Tribhuvan University
Birendra Multiple Campus

Final Year Project Report
On
Devanagari Handwritten Character Recognition
Course Code: CSC-404

A final year project submitted to the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Information Technology awarded by Tribhuvan University.

Submitted by
Abhishek Gupta (T.U Exam Roll No. 8328/072)
Arjun Gautam (T.U Exam Roll No. 8332/072)
Kamal Gautam (T.U Exam Roll No. 8348/072)
Ramkrishna Acharya (T.U Exam Roll No. 8367/072)

Submitted to
Department of Computer Science and Information Technology
Birendra Multiple Campus
Institute of Science and Technology
Tribhuvan University
August 30, 2019

Tribhuvan University

Institute of Science and Technology

Birendra Multiple Campus

Department of CSIT

Date: August 30, 2019

Recommendation Letter of Supervisor

I hereby recommend that the project prepared under my supervision by **Abhishek Gupta, Arjun Gautam, Kamal Gautam and Ramkrishna Acharya** entitled **"Devanagari Handwritten Character Recognition",** be accepted as fulfilling in part requirements for the degree of Bachelor of Science in Computer Science and Information Technology. In my best knowledge this is an original work in computer science.

……..

Er. Binod Sharma
Lecturer, Department of CSIT
Birendra Multiple Campus

Tribhuvan University

Institute of Science and Technology

Birendra Multiple Campus

Department of CSIT

Date: August 30, 2019

Letter of Approval

This is to certify that this project prepared by **Abhishek Gupta, Arjun Gautam, Kamal Gautam and Ramkrishna Acharya** entitled "**Devanagari Handwritten Character Recognition**" in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

……

Mr.

External Examiner

Tribhuvan University, Kirtipur

Tribhuvan University

Institute of Science and Technology

Birendra Multiple Campus

Department of CSIT

Date: August 30, 2019

Letter of Approval

This is to certify that this project prepared by **Abhishek Gupta, Arjun Gautam, Kamal Gautam and Ramkrishna Acharya** entitled "**Devanagari Handwritten Character Recognition**" in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

……

Er. Binod Sharma

Program Co-ordinator, Department of CSIT

Birendra Multiple Campus, Chitwan

ACKNOWLEDGEMENT

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide **Er. Binod Sharma** for his valuable guidance, and encouragement in making this project possible. His constructive suggestions regarding this project work and consistent support are sincerely acknowledged.

Finally, we would like to express our sincere thanks to all our friends and all those who supported us directly or indirectly during this project work and make this project a successful one.

Abhishek Gupta (T.U Exam Roll No. 8328/072)
Arjun Gautam (T.U Exam Roll No. 8332/072)
Kamal Gautam (T.U Exam Roll No. 8348/072)
Ramkrishna Acharya (T.U Exam Roll No. 8367/072)

ABSTRACT

Devanagari Handwritten Character Recogniton is the system which provides user the Optical Character Recognition functions on Devanagari handwriting. This Devanagari Handwritten character recognition system has mainly five stages: Preprocessing, Segmentation, Feature Extraction, Prediction and Postprocessing. It uses the Convolutional Neural Network to train the model and uses Image Processing techniques to recognize user image via camera or local storage. The system recommends the external camera with good image quality to get best results. Devanagari Handwritten Character Recognition system was carried out by the team of members, who were devoted to their respective field to develop the best possible. This system uses python programming language and different hardware platforms, software platforms to make the system as a whole. During the development phase different modules were made separately by different members and were integrated at last. Along with it testing were done. After the system has been developed it is found that different algorithms gave different level of correctness.

Keywords: Handwritten Nepali alphabets; Handwritten Nepali digits; Convolutional Neural Network; Segmentation; Templates matching

Table of Contents

List of Figures

List of Abbreviations

DCR	Devanagari Character Recognition
NN	Neural Network
CNN	Convolutional Neural Network
DFD	Data Flow Diagram
ERD	Entity Relationship Diagram
FFNN	Feed Forward Neural Network
RELU	Rectified Linear Unit
ML	Machine Learning
DL	Deep Learning
AI	Artificial Intelligence
OCR	Optical Character Recognition
IDLE	Integrated Development Environment
CSV	Comma Separated Value

Chapter 1: Introduction

1.1 Introduction

1.1.1 Devanagari Characters

Devanagari is the national font of Nepal and is used throughout the various part of India also. Devanagari system have 10 numeral characters(०, १, २, ३, ४, ५, ६, ७, ८, ९) and 36 consonants (क, ख, ग, घ, ङ, च, छ, ज, झ, ञ, ट, ठ, ड, ढ, ण, त, थ, द, ध, न, प, फ, ब, भ, म, य, र, ल, व, श, ष, स, ह, क्ष, त्र, ज्ञ) with 13 vowels.

Devanagari Characters have some characters with similar structures. The 'ब' and 'व' are different in only the cross inside circle. Similarly 'ङ' and 'ड' have only difference is dot. Also the character 'प',' म ', and 'य' are almost same. The character '२' and 'र' are almost same.

Some characters like 'क्ष', 'त्र', 'ज्ञ' are derived from other previous characters like 'ग', 'य' etc.

1.1.2 Optical Character Recognition

The character recognition system had been extremely popular from decades. They are applied in various field of applications. The most common ones are OCR(Optical Character Recognition), text conversion, text recognition, robotic vision, number plate scanning, etc. The character recognition systems mainly is of two types- Online Character Recognition and Offline Character Recognition. On Online recognition, the model is usable only when model is created whereas offline recognition works anywhere once OCR is created. Basically OCR uses Machine Learning.

1.1.3 Machine Learning

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. ML algorithms are used in a wide variety of applications, such as email filtering, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. ML is closely related to computational statistics, which focuses on making predictions using computers.

Some popular terminology used around ML environments are:

1. **Supervised Learning:** Thee algorithm builds a mathematical model from a set of data that contains both the inputs and the desired outputs. For example, if the task were determining whether an image contained a certain object, the training data for a supervised learning algorithm would include images with and without that object

(the input), and each image would have a label (the output) designating whether it contained the object.

2. **Unsupervised Learning:** The algorithm builds a mathematical model from a set of data which contains only inputs and no desired output labels. Unsupervised learning algorithms are used to find structure in the data, like grouping or clustering of data points. Unsupervised learning can discover patterns in the data, and can group the inputs into categories, as in feature learning. Dimensionality reduction is the process of reducing the number of "features", or inputs, in a set of data.

3. **Optimization:** It is the most essential ingredient in the recipe of machine learning algorithms. It starts with defining some kind of loss function/cost function and ends with minimizing the loss using one or the other optimization routine. The choice of optimization algorithm can make a difference between getting a good accuracy in hours or days. The applications of optimization are limitless and is widely researched topic in industry as well as academia. Examples are SGD(Stochastic Gradient Descent), ADAM(Adaptive Momentum), Gradient Descent etc.

4. **Loss:** It is a non-negative value, where the robustness of model increases along with the decrease of the value of loss function. Common loss functions are MSE(Mean Squared Error), Categorical Cross Entropy etc.

5. **Underfitting and Overfitting:** Underfitting is when there is few train dataset and the loss becomes high for test set. Overfitting is when there is few dataset and the model is trained for long time and eventually model remembers the dataset. Both of these problems cause high loss and less accuracy. Using generalization techniques, proper loss function, train data and learning rate can solve these problems.

1.1.4 Deep Learning

Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network. Convolutional Neural Networks along with other Neural Networks are used on Deep Learning techniques.

1.1.5 Convolutional Neural Networks (CNN)

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They are algorithms that can identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data.

Convolutional networks perform optical character recognition (OCR) to digitize text and make natural-language processing possible on analog and hand-written documents, where the images are symbols to be transcribed. CNNs can also be applied to sound when it is

represented visually as a spectrogram. More recently, convolutional networks have been applied directly to text analytics as well as graph data with graph convolutional networks.

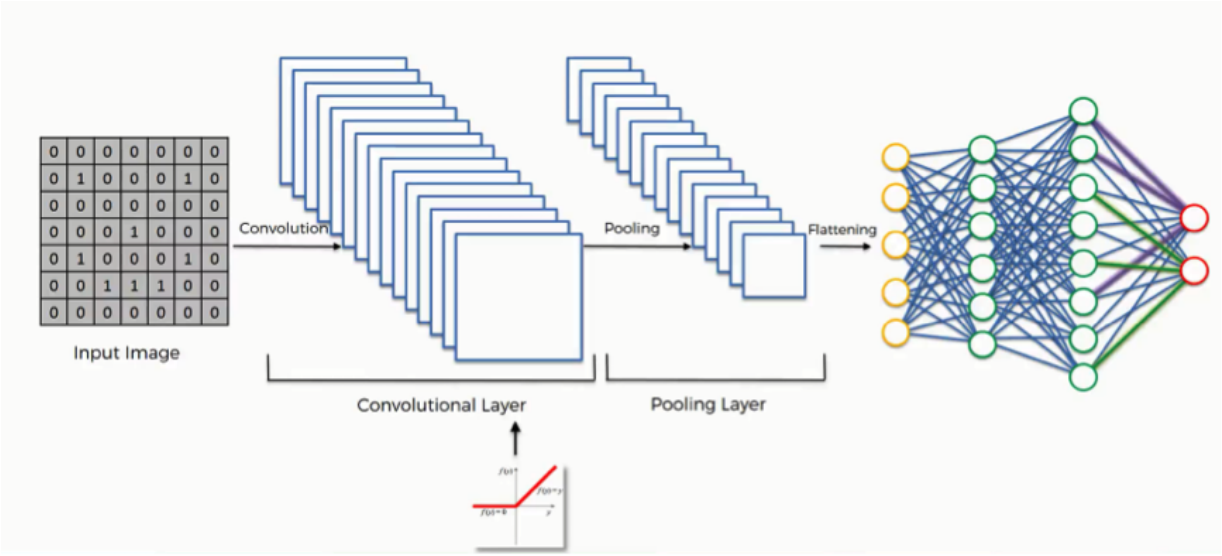


Figure 1. 1: CNN

The efficiency of convolutional nets (ConvNets or CNNs) in image recognition is one of the main reasons why the world has woken up to the efficacy of deep learning. They are powering major advances in computer vision (CV), which has obvious applications for self-driving cars, robotics, drones, security, medical diagnoses, and treatments for the visually impaired.

Some frequently used layers in CNNs are:

1. **Convolutional layer** is the layer where convolution operation happens. Convolution on here is same as on image processing. A filter of same row and column or square size is taken and multiplied across the window that fits filter. The element-wise product is done and summed all. We generally use stride, as how much pixel shift after doing one convolution. Also zero padding is done sometime to add zeros. The convolution layer gives number of filters with same properties. Here more the number of filter, more the accurate model can get but computational complexity increases.

2. **Max-Pooling layer** is a layer where we take only few pixels from previous layers. We must provide a pool size and then that pool size is used on input pixels. The pool window is moved over entire input and max value within the overlaped input is taken. For example, pool of size (2, 2) gives half of input data.

3. **Dropout** is a layer which is actually used to avoid overfitting. This layer randomly cuts the connection between two neurons of different layers. For example, dropout of value 0.5 will cut the half of input's connection. Due to this effect, a NN couldn't memorize input sequence.

4. **Flatten layer** is one where multiple sized input is converted into 1d vector.

5. **Dense layer** on CNN is mostly used to do classification after doing whole convolution thing.

1.2 Problem Definition

Character recognition systems are the part of Computer Vision system that deal with the problem of recognizing the characters found on the image by processing various pixel information on real time. There has been huge research on MNIST handwritten digits and English font recognition from decades but our DCR systems are not progressed well. The automated traffic system can make huge use of this system by doing number plate scanning. This system also is one of basic step of NLP research on Devanagari system. Due to being complex character formation rules and word formation rules, the Devanagari system itself is a problem on recognition task.

1.3 Objectives

The objective of this system is to do possible research on Devanagari Characters for Computer Vision. This system tries to provide as accurrate as possible prediction of characters for the user input image which can be taken real time or from local storage. This system is made simple to understand and using simple Image Processing techniques this system is made abstract as possible. However, it also aims to serve the following objectives:

1. Predict the characters found on the local storage image.
2. Predict the characters found on the image taken by a system camera device.
3. Predict the characters found on the video(real time).

1.4 Scope and Limitation

This project is solely based on Deep Learning techniques, to create the model with high accuracy. The project intends to recognize the characters found on the image by using various image process techniques on backend.

This system is academic project so we are lacking many resources and especially high processing power but we have tried to reach satisfying state of accuracy by using Google Colaboratory for model creation and our system is tested on limited examples.

1.5 Significance of the Project

Handwritten character recognition systems are the one of simple Computer Vision system. This the age of AI and this project is totally done using concept of AI. In depth this project exercises various image processing techniques using simple concept. Implementation of various image processing techniques along with Convolutional Neural Networks for character classification are the implication of this project.

1.6 Features of Project

This system takes the image as input either by local storage or by the camera device or even the video and returns the predictions for the characters present there. Supervised learning was used to train the model and this system works offline. While working offline the system uses the saved trained model from local storage.

Using only the saved model also hides the complexity behind the model creation and further increases the recogniton time.

1.7 Report Organization

- In chapter 1, this report introduce why system is built and how. And also, objectives and features of the project are explained in detail with scope and limitation of the system.
- In chapter 2, this report discuss about the existing system.
- In chapter 3, this report discuss about the requirement as well as feasibility analysis of the system. The process modeling technique are used to give the information about the system requirement.
- In chapter 4, this report discuss about the system design and implementation. We also discuss Neural Network Design and Implementation
- In chapter 5, this report discuss which tools are used in this project to make it possible. The testing is also explained in this part with detailed tabular input and output.
- In chapter 6, this report discuss about the maintenance and support implemented on the project.
- In chapter 7, this report discuss about the conclusion and future enhancements of the project.

Chapter 2: Literature Review

The purpose of this system is to recognize the Handwritten Devanagari Character. Some group of researchers have worked on this task to make accurrate system. Though this project aims to recognize the simple charcters, it is obviously not the first one. It would have been impossible to proceed without the study of previously developed similar typer of projects and systems. Character recogniton system have been developed as OCR on many countries but on our country this project tries to define a problem on different way. Thus study and analysis of few of such project was surplus into our project. Using pretrained model this system tries to classify the individual character present on the image. This project also can be inspiration for AI passionate students.

The most popular Devanagari Handwritten Character Recognition system was designed on 2015 and it is written on the paper that system have achieved 0.98471 accurracy using CNN.

The dataset is publicly available by authors from Computer Vision Research Nepal.

