Devanagari and Gurmukhi Handwritten Character Generation using Generative Adversarial Networks

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By
Simerpreet Kaur
(Roll No. 801632048)

Under the supervision of:
Dr. Karun Verma
Assistant Professor, CSE Department

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
PATIALA – 147004

June 2018

CERTIFICATE

I hereby certify that the work which is being presented in the dissertation entitled, "*Devanagari and Gurmukhi handwritten character generation using Generative Adversarial Networks*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Karun Verma* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the dissertation has not been submitted for award of any other degree of this or any other University.

(Simerpreet Kaur)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Karun Verma)
Assistant Professor,
CSED

ACKNOWLEDGEMENT

I would like to express my grateful thanks to my supervisor **Dr. Karun Verma**. This work would not have been possible without his encouragement and guidance. I thank my supervisor for his time, patience, discussion and valuable comments. His enthusiasm and optimism made this experience both rewarding and enjoyable.

I am equally grateful to **Dr. Maninder Singh**, Professor and Head of Computer Science and Engineering Department, an excellent teacher and a well-credited researcher, who always encouraged us to keep going with the work.

I will be failing in my duty if I don't express my gratitude to **Dr. S.S. Bhatia**, Senior Professor and Dean of Academic Affairs, Thapar Institute of Engineering and Technology, for making all the provisions of infrastructure such as library facilities, computer labs equipped with net facilities, immensely useful for learners to equip themselves with latest in the field.

I am also thankful to entire faculty and staff members of Computer Science and Engineering Department for their direct-indirect help and cooperation, which made my stay at Thapar Institute of Engineering and Technology memorable.

Date: June, 2018 (Simerpreet Kaur)

Place: Thapar Institute of Engineering and Technology

ABSTRACT

Today, computers have influenced the life of human beings to a great extent. The theory that computers can learn without being programmed to perform a specific task, have attracted the researchers to see if computers could learn from data. As deep learning became popular, the need for huge amounts of data has risen. The major problem faced in the area of deep learning is the data availability. In this dissertation, a generative technique is used to generate the handwritten Gurmukhi and Devanagari characters. This dissertation describes the implementation of handwritten Gurmukhi and Devanagari characters using deep learning technique named as Generative Adversarial Networks. Gurmukhi and Devanagari script is chosen for this research work as it is used directly or indirectly by more than 500 million people in the Indian subcontinent and less work is done on Devanagari and Gurmukhi script as compared to work done on other scripts such as English and Chinese. GANs are chosen for the implementation as it is proven to be the very powerful generative method. Here, we use 3-layer CNN having stride value of 2 for the feature extraction of handwritten character. The characters generated look like the character in the original dataset. Also by increasing the number of epochs the images are more recognizable and clear as well as the loss starts decreasing.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

GANs	Generative Adversarial Networks
CNN	Convolutional Neural Network
ReLU	Rectified linear Unit
Tanh	Hyperbolic Tangent
DCGANs	Deep Convolutional Generative Adversarial Networks
HCR	Handwritten Character Recognition
SVM	Orthogonal Rotation Invariant Moments
PCA	Orthogonal Fourier-Mellin Moments
RNN	Recurrent Neural Network
HOG	Histogram of oriented gradients
IAN	Introspective Adversarial Network
VAE	Variational Autoencoders
LSTM	Long short Term Memory
DHCD	Devanagari Character Dataset

CHAPTER 1

INTRODUCTION

Handwritten character recognition (HCR) is the process of converting the written text into a symbolic representation. In recent years handwritten recognition has been studied largely for its diverse real-world applications. It is also getting attention because of its usage in various applications in banking, visually impaired users, health-care, and human-robot interaction. To solve handwritten character recognition problem with a regular computer is difficult to solve. For a traditional computer to perform any task a computer program is written with particular instructions. This makes the job of recognizing a handwritten character relatively difficult to do without any kind of machine learning. Variety of handwriting styles by different writers and recognition of isolated handwritten character image with these writing styles creates challenging research problem. The character recognition problem is further complicated due to similarities in different character shapes, the overlaps, the interconnections of the neighboring characters and the complex features of the handwritten characters. Machine learning is an answer for several jobs that are hard to perform with customary computer programs. In this dissertation, Deep learning has been practiced to enhance the recognition accuracy of handwritten characters of Gurmukhi and Devanagari scripts by applying Generative Adversarial Networks (GANs) to generate data.

Handwriting character recognition generally falls into two categories: recognition of online and offline handwritten characters. In recognition of offline handwritten characters, the handwritten characters are scanned and then these scanned images are used for training, testing and validating the neural network model and used for recognition process. Here the stroke information is not given. Whereas in online handwritten recognition process, the values are recorded in parallel while the write operation is performed. Also this information is stored in sequence of strokes in a specific temporal order by the combination of which individual akshara Gurmukhi character is made. In online handwritten character recognition, any pen pointing device such as digital pen, stylus pen are used for writing. The sensors are used to note down the number of strokes used to write a Gurmukhi character. The point when

pen tip touches the screen to the point when pen point is picked up from the screen is considered as one stroke.

Many applications have been provided by handwritten character recognition such as recognizing postal codes, signatures recognition, digital libraries, invoice and receipt processing. Many researchers have been done in research areas such as image processing, pattern recognition, artificial intelligence and cognitive science etc. to solve handwritten character recognition problem. The data availability is a big problem in the machine learning field. There exist different methods for growing and generating datasets, but none are perfect. In this dissertation, GANs is examined in the process of data generation will enhance the results of handwritten recognition.

1.1 Artificial Neural Networks

Artificial Neural Network (ANN) is proven to be a paradigm of information processing that follows the same process as followed by biological nervous system like the human brain processes information. In ANN to solves a complex problem large number of neurons are interconnected in unison. Many specific applications has been configured by the Artificial Intelligence (AI) like pattern recognition, following a learning process.

A neural network takes different approaches to solve a problem than used by traditional computers. In order to solve a problem, traditional computers use algorithmic approach in which computer has to perform a specific set of instructions. Therefore, information in neural networks is processed in a similar way the human brain does, therefore neural network learn by example. Number of layers constitutes to form a neural network. Number of interconnected nodes combined to form neural network layers which contains an activation function. Various patterns are provided as an input to the input layer that is further connected to one or more hidden layers which acts as the brain of neural network where the actual processing is done. The hidden layers then link to an output layer which gives us the required result. A more sophisticated example of neuron is shown in Fig. 1.1. Here the weighted inputs are used, decision of the neural network depends on the particular input weight. The inputs provided to the input layer are multiplied with their respective weight values which

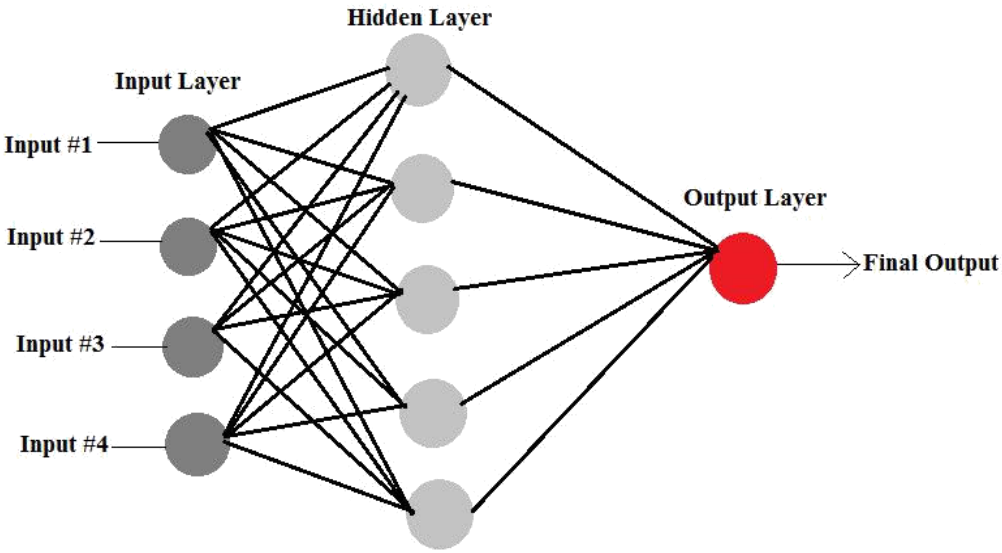


Figure 1.1: A simple neural network

gives the required weighted input values. Summation of these weighted inputs are performed and then if computed sum is greater than pre-set threshold value, the neuron fires. If it is less than pre-set threshold value, the neuron does not fire. A simple example of how multilayer perceptron neural network works is shown in Fig. 1.2 where

$$x_1, x_2, x_3 \dots, x_N \tag{1.1}$$

are the inputs;

$$w_{1J}, w_{2J}, w_{3J} \dots, w_{NJ} \tag{1.2}$$

are the respective weights. Mathematically, the neuron fires if and only if;

$$x_1 w_1 + x_2 w_2 + x_3 w_3 + \dots\dots\dots\dots\dots + x_N w_N > T \tag{1.3}$$

where, T denotes the specified pre-set threshold value. The neuron is made very flexible and powerful by the addition of input weights and threshold. The multilayer perceptron has the ability to adapt to a particular situation by changing its weight and/or threshold value. Backpropagation can be used to update the values of weights and biases at the end of each epoch.

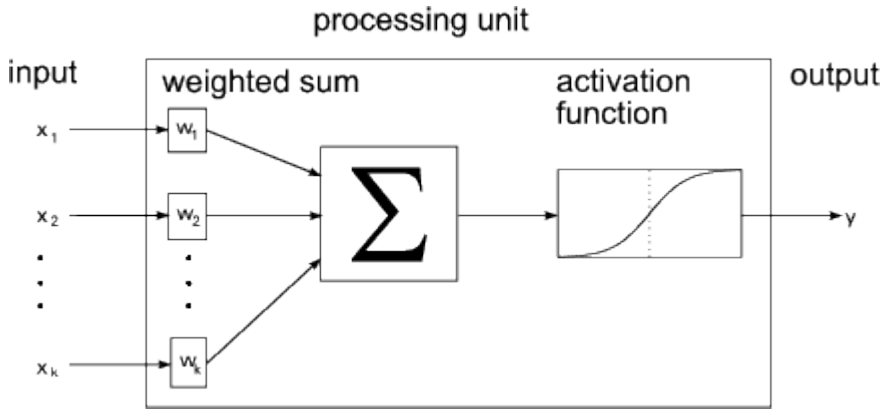


Figure 1.2: Multilayer perceptron neural network [36]

1.1.1 Backpropagation

Backpropagation [1] is considered as a very effective technique of training neural networks and used along with optimization methods like gradient descent. Backpropagation algorithm consists of two-phase cycles which are weight update and propagation. Process followed by backpropagation algorithm is explained ahead.

It consists of two phases, forward pass and backward pass. In forward pass, an input vector is presented to the network which is propagated forward through the network layer by layer until it reaches the output layer. After the output of the network is computed and calculated, then the output of the developed network model is compared to the actual output to compute error loss function for each of the neurons in the output layer. These calculated error values are then propagated backwards, starting from the output to the first neuron input layer until each neuron have their associated error function value that represents their respective contribution in giving the output value. Backpropagation makes use of the calculated error function values to compute loss of the gradient function regarding the weights used in the network. While in the phase second named as backward phase, this computed gradient value is fed to an optimization method which will update the values of biases and weights to minimize the error rate function value and thereby to improve the performance rate of the network model. So as when later on, a new input pattern that may contain noise or incomplete, neurons that are present in the hidden layer of the developed network model respond by giving an active output and thereby recognized efficiently.

1.2 Convolutional Neural Network

The convolutional neural network is also known as CNN or ConvNet is an artificial neural network that is so far been most popularly used for analyzing images. Although image analysis has been the most widespread use of CNN's. They can also be used for other data analysis or classification problem. CNN has a specialization for being able to pick out or detect patterns and make sense of them. This pattern detection is what makes CNN so useful for analysis. The explicit premise is made by the ConvNet architectures that the inputs are images, which enables us to encode some characteristics into the architecture. These then make the forward function more effective to implement and the number of parameters are largely reduce in the network. CNN is differentiated from a standard multilayer perceptron due to the presence of various hidden layers. CNN architecture consists of three main layers as shown in Fig. 1.3 convolutional layer, pooling layer, and regular fully-connected layers. All layers are explained in the coming subsections.

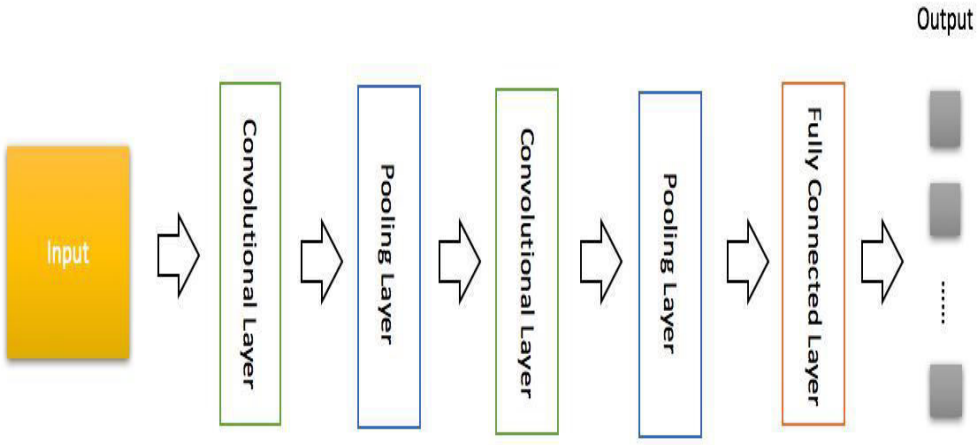


Figure 1.3: Convolutional network architecture

1.2.1 Convolutional Layer

The basis of CNN is a convolutional layer. Just like any other layer convolutional layer receives input and then transform the input in some way and then output the transformed input to the next layer. A convolution is an orderly procedure where two sources of information are intertwined. With each convolutional layer, filters are initialized to each layer and filter can technically just be thought of as a relatively small matrix whose number of

columns and rows are random numbers as shown in Fig. 1.4. Then filter is going to slide across each block from the input and then dot product of filter and input is done which is going to obtain activation maps. Activated regions are shown by Activation maps. These are the regions where kernel specific features are identified as the input. With every learning iteration, over the training set the filter values are changed, showing the important features extraction from the data is learned by the network. For the entire input image, the patch selection is done by sliding (towards the downwards, or right when the end of the matrix is reached) by particular value called the stride value, and the process is repeated.

