# ABSTRACT

Optical Character Recognition (OCR) is the process that includes extraction and re-use of the text from images and hard-copy documents. Digital Transformation of documents and making them editable is the major idea behind OCR. The Optical Character Recognition provides an easygoing approach to image processing and pattern recognition.

The subject of character recognition has been carved into existence for well over a decade with the development of Artificial Intelligence and Deep Learning. But still, even after ten years of development, there is no commercial use of Nepali Character Recognition. So, to solve this problem we have decided to develop a basic version of the Nepali Character Recognition which will be able to recognize the Nepali Consonants (क- ञ). There are number of approaches and algorithms developed with wide variety of features like word detection as well as sentence detection. With OCR, the speed of input operation in increased which is popular mostly for International Languages.

There are four algorithms used viz. **Convolutional Neural Network, Multilayer Perceptron, Logistic Regression and K-Nearest Neighbors.** Logistic Regression is implemented using One-Vs-Rest approach. The accuracy of 99% is obtained in the dataset used through Convolutional Neural Network. Thus, the system has an excellent accuracy of identifying the characters in the dataset.

***Keywords: Accuracy, One-Vs-Rest, K-Nearest Neighbors, Convolutional Neural Network, Deep Learning, Extraction.***

# TABLE OF CONTENTS


**SUPERVISOR'S RECOMMENDATION** ..... i

**CERTIFICATE OF APPROVAL** ..... ii

**ACKNOWLEDGEMENTS** ..... iii

**ABSTRACT** ..... iv

**LIST OF TABLES** ..... vii

**LIST OF FIGURES** ..... viii

**LIST OF ABBREVIATIONS** ..... ix

**CHAPTER 1 - INTRODUCTION** ..... 1

1.1 Project Introduction ..... 1

1.2 Problem Statement ..... 2

1.3 Objectives ..... 2

1.4 Scope and Limitations ..... 2

1.5 Report Organization ..... 3

**CHAPTER 2 - LITERATURE REVIEW** ..... 4

**CHAPTER 3 - REQUIREMENT ANALYSIS AND FEASIBILITY STUDY** ..... 6

3.1 Requirement Analysis ..... 6

3.1.1 Functional Requirements ..... 6

3.1.2 Non-functional Requirements ..... 9

3.1.3 Software Requirement ..... 10

3.1.4 Hardware Requirement ..... 10

3.2 Feasibility Analysis ..... 10

3.2.1 Technical Feasibility ..... 11

3.2.2 Operational Feasibility ..... 11

3.2.3 Economic Feasibility ..... 11

3.2.4 Scheduling Feasibility ..... 11

**CHAPTER 4 – SYSTEM DESIGN** ..... **13**

4.1 Data Description ..... 13

4.1.1 Data Visualization ..... 13

4.2 UI Design ..... 14

4.3 System Workflow ..... 14

4.3.1 Dataset Preparation ..... 14

4.3.2 Model training ..... 15

4.3.3 Model Tuning ..... 15

4.3.4 Model testing/ evaluation ..... 15

**CHAPTER 5 – IMPLEMENTATION** ..... **16**

5.1 Tools Used ..... 16

5.2 Algorithms Used ..... 16

5.2.1 Convolutional Neural Network ..... 16

5.2.2 Multilayer Perceptron ..... 25

5.2.3 K-Nearest Neighbors ..... 29

5.2.4 Logistic Regression ..... 32

**CHAPTER 6 – SYSTEM TESTING** ..... **37**

6.1 Unit Testing ..... 37

6.2 Integration Testing ..... 38

6.3 Result Analysis ..... 39

**CHAPTER 7 – CONCLUSION AND RECOMMENDATION** ..... **40**

7.1 Conclusion ..... 40

7.2 Future Enhancements ..... 40

**REFERENCES** ..... **41**

**APPENDIX** ..... **42**

# LIST OF TABLES

Table 3. 1: Select Algorithm ..... 8
Table 3.2: Choose Image ..... 8
Table 3.3: Predict the Character ..... 8
Table 3.4: View Results ..... 9
Table 5. 1: Table showing CNN Summary ..... 24
Table 5.2: Table showing MLP Summary ..... 28
Table 6. 1: Table Showing Unit Testing for Nepali OCR ..... 37
Table 6.2: Table Showing the accuracy of different Algorithms in the system ..... 39

# LIST OF FIGURES

Figure 3. 1: Use Case Diagram for Nepali OCR ..... 7
Figure 3.2 : Gantt Chart for Nepali OCR..... 12
Figure 4. 1: Data Visualization used in the system..... 13
Figure 4.2: System Workflow..... 14
Figure 5. 1: Architecture of CNN ..... 17
Figure 5.2: Max and Average Pooling Layers ..... 19
Figure 5.3: ReLU Activation Graph ..... 21
Figure 5.4: Cross Entropy Loss for Training and Validation set of data ..... 23
Figure 5.5: Architecture of MLP ..... 25
Figure 5.6: Curve showing validation error for different values of k ..... 31
Figure 5.7: Graph showing Sigmoid activation function..... 33
Figure 5.8: Code Snippet for One-Vs-Rest Approach ..... 34
Figure 5.9: Curve showing Sigmoid activation and Decision Boundary..... 35
Figure 5.10: Code Snippet for Decision Boundary..... 35
Figure 6.1: Integration Testing ..... 38

# CHAPTER 1 - INTRODUCTION

## 1.1 Project Introduction

Optical Character Recognition (OCR) is the process that includes extraction and re-use of the text from images and hard-copy documents. Digital Transformation of documents and making them editable is the major idea behind OCR. The Optical Character Recognition provides an easygoing approach to image processing and pattern recognition. In the modern era, the storage of printed books, magazines, documents in a digital form is increasing. This is considered as the easiest way of storing old documents and thus can be reused. In such cases, the documents may have to be modified and going through each character manually, typing them again is obviously a tedious task. Thus, there is a need of software that could recognize characters and make the document analysis work much easier.

The subject of character recognition has been carved into existence for well over a decade with the development of AI and Deep Learning. But still, even after ten years of development, there is no commercial use of Nepali Character Recognition. So, to solve this problem the basic version of the Nepali Character Recognition is developed which will recognize the Nepali Consonants (क - ज्ञ). There are number of approaches and algorithms developed with wide variety of features like word detection as well as sentence detection. With OCR, the speed of input operation in increased and is popular mostly for International Languages.

This project consists of individual characters, processed and feed to the system for training. Once the model is trained, it becomes ready to recognize the new characters, that are the handwritings of different persons other than that of training set. This system allows the user to choose character image for recognition along with the algorithms. Later, the same system can be extended where the input will be a full document rather than individual character. This project is helpful mainly for government offices in Nepal where many documents are being digitized and stored electronically.

## 1.2 Problem Statement

In the ever-changing world, there is a high demand for software systems that can recognize characters in the computer system. People nowadays need an automated system for tedious tasks like OCR. Furthermore, digitizing the documents and making them ubiquitous is what people want as they refuse to carry the paper documents with them. Also, it can be a system for the future as people are moving more and more towards the digital scripts rather than providing hard copies for documents. It provides a more secure and efficient approach of data communication.

Nepali Character Recognition is still going on and the professors from renowned Universities along with other researchers from different colleges, actively working on it. There are projects which are being carried out similarly to improve the accuracy of the recognition. The aim behind this project is to deal with a variety of handwritten characters and develop a precise way of recognition. Furthermore, the project can be extended for Document Analysis for Nepali Language.

Though the dataset for Nepali Character Recognition is available, it is not 100% accurate, the decision is to create one's own dataset and carry out the project from the beginning. As this is an academic project, wanting to learn everything from the very beginning and implement them in the project is the sole purpose. Doing this improves knowledge and will result in a practical implementation of Nepali OCR.

## 1.3 Objectives

The main objectives of the project are:

- To identify the Nepali Consonant characters from the image.
- To compare the accuracy and effectiveness of algorithms used i.e. CNN, KNN, LR and MLP.

## 1.4 Scope and Limitations

This project is built for the computer system to recognize the Nepalese Devanagari Scripts in a digitized manner. With this software the computer will be able to recognize the individual characters with up to 95% accuracy depending on the handwriting and the quality of data provided by the user to the software. It can be used on a large scale for digital recognition of individual characters.

The limitations of the system are as illustrated below:

- The system is only able to recognize the Devanagari consonants, not the vowels.
- The system is unable to recognize the characters such as (की,कु,कू,कृ) that can be used in the scripts.
- The system is not able to recognize multiple characters in a single images as the concept of image segmentation needs to be incorporated on it, which is not the part of the system built, currently.
- Due to limitation in the dataset, the precision of the system may not be as significant as it should be.

## 1.5 Report Organization

Our report is organized into 6 chapters:

**Chapter 1: Introduction**

In this section the brief introduction of our project, statement of the problem and its objectives are discussed.

**Chapter 2: Literature Review**

The previous work related to our projects were studied and are briefly summarized in this section.

**Chapter 3: Requirement Analysis and Feasibility Study**

The functional and nonfunctional requirement of the system along with different feasibility studies are discussed in this section.

**Chapter 4: System Design**

In this section we have designed the structuring system requirement like activity diagram, sequence diagram, database design etc.

**Chapter 5: Implementation and testing**

The various implementation method and tools are described in this section. This part also contains the description of various testing and results we got after performing them.

**Chapter 6: Conclusion and Recommendations**

This section contains the conclusion and recommendation based on our project.

