NEPALI HANDWRITTEN LETTER GENERATION USING GAN

Basant Babu Bhandari[1], Aakash Raj Dhakal[1], Laxman Maharjan[1], Asmin Karki[1]

[1]*Department of Computer Engineering, Khwopa College of Engineering, Bhaktapur, Nepal*

Abstract

'he generative adversarial networks seem to work very effectively for training generative deep neural networks. The aim is to generate Nepali Handwritten letters using adversarial training in raster image format. Deep Convolutional generative network is used to generate Nepali handwritten letters. Proposed generative adversarial model that works on Devanagari 36 classes, each having 10,000 images, generates the Nepali Handwritten Letters that are similar to the real-life data-set of total size 360,000 images. The generated letters are obtained by simultaneously training the generator and discriminator of the network. Constructed discriminator networks and generator networks both have five convolution layers and the activation function is chosen such that generator networks generate the image and discriminator networks check if the generated image is similar to a real-life image dataset. To measure the quantitative performance, Frechet Inception Distance (FID) methodology is used. The FID value of 18 random samples, generated by networks constructed, is 38413677.145 . For a qualitative measure of the model let the reader judge the quality of the image generated by the generator trained model. The Nepali letters were generated by the adversarial network as required. The evaluation helps the generative model to be better and further enables a better generation that humans have not thought of.

Keywords: Nepali Handwritten Letter Generation, Neural Network, Machine Learning, Deep Convolutional Generative Adversarial Network

1. Introduction

A generative adversarial network (GAN) is a class of machine learning frameworks designed by Ian Goodfellow (Goodfellow, 2014) and his colleagues in 2014. Two neural networks contest with each other in a game. Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics.

Though originally proposed as a form of generative model for unsupervised learning, GANs have also proven useful for semi-supervised learning, fully supervised learning, and reinforcement learning.

We use unsupervised learning; one trains the machine with unlabeled data. This allows for producing output based on previous experience. This implementation maps the input variables to an output variable and uses an algorithm to learn the relationship between them. This involves learning to generate image data that is similar to the real dataset of image. In this implementation, Devanagari characters' image dataset are fed into the GAN as input. The network will then try to generate the characters as accurately as possible.

2. Literature Review

The major research paper for GANs was submitted by Ian Goodfellow (Goodfellow, 2014) in 2014. In this paper, he used the MNIST (Modified National Institutes of Standards and Technology) dataset for testing his proposed framework and generated the handwritten digits. The basic structure of GAN that he proposed is shown in Fig. 1. In which, there is simultaneous training of generator and discriminator networks. The generator network is responsible for generation of image and discriminator network estimates the probability of generated samples are

*Corresponding author: Basant Babu Bhandari
Department of Computer Engineering, Khwopa College of Engineering, Bhaktapur, Nepal
Email: basantbhandari2074@gmail.com
(Received: June 5, 2021, Accepted: August 2, 2021)

similar to real dataset. This project's basic principle inspired from two player game theory. In his result, he made no claim that the resulting samples are better than samples generated by existing methods, but believed that these samples are at least competitive with the better generative models.

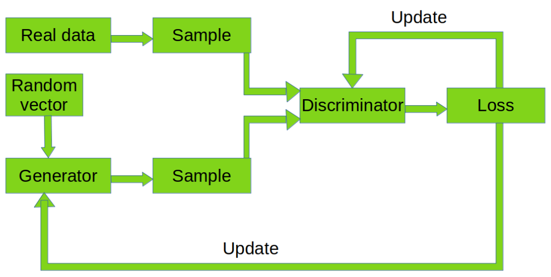

Fig. 1 Basic GAN Model.

Another paper that actually uses deep convolution in generative adversarial training to overcome drawbacks of simple GAN and to show different new things that are possible CNN with unsupervised learning. They observed the huge adoption of convolution networks in supervised learning applications. In order to fill the gap between CNN and unsupervised learning, they proposed a model called DCGAN (Radford et al., 2015) that has certain architecture constraints. They analyzed and explained why GAN is known for being unstable to train.

Generation of Bangla Handwritten Digit using the DCGAN architecture was carried out by Mustapha et al. (2021). In this paper, to achieve the goal they used the three most popular Bangla handwritten datasets CMATERdb, BanglaLekha-Isolated, ISI and their own dataset Ekush. The proposed DCGAN that successfully generates Bangla digits with higher efficiency, which makes it a robust model to generate Bangla handwritten digits from random noise. The losses in each dataset are shown below along with the output. Our project is inspired from this work. The paper is about generating Arabic Letters using DCGAN architecture. The dataset is composed of pictures for 33 alphabets from ’alef’ to ’yeh’, 480 images per character handwritten with image size of 32x32.