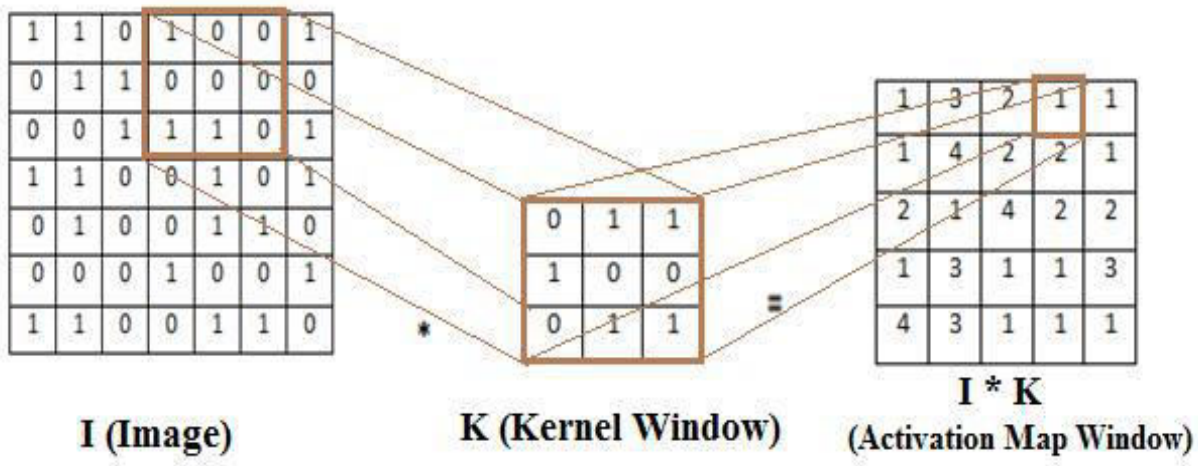


Figure 1.4 Convolutional layer example

1.2.2 Pooling Layer

Pooling decreasing the spatial dimensions (Width × Height) of the Input Volume for the next Convolutional Layer. The depth dimension of the Volume is not affected. The change is either done by picking the highest value amongst the kernel size window called MAX pooling or by picking the average of the values coming under kernel size window in AVERAGE pooling operation. Due to better performance characteristics Max pooling has been preferred over others. It is also called downsampling. Pooling layer is placed in between the two convolutional layers. It also helps to reduce overfitting. In Fig. 1.5, max pooling is used on input volume having kernel size window of 2×2. From each 2×2 kernel size window, the maximum pixel value is picked up for consideration as it is considered as a more relevant

feature from its neighborhood pixels that will be used efficiently to recognize and classify the input.

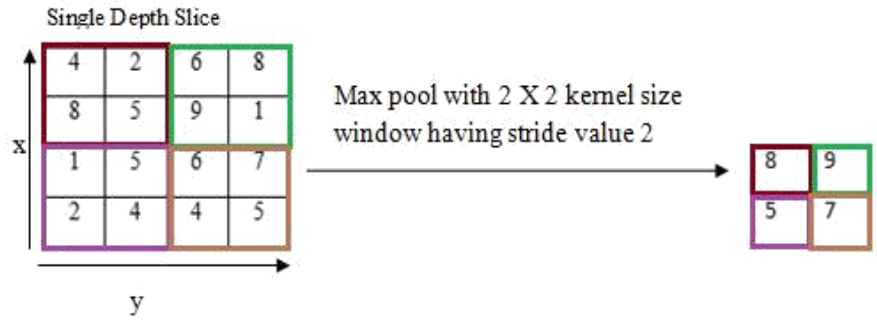


Figure 1.5: Example of max pooling layer having 2 × 2 kernel size

1.2.3 Fully-Connected Layer

Fully connected layer composes of neurons that are fully connected to all the neurons that are present in its previous layer same as in regular neural networks. Then matrix multiplication followed by bias offset is performed which is also called activation. The main difference that lies between the fully connected and convolutional layer is that in the convolutional layer, in the input neurons are connected only to a small local region and also they share parameters. However, the neurons in both the layers still calculate dot products so their working form is same. Fully connected layer receives an input volume which will be the output of its previous layer and it outputs an N-dimensional vector where N indicates the number of classes that the program has to choose from. Each number in this N-dimensional vector describes the probability of a certain class. The way in which fully connected layer works is that it views the output of the previous layer (activation maps of the high-level features) and decides which features must belong to a particular class. Class comprising largest probability will be regarded as the winning class. In simple words, fully connected layer views at what high level features most strongly correspond to an appropriate class and has particular weights so that when you calculate the products between the weights and the previous layer, you get the right probabilities for the different classes.

1.2.4 Activation Layer

Deep neural networks require a lot of non-linear transformation but the convolutional operation is linear in nature. So to deploy the non-linearity into deep neural networks the activation functions are required. They are one of the important factors which affect your

results and accuracy of your model. Activation functions are inspired by the human brain. There are four main activation functions, Sigmoid, Hyperbolic Tangents ($tanh$), Rectified Linear Unit (ReLU) and Leaky ReLU. Sigmoid and $tanh$ are often used in the fully-connected layer. ReLU and Leaky ReLU are often used in the convolutional layer.

- **Sigmoid function:**

The sigmoid function maps the input between the interval 0 and 1. The curve produced by the sigmoid function is of 'S' shape as shown in Fig. 1.6. The function applied by the sigmoid function is:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.4}$$

Sigmoid function mainly causes the problem of the vanishing gradient, while performing gradient-based training.

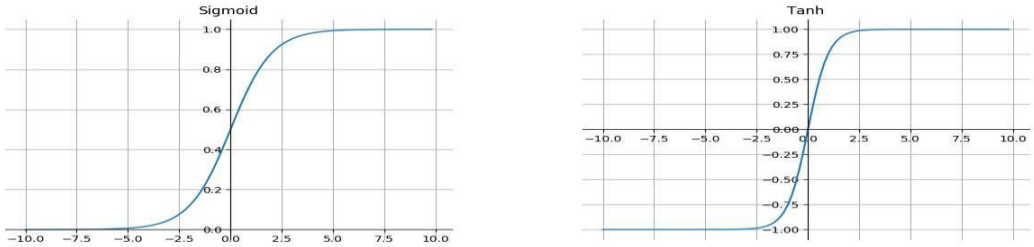


Figure 1.6: Sigmoid (left), $tanh$ (right)

- **Hyperbolic Tangents**:

It is also known as $tanh$. It is nothing but the rescales version of sigmoid as shown is Fig. 1.5. The $tanh$ function maps the input between the interval -1 and 1. The function applied by $tanh$ is:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{1.5}$$

- **Rectified Linear Unit**:

The ReLU has become very popular in the last few years because it overcomes all the disadvantages of sigmoid and $tanh$. Also, ReLU has faster convergence and minimal cost. The value of ReLU ranges between 0 and infinity. The ReLU applies the following function:

$$y = \max(0, x) \tag{1.6}$$

However, the ReLU units can "die" during training, where ReLU always the same value is produced as output for any input.

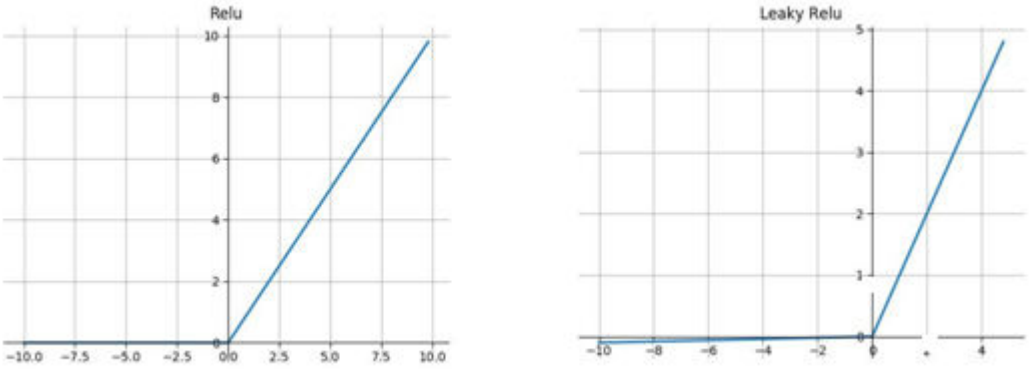


Figure 1.7: ReLU (left), Leaky ReLU (right)

- **Leaky ReLU**

The Leaky ReLU overcomes the dying problem of ReLU. It works similar to ReLU the only difference is that it does not make the negative input 0 just reduce the magnitude of it. The Leaky ReLU layer applies the following function:

$$y = \begin{cases} 0.01 * x, & when\ x < 0 \\ 1, & when\ x \geq 0 \end{cases} \tag{1.7}$$

where x is a small constant.

1.2.5 Dropout Layer:

Dropout is a vital feature in almost every state-of-the-art neural network implementation. Dropout forces an artificial neural network to learn multiple independent representations of the same data by alternately randomly disabling neurons in the learning phase. Dropout layer is used in the CNN to reduce the chance of overfitting and is very powerful regularization technique. To perform dropout on a layer, you randomly set some of the layer's values to 0 during forwarding propagation. Then, go through each of the layers of the network and set some probability of eliminating a node on a network. After, removing the entire ingoing and outgoing layer and at the end, get a smaller network, and then do backpropagation and training on this diminished network.

1.2.6 Batch Normalization:

When training a neural network, normalize or standardize is the preprocessing step before training. This is a step where data get ready for training. Normalization has an objective of transforming the data to put all the data points on the same scale. A typical normalization process consists of scaling the numerical data down to be on a scale from 0 to 1. Some of the numerical data points in our data set might be very high and others might be very low. The larger data points in the non-normalized dataset can cause instability in neural networks. The relatively large input can cascade down through the layer in the network which may cause imbalance gradients which lead to exploding gradient problem. Non-normalized data are drastically harder to train and decrease our training speed.

Even the normalized data can also create problem if one of the weights ends up becoming drastically larger than other weights, while training. This large weight will cause the output from its corresponding neuron to be extremely large. This imbalance will continue to cascade through the neural network causing instability. This is where batch normalization comes into play.

Batch normalization [2] is applied to layers of our choice within our network. While applying batch-norm to the layers the first thing it does is normalize the output from the activation function. Everything in the batch normalization process occurs on a per batch basis, these batches are determined by the batch size of neural network. Therefore normalization is done

before passing the input to the network and batch-norm normalizes the output from the activation functions for an individual layer within our model as well.

1.2.7 Loss Function:

It is a function for finding the optimal gradients/coefficients of a machine learning model and it does so by minimizing the error. It is a way of measuring how well your classification or regression algorithm is working. The loss function is used to guide the training process of a neural network. For classification problem, mean squared error (L2 loss) and cross entropy loss are widely used.

- **Quadratic Cost:**

It is also known as mean squared error. The formula defined for quadratic cost is,

$$Loss(x, y) = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i|^2 \tag{1.8}$$

where x represents n predictions of neural networks, and y represents the real classes of the input images.

- **Cross Entropy:**

The cross entropy of a distribution P with respect to a distribution Q measures how many bits are needed on average to encode data from P with the code that is the optimal for Q. It is defined as

$$Loss(x, y) = -\sum_{i=1}^{n} y_i * \log\frac{\exp(x_i)}{\sum_{j=1}^{n}\exp(x_j)} \tag{1.9}$$

where x is a vector of n predictions of neural networks, and y is a binary vector full of $0's$ besides $1's$ representing the real classes of the input images.

1.3 Generative modeling:

Generative modeling can be defined as a process of taking a collection of training example and forms some representation of probability distribution that explains where these training examples came from. Generative modeling can performs two essential things, one takes a collection of points and density function that describes the probability distribution and

another takes a machine that observes many samples from a distribution and creates samples from the sample distribution. Generative Adversarial Networks (GANs) lie in the second category. Generative models are the class of models for unsupervised learning where given training data generates new samples from the same distribution. Training data is generated from some distribution $p_{data}(x)$ and $p_{model}(x)$ model is learned to generate samples from the same distribution. Generative models address density estimations. Generative models can be used to do explicit density estimation is defined explicitly, and $p_{exp}(x)$ is solved. Also, implicit density estimation can be performed, where a model can learn to produce a sample from $p_{imp}(x)$without explicitly defining it. Generative modeling is an interesting core problem used deep learning due to following reason:

- Generative model time series data can be used for simulation and planning and so this will be used for reinforcement learning applications
- Training generative models enable inference of latent representations that can be used as general features for downstream tasks.
- Generative adversarial models are able to handle missing data much more effectively as they are able to fill missing inputs and are also able to learn when some of the labels in datasets are missing.

1.3.1 Generative model working:

Most of the generative models work using the principle of maximum likelihood. In maximum likelihood, the probability distribution of a model is estimated. Machine learning consists in measuring the large probability density function that assigned to all training data and adjusting the parameters data to increase that probability. In this, the density function that model describes, is written which is represent as $p_{model}(x)$ where 'x' is the vector describing the input and $p_{model}(x)$is the distribution controlled by the parameters of data that describes exactly where the data concentrates.

$$\theta^* = arg\ max \sum_{i=1}^{m} log p_{model}\left(x^{(i)}; \theta\right) \tag{1.10}$$

There were many drawbacks associated with other generative models which were avoided by GANs.

1.4 Generative adversarial networks:

GANs are used to generate data that looks similar to training data. The idea behind GANs as illustrated in Fig. 1.8, has two networks a generator (G) and a discriminator (D), and those two networks are going to compete against each other. Both the networks are trained at the same time like in minimax game. The generator generates images by sampling a vector noise from simple distribution and then this vector is unsampled into an image. The purpose of the discriminator is to classify, the data it received is real or fake, it wants to output data that look as close as possible to real data, the discriminator is trained to figure out which data is real and which is fake. Then through a back-propagation step generator receives the feedback from discriminator, and becomes better at generating images.

.

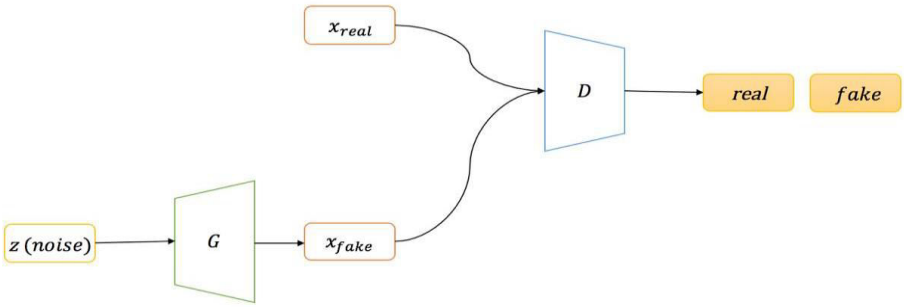


Figure 1.8: The GANs architecture [37]

1.4.1 Adversarial loss:

GANs [24] use the minimax objective function for matching the true probability distribution.

$$V(D,G) = E_{xP_{data}}\left[log\left(D(x)\right)\right] + E_{zP_z}\left[log\left(1 - D\left(G(z)\right)\right)\right] \quad (1.11)$$

The first term corresponds to optimizing the probability data (x) is real which is coming from discriminator, the second term corresponds to the optimizing the probability that the generator data G_z is rated poorly. The lost function used for the discriminate or L_D :

$$L_D(\theta_d) = -E_{x-P_{data}}\left[log\left(D(x;\theta_d)\right)\right] - E_{z-P_z}\left[log\left(1 - D\left(G(z;\theta_d)\right)\right)\right] \quad (1.12)$$

The lost function used for the Generator L_G:

$$L_G(\theta_g) = E_{z\sim P_z}\left[\log(1 - D\left(G(z;\theta_g)\right)\right] \quad (1.13)$$

1.5 Deep Convolutional Generative Adversarial Networks (DCGANs):

There is a lot of research in progress on improving GANs and one of a work that really took a big step towards improving the quality of samples in this work is adding convolution architecture to introduce a new architecture known as Deep Convolutional Generative adversarial Networks (DCGANs). DCGANs consist of the generator which is a deconvolutional net and the discriminator which is a convolutional net.