# CHAPTER 2 - LITERATURE REVIEW

Optical Character Recognition has been a topic of interest for researchers. There are various approaches used for character recognition along with their individual accuracy, merits and demerits. Converting the handwritten or printed documents into digital format has been a problem and the involvement of human is a tedious task. This is where a good OCR system plays a significant role.

The research made by Sudan Prajapati and the team has dataset of 69 Nepali fonts. The paper implements Tesseract and Artificial Neural Network. With Tesseract, the overall accuracy of 96% was obtained in the training phase and 69% in testing phase. Similarly, with ANN, an accuracy of 98% was obtained in training and 81% in testing. This paper has good explanation of Image Acquisition, Preprocessing, Feature Extraction and Classification. [1]

The next research was carried out by Manish K. Sharma and team with the implementation of Convolutional Neural Network based OCR system for Nepali Language, a commonly spoken language in Nepal. The system has been developed in Python using Keras Library on top of Theano and NumPy. This paper implements the MaxPooling and Dropout concepts in CNN. This paper has shown a good accuracy; however, their confusion matrix shows that the two characters (ङ, ङ) were represented interchangeably. This has resulted in a high error-rate. Also, this paper implements the concept of Artificially Synthesizing the dataset. [2]

Similarly, diving deep into OCR, especially for Nepali Language, research by Nirajan Pant and Bal Krishna Bal proposed a hybrid system for printed Nepali text using the Random Forest Machine Learning technique. They used two approaches, holistic and character level recognition in this paper and obtained a recognition rate of approximately 78.87 % for character level recognition method.[3]

Moving on, research by Sailesh Acharya, et al. introduces a new public image dataset for Devanagari script. This paper explains the implementation of deep learning architecture for recognition of characters. According to this paper, Deep Convolutional Neural Network have shown superior results to traditional shallow networks in many recognition tasks. Also, a good implementation of dropout and data-set increment approach has been introduced to improve test accuracy. [4]

Similarly, research by A.K. Pant and team has implemented an offline Nepali Handwritten Character Recognition, based on the Neural Networks. A good set of spatial features are extracted from character images. Accuracy and efficiency of Multilayer Perceptron and Radial Bias Function classifiers are analyzed. The strength of this research is the efficient feature extraction and the comprehensive recognition techniques, due to which, the recognition accuracy of 94.44% is obtained for numerical dataset, 86.04% is obtained for vowel dataset and 80.25% is obtained for consonant dataset. [5]

**Use Case Diagram**

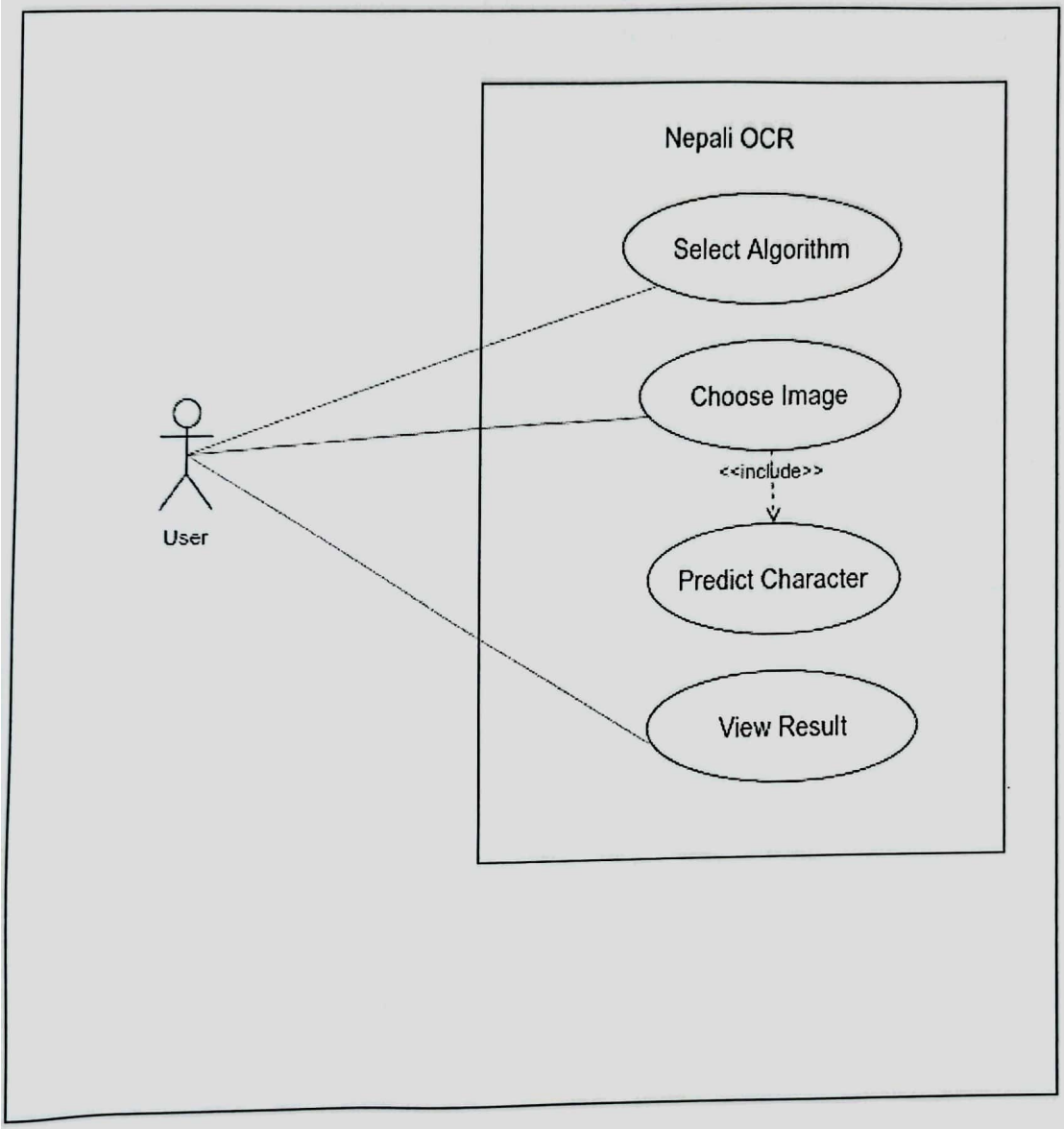



*Figure 3. 1: Use Case Diagram for Nepali OCR*

**Use-case Description**

**Table 3. 1: Select Algorithm**

| | |
|---|---|
| Use-case 1 | Select Algorithm |
| Primary Actor | User |
| Description | The user selects the desired algorithm for predicting the character. |
| Pre-Condition | - |
| Post-Condition | The trained model of selected algorithm must be loaded. |
| Failure | Unable to select the algorithm. |

**Table 3.2: Choose Image**

| | |
|---|---|
| Use-case 2 | Choose Image |
| Primary Actor | User |
| Description | The user browses the character image to be predicted from the computer. |
| Pre-Condition | The algorithm must be selected. - |
| Post-Condition | - |
| Failure | The selected image does not exist or failed to choose. |

**Table 3.3: Predict the Character**

| | |
|---|---|
| Use-case 3 | Predict the character |
| Primary Actor | User |
| Description | The model predicts the character |

| | |
|---|---|
| Pre-Condition | The model should be properly trained with the selected algorithm. |
| Post-Condition | - |
| Failure | Wrong prediction of the character. |

**Table 3.4: View Results**

| | |
|---|---|
| Use-case 4 | View results |
| Primary Actor | User |
| Description | The User can view the results along with the other visualizations. |
| Pre-Condition | The model should predict the character. |
| Post-Condition | - |
| Failure | There is error in the result and visualizations. |

### 3.1.2 Non-functional Requirements

Non-functional requirement includes the aspects of the system that are not directly connected with the specified function of the system. It mainly focuses on the reliability, performance, security and user convenience. The non-functional requirement of Nepali OCR includes the following:

- Reliability:

  Reliability for the given system refers to the accuracy of the prediction of characters present in an image. The system must generate all the possible visualizations, reports such as accuracy, probabilities, F1-score, Precision and Recall.

with proper preliminary analysis and a good research on the project and ideas. The project involves the following analysis:

### 3.2.1 Technical Feasibility

Technical feasibility focuses on technical resources available for the decided system. It involves evaluation of hardware, software, and other technical requirements of proposed system. The system **Nepali OCR** is easy to use as it is based on Python programming. To train the ML model, frameworks and Machine Learning libraries like PyTorch, NumPy, Pandas and likewise are used. Nepali Devanagari Dataset is used to train the model and the system needs the hardware as discussed above. The system is technically feasible as it does not require complex hardware and software resources.

### 3.2.2 Operational Feasibility

Operational Feasibility involves the study of examining how a project plan satisfies the requirement analysis in the requirement analysis phase of the system development. The proposed system is operationally feasible and easy to use as it has simple UI. Anyone with basic knowledge of computer can use the system as it is user-friendly.

### 3.2.3 Economic Feasibility

Economic feasibility involves a cost/ benefit analysis of project and allocating the financial resources to the system. As this is an academic project and all the libraries and resources required for the system were open-sourced, the system is economically feasible.

### 3.2.4 Scheduling Feasibility

Scheduling feasibility plays significant role in success of a project. A project is said to be failed, if it cannot be completed on time. All the team members were responsible,

## 4.2 UI Design

The UI of the system is simple and is developed using Tkinter Library, which is simple to use. The UI allows user to choose the algorithms and image for prediction. It then provides the predicted character along with the visualizations and accuracies.

