# Research on Generative Sign Language using Neural Networks

by

Bushra Binte Selim
21141052
Maliha Iqbal
21141050
Asif Shahriar
16301040
Fauzia Faria
17141007
Rafid Mostafa
16101069

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering (16301040, 16101069) and
B.Sc. in Computer Science (17141007, 21141052, 21141050)

Brac University
Department of Computer Science and Engineering
The School of Data Sciences
June 2021

## Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.

2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.

3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.

4. We have acknowledged all main sources of help.

**Student's Full Name & Signature:**

[illegible]

Bushra Binte Selim
21141052

Maliha Iqbal

Maliha Iqbal
21141050

Asif

Asif Shahriar
16301040

FauziaFaria

Fauzia Faria
17141007

[illegible]

Rafid Mostafa
16101069

# Approval

The thesis titled "Research on Generative Sign Language using Neural Networks" submitted by

1. Bushra Binte Selim (21141052)
2. Maliha Iqbal (21141050)
3. Asif Shahriar (16301040)
4. Fauzia Faria (17141007)
5. Rafid Mostafa (16101069)

Of Spring, 2021 has been accepted as satisfactory in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering (16301040, 16101069) and B.Sc. in Computer Science (17141007, 21141052, 21141050) on June 06, 2021.

**Examining Committee:**

Supervisor:
(Member)

Moin Mostakim

Moin Mostakim
Lecturer
Department of Computer Science and Engineering
Brac University

Co- Supervisor:
(Member)

[illegible]
8.6.21

Arif Shakil
Lecturer
Department of Computer Science and Engineering
Brac University

Thesis Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

[illegible]

Sadia Hamid Kazi
Associate Professor and Chairperson
Department of Computer Science and Engineering
Brac University

## Ethics Statement

We, the members, hereby and sincerely declare that this thesis has been done based on the findings of our extensive research. All the materials, which have been used are properly noted and cited in this report. This research work, neither in full nor any part has never been submitted by any other person to another university or any institution for the award of any degree or any other purpose.

# Abstract

Sign gesture, which is one type of non-audible specialized strategy is the medium to correspond with individuals having auditory and talking incompetency. There are numerous computerized methods of creating gesture-based communication to provide aid among the hearing impaired. Particularly, for Bengali sign dialect, quite a few measures have been taken for generation of automated Bangla sign gestures. With an authentic dataset and approach, an apparent communication mode to assist this non-privileged community can be attained. Our method proposes a convolutional neural network (CNN) to derive a picture of the appropriate sign gesture of a particular Bangla alphabet. After our examinations and multiple experiments, we have come up with the simplest and most striking methodology to perform the mentioned task. Our model worked promptly and provided remarkable accuracy. Needless to mention that, communication through gestures aided by artificial means is another corner that needs to be explored more. Henceforth, our work can have an added value to this ongoing inspection.

**Keywords:** Sign Language Generation, Neural Network, Neural Networks for Sign Language, Generative Sign, CNN, Inceptionv3

# Acknowledgement

Firstly, all praise to the Great Almighty for whom our thesis has been completed without any major interruption in this pandemic situation.
Secondly, to our supervisor Moin Mostakim and our co-supervisor Arif Shakil for their kind support and advice in our work. They helped us whenever we needed help.
Thirdly, we want to take a moment to thank our supportive friends who have been there to give us mental support in the hard times.
Finally, to our parents; without their love and support it may not be possible. With their kind support and prayer, we are now on the verge of completing our graduation.

# Table of Contents

# List of Figures

# List of Tables

# Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

| | |
|---|---|
| *ANN* | Artificial Neural Network |
| *ASL* | American Sign Language |
| *BdSL* | Bangladeshi Sign Language |
| *CNN* | Convolutional Neural Network |
| *ConvNet* | Convolutional Neural Network |
| *GAN* | Generative Adversarial Networks |
| *KNN* | K Nearest Neighbor |
| *NMT* | Neural Machine Translation |
| *ReLU* | Rectified Linear Unit |
| *ResNet* | Residual Neural Network |
| *RMSprop* | Root mean square prop |
| *SGD* | Stochastic Gradient Descent |
| *SLP* | Sign Language Production |
| *SLR* | Sign Language Recognition |
| *SPF* | Serial Particle Filter |
| *SVM* | Support Vector Machine |

# Chapter 1

# Introduction

Communication is demonstrated utilizing symbols, signs, and semiotic norms, both sender and recipient are mutually aware of the transmission of meanings or messages from an individual or group to another. One of the most crucial life skills is the ability to interact effectively. This enables us to communicate with others and comprehend information that is conveyed to us. Natives utilize their mother tongue everywhere as tools for communication. Hearing deprived populations, however, are communicating via Sign Language that is not generally used by individuals. As a consequence, there is still a communication gap. Some disabilities exist in approximately 1 billion people worldwide. The second greatest of these is hearing impairment [1]. Sign language implies how individuals with hearing impairments communicate their feelings, contribute to a discussion, educate and live as normal as possible in general. The Government of Bangladesh determined that the second most prevalent sort of disability in Bangladesh is hearing impairment [2]. As of November 2003, Bangladesh has over 7.6 million deaf people [3].

## 1.1 Background

Sign Language involves various forms of gestures in order to communicate, especially of the hands and arms, when it is impossible or undesirable to continue verbal communication. The practice of the language is perhaps older than speech. Each sign language is a discrete natural language with its own syntax and dictionaries, which is unique to the nation. Sign language is the combination of coded manual signs, reinforced by facial expression and conceivably augmented by words spelled out in a manual alphabet [4]. This can be used to keep the head above water when vocal communication is impossible. For instance, when one or more would-be communicators has speech or hearing disabilities.

As sign languages are the most used means of communication right after verbal communication, research on sign language has gained massive popularity over the past decade in the recognition field. However, even after successfully recognizing the gestures, there are not enough models that generate sign language to help the impaired population convey and comprehend other languages.

Hands can communicate. That too in a language that potentially has the most components when weighed to any other spoken language of the globe. In this language, fingers spell, the face talks, and the body transmits a person's inward musings to the communicators.

Sign Language is a perceptible language that consolidates facial impressions, looks, head movements, non-verbal communication, and surprisingly the space around the speaker. Hand gestures are the basis of sign language. Numerous signs are iconic, which means the gestures use a visual picture that looks like the idea it addresses. For example, to communicate the idea of "window" in BdSL, one would situate your palms in front and move it front and back to address the opening and shutting of a window. Actions are regularly communicated through hand signals that copy the addressed concept - if one wished to sign the idea "eat," he would bring your fingers and thumb of the dominant hand together as though holding food and afterward push the hand toward the mouth.

The alphabets are a significant series of signs [5] Some hand gestures for letters take after the composed type of the individual letter. It is called finger spelling when one uses gestures of letters in order to spell out a word. Finger spelling is helpful to pass on names or to ask somebody the sign for a specific idea. Though some sign languages use two hands, BdSL utilizes single-hand signals for each of the alphabets. Numerous individuals find finger spelling the most strenuous when grasping sign language, as refined speakers are exceptionally quick finger spellers.

Our proposed method will not only help the deaf and mute people but also help understand the tough finger spelling style of sign language. Previously many researches were done that recognized Bangla language from sign gestures. Architectures like Convolutional Neural Network,VGG19 and Artificial Neural Network (ANN) were used. Also, some animated works were also done using NMT, GAN etc.

The method that we came up with uses the feature that neural networks offer and generate appropriate sign language of the given input (Bangla alphabet). The main architecture that we are going to use is CNN (Convolutional Neural Network). At first, the classifier will be trained with Bangla Alphabetic Dataset. After that, while testing the image input would be given. Subsequently, mapping of Bangla alphabet to sign gesture will be done. The model will retrieve the corresponding sign gesture of the test alphabet as shown in Figure 1.1.

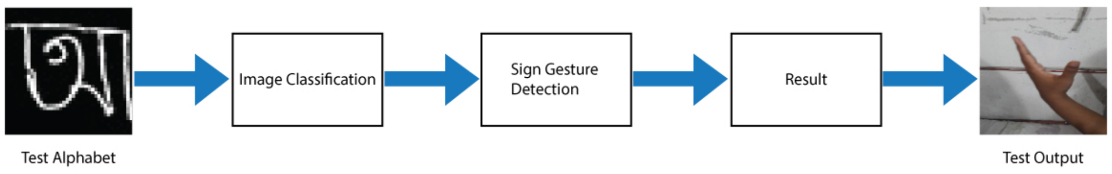


Figure 1.1: Workflow of the proposed method

Since our model will predict sign gestures from Bangla Alphabets, the higher accuracy will determine the usability of the model. To ensure this, we will be using various models to calculate the accuracy in order to compare it with the accuracy level of our model. Models like: Inception-V3, VGG16 and ResNet50 will be used for accuracy calculation and comparison.

## 1.2 Problem Statement

Our primary focus is the Generative Bangla Sign Language. Due to many technological advancements, dedicated scientists, and researchers, there are many research pieces regarding Sign Language technology. The research communities are currently working primarily on Sign Language Recognition. This may help us understand sign languages but will not help the deaf and mute community in a more significant margin. There is a considerable lack of resources when it comes to generative sign language. As our study is focused on accurately generating sign language alphabets, it is mandatory to have accurately trained our dataset accordingly. A learning model gets ideal effectiveness when it is fed with enough training data. Nevertheless, because of computational resource limitation, small sets of data have to be trained first to check the working model. Overfitting [6] happens when the neural network familiarizes itself with the training set so closely that it loses the capability to speculate and make a prognosis for new, obscure data. Hence, the network gets biased. Gradient vanishing problem is the derivatives [7] which were to be multiplied with the same amount of hidden layer, are smaller than usual; while back propagating the gradients will decrease gradually, and after a while, they will vanish.

## 1.3 Motivation

More than 7.6 million deaf people reside in Bangladesh. Their primary mode of communication is Bangla Sign Language. Rather than verbal correspondence, deaf individuals utilize signs to communicate and impart to other people. Among the language-based minorities, Bangla sign language users' society is in Bangladesh [8] Unfortunately, it is alarming that not many of us care to be acquainted with this communication mode, even though a large portion of our population uses it. Consequently, it leads to the suffering of the deaf-mute. Since most of the population are not accustomed to their mode of communication, they face immense difficulties communicating with the rest of their peers. Just because a portion of the population cannot hear or speak does not mean that their form of communication should not advance along with the rest of the world.

## 1.4 Research Objective

As stated in the problem statement, our primary focus of this study is to develop a new model that will take test alphabets and classify them with the help of our neural network model. Finally, it will generate the reciprocal sign gesture output of the given alphabet. As it is a generator of alphabets, the accuracy will have to be high enough for peak performance without suffering from much data leakage.

Therefore, in this research, we have focused on feeding our model with a big enough dataset to make it efficient while predicting the outcome. Meanwhile, we shall be using various models to calculate the accuracy for comparison purposes. The significant contributions of this thesis are stated as follows:

- Novelty: As mentioned, there is very little research on sign language generation. We have proposed a new method, first in our country, that generates Bangla Sign Gestures when the input is Bangla Alphabets.
- Accuracy: Our model has achieved 97.46% accuracy, which was acquired from the generic CNN model that we employed.

The rest of the report has been organized in the following manner. Chapter 2 discusses the literature reviews and related work on Sign Language. Chapter 3 describes the implemented dataset. Subsequently, in Chapter 4, we have articulated the implementation of the existing model, and our proposed method has been narrated in Chapter 5. After that, our experimental setup has been stated in Chapter 6. Finally, Chapter 7 concludes our thesis.

# Chapter 2

# Literature Review and Related Work

## 2.1 Convolutional Neural Network

A convolutional neural network is an arrangement of convolutional and pooling layers which allow removing the significant highlights from the pictures and accumulate them to form a feature map [9]. Convolutional neural networks collect data and transform it using a series of hidden layers. Initially, each concealed layer contains a large number of neurons, with each neuron being totally associated with all neurons in the previous layer, and neurons in a singular layer working transparently and sharing no affiliations. The hidden layer is the last linked layer, and both of the settings affect the course scores. Convolutional Neural Structures take advantage of the fact that the data contains pictures to make the preparation more sensible. The layers of a ConvNet have neurons masterminded in three assessments: width, stature, and importance, in contrast to a typical Neural Set up. There are three types of layers in CNN. Convolutional layers, pooling layers, and fully connected layers are the three types. CNN will transform the first input layer by layer using convolutional and down sampling procedures to produce course scores for classification and relapse purposes using this specific technique [10]. The preview of a basic diagram of CNN is given in Figure 2.1.

