Proceedings of 10th IOE Graduate Conference
Peer Reviewed
ISSN: 2350-8914 (Online), 2350-8906 (Print)
Year: 2021 Month: October Volume: 10

Generating Nepali Handwritten Letters and Words Using Generative Adversarial Networks

Raj Kiran Chhatkuli [a], Hari Prasad Baral [b], Surendra KC [c]

[a, b, c] *Department of Electronics and Computer Engineering, Paschimanchal Campus*
Corresponding Email: [a] raazkeyrun@gmail.com

Abstract
The power of machine learning in the field of image processing has increased dramatically because of the advancement in deep neural networks. Many works on handwritten English, Arabic, Chinese, and Japanese scripts have previously been completed. So, this paper aims to develop Nepali Handwritten letter and words Generator using Generative Adversarial Networks (GAN). Basically, the Nepali script has 12 vowels, 36 consonant basic forms, ten numeric charcters, and a few special characters. Moreover, there are some special characters in the script. The input that is used is a low resolution noise image i.e 100-dim z-vector generated by a uniform distribution with a range of -1.0 to 1.0. The dataset used for this paper comprises of Nepali handwritten letter and word dataset both in Devanagari script. And using those dataset in GAN Network it was discovered that recognizable and readable Nepali handwritten letters and words could be constructed by merely beginning from noise.

Keywords
Word Generation, DNN, GAN, Latent Space

1. Background

The Brahmi script gave rise to Devanagari. Scholars speculate that the word Devanagari is a mix of two Sanskrit words, 'Deva' (God, ruler, or Brahmans) and 'Nagari' (language) (city). Devanagari is the only script with particular signs (graphemes) for the phonetically organized sounds of human speech (phonemes), yet it is flexible enough to represent foreign sounds by attaching marks to the closer grapheme.

GANs have become increasingly popular among researchers in recent years due to their ability to learn high-dimensional, complex data distributions in the field of recognition [1] and generation. GANs are part of the generative models family. However, unlike autoencoders [2], given any encoding, generative models can generate new and meaningful outputs. By training two competing (and cooperative) networks referred to as generator and discriminator, GANs can learn how to simulate the input distribution (also sometimes known as critic). The generator's main job is to continuously figure out new ways to create bogus data or signals (including audio and images) that can mislead the discriminator. Similarly, the discriminator has been taught to differentiate between bogus and real images. The discriminator will no longer be able to distinguish between the synthetically generated data and the actual ones as training advances. From there, the discriminator can now be removed, and the generator can be used to generate new realistic images or outcomes that have never been seen before.

The figure 1 below depicts the Generative Adversarial Network's core design.

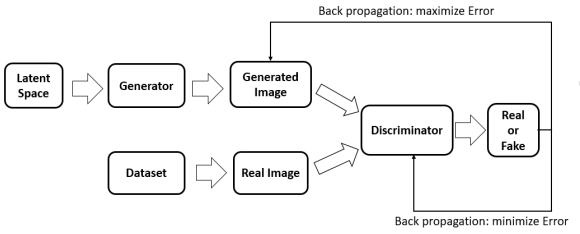

Figure 1: Basic Architecture for GAN.

A generative adversarial network (GAN) has two parts: The generator, which learns to generate plausible data, and the discriminator, which learns to tell the difference between the generator's fake data

and genuine data. When training begins, the generator generates clearly bogus data, which the discriminator soon recognizes as such. Both the generator and the discriminator are none other than neural networks. Here, the discriminator input is directly connected to the generator output. The discriminator's output or classification gives a signal that the generator uses to update its weights via backpropagation.

Although several papers has been emerged for classification of other languages such as hindi and English letters and as well as Nepali handwriting letter classification using CNN, there still lack a generator for Nepali handwriting letters and words. As a result, the goal of this article is to use Generative Adversarial Networks to create a Nepali Handwriting Letter Generator and a word generator. Thus, the methods that will be employed in this project begin with automatic pre-processing. Then the combination of generator & discriminator handles the generation part. For betterment in the result we could use other refinement techniques if required. We will use these two different datasets for letters and words separately in Devanagari script.