2.1 Review of Papers

2.1.1 Offline Handwritten English Numerals Recognition using Correlation Method

In this paper author has proposed system is to efficiently recognize the offline handwritten digits with a higher accuracy than previous works done. Also previous handwritten number recognition systems are based on only recognizing single digits and they are not capable of recognizing multiple numbers at one time. So the author has focused on efficiently performing segmentation for isolating the digits.[8]

2.1.2 Devanagari Character Recognition Using Neural Networks

Neural network approach is proposed to build an automatic offline character recognition system. Devanagari is an Indo-Aryan language spoken by about 71 million people mainly in the Indian state of Maharashtra and neighboring states including Nepal. One may find so much work for Indian languages like Hindi, Tamil, Bangala, Malayalam etc but Devanagari is a language for which hardly any work is traceable especially for character recognition. In this paper, work has been performed to recognize Devanagari characters using multilayer perceptron with hidden layer. Various patterns of characters are created in the matrix (n*n) with the use of binary form and stored in the file. The back propagation neural network has used for efficient recognition and rectified neuron values were transmitted by feed forward method in the neural network.[7]

2.1.3 Handwritten Devanagari Character Recognition using Neural Network

In this digital era, most important thing is to deal with digital documents, organizations using handwritten documents for storing their information can use handwritten character recognition to convert this information into digital. Handwritten Devanagari characters are more difficult for recognition due to presence of header line, conjunct characters and similarity in shapes of multiple characters. This paper deals with development of grid

based method which is combination of image centroid zone and zone centroid zone of individual character or numerical image. In feature extraction using grid or zone based approach individual character or numerical image is divided into n equal sized grids or zones then average distance of all pixels with respect to image centroid or grid centroid is computed. In combination of image centroid and zone centroid approach it computes average distance of all pixels present in each grid with respect to image centroid as well as zone centroid which gives feature vector of size 2xn features. This feature vector is presented to feed forward neural network for recognition. Complete process of Devanagari character recognition works in stages as document preprocessing, segmentation, feature extraction using grid based approach followed by recognition using feed forward neural network.[3]

Chapter 3: System Analysis

3.1 Requirement Analysis

The requirement includes both functional and non- functional requirements:

3.1.1 Functional Requirements

Functional requirements specify the function that the system or the component of the system must able to perform. It can be well represented by using Use Case Diagram.

Use Case Diagrams are valuable for visualizing the functional requirements of a system that will translate into design choices and development priorities.

They also help identify any internal or external factors that may influence the system and should be taken into consideration.

They provide a good high level analysis from outside the system. Use case diagrams specify how the system interacts with actors without worrying about the details of how that functionality is implemented.

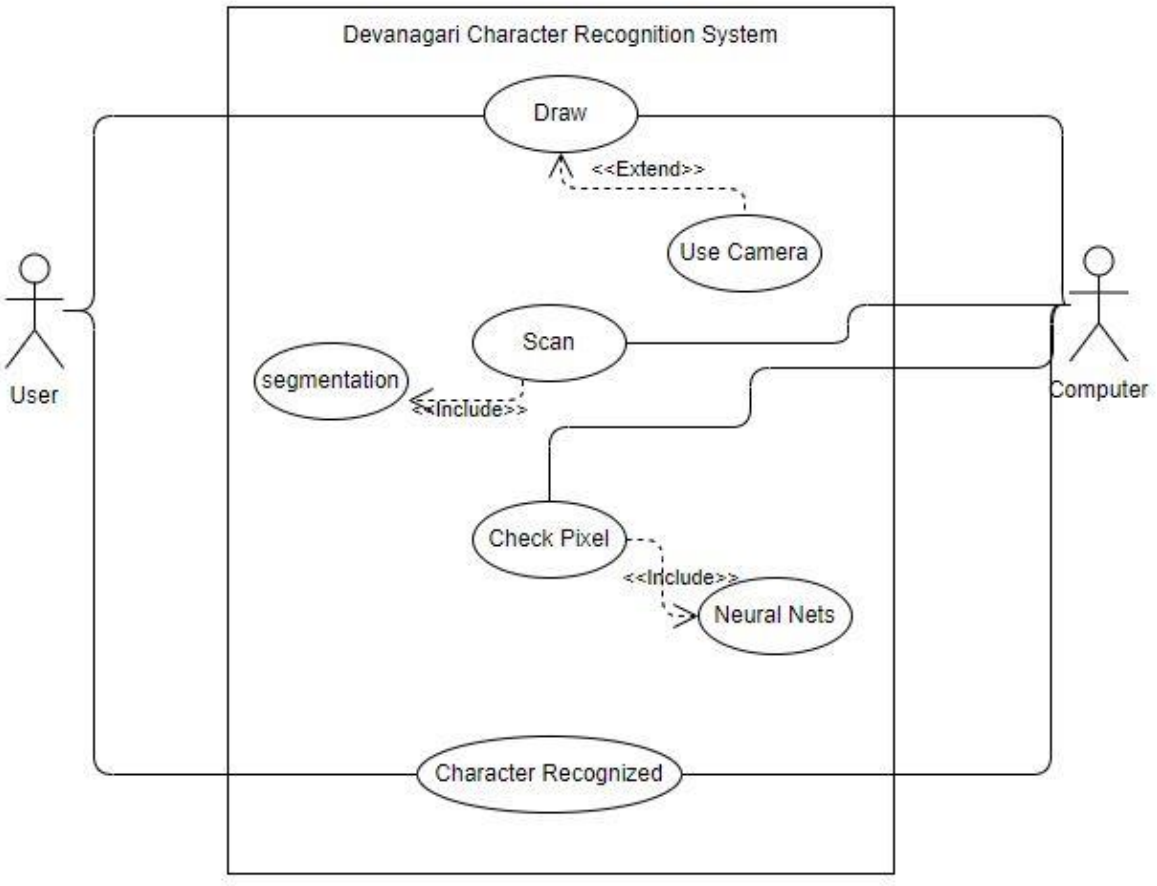


Figure 3. 1: Use Case Diagram

The Use Case Diagram of DCR consisted of the two actors i.e. user and computer. The actors interact with the DCR system shown in the middle rectangle. The use cases are represented in the oval shape. The include and extend links are used to show the relation between the use cases.

The Devanagari Handwritten characters should be drawn by the user. The character dawn are given to the DCR system directly using the file name where the image is saved or using the camera. The DCR system should have the functionality to scan the image given by the user. The scanning process includes pre-processing and segmentation functions serially. The DCR system should also consist the functionality to check the pixels using neural network for the purpose of recognizing the characters. The system ten finally have the functionality to display the character recognized using the user interface.

The above described functional requirements are mandatory for developing the well-functioning DCR system.

3.1.2 Non-Functional Requirements

The non-functional requirements are:

Performance

DCR system should recognize the character accurately. It should be efficient in resource utilization like memory, CPU, storage etc.

Scalability

DCR system should be scalable i.e. it should be able to recognize large number of characters as per demand.

Reliability

The DCR system should classify individual character accurately and must use CNN.

Recoverability

In case of wrong prediction it should be able to suggest users a solution.

Usability

This system can be used by a user who have python and few Machine learning environments.

Interoperability

DCR system is made by using various ML frameworks and python. So, they must operate together.

3.2 Feasibility Analysis

Feasibility study provides viability and quality information for decision making. A well conducted feasibility study has been done to evaluate that the project has potential for success. Thus, following feasibility analysis is taken into consideration for the DCR system:.

3.2.1 Economic Feasibility

Software to be used in the development of the DCR system are python, anaconda and are available for free. For model creating Google Colab is freely availabe and we can use nearly 12 GB of GPU. Hardware like computer, webcam to be used in this project are personnel belonging.

3.2.2 Operational Feasibility

DCR system is based on CNN and is made to recognize character with some level of error. It is planned to a python program system with a very simple user interface. It can be easily used by normal users by installing python. There is desire for this kind of system as Nepal does not have popular research on NLP.

3.2.3 Technical Feasibility

DCR system is to be developed using Python and Machine Learning framework Keras and tensorflow. We have technical knowledge of Machine Learning algorithm and Image Processing.

3.2.4 Schedule Feasibility

The Gantt chart of the DCR project is as follow:

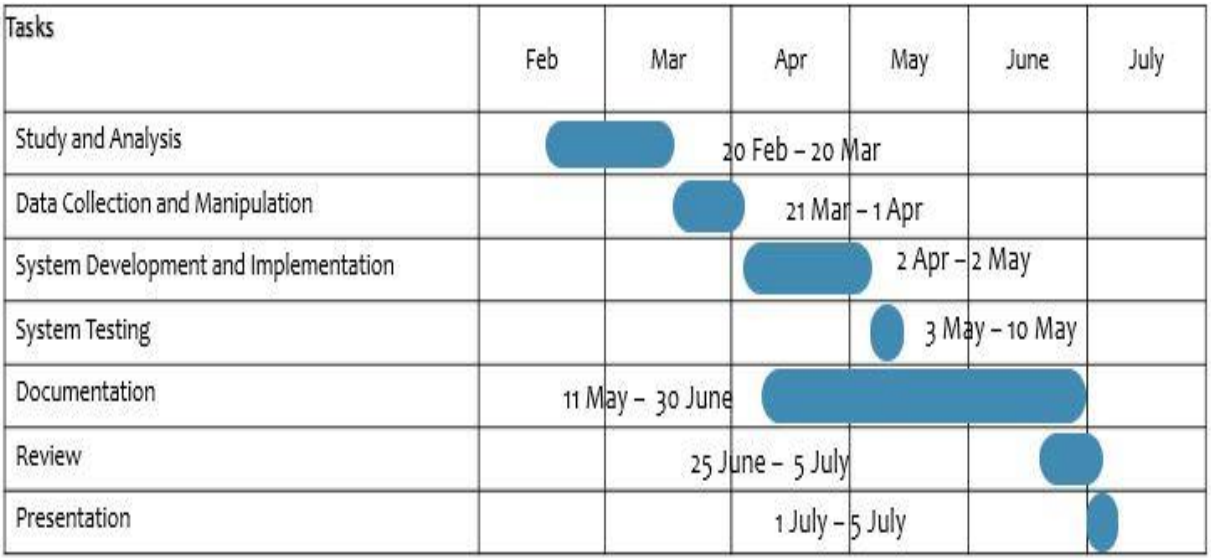


Figure 3. 2: Gantt Chart

The schedule of DCR system has 7 major activities. Study and analysis is scheduled to be completed on first month. Data Collection and Manipulation is to be done on following 10 days. Development of System and implementation starts on third phase and is selected to be run for 1 month. In next 7 days, system testing is scheduled to be done. The documentation is to be done after the system development started and is to be done for 50 days. The system is reviewed on 6^{th} phase and is done for 10 days. On last phase presentation is to be done for 5 days.

3.3 Constructing System Requirements

- **Process Modeling**

  Process modeling graphically represents the process that capture, manipulate, store, and distribute data between a system and its environment and among system components. Each process transforms inputs into outputs.

  Data Flow Diagram (DFD) is commonly used in process modeling.

  A DFD is a pictorial representation of the movement of data between entities and the processes and data stores within a system. They can be used to analyze an existing system or model. The entities are represented by rectangular boxes, the process is represented by the circle and the directional link shows the flow of data between entities and the processes.

  The DFD of DCR system in shown in the next page. The 3 levels of DFD are shown in the figure. As we go deep in analyzing the data flow, we move to the next level DFD. The three levels are described below with the higher level being more detailed than the consecutive lower level.

  1. **Level 0 DFD:** It is the simplest form of flow of data in the system. It gives the basic idea about the flow of data. Level 0 DFD shows that the input image is given to the DCR system and the recognized character is given as output by the system.

  2. **Level 1 DFD:** Level 1 DFD is the expansion of level 0 DFD. The input image is given to the DCR system by the help of user interface. The scanned image from the user is then pre-processed. The pre-process procedure involves finding the borders and segmenting the image if it contains multiple characters. The features extraction process extracts the features from the pre-processed image of character. The extracted features are passed to the process recognition of character and the recognized character are displayed to the user.

  3. **Level 2 DFD:** Level 2 DFD is the more detailed expansion of process in level 1 DFD. The Level 2 DFD shown in the figure is the division of the process 1.4 in level 1 DFD into two sub processes i.e. 1.4.1 and 1.4.2. At first the co-relation is calculated for all nepali alphabets using the extracted features. Then the maximum co-related value is obtained. The alphabet with the maximum co-related value is displayed as the recognized character.