Wu et al. (2020) uses DCGAN architecture to generate the Tomato leaf disease image using a given real Tomato leaf diseases dataset so that the total dataset of diseased leaf dataset becomes large. This technique is called dataset augmentation using DCGAN. The traditional augmentation techniques like flip, translation and rotation. Sometimes do not generalize the dataset. They proposed a new method called DCGAN based augmentation. Using this technique, they obtained high model identification accuracy then while using dataset with traditional augmentation methods. They observed that DCGAN gives better convincing results than the t-Distributed Stochastic Neighbor Embedding and visual Turing Test.

Another approach is the conditional version of generative adversarial networks (Mirza & Osindero, 2014). They feed the condition value to the generator and discriminator network. Class labels condition is used in this. The network is forced to train the network such that only conditioned values are generated.

Nvidia scientists (Karras et al., 2017) found a new way to control the generator output. They use a little different approach for training the GAN network. They use the idea in which the generator and discriminator network is progressively growing. Initially they train the computer to learn lesser complex patterns, progressively the complexity increases. They use low resolution image generation techniques and as iteration goes on the network such that the generator network can generate high resolution images.

Although we can generate data that is similar to the real dataset using GAN, we have no control over generator output. The styleGAN (Karras, 2017) based approach uses progressive training GAN and gives fine control over generator output. Although the styleGAN is a great research in the field of GAN, styleGAN2 (Karras et al., 2020) was published in 2020 that discusses more details about improvement of generator output and shows application of such a model. The styleGAN2 is an improvement on styleGAN1. Ian Goodfellow proposed another paper (Goodfellow, 2016) that gives the answer of why GAN is worth studying, how the GAN generative model works, comparison with other models, research frontiers in GAN, state-of-the-art image model and more.

There is plenty of research in the area of GAN. However, it has a limitation when the goal is for generating a sequence of tokens. SeqGAN (Yu et al., 2017) is responsible for sequential data generative GAN.

A learn transformation between two image distributions, another GAN approach is used called cycleGAN (Chu et al., 2017). This GAN learns to transfer information from one source distribution into another source distribution dataset and can be retrieved back and form a cycle thus called cycleGAN. GlyphGAN (Hayashi et al., 2019) approach is used for the different font generation. Using this technique, generation of different consistent fonts becomes easier.

A new GAN is proposed to upsample images from lower resolution to high resolution is called NU-GAN (Kumar et al., 2020). Another GAN (Kovalenko, 2017) was proposed to resample the audio from lower sampling rate to higher sampling rate. Many other techniques that help in the tuning of the GAN networks are also proposed there (Shmelkov et al., 2018). Ian Goodfellow (Goodfellow, 2016) shows all the possible things that are necessary to perform with GAN and shows plenty of applications of GAN in near future.

There are various drawbacks of AI and GAN which are also there and are a serious concern for many scientific researchers explained in the paper (Brundage et al., 2018) and paper (Maas et al., 2013). Using the GAN architecture, there is audio generation (Liu et al., 2020) and video generation techniques are also available.

3. Problem Description

The main objective is to generate Nepali Handwritten letters using a generative adversarial network (GAN). There is lots of work going in the field of unsupervised learning. Different scientists from different countries use DCGAN architecture to generate their own alphabets for different languages. So, we decided to apply the GAN architecture on Nepali Handwritten characters.

4. Dataset

We found the dataset of Nepali handwritten letters on the internet prepared by Ashok Kumar Pant and his colleague in the UCI machine learning repository (Acharya et al., 2015). This consists of 46 classes each class having 10000 images of resolution 32X32. We upsample the resolution with the cv2 library to make the resolution 64X64 then we perform some transformation on it to make it usable and understandable for our model.

5. Methodology

We developed a Nepali Handwritten letter Generator by building the GAN model. Then we pass the training data-set into the model and finally, after training it can output the realistic Nepali Handwritten letter. Here, the diagram consists of the discriminator model and generator model as represented by gray red and light blue color respectively.