## 4.3 System Workflow

System workflow involves the stepwise implementation of modules into a system. It represents the overall design of the system and provides a vivid picture of how system works. System workflow provides the blueprint that helps to understand the architecture of the system along with the data flow. Data here is the dataset that are modified along with the steps and algorithms.

The overall workflow is represented in the figure below:

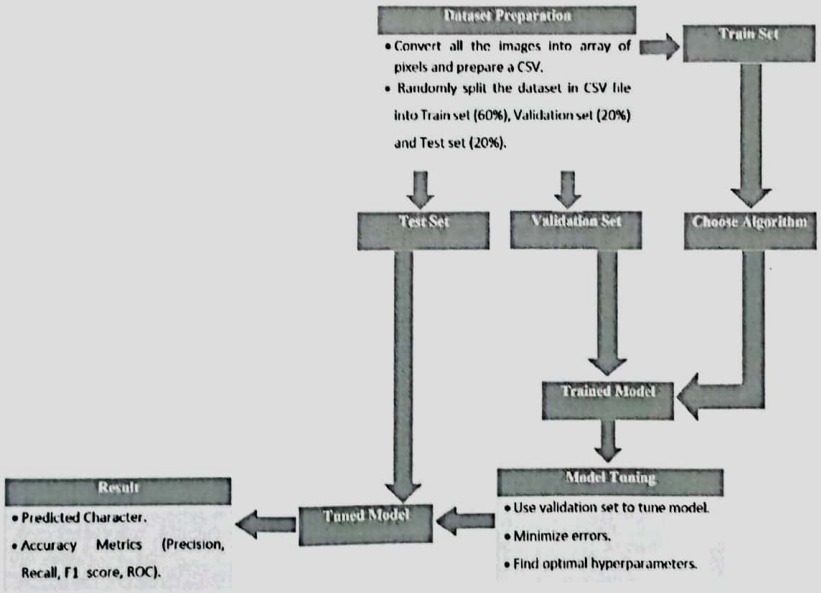



*Figure 4.2: System Workflow*

### 4.3.1 Dataset Preparation

As the dataset contains images each of 32*32 pixels, images are converted to array of pixels. These arrays of pixels are assembled into a single CSV file. The CSV file

contains 1025 column along with 1024 pixels and a column containing the actual character that the pixel contains. Once the dataset is ready, the different CSVs of training and test set are prepared for the purpose of training the model and for evaluating the trained model.

Also, the training data is further divided into train and validation set to train and evaluate the model during the training process. Here, the validation set is used to evaluate the accuracy of the model being trained during the training process.

### 4.3.2 Model training

Training a model is a major part of the Machine Learning project. It is the step where machine learns from data. In other words, it is the phase where machine identifies the patterns and discover new insights from data. During the training phase, the model is trained using algorithms viz. CNN, KNN, LR and MLP. The model is trained using the training and regularly validated using the validation set. Once, the model is trained, it is saved and later loaded for prediction.

### 4.3.3 Model Tuning

Tuning a model refers to a process in which the accuracy of model is increased using varieties of techniques like changing parameters and hyperparameters, changing the cost functions, regularizations, etc. The model tuning process are different according to the algorithms used and thus is described more in implementation phase.

### 4.3.4 Model testing/ evaluation

Evaluating the tuned model involves the process of providing the unseen data to the trained model and making the prediction. Here, the system has test set as unseen data which contains images that has character on it. The system should be able to recognize the character present in image and provide the predicted character as result.

# CHAPTER 5 – IMPLEMENTATION

The main purpose behind the implementation of this project is to identify the Nepali Characters present in the image. This project is implemented using five different algorithms and a comparison is made between them.

## 5.1 Tools Used

The programming logic for the system is built in python. The UI is designed using Tkinter Library of Python and the algorithms serves as backend of the system.

## 5.2 Algorithms Used

Machine Learning and Deep Learning are being popular nowadays. The availability of tremendous amount of data leads to the evolution of effective algorithms. To carry out the comparison, the system has been implemented using two Deep Learning Algorithms namely, CNN and MLP and two Machine Learning Algorithms namely, KNN and LR.

### 5.2.1 Convolutional Neural Network

A Convolutional Neural Network is a special kind of Fully Connected Neural Network which significantly reduces the number of parameters in each layer of deep neural network without losing the information of quality of the model. CNN is mostly used for image processing, object detection, character recognition and classification. In short, it is used for Computer Vision and related activities.

The input to a convolutional layer is a 'm x m x r' image where m is the height and width of the image and r is the number of channels, e.g. an RGB image has 'r=3'. The convolutional layer will have k filters (or kernels) of size 'n x n x q' where n is smaller than the dimension of the image and q can either be the same as the number of channels 'r' or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size 'm−n+1'. Each map is then subsampled typically with mean or max pooling over 'p x p' contiguous regions where p ranges between 2 for small images and is usually not more than 5 for larger inputs. Either before or after the sub sampling layer activation functions like Sigmoid, ReLU, tanh, etc. are applied to each feature map.

The general architecture of CNN as implemented in this system is presented below:

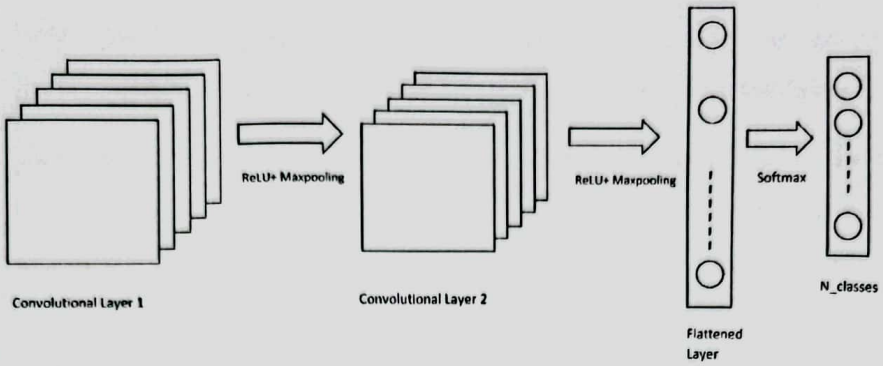



*Figure 5. 1: Architecture of CNN*

Based upon the number of input and output channels, the architecture of CNN differs. The first architecture of CNN is the **LeNet,** demonstrated by **Yann LeCun** for recognizing handwritten digits of a popular dataset known as **MNIST**. After this, there are several convolutional architectures developed such as **AlexNet, VGG-16, VGG-19,** etc.

CNN has different layers to work with the input image, the layers that has been used in this system are presented below:

- **Convolutional Layer**

  Convolutional layer involves the process of applying filters to the input images or the local connections. It computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. The convolutional layer is used to recognize the spatial patterns in image, such as lines and parts of objects. This is the layer which contains most of the user-defined parameters of the CNN architecture such as:

  - Number of Kernels
  - Size of Kernels (includes only height and width, the dimension is specified by the input image)
  - Strides
  - Padding
  - Activation Functions

The output of the convolutional layer is passed immediately through activation functions.

There are two convolutional layers used in the system. As the system is implemented using PyTorch framework, the conv2d under **torch.nn** is used.

The framework provides the abstraction and allows us to use all the features, however the mathematical relation on how to calculate the number of input and output features for a convolutional layer is the major challenge for CNN. The system has two convolutional layers implemented along with the activations and max-pooling layer.

The mathematical formulation for the 1st convolutional layer is presented below:

- **Input size - 32*32*1 (n*n*1)**
  Here the input image has the height of 32-pixel, width of 32-pixel and a single channeled, i.e. image is in greyscale mode.
- **Kernel size - 5*5 (f*f)**
  Here, the convolutional filter of size 5*5 is applied.
- **Output channel – 6**
  The output channel of the first convolutional layer is chosen as 6.

The output dimension of the first convolutional layer is given as:

$$\left\lfloor \frac{n-f}{s} + \frac{1}{1} * \frac{n-f}{s} + \frac{1}{1} \right\rfloor$$

$$= \left\lfloor \frac{32-5}{1} + \frac{1}{1} * \frac{32-5}{1} + \frac{1}{1} \right\rfloor$$

$$= 28 * 28 * 6$$

The output of convolutional layer is provided to activation function and then to max-pooling layer.

- **Pooling Layer**
  The pooling layer in CNN is used to reduce the dimensionality. Pooling layer involves the process of decreasing the number of parameters to be calculated by the network in each layer. Thus, pooling layers helps to decrease the

computational complexity of the network. Pooling layer works on the principle of sliding window. There are two common pooling layers as discussed below:

- Max-pooling Layer

  The max pooling layer, as the name suggests, stores maximum value obtained in the window when it slides over the window frame of size (f,f).

- Average-pooling Layer

  The average pooling layer, as the name suggests, stores the average value obtained in the window when it slides over the window frame of size (f,f).