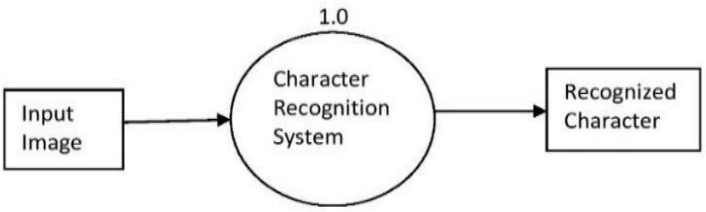


Fig 1: Level 0 DFD

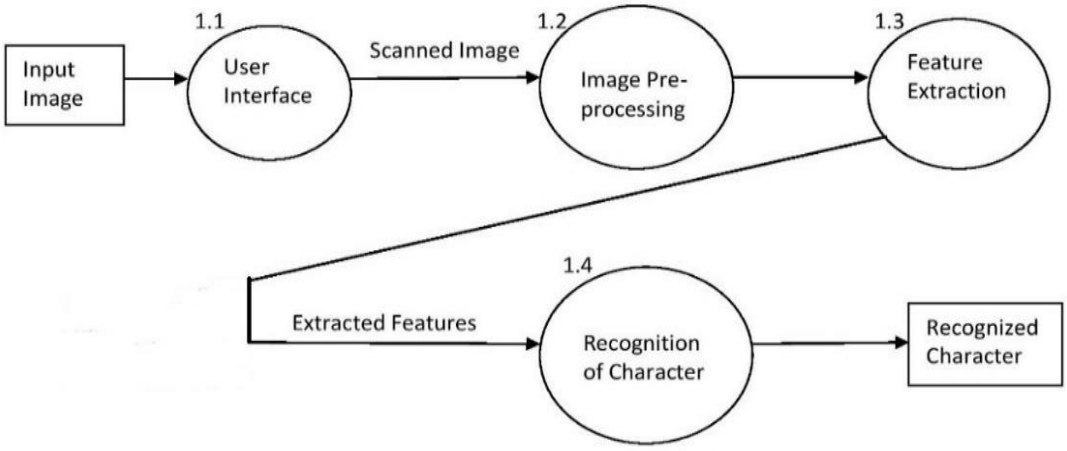


Fig 2: Level 1 DFD

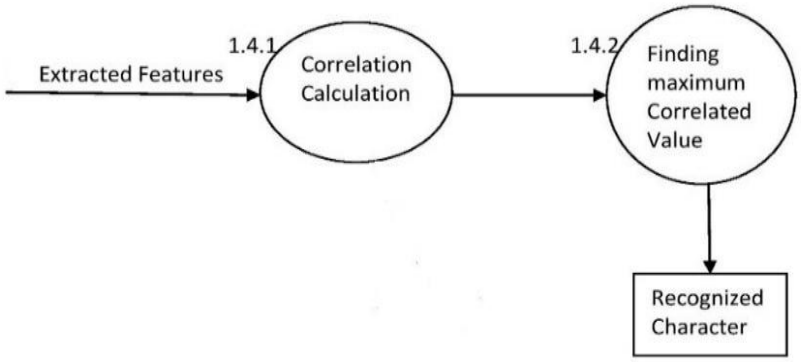


Fig 3: Level 2 DFD

Figure 3. 3: DFD

Chapter 4: System Design and Implementation

4.1 System Design

4.1.1 Block Diagram

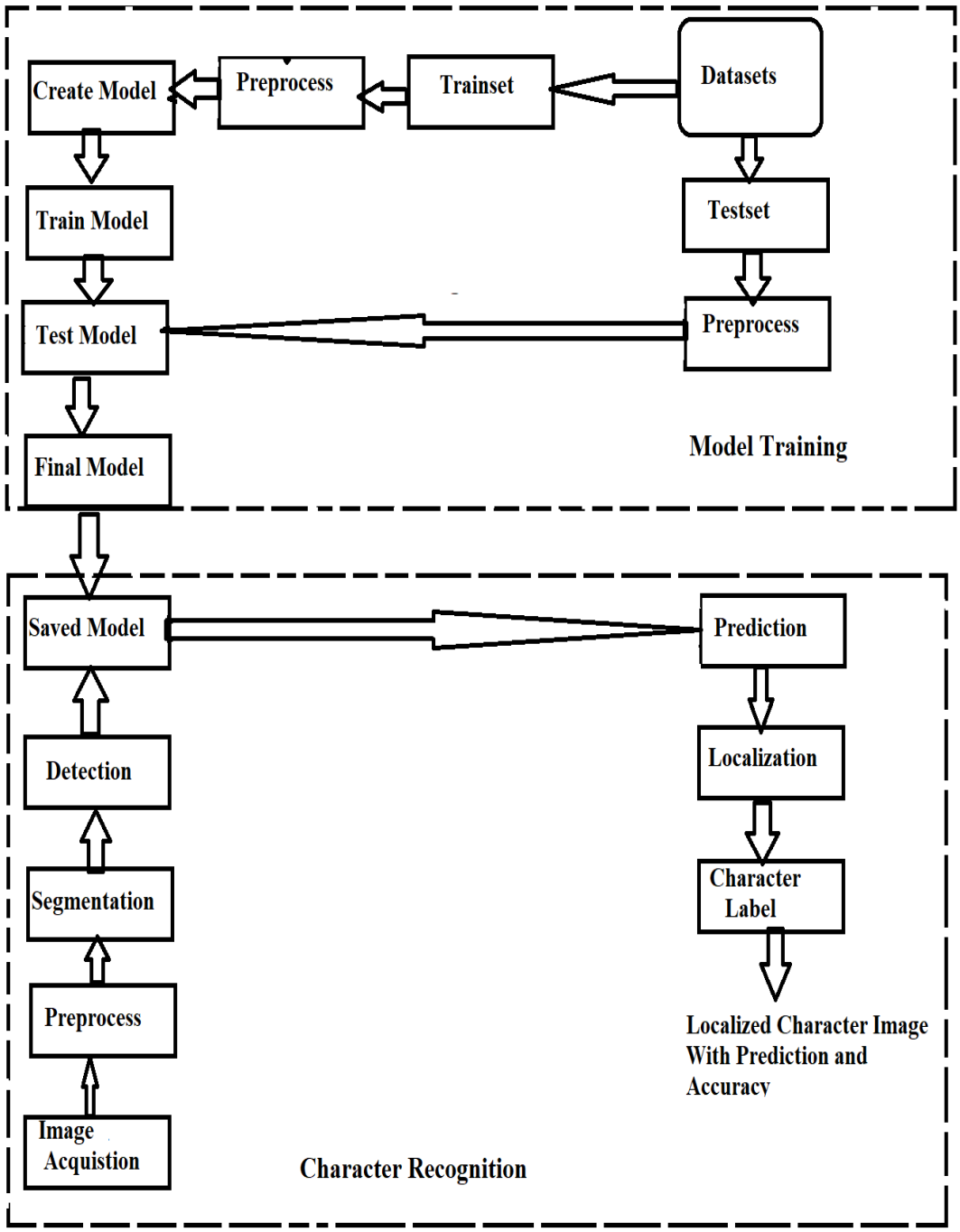


Figure 4. 1: Block Diagram of DCR system

Model Training: The training set is developed from the data set which have been downloaded from the online source. After this, the training data set is pre-processed. Neural network model using Keras Framework is created. After this the training set is passed to the created neural network model. The model will be train accordingly and gives us test model.

The test set is also developed from the data set. The test set is pre-processed and passed to the test model which have been already created. After the desired accuracy have gained, the test model is finalized to get the final model and the final model is saved.

Character Recognition: In the first step the raw image is acquired with the help of camera. This image is pre-processed for abstracting the features. The image is segmented if it has more than one character. The segmented image is passed to the saved model. The model will predict the characters, localize image and display it to the user.

4.1.2 Dataset Preparation

The dataset is publicly available on the web. The dataset is divided into two parts, test and train. After downloading the dataset from web it should be converted to the best format to feed into NN. There are different collections of train and test images in grayscale. Trainset have 78,200 examples with 1700 for each character. And testset have 200 examples for each character.

The foreground of all images are in white and the background are black color. Each image have 2 pixels border on each side of character. So the character is 28 * 28 pixel grayscale image plus 2 pixels border on each side makes the image 32 * 32 pixel. Each image is on PNG format and each example is inside the folder of its label.

To train a model, passing the entire image file by moving back and forth to folders increases the runtime complexity of the process so to avoid these problems the entire image is saved into a CSV file by converting every image pixels in range 0 to 255. On CSV file first column is the label and rest 1024 are pixel values. Labeling is done 0 to 46 serially like:- 0 to ‘0’, 10 to ‘ka’ and 46 to ‘gya’. Both train and test images are saved into individual single CSV file and it takes nearly 10 minutes to complete on local device. But size of CSV file is larger than the entire image files size.

4.2 Implementation

4.2.1 Implementation in Python

All the coding was done on python 3.6 along with popular libraries like Numpy, Matplotlib, Opencv, Keras, Tensorflow and other bulit in libraries. Keras is a popular neural networks framework using Theano or Tensorflow as a backend. It supports GPU training, that saves a lot of time. Opencv is popular for image processing applications, numpy is famous for array operation and matplotlib is twin of Matlab. Tensorflow is used as backend here.

4.2.2 Implementation in Jupyter Notebook

In addition to displaying/editing/running notebook documents, Jupyter Notebook was used as major component during the development of DCR system in this project. It was also used for the reporting and visualization of train/test.

4.2.3 Implementation on Google Colab

Google Colab provides nearly 12gb of GPU and high TPU, and also CPU for model train/test online. Training the large dataset on our system was challenging due to lack of resources and Google Colab is nearly 8 times faster than our system so Colab is our best choice. But creating a working environment on Colab is really a challenging task.