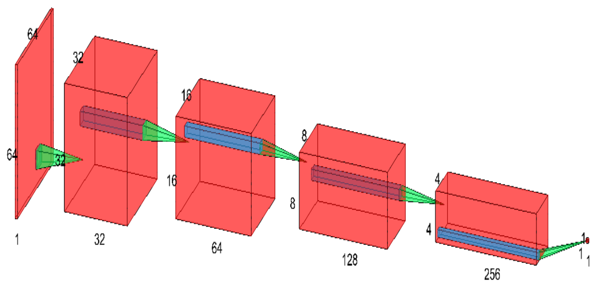

Fig. 2 Discriminator model used for the present study

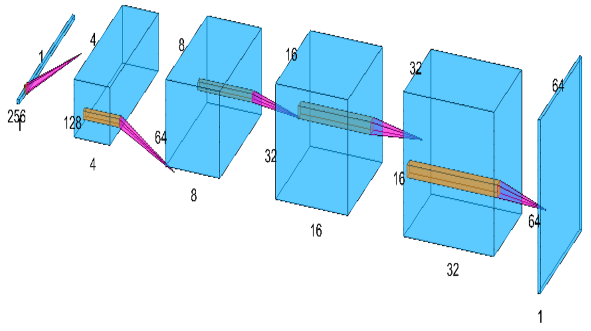

Fig. 3 Generator model used for present study

Fig. 2 is the discriminator model architecture and Fig. 3 represents generator model architecture. Discriminator consists of five convolution layers and the generator network also consists of five convolution layers. Following topics describe all the algorithms and models used in the project in detail.

5.1. Model Development

We attempted to make the model through the GAN model which was based on the Ian Godfellow model, but due to the failure in expectation of desired output using this model. We decided to use convolution techniques to build the network since we have seen its great results with image dataset. We chose DCGAN that used a convolutional neural network rather than a densely connected artificial neural network. PyTorch library is used in the process of developing the GAN model. Pickle is used for data storage and retrieval. The generator and discriminator are defined to train the model. Firstly, the discriminator is defined with five layers. Along all the layers, they have kernel size of four, stride two and padding of one for all. Leaky-ReLU is used as an activation function for the first four layers and sigmoid is used for the last layer. The layer is obtained as (Nx1x1x1). Secondly, the generator is defined with five convolutional layers. Along all the layers, they have kernel size of four, stride two and padding of one for all except the first. ReLU is used as an activation function for the first four layers and tanh is used for the last layer. We obtain the output in the form of (N x channels_img x 64 x 64). Then the hyper parameters are defined. The hyper parameters are learning rate, batch_size, image_size, channel_image, channel_noise, number_of_epochs and number_of_pixels. The channels for discriminator and generator are defined

Afterwards, datasets are loaded. And the numpy array is converted to tensor form. The datasets are filtered. And feature scaling is done to get the required dimension of the image. The optimizers are set up as Adam optimizers and training is done to get the model to perform the desired work. Then the loaded model is saved to get the desired output

5.2. Model Architecture

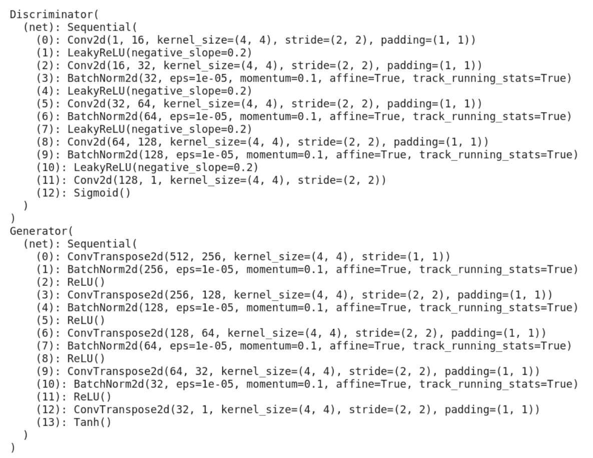
```
Discriminator(
  (net): Sequential(
    (0): Conv2d(1, 16, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2)
    (5): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (6): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2)
    (8): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2)
    (11): Conv2d(128, 1, kernel_size=(4, 4), stride=(2, 2))
    (12): Sigmoid()
  )
)
Generator(
  (net): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU()
    (9): ConvTranspose2d(64, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (10): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU()
    (12): ConvTranspose2d(32, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (13): Tanh()
  )
)
```

Fig. 4 DCGAN architecture in code

For model development of DCGAN we need to develop two CNN networks called discriminator network and generator network. In the discriminator network with five convolution layers, Leaky-ReLU has an activation function except at the final layer to avoid a vanishing gradient problem. For the output layer of the discriminator network we use sigmoid as activation function and for generator network we use tanh as activation function for last layer output. The generator network consists of a five convolution layer. The summary of our model architecture we used is shown in Fig. 4.