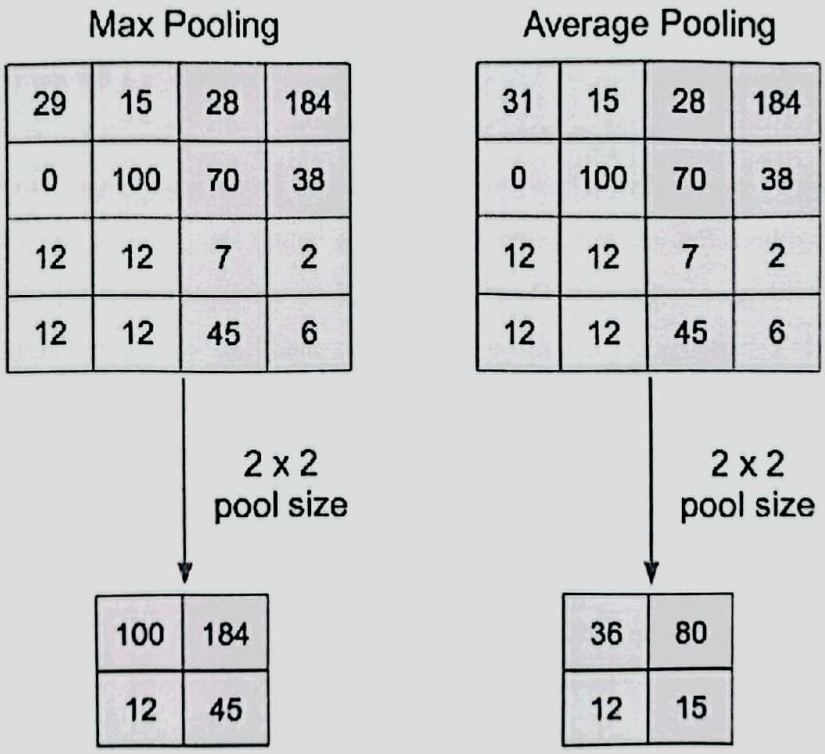



*Figure 5.2: Max and Average Pooling Layers*

As Max-pool layer works better than Average-pool layer, Max-pooling layer is used in the system. The mathematical formulation of 1st Max-pooling layer is presented below:

- **Input size – 28*28*6 (output from 1st convolutional layer)**
- **Filter size – 2*2 (f*f)**
- **Stride – 2 (s)**

The output dimension of first Max-pooling layer is given as:

$$\left\lfloor \frac{n-f}{s} + \frac{1}{1} * \frac{n-f}{s} + \frac{1}{1} \right\rfloor$$

$$= \left\lfloor \frac{28-2}{2} + \frac{1}{1} * \frac{28-2}{2} + \frac{1}{1} \right\rfloor$$

$$= 14 * 14 * 6$$

The Output of first Max-pooling layer is provided to second convolutional layer and the output of second Max-pooling layer is provided to the Linear layer which is also called Fully connected or Flattened Layer.

There are two pooling layers used in the system, and the **nn.MaxPool2d** module of PyTorch framework is used to create pooling layers.

- **Fully Connected Layer**

  The fully connected layer has full connections of neurons to all activations in the previous layer, as seen in regular Neural Networks architecture. Their activations can hence be computed with a matrix multiplication followed by a bias offset. The main objective of this layer in CNN is to take the result of convolutional or the pooling layer, converts them into a feature vector also known as the flattening process. The flattened vector is then multiplied by weights and after adding bias, the output is passed through the activation function and to other layers in the network.

  The second Fully Connected Layer in the system is followed by a softmax activation, which finally provides the probabilities of the predicted character. According to the system architecture, the output from second Max-pooling layer is then provided as an input to the Fully Connected Layer.

  There are two Fully Connected layers used in the system, and the **nn.Linear** module of PyTorch framework is used to create Fully Connected layers.

- **Activation Layers/ Functions**

  Activation functions are the mathematical equations that determines the output of a neural network. They decide whether a neuron should be activated or not by calculating weighted sum and bias. Most of the activation functions transform the input to output along with the addition of non-linearity. The

activation function used in the system is **ReLU.** It is defined under **nn.Relu()** in PyTorch.

The ReLU activation function provides a simple non-linear transformation. The performance of this function is good on predictive models. The mathematical concept behind this is also simple and for any given input **x**, the function is defined as the maximum of 0 and the provided input.

$$ReLU(x) = max(0, x)$$

This shows that ReLU activation function retains only the positive inputs and discards all the negative inputs which decreases the complexity as the negative weighted neurons are deactivated. ReLU allows the network to converge quickly. There are varieties of ReLU activation such as **Leaky ReLU, Parametric ReLU,** however the system implements the very basic one discussed above.

The following figure illustrates the ReLU activation function:

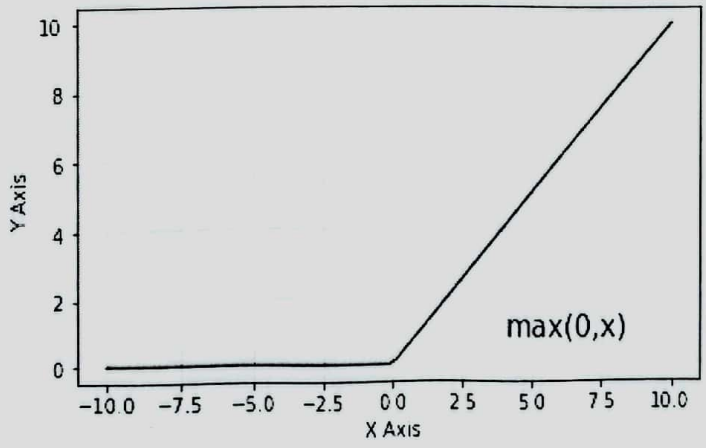



*Figure 5.3: ReLU Activation Graph*

- **Softmax Layer**

  Softmax layer consists of a Softmax activation function applied to the output of previous layer. Softmax activation function is a great choice for multiclass classification where the output should be probabilities of the classified label. Softmax activation is generally used in the output layer.

The mathematical concept behind Softmax for an input **x**, is given as:

$$\text{Softmax}(X_i) = \frac{e^{X_i}}{\sum_{j=1}^{k} e^{X_j}} \text{ for i = 1 to k}$$

The **nn.LogSoftmax()** module of PyTorch is used to implement the Softmax in output layer. There are 37 nodes in this layer, as there are 36 characters to predict and the additional node for the characters other than Nepali consonant characters.

- **Cost Function**

Cost function is used to measure the performance of the Machine Learning model in given dataset. It is used to quantify the errors between the actual and the predicted value by the model. Based on the cost function, concepts like gradient descent, stochastic gradient descent, mini-batch gradient descent, dropout, etc. are developed as an optimization technique.

There are varieties of cost functions available like MSE, RMSE, Cross Entropy, etc. Based upon the type of Machine Learning Model, the cost function are selected. As for example, if we are in regression task, the MSE would be the best choice. Here, the system implements Cross Entropy loss as the cost function. The **nn.CrossEntropyLoss()** of PyTorch is incorporated in the system.

The Cross Entropy Loss is essential when there is a multiclass classification problem with **n** classes. As the system deals with multiclass classification, the use of this cost function is best for the model. The lowest possible value of the cost function is considered as the best model, as the difference between actual and predicted value is low.

The figure below represents the Cross Entropy Loss, implemented in the system for training and validation set of data.

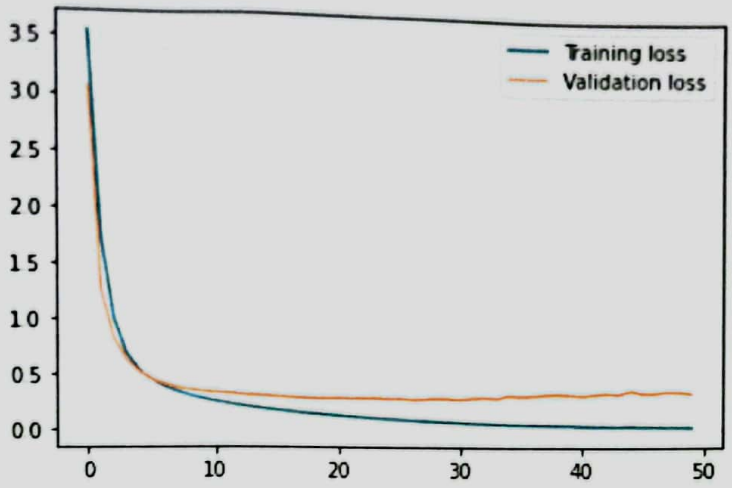



*Figure 5.4: Cross Entropy Loss for Training and Validation set of data*

- **Optimizer (Gradient based Optimization)**

  Model optimization is used in Machine Learning and Deep Learning in order to improve the accuracy, efficiency of the model. Optimization is basically a process to select the best hyperparameter for the model. These parameters cannot be learned by the model. Hyperparameters includes parameters like number of hidden layers in a network, learning rate, activation function, etc. Finding the optimal hyperparameter is a challenging task.

  The system here implements Gradient Based Optimization technique. It computes the gradient with respect to the hyperparameters and optimize them using gradient descent algorithm. Among the gradient based approach, the system makes use of Stochastic Gradient Descent which picks a data randomly in each iteration from dataset and calculates the gradient. This reduces the complexity of the algorithm and helps in quicker convergence.

  The stochastic gradient descent is implemented using **torch.optim.SGD()** module of PyTorch.

- **CNN Architecture Summary**

The table below represents the overall architecture of CNN used in the system. All the layers and filter sizes are calculated, and a CNN architecture is designed based upon the input image.