4.3 Neural Network Design and Implementation

4.3.1 Feed Forward Neural Network

First network is FFNN and it is the simple among the NN class. Here is the model's summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dense_2 (Dense)              (None, 512)               131584
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 46)                11822
=================================================================
Total params: 799,790
Trainable params: 799,790
Non-trainable params: 0
```

Figure 4. 5: FFNN Architecture

And the model was trained for 30 epochs. It was trained using 'adam' optimizer and 'categorical_crossentropy' as loss. All the internal layers used the 'sigmoid' function whereas the last layer used 'softmax'. 20% of training data was split for validation. The batch size was 32. The time taken was 42 minutes on Dell I5 with 8gb RAM. Here is the training progress:

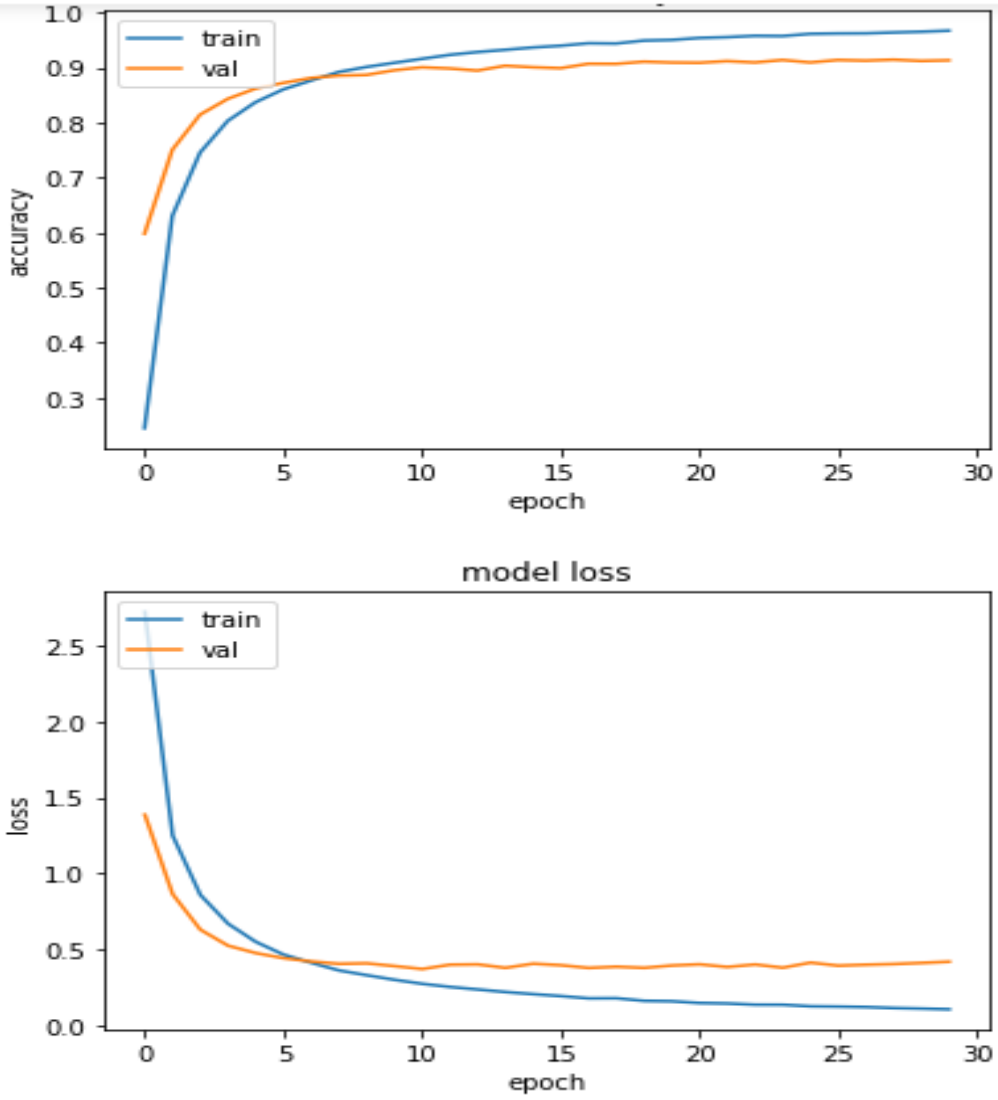


Figure 4. 6: FFNN Training Progress

The training process didn't progress after 5th epoch which clearly shows that this model is suffering from overfitting. Overfitting can be decreased by adding dropout layers, regularizations and some noise also. The test accuracy was 91.77%.

4.3.2 Convolutional Neural Network

Multiple CNNs with same network architecture but with different parameters were trained. There are lots of advantages of using CNN, one of them is few learning parameters.

The architecture parameters in each layer is as follows:

- Number of filters: 32
- Filter size: (3, 3)
- Pool size: (2, 2)
- Activation function: ReLu on except final layer, softmax on final layer
- Dropout: 0.25 on inner and 0.5 on dense layer

The architecture is as follows:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 30, 30, 32)	320
conv2d_2 (Conv2D)	(None, 28, 28, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 32)	0
dropout_1 (Dropout)	(None, 14, 14, 32)	0
conv2d_3 (Conv2D)	(None, 12, 12, 64)	18496
conv2d_4 (Conv2D)	(None, 10, 10, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 5, 5, 64)	0
dropout_2 (Dropout)	(None, 5, 5, 64)	0
flatten_1 (Flatten)	(None, 1600)	0
dense_1 (Dense)	(None, 256)	409856
dropout_3 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 46)	11822

Total params: 486,670
Trainable params: 486,670
Non-trainable params: 0

Figure 4. 7: CNN Architecture

A. CNN-0

- Loss: Categorical Cross Entropy
- Optimizer: Adam
- Batch size: 100
- Epochs: 100
- Validation split: 0.3
- Train time: 21.004 minutes on Google Colab
- Test accuracy: 99.21%

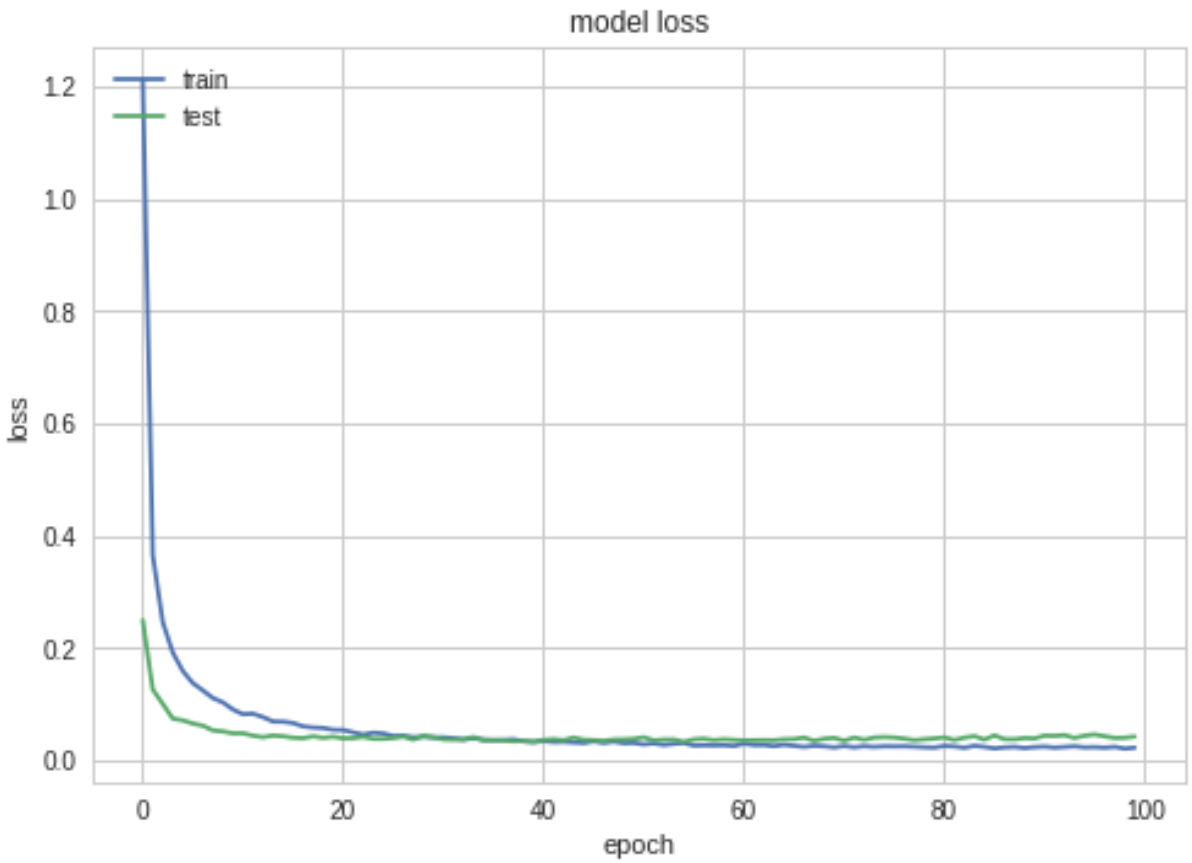


Figure 4. 8: Loss CNN-0

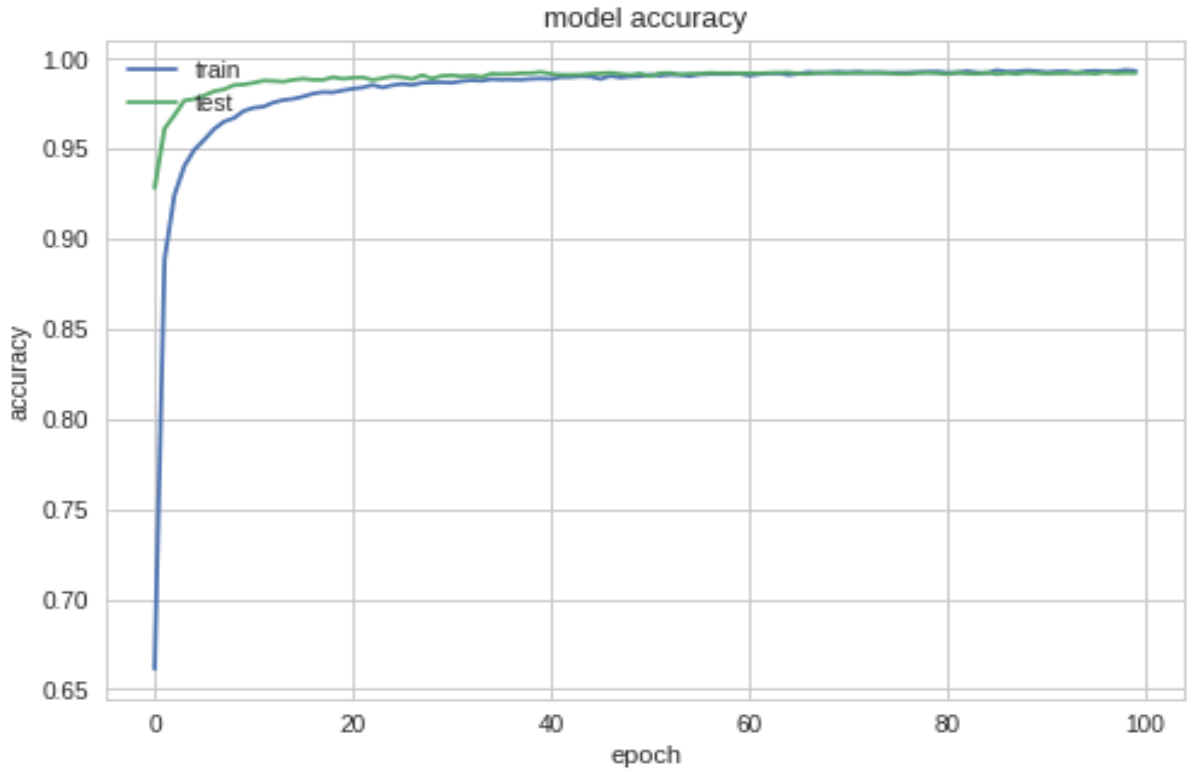


Figure 4. 9: Accuracy CNN-0

B. CNN-1

- Loss: Sparse Categorical Cross Entropy
- Optimizer: SGD
- Batch size: 100
- Epochs: 100
- Validation split: 0.3
- Train time: 18.91 minutes on Google Colab
- Test accuracy: 99.13%

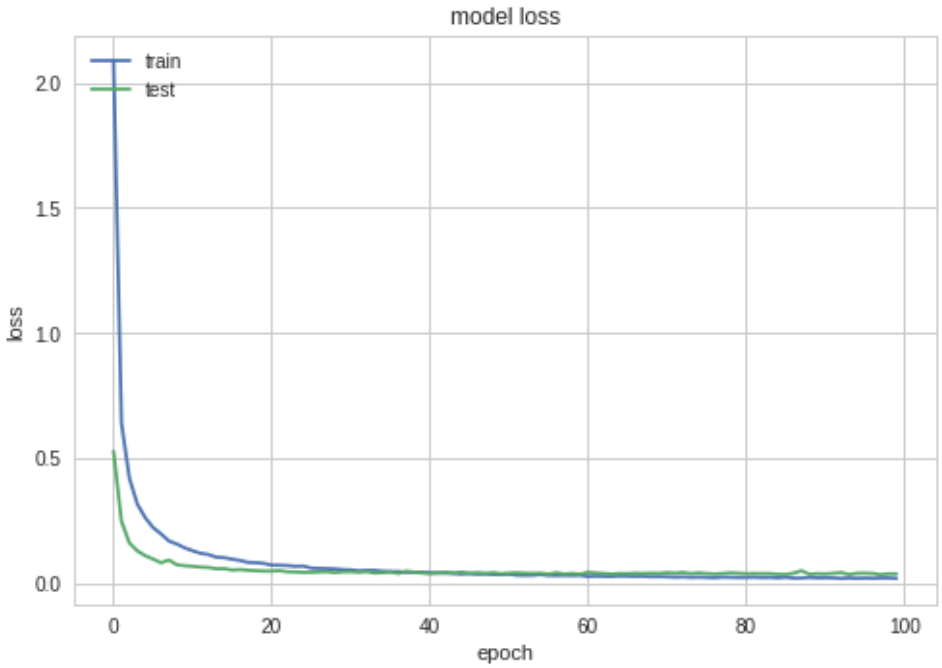


Figure 4. 10: Loss CNN1-1

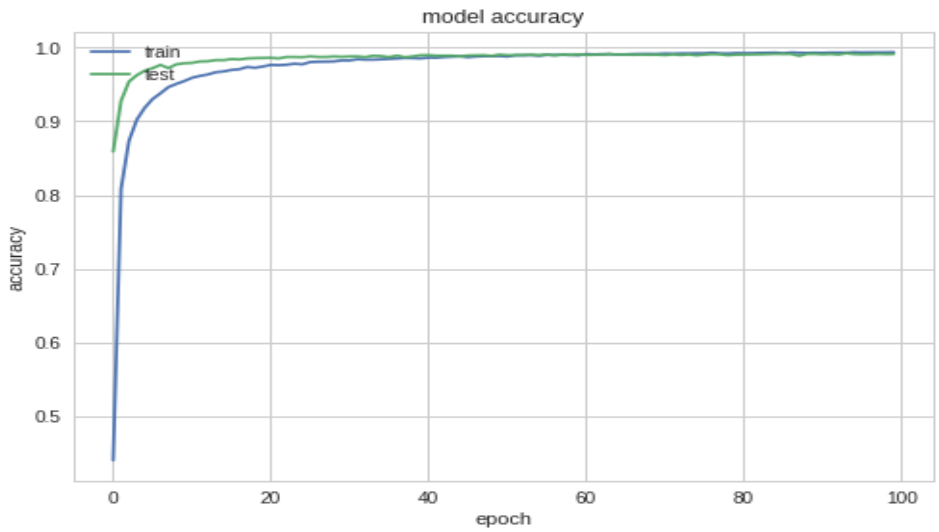


Figure 4. 11: CNN-1 Accuracy

C. CNN-2

- Loss: Categorical Cross Entropy
- Optimizer: SGD
- Batch size: 32
- Epochs: 100
- Validation split: 0.2
- Train time: 37.86 minutes on Google Colab
- Test accuracy: 99.29%

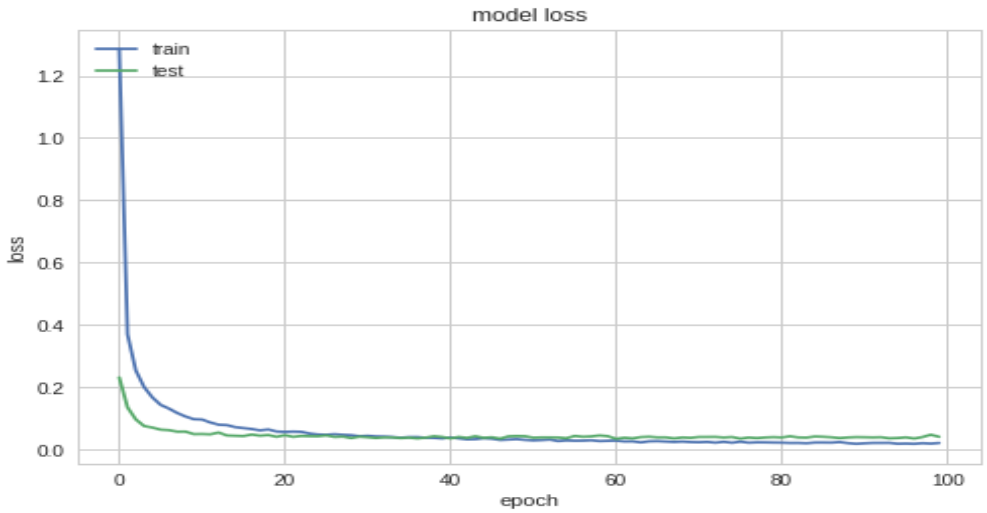


Figure 4. 12: CNN-2 Loss

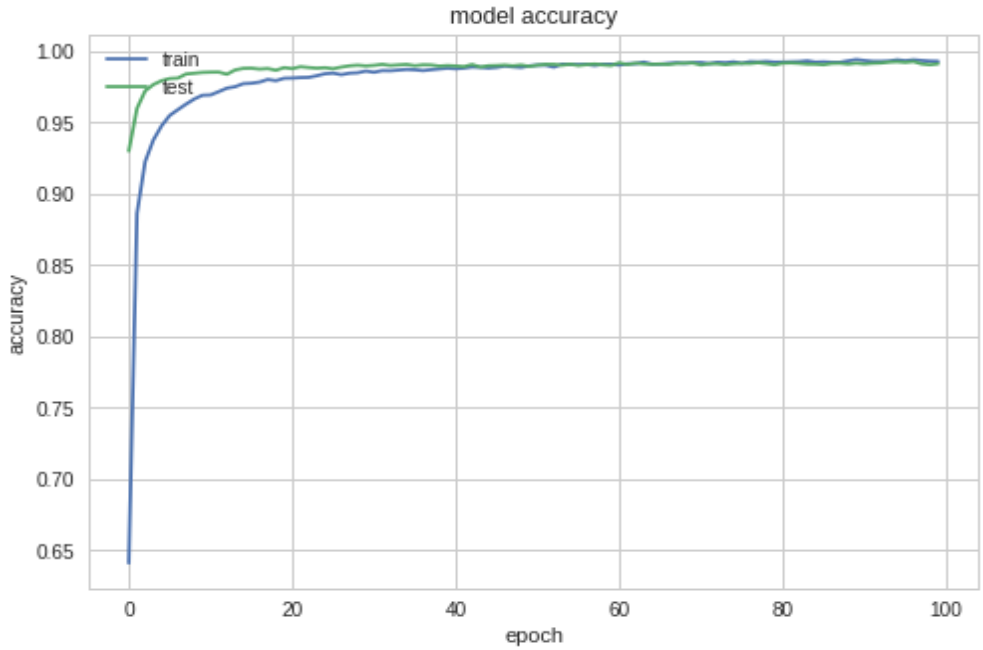


Figure 4. 13: CNN-2 Accuracy

4.4 Image Processing Model

The model training is finished and the intention was real world character recognition. Character Detection is an electronic conversion of images of typed, handwritten or printed text into machine-encoded text. Here input image is provided from directory or is taken from computers' camera. The input image is thus processed and segmented and provided to NN for prediction. In most of the cases, input image may contain lots of noise which make difficult for our NN to predict character on the image. The provided image may also include less brightness and not well written characters which make difficult or sometimes unable to predict characters. So provided images here are assumed as high quality and only high quality images are easier or possible to predict character.

4.4.1 Image Acquisition

Image acquisition part plays the main role on recognition problem. Because no matter how accurate model was on testing, the real world image will never be same. In real world image, there will be lots of noise, blur, and many other quality degradation problems. Image acquisition devices like Camera also affects the property of image taken. This project can work with laptop camera which is nearly 1 MP.

For beginning phase, we worked with the text written using paint's pencil.

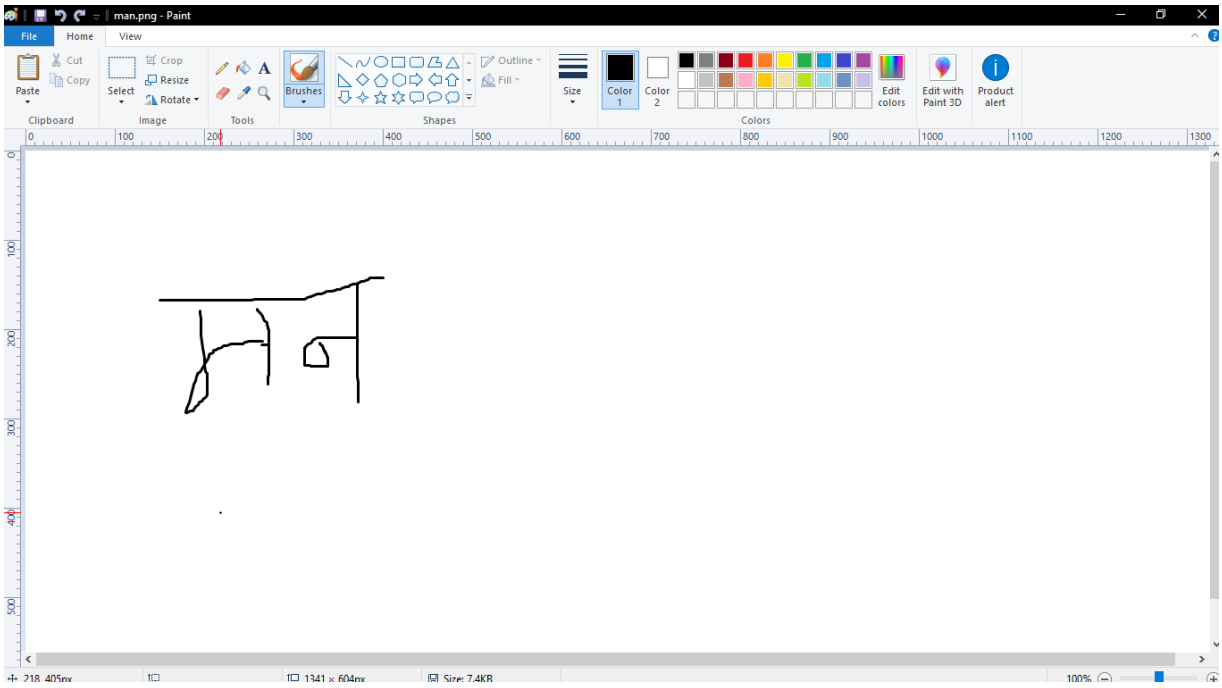


Figure 4. 14: Character Creation using paint

4.4.2 Image Preprocessing

In this stage the image was converted into grayscale, and a numpy array was prepared to store the image pixels. After this the intention was to find foreground and background colors. Removing some noise and doing threshold makes it easier for image to recognize text, and find foreground color. Here we used the combined threshold of Otsu and Binary.

Image thresholding is a simple form of image segmentation. It is a way to create a binary image from grayscale or full-color image. This is typically done in order to separate “object” or foreground pixels from background pixels to aid in image processing.

Otsu’s method is used to automatically perform clustering-based image thresholding, or, the reduction of gray level image to binary image. The algorithm assumes that the image contains two classes of pixels following bi-model histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating two classes so that their combined spread is minimal, or equivalently, so that their- inner -class variance is maximal. It involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold i.e. the pixels that either fall in foreground or background. The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum.

The binary Thresholding function creates a raster output that divides your raster into two distinct classes. The algorithm behind the binary Thresholding function, the Otsu method, was designed to distinguish between background and foreground in image by creating two classes with minimal intraclass variance. When working with a raster dataset that has a unimodal distribution, Binary Thresholding divides the data into two distinct classes. It creates a high-value class, displayed with white pixels, and a low-value class, displayed with black pixels.

The detection of foreground and background pixel is automatically done by Otsu’s thresholding as mentioned above. In this project Otsu’s method was used to separate the foreground (i.e. text color) and background (i.e. paper color) pixel.

The main problem arise during thresholding in this project is that it sometimes fails to detect the exact foreground and background pixels, which results in incorrect detection of the actual text or number. This is due to reflection of light from paper to camera. If there is unequal distribution of light falls on paper then the part where light falls seems to have high intensity than that of other part then the algorithm (i.e. thresholding function) separate that part as foreground and other part as background of text or vice-versa. So that the text detection may fails to represent or predict the actual text. Similarly, high brightness may cause difficult to detect the foreground and background pixels and the intensity of text must be high means that it is better if we use bold letter written with marker for example. The paper must be almost noise free and the text must be clear with same color.

For our project the image after thresholding will be pure binary. The text can be either on black or white but after thresholding it will not be clear. So we need to write a method to obtain exact color. We check the color value of 5 pixels from top left corner to bottom left. This 5 diagonal pixels determines our foreground and background color. This has to be done to represent it as train image’s property so that our system could predict with high accuracy.

4.4.3 Segmentation

The segmentation means to break the entire image into small fragments. On this step, checking of image was done to determine if there are multiple characters present or not. Segmentation process was done here on unique way by defining the possible rows for top most part of character and the possible percentage of space between characters. Each segment is further passed to a prediction process.

Segmentation function was used in order to segment each digit or the character from the group of digits or words for the purpose of detecting multiple characters. Each segments were given as input to neural network which was trained by providing set of single character or digit. Thus, it could predict each input and finally gives the desired output. Here the input was scanned from left to right after rotating it towards left (excluding top joining line i.e. called as 'diko' in Nepali language) until background pixel is found and when background pixel is encountered, function store it as first character and predict it and process is continued. But the problem arises in some characters like ग, ण which contains background pixels in the same word when passed to segmentation function. And the result fails to meet pre-defined determination. This could happen with others word too. Thus, we define the gap between foreground and background pixel above which function store it as a single character and store those background location or columns in array and send only those part of array to the prediction function. As a result the prediction with high accuracy is obtained.

4.4.4 Character Prediction/classification

Each segment was passed to prediction process. Before doing actual prediction, the shape of segment must be resized as that of the neural network input. Thus, each segmentis converted into 30 by 30 sized image and in addition, we added a 1 pixel borders around it with background color. Then our segments will be of 32 by 32 shape which is the input shape for our model. Then it is to the neural network. If the segment have high prediction then is assumed that the character should be shown. Prediction can be wrong also depending upon the image quality due to the false segmentation. Then that segment is passed to localization.

When the product of rows and columns of the image to be resized is larger than 2^31, ssize.area() results in a negative number and thus images sometimes cannot be converted into 32x32 size. This appears to be a bug in OpenCV and hopefully will be fixed in the future release. A temporary fix is to build OpenCV with this line:
CV_Assert(ssize.area()>0);

The above applies only to image whose width is larger than height. For images with height larger than width the following code is applied as temporary fix to OpenCV:
CV_Assert(dsize.area()>0);

On the concept of finding foreground and background color 5 pixels from the top left corner were checked. In most of the cases find foreground and background color is detectable but what if in the case of image whose text pixels are on the positions where color checking is being done then in such condition foreground and background pixels values would be taken as same.

If the provided image has very minimum noise, has good brightness and have high quality regarding to its writing then the neural network can predict the provided character in the image with high prediction and if the above characteristic is failed to meet in provided image then neural network is unable to predict character and shows NaN(Not a Number) message.

It can also predict fake images which are non Devanagari character because it is limitation of this project. In fact it is feature of NN with the fact that it can return only labels that was trained. This can also be called as a sensitive part of our project because it can return any character which is not in the fake image. And our project does not have any code to detect "this is fake image" and return to user with the dialog box "Try Again".

The below input is fake image and our NN gave prediction ७.

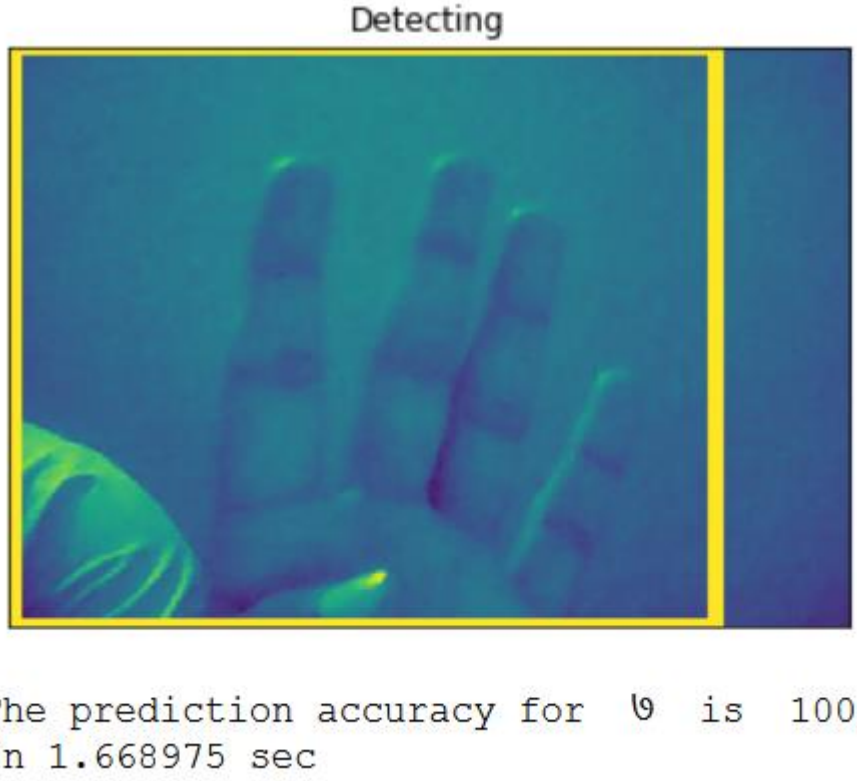


Figure 4. 15: False Image Prediction

4.4.5 Character Localization

Here the intention was to find each segment inside an original image by using concept of template matching. If the template is matched then make a rectangle around the matched place on the original image. This process is done in order to remove unwanted spaces between characters which actually removes the chances of having false segments between characters thereby increasing the accuracy of character prediction. It can be also known as improved version of segmentation. Here cropping is done from borders on the segments.

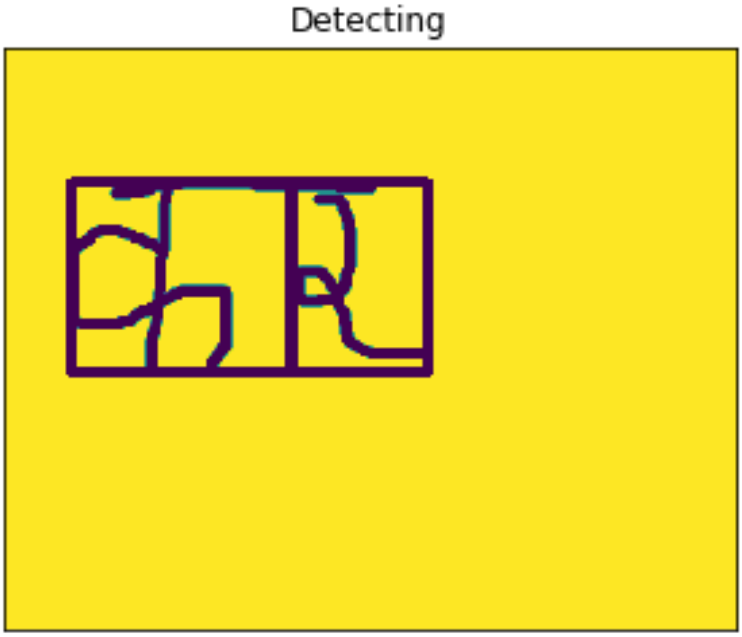


Figure 4. 16: Template Localization

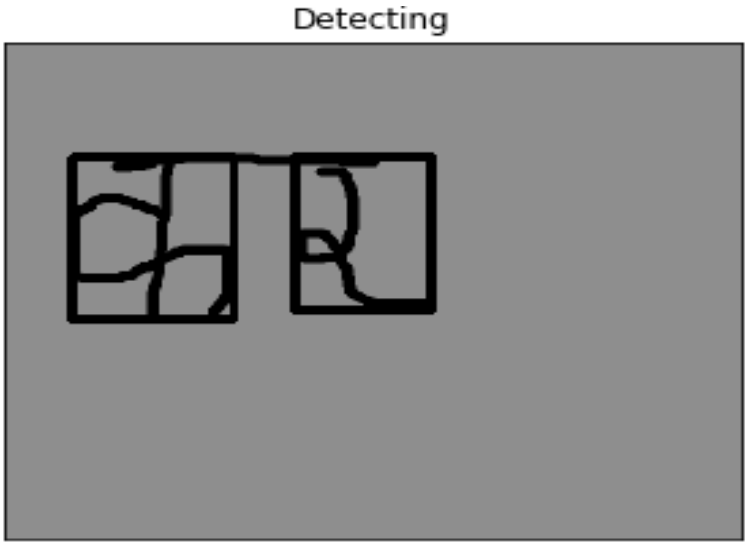


Figure 4. 17: Improved Template Localization

4.4.6 Character Recognition

This is the overall collection of previous processes. The actual recognition is seen after there is bordered around the found character and its corresponding label as well. The final detection takes some time and gives accurate prediction. If poor image is given then it gives false prediction. So high characteristics images is recommended.

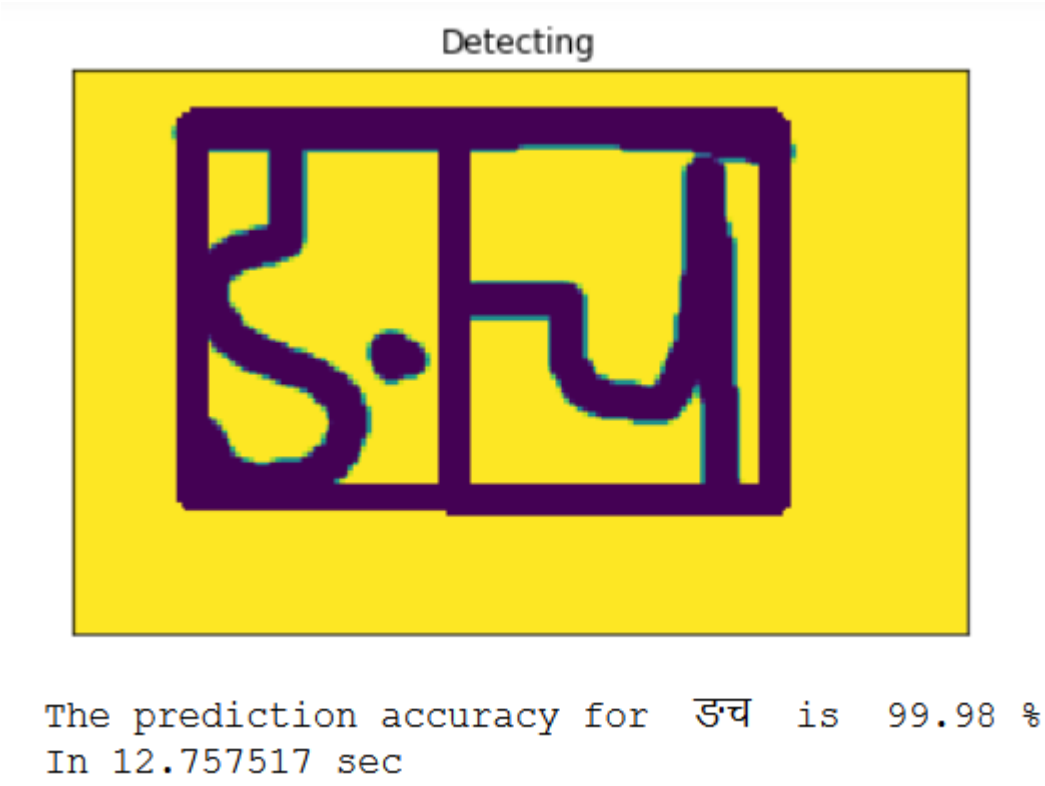


Figure 4. 18: Character Recognition

4.5 List of Algorithms

4.5.1 Create Dataset

Step 1: Start

Step 2: For every directory inside given directory, if exists:

i) Open the images inside that directory on grayscale.
ii) Find the label from the folder name and convert it into appropriate form.
iii) Create array of size 1025 and place label on 1st column them pixels on 1024.
iv) If CSV file exists: Append the array on the top of corresponding file. Otherwise create a CSV file and append.

Step 3: Stop

4.5.2 Create CNN

Step 1: Start

Step 2: Define input shape to network.

Step 3: For each convolutional block :

a. Define no. of output filters, kernel size, stride, padding.
b. Define activation function.
c. Add max pooling layer.
d. Add drop out layer.

Step 4: Add flatten the output of convolution process.

Step 5: Add some dense layer with output shape and activation function.

Step 6: Add drop out layer.

Step 7: Add output layer.

Step 8: Stop.

4.5.3 Train CNN

Step 1: Start.

Step 2: Define loss function, learning rate, optimizer.

Step 3: Compile the model.

Step 4: Define batch size, epochs.

Step 5: Begin training by passing input and desired output.

Step 6: For each epoch:

a. Visualize the training process.
b. If validation accuracy is greater than desired accuracy save the model and go to step 7. Otherwise go to step 2.

Step 7: Stop.

4.5.4 Image Processing

A. Image Preprocessing

Step 1: Start.

Step 2: Get the input image and blur.

Step 3: Threshold the blurred image to get binary image.

Step 4: Find background color as:

```
bg_test = array[thresholded_img[i][i] for i  in range(5)]
if bg_test.all() == 0 then text_color = 255
    Otherwise text_color = 0
```

Step 6: Crop the image to get the area of interest.

Step 7: Stop.

B. Image Segmentation

Step 1: Start.

Step 2: Get the cropped image.

Step 3: Eliminate the 31% of initial rows to remove the “dika” from image.

Step 4: Invert the image.

Step 5: Find the segmenting rows between two characters by checking the block of rows.

Step 6: Find the segments in original cropped image using segmenting rows.

Step 7: Crop the each segments to get the area of interest.

Step 8: Return the segments.

Step 9: Stop.

4.5.5 Recognition/Prediction

Step 1: Start.

Step 2: Get each segments of image.

Step 3: Convert each segments into 32*32 segmented image.

Step 4: Predict the character in segmented image using the CNN.

Step 5: If prediction accuracy is high localize original segment in original image.

Step 6: Display the original localized image.

Step 7: Display the predicted characters.

Step 8: Stop.

Chapter 5: System Testing

The testing phase can be carried out manually or by using automated testing tools to ensure each component works fine. After the project is ready its various components were tested in terms of quality, performance to make it error free and remove any sort of technical jargons. Testing also is done to measure the difference between the desired and the developed system. Testing is need on development cycle of system to ensure that the system's every component works fine.

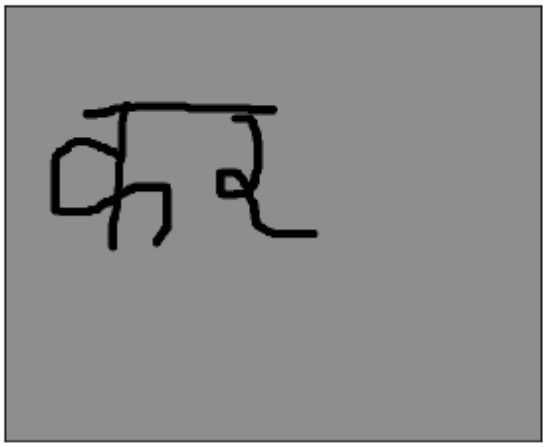
5.1 Unit Testing

During the coding phase each individual module was tested to check whether it works properly or not. Different errors found during unit testing were debugged. Some of the major test cases are listed below:

Test Case 1: Border finding and Crop Character

Inputs: RGB Image