5.2.1. Generative model

The input to the generator is typically a vector or a matrix which is used as a seed for generating an image particularly size of 256. Once again, to keep things simple, we'll use a feed forward neural network through the generator CNN layers, and the output will be a batch of 64 images, which consist of each image of size 64×64 pixels image. Since, training in deep learning works with a batch of images, we send a batch of image data to the model for training. This is the random input to the Generator model and output is an image of dimension 64x64. This is a Generator Network Model. We use the tanh() activation function for the output layer of the generator. Note that since the outputs of the tanh() activation lie in the range [-1,1], we have applied the same transformation to the images in the training dataset. Let's generate an output vector using the generator and view it as an image by transforming and de-normalizing the output.

$$L(G) = \min[log_e D(x)] + 1 - log_e D[G(z)] \qquad (1)$$

Equation (1) is the loss function for the generator network based on the discriminator output. And generators need to minimize its value as training goes on. That tells us that the generator is learning to generate the image that is similar to the real image. This formula tells us the difference between the discriminator output for real and synthesized images should be minimum.

The ReLU activation is used in the generator except for the output layer which uses the Tanh function. We observed that using a bounded activation allowed the model to learn more quickly to saturate and cover the colour space of the training distribution.

Within the discriminator we found the leaky rectified activation to work well, especially for higher resolution modelling.

In between each convolution layer we call the normalization function that is standard practice to get better and less failure case outcomes.

5.2.2. Discriminative model

The discriminator takes an image as input, and tries to classify it as "real" or "generated". In this sense, it's like any other neural network. While we can use a CNN for the discriminator, we'll use a simple feed forward network through the CNN layers to get the discriminated output value. We send a batch of images of size 64x64 to the discriminator. Input for the discriminator model will be of 64 images. This is a Discriminator Network Model. We use the Leaky ReLU activation for the discriminator. The output of the discriminator is a single number between 0 and 1, which can be interpreted as the probability of the input image being fake, i.e. generated.

$$L(D) = \max[log_e D(x)] + 1 - log_e D[G(z)] \qquad (2)$$

Here the discriminator needs to maximize Equation (2) function. And based on this value the model knows how to learn about generation of images. Unlike the regular ReLU function, Leaky ReLU allows the pass of a small gradient signal for negative values. As a result, it makes the gradients from the discriminator flow stronger into the generator. Instead of passing a gradient of 0 in the back-prop pass, it passes a small negative gradient.

5.3. Training

Finally, now that we have all the necessary parts of the GAN framework defined, we can train it. The training GANs is somewhat of an art form, as incorrect hyperparameter settings lead to mode collapse with little explanation of what went wrong. Here, we will closely follow Algorithm 1 from Goodfellow's paper (Goodfellow, 2014), while abiding by some of the best practices shown in ganhacks. Namely, we will construct different mini-batches for real and fake images, and adjust G's objective function to maximize $log_e\left[D[G(z)]\right]$. Training is split up into two main parts. Part 1 updates the Discriminator and Part 2 updates the Generator.

The goal of training the discriminator is to maximize the probability of correctly classifying a given input as real or fake. In terms of Goodfellow (Goodfellow, 2014), we wish to "update the discriminator by ascending its stochastic gradient". Practically, we want to maximize $log_e D(x) + log_e(1 - D[G(z)]$. Due to the separate mini-batch suggestion from ganhacks, we will calculate this in two steps. First, we will construct a batch of real samples from the training set, forward pass through D, calculate the loss $log_e D(x)$, then calculate the gradients in a backward pass. Secondly, we will construct a batch of fake samples with the current generator, forward pass this batch through D, calculate the loss $log_e\left[1 - D[G(x)]\right]$, and accumulate the gradients with a backward pass. Now, with the gradients accumulated from both the all-real and all-fake batches, we call a step of the Discriminator optimizer.