1.5.1 Architecture of DCGANs:

The initial layers of DCGANs contain a convolutional network in which stride convolution is performed. As shown in Fig. 1.9 unsampling is performed by the generator. Similarly, in the discriminator, the same convolutional layers are present in reverse order as shown in Fig. 1.10.

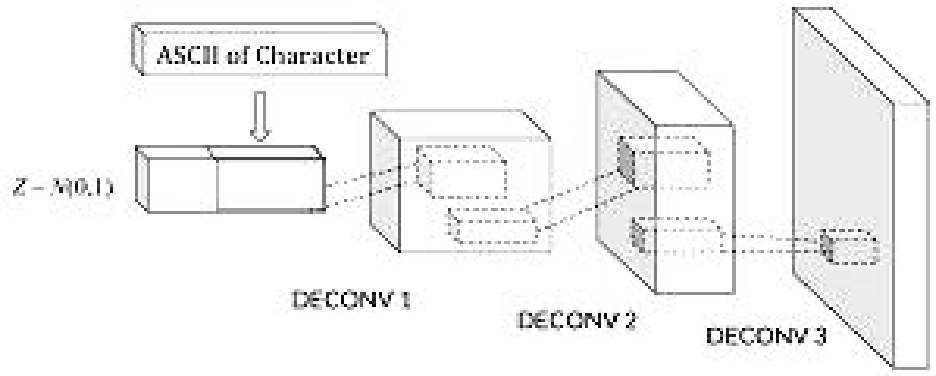


Figure 1.9: The Generator network [32]

In DCGANs the fully connected layers are not present. The input to the generator and the output to the discriminator are the features which are extracted from the highest convolutional layer. For rescaling an input at each layer, batch normalization is applied at discriminator input.

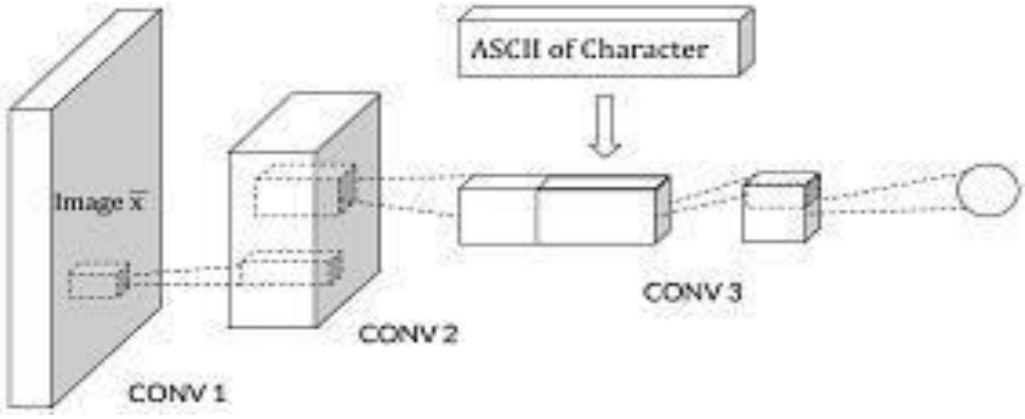


Figure 1.10: The Discriminator network [32]

1.6 Dissertation Outline

The purpose of this dissertation is to develop a system for generating an image by training GANs. The dissertation is organized in the following manner. In the present chapter, the introduction of GANs, CNN, and overview of DCGANs is presented. Chapter 2 describes the literature survey on handwritten recognition of Indic languages and image generation systems using different GANs. Chapter 3 describes the problem statement. In Chapter 4 Devanagari and Gurmukhi script is explained. Chapter 5 illustrates the various phases of preprocessing, training the GANs and generation of output. Results obtained by generating image model by training all these networks described and discussed in Chapter 5. The conclusion drawn from the results obtained is discussed in Chapter 6.

CHAPTER 2

LITERATURE SURVEY

In this Chapter, a brief literature survey on the topics included in this work is presented.
HCR always has been popular research topic, even before recent search of deep learning popularity. A few state of the art approaches that use hand written character recognition for Indian languages have been summarized in this chapter. Recently, another hot research topic has been handwritten data generation. The generative model's field has grown significantly in recent years, fueled to a large extent by the introduction of Generative Adversarial Networks (GANs).

LeCun et al. [3] used backpropagation network for the recognition of handwritten digits. Preprocessing techniques are applied for cleansing and smoothening of the handwritten input images. The input provided to the developed neural network is the normalized handwritten isolated digit images. Error rate of 1% was achieved when neural network having backpropagation is applied on zip code digits.

Verma [4] used radial basis function and multilayer perceptron neural networks for recognizing the handwritten characters of Devanagari script. Backpropagation error algorithm is also used to improve the recognition rate. In this proposed system, they compare the results obtained from radial basis function (RBF) networks and multi-layer perceptron. Dataset consists of 245 samples written by five different users. The results so obtained show that multilayer perceptron (MLP) networks performs better than that of radial basis functions. But MLP networks training time is more as compared to radial basis function networks. Highest recognition rate so obtained using radial basis function and multilayer perceptron (MLP) networks are 85.01% and 70.8% respectively.

Sethi and Chatterjee [5] used structured method for recognizing the numerals of handwritten Devanagari dataset which obtained the positions and the existence of vertical and horizontal line segments, C-curve, D-curve, right and left slant. To analyze the given

input pattern, the frame's background is divided into eight test region. The experiment was done on the Devanagari numerals where the 94% recognition efficiency has been achieved.

Sharma et al [6] used a quadratic classifier based scheme on handwritten Devanagari dataset , where for extracting the features of a character the directional chain code based method was applied. Blocks of sample character were obtained using a bounding box, and from each block 64-D, direction chain code features are computed. The authors reported 80.36% recognition accuracy for the character.

Deshpande et al [7] applied the directional chain code based feature extraction method with a regular expression and minimum edit distance method. In this encoded string is generated from characters, and matching of a regular expression is done with those strings. The samples which are not matched with the pattern are filtered out by minimum edit distance classifier. The authors achieved the accuracy of 81.10% for Devanagari characters.

Arora et al [8] proposed two-state approach for classifying handwritten characters structural properties were deducted during the first stage which includes shirorekha and spin in character. In another stage intersect feature of characters are exploited. The technique used to find a near shirorekha and spine is a differential distance based. Then for further classification, it is fed to the neural network. This approach was tested on 5000 Devanagari characters and reported the accuracy of 89.12%.

Hanmandlu et al [9] reported a fuzzy model where a box approach is used for extracting the features in which characters are divided into 24 cells and for each box, a normalized vector distance was calculated. To increase the learning speed a reuse policy is used on 4750 samples and 90.65% accuracy is obtained. Therefore there is an increase in the speed of convergence by a 25-fold by applying the reinforcement learning.

Pal et al [10] proposed the system named as the quadratic classifier. In this features are extracted using the arctangent of the gradient. Firstly for 4 times 2×2 mean filtering is done, the image is normalized. After normalization, the image is segmented to 49×49 blocks. To obtain the gradient image Roberts filtering is done. Finally, the 392-dimensional feature vector is obtained by applying the Gaussian filter. A 5-fold cross-validation scheme is applied on 36172 handwritten images and accuracy of 94.24% is obtained.

Bhattacharya and Chaudhuri and Bhattacharya [11] provided system for handwritten numeral databases of Indian scripts and also multistage recognition of mixed numeral is performed. Different handwritten scripts are used in this proposed system: Devanagari, Bangla and English. Database consists of 22,556 and 23,392 samples of handwritten Devanagari and Bangla characters collected from various real-life situations. After data collection, these are fed to the three multilayer perceptron classifiers in a cascaded manner. If rejection occurs at highest resolution, then another multilayer perceptron is used to recognize and classify the handwritten characters by combining output results of the previous stages classifiers. Handwritten numerals are collected from tabular form documents and Indian postal mails. Recognition accuracy rate of 99.26%, 98.47%, 97.52% for Bangla, Devanagari and English script was obtained.

Kumar [12] tried five different features named directional distance distribution, distance transform, gradient, chain code, and Kirsch directional edges. The gradient feature performed the best among the remaining four features using SVM as the classifier, and the weakest performer was the Kirsch directional edges feature. After calculating the neighborhood pixels' weight and gradient map a unique kind of feature was also discovered that calculated all the four directions distance from the binary image of the 25,000 Devanagari handwritten characters and 94.1% accuracy was reported.

Arora et al [13] combined different recognition algorithm and feature extraction method. Firstly the preprocessing of character is done and then various features such as shadow based, four side views and chain codes are extracted. Thus features are fed into a multilayer perceptron neural network. The experiment was performed on 1500 Devanagari handwritten characters and accuracy of 89.58% was obtained.

Arora et al [14] deals new method in which two well-known techniques of pattern recognition are used which is applied on a different set of Devanagari handwritten characters for recognition. The two techniques are neural networks and minimum edit distance. The chain code features, shadow features, a minimum edit distance method and classified the samples of 7154 characters using two multilayer perceptron. The 90.74% recognition accuracy is obtained.

Kumar et al [15] have presented four type of topological features which include centroid diagonal, vertically peak extent and horizontally peak extent features for the recognition of 3500 Gurmukhi characters. The further classification is done by SVM classifier. The feature selection technique used for this purpose includes consistency based (COW), correlation-based (CFS) and principal component analysis (PCA). The proposed method obtained 91.60% recognition accuracy.

Verma and Sharma [16] proposed a system for the recognition of handwritten Gurumukhi characters using SVM based stroke classifiers. Two stages are used in this proposed system to classify the images. Strokes are recognized in the first stage. And evaluation of characters is done based on the recognized strokes in the second stage. To improve the recognition accuracy results, features extraction is done. Stroke database is used to store the used strokes information i.e. stroke script number, stroke number and trajectories values corresponding to each stroke. Dataset samples are collected from 82 writers and 92.07% recognition accuracy is achieved. In **[17]** authors compared SVM and HMM for Gurmukhi script containing 1750 characters and obtained accuracy of 96.7% accuracy for SVM and 96.4% for HMM.

Verma and Sharma [18] proposed HCR for Gurmukhi language using zone identification algorithm and SVM. Zone identification is for classification of strokes and SVM is for recognition. The experiment is performed in 40 classes of Gurmukhi dataset containing 428 characters. The strokes recognition accuracy is 93.7% and identification accuracy is 74.8%. In **[19]** authors proposed writing zone identification algorithm on 4280 characters and attained 97.7% writing zone identification accuracy and 88.4% character identification accuracy.

He et al [20] developed ResNet or Deep Residual Networks, is the networks that are supposed to be the greatest and newest in applying CNNs for recognition of the image. It uses batch normalization profoundly and includes skip connections. Currently, ResNets are advanced and when it comes to practicing convolutional neural networks, often the default choice is these networks.

Kaur and Gurm [21] used convolutional neural networks for recognition of machine printed Gurmukhi numerals. Random generation matrix is one of the feature extraction methods that are used in this proposed system. CNN concentrates on the dynamic features of the image. Accuracy of the work is measured with k-means and HOG algorithms. By applying

combination of multi-level pooling and convolutional layers, they achieved recognition rate of 95.67%. HOG and SVM method overcome the main challenges associated with the natural scene images like orientation of the text, different font styles and complex background in the images. Backpropagation is used to reduce the rate of overfitting and to update the values of weights and biases to achieve efficient recognition rates.

Jangid and Srivastava [22] proposed technique using the deep convolutional network (DCNN). In this technique they added a convolutional layer, pooling layer, and fully connected layer and after that weights are updated using backpropagation algorithm. The experiment is performed on ISIDCHAR and V2DMDCHAR datasets and used six different types of adaptive gradient methods to get the faster convergence rate. The highest accuracy obtained for recognition is 98%.

Hinton at el [23] describes an unsupervised learning algorithm for a model to learn to generate both the label and the data where the generative model performs better than the discriminative method on MNIST dataset.

Goodfellow et al [24] proposed the new structure where adversarial training is applied to the generative models. The framework is similar to the minimax game, where there are two models are trained simultaneously. One model is generated which reproduce the data distribution and other is discriminator which estimates the probability of how much the data belongs to the original set. For defining the generator and discriminator a multilayer perceptron are used, for correct weights defining backpropagation is used. Various experiments are performed to determine the potential of the architecture by evaluating the generated sample qualitatively and quantitatively.

Radford et al [25] proposed a new class of CNNs which they called as Deep Convolutional Generative Networks (DCGANs). The author applied new parameter to DCGANs architecture where they have used strided convolutions instead of pooling layers, batch-normalization is used in both the networks, there are no fully-connected hidden layers, Leaky ReLU is used in all the layers of discriminator, ReLU is used in generator is hidden layers and the output layer of generator uses $tanh$ activation function. Model training is done with the batch size of 128 with mini-batch stochastic

gradient descent (SGD), the normal distribution is used for initializing the weights with standard deviation 0.02 and Adam optimizer is used with the learning rate of 0.0002.

Reed et al [26] trained a model which created a new image from the sentences. The algorithm is created by training two neural networks. The experiment was performed on MS-COCO data where it uses the DCGANs architecture. It used the Adam optimizer with the learning rate of 0.0002 doing training for 600 epochs.

Jin et al [27] trained the model by GANs which generate the random anime character. The experiment was performed on clean and good anime dataset with 31255 training samples each of 128×128 image size where they train the image without tag data. The generator and discriminator are similar to SRResNet where generator contains 16 ResBlocks and discriminator contains 10 ResBlocks. Batch normalization is only applied to generator network but removed from the discriminator network. A fully connected layer is added for the attribute classifier as the last CNN layer. Gaussian distribution is used for initializing all weights with 0 mean and 0.2 standard deviations. For optimizing the code Adam optimize algorithm is used with a learning rate of 0.0002 having 64 batch-size for training. For evaluating the model they used Frechet Inception Distance (FID) which decreases with the increase in iterations and image become more noise free.

Brock et al [28] proposed a new architecture named Introspective Adversarial Network (IAN), which is a combination of GANs and Variational Autoencoders (VAE). The architecture increases the photorealistic sample quality of the image and was performed on CelebA, SVHN and CIFAR-100 datasets. For the generator network, the decoder network is used and discriminator is combined with encoder network of the autoencoder. In both the networks Adam optimizer and batch normalization is done. To improve the quality of the image IAN used orthogonal regularization and multiscale dilated convolution block.

Dilipkumar [29] has refined the image of the synthetic handwritten data which has been generated using SimGANs. In this experiment, the author has used two datasets one is Boing aircraft and another one is IAM Handwriting database using three variants of simGANs which include vanilla SimGANs, SimGANs with abstract loss and

SimGANs with Wasserstein loss. The images generated are then further passed to the CNN model for measuring the accuracy of the synthetic data. The generated has two convolution layers, 10 ResNets blocks with filter size and a fully-connected output layer having tanh as the activation function. Discriminator network has three hidden convolution layers and softmax layer. CNN architecture using for the word classification uses 7 convolutional layers, max-pool layer, 2 fully connected, and softmax layer. GANs training used Adam optimizer whereas CNN used AdaGrad optimizer. Abstract SimGANs for both the dataset produced the better results.