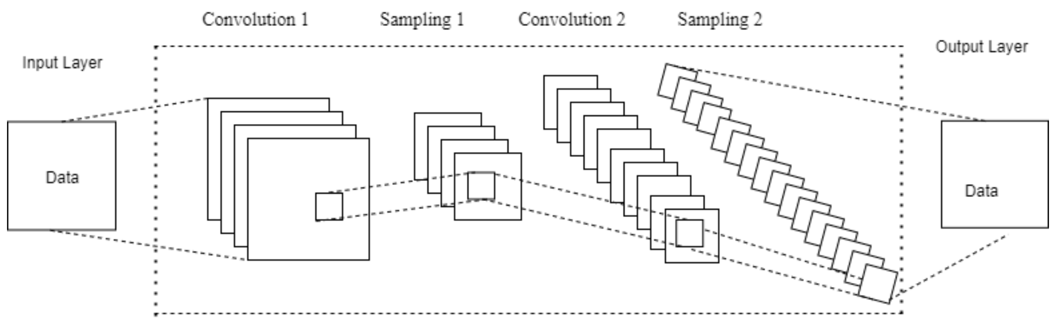


Figure 2.1: Basic CNN Architecture 01

## 2.2 Layers

### 2.2.1 Convolutional layer

The convolutional layer will determine the yield of neurons bound to neighboring input locales by measuring the scalar product of their weights and the input volume's locale. The corrected straight unit points to apply Introduction to Convolutional Neural System enactment work such as sigmoid to the yield of the activation delivered by the past layer [11]. Convolution is a specialized sort of straightforward operation utilized to include extraction, where a little cluster of numbers, called a part, is connected over the input, which is a cluster of numbers, called a tensor. As a result, the kernels have a low spatial dimensionality and are distributed throughout the entire depth of the input. As the data reaches a convolutional layer, it convolves each channel over the input's spatial dimensionality to create a 2D actuation outline [9].

Kernel convolution isn't just for CNN anymore; it's a crucial part of a variety of other computer vision calculations. It is a setup in which we take a small network of numbers, transfer it over our picture, and adjust it depending on the channel's values. The subsequent include outline values are determined using the following equation, where the input picture is denoted by f and our bit by h. The consequence network's line and column lists are verified with m and n, respectively. Which is shown in Equation 2.1

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k] \qquad (2.1)$$

We take each estimation from bit and duplicate them in sets of corresponding values from the image after placing our channel over a chosen pixel. Finally, we totalled everything and placed the outcome in the appropriate place inside the yield, including the outline [12].

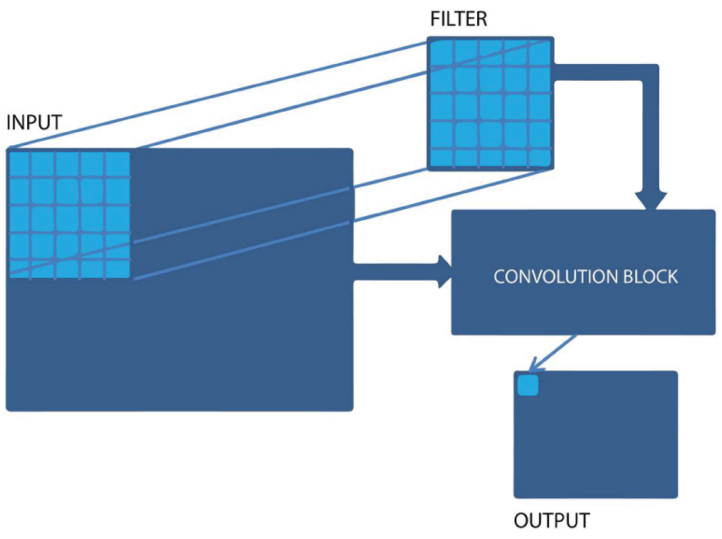


Figure 2.2: Basic CNN Architecture 02

In the above Figure 2.2, a basic internal architecture of CNN has been shown.

### 2.2.2 Pooling layer

Pooling is best thought of as down-sampling, which reduces the difficulty of promoting layers. Within the image processing space, it can be considered as comparative to reducing the determination. In a ConvNet architecture, intermittently embedding a Pooling layer in the middle progressive Conv layers is not uncommon. It should theoretically reduce the spatial scale of the representation in order to reduce the number of borders and calculations in the entity and, as a result, manage overfitting. Using the Maximum operation, the Pooling Layer acts independently on each profundity cut of the information and resizes it spatially. The most well-known arrangement is a pooling layer of channels of size $2 \times 2$ added with 2 downsampled, each profundity cut in the contribution by 2 along both width and stature, and each profundity cut in the contribution by 2 along both width and stature, removing 75 percent of the enactments [13]. The depth measurement remains unchanged.

### 2.2.3 Fully connected layer

Fully connected layer is just a feed-forward neural network. These layer structure the last couple of layers in the network. The contribution to the completely associated layer is the yield from the last. A convolutional neural arrangement is an arrangement of convolutional and pooling layers that allow removing the foremost highline fully-connected layer that is comparable to how neurons are organized in a conventional neural arrangement [11]. In this manner, each node in a fully connected layer is straightforwardly associated to every node in the past and also associated with the later ones. After the highlights are extracted by the convolution layers and downsampled by the pooling layers, they are mapped to the ultimate yields of the final layer of fully connected layer [13]. The number of yield nodes in the ultimate fully connected layer is usually equal to the number of groups. As seen in Figure 2.3 [14], each fully connected layer is followed by a nonlinear work, such as ReLU.

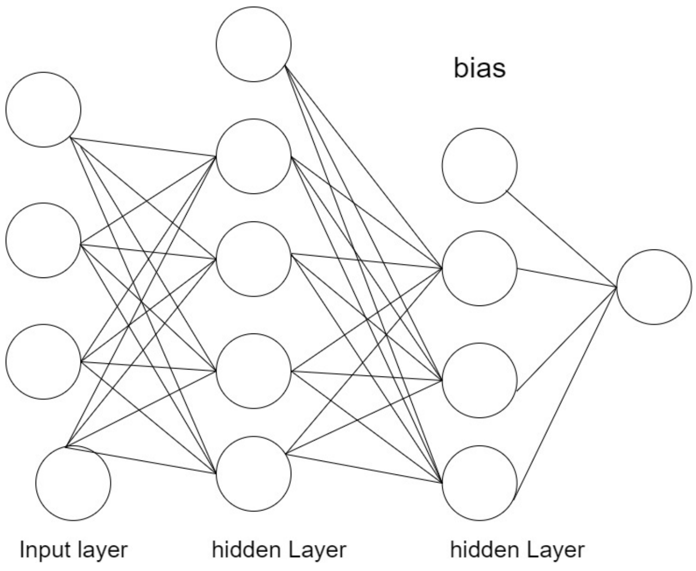


Figure 2.3: Fully connected layer

## 2.3 Activation Functions

The primary function of the activation function is to calculate weights with bias of the hidden layers and then determine the activation of neurons. It defines how input weight sum converts into outputs in the neurons of the layers. These functions are used to control gradient and learning rate in a network model. Basically, activation functions are classified into three parts: Binary, Linear, and Non-Linear activation function. In our case, we used Linear activation functions. Some basic activations are demonstrated below:

### 2.3.1 Sigmoid Function

This particular function is used [15] mainly in basic neural network application and logistic regression. But for more complex and leading-edge neural models, the sigmoid function is avoided. The equation for sigmoid function is provided in Equation 2.2:

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{2.2}$$

This function exists [16] between (0,1). So, sigmoid will be used if the prognosis of probability is in between the range of 0 and 1. It has a S-shaped differentiable curve. This function is monotonic. The main disadvantage of this function is data loss; the deeper the network, the more loss it gains.

### 2.3.2 Hyperbolic Tangent (TanH)

The [16] TanH function is in the range of -1 to 1. It is also a sigmoidal (S-shaped) function, the only difference is the range is mapped in a way that negative points are mapped at the negative axis and zeros are mapped very close to zero.

$$y = 2\sigma(2x) - 1 \tag{2.3}$$

In the above Equation 2.3, we can see the depiction of the activation function. TanH is used basically in allocation of classes. This function is zero centered [17] but has a Vanishing Gradient issue.

### 2.3.3 Rectified Linear Unit (ReLU)

ReLU is a piecewise linear function which gives direct output of the given input. It has a range of (0, $\infty$). ReLu is the most famous activation function in the deep learning era which can be briefly depicted in the following Equation 2.4:

$$y = max(0, x) \tag{2.4}$$

The main advantage [15] of using ReLU is it solves the vanishing gradient problem, it is one sided unlike TanH, has sparse activation (50%), it does not have back propagation error. This function and its derivatives are monotonic. The drawbacks are that this function is non-zero centered and non-differentiable by zero. Another drawback is the dying ReLU problem because half of its outputs remain inactive (returned as 0) for non-zero centered action.

### 2.3.4 Leaky ReLU

For solving the dying ReLU problem, this Leaky ReLU appeared in the neural network field. It defines [18] an extremely small linear numerical component to overcome the non-zero input issue.

$$y = max(0.01 * x, x) \tag{2.5}$$

The above equation (Equation 2.5), briefly defines the activation function. For any negative input, it returns the x multiplied with 0.01 so that negative output also can be gained. The range [16] is $(-\infty,\ \infty)$ and both function and derivatives are monotonic.

### 2.3.5 Softmax

This function gives [15] us probability distribution output. It maps the input as the total summation becomes 1. The following Equation 2.6 defines this function:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}}; \quad for\ j = 1, 2, 3, ....\ k \tag{2.6}$$

It is used for multi-classification in a logistic regression model, used in output layers for those networks that classify the inputs into multiple groups.

## 2.4 Existing Deep Learning Models

### 2.4.1 VGG16

VGG16 is a convolutional neural mastermind. Educators K. Simonyan and A., in their paper "Significant Convolutional Frameworks for Huge Scope Picture Acknowledgement", set up the show for VGG16. The show achieves 92.7% top-5 test precision [19] in ImageNet, that could be a dataset consisting over 14 million images that have a spot to 1000 classes. It rolls out the refinement over AlexNet by replacing gigantic portion computed channels (11 and 5 inside the $1^{st}$ two convolutional layers, independently) with various 3×3 bit approximated channels in a stable progression. Using NVIDIA Titan Dual GPU, VGG16 was organized for quite a while [20].

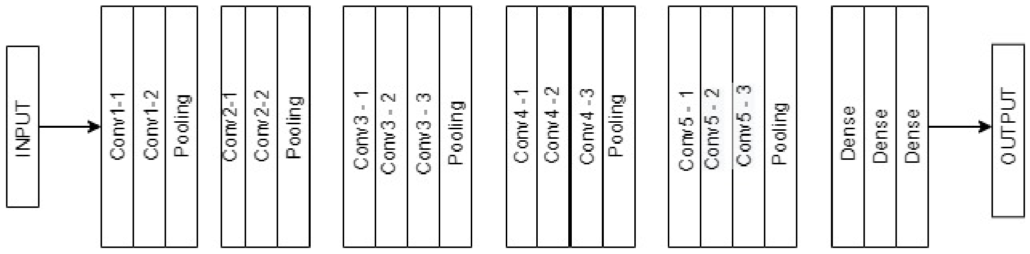


Figure 2.4: VGG16 Basic Architecture

In this figure 2.4, The obligation to the cov1 layer is of settled gauge $224 \times 224$ RGB picture[21]. The image has been undergone a number of convolutional layers. In which, $3 \times 3$ estimated narrow channels were used. It also occupies $1 \times 1$ convolution channel, and this is viewed as a linear modification of the information channels (followed by non-linearity). The convolution step is settled to 1 pixel; the spatial cushioning of Conv layer input is with the end target that the spatial goal is obtained after convolution, for outline the cushioning is 1-pixel for $3 \times 3$ Conv layers [22]. Five max-pooling layers are used to finish spatial pooling, which take after a package of the Conv. layers. Three fully connected layers take after a number of convolutional layers (which wires an assorted significance in multiple models): the crucial two have 4096 channels each, the third carries out 1000-way ILSVRC [19] mastermind and in such way holds 1000 channels (one for each course). All covered layers are set up with the modification (ReLU) non-linearity. It is also notable that none of the frameworks (in any case for one) contain Local Response Normalization (LRN); such kind of normalization does not ground the execution on the ILSVRC dataset, anyway prompts extended memory use and computation time.

### 2.4.2 RESNET 50

ResNet50 could be a variety of ResNet models with 48 Convolution layers in addition to 1 MaxPool and 1 Typical Pool layer. It also has $3.8 \times 10^9$ Drifting centers tasks [23]. It very well is a comprehensively employed ResNet show, and we have inspected ResNet50 designing significance. In 2012 at the ILSVRC2012 order challenge AlexNet [24] won the essential expense; afterwards, ResNet was the first inquisitive aspect which happened to the PC vision and the 13 significant learning world. Due to the framework showed by ResNets, it got possible to plan ultra-significant neural frameworks, and through that, we are remorseless that we can sort out big-amount of layers and still gain excellent execution. The ResNets were first applied to the picture acknowledgment task, but since it is indicated inside the paper, the framework can too be employed for non-PC vision works besides achieving better precision. In the Figure 2.5, a basic architecture of Resnet50 is shown.

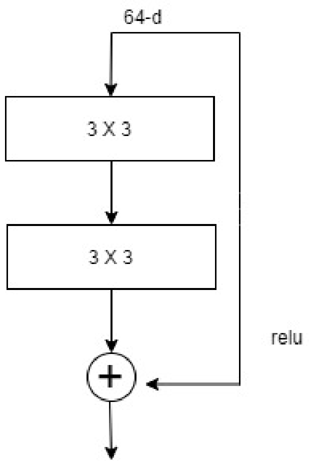


Figure 2.5: Resnet50 Architecture

In the wake of starting with a solitary [23] Convolutional layer and Max Pooling, there are 4 similar layers with reasonable changing channel sizes – every one of them using $3 \times 3$ convolution activity. As well, after every 2 convolutions, we are bypassing/skirting the layer in the middle. These skipped affiliations are called 'character backup way to go associations', and livelihoods are called extra squares.