2. Related Work

i) Devanagari Script Character Recognition using Genetic Algorithm: The automatic or digital translation of handwritten, typewritten, or printed text images into machine editable text is referred to as character recognition in this paper[3]. The input image is scanned and then proceeded further for noise removal. The images are then normalized in the form of grayscale and is converted to binary image by taking some threshold value and the obtained binary formed image is then extracted to provide a shape to them by thinning mechanism. Furthermore, the pre-processed image then goes through a segmentation which provides a character image with some information. Then, finally the feature extraction technique is used to remove that information from the image. Different problems occurred in the system could have been solved even if genetic algorithm is not used in this paper.

ii) Offline Hindi Handwriting Character Recognition: This paper [4] is primarily concerned with the recognition of the characters created by a human when writing with pen/pencil on paper, which is then scanned into digital format via a scanner. The most common neural network which is used in this system is multilayer perception with feed forward networks. It's still unclear how the combination technique can properly exploit the strength of sub-classifiers and deal with the balance between effectiveness and combinations. The majority of the errors that occur during the recognition of printed characters are caused by erroneous character segmentation of touched or broken characters.

iii) Nepali Handwriting Recognition using Convolution Neural Network: This paper [5] was proposed to analyze and recognize handwritten Nepali character using Convolution Neural Network. The preliminary experiment included 92 thousand photos of 46 different classes of 32 * 32 characters of Nepali handwriting that were subjected to several pre-processing phases such as clipping and cropping, grayscale conversion, and feature extraction, among other things. The recognition has been experimented with the help of template matching technique. In character recognition, the convolution neural network model outperforms the Feed Forward neural network.

iv) Development of English Handwritten Recognition Using Deep Neural Network: The goal of this paper [6] is to create a DNN-based offline handwritten recognition system. First, MNIST and EMNIST, two popular English digits and letters databases, were chosen to provide dataset for DNN training and testing. There are ten digits [0-9] and 52 letters [a-z, A-Z] in total, and the proposed DNN uses two auto encoder layers and one softmax layer stacked on top of each other. Performance comparision has been done among patternnet, feedforwardnet, and proposed DNN.

v) Bangla Handwritten Digit Recognition and Generation: Here a Semi-Supervised Generative Adversarial Network or SGAN, was used to generate Bangla handwritten numbers in this article [7], and it successfully created Bangla digits. On the BHAND dataset, the architecture achieved a validation accuracy of 99.44%, outperforming Alexnet and Inception V3 architecture.

vi) A System for Offline Character Recognition Using Autoencoder Networks: This article [8] builds DNNs by stacking Auto-encoders that have been trained in an unsupervised layer-wise method. Then, using two and three hidden layer DNNs, they did supervised fine-tuning to train the complete network and gave findings on Consonant and Vowel Modifier Datasets. After combining classifiers to form an ensemble classifier

of four different two hidden layer networks, it has 94.25% accuracy on consonant data and 94.1% on Vowel Modifier Dataset, which increases to 95.4% for consonant and 94.8% for Vowel Modifier Dataset.

3. Methodology

3.1 Model Development

Here the model aims to produce realistic looking handwritten images beginning with the low resolution noise input as previously done in earlier research based on handwritten images in order to stabilise GAN [9]. And replacing noise by some input, this paper aims that it would be a base to reconstruct even the distorted handwritten letters and words from old historical books and papers. This can be achieved using a deep learning approach which is shown in figure 2.