Gregor et al [30] propose a new mechanism named Deep Recurrent Attention (DRAW) for the generation of an image. The architecture is the combination of encoder and decoder which uses LSTM recurrent network. The experiment was performed on the MNIST, street View House Number (SVHN), CIFAR-100 and MNIST. The architecture produced the results which are indistinguishable from the original data.

Kataoka et al [31] proposed the combination of attention mechanism and GANs to produce the more relevant result than the standard GANs and DRAW. The experiment is performed on MNIST dataset. The generator and discriminator networks are made using Recurrent Neural Networks (RNN) having 256 hidden layer units. Therefore images produced by the attention mechanisms with GANs produced more clear results than standard GANs and DRAW.

Gosh et al [32] proposed the modified structure of DCGANs and apply reinforcement learning technique to learn fast on MNIST dataset. The authors aim to build a system that can easily separate the alphabets, which is similar to humans. The generator network performs the upsampling, used convolution layer with stride. The discriminator network performs the down-sampling, using Leaky ReLU in between the hidden convolution layers. To normalize the data batch-normalization is done. Further reinforcement technique is applied so that generator is able to learn the handwriting with greater accuracy.

CHAPTER 3

PROBLEM STATEMENT

3.1 Problem Formulation

Computers have influenced the life of human beings to a great extent in today's world to make it flexible. Handwritten recognition plays a vital role in various applications such as signatures identification, in banking applications and so on which is less error prone as compared to taking input via mouse and keyboard. The major problem faced in the area of deep learning is the data scarcity. There are various techniques for growing and generating datasets, but have flaws in all. Because of scarcity of data, it is difficult to train the neural network and also difficult to obtain high recognition accuracy results and thereby making this an interesting research topic to be considered.

Generative models are one of the most promising approaches toward this goal. To train a generative model a dataset is collected in some domain and then training of model is performed to generate data like it. The main motive of this dissertation is to generate the handwritten character of Gurmukhi and Devanagari, for which GANs are, used which help us to generate images from the vector representation.

There is a lot of research in progress on improving GANs and one of a work that really took a big step towards improving the quality of samples is adding convolution layers to introduce a new architecture known as Deep Convolutional Generative adversarial Networks (DCGANs). DCGANs consist of the generator which is a deconvolutional net and the discriminator which is a convolutional net. This popular architecture is applied to the Devanagari and Gurmukhi dataset in this dissertation. The major problem with the DCGANs is that they are notoriously difficult to train. Without the right hyper-parameters, network architecture, and training procedure, there is a high chance that either the generator or discriminator will overpower the other.

3.2 Research Gaps

- In the previous papers authors had mentioned the data scarcity as the major drawback for improving the recognition accuracy.
- Data will increase so single CPU cannot handle it. Therefore the GPU architecture is required for training models to perform recognition tasks.

3.3 Research Objectives

- To generate training data for Devanagari and Gurmukhi scripts which deep learn original image and then generate the new image using Generative Adversarial Networks.
- To use high-level neural network open source library Keras with tensorflow as a backend for the implementation of GANs this provides many inbuilt functions for the computation of complex calculations.
- As more and more deeper convolutional layers are used to build the convolutional neural networks, the complexity of computing mathematical calculations also gets reduced rapidly and more deeper features such as lines, curves, edges, headline are able to find out efficiently.

CHAPTER 4

GURMUKHI AND DEVANAGARI SCRIPT OVERVIEW

4.1 Gurmukhi Script

4.1.1 Overview

Gurmukhi is mostly widely spoken language over millions of native speakers. It comes at 14th position in most widely spoken languages in the world. Gurmukhi word is composed of two parts: “Panj” and “ab” where “Panj” indicates five and “ab” refers to five major tributaries of the Indus river. Gurmukhi language originates from the Sanskrit language. Gurmukhi is oftenly written by the people of East Punjab in the Gurmukhi script. Gurmukhi script is written from left-to-right direction and in top down approach. Gurmukhi is used by the second Sikh Guru, Guru Angad Dev Ji. Gurmukhi script has 35 consonants, 6 special characters and 10 vowel identifiers.

Gurmukhi script can be divided into three horizontal zones as shown in Fig. 4.1 namely upper zone, middle zone and lower zone. The region above the headline, where some of the vowels and subparts of some other vowels resides represents upper zone, while the area below the headline where the consonants and some sub-parts of vowels are present represents middle zone. The middle zone is the busiest zone. The area below middle zone where some vowels and certain half characters lie in the foot of consonants represents lower zone.

.

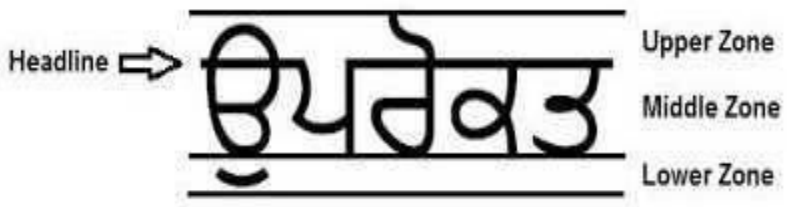


Figure 4.1: Different zones and headline of Gurmukhi script [38]

4.1.2 Gurmukhi consonants identifiers

In our proposed system, 35 consonants of Gurmukhi script specified in Fig. 4.2 have used. At the end, there is the total of 35 classes in Gurmukhi script. Each class contains 200 images in which variation occurs due to difference in the writing style of different users and also in the user's own writing style due to mood, speed of writing at different instant of time.

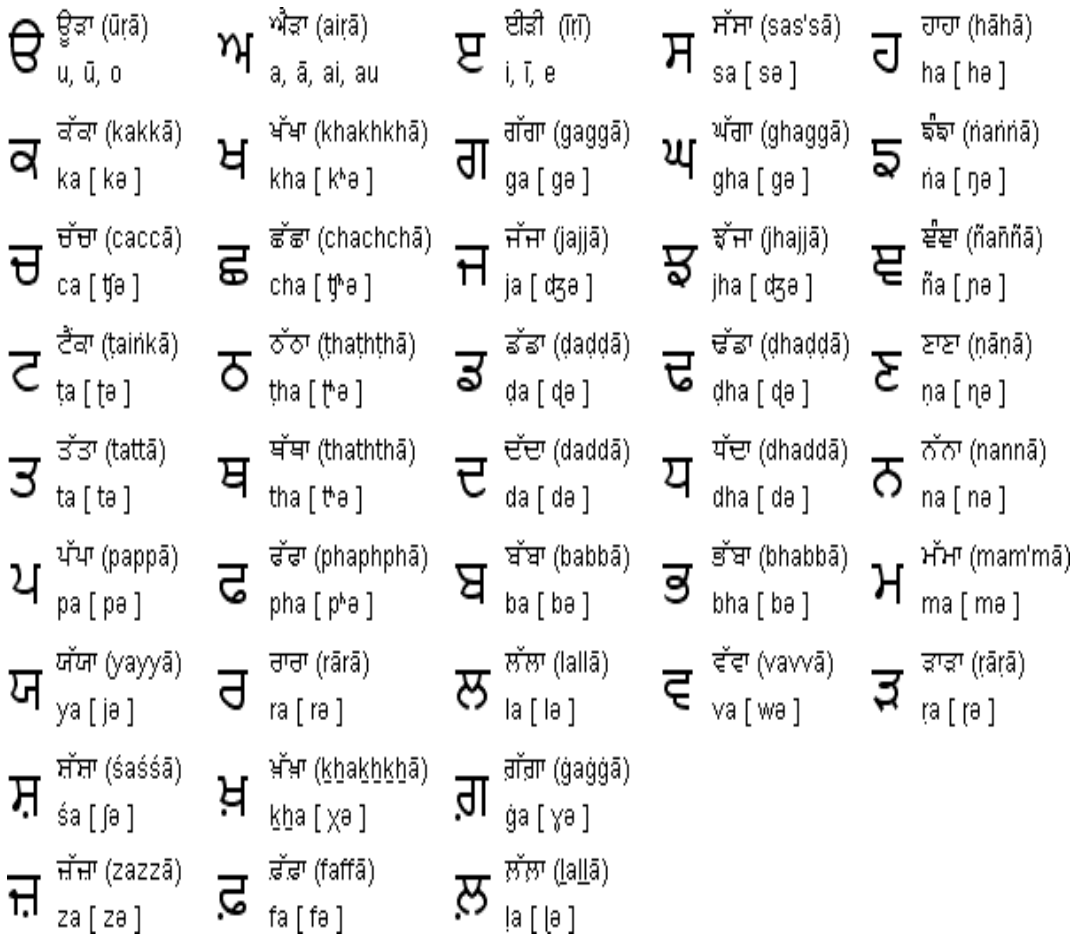


Figure 4.2: Consonants used in Gurmukhi script [39]

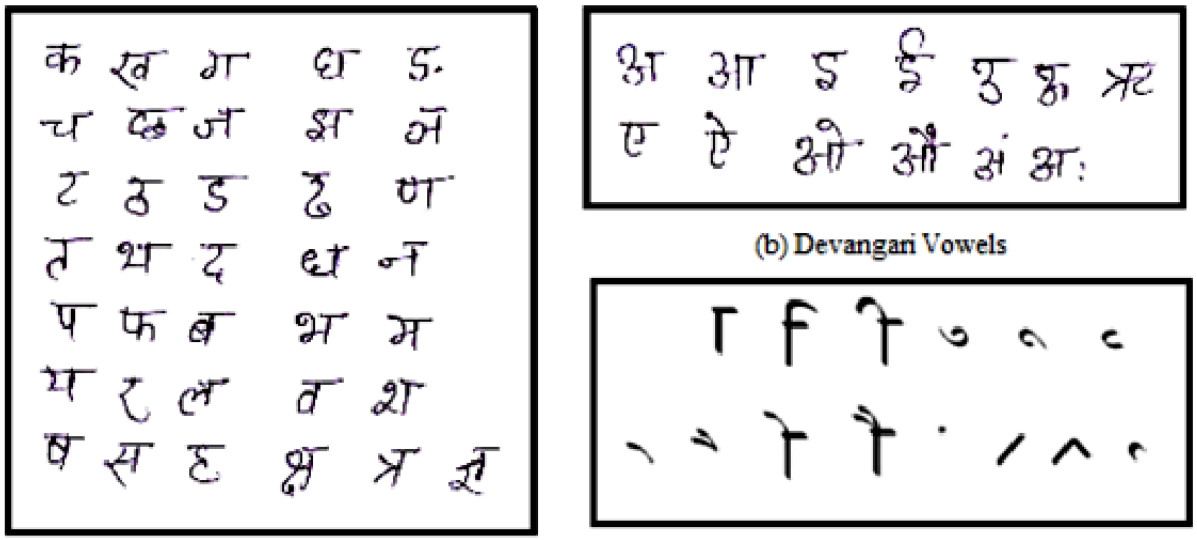
4.2 Devanagari Script

4.2.1 Overview

Devanagari is the part of the Brahmin family which is used for writing scripts of Tibet, South-East Asia, India, and Nepal. Hindi is used directly or indirectly by more than 500 million people in the Indian subcontinent and other parts of the globe.

4.2.2 Devanagari consonants and digits identifiers

Our model has been tested on a DHCD (Devanagari Character Dataset) which is publicly available dataset [33]. The dataset consists 36 consonants and 10 numerals, therefore 46 different classes of characters of Devanagari script as shown in Fig. 4.3. The images of 92000 characters of DHCD were created by collecting the samples from different age group people to provide the highest variety of written characters. Also in DHCD many of handwritten documents are manually scanned and cropped.



(a) Devanagari Consonants

(b) Devangari Vowels

(c) Modifiers

Figure 4.3: Consonants, vowels and modifiers used in Devanagari script [40]

CHAPTER 5

IMPLEMENTATION

In the present study, GANs have trained on the preprocessed handwritten character dataset and after that synthetic image is generated by deep learning the input image. Development stages of image generation system using GANs are shown in Fig. 5.1. Section 5.1 illustrates the data collection. Section 5.2 demonstrates the Preprocessing of input image. Section 5.3 demonstrates the building of generative model by training CNNs. Section 5.4 discusses the final output generation. In this proposed system, Ubuntu environment is used for installation of Keras library for the implementations of GANs.

5.1 Data Collection

In machine learning and Deep learning algorithms for classifying and recognizing, there is a need for a large amount of data to perform the task effectively. In this dissertation, an experiment has been performed on two datasets one consists of Devanagari script, and another is Gurmukhi script. In the Gurmukhi script, the 7000 samples of Gurmukhi character have been collected from one hundred different writers. The request was made to the writers to write each Gurmukhi character. There are total 35 classes in Gurmukhi dataset where each class has 200 images. The Gurmukhi characters are digitized into electric form from the paper-based handwritten character as shown in Fig. 5.1. A proper procedure is applied for the electronic conversion of the image, and then the image is scanned and saved in the TIFF format. The digital image is produced by the digitization procedure which is further fed for the preprocessing. The model has been tested on a DHCD (Devanagari Character Dataset) which is publicly available dataset. The dataset consist 36 consonants and 10 numerals, therefore 46 different classes of characters of Devanagari script. The images of 92000 characters of DHCD were created by collecting the samples from different age group people to provide the highest variety of written characters. Also in DHCD many of handwritten documents are manually scanned and cropped.

Figure 5.1: Gurmukhi character (ਕ (ka)) digitized image

5.2 Preprocessing

In image generation system, preprocessing is the first and very important stage. Preprocessing of input data is done to make input image appropriate to fed into neural network. This includes converting binary image into gray scale image, changing 100×100 bits image into 28×28 bits image, converting image into CSV format and normalization. In this phase by using Nearest Neighborhood Interpolation Algorithm (NNI) the gray scale level of Gurmukhi character is normalized into 100×100 window size from which a thinned bitmap image is produced using the parallel thinning algorithm which was proposed by Zhang and Suess [34] as shown in Fig. 5.2. As the 100×100 size image takes a lot of computation power and time to train therefore all the Gurmukhi characters of the image are reformed into 28×28 image size as shown in Fig. 5.2. Then both the Devanagari and Gurmukhi characters are converted into CSV file because neural network is easy to train on CSV format. In which one column consists of a class id and other columns contains pixel values in row-major format. Then both the images pixels value are normalized to bring down all the pixel values of the character on a scale from 0 to 1 as the non-normalized data cause instability in neural networks.

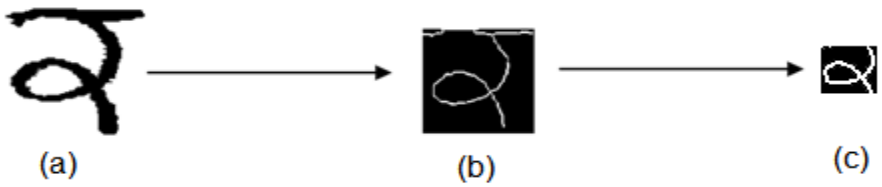


Figure 5.2: (a) Gurmukhi character (ਕ (ka)) Digitized image (b) Gurmukhi character (ਕ (ka)) thinned image (c) Gurmukhi character (ਕ (ka)) reformed image.