As stated in the original paper (Goodfellow, 2014), we want to train the Generator by minimizing $log_e\left[1 - D[G(z)]\right]$ to generate better fakes. As mentioned, this was shown by Goodfellow (Goodfellow, 2014) to not provide enough gradients, especially early in the learning process. As a fix, we instead wish to maximize $log_e\left[D[G(z)]\right]$.

In the code we accomplish this by: classifying the Generator output from Part 1 with the Discriminator, computing G's loss using real labels as GT, computing G's gradients in a backward pass, and finally updating G's parameters with an optimizer step. It may seem counterintuitive to use the real labels as GT labels for the loss function, but this allows us to use the log(x) part of the BCELoss (rather than the $log_e(1 - x)$ part) which is exactly what we want.

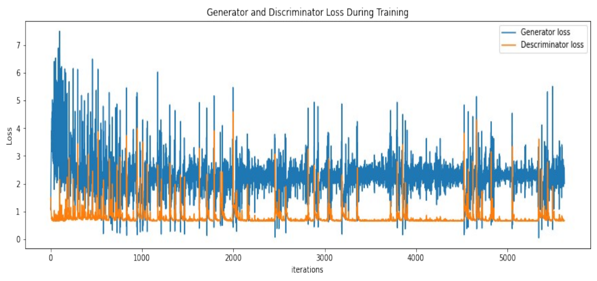

Fig. 5 Loss graph of generator and discriminator

A GAN can have two loss functions: one for generator training and one for discriminator training. In the loss schemes we'll look at here, the generator and discriminator losses derive from a single measure of distance between probability distributions. In both schemes, however, the generator can only affect one term in the distance measure: the term that reflects the distribution of the fake data. So, during generator training we drop the other term, which reflects the distribution of the real data.

The most popular graph in GAN based networks is loss graph, where we plot discriminator loss (orange) and generator loss (blue) while training the network. Here we use the Binary Cross Entropy Loss function. We can clearly observe in Fig. 5, initially for untrained networks the generator has highest loss because it is not trained yet. But as iteration goes on, the generator network learns how to generate images which are similar to the real data-set, the loss of the generator gradually decreases. In case of a discriminator the loss lies between 0 to 1.

6. Results and Discussions

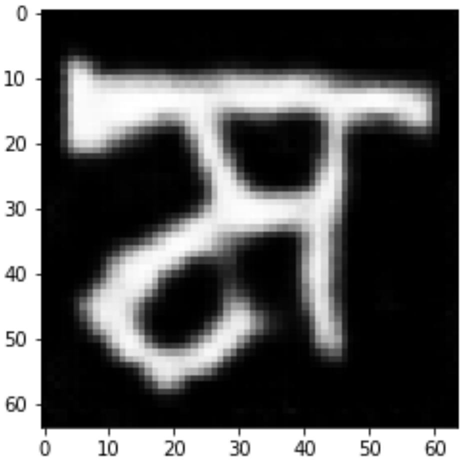

Fig. 6 Synthesized “म”

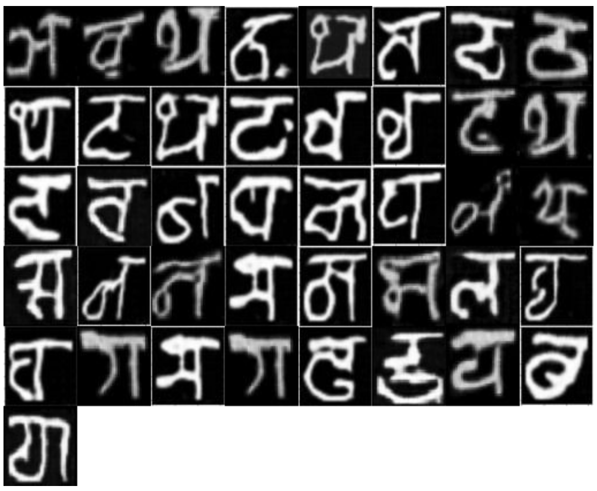

Fig. 7 Synthesized Nepali handwritten letters

The Nepali letters were generated as required. Those Nepali letters were clear and understandable. We use the Frechet Inception Distance (FID) method to do quantitative analysis. Using this method, we get the FID distance of 18 random samples is 38413677.145. We obtained the exact output of the Nepali write text generation. Fig. 6 is an image of “म” generated by our generator network.

Fig. 7 shows 18 random samples generated by generator while Fig. 8 is 18 random real Nepali Handwritten letters. The result shows that the generated sample is quite similar to the handwritten real sample. That means our generator network is well trained.