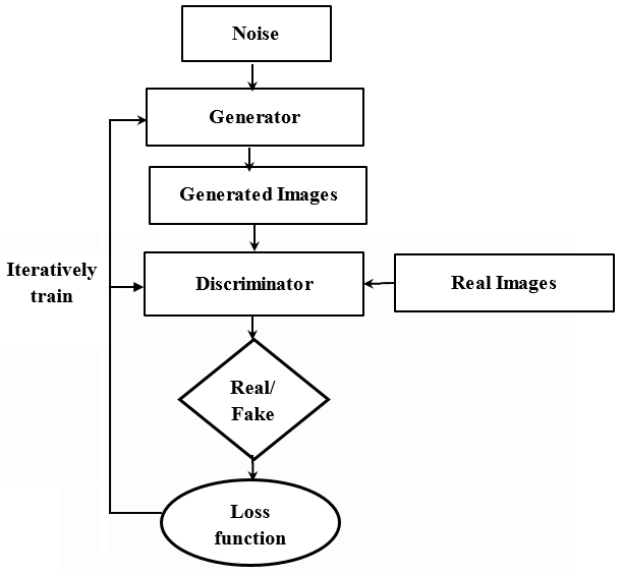

Figure 2: Research methodology.

3.2 Datasets

We use Devanagari (Nepali) Handwritten Character Dataset, which contains 92 thousand images of 46 different classes of Devanagari script characters, and Devanagari Dataset (IIIT-HW-Dev), which has over 95K handwritten words, as our dataset. And this dataset will be used to train in order to generate Nepali letters and words. The handwritten datasets were preprocessed first. And then we reshape it again using our methods during this experiment. The dataset samples used are shown in figure 3.

Figure 3: Dataset Samples

3.3 GAN model

In domains like semi-supervised learning and image to image translation, GANs are the tip of the spear, and they're commonly implemented using two neural networks: the Generator and the Discriminator. These two models then compete against one another in a game context. The GAN model would be trained using both real data and generated data. The discriminator's main task is to distinguish between bogus and real data. Because the generator is merely a learning model, it is likely to produce low or even fully noisy data that does not match the true distribution or features of the real data at first. Our generator is essentially a decoder that receives random Gaussian vectors (100 elements of noise) and generates 28x28x1 images, which are then handed to the discriminator, a normal CNN, to determine whether they are real or false.

The model starts with some noise, commonly Gaussian noise, and outputs an image in the form of a vector of pixels. After then, the generator must figure out how to deceive the discriminator and win a positive categorization (produced image classified as real). When any of those created images is successfully recognized as "fake" by the discriminator, the loss of the creation step is computed. The discriminator must gradually learn how to distinguish between real and fraudulent images. When the model fails to distinguish a bogus image, the discriminator is assigned a negative loss. The basic principle that will be used here is that the generator and discriminator are both trained at the same time.

The basic GAN model architecture that we have used in this paper consists of usual elements as shown below in figure 4.

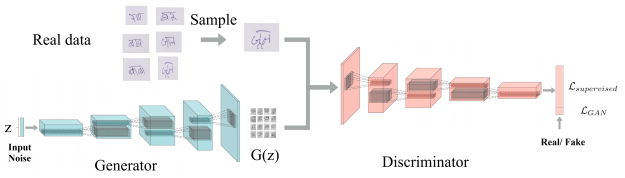

Figure 4: Used GAN Model Architecture.

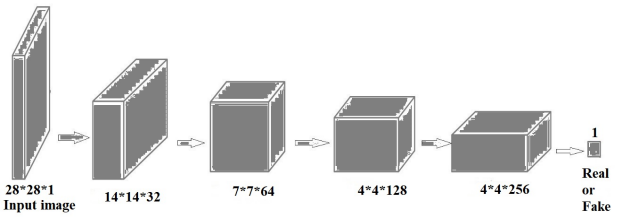

Figure 5: Discriminator Architecture.

As shown in figure 5 above, the input of discriminator is the real/generated images of size 28×28×1. There are four CNN layers. Except for the last convolution that uses strides=1, each Conv2D uses strides = 2 to down sample the feature maps by two. The size of the kernel is 5. Similarly the second, third and fourth layer consists of 2D convolution with filter size of 64, 128, and 256 respectively. Each convolution is followed by Leaky ReLU as activation function with alpha value of 0.2. The output of fourth layer is then fed to dense layer, which is fed to dense units of 1 as output for real and fake output using sigmoid activation function. The learning rate used is 2e-4 while the decay is 1e-4. The learning rate of the adversarial is set equal to half of the discriminator so that it will result in a more stable training.