5.3 GANs Network Building

After preprocessing of the dataset, GANs are trained to build a network. The network deep learns the input image to generate a character. After that loss is calculated by training the GANs with Adam optimizer and back-propagation to verify implementation gradient checking and finally image is generated. The stages of dataset generation are shown in Fig. 5.3.

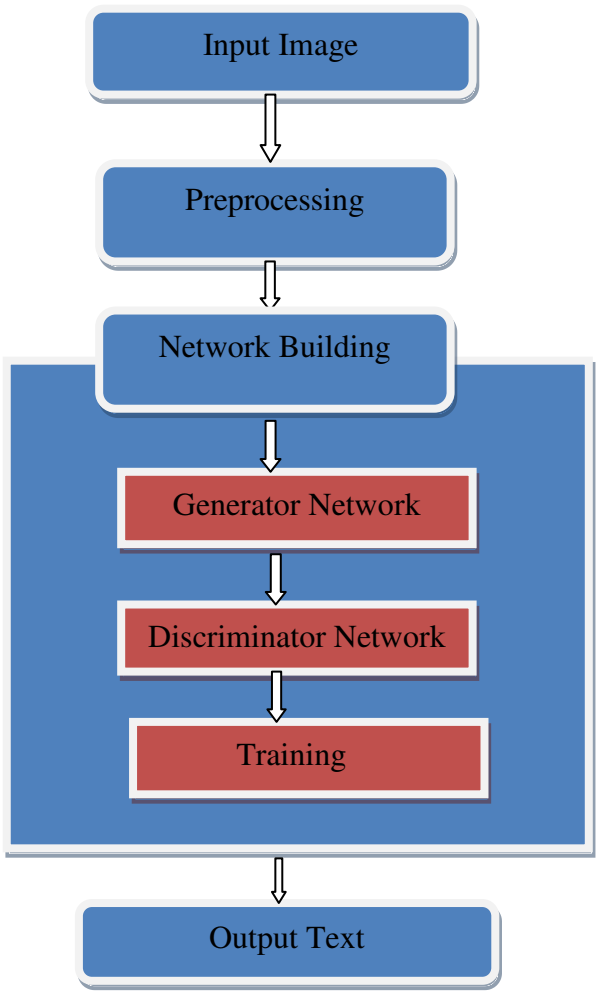


Figure 5.3: Development stages of Image Generation System

The general structure of GANs model is shown in Fig. 5.4 using Gurmukhi and Devanagari character dataset. The noise vector z is the random vector generated which is generated from a 100-dimensional noise which is uniformly distributed between -1.0 to 1.0 and mapping

them between [1, 28, 28] and [1, 32, 32] to match Gurmukhi and Devanagari data shape respectively. The activation function used by the neural network in between the layer is Leaky ReLU for the non-linearity and the activation function used in the last layer of the generator is sigmoid and the discriminator is softmax.

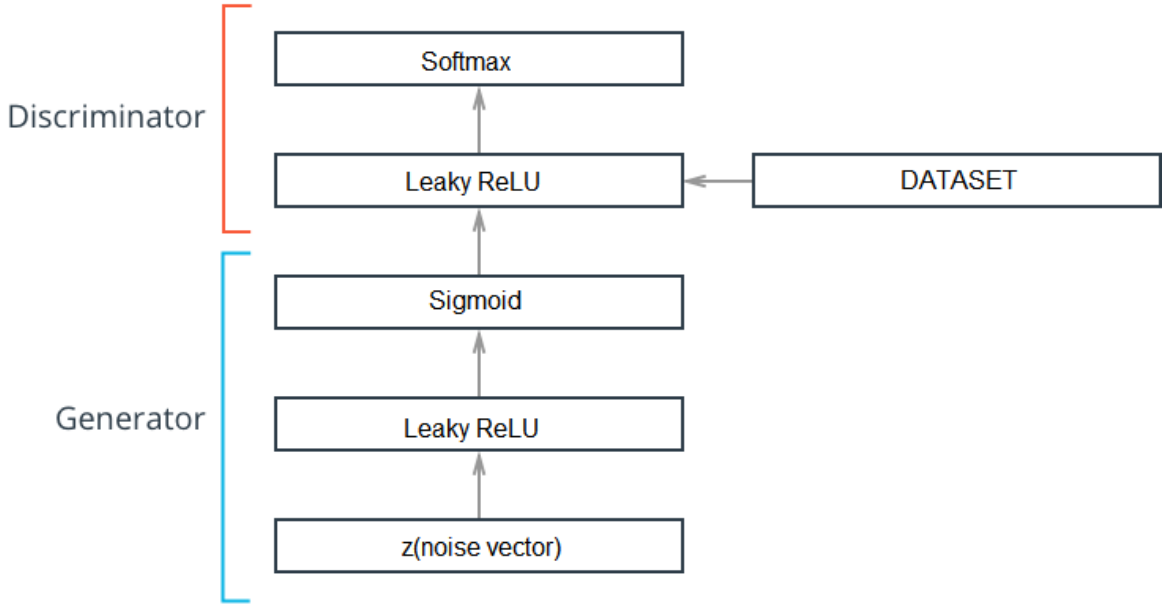


Figure 5.4: GANs Network

5.3.1 Generator Network

Table 5.1 and Table 5.2 show the structure of the generator network of Devanagari and Gurmukhi dataset. The purpose of the generator network is to learn how to create an image of the Gurmukhi and Devanagari handwritten character. Therefore the output of this network is a new image. Therefore it returns the 1024 values vector for Devanagari character which corresponds to flattened 32×32 size image and the 784 values vector for Gurmukhi character which corresponds to flattened 28×28 size image.

Table 5.1: Generator Network of Devanagari handwritten character image

INPUT:100-dimenctional uniform noise from [0,1]
Regular Fully-Connected Layer
Batch Normalization
Activation Layer(Leaky ReLU)

Convolutional Layer with 32 filters where each filter is 3×3
Batch Normalization
Activation Layer(Leaky ReLU)
Convolutional Layer with 64 filters where each filter is 3×3
Batch Normalization
Activation Layer(Leaky ReLU)
Deconvolutional with 1 filter of 1×1 having sigmoid activation

Therefore generator network has three hidden layers followed by Leaky ReLU activation function for the non-linearity and batch normalization is performed after each layer to normalize the weights into one scale to avoid the network to cause instability. On the output layer, is sigmoid activation function to help saturate pixels into 0 or 1 state.

Table 5.2: Generator Network of Gurmukhi handwritten character image

INPUT:100-dimenctional uniform noise from [0,1]
Regular Fully-Connected Layer
Batch Normalization
Activation Layer(Leaky ReLU)
Convolutional Layer with with 28 filters where each filter is 3×3
Batch Normalization
Activation Layer(Leaky ReLU)
Convolutional Layer with with 56 filters where each filter is 3×3
Batch Normalization
Activation Layer(Leaky ReLU)
Deconvolutional with 1 filter of 1×1 having sigmoid activation

5.3.2 Discriminator Network

Table 5.3 and 5.4 shows the discriminator structure of Gurmukhi and Devanagari handwritten character image. The input of this network is a flattened image and returns the probability of the generated image. The input size is 1024 vector for Devanagari character

and 784 for Gurmukhi character. There are three hidden layers and each layer is followed by Leaky ReLU, dropout to avoid overfitting and a two element softmax output layer.

Table 5.3 Discriminator Network of Devanagari handwritten character image

INPUT: 32 32 Gray scale image
Convolutional Layer with 256 filters where each filter is 5×5
Activation Layer(Leaky ReLU)
Dropout Layer
Convolutional Layer with 512 filters where each filter is 5×5
Activation Layer(Leaky ReLU)
Dropout Layer
Fully-Connected Layer with 1024 neurons
Activation Layer(Leaky ReLU)
Dropout Layer
Fully-Connected Layer with 2 neurons with softmax activation

5.3.3 Loss Calculation

Loss is calculated to know how well our model behaves after each iteration or optimization. Minimizing the loss function with respect to parameters of model using different optimization techniques like back-propagation is an objective in learning model. The loss function used in the network is the binary cross entropy because it resembles the log-loss for Generator and Discriminator function as shown in equation 1.12 and 1.13 in the introduction. Average of the loss is calculated for each mini-batch.

Table 5.4: Discriminator Network of Gurmukhi handwritten character image

INPUT: 28 28 Grayscale image
Convolutional Layer with 256 filters of 5×5
Activation Layer(Leaky ReLU)
Dropout Layer
Convolutional Layer with 512 filters of 5×5

Activation Layer(Leaky ReLU)
Dropout Layer
Fully-Connected Layer with 256 neurons
Activation Layer(Leaky ReLU)
Dropout Layer
Fully-Connected Layer with 2 neurons with softmax activation

5.3.4 Optimizer

For the optimization of both the network, Adam optimization algorithm [35] is used with the learning rate of 0.002. The proposed learning rate was obtained after testing with several values. The Adam optimization is applied because it performs the best among stochastic gradient descent optimizer and RMSProp optimizer.

5.3.5 Running Training Session

The final is the adversarial model in which generator and discriminator are stacked together. In the adversarial half of the network, weights are frozen during back-propagation of the joint mode. The training routine for the architecture looks like this, images are generated using a generator (G), and weights are updated in discriminator (D) and generator (G).

5.4 Output Generation

After training the image is generated which is obtain by converting vector by extracting the features from the real images and updating the weights. The model is trained for different epochs to get to know when the model will produce the desired image.

CHAPTER 6

RESULTS AND DISCUSSION

As discussed in Chapter 5, trained GANs are trained with Keras to develop the generative model. The experiments have performed by training GANs on Indic language dataset which includes Devanagari and Gurmukhi script. In Devanagari script there are 92000 grayscale format handwritten characters and in Gurmukhi script there are 7000 grayscale format handwritten characters. Each pixel value is in the range between 0 and 255. In experiment, pixel value is normalized to [0, 1]. The batch size used in our experiment is 128 and Adam for optimization with learning rate of 0.0002.

6.1 Experiment 1

In this experiment, the Devanagari dataset has considered testing the performance of the proposed methodology.

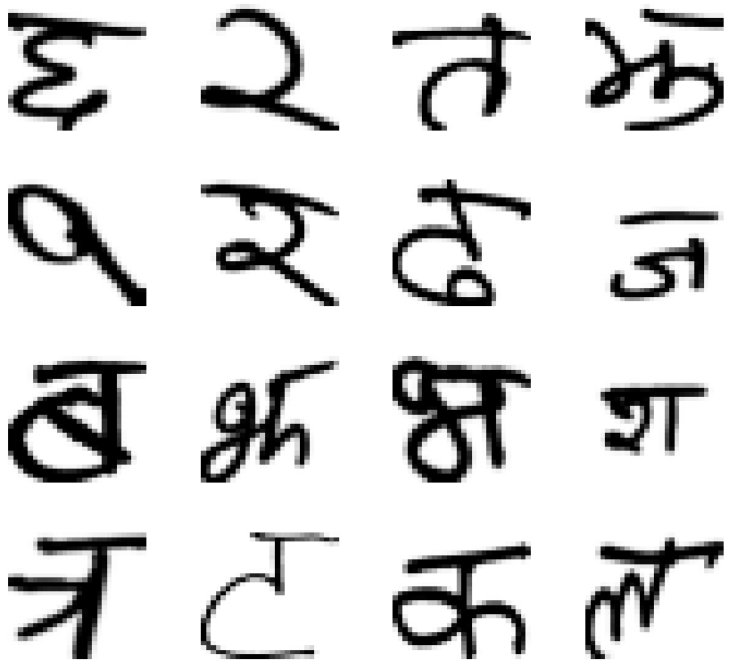


Figure 6.1: Sample character from real Devanagari dataset

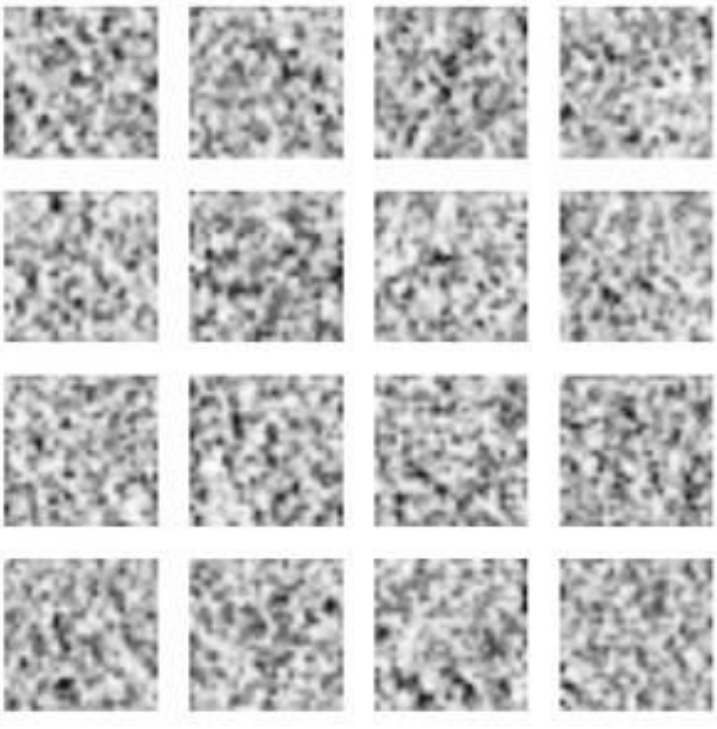


Figure 6.2: Noise images generated from an untrained generator

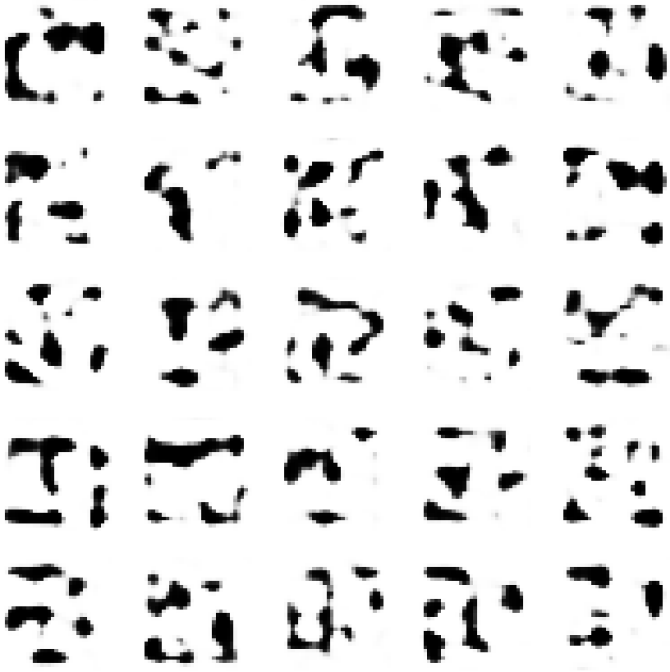


Figure 6.3: Sample of Devanagari images generated after 400 epochs.