**Table 5. 1: Table showing CNN Summary**

| Layers | Activation Shape | Size |
|---|---|---|
| Input | (32,32,1) | 1024 |
| Convolutional layer 1 (f = 5, s = 1) | (28,28,6) | 4704 |
| Max Pool layer 1 (f = 2, s = 2) | (14,14,6) | 1176 |
| Convolutional layer 2 (f = 5, s = 1) | (10,10,12) | 1200 |
| Max Pool layer 2 (f = 2, s = 2) | (5,5,12) | 300 |
| Fully Connecter layer 1 | 300 | 300 |
| Fully Connecter layer 2 | 120 | 120 |
| Softmax Layer | 37 | 37 |

### 5.2.2 Multilayer Perceptron

A Multilayer Perceptron is a class of ANN, that consists of several perceptron stacked in several layers to solve the complex problems, such as multiclass classification. MLP is used to solve the problems with large datasets and multiple variables, that are totally out of human grasp. MLP is also known as Feed Forward Neural Network, which makes a clear understanding that input is always passed to forward or the preceding layers of the network.

An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

The input to MLP is the flattened vector, provided to the input layer, the input is then associated with the corresponding weights and bias. The output of the first layer is then passed to second layer through activation function like ReLU, Sigmoid, tanh, etc. The layers that are connected to input layers are the hidden layers which is one of the hyperparameter in MLP. ReLU activation is the most used activation function in hidden layers. Similarly, the output from the final hidden layer is passed to the output layer which is then passed through Softmax activation function for multiclass classification problem.

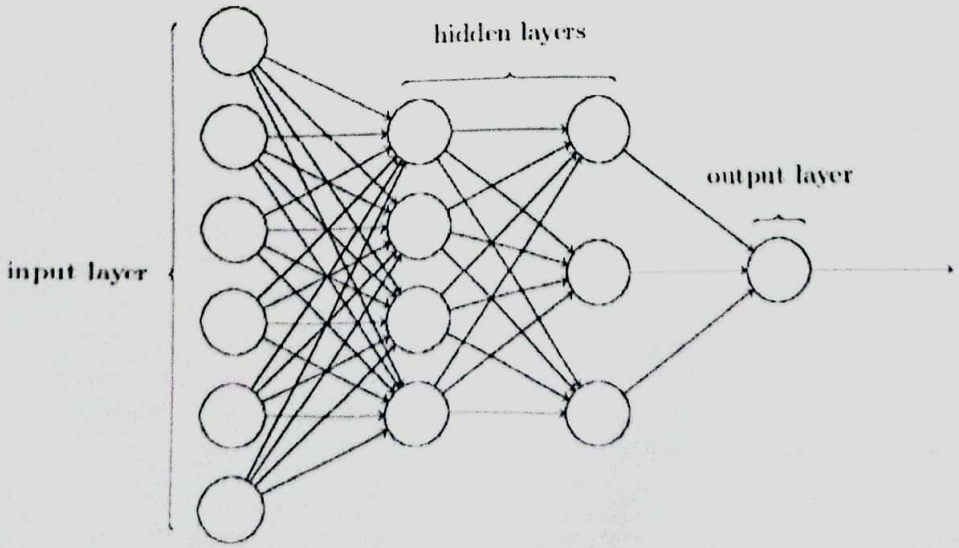



*Figure 5.5: Architecture of MLP*

The overall working of the MLP along with underlying mathematical concepts are illustrated below:

- **Initialization of weights and bias**

  Weights, in simple meaning are the coefficients of the equation. Since, every layer in Neural Network works as a function, it has the role to solve the equation with these coefficients. Bias is a constant value that is added along with the product of weights and input. Bias is used to shift the result of activation function towards negative or the positive side.

  Before starting the training process, the weights and bias are initialized to small random values. Though there are different weight initialization methods like Xavier Initialization, He initialization, Zero initialization and Random initialization, the random method is the preferred one.

- **Training the Network**

  Training the network involves the process in which machine learns the pattern of data along with the output label in the case of supervised learning. The system, being a supervised approach, training the network means to teach the machine to correctly identify the pixel value with correct label. Incase of MLP, the training of network involves two phases as illustrated below for each input vector:

  - **Forward Phase**

    In this step of training the model, we just pass the input to model and multiply with weights and add bias at every layer and find the calculated output of the model. The weights and bias of different layers are different. The major tasks in forward phases are illustrated stepwise as:

    - Compute the activation of each neuron j in hidden layer(s) using the following mathematical relation:

      $$Z[l] = W[l]A[l-1] + b[l]$$

      $$A[l] = g[l](A[l])$$

      where,

      l - layer,

Z[l] - output or the layer l,

W[l] - weight of layer l,

b[l] - bias of layer l,

A[l] - activation function of layer l,

A[l-1] - activation function of layer l-1 i.e. previous layer,

g[l] - activation function used in layer l.

- Work through the network until the output layer is reached, the output layer has different activation function that may be Sigmoid for binary classification and Softmax for multiclass classification.

- **Cost Function Calculation Phase**

  When we pass the data instance(or one example) we will get some output from the model that is called Predicted output(y_pred) and we have the label with the data that is real output or expected output(y_true). Based upon these both we calculate the loss that we must backpropagate (using Backpropagation algorithm). There are various Loss Function that we use based on our output and requirement.

  The system however implements **nn.NLLLosss()** module of PyTorch, which is the negative log likelihood loss, useful to train a classification model with **n** number of classes. Once we calculate the loss, we move into the next phase, known as backward phase.

- **Backward Phase**

  After calculating the loss, we backpropagate the loss and updates the weights of the model by using gradient. This is the main step in the training of the model. In this step, we adjust the weights according to the gradient flow in that direction. The main mathematical concept to update the weight is represented below:

$Weight_{new} = Weight_{old} - learning\_rate * gradient$

The specific gradients of the layer are used in the above relation and weights are updated to minimize the loss.

Here, in the system **loss.backward()** propagates the loss in backward direction and **optimizer.step()** applies the SGD optimizer and then forwards the input to next phase of training.

- **MLP Architecture Summary**

The table below represents the overall architecture of MLP used in the system.

**Table 5.2: Table showing MLP Summary**

| Name of Layer | Size of layer | Activation Function |
|---|---|---|
| Input Layer | 1024 | - |
| Hidden Layer 1 | 256 | ReLU |
| Hidden Layer 2 | 128 | ReLU |
| Hidden Layer 3 | 64 | ReLU |
| Output Layer | 37 | Softmax |

### 5.2.3 K-Nearest Neighbors

The K-Nearest Neighbors is a simple and robust supervised algorithm that is often used as a benchmark for more complex classifiers such as ANN and SVM. KNN can be used both for classification and regression, however, it is mainly used for classification. KNN has two unique properties that distinguish it from other Machine Learning Algorithms.

- **Non-parametric Algorithm**

  KNN is a non-parametric algorithm as it makes no explicit assumptions about the underlying data. It does not assume any functions and tries to learn functional form from the training data.

- **Lazy Learning Algorithm**

  KNN does not have specialized training phase and uses all data during classification. KNN doesn't explicitly learn a model, it rather chooses to memorize the training instances which are subsequently used as **knowledge** for the prediction phase.

The KNN algorithm uses **feature similarity** to predict the values of new data points, where new data points are assigned a value based on how closely it resembles the data point in the training phase. During the prediction phase, it searches through the entire dataset for k-most similar instances and the data with the most similar instance is finally returned as the prediction. The major hyperparameter in this algorithm is the value of **k** which greatly influences the classification results.

The implementation of KNN includes the following steps:

- Load the training as well as test data.
- Choose the value of **k** i.e. the nearest data points (odd value is mostly preferable for k)
- For each point in the test data, the following operations are performed
  - Calculate the distance between the test instance and each row of the train set.
  - Based on the distance, sort them in ascending order.
  - Keep the distance of k smallest ones.

- Get the label for target variable of training instance with the smallest distance.

- The label with the majority is selected as the predicted label.

For calculating the distance between the data points, there are several methods, as discussed below:

- **Euclidean Distance**

  Euclidean distance is calculated as the square root of the sum of squared difference between a new point ($x_i$) and an existing point ($y_i$).

  Mathematically,

$$D = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

- **Manhattan Distance**

  Manhattan distance is calculated as the sum of absolute value between a new point ($x_i$) and an existing point ($y_i$).

  Mathematically,

$$D = \sum_{i=1}^{k}|x_i - y_i|$$

- **Hamming Distance**

  Hamming distance is used for categorical variable. If the value of x and y are same, the distance D will be zero, otherwise D =1.

  Mathematically,

$$D = \sum_{i=1}^{k}|x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

The system however implements Euclidean distance for calculating the distance, as it is simple and the most common one. Also, k = 5 is used as the hyperparameter in the system.

There are various methods for choosing the value of k, one of the famous techniques is using the concept of elbow curve. The k value with low validation error is chosen as the best value of k.