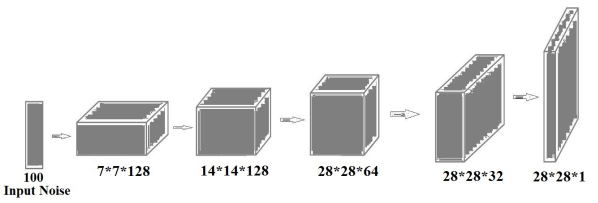

Figure 6: Generator Architecture.

As shown in figure 6 above, the Generator starts with a noise vector z. The generator learns to generate fake images from 100-dim input vectors. The discriminator then classifies real from fake images but inadvertently coaches the generator how to generate real images when the adversarial network is trained. The kernel size used in our implementation is 5. The generator accepts the 100-dim z-vector generated by a uniform distribution with a range of -1.0 to 1.0. The first layer of the generator is a 7 × 7 ×128 = 6,272 unit dense layer. The number of units is computed based on the intended ultimate dimensions of the output image (28 × 28 × 1, 28 is a multiple of 7) and the number of filters of the first Conv2DTranspose, which is equal to 128. We can also assume transposed CNNs (Conv2DTranspose) as the reversed process of CNN. Then we reshape the vector into a three-dimensional hidden layer with a small base (width × height) and large depth. Using the transposed convolutions, the input is progressively reshaped such that its base grows while its depth decreases until we reach the final layer with the shape of the image we are seeking to synthesize, 28 × 28 × 1. After undergoing two Conv2DTranspose with strides = 2, the feature maps will have a size of 28 × 28 × number of filters. Each Conv2DTranspose is preceded by batch normalization and ReLU. The final layer has sigmoid activation that generates the 28 × 28 × 1 fake images. At the final layer, we do not apply batch normalization and, instead of ReLU, we use the sigmoid activation function. GAN views the total of the discriminator and generator losses as a zero-sum game for training the generator. The negative of the discriminator loss function is the generator loss function:

$$L^{(G)}(\theta^{(G)}, \theta^{(D)}) = -L^{(D)}(\theta^{(G)}, \theta^{(G)}) \quad (1)$$

This can then be rewritten more aptly as a value function:

$$V^{(G)}(\theta^{(G)}, \theta^{(D)}) = -L^{(D)}(\theta^{(G)}, \theta^{(D)}) \quad (2)$$

The generator training criterion can be written as a minimax problem:

$$\theta^{(G)*} = arg \min_{\theta^{(G)}} \max_{\theta^{(D)}} V^{(D)}(\theta^{(G)}, \theta^{(D)}) \quad (3)$$

Due to custom training, we do not use the usual fit function. The generator is then trained via an adversarial network. The training first randomly picks a batch of real images from the dataset. This is labeled as real (1.0). Then a batch of fake images will be generated by the generator. This is labeled as fake (0.0). The two batches are concatenated and are used to train the discriminator.

The two networks are trained alternately for about 20,000 steps. At regular intervals, the generated results based on a certain noise vector are saved on the google drive. The generator model is also saved on a google drive so we can easily reuse the trained model for future generation purpose.

3.4 The Training Objective Function

A minimax function can be used to represent the objective function in this case. We can execute gradient ascent on the objective function since the discriminator tries to maximize the objective function. Because the generator's primary goal is to minimize the objective function, we may use gradient descent to

reduce the objective function's size. Finally, the network can be trained by alternating between gradient climb and descent. Considering the GAN problem a min-max game, the loss function for the Discriminator is represented as:

$$J^{\mathrm{D}} = \mathbb{E}_{x\sim p_r} log[D(x)] + \mathbb{E}_{z\sim p_g} log[1 - D(G(z)] \quad (4)$$