### 2.4.3 Inception v3

The inception [25] engineering is highly tunable, meaning that there are several conceivable changes to the number of channels within the various layers that do not influence the qualityof the completely prepared network. Initiation v3 remained as the main sprinter up of ILSVRC-2015 [26]. In the meantime, it had been pronounced with finishing 12 million tasks. Commencement v3's building view holds significant differential enhancement in channel sizes. As a model, Initiation v1 has a convolution channel estimated around $5 \times 5$, which is supplanted in v3 with two $3 \times 3$ filters as shown in Figure 12.1.4 In beginning v3, a proficient matrix size decrease is proposed, highlight maps are finished by convolutions and by pooling independently. In both arrangements of highlight, maps are linked and sent to the following beginning module. Inception v3 is joined with 42 layers. Notably, each channel is factored in. Amazingly, it is rumored with 78.8% accuracy [27] obtained as indicated by Top 1 accuracy ImageNet—inception module with factorization of $n \times n$ convolutions. At the beginning of v3, as convolutions are factored to more modest and into deviated convolutions, another origin module can be formed. With 42 profound layers, the calculation expense is around 2.5 more than the beginning and significantly more productive compared to VGGNet. The inception-v3 model got symmetric and asymmetric building blocks in addition to convolutions, max-pooling layers, average pooling, dropouts, and fully connected layers. Its architecture is shown in figure 2.6.

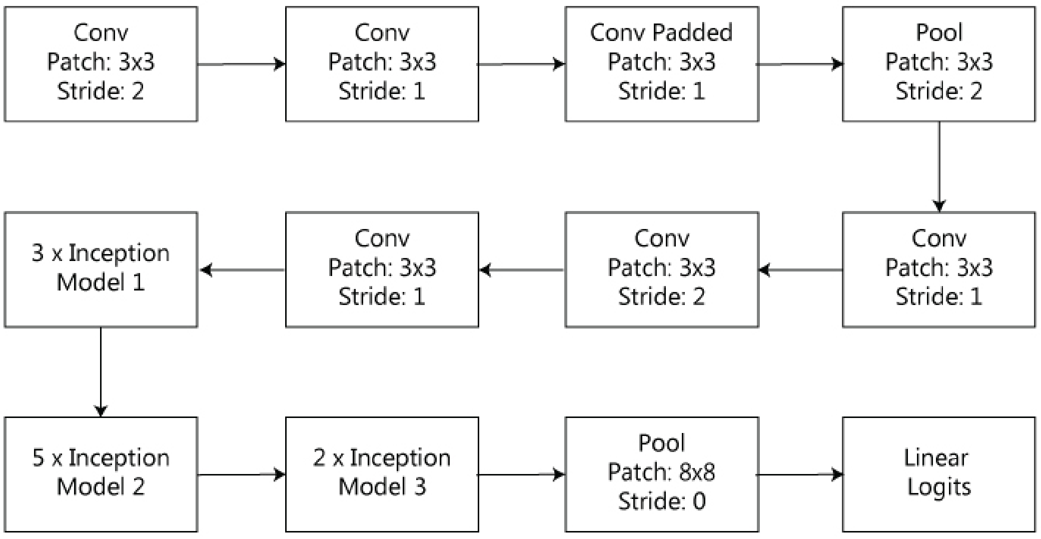


Figure 2.6: Inception v3 Architecture

The following Figure 2.7 depicts the workflow of each of the models we employed to train our dataset and find its accuracy over different models.

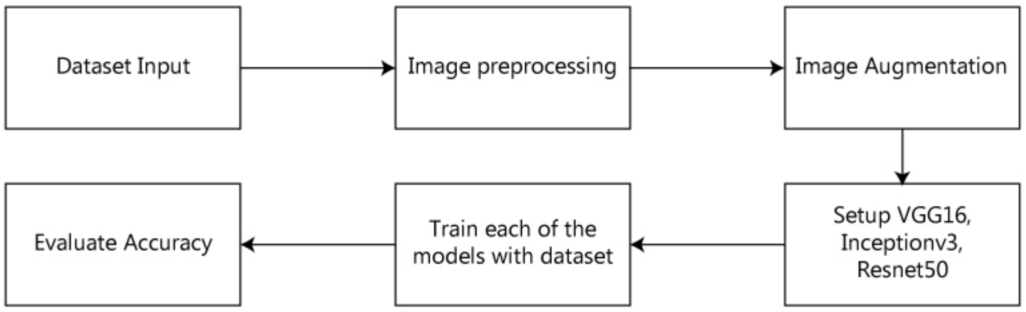

Dataset Input
Image preprocessing
Image Augmentation
Setup VGG16, Inceptionv3, Resnet50
Train each of the models with dataset
Evaluate Accuracy

Figure 2.7: Workflow of each model

## 2.5 Loss Function

### 2.5.1 Cross Entropy Loss

Cross-entropy loss [28], which is also known as log loss, is a method which amounts to the pursuance of a codification model which gives a probabilistic output between 0 and 1. Where the predicted probability differs from the real mark, cross-entropy loss increases. The cross-entropy loss is defined below in Equation 2.7:

$$CE = -\sum_{i}^{c} t_{i\ log(s_i)} \tag{2.7}$$

In Equation 2.7, $t_i$ and $s_i$ are the ground truth and the score for each CNN class i in C. SoftMax loss, also known as categorical cross entropy loss, is a combination of SoftMax activation and a cross entropy loss, which is shown below in Equation 2.8:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^c + e^{s_i}} + CE = -\sum_{i}^{C} t_{i\ log(f(s_i))} \tag{2.8}$$

Binary cross entropy loss, also known as Sigmoid loss, is a combination of the Sigmoid activation function and cross entropy loss. Unlike Softmax loss, it acts sovereign for each vector component. In the Equation 2.9 given below, we can see its mathematical depiction:

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1-t_1) log(1-f(s_1)) \tag{2.9}$$

Cross-entropy [29] is a capacity from the field of information theory, gaining on entropy and commonly measuring the difference among probability distributions. It measures the number of bits required to serve or broadcast an average event from one dissemination to another.

## 2.6 Optimization Algorithm

In case of deep learning, optimization algorithm is used for training purposes, it upgrades the cost function. The generic equation (Equation 2.10) of this algorithm is given below:

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} L(y'^i, y^i) \tag{2.10}$$

In our model, we used Adam optimizer which is explained below:

### 2.6.1 AdaM

AdaM is a very famous algorithm known for its quick performance. It stands for Adaptive Momentum. AdaM associates the propulsion RMSprop in at once. This makes AdaM a very strong and fast algorithm. This method [30] has a very forthright implementation, very efficient in estimation, and takes up very less memory. It is more efficient for larger problems with data/parameters. The below Equations 2.11 and 2.12 define the optimizer:

$$\alpha_t = \alpha . \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \tag{2.11}$$

$$\Theta_t \leftarrow \Theta_{t-1} - \alpha_t . m_{t/(\sqrt{v_t}+\hat{\epsilon})} \tag{2.12}$$

As we faced gradient descent issues, we used an AdaM optimizer to overcome the issue as it gives us advantages to store exponentially disintegrating averages of previous similar gradients.

## 2.7 Related Works

To ensure open and direct contact between the hearing impaired and mute communities, it is critical to develop comprehensive programs that can convert spoken languages into sign languages. This can be accomplished by Sign Language Recognition and Production (SLR/SLP). We are attempting to develop a device that can understand text and translate it into Bangla Sign Language, as previously mentioned. We want to strive on SLP because most deaf people find reading spoken language difficult. As a result, conversion into sign language is a necessity.

Many previous studies have been performed on detecting Bangla sign language and classifying hand gestures using SVM, ANN, and KNN. Despite this, we have been unable to locate study that uses CNN to classify Bangla sign language hand signals (Convolutional Neural Network). However, the usage of CNN to recognize sign language is nothing new. In a recent work [31] on recognition of Bangla sign language, the authors used CNN architecture to recognize numerical numbers and alphabets separately. For this, they have used a dataset of 30916 samples which contained 23864 alphabets and 7052 numerical samples. For data pre-processing, they normalized the greyscale images by dividing with highest grey level (255) and resized the gestures into $64 \times 64$ pixels. Their proposed network contained 6 convolutional layers, 3 pooling layers and a kernel size of $5 \times 5$ and $3 \times 3$. For training, they did not use any separate sets; they divided the total data randomly into $K = 10$ folds, where 9 folds for training and the remaining one for testing. They trained their model with 0.001 learning rate, 200 epochs, batch size of 128 and steps per epoch of 64. Using this training method, they have got 100% testing accuracy on numerical numbers and 99.83% testing accuracy for recognition of Bangla sign gestures.

In another recent similar work [32] on recognition of Bangla sign gestures, the authors used a pre-trained VGG19 model of CNN architecture then modified that model into recognizing gestures. For the dataset, they used a total of 320 samples of each character. They resized their images into 224 $\times$ 224 for data preprocessing. They modified the VGG-19 architecture by taking frozen outputs from the intermediate layer to prevent updating during backpropagation and the last MaxPool layer was left out by the authors from VGG-19; dropout was used for reducing overfitting and on the final dense layer, a Softmax activation is added. The authors trained the model by splitting the dataset into training and testing parts. They used an SGD optimizer with a learning rate of $1e-3$. They used 100 epochs for training. Using the highest confirmation accuracy's weight, they achieved their testing accuracy at 89.6% for recognition of Bangla sign gestures.

As per our research, we found out that there is no research available on generating Bangla sign language from text (Bangla alphabet), but there are some papers on generating sign language from text in different languages. In this paper [33] on English speech to American sign language transformation, the authors used automatic speech recognition block for recognizing speech from a video, Neural Machine Translation (NMT) module for transforming that speech into ASL, a video generator which generates animated puppet with ASL gestures. For training purposes, they used ASLG-PC12 corpus as their dataset and split it into three sections: training, development and testing. Then they trained their model for an epoch of 50, batch size 64, Adam Optimizer with $\beta_1 = 0.9, \beta_2 = 0.98$ and $\epsilon = 10^{-9}$. They did not speak clearly of their accuracy in their journal.

There was a research not so long ago in India [34]. where they generated animation based Indian sign gestures where the authors chose a different approach using HamNoSys notation. The generated HamNoSys based on Indian sign language and then generated SiGML corresponding to the HamNoSys notations created earlier. As the model has been tested on only 100 words, the testing accuracy was 100%.

In another recent work [35] on sign language production, the authors used Neural Machine Translation (NMT) and Generative Adversarial Networks (GAN) architecture. The authors used NMT for translation purposes (spoken language to sign gloss sequence), then used OpenPose for mapping purpose between glosses and skeletal sequences, lastly used GAN for generating sign language video sequences. For training purposes, the authors used PHOENIX 14TH and SMILE Sign Language Assessment Dataset. For German to gloss transformation, they used four layers with 1000 GRUs each. The authors used Luong et al.'s approach as an attention mechanism with a learning rate of $10^{-5}$, 30 epochs and 0.2 dropout. The input images enter into 5 convolutional layers with a 128$\times$128$\times$10 binary heat map. For generating sign language video, the authors trained their model with a learning rate of $2 * 10^{-5}$. The system generates a 4sec./frame video. The generated skeletal sequences are passed down to the "LUT + GAN" approach and then tested the system against the dataset in the "FULL" approach.

In another work, Bangla sign language was implemented using SIFT feature extraction and Convolutional Neural Network (CNN) [36]. The authors used SIFT and CNN to remove scaling problems and to classify the hand gestures precisely. The dataset includes 7600 images of Bangla signs and letters for testing and training. They used the skin masking technique to get only the Region of Interest for the image of the hand. Feature extraction was done using SIFT and k-means clustering and Bag of Features to represent the features. Finally, they used CNN as histograms. They obtained 88-90% accuracy with SIFT and 77-78% accuracy without SIFT in the Bangla letters.

Another Indian sign language recognition research was done [37]: a combination of Neural Network with Genetic Algorithm, Evolutionary algorithm (EA), and Particle Swarm Optimization (PSO). The researchers used a set of 22 ISL hand gestures as a dataset, and 70% of the data was used to train the Neural Network. The accuracy achieved by them was 99.96% by NN-PSO.

One more research was done which developed a system that recognized speech and generated the corresponding sign language in an animation form. They used a technique where data was read from a floppy disk and then transferred to a video RAM, displayed on CRT display in an animation form. They used some Japanese sign language and letters for the system. Kawai.H has created a system that can identify speech in real time and make the associated animation in a personal computer like a sign language sequence. It consists of three video RAMs (VRAM) and a speech recognition board, which recognizes each individual speaker's voice 22 who is registered. Currently, on two floppy disks there are 40 sign language patterns and 50 finger spelling stored [38].

A serial particle filter [39] was used for isolated sign language recognition for American sign language in more recent research. They used single gesture detection and used a Serial Particle Filter (SPF) to feature a covariance matrix. They obtained an accuracy of 87.33% to recognize the American Sign Language.

There was another research [40] done for recognition of sign language using Convolutional Neural Networks (CNN). They used CNN, Microsoft Kinect, and GPU acceleration for the recognition system. They used the dataset from ChaLearn Looking at People 2014, which includes 20 Italian gestures in different environments by different performers. They used a total of 6600 gestures, where 4600 gestures are for training and 2000 for testing. They achieve an accuracy of 91.7%.