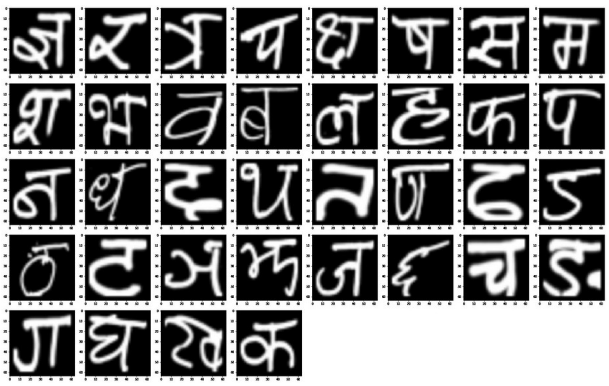

Fig. 8 Real Nepali handwritten letters

7. Conclusions and Recommendations

We demonstrated that generative adversarial networks, and, their deep convolutional variants, function exceptionally well as generative models. We described DCGAN model implementations and showed the realistic Nepali handwritten letter generation.

This study successfully demonstrates the implementation of DCGAN model on real-life datasets. Even though the generated characters were noisy, further fine tuning can be done to give significantly better results. Furthermore, our model architecture can be modified and trained to generate a new kind of Nepali calligraphy. The DCGAN model was trained for 5625 iterations. At each iteration, the output of the model was stored. Despite the noise, the generated characters were readable to the human eye.

Therefore, generative models offer more representational power than their discriminative counterparts. We foresee great future success with GANs, DCGANs, and generative models in general. Our recommendation after doing hand written letter generation using DCGAN project are:

- Generation of high-quality handwritten letters with SeqGAN
- Font generation using GlyphGAN on Nepali Handwritten letters.
- Recognition of generated letters by a child so that we can test the child on generated character.
- Development of other artistic DCGAN models by replacing the dataset.

- DCGAN based Data Augmentation technique
- Realistic result can be possible with Progressive GAN, etc.

References

[1] Acharya, P., & Gyawali. (2015). Deep Learning based large-scale handwritten Devanagari character recognition. In *Proceedings of the 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, (pp.121-126).

[2] Brundage, M. et al., (2018). The malicious use of artificial intelligence: Forecasting, prevention, and mitigation. *arXiv preprint arXiv:1802.07228*.

[3] Chu, C., Zhmoginov, A., & Sandler, M. (2017). Cyclegan, a master of steganography. *arXiv preprint arXiv:1712.02950*.

[4] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems, 27*.

[5] Goodfellow, I. (2016). Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

[6] Hayashi, H., Abe, K., & Uchida, S. (2019). GlyphGAN: Style-consistent font generation based on generative adversarial networks. *Knowledge-Based Systems*, *186*, 104927

[7] Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive growing of GANs for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

[8] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., & Aila, T. (2020). Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (pp. 8110-8119).

[9] Kovalenko, B. (2017). Super resolution with generative adversarial networks. *cs231n. stanford. edu/reports*.

[10] Kumar, R., Kumar, K., Anand, V., Bengio, Y., & Courville, A. (2020). NU-GAN: High resolution neural upsampling with GAN. *arXiv preprint arXiv:2010.11362*.

[11] Liu, J. Y., Chen, Y. H., Yeh, Y. C., & Yang, Y. H. (2020). Unconditional audio generation with generative adversarial networks and cycle regularization. *arXiv preprint arXiv:2005.08526*.

[12] Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013, June). Rectifier nonlinearities improve neural network acoustic models. *In Proc. icml* (Vol. 30, No. 1, p. 3).

[13] Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

[14] Mustapha, I. B., Hasan, S., Nabus, H., & Shamsuddin, S. M. (2021). Conditional Deep Convolutional Generative Adversarial Networks for Isolated Handwritten Arabic Character Generation. *Arabian Journal for Science and Engineering*, 1-12.

[15] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

[16] Shmelkov, K., Schmid, C., & Alahari, K. (2018). How good is my GAN?. In *Proceedings of the European Conference on Computer Vision (ECCV)*, (pp. 213-229).

[17] Wu, Q., Chen, Y., & Meng, J. (2020). DCGAN-based data augmentation for tomato leaf disease identification. *IEEE Access, 8*, 98716-98728.

[18] Yu, L., Zhang, W., Wang, J., & Yu, Y. (2017). Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 31, No. 1).