Here, $\mathbb{E}_x$ denotes expectation across either x (true data distribution) or z (latent space), D denotes the Discriminator's function (mapping picture to probability), and G the Generator's function (mapping latent vector to an image). This first equation is somehow similar to any binary classification problem. Getting rid of the complexity, we can rewrite this equation as follows:

$$J^{\mathrm{D}} = D(x) – D(G(z),\ for D(x), DG(z)\ \ \varepsilon(0,\ 1) \quad (5)$$

Above equation shows that, our Discriminator is attempting to reduce the chances of mistaking a real sample for a fake one (first part) or a fake sample for a real one (second part). Similarly, the Generator's loss function is:

$$J^{\mathrm{G}} = -J^{\mathrm{D}} \quad (6)$$

We only have two agents and they are competing against each other, it is obvious that the Generator's loss would be a negative of the Discriminator's. As a result, we have two loss functions, one of which is the inverse of the other. The adversarial nature here is clear. The Generator is trying to outsmart the Discriminator.The Discriminator is nothing more than a binary classifier. The Discriminator also only produces one number, but not the binary class. As a result, it is penalized for its arrogance.

3.5 Experimental tools

The tools and software's used in this project work are listed below:

a) Python

- Tesorflow Library
- Keras
- Python Imaging Library

b) Google Colaboratory

c) Kaggle

The whole experiment is carried out in Google Colaboratory. Kaggle is used to store and load the datasets into our colab notebooks. Tesorflow is a machine learning platform that is open source and it's preferable to think of it as a vast and adaptable ecosystem of tools, libraries, and other resources that enable high-level API processes. This framework provides different layers of concepts from which you may pick to develop and deploy machine learning models. Keras, on the other hand, is a high-level neural network library that runs on top of TensorFlow. Keras enables for easy and quick development as well as smooth execution on GPU and TPU when used in deep learning. This framework is written in Python programming language, which is simple to code, debug and as well as it enables for straightforward expansion.

4. Results

4.1 Training Details

The constructed model is able to generate the Nepali handwritten Letters and words in Devanagari script. The results were obtained by implementing the training parameters as follows.

Input image size: 32

Image re-size: 28

Batch size: 64

Latent size: 100

Learning rate: 2e-4

Decay: 6e-8

Optimizer: RMSprop

Training steps: 20000 for letter and 15000 for words

Discriminator and Adversarial Loss: 'binary cross-entropy'

Using these parameters, we train the discriminator and generator networks alternatively by batch. At first, the discriminator is trained with properly real and fake images. And then adversarial network is trained next with fake images pretending to be real. And then the results were recorded on these hyper-parameters.

4.2 Output

The output images obtained for the Nepali letter datasets were saved in filesystem. The output as per training steps are shown below as 1000, 5000, 10000, 20000, 25000 and 30000 training steps respectively.

Figure 7: Output results for Nepali letters in progressive training steps.

The output as per training steps for Nepali words are shown below as 1000, 5000, 10000, 20000, 25000 and 30000 training steps respectively.

Figure 8: Output results for Nepali words in progressive training steps.

The output as per training steps for Nepali complex words are shown below as 1000, 5000, 10000, 20000, 25000 and 30000 training steps respectively.

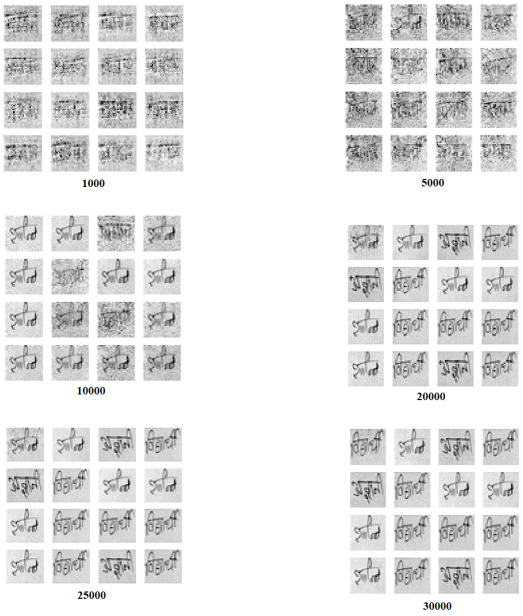

Figure 9: Output results for Nepali complex words in progressive training steps.