# Chapter 3

# Dataset

We used two types of static datasets for our model.

- Bangla Handwritten alphabet dataset
- Bangla Sign Language alphabet dataset

Bangla Handwritten alphabet dataset consists of 98164 images. This is split into a training dataset consisting of 78513 images and a testing dataset of 19651 images. Our train to test ratio is 80:20. In Figure 3.1, few images from our dataset is shown. We collected this dataset from BanglaLekha-Isolated and later on done our own labelling and rescaling [41]

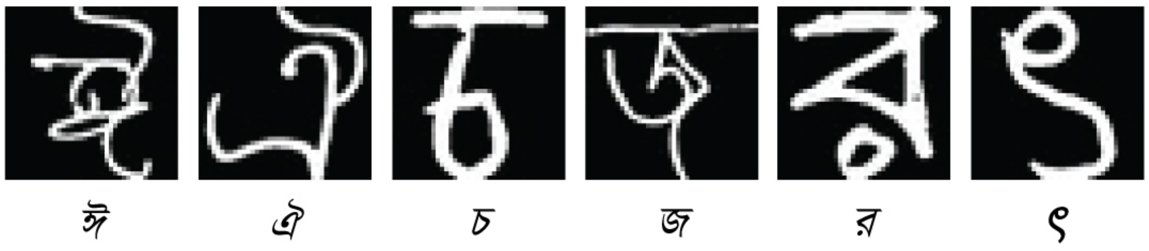


Figure 3.1: Bangla Handwritten Alphabet Dataset (Sample)

Our aim was the generation of corresponding Bangla Sign images against their equivalent Bangla Handwritten letter. We used a dataset from existing research from Kaggle for this part of our work. The dataset consists of images of bare hands. These were collected with the help of National Federation of the Deaf, 12581 different images in forms of hand gestures of 38 BdSL alphabets [42]. Each of the 38 sub-sets consist of approximately 290-312 images. Some sample images in the dataset are shown in Figure 3.2. The images were re-sized to 224 $\times$ 224 pixels.

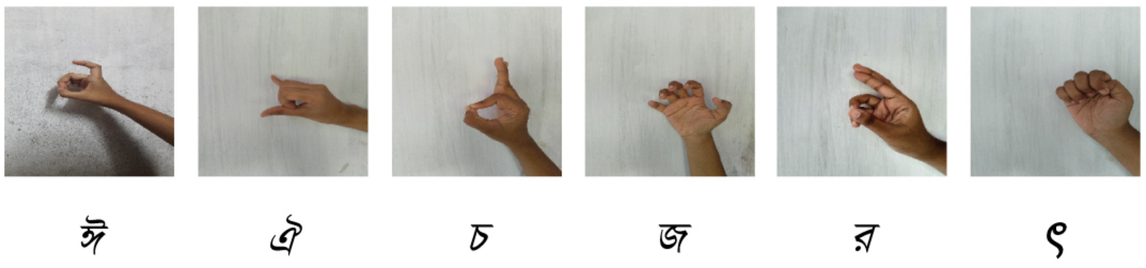
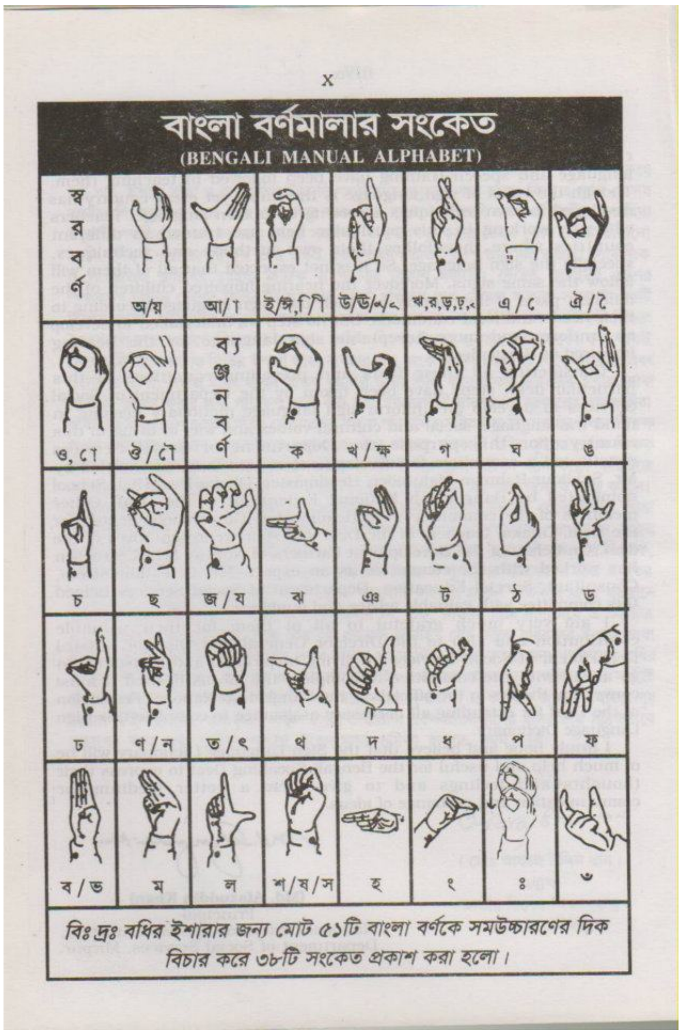


Figure 3.2: Bangla Sign Alphabet Dataset (Sample)

## 3.1 Dataset Labelling

The pre-existing dataset used numerical values to label the sign alphabets. For our ease of work, we re -labelled them according to the Bangla letter they represent. We used the sign language chart in Figure 3.3 to correctly label our sign gesture images to their corresponding letter.



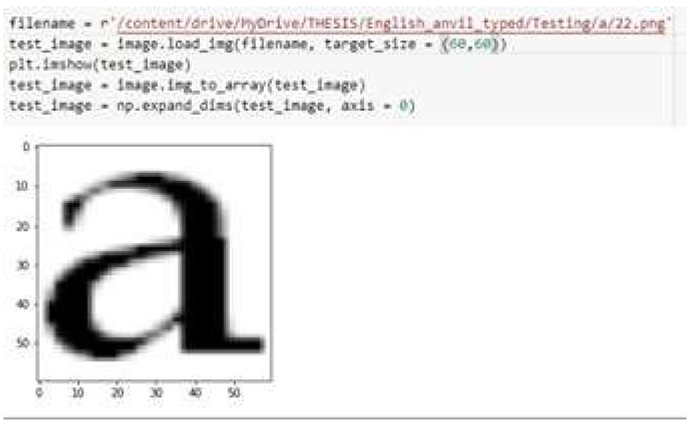
বিঃ দ্রঃ বধির ইশারার জন্য মোট ৫১টি বাংলা বর্ণকে সমউচ্চারণের দিক বিচার করে ৩৮টি সংকেত প্রকাশ করা হলো।

Figure 3.3: Bangla Sign language alphabet chart

## 3.2 Working with the dataset

We imported all the images in a sequential manner from the dataset image directory using the command *flow_from_directory*, and processed them using ImageDataGenerator. We stated batch size, which represents the number of training examples that will be used in each iteration, while keeping the target size of the photos proportional to the input size. Because each of the datasets we used had numerous classes on which our model will make predictions later, we set the *class_mode* to categorical.

A pre trained CNN model was tested with different datasets. Before training the model with our Handwritten Bangla alphabets datasets, did some trial runs with some various datasets we collected from studies done on sign language. An English-Alphabets dataset was used to test the accuracy of the CNN model. After 100 epochs, We achieved training accuracy of over 97% after the model ran 100 epoch. We had an acceptable visual of the training set from the dataset which is shown in Figure 3.4.

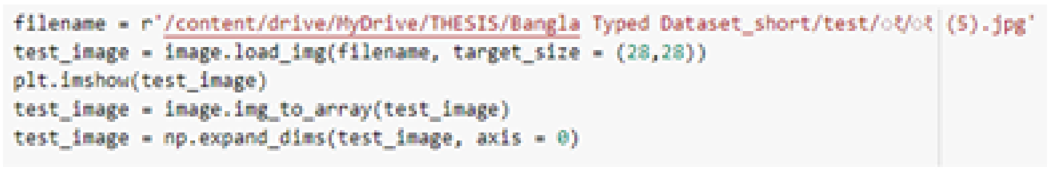


```
filename = r'/content/drive/MyDrive/THESIS/English_anvil_typed/Testing/a/22.png'
test_image = image.load_img(filename, target_size = (60,60))
plt.imshow(test_image)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
```

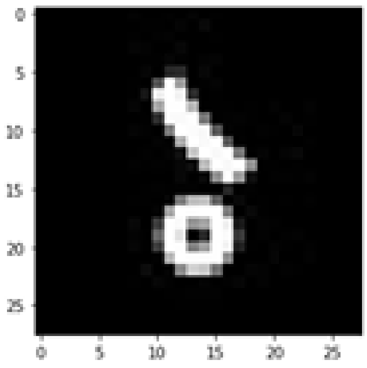
```
Epoch 98/100
16/16 [==============================] - 3s 180ms/step - loss: 0.1756 - accuracy: 0.9565
Epoch 99/100
16/16 [==============================] - 3s 178ms/step - loss: 0.0854 - accuracy: 0.9631
Epoch 100/100
16/16 [==============================] - 3s 176ms/step - loss: 0.0577 - accuracy: 0.9762
```

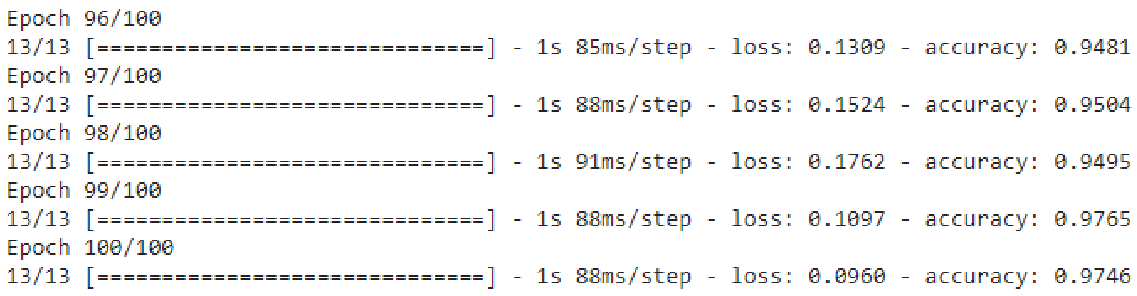
Figure 3.4: Test results on English alphabet dataset

While working with our Bangla handwritten datasets, we faced issues since our dataset was huge, we also faced some data losses. As per the instruction of our supervisor we created a smaller version of a digitally typed Bangla alphabet datasets consisting of 2500 images, with each of the letters having 50 variants of images (tilted, rotated, flipped). In our CNN model we had 97% accuracy for that particular dataset. Which is shown in Figure 3.5.



```
filename = r'/content/drive/MyDrive/THESIS/Bangla Typed Dataset_short/test/ঠ/ঠ (5).jpg'
test_image = image.load_img(filename, target_size = (28,28))
plt.imshow(test_image)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
```

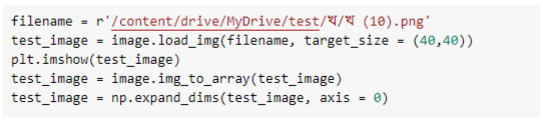




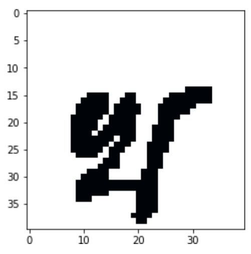
```
Epoch 96/100
13/13 [==============================] - 1s 85ms/step - loss: 0.1309 - accuracy: 0.9481
Epoch 97/100
13/13 [==============================] - 1s 88ms/step - loss: 0.1524 - accuracy: 0.9504
Epoch 98/100
13/13 [==============================] - 1s 91ms/step - loss: 0.1762 - accuracy: 0.9495
Epoch 99/100
13/13 [==============================] - 1s 88ms/step - loss: 0.1097 - accuracy: 0.9765
Epoch 100/100
13/13 [==============================] - 1s 88ms/step - loss: 0.0960 - accuracy: 0.9746
```

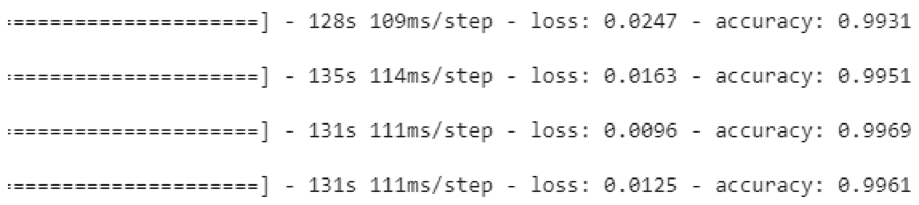
Figure 3.5: Test results on Typed Bangla Alphabet Dataset

We also obtained a digitized Bangla-alphabets dataset from OnkoGan [43], with each image measuring 40 × 40 pixels. We used this dataset to train the model, and after 100 epochs, we had an accuracy of almost 99 percent. The inference-image of the input image reveals that the images in the dataset are quite perceptible for the model (shown in Figure 3.6.)



```
filename = r'/content/drive/MyDrive/test/খ/খ (10).png'
test_image = image.load_img(filename, target_size = (40,40))
plt.imshow(test_image)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
```



