CSE
akumar58@buffalo.edu	sajmaru@buffalo.edu	siddhima@buffalo.edu
50419133	50416772	50365530

Abstract

Generative Adversarial Networks or GANs are a generative modeling approach using Deep Learning to train the model to generate photo-realistic images. The proposed GAN model is built by conditioning the generated images on a text description instead of on a class label.

To estimate complicated and joint example probability, GANs promote an adversarial game. To approach realistic data distributions, networks driven by noise produce artificial samples. In order to transfer attribute vectors to the relevant data, the conditional GAN combines previous conditions as input. However, because the indirect learning guide is inefficient, the CGAN is not designed to deal with high-dimensional circumstances.

We implemented a Deep Residual GAN (DRGAN) network to create fine pictures from very latent noise. The coarse images are aligned to attributes and are embedded as the generator inputs and classifier labels. A straight route, similar to the Resnet, is covered in a generative network to directly transport coarse pictures to higher layers. In addition, adversarial training is used in a cyclic fashion to prevent picture degradation. Experimental results of applying the DRGAN model to datasets BIRD CUB-200 and FLICKR8K show its higher accuracy than the state-of-art GANs

1 Related Work

The traditional conditional GAN models consist of generators and discriminators provided with conditions as vector inputs and generate images, texts, or audio similar to the real data.

Auxiliary Classifier GANs (ACGANs) in [1] is a form of GAN employing label conditioning for image synthesis. Instead of providing the discriminator with the class label, as traditionally done in conditional GANs, the discriminator is tasked with predicting the class label. The authors claim that this increases the generation performance as the generator learns to generate realistic images that the discriminator classifies as the correct class.

For the generation of image data, a convolutional neural network has been introduced to the GAN model. In [2] A three-stage MirrorGAN approach is used to get the text embeddings of the captions as an input to the generators using word-level and sentence-level attention. Finally, an image captioning model is used to align the caption from the generated image with the input text

description.

Text-to-image creation has received much interest and work recently. [4] presented the AlignDRAW model, which created image patches in many phases using an attention mechanism over words in a caption. The CGAN was initially used by [5] to produce plausible visuals based on text descriptions. Text-to-image production was split into many steps by [6], with images ranging from coarse to fine. All of the previous methods, on the other hand, are primarily focused on creating a new high-quality image from a given text, and do not allow the user to modify the development of certain visual features using natural language descriptions.

2 Motivation

One of the most challenging problems in the world of Computer Vision is synthesizing high-quality images from text descriptions, and GANs have been found to be the most powerful neural network architecture to generate good results.

Generating photo-realistic images from text has tremendous applications, including photo-editing, computer-aided design, etc. Through this project, we wanted to explore architectures that could help us achieve our task of generating images from given text descriptions.

3 Dataset

3.1 Birds 2011 (CUB-200)

Caltech-UCSD Birds 2011 (CUB-200) is an image dataset with photos of 200 bird species (mostly North American). The total number of categories of birds is 200 and there are 6033 images in the 2010 dataset and 11,788 images in the 2011 dataset. Annotations include bounding boxes and segmentation labels.

3.2 Flickr8K

A new benchmark collection for sentence-based image description and search, consisting of 8,000 images that are each paired with five different captions which provide clear descriptions of the salient entities and events.

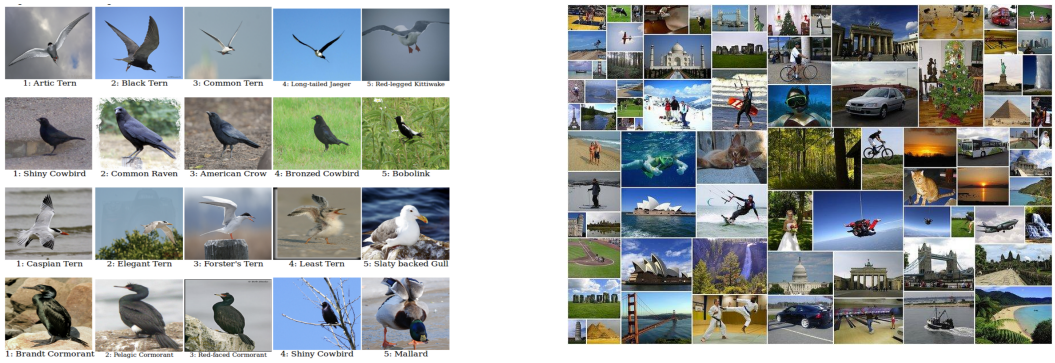

Figure 1: Bird CUB_200_2011 & Flickr8K dataset

4 Data Preprocessing

4.1 Image Preparation

We had to do some preprocessing on the images for them to be fed to the model.