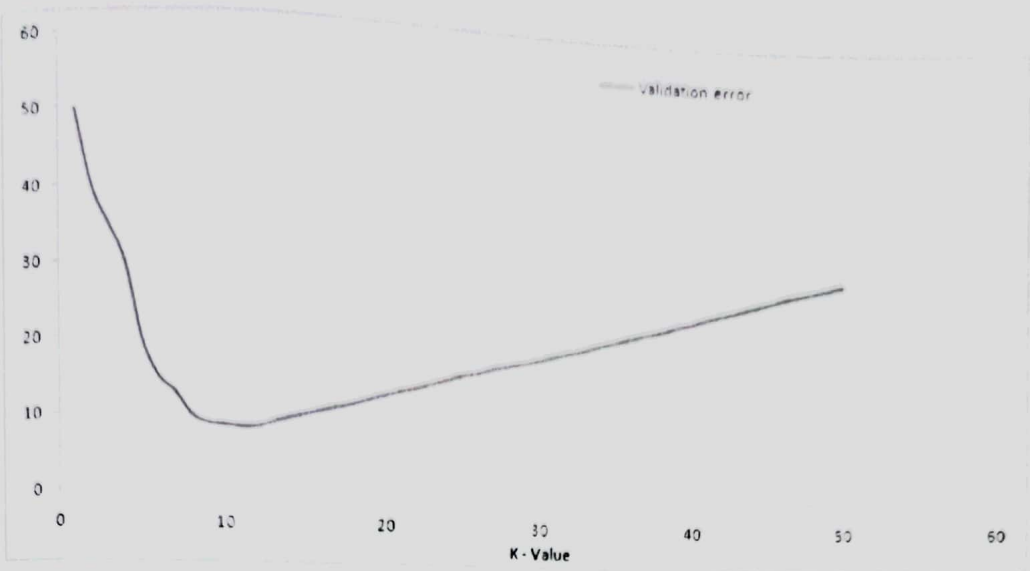



*Figure 5.6: Curve showing validation error for different values of k*

### 5.2.4 Logistic Regression

Logistic Regression, a popular machine learning classification algorithm is used to estimate the probability that an instance belongs to a particular task. Logistic regression model computes a weighted sum of the input features along with the bias added with the result. The sum is then passed through the sigmoid function to achieve the probability. The mathematical model of logistic regression is presented below:

$$\hat{y} = W.X^T + b$$

Where,

$\hat{y}$ – Predicted label by algorithm

W – Weight matrix

$X^T$ – Transpose of the input features

b – Bias term

The predicted value $\hat{y}$ is then passed to the sigmoid activation function to obtain the probability.

The Sigmoid activation function maps the input values to 0 or 1. The sigmoid activation function is defined mathematically as:

$$S(\hat{y}) = \frac{1}{1+e^{\hat{y}}}$$

Where,

$S(\hat{y})$ – output of Sigmoid function, output between 0 and 1 (probability)

$\hat{y}$ – input to the sigmoid function

e – base of natural log

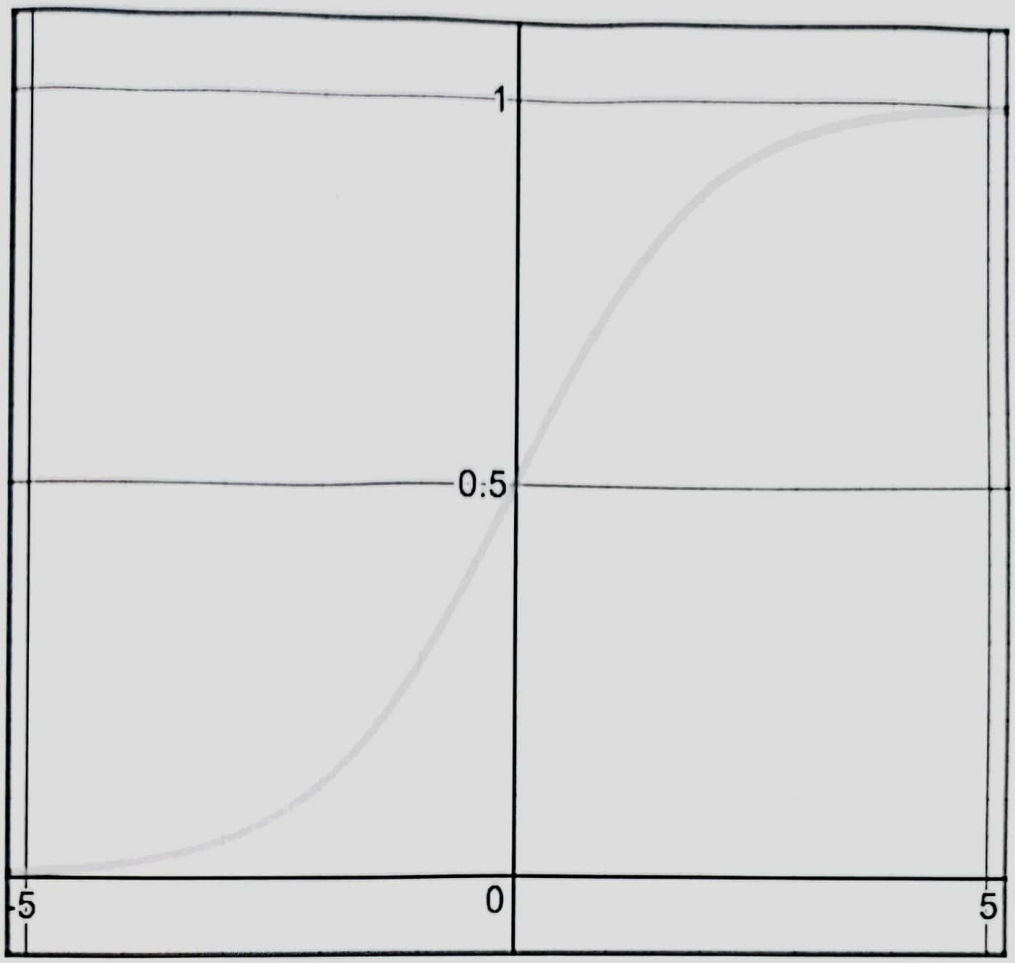



*Figure 5.7: Graph showing Sigmoid activation function*

The above concepts are for the binary classification i.e. used only when there are two classes. The system however needs to predict the total of 36 classes, same concept can be extended to the Multiclass classification. The multiclass or multinomial classification is a classification method that generalizes logistic regression to multiclass problems, i.e. with more than two possible discrete outcomes. To implement the multiclass classification, there are two approaches as illustrated below:

- **One-Vs-Rest Approach**

  One-Vs-Rest is a heuristic method for using a binary classifier for multiclass classification. It involves the process of splitting the classification problem into multiple binary classification where datasets are passed to the model and the correct label we want is labelled as '1' and other data passed are labelled as '0'. Since the labelling now is a binary one, the same approach of binary

classification is used. For example, if the problem is of classifying whether an image contains क, ख, ग, the following binary classifiers are prepared:

- Classifier 1: क vs rest
- Classifier 2: ख vs rest
- Classifier 3: ग vs rest

- **One-Vs-One Approach**

  One-Vs-One is a heuristic method for using a binary classifier for multiclass classification which involves the process of splitting the classification problem into multiple binary classification and the dataset is split into one dataset for each class versus every other class. For example, if the problem is of classifying whether an image contains क, ख, ग, the following binary classifiers are prepared:

  - Classifier 1: क vs ख
  - Classifier 2: क vs ग
  - Classifier 3: ख vs ग

The system implements One-Vs-Rest approach and the code that implements this approach is shown below:

```
train_df = shuffle(pd.read_csv(path))
pos_data = train_df[train_df.Label == label].copy()
neg_data = train_df[train_df.Label != label].copy()

pos_data.Label = 1
neg_data.Label = 0

train_df = pd.concat([pos_data, neg_data]) # new dataframe that has 1 on the desired data passed and 0 as label
                                           #on rest of the data. So, this resembles one vs rest approach

train_df = train_df.sample(frac=1).reset_index(drop=True)
```

*Figure 5.8: Code Snippet for One-Vs-Rest Approach*

The components used in the Logistic Regression are discussed below:

- **Decision Boundary**

  The sigmoid activation function returns a probability of 0 or 1. In order to map this to a discrete class, we select a threshold value, above which we classify values into class 1 and below which we classify the values into class 2.

  In the system, the decision threshold of 0.5 is used whose graph is presented below:

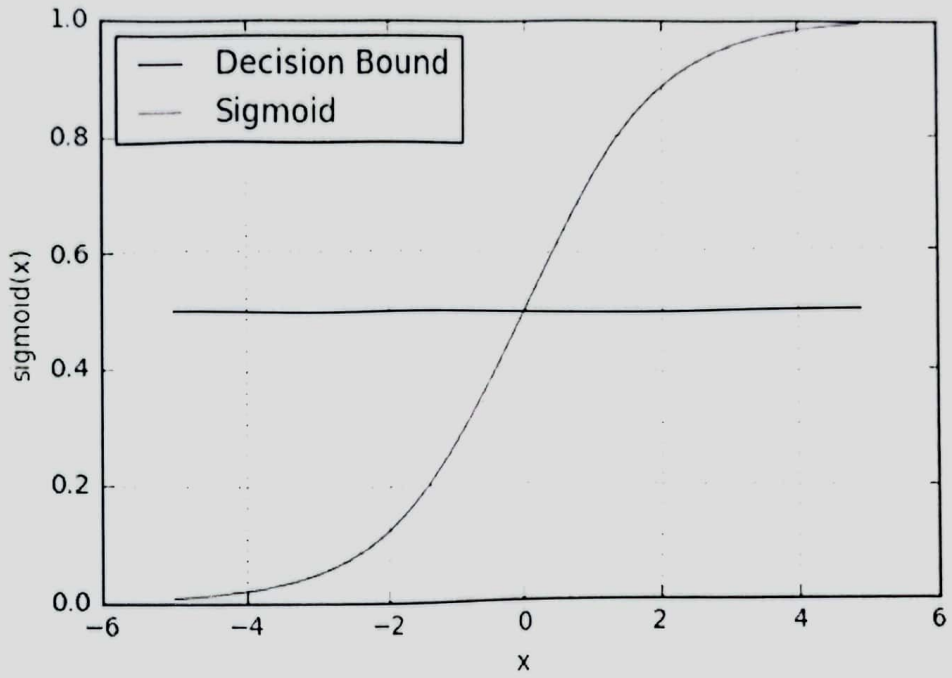



*Figure 5.9: Curve showing Sigmoid activation and Decision Boundary*

```
def decision_boundry(self,y_pred):

    y = y_pred.copy()
    y[y >= 0.5] = 1
    y[y < 0.5] = 0

    return y
```