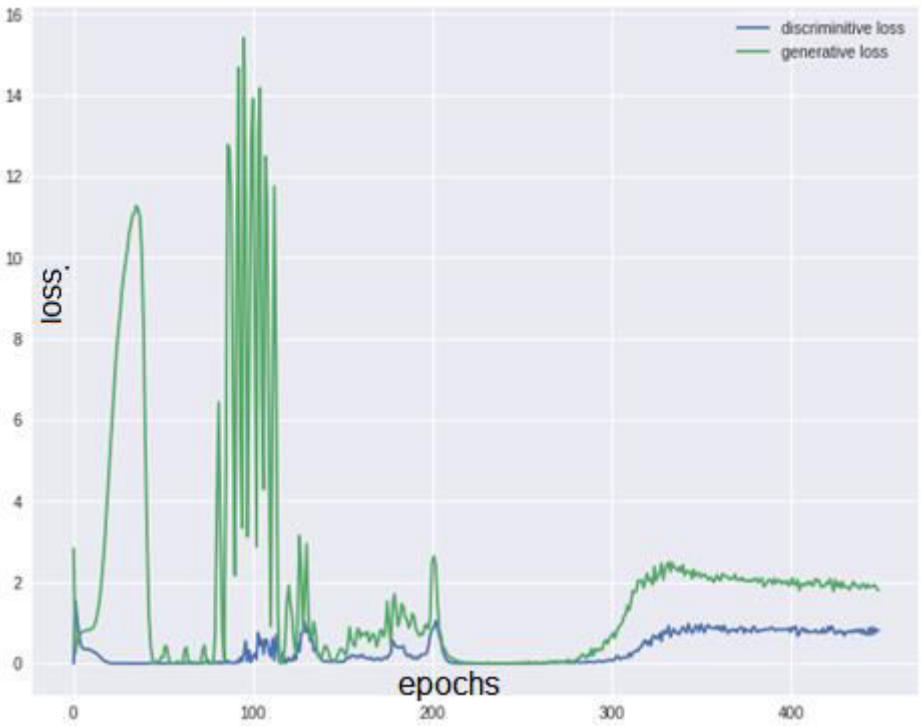


Figure 6.4: GANs training loss after 400 epochs

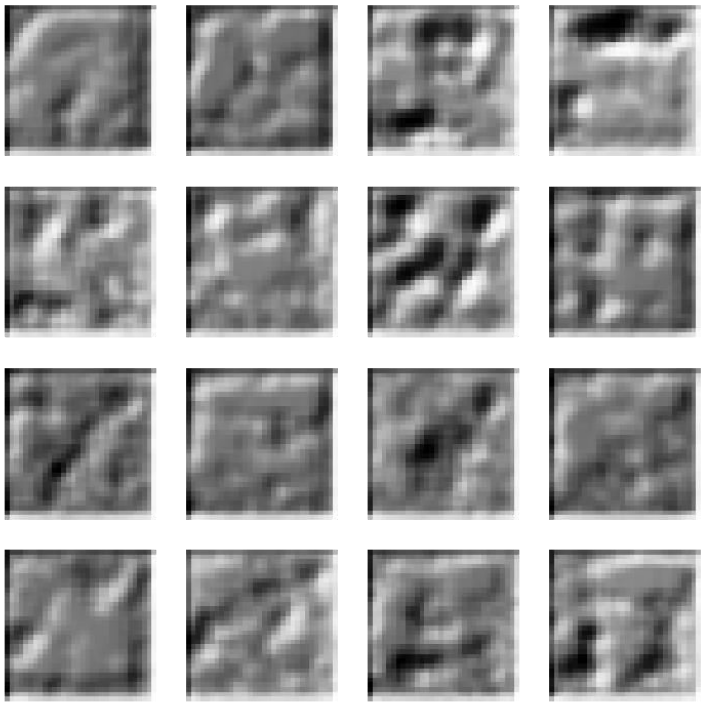


Figure 6.5: Sample of Devanagari images generated after 6000 epochs

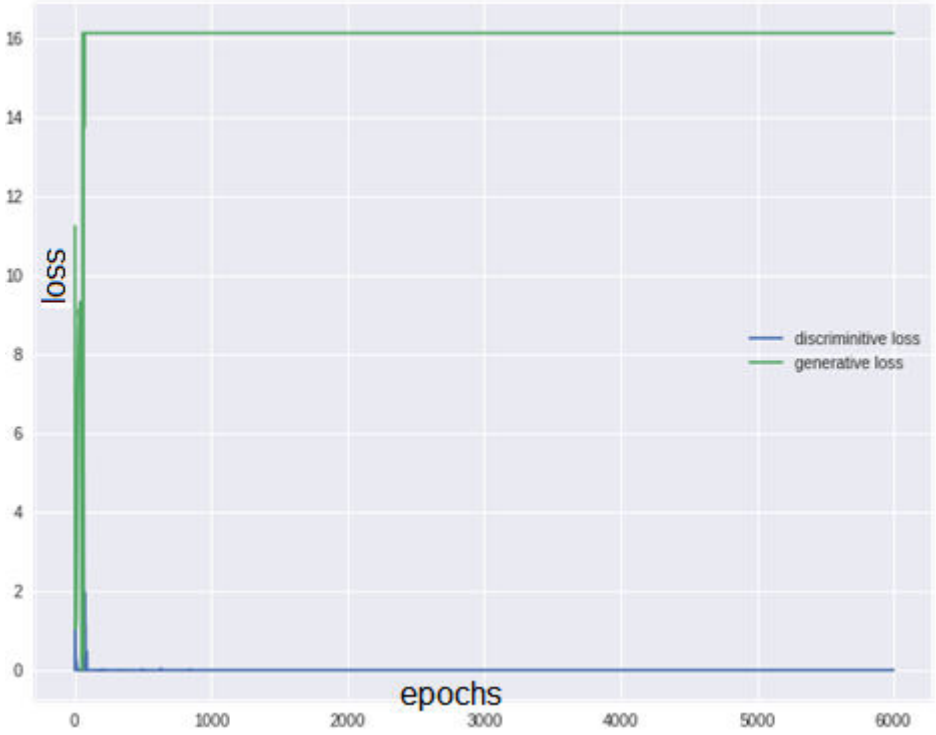


Figure 6.6: GANs training loss after 6000 epochs

Figure 6.7: Sample Devanagari of images generated after 8000 epochs

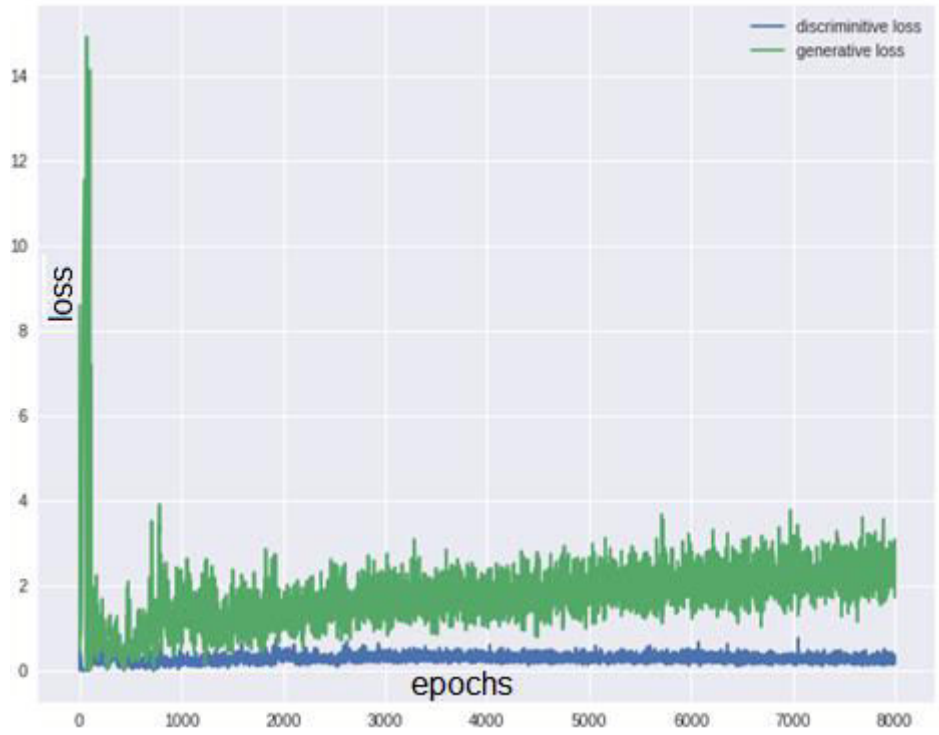


Figure 6.8: GANs training loss after 8000 epochs

Figure 6.9: Sample of images generated after 12000 epochs

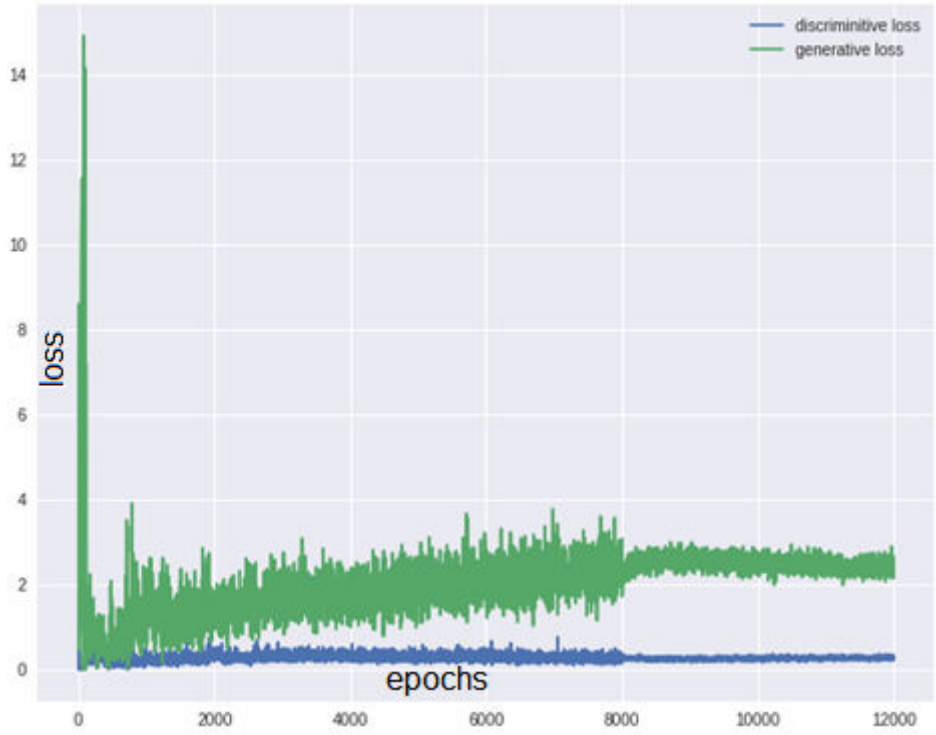


Figure 6.10: GANs training loss after 12000 epochs

Figure 6.11: Sample of images generated after 16000 epochs

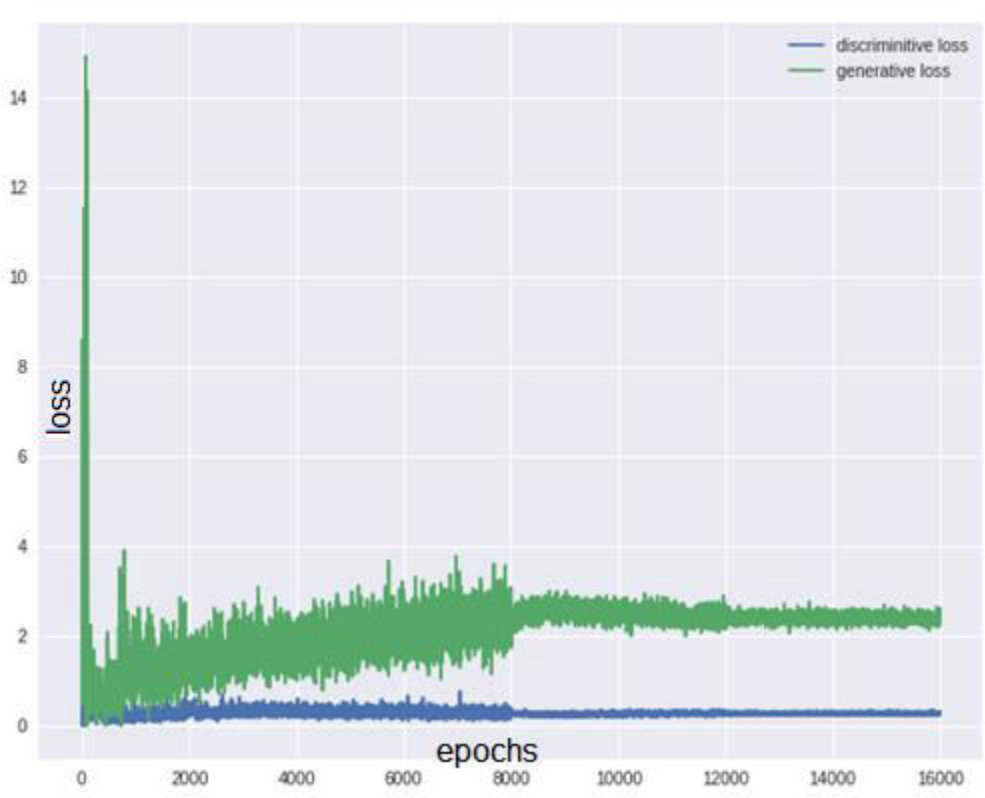
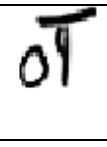
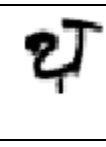


Figure 6.12: GANs training loss after 16000 epochs

Table 6.1: Devanagari character samples generated by GANs

				
(ka)	(ta)	(la)	(waw)	(thaa)
				
(na)	(yaa)	(tra)	(gya)	(tha)
				
(gha)	(pa)	(ksha)	(Ta)	(dha)

Table 6.2: Noisy and blurry Devanagari character generated by GANs

6.2 Experiment 2

In this experiment, the Gurmukhi dataset has considered testing the performance of the proposed methodology.