- We first created a python script to fetch the data and store it in google drive since we were using google colab for development,

- After loading the dataset we then reduced the size of each image and converted them to a NumPy array. The image vectors were normalized from 0-255 to -1,1 to compact the value distribution.
- Finally, all the NumPy arrays of each image were stored in a pickle file and were then used for training the model.

These steps were followed for both Flickr 8K as well as Bird data set with slight differences in the preprocessing steps that varied depending on the size of the image, directory hierarchy, and image distribution. To feed the database to the discriminator we converted all the images to a NumPy array.

4.2 Caption Preparation

Captions are the building block of a text-to-image generation model. Mapping the caption to its corresponding image and extracting words or phrases that best describe the image is critical in such applications. We had to apply natural language processing to the captions which is standard practice for any text-related data.
For caption preparation,

- We first created a CSV file that mapped the caption to its images, we then processed the caption by removing punctuations, stop words, spaces, etc.
- To convert the caption to vectors we used the word_to_vec pre-trained model.
- Word2Vec is trained on the Google News dataset (about 100 billion words). It has several use cases such as Recommendation Engines, and Knowledge Discovery, and is also applied in the different Text Classification problems. We have used it to vectorize our captions.
- For the words which were not available in the word2vec model, we created aliases for those which to their close meaning words and used its vector.
- Finally, after vectorizing all the captions we dumped them into a pickle file and used it in our main model.

5 GAN Review

According to the original GANs, the training process is viewed as a game between the discriminator D and the generator G, with the adversary's purpose incorporated into a uniform equation that is a max-minimizing process with adversarial iterations to get parameters for both modules. The adversarial iterations will continue to approach the theoretical balance solution, eventually learning the module parameters that can accurately create or discriminate the data x distribution. Another GAN variant, the WGAN, uses adversarial learning on several loss measurements, with the final output determined by minimizing the reduced Wasserstein distance between the fake and actual samples.

The representation of condition y is then implicitly linked to the conditional probability p(x|y) under the premise that the distribution of data x is the margin of the conditional distribution P(y). According to this, the CGAN's object equation is as follows:

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

where the condition y is the control input to both the modules D and G.

5.1 DR-GAN

DR-GAN Discriminator

The discriminator classifies both real data and fake data from the generator.The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network. Due to the complex and unknown distributions of real images, image restoration is a difficult task.

Our discriminator network consists of blocks of stridden Conv-BatchNorm-leakyRelu layers with a Flatten and Dense layer at the end. We use a binary cross-entropy to calculate the loss and Adam as an optimizer for the discriminator network. We set the learning rate to be learning_rate=0.00004.

DR-GAN Generator with Resnet

Our GAN generators are deconvolution layers stacked on top of each other, and the entire structure is made up of deconvolution networks (DCNNs). DCNNs have been shown to be effective in generating realistic images.

The generator model in our architecture consists of convolutional layers with ResNet blocks to overcome vanishing gradient problems in the neural network and have better feature learning. We calculate the loss of the generator network using binary cross-entropy and use Adam optimizer to update the gradients. We set the learning rate to be learning_rate=0.00004.

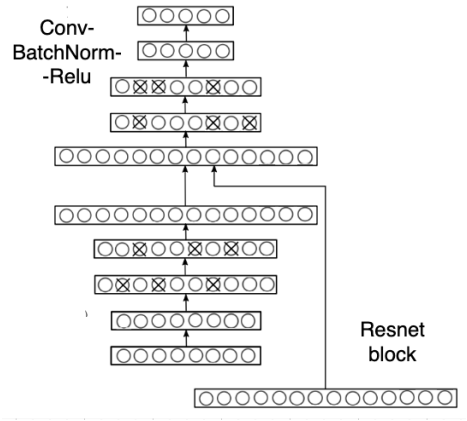

Figure 2: Proposed Deep Residual Generator model for our GAN model.