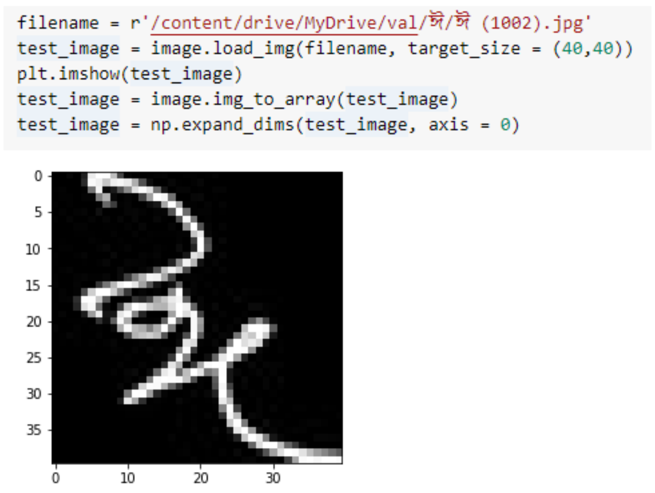


```
:====================] - 128s 109ms/step - loss: 0.0247 - accuracy: 0.9931

:====================] - 135s 114ms/step - loss: 0.0163 - accuracy: 0.9951

:====================] - 131s 111ms/step - loss: 0.0096 - accuracy: 0.9969

:====================] - 131s 111ms/step - loss: 0.0125 - accuracy: 0.9961
```

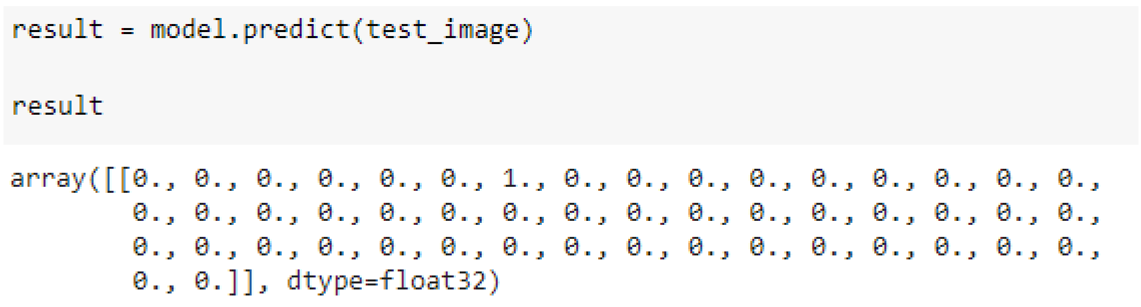
Figure 3.6: Test results on OnkoGan

Unfortunately, even with the promising results we could not proceed with the dataset as we failed to obtain a complete dataset for all the alphabets as there had occurred some data loss from where the dataset originated.

We proceeded with our original handwritten Bangla alphabet dataset. This time we were satisfied with the results. Here in figure 3.7 is a visual of our sample data in the code.



```
filename = r'/content/drive/MyDrive/val/ঈ/ঈ (1002).jpg'
test_image = image.load_img(filename, target_size = (40,40))
plt.imshow(test_image)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
```

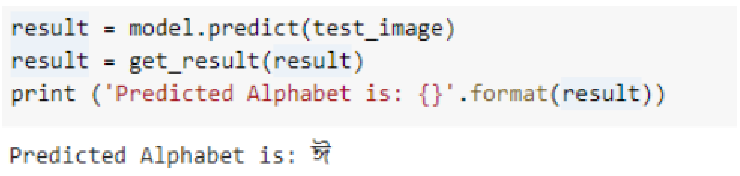


```
result = model.predict(test_image)

result
```

```
array([[0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0.]], dtype=float32)
```

Figure 3.7: Test image and Array Location of the test image

After training, our model could successfully identify the Bangla letter we fed into for testing and produce corresponding sign gestures for said letter which is shown in figure 3.8.



```
result = model.predict(test_image)
result = get_result(result)
print ('Predicted Alphabet is: {}'.format(result))
```

```
Predicted Alphabet is: ঈ
```

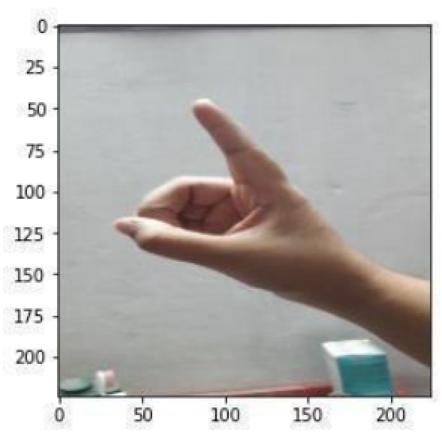


Figure 3.8: Generated sign gesture

## 3.3 Data Availability

The data used to support the findings of this study are available at -

- Bangla Handwritten dataset
- Bangla sign language dataset
- OnkoGan

# Chapter 4

# Implementation

## 4.1 Data preprocessing

We retrieved our dataset from Mendeley where the dataset is named as "BanglaLekha-Isolated". There are in total 98164 images in our dataset which we later on divided into 80:20 ratio for training and testing purposes. It is essential to process the data in particular manners prior to employing them to train a model and perform different tasks. Before using that data for the purpose that we want, we need it to be as organized and "clean" as possible [44]. We did batch resizing to the whole dataset and set the dimension of each of the images as $40 \times 40$. As for training CNN with a particular dataset, it is essential for the data to be labelled, we did a batch labelling to the dataset as well. We kept the size of the input samples reasonably small, which is $40 \times 40$, as we will be able to train it even with a thin network and by doing that, we can avoid the possibility of overfitting as well. Besides, if we did not keep the image size small, then we might have to face data loss while feeding them into deeper layers. Hence, we tried to keep the size of the data as small as possible from which we will get the maximum amount of image details and eventually train our network for image classification. In Figure 4.1 we have shown our dataset splitting.

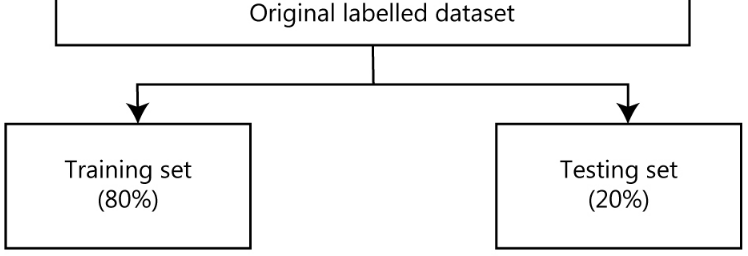


Figure 4.1: Dataset splitting

We used multiple models for training our dataset and finding out which one provides us with the highest accuracy. In each case, we followed the generic ways of preprocessing data which is discussed below:

## 4.2 Data augmentation

With a view to avoid overfitting issues, we did data augmentation with our existing dataset. We basically instigated variability on our dataset with accumulating no new data. Data augmentation can be used to address both the requirements, the diversity of the training data, and the amount of data [45]. Although, our neural network thinks of them as distinct data, we altered a few things from our training dataset:
rescaling- For rescaling, we took a value of 1/255 with which each of the training data gets multiplied.
shear_range- This attribute shears the angle of each of the images counterclockwise where the direction is in degrees. We applied .2 as the value for this attribute.
zoom_range- We used this attribute for zooming the images randomly. We set the value as .2.
horizontal_flip- We set the attribute as true, hence it randomly flips the images horizontally.
rotation_range- This attribute is to degree range for random rotations of the images.
width_shift_range- We set its value as .2. We used it for shifting each image to the left or to the right, horizontally.

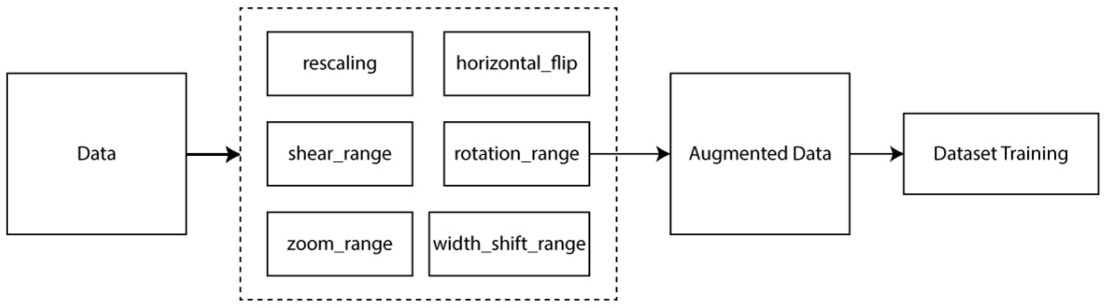


Figure 4.2: Data augmentation

The above Figure 4.2 shows the way we augmented our data.

## 4.3 Normalization

For keeping the gradient of a model in a stable state and for training it, the data has to be scaled in a particular way which is called normalization. It is basically used right after performing each layer of convolution operation. There are multiple techniques of doing normalization, for instance, group norm, instance norm, batch norm, layer norm etc. However, in our work we only used batch normalization.

## 4.4 Training set

Data augmentation makes several views of the images other than just one view of them, like, rotate, shift, flip them. This way the effect of overfitting gets minimized and the dataset gets enlarged. Our training augmentation has been already stated in the “Data Augmentation” section and for our test set, we kept the sample as it is, only for normalization, we rescaled the image by 1/255.

## 4.5 Model Training

We split our collected dataset into training and testing sets. These two sets are kept in two different directories. In the directory, where the training set is kept, it again divided into 50 subsections. As we used our model to classify 50 Bangla alphabets, each alphabet is kept in a different folder. In that very folder, there are hundreds of images of the same alphabet in different orientations and fonts. We tried to keep our inventory of images reasonably diversified so that the training of the model goes right and it can classify the image while we run the testing phase and come up with accurate prediction of the given input.
The figure 4.3 below portrays our implemented CNN model clearly with a visual representation that is given below:

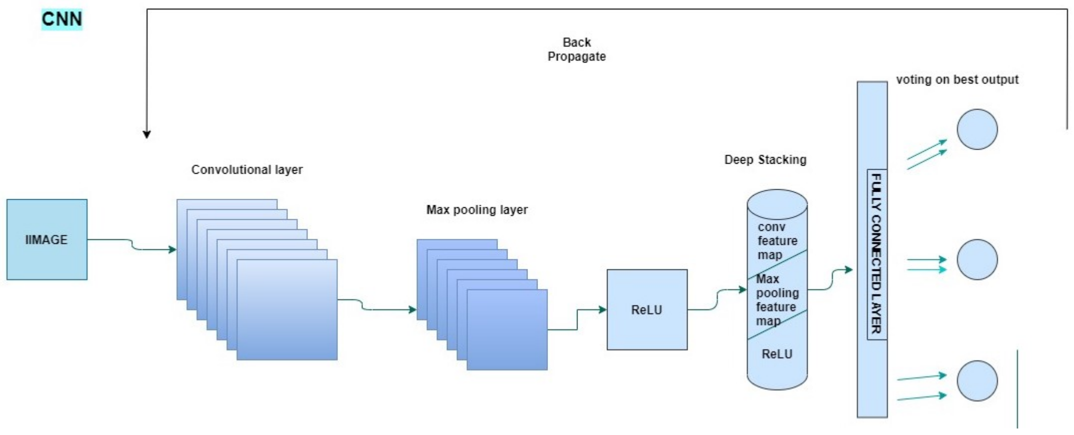
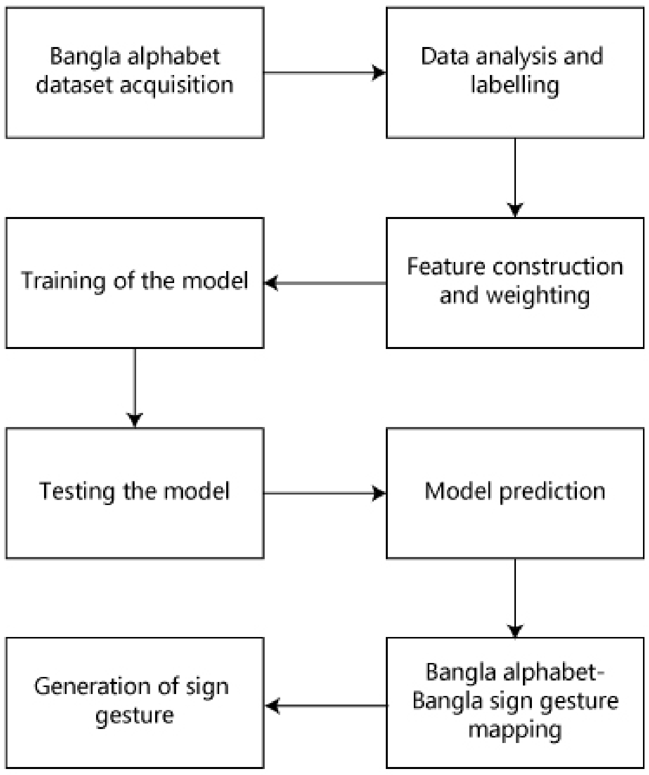


Figure 4.3: Visual Representation of CNN model

We trained our CNN model with the dataset and afterwards did a one-to-one mapping between the output that we get while testing phase, which is an alphabet and its corresponding sign gesture. The workflow of the whole thing is provided below in the figure 4.4:



Bangla alphabet dataset acquisition
Data analysis and labelling
Feature construction and weighting
Training of the model
Testing the model
Model prediction
Bangla alphabet-Bangla sign gesture mapping
Generation of sign gesture

Figure 4.4: Workflow of the CNN model

# Chapter 5

# Challenges

## 5.1 Data Dependency

A learning model gets optimum efficiency when it is fed with enough training data. The sum of data is directly proportional to the performance; the more data, the greater the performance. In our case, we faced some challenges while training our model.