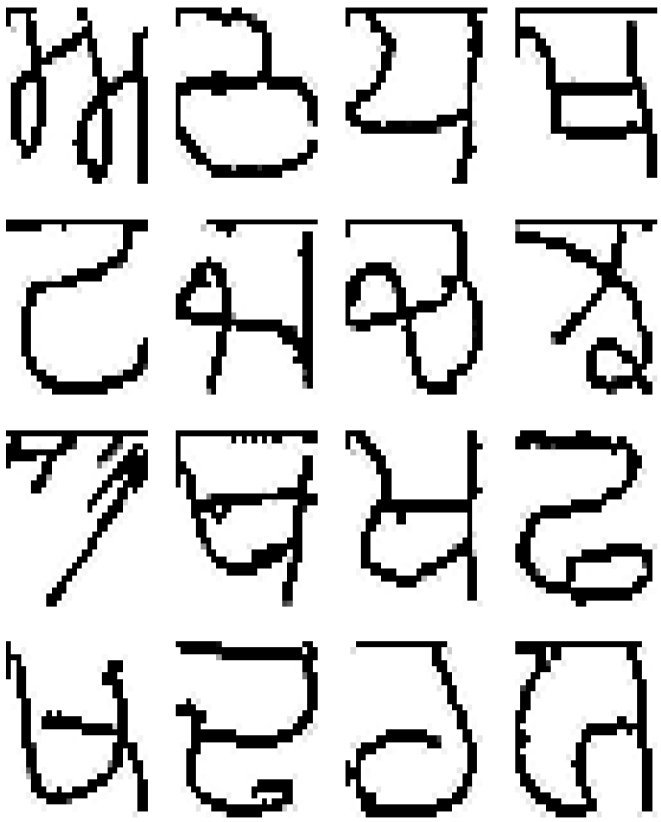


Figure 6.13: Sample character from real Gurmukhi dataset

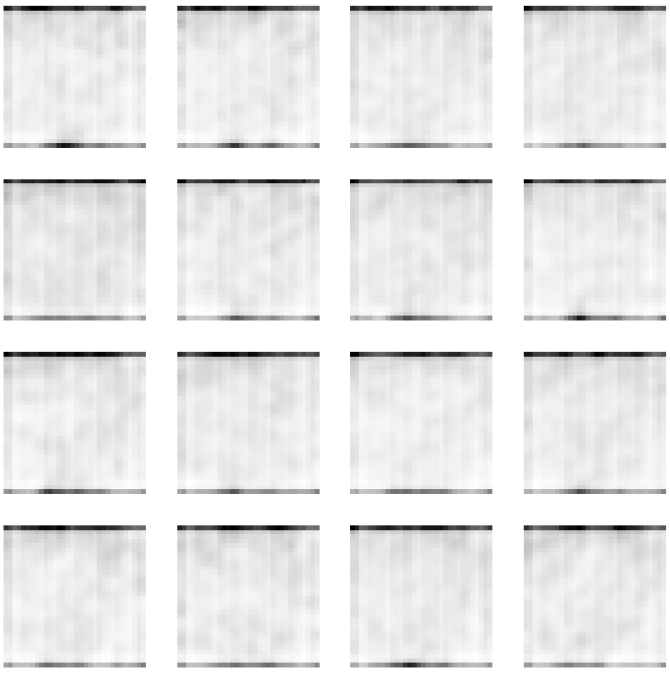


Figure 6.14: Sample of Gurmukhi images generated after 400 epochs

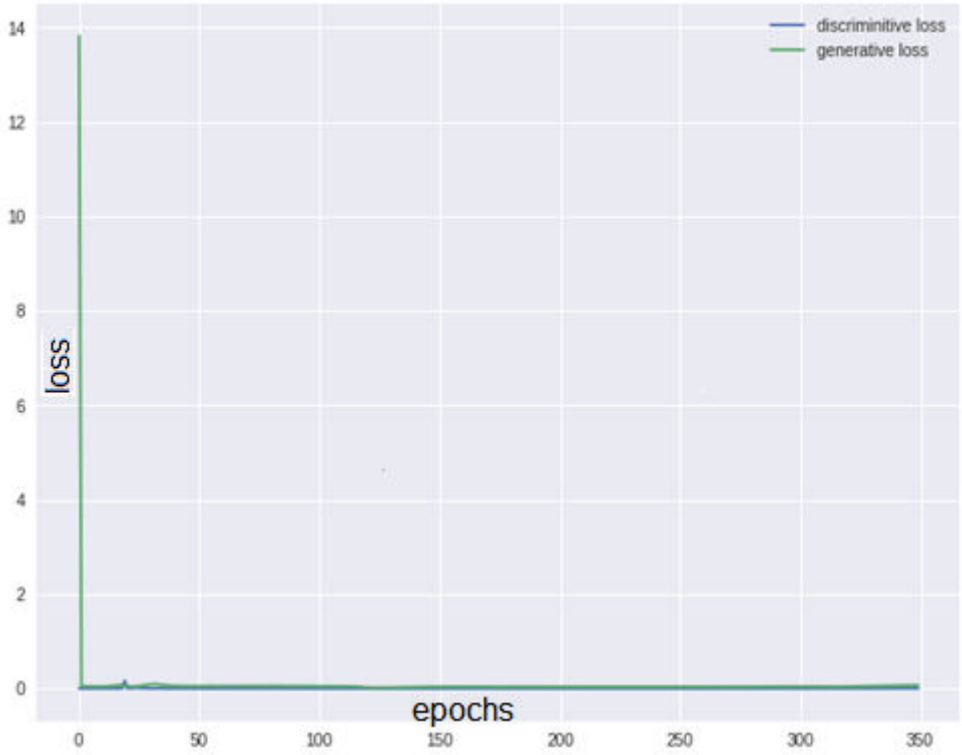


Figure 6.15: GANs training loss after 400 epochs

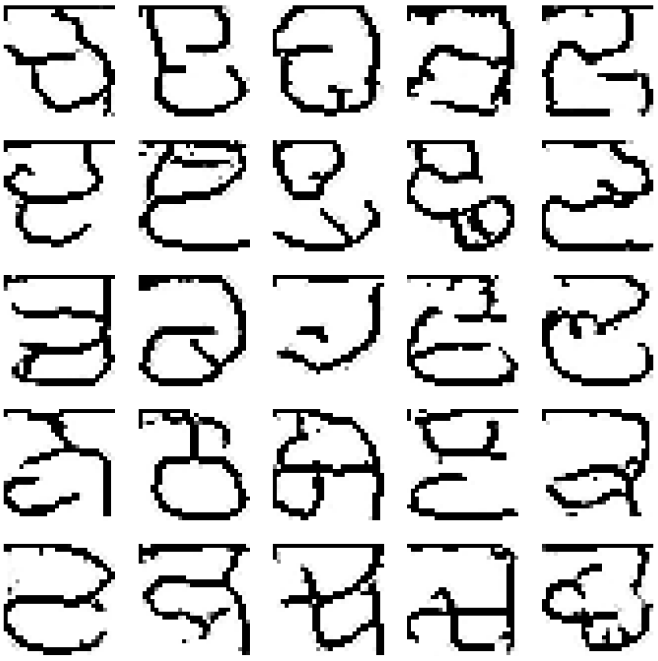


Figure 6.16: Sample of Gurmukhi images generated after 8000 epochs

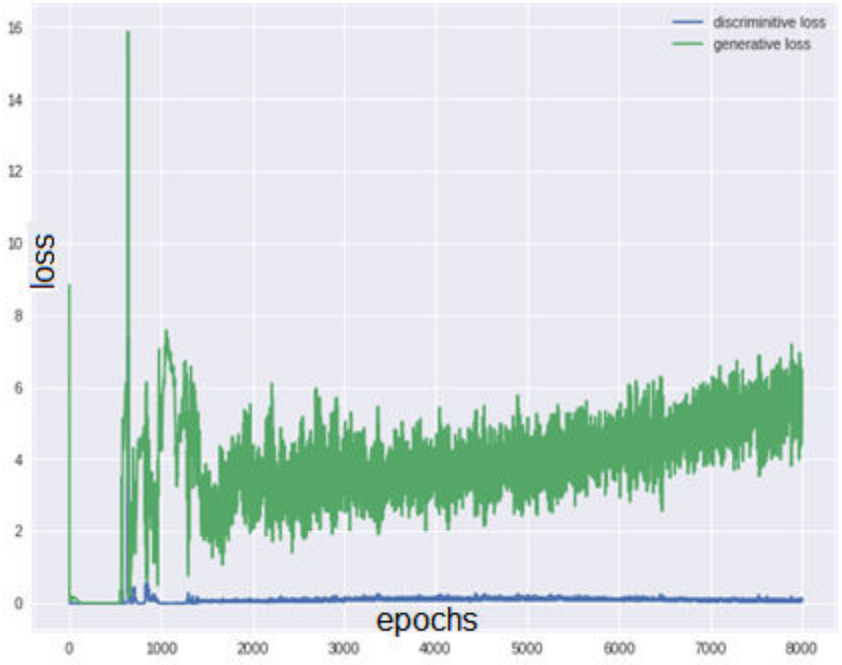


Figure 6.17: GANs training loss after 8000 epochs

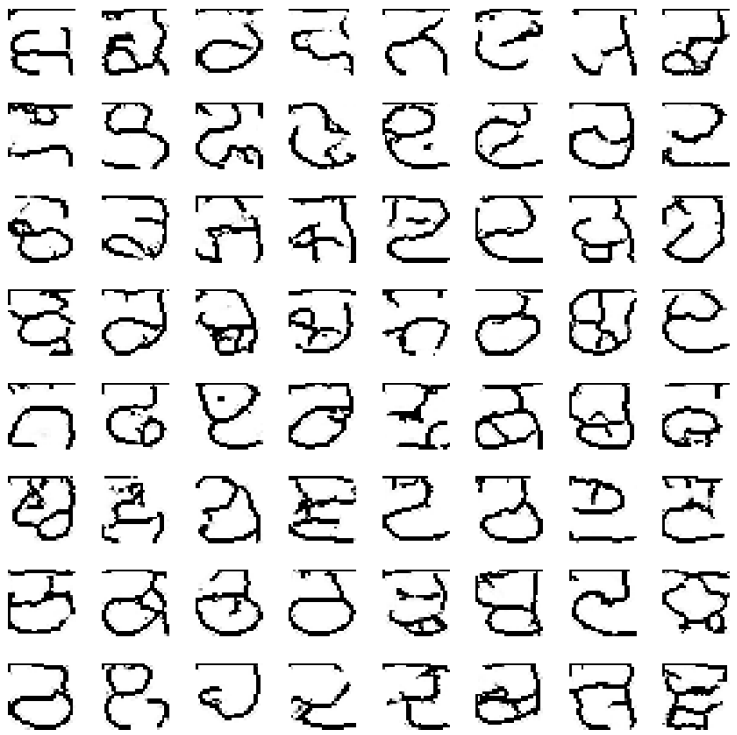


Figure 6.18: Sample of Gurmukhi images generated after 10000 epochs

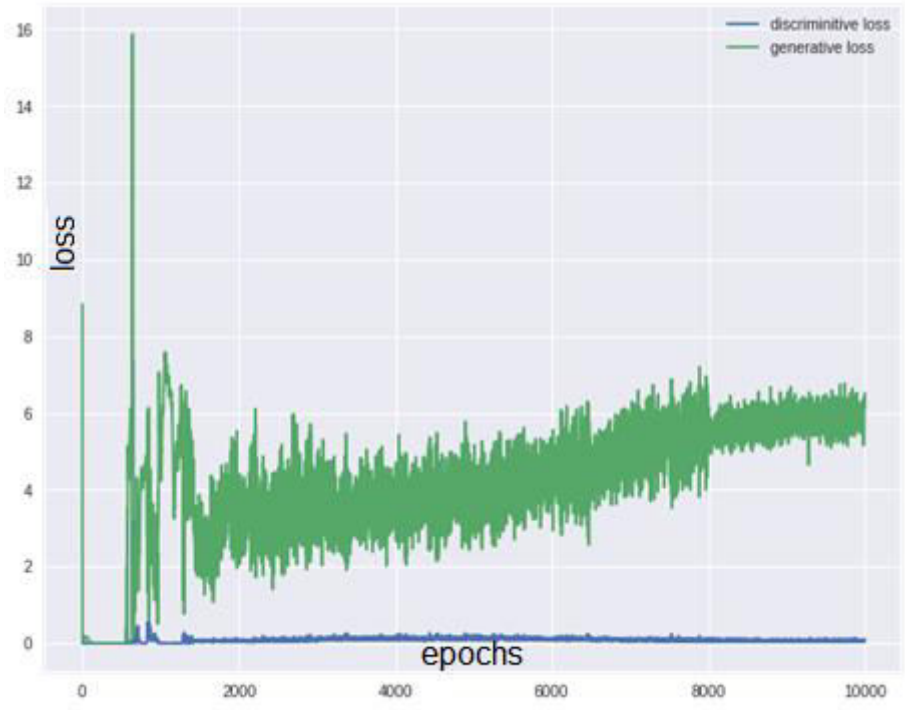


Figure 6.19: GANs training loss after 10000 epochs

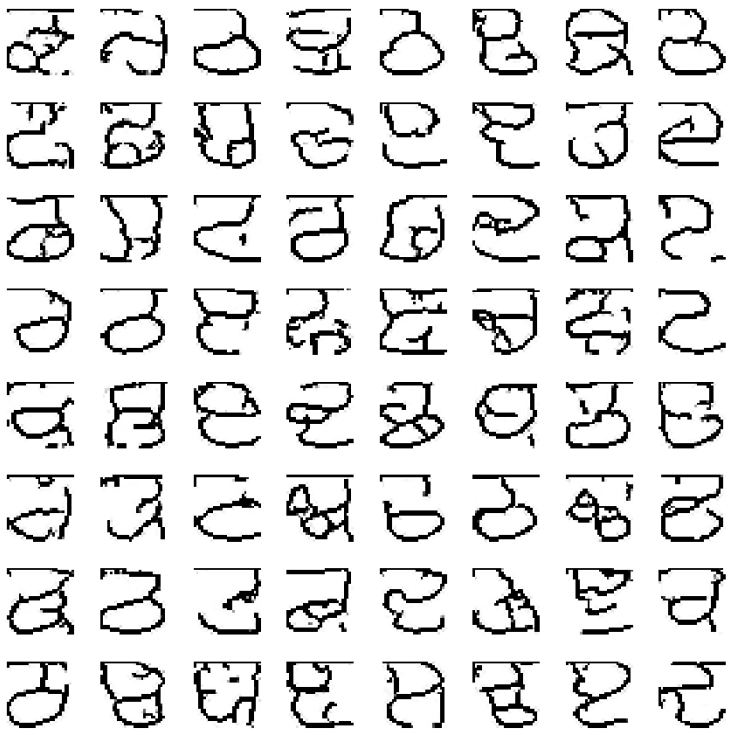


Figure 6.20: Sample of Gurmukhi images generated after 12000 epochs

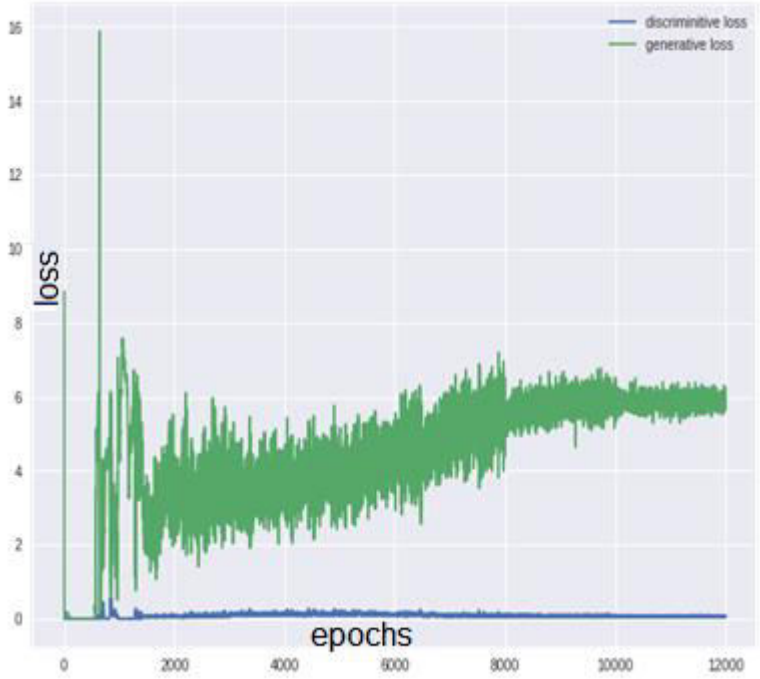


Figure 6.21: GANs training loss after 12000 epochs

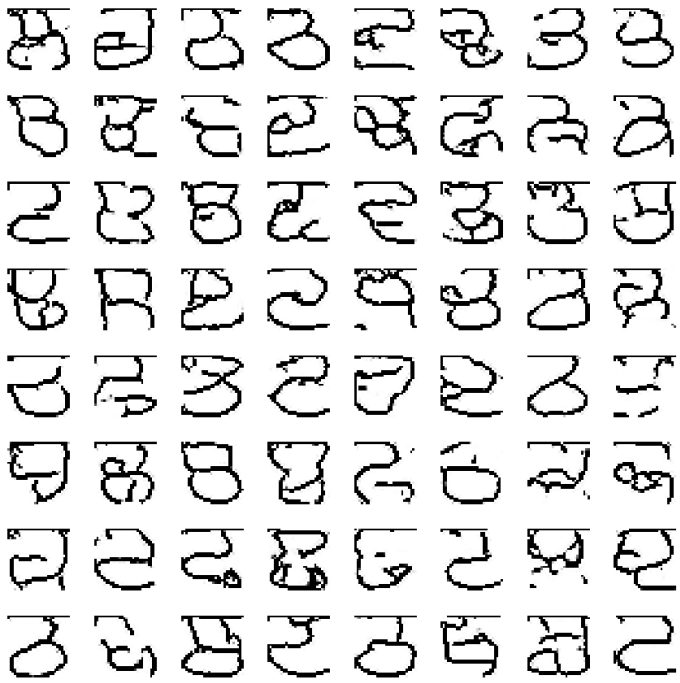


Figure 6.22: Sample of Gurmukhi images generated after 14000 epochs

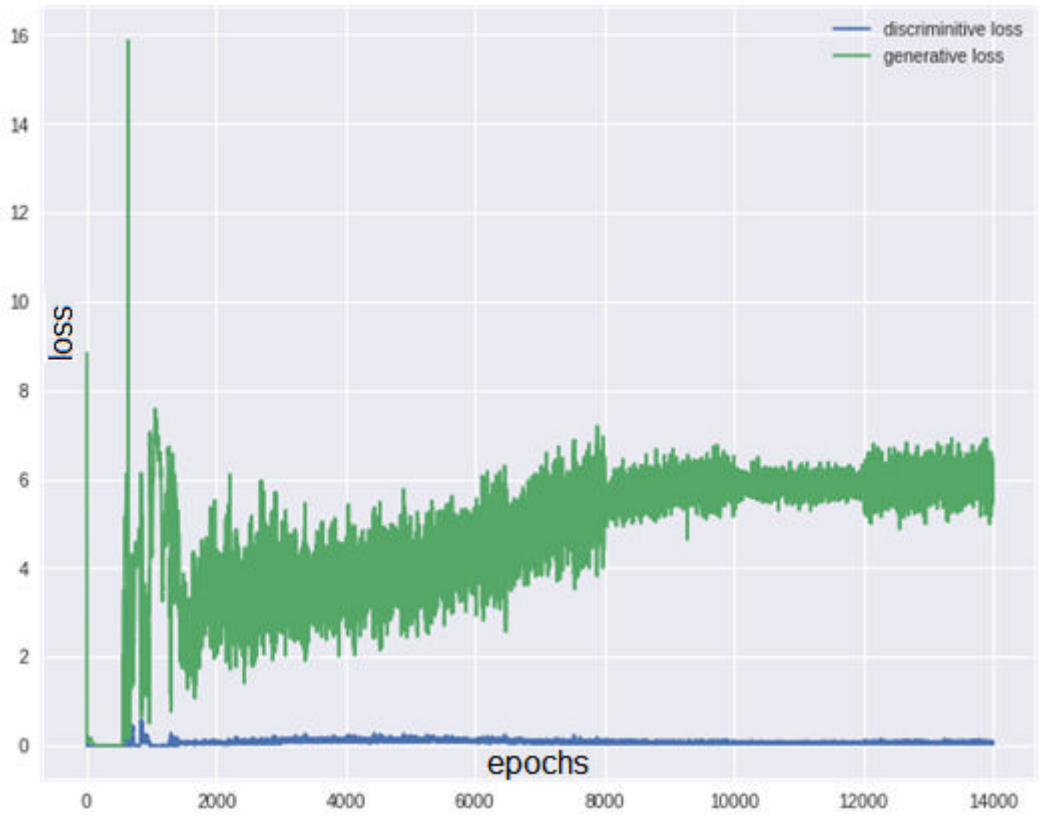


Figure 6.23: GANs training loss after 14000 epochs

Table 6.3: Samples of Gurmukhi character generated by GANs

				
(la)	(tha)	(cacca)	(bha)	(ra)
				
(iri)	(nanna)	(ra)	(ha)	(da)

Table 6.4: Noisy and blurry image of Gurmukhi character generated by GANs

In above figures plot of a generator and discriminator loss and some samples from the trained generated model and some examples from the original dataset as shown in Fig. 6.1 and Fig. 6.13 for comparison. The noise images before training the generator are shown in Fig. 6.2. As the GANs always had a problem with convergence and as a consequence, we did not know when to stop training. That's why the model is trained for different epochs to get to know when model will produce the desired output. Recognizable results begin to appear after 8000 cycles as shown in Fig. 6.7 and Fig. 6.16 will improve after that. The results after 400 epochs are still noisy as shown in Fig. 6.3 and Fig. 6.14. It can observe that by increasing the number of epochs the quality of generated loss decreases and refinement of image increases, which look relatively like the original Devanagari and Gurmukhi characters as shown in Fig. 6.11 and Fig. 6.22, the quality of generated loss decreases as shown in Fig. 6.9 and Fig. 6.21 and refinement of image increases, which look relatively like the original Devanagari and Gurmukhi characters. Additionally, some characters of the generated images are noisy and blurry as shown in Table 6.2 and Table 6.4. The GANs are difficult to train sometime

increase in the generator loss increase the refinement of image and sometime the quality of image decreases by increasing the generator loss.

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

This dissertation presents an implementation of handwritten character generation system using generative adversarial networks. To train deep learning models it is hard to find large labeled datasets, therefore, generation of synthetic data is gaining popularity in recent years. The DCGANs method and its variants performance is analyzed on an off-line Devanagari and Gurmukhi handwritten character recognition datasets. The DCGANs algorithm for image reconstruction gives satisfactory results. The characters generated look like the character in the original dataset. With the increase in number of epochs the images become more recognizable and clear till certain limit after which the loss increases. To generate handwritten characters for Gurmukhi and Devanagari scripts, convolutional neural networks have been used as feature extractor.

DCGANs are really effective in the generation of synthetic data and are notoriously difficult to train. Generally, the manual work is done to inspect the output. They are dependent on network architecture, hyper-parameters and training procedure. The wrong selection of any one of these may lead to the lack of synchronization between generator and discriminator which will affect the desired result.

Keras Library used for implementation provides many in-built methods to perform mathematical computations to improve the results and thereby helps in reduction of training time.

7.2 Future Scope

An interested researcher may implement a few more things to this work:

- Some of the generated characters are observed to be noisy and blurry. So in future some restoration techniques can be applied to remove such degradation.

- The generated images can be further preprocessed and classified using the convolutional neural network for handwritten character recognition.
- The more promising results can be generated by exploring other kinds of GANs.(e.g. Conditional GANs, Wasserstein GANs, InfoGAN etc)

PUBLICATION

1. Simerpreet Kaur, Karun Verma, "Handwritten Devanagari Character Generation Using Deep Convolutional Generative Adversarial Networks", communicated in 13th International Conference on Industrial and Information Systems (ICIIS 2018).

REFERENCES

[1] Y. LeCun, D. Touresky, G. Hinton, & T. Sejnowski, "A theoretical framework for back-propagation," in *Proceedings of the connectionist models summer school"* pp. 21-28, 1988

[2] S. Ioffe, & C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167, 2015*

[3] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, & L. D. Jackel, " Handwritten digit recognition with a back-propagation network,". in *Advances in neural information processing systems*,pp. 396-404, 1990

[4] B. K. Verma, "Handwritten Hindi character recognition using multilayer perceptron and radial basis function neural networks," in *Neural Networks, 1995. Proceedings, IEEE International Conference on*. vol. 4, pp. 2111-2115, 1995

[5] I. K. Sethi, & B. Chatterjee, "Machine Recognition of Hand-printed Devnagri Numerals," *IETE Journal of Research*, vol. *22, no.*8, pp.532-535, 1976

[6] N. Sharma, U. Pal, F. Kimura, & S. Pal, "Recognition of off-line handwritten devnagari characters using quadratic classifier," in *Computer Vision, Graphics and Image Processing*, Springer, Berlin, Heidelberg, pp. 805-816, 2006