```
Please enter the image directory with name.
tanak.png
```



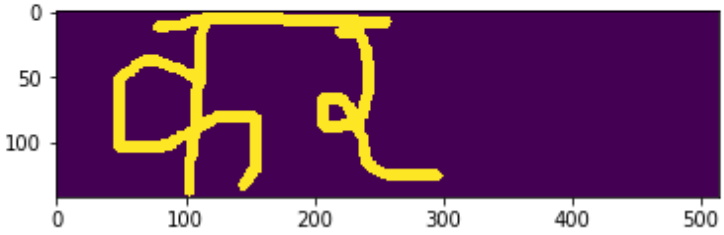


Figure 5.1. 1: Top down extreme position

Figure 5.1. 2: Left Right extreme position

Figure 5.1. 3: Cropped image (original template)

Figure 5. 1: Test Case for Cropping the character from Background

Test Case 1: Results:

- Test passed for paint image with no noise.

Test Case 2: Removing the topmost part of characters

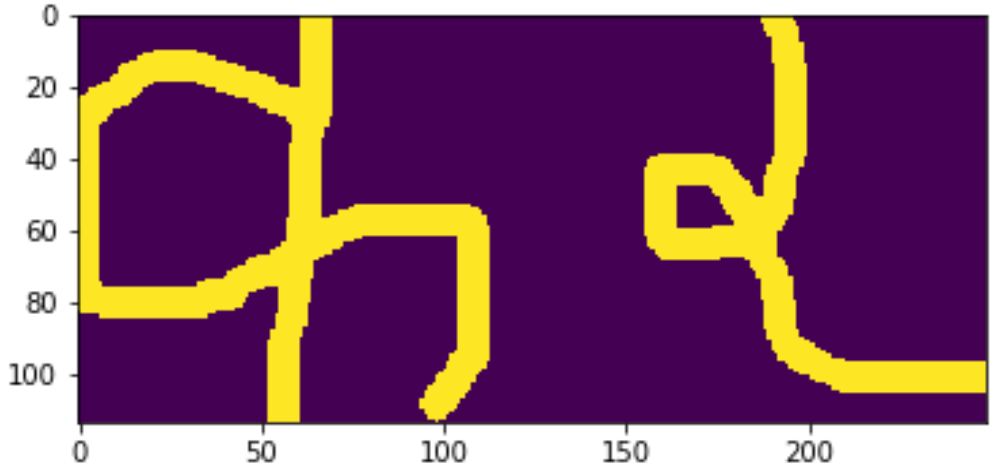


Figure 5. 2: Test Case for removing top most part of image to do segments

Test Case 2: Results:

- Test passed for paint image with no noise.

Test Case 3: Segmenting the characters

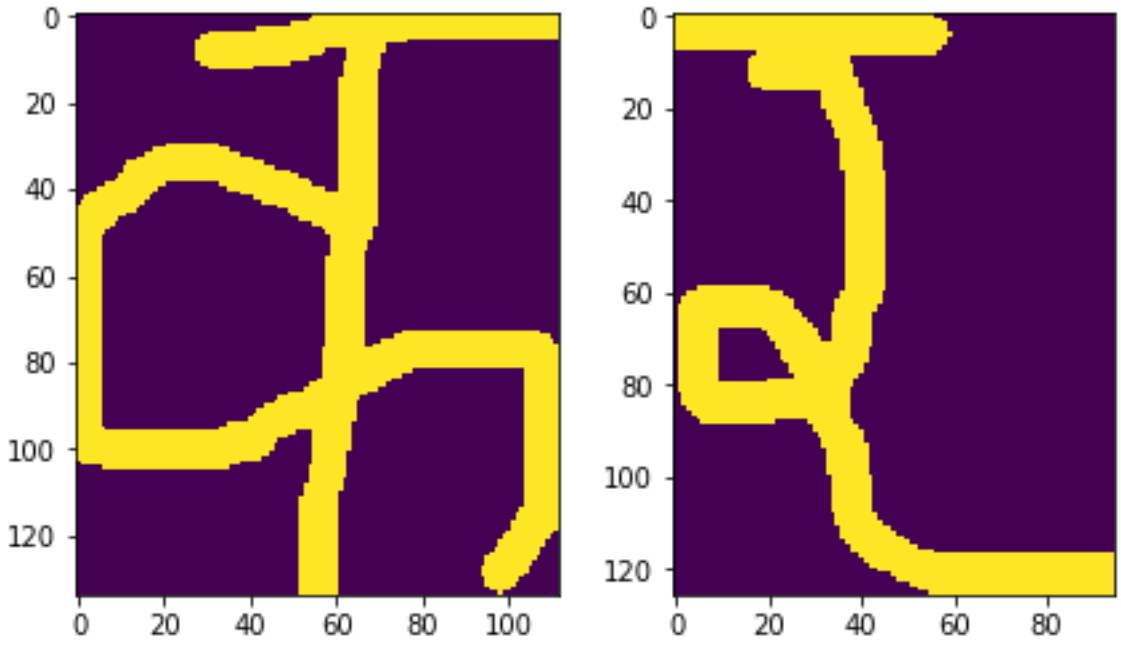


Figure 5. 3: Test case for segmentation of the characters

Test Case 3: Results:

- Test passed for paint image with no noise.

Test Case 4: Template localization

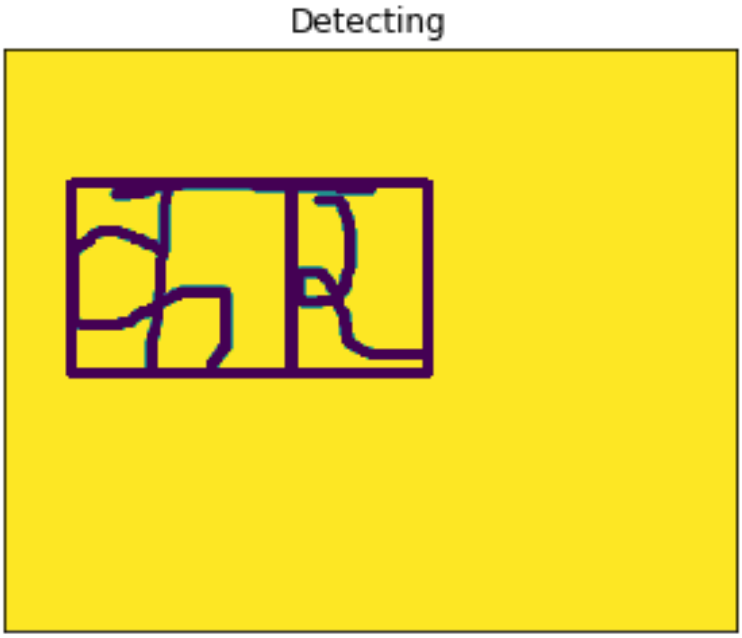
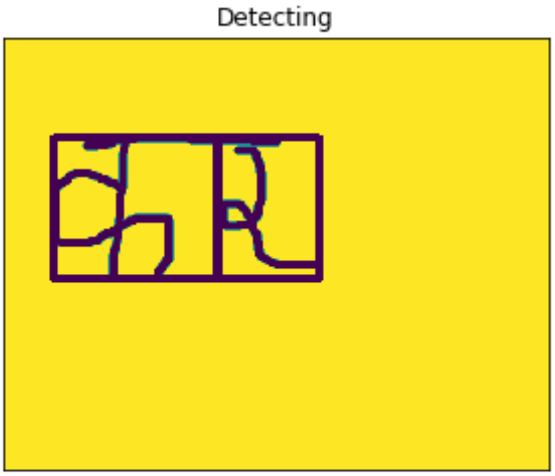


Figure 5. 4: Making border around the matched template of the segments

Test Case 4: Results:

- Unwanted space left between two characters.

Test Case 5: Predicting the characters




```
The prediction accuracy for  करे  is  94.78 %
In 2.863430 sec
```

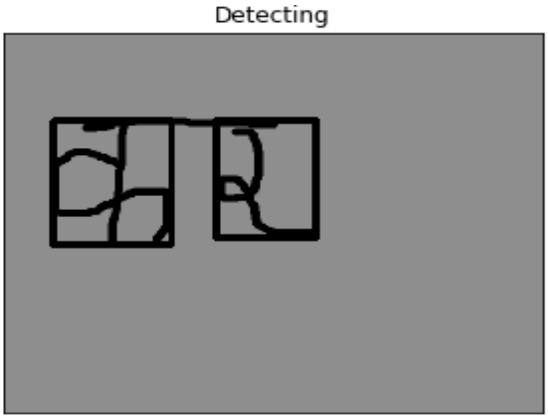
Figure 5. 5: Test case for Predicting Character

Test Case 5: Results:

- Due to huge space between two segments false prediction given.

Test Case 6: Improving segmentation process

Do cropping from borders on the segments also. Which actually removes the chances of having false segments between characters and removes the huge space around the characters.