*Figure 5.10: Code Snippet for Decision Boundary*

- **Cost Function**

The system implements Cross-Entropy loss, which is also known as Log Loss. The mathematical representation of the cost function is:

$$\text{Loss} = -\frac{1}{m}\sum_{i=1}^{m}\left[y^i \log\hat{y}^i + (1-y^i)\log(1-\hat{y}^i)\right]$$

The above loss can be divided into the following separate cost function:

$$\text{Loss1} = -\, y^i \log\hat{y}^i \quad \text{if y} = 1$$

$$\text{Loss2} = -\,(1-y^i)\log(1-\hat{y}^i) \quad \text{if y} = 0$$

- **Gradient Descent**

Gradient Descent is used to optimize the model by decreasing the cost function. Cost function is the error or difference between actual value and the predicted value by the algorithm. The gradient of above cost function with respect to the weight W is given as:

$$\frac{\Delta Loss}{\Delta W} = \frac{1}{m}\mathrm{X}^{\mathrm{T}}(\hat{y}-y)$$

In each iteration, the weight is updated using the above gradient as:

$$\text{Weight}_{\text{new}} = \text{Weight}_{\text{old}} - \text{learning\_rate} * \text{gradient}$$

Here, the learning rate is the hyperparameter which needs to be adjusted in order to achieve the greater accuracy.

# CHAPTER 6 – SYSTEM TESTING

System testing, especially in Machine Learning model is all about validating the dimensions of the dataset or the data passed in every module. Most of the testing rely on the manual inspection of dimension. The system was tested using different test dataset and the output was compared.

## 6.1 Unit Testing

Unit testing is the level of software testing of the project's system in which the smallest testable parts of a system called unit is individually tested. Unit testing concentrates on each unit of the system as implemented in the source code. The main purpose of unit testing is to validate each unit of the software to perform as designed. The system has many smaller units like data preparation unit, data loader units, data visualization and so on. The unit test carried out on these units are discussed below:

**Table 6. 1: Table Showing Unit Testing for Nepali OCR**

| Test Case No | Description | Test Input Data | Expected Result | Obtained Result | Status |
|---|---|---|---|---|---|
| 1. | Test Data Preparation | Images of 32*32 pixels | Proper Conversion of Image to pixels and then to a CSV file | A CSV file with the pixel value of the test set images. | Successful |
| 2. | Random CNN Model was introduced. | Training set data | The random model should not work as the dimension of data differs. | Error due to data dimension mismatch. | Successful |

| | | | | | |
|---|---|---|---|---|---|
| 3. | Predict individual Image | Image without converting to csv | The correct classification of image. | The image was classified correctly. | Successful |
| 4. | Predict Image that is out of the dataset distribution | Image from different distribution | Misclassification of the image | The image was misclassified | Successful |
| 5. | Choose the Image with dimension other than 1*32*32. | Image with dimension 3*32*32 | Error loading the image. | Image could not be loaded as the system expects 1*32*32 dimension. | Successful |

## 6.2 Integration Testing

After all the units and modules are tested, the system was constructed by integrating all the algorithm along with respective modules. Then the whole system was checked by selecting the algorithms to determine if the system could correctly classify the character present in the image. The following figure illustrates the integration testing along with the correct and incorrect prediction of characters with respect to the true character.


*Figure 6.1: Integration Testing*

The above figure represents that the green-colored characters are correctly classified, and the red-colored were misclassified. The system works well to fulfill the major objective defined.

## 6.3 Result Analysis

After integrating all the algorithms, the performance of the system was tested with different test images and the following result was obtained.

**Table 6.2: Table Showing the accuracy of different Algorithms in the system**

| S.N. | Algorithm | No. of Test Images | Number of Correctly Classified Images | Number of Misclassified Images | Accuracy |
|---|---|---|---|---|---|
| 1. | Logistic Regression | 10800 | 8702 | 2098 | 80.57% |
| 2. | K- Nearest Neighbors | 10800 | 9507 | 1293 | 88.027% |
| 3. | Multilayer Perceptron | 10800 | 9923 | 877 | 91.87% |
| 4. | Convolutional Neural Network | 10800 | 10104 | 696 | 93.5556% |

So, from the test of the system, CNN has the maximum accuracy of **93.5556%** on the dataset used whereas Logistic Regression shows the lowest accuracy of **80.57%.**

# CHAPTER 7 – CONCLUSION AND RECOMMENDATION

## 7.1 Conclusion

Nepali OCR was developed using the four algorithms, two from Machine Learning and the two from Deep Learning. The system developed thus can take an image from user and predict the character in the image. The user was able to select the algorithms and see the results based upon the algorithm used. The underlying concepts of the algorithms were properly identified and how these algorithms were incorporated in PyTorch framework was identified. Based upon the algorithm, different approaches were followed for different modules and units like data loading, visualization, character mapping, etc. Thus, the Nepali OCR was developed with the overall accuracy of 94%.

## 7.2 Future Enhancements

Till now, our project can recognize the characters with over 90% accuracy, but it cannot be deemed to be the final product. With more research and development, much enhancement can be done in this system. The system is limited to recognize only the Nepali consonants, but same algorithms can be used to recognize the vowels and the special characters used in the Devanagari scripts, using the concept of Transfer Learning. Also, the same concept can be applied to recognize English alphabets and numbers. The dataset for the system was limited, thus, the reliability of the system can be enhanced by collecting more datasets ang using techniques like data augmentation.

The most important enhancement that should be done to the system so that it can be used in a commercial scale is that the system reads the individual characters of the Devanagari script but not the word as a whole which is not considered to be an efficient manner of character reading. The system should be further enhanced to read the word as a whole and it can also be used for word correction to filter any incorrect words that have been written and correct them accordingly. This concept can be fulfilled using image processing techniques like object localization, image segmentation, etc.

# REFERENCES


[1] Prajapati, Sudan, Shashidhar Ram Joshi, Aman Maharjan, and Bikash Balami. "Evaluating Performance of Nepali Script OCR Using Tesseract and Artificial Neural Network." 2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS), 10 2018. doi:10.1109/cccs.2018.8586808.

[2] Sharma, Manish K., and Bidhan Bhattarai. "Optical Character Recognition System for Nepali Language Using ConvNet." Proceedings of the 9th International Conference on Machine Learning and Computing - ICMLC 2017, 2017. doi:10.1145/3055635.3056635.

[3] Pant, Nirajan, and Bal Krishna Bal. "Improving Nepali OCR Performance by Using Hybrid Recognition Approaches." 2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA), 07 2016. doi:10.1109/iisa.2016.7785384.

[4] S. Acharya, A. K. Pant and P. K. Gyawali, "Deep learning based large scale handwritten Devanagari character recognition," 2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA), Kathmandu, 2015, pp. 1-6.

[5] A.K. Pant, S. P. Panday and S. R. Joshi, "Off-line Nepali handwritten character recognition using Multilayer Perceptron and Radial Basis Function neural networks," 2012 Third Asian Himalayas International Conference on Internet, Kathmandu, 2012, pp. 1-5.