## 5.2 Overfitting

Overfitting [6] happens when the neural network familiarizes itself with the training set so closely that it loses the capability to speculate and make prognosis for new, obscure data. It naturally happens when the dataset is not enough with respect to the network profundity. The network gets biased on the split training set and cannot recognize the testing set accurately. In our case, the dataset was quite small in the first phase of training the model. So, we faced some overfitting problems. Later we were able to increase our dataset sample in order to reduce overfitting.

## 5.3 Data Leakage

Data leakage happens when [6] the dataset is not split properly for training and testing purposes. It can also happen when the same data is split in both training and testing sides. This problem hides the actual performance and the performance degrades when it gets familiarized with new data. In our case, we split the dataset into training (78,513 data) and testing (19,651 data), while training, some data got lost in the process.

## 5.4 Excessive training time

We, the authors, did not have access to a high functioning computer with GPU so that we had to train our model using Google Collaboratory. It took too much time and some network issues increased our training complication.

## 5.5 Gradient Vanishing

Gradient vanishing problem is basically the derivatives [7] which were to be multiplied with the same amount of hidden layer, are smaller than usual; while back-propagating the gradients will decrease gradually and after a while they will vanish. In our case we faced this issue while training, while we tried to back propagate, the loss function gets small enough so the model cannot differentiate between previous and updated weight. While fetching the weights from the hidden layers, the model fails to recognize the accurate weight to be fetched.

# Chapter 6

# Experimental Setup

## 6.1 Model Testing

The testing is done to detect the predicted output of the model. During the training period, we have taught our model to identify different kinds of Bangla alphabets. For that we used a labelled dataset, where each of the images of an alphabet is named by the name of the alphabet itself. For testing the model, we take a directory of a particular image from the test dataset. The image has been kept completely isolated from the training dataset but due to extensive training of the model, it can successfully predict the name of the alphabet visible on the test image. In figure 6.1, a complete workflow of our task is shown.

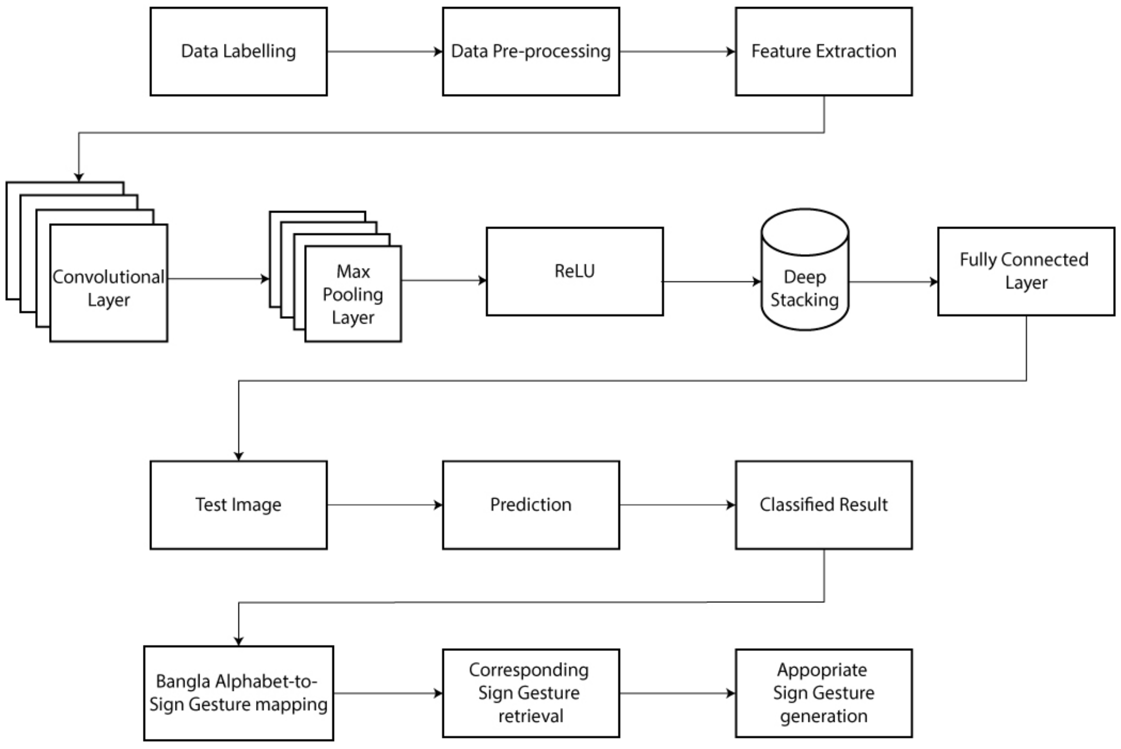


Figure 6.1: Complete workflow

## 6.2 Results and analysis

To evaluate the performance of a model, we have to rely on a few attributes namely Testing Accuracy, Epoch and Loss Function. Testing accuracy per epoch defines the correctness rate of a particular model and that of the loss function defines the difference between our generated result and the accurate result. The latter one is used for determining the error rate of the model. In our experiment, we tried to extract results with least loss function and for that we employed high epoch, and in the meantime, used a reasonably big dataset that assures the model to be trained properly. For image classification, we took four approaches, as in, used for models to figure out variations of accuracy in different models. For that, we used a general CNN model, VGG16, Inceptionv3 and Resnet50. As mentioned before, to achieve our goal, first we trained a classifier so that it can successfully recognize a particular alphabet during the testing phase. Henceforth, our initial challenge lies on predicting the alphabet accurately. Afterwards, we simply did a mapping between the predicted alphabet and its corresponding sign gesture. So, here our training and testing phase comprises all the experiments that we conducted on our alphabet dataset.

| epochs | Training Accuracy | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 21.30 | | | | | | | | | |
| 2 | 29.30 | 30.10 | | | | | | | | |
| 3 | 21.21 | 29.94 | 31.62 | | | | | | | |
| 4 | 22.34 | 22.24 | 23.44 | 33.18 | | | | | | |
| 5 | 29.22 | 29.67 | 30.20 | 31.34 | 31.22 | | | | | |
| 6 | 20.54 | 21.50 | 21.43 | 21.27 | 22.19 | 22.13 | | | | |
| 7 | 32.22 | 33.43 | 31.25 | 31.76 | 33.49 | 34.21 | 36.33 | | | |
| 8 | 24.57 | 22.76 | 23.21 | 23.54 | 24.33 | 25.10 | 26.41 | 26.13 | | |
| 9 | 31.23 | 32.41 | 34.10 | 32.81 | 35.07 | 32.65 | 31.76 | 33.34 | 34.59 | |
| 10 | 23.21 | 24.99 | 25.23 | 32.82 | 35.31 | 35.64 | 34.02 | 34.59 | 36.17 | 35.11 |

Table 6.1: Accuracy of VGG16 classifying image

In the above table 6.1, we found that, while training the model with our dataset, the accuracy differs based on the number of epochs set for each training. For this experiment, we trained the model for 10 times and in course of time, we increased the number of epochs. On the very first experiment, when the epoch is set as 1, the accuracy happens to be pretty low, which is only 21.30%. It fluctuates through-out the whole experiment and at the end of 10th experiment and on its last epoch, we got the accuracy as 35.11%. Point to be noted, this is only the initial stage of training where the epoch is set as low. After training each of the models for a long time, we eventually got decent accuracy for each of the models.

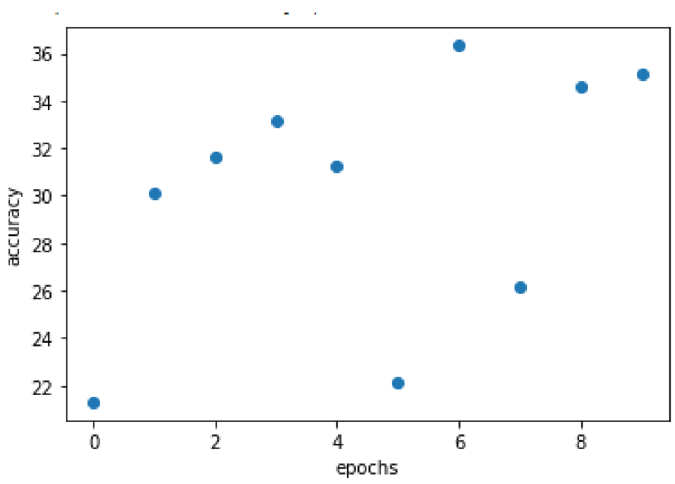


Figure 6.2: Accuracy of VGG16 for 10 epochs

For plotting the above scattered diagram 6.2, we took the accuracy of the final epoch of each experiment. Here we tried to portray the distribution of the accuracies in terms of epoch-count. We can see that the 7th experiment provided us with the highest accuracy in comparison to the others

| epochs | Training Accuracy | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 29.70 | | | | | | | | | |
| 2 | 31.30 | 31.23 | | | | | | | | |
| 3 | 21.76 | 22.54 | 22.23 | | | | | | | |
| 4 | 23.80 | 23.21 | 24.10 | 27.18 | | | | | | |
| 5 | 23.00 | 23.45 | 24.24 | 24.18 | 24.12 | | | | | |
| 6 | 24.54 | 24.11 | 26.53 | 27.19 | 37.25 | 39.24 | | | | |
| 7 | 30.61 | 31.25 | 31.76 | 32.44 | 33.18 | 34.67 | 34.17 | | | |
| 8 | 34.57 | 33.15 | 35.76 | 25.76 | 28.90 | 30.32 | 30.18 | 32.53 | | |
| 9 | 24.23 | 35.25 | 36.26 | 36.78 | 29.12 | 30.53 | 30.34 | 31.86 | 32.87 | |
| 10 | 22.21 | 22.34 | 22.65 | 23.46 | 34.51 | 25.33 | 24.18 | 23.42 | 34.57 | 35.38 |

Table 6.2: Accuracy of Inceptionv3 classifying image

In the above table 6.2, we see the accuracies of inceptionv3 in classifying the images. We did 10 different experiments for this and in every case, we increased the number of epochs by one, more than the previous one. Even though, the final accuracy in each experiment somewhat fluctuates through-out the whole experiment, yet it's evident that the accuracy increases over time, provided that the epoch-count gets increased. From the above experiment, we got the highest accuracy on the 6th experiment, where its final accuracy happens to be 39.24%. Though after increasing the number of epochs up to 10, we ended up having accuracy of 35.38%.

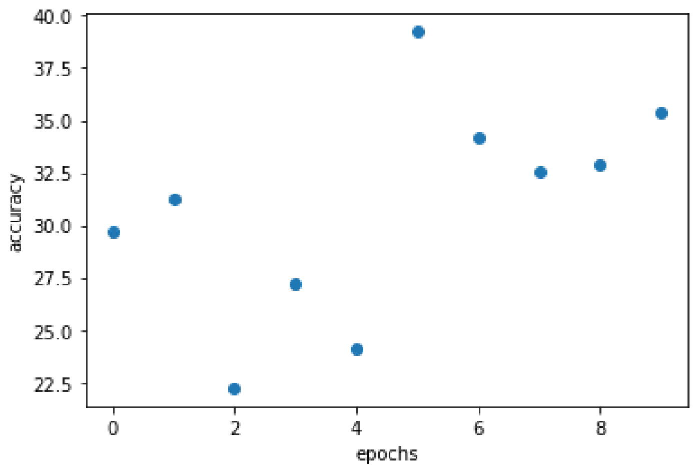


Figure 6.3: Accuracy of Inceptionv3 for 10 epochs

This figure 6.3 shows the training accuracy of inceptionv3 over 10 epochs. Here we can see that, on the 6th experiment we got highest accuracy at the end of the training, which is 39.24%. And the accuracy scored the lowest at the end of 3rd experiment. We plotted the scattered diagram taking the final accuracy of each of the experiments.

| epochs | Training Accuracy | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 28.32 | | | | | | | | | |
| 2 | 30.25 | 31.43 | | | | | | | | |
| 3 | 31.41 | 32.51 | 32.03 | | | | | | | |
| 4 | 22.23 | 23.46 | 35.80 | 26.48 | | | | | | |
| 5 | 24.22 | 23.55 | 24.29 | 24.21 | 24.43 | | | | | |
| 6 | 23.54 | 24.36 | 33.26 | 35.19 | 37.44 | 38.34 | | | | |
| 7 | 23.20 | 21.47 | 31.77 | 32.44 | 33.78 | 34.17 | 34.44 | | | |
| 8 | 26.57 | 27.25 | 27.53 | 29.16 | 28.10 | 30.12 | 30.40 | 32.61 | | |
| 9 | 31.23 | 36.15 | 36.46 | 36.98 | 28.99 | 31.43 | 32.46 | 31.80 | 32.29 | |
| 10 | 22.94 | 31.24 | 30.44 | 31.20 | 32.55 | 33.88 | 33.35 | 34.42 | 35.50 | 35.61 |

Table 6.3: Accuracy of general CNN classifying image

The above table 6.3 defines different training accuracy of CNN for our dataset. After training our general CNN model, which we eventually employed to complete our full task, that is, generation of sign gesture, we found out variations in the accuracy. We conducted 10 different experiments for it and increased the epochs over time. In each experiment, though the accuracy fluctuated a little, in most cases, it got increased over time. Here, the least accuracy marked is 21.47% and that of the highest is 38.34%.