```
The prediction accuracy for  करे  is  74.60 %
In 2.600607 sec
```

Figure 5. 6: Improved Segmentation process

Test Case 6: Results:

- The space between characters removed.
- The test passed accurately but with less accuracy.
- This process might take more time than previous one.

Test Case 7: Initializing Camera

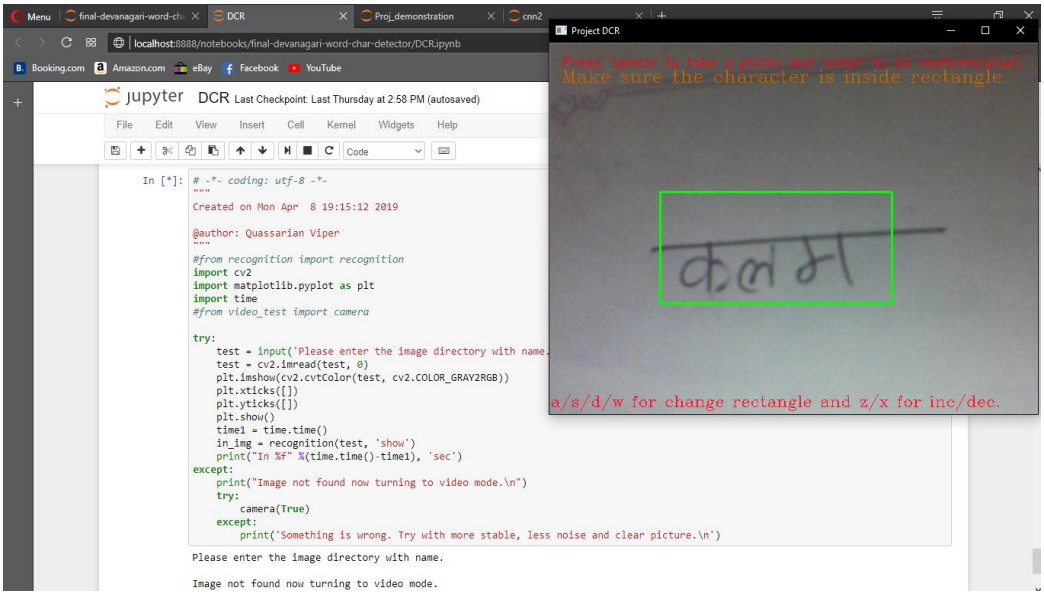


Figure 5. 7: Test case for Initializing Camera

Test Case 7: Results:

- Test passed for laptop with web cam.

5.2 Integration Testing

Integration testing integrates individual modules and tested as groups. Integration testing takes the unit tested modules, group them into the larger aggregates, applies tests and delivers the output.

Some of the major test cases are listed below:

Test Case 1: Integration testing of camera and preprocess module

In this test unit tested camera module and preprocess module are integrated in order to make them work together. Various problems have been solved and the snapshots of test pass are shown below.

Figure 5. 8: Integration testing of camera and preprocessing

Test Case 1: Results:

- The image should have good brightness and the less blur to do better preprocessing.
- The blurring on the image causes the false segmentation and false prediction.

Test Case 2: Integration of Camera and Preprocess with Segmentation module

After the integration of camera and preprocess module have passed the test, segmentation module is integrated. Snapshots of this integration is shown below.

Figure 5. 9: Integration testing of camera, preprocessing and segmentation

Test Case 2: Results:

- The image should have good brightness and the less blur to do better preprocessing.
- The blurring on the image causes the false segmentation.

Test Case 3: Integration testing of Camera, preprocessing and segmentation with detection

After segmented image without error is found as outpur in previous integration testing, finally module of detecting characters is integrated in DCR system. The snapshots of this testing is shown below.

Figure 5. 10: Integration testing of camera, preprocessing, segmentation and detection.

Test Case 3: Results:

- The image should have good brightness and the less blur to do better preprocessing.
- The blurring on the image causes the false segmentation and false prediction.

5.3 System Testing

System testing has done after integrating testing in order to ensure that the whole systems functions properly. After the integration testing the whole system working process was checked. The output was as per the system specifications and hence the system was found to work properly. But there are some important things to be aware about this system's recognition time.

Results:

- If the system is running for long time and memory usage is high then same image takes more time to recognition than on normal condition.
- Cropping each segments might save time while recognition if there are few segments. But if there is large number of segments then it takes more time to do recognition.
- The image taken from web cam usually have few pixel (low quality) hence it takes less time to do recognition but image with better quality have more pixels and so the recognition time increases.

Chapter 6: Maintenance and Support

6.1 Maintenance

DCR is implemented on the windows system with python 3.6 and libraries with specific versions. Since this system have small quality camera, there will not be fine image. As the recognition times increases, the system becomes slower due to memory usage. One possible solution to the low quality image is to show the light toward that object or written text due to which the paper will reflect much light and hence the good image is captured. Also there is always a high chance of getting false segments so changing some parameters around will decrease this problem. The recognition on the video was also done but doing recognition on the every frame of video causes the system to crash because it needs more memory. A typical image requires average of 4sec of recognition time. The time can be decreased by either increasing resources or by optimizing the code.

6.2 Support

The advanced version of Waterfall Model is used in this project i.e. V Model. V model focuses on verification and validation so we can quickly change the requirement of the system. If any changes need to be made they can be accommodated in short period of time as V model allows us to do so. The recognition process can be made fast by increasing resources but this lies on the economic feasibility of system.

Chapter 7: Conclusion & Future enhancements

7.1 Conclusion

This is the age of Artificial Intelligence where intelligent programs/systems does many things. The research has been done on various computer vision field like number plate scanning, autonomous vehicle, text conversion, OCR etc. While the world has already seen the OCR for English characters, this project intended to do small effort for Devanagari Characters recognition. Because this system is still machine dependent it will not be that use for public yet. But with the development of mobile app for this system, language conversion feature can also be added. This system needs more researchers to improve the behavior.

During the entire project development period we were able to develop our skills. This project enabled us to manage time and resource besides of different constraints. We learned how to work on group and hence develop a system.

7.2 Future enhancements

Some of our future enhancements are:

1. Developing a mobile based applications (iOS and Android).
2. Adding the text conversion to different languages.
3. Adding the complex characters recognition.
4. Developing the OCR using Raspberry pi and Arduino.
5. Developing a reinforcement learning.
6. Developing web based application which can store the user details in database.
7. Adding Natural Language Processing features.

Bibliography

1. Chhabra, A. (2019). *Deep Learning Based Real Time Devanagari Character Recognition.* San Jose: SJSU ScholarWorks. doi:https://doi.org/10.31979/etd.3yh5-xs5s
2. Deshmukh, A., Meshram, R., Kendre, S., & Shah, K. (2014). *Handwritten Devanagari Character Recognition.* Pune: International Journal of Engineering Research.
3. Dongare, S. A., Kshirsagar, D. B., & Waghchaure, S. V. (2014). *Handwritten Devanagari Character Recognition using Neural Network.* Kopargaon: IOSR Journal of Computer Engineering (IOSR-JCE).
4. Gyawali, K. P., Acharya, S., & Pant, A. (2016). *Deep Learning based large scale handwritten Devanagari Character Recognition.* Kathmandu: Computer Vision Research Group.
5. Narang, V., Roy, S., Murthy, O. V., & Hanmandlu, M. (2013). *Devanagari Character Recognition in Scene Images.* Washington, DC: IEEE.
6. Negi, V., Mann, S., & Chauhan, V. (2017). *Devanagari Character Recognition Using Artificial Neural Network.* New Delhi: International Journal of Engineering and Technology (IJET). doi:10.21817/ijet/2017/v9i3/1709030246
7. Sayyad, S. S., Jadhav, A., Jadhav, M., Miraje, S., Bele, P., & Pandhare, A. (2013). *Devnagiri Character Recognition Using Neural Networks.* Ashta: International Journal of Engineering and Innovative Technology (IJEIT).
8. Vats, I., & Singh, S. (2014). *Offline Handwritten English Numerals Recognition using Correlation Method.* Chandigarh: IJERT.

APPENDIX

Appendix A:

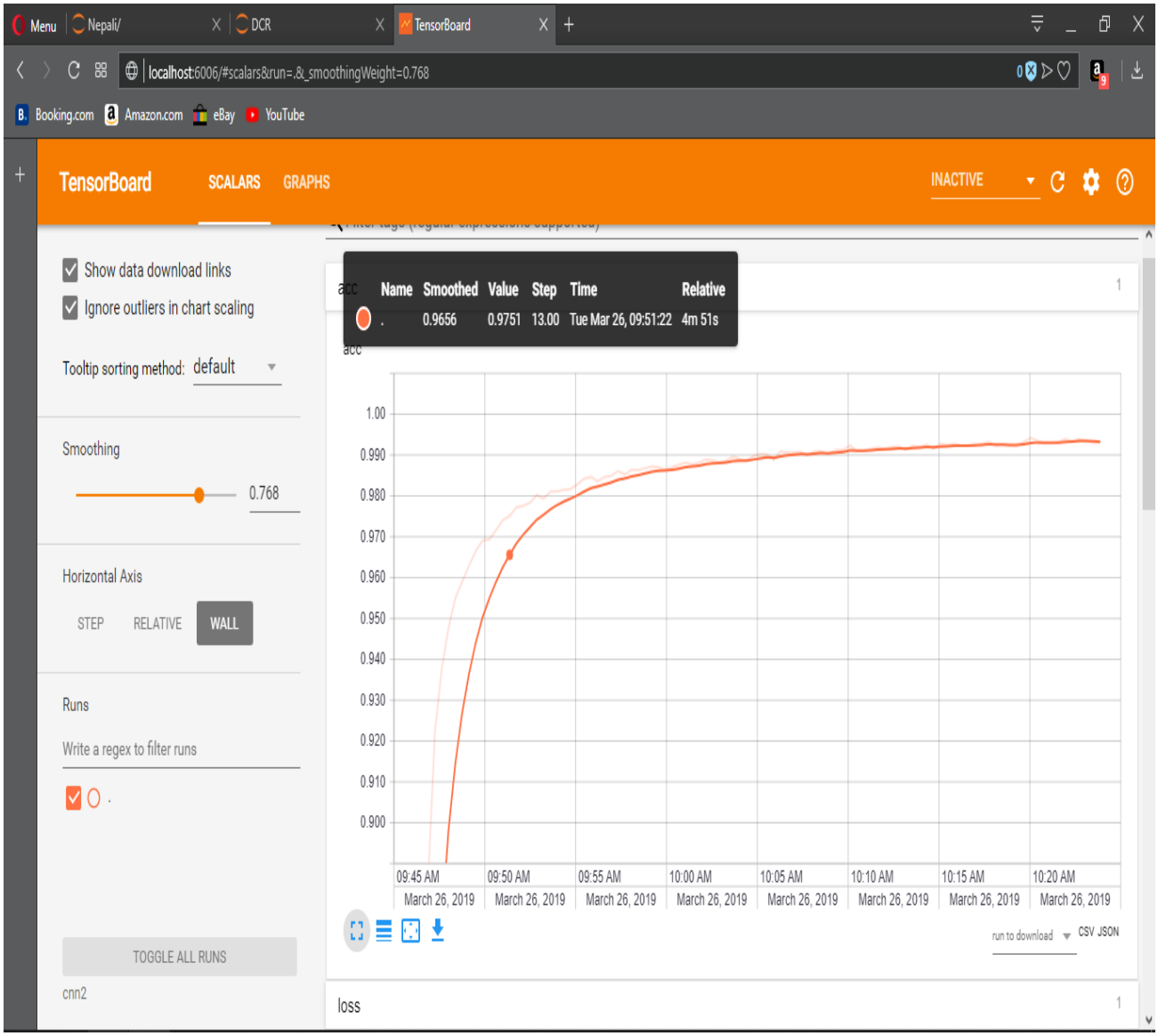


Figure 8. 1: Accuracy visualizing using Tensorboard

Appendix B:

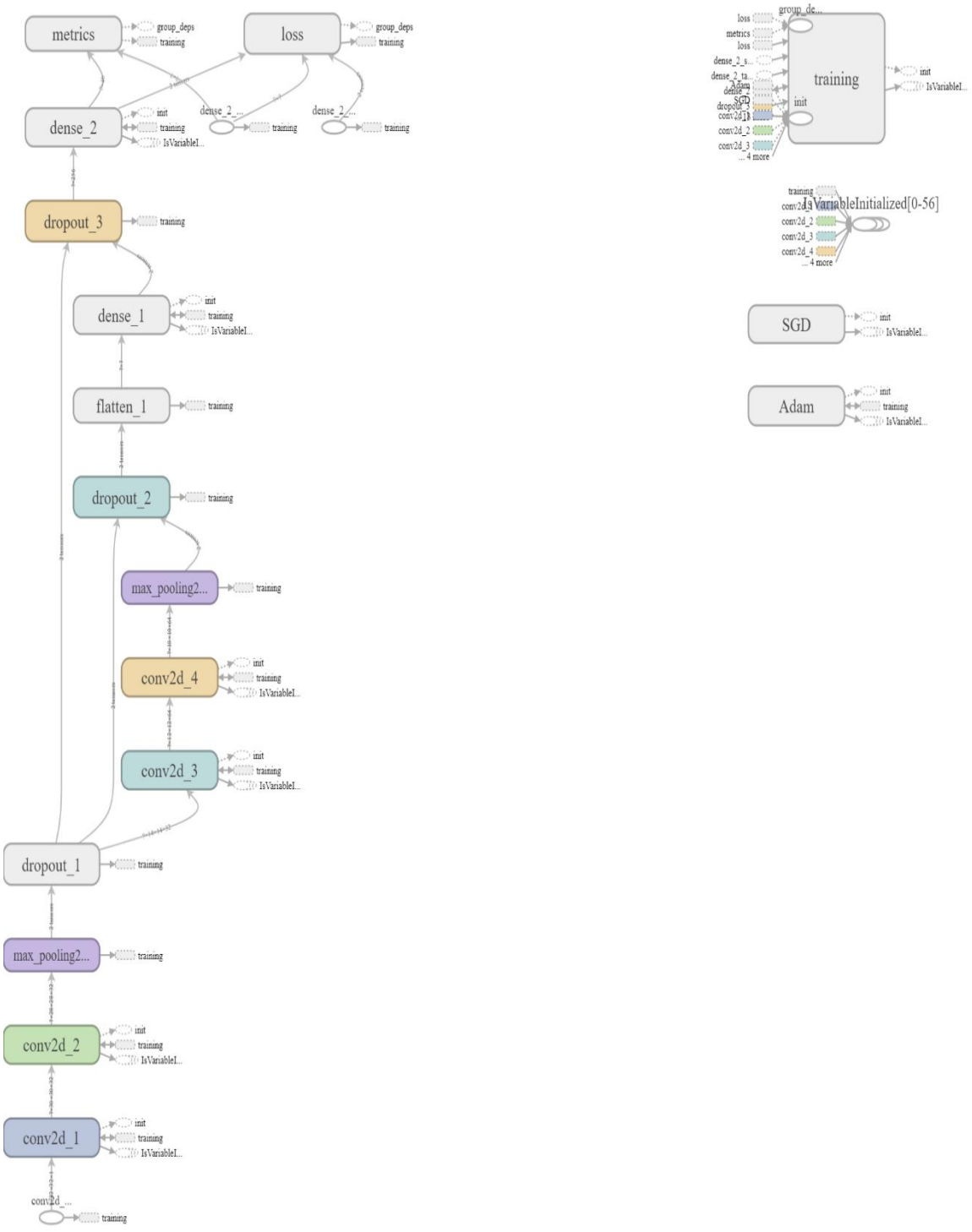


Figure 8. 2: Model Visualization using Tensorboard

Appendix C: Source Code

i. Prepare Dataset

```
#import libraries
import os
import numpy as np
import csv
import cv2


#define a method to wrap all essential processes, we will give location to the
function
def create_csv_data_file(location):


  #Check if files already exists
  exists = os.path.isfile(location[:len(location)-1] + '.csv')
  if(exists):
    print(location[:len(location)-1] +  'set already present.')
  else:
    print("Creating", location[:len(location)-1], " train file.")