# APPENDIX

## User Interface of the system

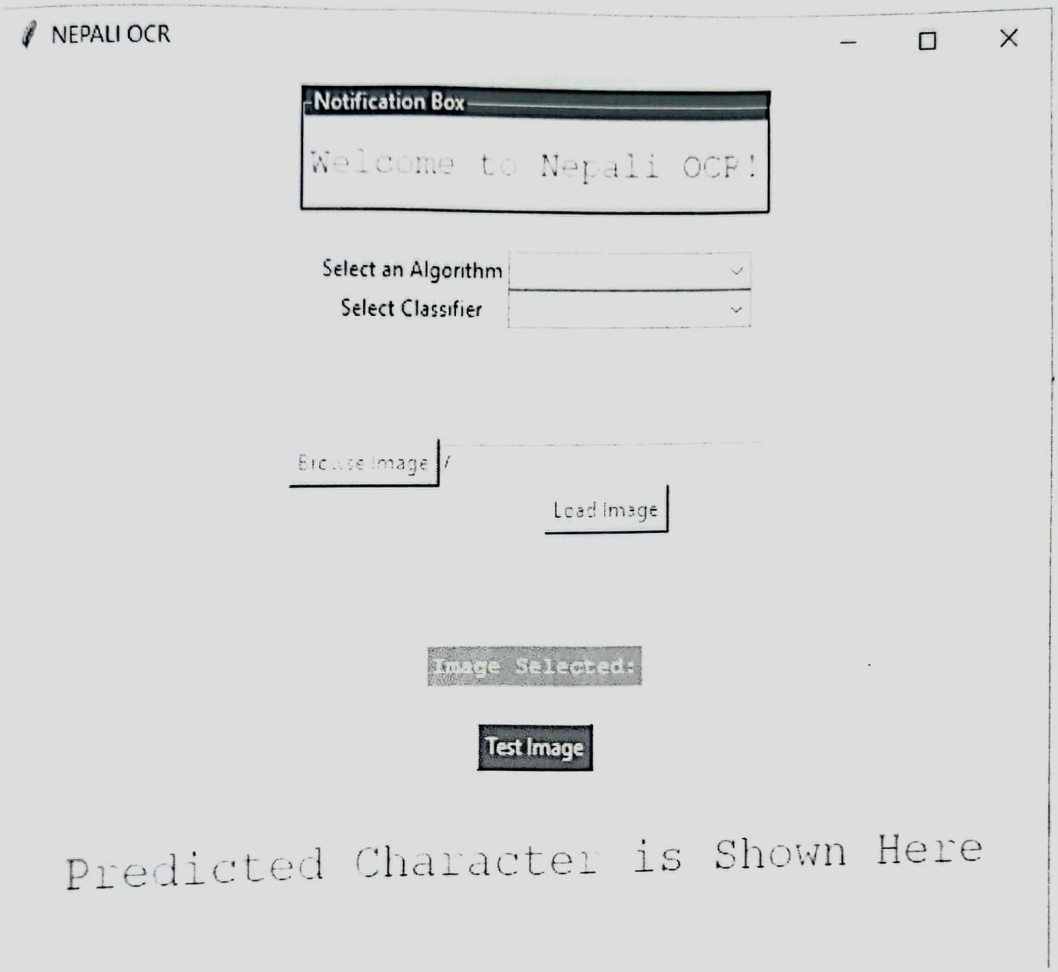



**Figure A: UI of the system before the prediction**

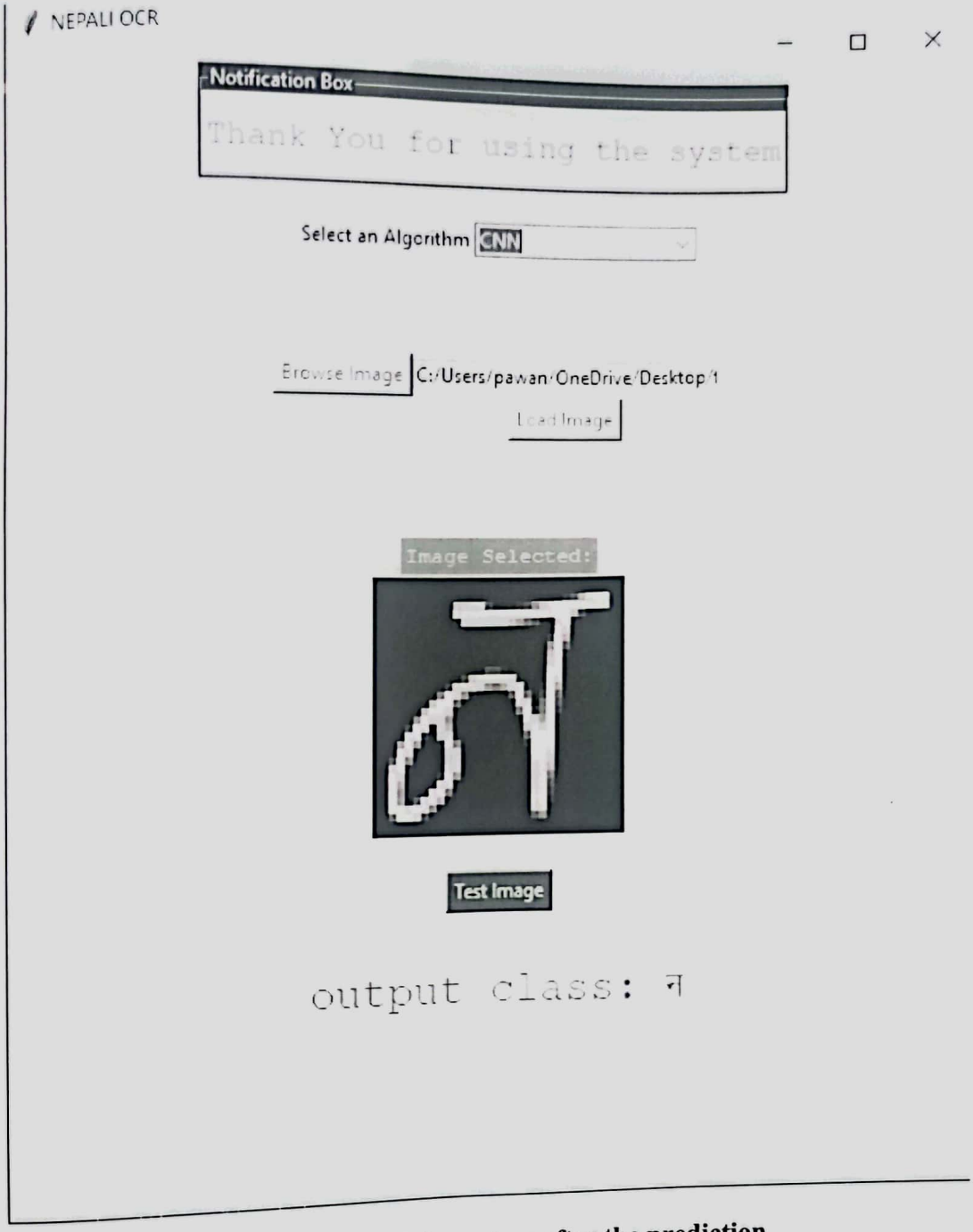
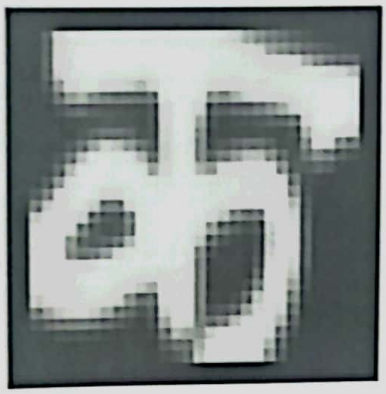



**Figure B: UI of the system after the prediction**

## Probability Visualization

The actual character is ञ
Predicted character = ञ
The probability of predicted character is 1.0



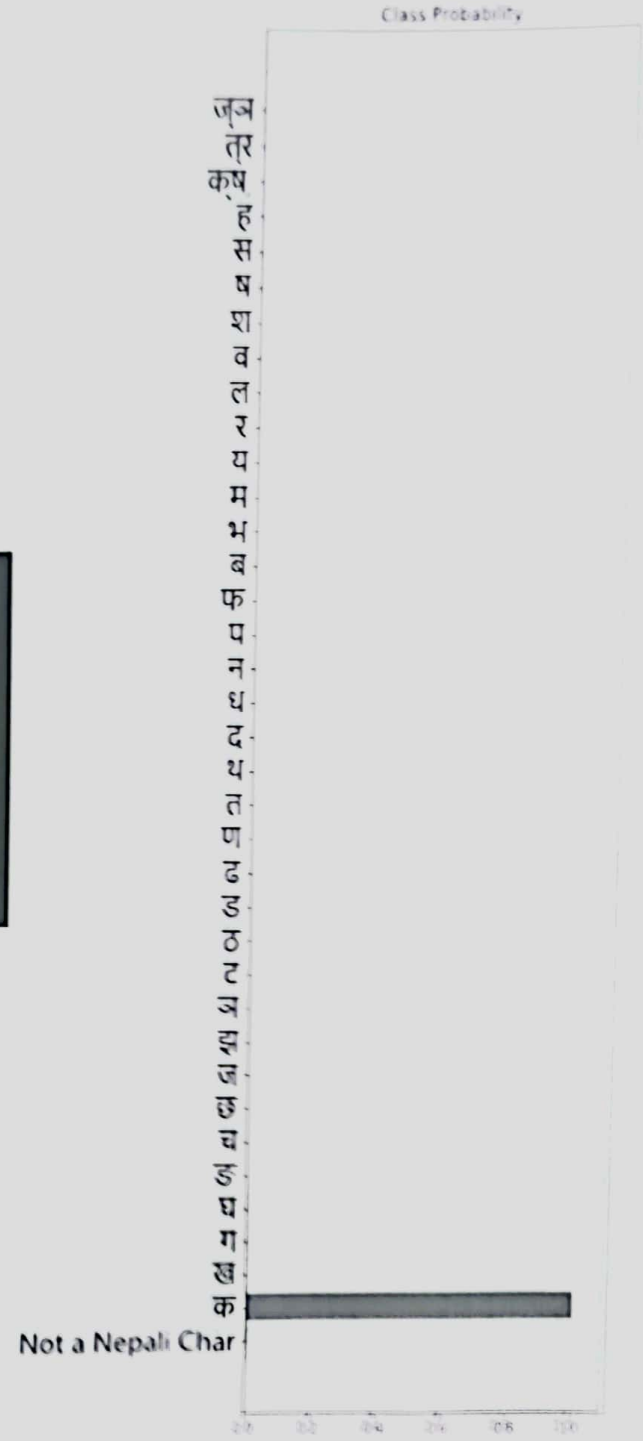



**Figure C: Probability of the predicted image along with the character**

## Image to CSV

```
def read_images_train(train_images_dir):
    pixels = np.array(['pixel_{:04d}'.format(x) for x in range(1024)])
    #images is of 32*32 pixels which results total of 1024 pixel

    flag = True

    for characters in sorted(os.listdir(train_images_dir)):
        char_dir = os.path.join(train_images_dir,characters)

        images_df = pd.DataFrame(columns=pixels)

        for img_file in sorted(os.listdir(char_dir)):
            image = Image.open(os.path.join(char_dir,img_file))
            images = image.resize((32,32))
            image = pd.Series((images).flatten(),index = pixels)
            images_df = images_df.append(image.T,ignore_index=True)

        images_df = images_df.astype(np.uint8)
        images_df['Label'] = characters
        #labelling is done according to the dir name where image resides

        images_df.to_csv('train_data.csv',index=False,mode='a',header=True)

        flag = False
```

**Figure D: Code Snippet showing Data Preparation**