[7] P. S. Deshpande, L. G. Malik, & S. Arora, "Fine Classification & Recognition of Hand Written Devnagri Characters with Regular Expressions & Minimum Edit Distance Method," *Journal of Computers*, paper vol. *3,* no.5, pp.11-17, 2008

[8] S.Arora, D. Bhatcharjee, M. Nasipuri, & L. Malik, " A two stage classification approach for handwritten Devanagari characters," in Proceedings of the *International Conference on Computational Intelligence and Multimedia Applications ,* vol. 35, pp. 399-403, 2007

[9] M. Hanmandlu,V.R Murthy, & V.K.Madasu, "Fuzzy Model based recognition of handwritten Hindi Characters" in Proceedings of the *9th Biennial Conference of the Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications*, pp. 454-46, 2007

[10] U. Pal, N. Sharma, T. Wakabayashi, & F. Kimura, "Off-line handwritten character recognition of Devnagri script," in *Document Analysis and Recognition,* Ninth International Conference on IEEE, no.1, pp. 496-500, 2007

[11] U. Bhattacharya, & B. B. Chaudhuri, "Handwritten numeral databases of Indian scripts and multistage recognition of mixed numerals," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31 , no. 3, pp. 444-457, 2009

[12] S. Kumar, "Performance comparison of features on Devnagri hand printed dataset," *International Journal of Recent Trends* , vol. 1, no.2, pp.33-37, 2009

[13] S. Arora, D. Bhattacharjee, M. Nasipuri, D. K. Basu, M. Kundu, & L. Malik, "Study of different features on handwritten Devnagari character," In *Emerging Trends in Engineering and Technology , 2nd International Conference on* , pp. 929-933, 2009

[14] S. Arora, D. Bhattacharjee, M. Nasipuri, D.K. Basu, & M. Kundu, " Recognition of non-compound handwritten Devanagari characters using a combination of MLP and minimum edit distance," *arXiv preprint arXiv:1006.5908*, 2010

[15] M. Kumar, R. K. Sharma, & M. K. Jindal, "Segmentation of lines and words in handwritten Gurmukhi script documents," in *Proceedings of the First International Conference on Intelligent Interactive Technologies and Multimedia*, pp. 25-28, 2010

[16] K. Verma, & R. K. Sharma, " Performance analysis of zone based features for online handwritten Gurmukhi script recognition using support vector machine," in *Progress in systems engineering*, pp. 747-753, 2015

[17] K. Verma, & R. K. Sharma, "Comparison of HMM-and SVM-based stroke classifiers for Gurmukhi script," *Neural Computing and Applications*, vol. *28*, no. 1, pp. 51-63, 2017

[18] K. Verma, & R. K. Sharma, "Recognition of online handwritten Gurmukhi characters based on zone and stroke identification," *Sadhana: Academy Proceedings of Engineering Sciences, Springer*, vol. *42*, no. 5, pp. 701-712, 2017

[19] K. Verma, & R. K. Sharma, " An Efficient Writing-Zone Identification Technique for Online Handwritten Gurmukhi Character

Recognition," *Proceedings of the National Academy of Sciences, India Section A: Physical Sciences*, pp. 1-11, 2017

[20] K. He, X. Zhang, S. Ren, & J. Sun, " Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition* , pp. 770-778, 2016

[21] D. Kaur, & M. R. K. Gurm, "Machine Printed Gurumukhi Numerals Recognition using Convolutional Neural Networks," 2016

[22] M. Jangid, & S. Srivastava, "Handwritten Devnagri Character Recognition Using Layer-Wise Training of Deep Convolutional Neural Networks and Adaptive Gradient Methods," *Journal of Imaging*, vol.4, no. 2, pp.41, 2018

[23] G. E. Hinton, S. Osindero, & Y.W. Teh, " A fast learning algorithm for deep belief nets," *Neural computation*, vol. *18,* no. 7, pp. 1527-1554, 2006

[24] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D.Warde, S. Ozair, & Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems* , pp. 2672-2680, 2014

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

[26] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, & H. Lee, "Generative adversarial text to image synthesis," *arXiv preprint arXiv:1605.05396, 2016*.

[27] Y. Jin, J. Zhang, M. Li, Y. Tian, H. Zhu, & Z. Fang," Towards the Automatic," Anime Characters Creation with Generative Adversarial Networks, *arXiv preprint arXiv:1708.05509*, 2017

[28] A. Brock, T. Lim, J. M. Ritchie, & N. Weston, "Neural photo editing with introspective adversarial networks," *arXiv preprint arXiv:1609.07093*, 2016

[29] D. Dilipkumar, "Generative Adversarial Image Refinement for Handwriting Recognition," 2017

[30] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, & D. Wierstra, "DRAW: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, 2015

[31] Y. Kataoka, T. Matsubara, & K. Uehara," Image generation using generative adversarial networks and attention mechanism, " in *Computer and Information Science (ICIS), 2016 IEEE/ACIS 15th International Conference,* pp. 1-6, 2016

[32] A. Ghosh, B. Bhattacharya & S.B.R Chowdhury, "Handwriting Profiling Using Generative Adversarial Networks," *in* Association for the Advancement of Artificial Intelligence*, p*p. 4927-4928, 2017

[33] S. Acharya, A. K. Pant, & P. K. Gyawali, "Deep learning based large scale handwritten Devanagari character recognition" in *Software, Knowledge, Information Management and Applications, 9th International Conference on IEEE,* pp. 1-6, 2015

[34] T. Y. Zhang & C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," Communications of the Association for Computing Machinery, vol. 27, no. 3, pp. 236–239, 1984

[35] D. P. Kingma, & J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014

[36] N. Azzouz, S. Bechikh, & L. Ben Said, "Steady state IBEA assisted by MLP neural networks for expensive multi-objective optimization problems," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 581-588, 2014

[37] "All institutions," https://brage.bibsys.no accessed on 25 June 2018

[38] "Online Punjabi Teaching," http://www.discoversikhism.com accessed on 2 June 2018

[39] H. Kaur, & S. Rani, "Handwritten Gurumukhi Character Recognition Using Convolution Neural Network," in *International Journal of Computational Intelligence Research*, vol. 13, no. 5, pp. 933-943, 2017

[40] A. A. Malanker, & M. M. Patel, "Handwritten Devanagari Script Recognition: A Survey"