The discriminator vs generator loss and discriminator vs generator accuracy for Devanagari letters for first 5000 Training steps were plotted and is shown in figure 10 and 11 respectively

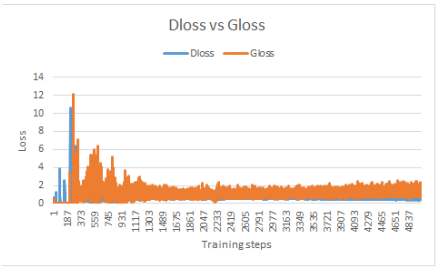

Figure 10: Dloss vs Gloss for Nepali letters

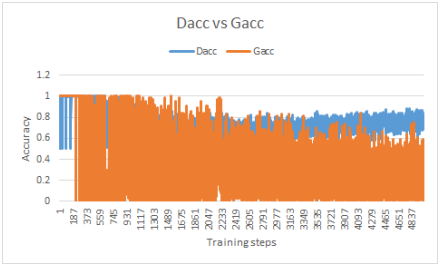

Figure 11: Dacc vs Gacc for Nepali letters

The discriminator vs generator loss and discriminator vs generator accuracy for Devanagari words for first 5000 Training steps were plotted and is shown in figure 12 and 13 respectively

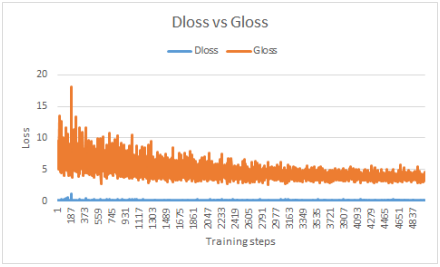

Figure 12: Dloss vs Gloss for Nepali words

As shown in figure 12, the generator and discriminator loss is comparatively higher at the beginning which comes to decline at increasing no of training steps. And then it seems to be constant with some slight changes only on further progressive training steps. As there seems some variance in generator and discriminator losses, it is so because the generator and discriminator are competing against one another and with the betterment of one there occurs the increase in the loss of the other.

Basically, the major goal of deep learning models is all about minimizing the losses. But in the case with GANs and especially handwritten images it is not favourable. It is so because a lower generator loss does not always imply a higher image quality. The generator loss is eventually compared to the present discriminator, which is actually improving all the time. As a result, the loss function at different places cannot be compared.

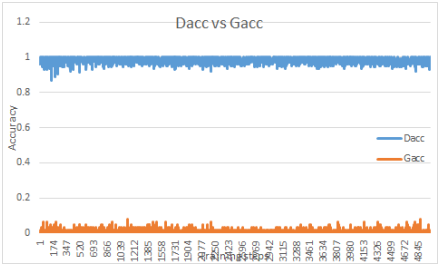

Figure 13: Dacc vs Gacc for Nepali words

The discriminator accuracy for detecting actual photos begins at a relatively high level, whereas the accuracy for detecting fraudulent images begins at a low level. At the completion of the training phases, the accuracy oscillates and becomes stable with variance, as illustrated in figure 13. High-quality images are produced, When the generator and discriminator are in a stable state of operation, , however whenever there is a significant variance loss, then lower-quality images are produced.

The discriminator vs generator loss and discriminator vs generator accuracy for Devanagari complex words for first 5000 Training steps were plotted and is shown in figure 14 and 15 respectively