    #iterate through the given location's every directory's list
    for each_dir in os.listdir(location):
      #again iterate through the given location's every directory's list
      for image in os.listdir(location + each_dir):


        #open every images inside that directory in grayscale
        images = location + each_dir +'/' + image
        open_image = cv2.imread(images, 0)
        image_labels = each_dir.split('_')


        #give label to each character from 0 to 9 and ka to gya
        #digits label are from 0 to 9 while letters are 10 to 45
        if(len(image_labels) == 3):
          label = int(image_labels[::-1][1]) + 9
```

```
            elif(len(image_labels) == 2):
                label = int(image_labels[::-1][0])


            #first column will hold the label of the example and rest 1024 will hold
        pixel info.
            image_array = np.hstack([np.array(label),
        np.array(open_image).reshape(1024)])


            #append the image information into corresponding file
            with open(location[:len(location)-1] + '.csv', 'a', newline = '') as f:
                writer = csv.writer(f)
                writer.writerow(image_array)



        create_csv_data_file('train/')
        print('Done Creating trainset !!!!\n')
        create_csv_data_file('test/')
        print('Done Creating testset !!!!')
```

ii. Model Creation and Train on google colab

```
import numpy as np
import keras
print(keras.__version__)
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD
from keras.models import model_from_json
from keras.models import load_model
import matplotlib.pyplot as plt
import cv2
import time
from keras.regularizers import l1

plt.style.use('seaborn-whitegrid')

from keras import regularizers

model = Sequential()
model.add(Conv2D(32, (3, 3), activation = 'relu', input_shape = (32, 32, 1),
data_format = 'channels_last'))
model.add(Conv2D(32, (3, 3), activation='relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(46, activation='softmax'))

model.summary()

# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth

from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

train_link = 'https://drive.google.com/open?id=104DhX-7q-
6gVxM6I7EB8C5XTF0YQ5njr'
fluff, id = train_link.split('=')
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('train.csv')

test_link = 'https://drive.google.com/open?id=1zZTF2b6p8aJoAaPGSrvd04Y5f-
AP1VPD'
fluff, id = test_link.split('=')
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('test.csv')

def current_example(example_number, data_set):
    current_datafile = open(data_set + '.csv', 'r')
    current_dataset = current_datafile.readlines()
    current_datafile.close()

    inputs = np.zeros((len(example_number), 32, 32, 1))
    targets = np.zeros((len(example_number), 1))
    index = 0
```

```
    for each_number in example_number:
        each_set = current_dataset[each_number]
        all_values = each_set.split(',')

        #the first value is the label
        correct_label = int(all_values[0])

        #Normalize the pixel values in range of 0.01 - 1
        inputs[index] = np.asarray(all_values[1:], dtype = np.float32).reshape(32, 32, 1) /
255 * 0.99 + 0.01
        targets[index] = int(all_values[0])
        index += 1

    return(inputs, targets)

#prepare training and test data
np.random.seed(0)
num_exmp = np.linspace(0, 78199, 78200, dtype = np.int32)
np.random.shuffle(num_exmp)
(x_train, y_train) = current_example(num_exmp, 'train')
y_train = keras.utils.to_categorical(y_train, num_classes=46)

num_exmp = np.linspace(0, 13799, 13800, dtype = np.int32)
x_test, y_test = current_example(num_exmp, 'test')
y_test = keras.utils.to_categorical(y_test, num_classes=46)

#Compile the model
sgd = SGD(lr=0.01, decay = 1e-6, momentum=0.9, nesterov=True)

model.compile(loss = 'categorical_crossentropy', optimizer = sgd, metrics =
['accuracy'])

#Evaluate the model,
time1 = time.time()
tensorboard_callback = keras.callbacks.TensorBoard(log_dir = './cnn2', histogram_freq
= 0, write_graph = True, write_images = True)
history = model.fit(x_train, y_train, batch_size=32, epochs = 100, validation_split =
0.2, shuffle = True, callbacks = [tensorboard_callback])
time2 = time.time()
score = model.evaluate(x_test, y_test, batch_size=32, verbose = 1)
time3 = time.time()
print("Train time = ", (time2 - time1)/60, "min", "\nTest time = ", (time3 - time2)/ 60,
"min", "\n Test loss: ", score[0], " Test accuracy: ", score[1])

# list all data in history
print(history.history.keys())
# summarize history for accuracy
```

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

#model.save('cnn2.hdf5')
model_json = model.to_json()

with open("cnn2.json", "w") as json_file:
   json_file.write(model_json)

# serialize weights to HDF5
model.save_weights("cnn2.h5")



from google.colab import files
files.download("cnn2.h5")
files.download("cnn2.json")
```

iii. Image processing functions

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
def borders(here_img, thresh):
   size = here_img.shape
   check = int(10 * size[0] / 600)
   image = here_img[:]
   top, bottom = 0, size[0] - 1
#    plt.imshow(image)
#    plt.show()
   shape = size
```

```
#find the background color for empty column
bg = np.repeat(thresh, shape[1])
count = 0
for row in range(1, shape[0]):
    if  (np.equal(bg, image[row]).any()) == True:
        #print(count)
        count += 1
    else:
        count = 0
    if count >= check:
        top = row - check
        break

shape = image.shape
bg = np.repeat(thresh, shape[1])
count = 0
rows = np.arange(1, shape[0])
#print(rows)
for row in rows[::-1]:
    if  (np.equal(bg, image[row]).any()) == True:
        count += 1
    else:
        count = 0
    if count >= check:
        bottom = row + count
        break
#print(count)

#plt.imshow(here_img[top:bottom, :])
#plt.imshow(here_img[top:bottom, :])
#plt.show()

d1 = (top - 2) >= 0
d2 = (bottom + 2) < size[0]
```

```
    d = d1 and d2
    if(d):
        b = 2
    else:
        b = 0
    return (top, bottom, b)


def detect_text(main_image, gray_img, localized, bc):
        cimg = cv2.resize(localized, (30, 30))
        bordersize = 1
        nimg = cv2.copyMakeBorder(cimg, top=bordersize, bottom=bordersize,
left=bordersize, right=bordersize, borderType=cv2.BORDER_CONSTANT,
value=[255-bc, 0, 0])


        return main_image, nimg


def segmentation(bordered, thresh):
    try:
        shape = bordered.shape
        check = int(100 * shape[0] / 320)
        image = bordered[:]
        image = image[check:].T
        shape = image.shape
        #plt.imshow(image)
        #plt.show()

        #find the background color for empty column
        bg = np.repeat(255 - thresh, shape[1])
        #    fg = np.absrep(thresh, shape[1])
        bg_keys = []

        for row in range(1, shape[0]):
            if  (np.equal(bg, image[row]).all()):
                bg_keys.append(row)
```

```
#    print(bg_keys)
 if len(bg_keys) > 1:
    lenkeys = len(bg_keys) - 1
    new_keys = [bg_keys[1], bg_keys[-1]]

    if lenkeys == 1:
       if (new_keys - bg_keys) < (18 * shape[1]/ 100):
          return [bordered]



    #print(lenkeys)
    for i in range(1, lenkeys):
       if (bg_keys[i+1] - bg_keys[i]) > check:
          new_keys.append(bg_keys[i])
          #print(i)

    new_keys = sorted(new_keys)
    #print(new_keys)
    segmented_templates = []
    first = 0
    for key in new_keys[1:]:
       segment = bordered.T[first:key]
#          plt.imshow(segment)
#          plt.show()
       segmented_templates.append(segment.T)
       #show middle segments
#          plt.imshow(segment.T)
#          plt.show()
       first = key
    last_segment = bordered.T[new_keys[-1]:]
    segmented_templates.append(last_segment.T)

    #check if each segment shape is enough to do recognition
    final_segments = []
```

```
            for segment in segmented_templates:
                shape = segment.shape
                if shape[1] > (shape[0] / 4):
                    th_img = segment[check:]
                    text_color = thresh
        #             tb = borders(th_img, text_color)
                    lr = borders(th_img.T, text_color)
                    dummy = 0
                    template = segment[:, lr[0]+dummy:lr[1]-dummy]
                    final_segments.append(template)
                  #     plt.imshow(template)
                  #    plt.show()



            return(final_segments)
        else:
            return [bordered]
    except:
#         print('exception on segmentation')
        return [bordered]


def localize(main_image, gray_img, localized, bc, show):
    #open the template as gray scale image
        template = localized
        #print(template.shape)
        width, height = template.shape[::-1] #get the width and height
        #match the template using cv2.matchTemplate
        match=cv2.matchTemplate(gray_img,template,
cv2.TM_CCOEFF_NORMED)
        threshold = 0.8
        position = np.where(match >= threshold) #get the location of template in the
image
        for point in zip(*position[::-1]): #draw the rectangle around the matched
template
            cv2.rectangle(main_image, point, (point[0] + width, point[1] + height), (255
- bc, 0, bc ), 2)
```

```
        return main_image


    def preprocess(bgr_img):#gray image
        img = bgr_img[:]
        blur = cv2.GaussianBlur(img,(5,5),0)

    ret,th_img=cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH
    _OTSU) #converts black to white and inverse


        rows, cols = th_img.shape
        bg_test = np.array([th_img[i][i] for i in range(5)])
        if bg_test.all() == 0:
            text_color = 255
        else:
            text_color = 0
        #print('Process: Localization....\n')
        tb = borders(th_img, text_color)
        lr = borders(th_img.T, text_color)
        dummy = int(np.average((tb[2], lr[2]))) + 2
        template = th_img[tb[0]+dummy:tb[1]-dummy, lr[0]+dummy:lr[1]-dummy]


        plt.imshow(template)
        plt.show()
        #print("Process: Segmentation....\n")
        segments = segmentation(template, text_color)
        #print('Process: Detection.....\n')
        return segments, template, th_img, text_color
```

iv. Camera function

```
import cv2
import time
from recognition import recognition

def camera(flag):
    choice = print("Click spacebar for photo and anything else for video.\n")
```

```
orig = 1
cap = cv2.VideoCapture(0)
tr = 0.1
br = 0.8
lc = 0.1
rc = 0.8
f = 0

while(flag):
    ret, frame = cap.read()
    if ret:
        #key event
        s = cv2.waitKey(2) & 0xFF

        if(chr(s) == 'x'):
            f = -1
        if(chr(s) == 'z'):
            f = 1

        if(chr(s) == 'a'):
            tr = tr + 0.1 * f
        if(chr(s) == 'd'):
            br = br + 0.1 * f
        if (chr(s) == 's'):
            lc = lc + 0.1 * f
        if (chr(s) == 'w'):
            rc = rc + 0.1 * f


        s_x, s_y = np.shape(frame)[0] * tr, np.shape(frame)[1] * lc
        e_x, e_y = np.shape(frame)[1] * br, np.shape(frame)[0] * rc
        s_x, s_y = np.int32(s_x), np.int32(s_y)
        e_x, e_y = np.int32(e_x), np.int32(e_y)

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        ogray = gray[:]
        gray = gray[s_y:e_y, s_x:e_x]
        #original = frame[s_y:e_y, s_x:e_x]

        if (s == 32): #space to capture image and do recognition
            time1 = time.time()
            plt.imshow(frame)
            plt.show()
            recognition(gray, 'show')
            print("In %f" %(time.time()-time1), 'sec')
        if (s == 13): #enter to do realtime recognition
            orig = 0
            cv2.destroyWindow('Project DCR')
```

```
            print("Doing RT...")
            recognition(ogray, 'no')


        else:
            if(orig != 0):
                show = frame[:]
                text = "Press 'space' to take a photo and 'enter' to do realtime(slow)."
                text1 = "Make sure the character is inside rectangle."
                text2 = "a/s/d/w for change rectangle and z/x for inc/dec."
            cv2.putText(show, text1, (15, 50), cv2.FONT_HERSHEY_COMPLEX,
            0.75, (0, 100, 200))
            cv2.putText(show, text2, (0, np.shape(frame)[0] - 10),
            cv2.FONT_HERSHEY_COMPLEX, 0.65, (50, 20, 255))
            cv2.rectangle(show, (s_x, s_y), (e_x, e_y), (0, 255, 0), 2)
            cv2.putText(show, text, (15, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (15,
            0, 255), lineType=cv2.LINE_AA)
            cv2.imshow('Project DCR', show)


        else:
            print('Trying.....\n')
            continue


        if s == 27:
            break
    cap.release()
    cv2.destroyAllWindows()
```

v. Main file

```
    from recognition import recognition
    import cv2
    import matplotlib.pyplot as plt
    import time
    from video_test import camera


    try:
        test = input('Please enter the image directory with name.\n')
        test = cv2.imread(test, 0)
        plt.imshow(cv2.cvtColor(test, cv2.COLOR_GRAY2RGB))
```

```
    plt.xticks([])
    plt.yticks([])
    plt.show()
    time1 = time.time()
    in_img = recognition(test, 'show')
    print("In %f" %(time.time()-time1), 'sec')
except:
    print("Image not found now turning to video mode.\n")
#    camera(True)
    try:
        camera(True)
    except:
        print('Something is wrong. Try with more stable, less noise and clear picture.\n')
```

vi. Prediction function

```
import numpy as np
from keras.models import model_from_json
from keras.models import load_model

def prediction(img):
    # load json and create model
    json_file = open('cnn2\cnn2.json', 'r')

    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)

    # load weights into new model
    loaded_model.load_weights("cnn2\cnn2.h5")
    #print("Loaded model from disk")

    loaded_model.save('cnn.hdf5')
    loaded_model=load_model('cnn.hdf5')
```

```
    characters =
'०,१,२,३,४,५,६,७,८,९,क,ख,ग,घ,ङ,च,छ,ज,झ,ञ,ट,ठ,ड,ढ,ण,त,थ,द,ध,न,प,फ,ब,भ,म,य,र
,ल,व,श,ष,स,ह,क्ष,त्र,ज्ञ'
    characters = characters.split(',')

    x = np.asarray(img, dtype = np.float32).reshape(1, 32, 32, 1) / 255

    output = loaded_model.predict(x)
    output = output.reshape(46)
    predicted = np.argmax(output)
    devanagari_label = characters[predicted]
    success = output[predicted] * 100

    return devanagari_label, success
```