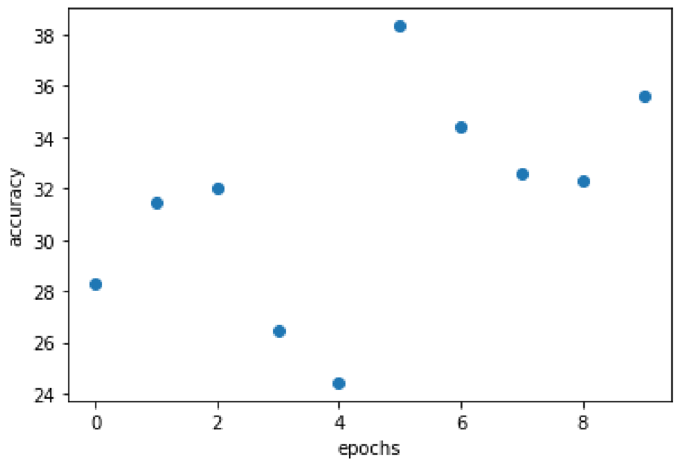


Figure 6.4: Accuracy of CNN for 10 epochs

The above graph 6.4 portrays the final accuracy of each of the experiments that we performed on our CNN model with our dataset. In these experiments, the 6th experiment scored the highest final accuracy among all the experiments performed.

| epochs | Training Accuracy | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 18.40 | | | | | | | | | |
| 2 | 19.30 | 21.36 | | | | | | | | |
| 3 | 22.61 | 21.81 | 22.10 | | | | | | | |
| 4 | 15.42 | 23.93 | 25.81 | 26.26 | | | | | | |
| 5 | 25.83 | 23.68 | 24.28 | 24.33 | 24.91 | | | | | |
| 6 | 23.54 | 24.82 | 24.29 | 25.60 | 26.36 | 27.91 | | | | |
| 7 | 24.67 | 26.21 | 17.77 | 21.57 | 22.41 | 23.78 | 24.20 | | | |
| 8 | 26.93 | 23.19 | 25.73 | 25.21 | 18.11 | 19.67 | 20.44 | 21.73 | | |
| 9 | 23.77 | 21.05 | 22.06 | 22.78 | 19.16 | 20.42 | 21.65 | 21.26 | 22.80 | |
| 10 | 24.12 | 20.31 | 22.05 | 22.56 | 23.55 | 24.29 | 24.99 | 25.24 | 24.08 | 24.81 |

Table 6.4: Accuracy of Resnet50 classifying image

The above table 6.4 shows the accuracy of Resnet50 after we trained it without our dataset. We trained our dataset with the resnet50 classifier as well. For this, we again performed 10 experiments with variations in the number of epochs. After performing the experiments, we observed that the highest accuracy achieved is 27.91% and that of the lowest accuracy is 15.42%.

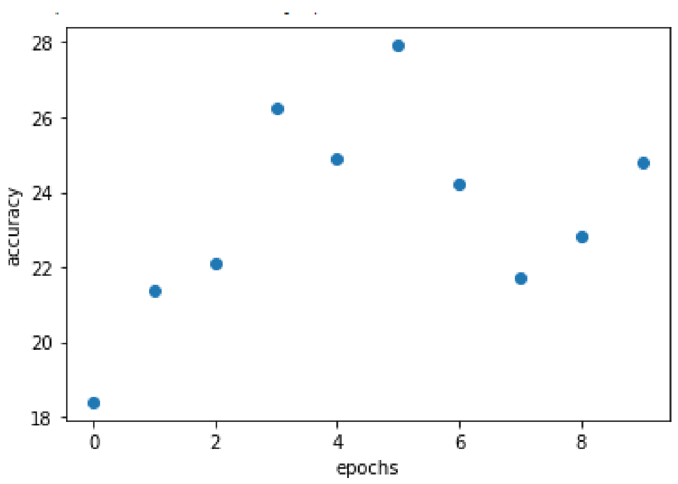


Figure 6.5: Accuracy of Resnet50 for 10 epochs

In the scattered diagram 6.5, we plotted the final accuracy of each of the experiments. In the first experiment, we got the lowest final accuracy and in the case of the 6th one, the final accuracy happens to be the highest, which is 27.91%.

| epochs | Test Accuracy (%) | | | | Loss Function (%) | | | |
|---|---|---|---|---|---|---|---|---|
| | VGG16 | inceptionV3 | CNN | Resnet50 | VGG16 | InceptionV3 | CNN | Resnet50 |
| 1st | 35.29 | 33.14 | 35.19 | 30.13 | 61.31 | 62.38 | 59.36 | 66.91 |
| 2nd | 34.46 | 34.82 | 36.40 | 31.44 | 53.12 | 55.27 | 54.12 | 65.17 |
| 3rd | 36.88 | 36.11 | 34.61 | 33.91 | 50.27 | 43.66 | 53.47 | 61.92 |
| 4th | 37.91 | 33.65 | 36.99 | 33.99 | 49.19 | 39.10 | 51.21 | 59.10 |
| 5th | 38.40 | 35.46 | 37.80 | 34.81 | 48.37 | 38.45 | 50.82 | 54.15 |

Table 6.5: Test accuracy and Loss function of all the models

The above table 6.5 shows the test accuracy and loss function of all the models which we used for training our dataset. We trained each of the models with our dataset. We set the epoch as 100 for training each model but for compiling them in a table, we just retrieved the test accuracy and loss function of each model up to 5th epoch. Here we can see, all the models show promising results in terms of accuracy as they proceed further, as in, the accuracy increases with the increase of epoch. For VGG 16, initially it is 35.29% but in course of time, on the 5thepoch it becomes 38.40%. As for Inceptionv3, the initial accuracy is 33.14% and at the end of 5th epoch, it becomes 35.46%. We also trained our generic CNN model and found the test accuracy on the 1st epoch as 35.19% and on the 5th epoch, it reached 37.80%. But in the case 42of Resnet50, the accuracy is a little less than other models. On the 5th epoch, for this model, we achieved 34.81% accuracy. As the accuracy increases in every epoch, the loss function of each of the models decreases accordingly. In case of VGG16, the loss function decreases from 61.31% to 48.37%. For Inceptionv3 it's 62.38% to 38.45%; 59.36% to 50.82% for CNN and lastly for Resnet50, it is 66.91% to 54.15%.

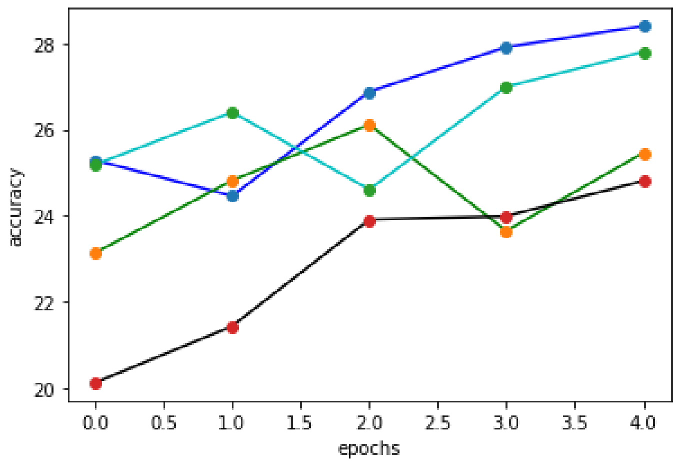


Figure 6.6: Test accuracy of different model

In the above figure 6.6, we made a graph that shows the differences between the test accuracy of different models over variable epochs. Here the blue, green, indigo, brown lines depict VGG16, Inceptionv3, CNN and Resnet50 respectively. It's evident that the accuracy of VGG16 is the highest, then comes CNN, and then Inceptionv3 followed by Resnet50.

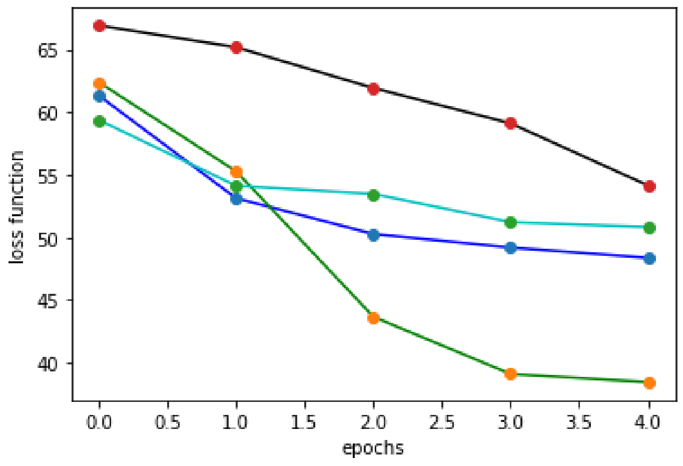


Figure 6.7: Loss function of different model

In this figure 6.7, blue, green, indigo, brown lines depict loss function of VGG16, Inceptionv3, CNN and Resnet50 respectively. Here, the loss function of Inceptionv3 decreases faster than any other models. And in the case of Resnet50, its loss function decreases the least.

| Batch Size | Accuracy |
|---|---|
| 16 | 96.91 |
| 32 | 96.67 |
| 64 | 92.28 |
| 128 | 94.19 |
| 512 | 94.83 |

Table 6.6: Accuracy of VGG16

The above table 6.6 shows the accuracy of VGG16 for different batch size. We trained our model, VGG16 with the dataset and set the epochs as 100. We intended to observe the variations between the accuracies with variations in the batch size. With the increase of batch size, the accuracy of the model decreases. When the batch size is set 16, the accuracy is 96.91% and right after the batch size is increased to 512, the accuracy comes down to 93.83%.

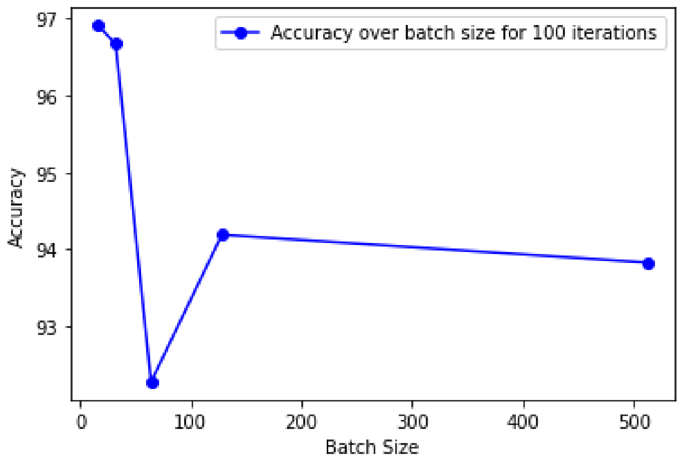


Figure 6.8: Accuracy of VGG16

In the figure 6.8, it shows how the accuracy of the model decreases over time, in terms of the increment of the batch size

| Batch Size | Accuracy |
|---|---|
| 16 | 96.34 |
| 32 | 95.88 |
| 64 | 93.18 |
| 128 | 92.71 |
| 512 | 91.91 |

Table 6.7: Accuracy of Inceptionv3

The above table 6.7 defines the accuracy of Inceptionv3 for different batch size of our dataset. In case this case, the accuracy after the training fell, once we started to increase the batch size. With a smaller size of the batch, we got higher accuracy of the overall model.