5.2 Training loss

Generator loss (mode seeking loss)

Our gen_loss is calculated using tensorflow's Binary cross entropy loss function using *fake output real text data*.

Steps: lz = tf.math.reduce_mean(tf.math.abs(fake_2-fake_1))/ tf.math.reduce_mean (tf.math.abs(noise_2-noise_))
eps = 1 * 1e-5, ms_loss_weight = 1.0 ,

loss_lz = 1 / (eps+lz) * ms_loss_weight,

Loss_new = gen_loss + loss_lz

Discriminator loss

We use binary cross-entropy to calculate the discriminator loss. We generate 3 different losses using, real image real text fake image real text, and real image fake text, and apply label smoothing to the output loss. We then add up all the losses to get the final discriminator loss.

total_disc = real_loss + alpha * (fake_loss) + (1- alpha) * fake_loss_ms where, alpha = 0.3

We generate discriminator loss for two outputs and add both the losses to get the total_disc_loss during training. **total_disc_loss** = disc_loss1 + disc_loss2

5.3 Label Smoothing

Label smoothing is a regularization technique for classification problems to prevent the model from predicting the labels too confidently during training and generalizing poorly.

We apply label smoothing to our outputs in the discriminator loss for both the positive and negative labels. These smoothed labels are then used to calculate the total discriminator loss.

```
#label smoothing
def smooth_positive_labels(y):
  return y - 0.3 + (np.random.random(y.shape) * 0.5)

def smooth_negative_labels(y):
  return y + np.random.random(y.shape) * 0.3
```

5.4 Training

We train our model using an alternate training method. We keep the generator constant during the discriminator training phase. As discriminator training tries to figure out how to distinguish real data from fake, it has to learn how to recognize the generator's flaws. That's a different problem for a thoroughly trained generator than it is for an untrained generator that produces random output. Similarly, we keep the discriminator constant during the generator training phase. Otherwise, the generator would be trying to hit a moving target and might never converge.

Due to limited resources we could only train our model for 1000 epochs on Google colab. We used a batch size of 64 with Adam optimizer with a learning rate of 0.0004 for training and saved checkpoints at every 60 epochs. We also generated and saved images while training at every 10 epochs to compare and check model training.

Training Algorithm :

Input: minibatch images x, matching text t, mismatching $\hat{t}$, number of training batch steps S
for $n = 1$ **to** S **do**
  $h \leftarrow \varphi(t)$ {Encode matching text description}
  $\hat{h} \leftarrow \varphi(\hat{t})$ {Encode mis-matching text description}
  $z \sim \mathcal{N}(0, 1)^Z$ {Draw sample of random noise}
  $\hat{x} \leftarrow G(z, h)$ {Forward through generator}
  $s_r \leftarrow D(x, h)$ {real image, right text}
  $s_w \leftarrow D(x, \hat{h})$ {real image, wrong text}
  $s_f \leftarrow D(\hat{x}, h)$ {fake image, right text}
  $\mathcal{L}_D \leftarrow \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2$
  $D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ {Update discriminator}
  $\mathcal{L}_G \leftarrow \log(s_f)$
  $G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ {Update generator}
end for

Figure 3: Our Text-To-Image training Algorithm

6 Experiments

We were using google colab for development purposes since it provides a free but limited GPU. Therefore training took a comparatively longer time due to reserved computational resources. For the entire implementation of the project, we did a number of parameter tuning and as well as applied other methods to improve the accuracy. These include GAN models with

6.1 Increased model parameters

We changed the model parameters by increasing the number of filters in each layer in our model by two or four times. By increasing the parameters we observed reduced generation loss as seen in the results section. Eg: for layer 4 in the generator we increased the filter size from 256 to 768.

6.2 Larger Batch Sizes:

We kept playing with the bath sizes of the input to the model. We got the best result when it was increased up to eight times its original size. Increasing the bath size and adjusting the learning rate quite boosted the image generation results, but on the other hand, the training time increased drastically. Eg. Batch size was increased from 24 to 64.