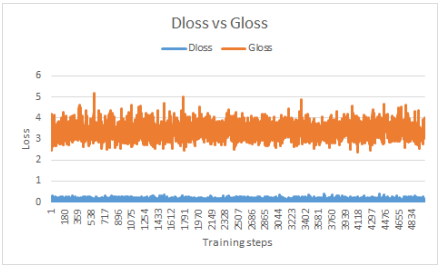

Figure 14: Dloss vs Gloss for Nepali complex words

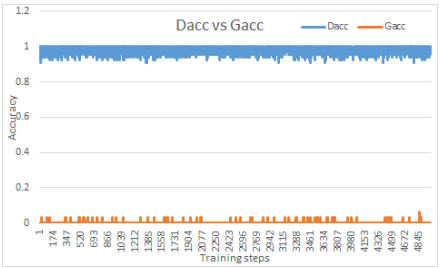

Figure 15: Dacc vs Gacc for Nepali complex words

5. Discussions

The handwritten word datasets were of different sizes initially. They were rehaped automatically into shape of 28*28. Large no of experiments were performed before the correct output appeared. On using the normalization operation on the individual image in the word dataset while training, output images appeared to be completely black. Output obtained due to use of normalization at 5000 and 10000 training steps as shown below:

Figure 16: Output obtained due to normalization

Also while using our handwritten pre-processed dataset as RGB i.e as value of 3 and size of 256, the

blue color distorted outputs were obtained. Output obtained due to this experiment at 4000 and 5000 training steps as shown below:

Figure 17: Output obtained while taking image as RGB

The word dataset used is of the format- black letter in a gray background. So, on reshaping and normalizing step during training, the images got distorted into dark black colors due to which the output appeared all black as the noise gets trained on that particular dark image on progressive training steps. Due to this factor, we were not able to generate the desirable output. So after eliminating normalization step and using the value gray for 'cmap' while plotting the images, we were able to generate Nepali letters and words after several training steps.

6. Conclusion

Although the images generated by the proposed model are not real, they are easily recognizable and readable as real letters and words. And this is an impressive achievement, given that proposed model only used a simple two-layer network architecture for both the Generator and the Discriminator.

The proposed GAN framework is incredibly adaptable and can be used to solve variety of fascinating problems.

Finally, GAN Network was applied in this research to the problem of Nepali handwriting generation and discovered that Nepali handwritten letters and words could be constructed by merely beginning from noise.

This thesis can be improvised in future by improving the quality of images generated and can be further implemented in order to reconstruct letters and words.

References

[1] Yash Gurav, Priyanka Bhagat, Rajeshri Jadhav, and Swati Sinha. Devanagari handwritten character recognition using convolutional neural networks. In *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, pages 1–6. IEEE, 2020.

[2] Tomoki Yamada, Mariko Hosoe, Kunihito Kato, and Kazuhiko Yamamoto. The character generation in handwriting feature extraction using variational autoencoder. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 1019–1024. IEEE, 2017.

[3] Vedgupt Saraf and DS Rao. Devnagari script character recognition using genetic algorithm for get better efficiency. *International Journal of Soft Computing and Engineering (IJSCE) ISSN*, pages 2231–2307, 2013.

[4] Madhuri Yadav, Ravindra Kumar Purwar, and Mamta Mittal. Handwritten hindi character recognition: a review. *IET Image Processing*, 12(11):1919–1933, 2018.

[5] Anuj Bhardwaj et al. Handwritten devanagari character recognition using deep learning-convolutional neural network (cnn) model. *PalArch's Journal of Archaeology of Egypt/Egyptology*, 17(6):7965–7984, 2020.

[6] Teddy Surya Gunawan, AFRM Noor, Mira Kartiwi, et al. Development of english handwritten recognition using deep neural network. *Indonesian Journal of Electrical Engineering and Computer Science*, 10(2):562–568, 2018.

[7] Md Fahim Sikder. Bangla handwritten digit recognition and generation. In *Proceedings of International Joint Conference on Computational Intelligence*, pages 547–556. Springer, 2020.

[8] Sagar Dewan and Srinivasa Chakravarthy. A system for offline character recognition using auto-encoder networks. In *International Conference on Neural Information Processing*, pages 91–99. Springer, 2012.

[9] Kensuke Nakamura, Simon Korman, and Byung-Woo Hong. Stabilization of generative adversarial networks via noisy scale-space. *arXiv preprint arXiv:2105.00220*, 2021.