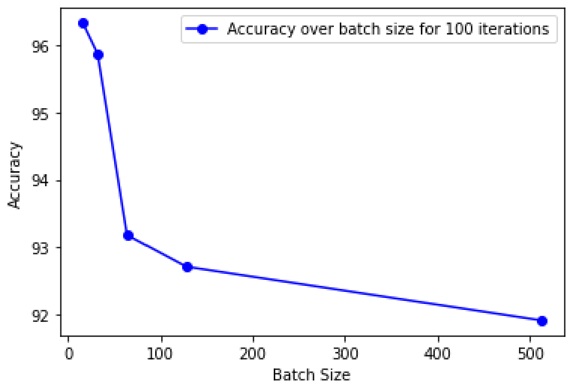


Figure 6.9: Accuracy of Inceptionv3

In this line diagram 6.9, we can see that the accuracy falls for the increased value of the batch size. As we double the batch size in each experiment, the accuracy of the model decreases accordingly.

| Batch Size | Accuracy |
|---|---|
| 16 | 97.79 |
| 32 | 97.59 |
| 64 | 97.46 |
| 128 | 93.90 |
| 512 | 93.55 |

Table 6.8: Accuracy of CNN

In this table 6.8, we can see the accuracy of the CNN model after training it while setting different values of batch size. When the batch size is 16, the accuracy that we achieved for the model is 97.79%.
Afterwards, we increased the batch size a few times and finally we set it as 512 and for that the accuracy turns out to be 93.55% which is pretty less than the one that we got for batch size 16. The accuracy is shown in Figure 6.10:

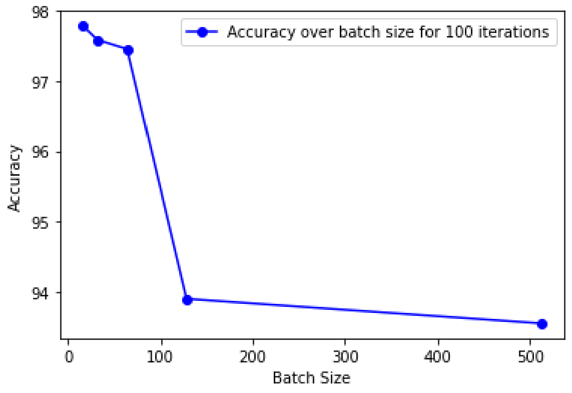


Figure 6.10: Accuracy of CNN

We plotted the graph taking the accuracy and batch size of the model in account. Just like mentioned above, the accuracy of the model got down as we increased the batch size which we showed in our graph.

| Batch Size | Accuracy |
|---|---|
| 16 | 90.10 |
| 32 | 88.64 |
| 64 | 87.45 |
| 128 | 86.62 |
| 512 | 86.41 |

Table 6.9: Accuracy of Resnet50

The given table 6.9 shows the accuracy of Resnet50 for variation in batch size. Last, we trained our Resnet50 model with the dataset where for batch size 16, we got accuracy which is 90.10%. We trained the model for five different batch sizes and the accuracy decreased for increased value of batch size. Hence, for the batch size of 512, we got an accuracy of 86.41%. The accuracy is shown in Figure 6.11:

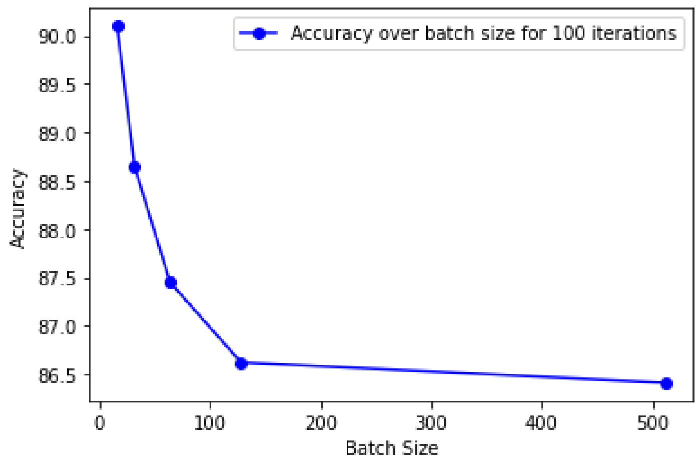


Figure 6.11: Accuracy of Resnet50

| Model | Epochs | Accuracy |
|---|---|---|
| VGG16 | 100 | 92.28 |
| Inceptionv3 | 100 | 93.18 |
| CNN | 100 | 97.46 |
| Resnet50 | 100 | 87.45 |

Table 6.10: Accuracy of different models

The above table 6.10 shows the final accuracy of different models. Finally, we got the accuracy of all the models that were trained and tested with the same dataset and the epoch count was set the same for each of them. As the batch size of 64 happens to be more feasible for us to run the codes, we considered the accuracy of each of the models having a batch size of 64. That way, we found the accuracy of the generic CNN model highest in terms of generating sign gestures, which is 97.46%. As for VGG16 and Inceptionv3 it's 92.28% and 93.18% respectively. In the case of Resnet60, we got the lowest accuracy, which is 87.45%.

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

Sign language is a way of conversing where one may communicate by visual movements and signals. Usually, they send messages using their hands, finger gestures, and facial expressions to convey the desired piece of information successfully. Generating sign language from text inputs using neural networks as an approach is established as a one-of-a-kind milestone. Many pieces of research are conducted in recognizing Bangla Sign Language but not in the generation sector. Our paper mainly used Convolutional Neural Network as a classifier and trained it further with the dataset by which we got the classified image. On top of that, we used one-to-one mapping in order to retrieve the corresponding gesture to our input. We used various pre-trained models like Inception-v3, VGG16, and ResNet50 to calculate and compare the accuracy levels. Eventually, we have efficiently retrieved the appropriate sign gestures after the model successfully classified the alphabets. The accuracy of classification is 97.46%. We got from the generic CNN model that we have employed.

## 7.2 Future Work

Setting aside what we have effectively accomplished, there are a few impediments that should be addressed. The major obstruction of this research is the scarcity of computational resources, such as the limitations of the GPU. Since the dataset we used is quite extensive, the training period took much longer because of the low GPU. In addition, we faced complications in classification and prediction since we used smaller datasets to train. We solved that by augmenting an improved dataset. We could have avoided this situation by accessing a higher GPU. Also, we did one-to-one mapping, which could be done automatically. Furthermore, in our research paper, we used letters for generating sign gestures. Our future goal is to improve this research to generate sign gestures by using sets of sentences—moreover, our target is to retrieve animated gestures. We could not accomplish it in the current research because of scarce time and resources. It can be done by using NMT (Neural Machine Translation) that will generate automated outputs. We will process it with GAN (Generative Adversarial Network) to create the final output that is the animated sign gesture of the given sentence.

# Bibliography

[1] J. Silberner, *Nearly 1 in 7 people on earth are disabled survey finds*, vol. 1, [Online]. Available: https://www.npr.org/sections/health-shots/2011/06/09/137084239/nearly-1-in-7-people-on-earth-are-disabled-survey-finds.

[2] "National strategy on prevention of deafness and hearing impairment in bangladesh: 2011–2016," 2011.

[3] I. Research Directorate and C. Refugee Board, *Bangladesh: Societal attitudes towards handicapped people, including those who are deaf and mute, and the treatment of those who promote their rights by the authorities and the jamaat-e-islami (2000-2004)*, 2004. [Online]. Available: https://www.refworld.org/docid/41501bec1c.html.

[4] The Editors of Encyclopaedia Britannica", *Sign language — communications*, Nov. 2020. [Online]. Available: https://www.britannica.com/topic/sign-language.

[5] J. Strickland, *How Sign Language Works*, Apr. 2021. [Online]. Available: https://people.howstuffworks.com/sign-language.htm.

[6] A. Munappy, J. Bosch, H. H. Olsson, A. Arpteg, and B. Brinne, "Data management challenges for deep learning," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2019, pp. 140–147. DOI: 10.1109/SEAA.2019.00030.

[7] R. Dagli, *Vanishing/ Exploding Gradients in Deep Neural Nets and solving them*, Apr. 2020. [Online]. Available: https://medium.com/swlh/vanishing-exploding-gradients-in-deep-neural-nets-and-solving-them-9d6070f28b29.

[8] *National Activities - Bangladesh - Bangladesh Sign language Day*, Feb. 2014. [Online]. Available: http://www.dpiap.org/national/article.php?countryid=017&id=0000029&country=Bangladesh.

[9] Arc, "Convolutional neural network," *Medium*, 2018. [Online]. Available: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05.

[10] *CS231n Convolutional Neural Networks for Visual Recognition.* [Online]. Available: https://cs231n.github.io/convolutional-networks/.

[11] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *ArXiv e-prints*, Nov. 2015.

[12] P. Skalski, *Gentle Dive into Math Behind Convolutional Neural Networks*, Apr. 2019. [Online]. Available: https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9.

[13] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[14] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, vol. 1, Apr. 2014. DOI: 10.3115/v1/P14-1062.

[15] H. S, *Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning*, Jan. 2020. [Online]. Available: https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e.

[16] S. Sharma, *Activation Functions in Neural Networks - Towards Data Science*, Dec. 2019. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[17] K. Srinivasan, A. K. Cherukuri, D. Vincent P M, A. Garg, and B.-Y. Chen, "An efficient implementation of artificial neural networks with k-fold cross-validation for process optimization," *Journal of Internet Technology*, vol. 20, pp. 1213–1225, Jun. 2019. DOI: 10.3966/160792642019072004020.

[18] H. Mujtaba, "What is Rectified Linear Unit (ReLU)? — Introduction to ReLU Activation Function," Dec. 2020. [Online]. Available: https://www.mygreatlearning.com/blog/relu-activation-function/.

[19] H. Qassim, A. Verma, and D. Feinzimer, "Compressed residual-vgg16 cnn model for big data places image recognition," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, 2018, pp. 169–175. DOI: 10.1109/CCWC.2018.8301729.

[20] M. Hassan, *VGG16 – Convolutional Network for Classification and Detection*, Feb. 2021. [Online]. Available: https://neurohive.io/en/popular-networks/vgg16/.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[22] P. Huilgol, *Top 4 Pre-Trained Models for Image Classification with Python Code*, Dec. 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/.

[23] E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. De Geus, "Malicious software classification using transfer learning of resnet-50 deep neural network," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2017, pp. 1011–1014.

[24] A. Kaushik, *Understanding ResNet50 architecture*, Jul. 2020. [Online]. Available: https://iq.opengenus.org/resnet50-architecture/.

[25] L. D. Nguyen, D. Lin, Z. Lin, and J. Cao, "Deep cnns for microscopic image classification by exploiting transfer learning and feature concatenation," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2018, pp. 1–5.

[26] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[27] S. M. Sam, K. Kamardin, N. N. A. Sjarif, N. Mohamed, *et al.*, "Offline signature verification using deep learning convolutional neural network (cnn) architectures googlenet inception-v1 and inception-v3," *Procedia Computer Science*, vol. 161, pp. 475–483, 2019.

[28] R. Gomez, "Understanding categorical cross-entropy loss, binary cross-entropy loss, softmax loss, logistic loss, focal loss and all those confusing names," *URL: https://gombru. github. io/2018/05/23/cross_ entropy_loss/(visited on 29/03/2019)*, 2018.

[29] J. Brownlee, "A gentle introduction to cross-entropy for machine learning," *Machine Learning Mastery*, vol. 20, 2019.

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

[31] M. S. Islalm, M. M. Rahman, M. H. Rahman, M. Arifuzzaman, R. Sassi, and M. Aktaruzzaman, "Recognition bangla sign language using convolutional neural network," in *2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, 2019, pp. 1–6. DOI: 10.1109/3ICT.2019.8910301.

[32] A. M. Rafi, N. Nawal, N. S. N. Bayev, L. Nima, C. Shahnaz, and S. A. Fattah, "Image-based bengali sign language alphabet recognition for deaf and dumb community," in *2019 IEEE Global Humanitarian Technology Conference (GHTC)*, 2019, pp. 1–7. DOI: 10.1109/GHTC46095.2019.9033031.

[33] D. Manzano, "English to asl translator for speech2signs," 2018.

[34] S. Kaur and M. Singh, "Indian sign language animation generation system," in *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, 2015, pp. 909–914. DOI: 10.1109/NGCT.2015.7375251.

[35] S. Stoll, N. C. Camgoz, S. Hadfield, and R. Bowden, "Text2sign: Towards sign language production using neural machine translation and generative adversarial networks," *International Journal of Computer Vision*, vol. 128, no. 4, pp. 891–908, 2020.

[36] S. S. Shanta, S. T. Anwar, and M. R. Kabir, "Bangla sign language detection using sift and cnn," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2018, pp. 1–6. DOI: 10.1109/ICCCNT.2018.8493915.

[37] S Hore, S Chatterjee, V Santhi, N Dey, A. Ashour, V. Balas, and F Shi, "Indian Sign Language Recognition Using Optimized Neural Networks," *Advances in Intelligent Systems and Computing*, pp. 553–563, 2016. DOI: 10.1007/978-3-319-38771-0\{_}54.

[38] H. Kawai and S. Tamura, "Deaf-and-mute sign language generation system," *Pattern Recognition*, vol. 18, no. 3, pp. 199–205, 1985, ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(85)90045-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0031320385900457.

[39] K. Lim, A. Tan, and S. Tan, "A feature covariance matrix with serial particle filter for isolated sign language recognition," *Expert Systems with Applications*, vol. 54, Feb. 2016. DOI: 10.1016/j.eswa.2016.01.047.

[40] L. Pigou, *Sign Language Recognition Using Convolutional Neural Networks*, Sep. 2014. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-16178-5_40?error=cookies_not_supported&code=01146a37-d59c-40e4-996d-c6d1aa413fa8.

[41] N Mohammed, S Momen, A Abedin, M Biswas, M. Shopon, G Shom, and R Islam, *BanglaLekha-Isolated*, Feb. 2017. DOI: 10.17632/hf6sf8zrkc.2.

[42] *Bengali Sign Language dataset*, Mar. 2020. [Online]. Available: https://www.kaggle.com/muntakimrafi/bengali-sign-language-dataset.

[43] S. Haque, S. Shahinoor, A. S. A. Rabby, S. Abujar, and S. Hossain, "Onkogan: Bangla handwritten digit generation with deep convolutional generative adversarial networks," in. Jul. 2019, pp. 108–117, ISBN: 978-981-13-9186-6. DOI: 10.1007/978-981-13-9187-3_10.

[44] R. Miller, *Data Preprocessing: what is it and why is important*, Dec. 2019. [Online]. Available: https://ceoworld.biz/2019/12/13/data-preprocessing-what-is-it-and-why-is-important/.

[45] Great Learning Team", *Understanding Data Augmentation — What is Data Augmentation how it works?* Aug. 2020. [Online]. Available: https://www.mygreatlearning.com/blog/understanding-data-augmentation/.