6.3 Architectural changes:

We encountered a situation where our generator got stuck in mode collapse where it wouldn't learn new images due to which the generated images were the same or almost identical.

So, we experimented with model architecture, and hidden layer parameters introduced new hyperparameters as well as tried varying the input and output size of the images. We also modified the regularization scheme to improve conditioning, which demonstrably boosted the performance. We used smaller kernels and increased the number of layers. We tried different padding and stride values. We changed the model weights initialization techniques by taking a random initializer instead of a standard initializer. We introduced BatchNormalization with a momentum of 0.8 which helped in confining the output values.

After some architectural changes, we were able to overcome the mode collapse issue and could successfully generate good quality and text-relevant images for Birds Dataset.

6.4 Image Generation for Flickr8k

We trained the same model on Flickr8K dataset to observe the results in a broader domain. Flickr8K dataset consists of images of various sizes and shapes and is more realistic with different scenes.

After Training for 1000 epochs, the results were relevant to the text. However, the generated images were comprehensible. We speculate that it is easier to generate birds, perhaps because birds have fewer structural irregularities across species that make it easier for Discriminator to spot a fake bird than to spot a fake image generated from the Flickr8k dataset.

Due to the diverse nature of the images in the Flickr8k dataset, training was quite slow. The results can be improved, if we train for more epochs and with a large dataset. Initial processing can also be changed if the images can be grouped and classified based on certain factors like scenes, time of the day in the picture captured or seasons etc. This will further help the model learn quicker and better.

7 Results

7.1 Quantitative results

The Loss for the first 500 and 100 epochs are illustrated in the below figures. We can see the generator loss decreasing and both the Generator and Discriminator loss reaching a balance which indicates an optimal solution for both G and D.

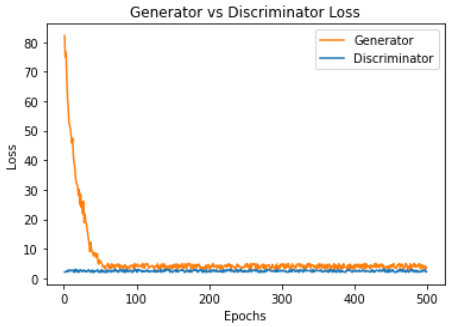

Figure 4: Generator vs Discriminator loss for 0-500 Epochs

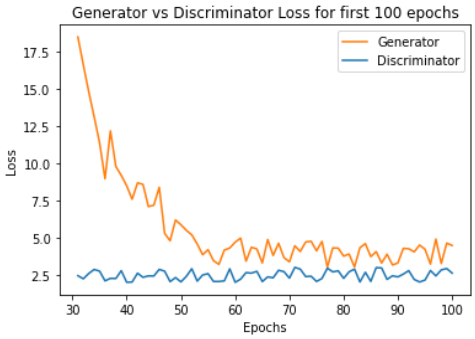

Figure 5: Generator vs Discriminator loss for 0-100 Epochs

Below is the GAN loss for model trained on Flickr8k dataset for 500 epochs.

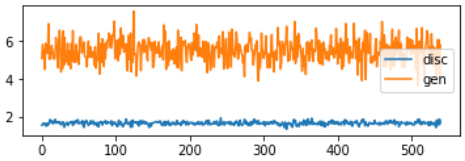

Figure 6: Generator vs Discriminator loss on Flickr8K Dataset for 0-500 Epochs

7.2 Qualitative results

We compared our DR-GAN model with vanilla GAN model on birds dataset and compared the outputs of both the models at 1000 epochs.
Results using vanilla GAN can be seen below. The images have some color information right, but the images do not look real. The model was subject to mode collapse as seen from the output images. However our proposed DR-GAN model was able to produce plausible images with the model trained for 1000 epochs.

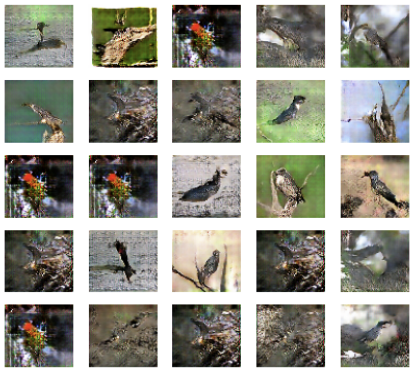

Figure 7: Output of Vanilla GAN on Bird Dataset

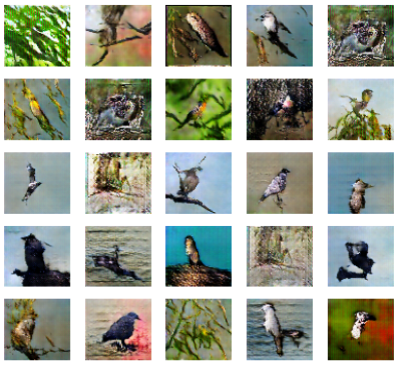

Figure 8: Output of DR-GAN model

In figure: 9 we can see some outputs of the DR-GAN model when the input text was given as "*a yellow bird with black tail*" & "*a green bird with black head*"

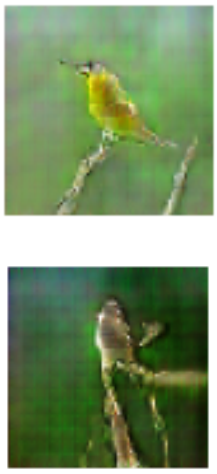

Figure 9: Output of DR-GAN model

Figure 10: Results on Flickr8K dataset

In figure:10 we can see the output of our DR-GAN model when trained for 1000 epochs on the Flickr8K dataset.

7.3 FID Score on Bird dataset

Model Inferencing using metrics in GANs is quite difficult due to its instability and issues like mode collapse. In GAN we generate images from random jitter and want them to be quite realistic. One way is to manually look at the generated images. Thus the evaluation metric needs to evaluate both the quality and relevance to the dataset. Frechet Inception Distance(FID) is one such metric we can use to evaluate GANs. FID is a metric that calculates the distance between feature vectors calculated for real and generated images.

We have used the pre-trained Inception V3 model in Keras which is trained on ImageNet Dataset. Below is the plot showing FID scores for fake and real images for Birds Dataset at various model checkpoints from 60 to 480.

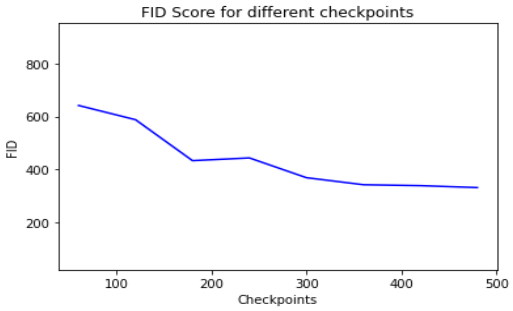

Figure 11: FID score decreases with the model checkpoint.

Overall FID score is relatively high because the Inception V3 is trained on Imagenet while our GAN is trained on the Birds dataset which is not able to capture all features. Training this inception model on a new dataset needs large data and more computational power.

8 Future Work and Conclusion

The performance of our text to image generation model was enhanced using the deep residual GAN model. We trained our model on CUBS 200-2011 and Flickr8k datasets and our model was able to generate images based on input textual embeddings with comparable loss and accuracy. We found that using residual blocks along with convolutional layers in our GAN generator made the model stable and more efficient.

To further study the proposed DR-GAN, the model can be implemented using improved configurations and different embedding techniques for captions like skip-thought vectors.

References

[1] Augustus Odena, Christopher Olah, Jonathon Shlens, Conditional Image Synthesis With Auxiliary Classifier GANs, CVPR 2016

[2] Tingting Qiao, Jing Zhang, Duanqing Xu, Dacheng Tao, MirrorGAN: Learning Text-to-image Generation by Redescription, CVPR 2019

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

[4] Elman Mansimov, Emilio Parisotto, Jimmy Lei Ba & Ruslan Salakhutdinov, Generating images from captions with attention. ICLR 2016

[5] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets”, arXiv preprint arXiv:1411.1784, pp. 1–7, 2014.

[6] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas, StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks

[7] https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

[8] https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/

[9] https://machinelearningmastery.com/a-gentle-introduction-to-the-biggan/