

IDENTIFYING HANDWRITTEN LETTERS WITH DEEP LEARNING

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

K. UDAY SAI GANESH (Reg.No - 39110459)

K.V.S. ADITYA REDDY (Reg.No – 39110463)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade “A” by NAAC
JEPPIAAR NAGAR, RAJIV GANDHISALAI,
CHENNAI - 600119

APRIL 2023



SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BONAFIDE CERTIFICATE**

This is to certify that this Project Report is the bonafide work of **K. UDAY SAI GANESH** (Reg.No - **39110459**) and **K.V.S. ADITYA REDDY**(Reg.No - **39110463**) who carried out the Project Phase-2 entitled "**IDENTIFYING HANDWRITTEN LETTERS WITH DEEP LEARNING**" under my supervision from January 2023 to April 2023.

Internal Guide

Ms. K. ANITA DAVAMANI, Asst. Professor

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.



Submitted for Viva voce Examination held on 20.04.2023

Internal Examiner

External Examiner

DECLARATION

I, **K. UDAY SAI GANESH**(Reg.No- 39110459), hereby declare that the Project Phase-2 Report entitled “**IDENTIFYING HANDWRITTEN LETTERS WITH DEEP LEARNING**” done by me under the guidance of **Ms. K.ANITA DAVAMANI** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 20.04.2023

PLACE: Chennai

A handwritten signature in blue ink that reads "Uday Sai Ganesh". The signature is written in a cursive, flowing style.

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph. D, Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms. K.ANITA DAVAMANI**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase-2 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Two crucial technological areas that are uniting the world are machine translation and natural language processing. The barrier to communication with others who speak a different language is removed via machine translation. Finding the word's native language and then translating it into the target language are the foundations of machine translation. If each letter in the word can be recognized, determining the language of the word may be simpler. We now arrive at "Character Recognition." This thesis focuses on evaluating the available techniques for handwritten character identification using deep learning and those techniques to recognize handwritten characters. The intriguing thing about machine learning is that the computers that are programmed to interpret, say, Chinese, can also be programmed to translate, say, Hindi or Telugu. This is a key characteristic that the thesis seeks to investigate because it is so potent. Beyond just detecting the characters, modern technology can also translate such characters between different languages. Depending on the character, the machine learning model created for this thesis can accurately identify the character or alphabets with a 70–90% rate of success. All data is saved and documented in datasets for further sharing and project usage. A large database of handwritten numbers is needed to train different methods for analyzing images the MNIST database, which is frequently used. The database is commonly used for developing and evaluating machine learning algorithms. It was created by recombining samples from the initial runs. 2017 saw the publication of EMNIST, an expanded dataset that is identical to MNIST and contains 240,000 training photos and 40,000 testing images of handwritten numbers and characters.

LIST OF FIGURES

FIGURE NO	FIGURE NAME	Page No.
4.1	Architecture diagram	19
4.2	CNN Architecture diagram	21
4.3	Number of data elements	28
4.4	Raw data images	28
4.5	Trained model validation	29
4.6	Final output	29
4.7	Performance measure	30

LIST OF ABBREVIATIONS

MNIST	Modified National Institute of Standards and Technology
CNN	Convolutional Neural Network
KNN	K-Nearest Neighbors
RNN	Recurrent Neural Networks
ReLu	Rectified Linear Unit

Chapter No.	TITLE		Pg. No
	LIST OF FIGURES		viii
	LIST OF TABLES		
	LIST OF ABBREVIATIONS		
1	CHAPTER 1		
	1.1	INTRODUCTION	1
	1.2	PROBLEM STATEMENT	4
	1.3	PROPOSED SOLUTION	4
	1.4	PROGRAMMING LANGUAGE & LIBRARIES	5
2	CHAPTER 2		
	2.1	LITERATURE REVIEW	8
	2.2	OVERVIEW OF EXISTING RESEARCH	9
	2.3	SUMMARY OF CONCEPTS & THEORIES ABOUT THE PROJECT	10
	2.4	JUSTIFICATION OF PROJECT'S IMPORTANCE	11
3	CHAPTER 3		
	3.1	DATA COLLECTION & PRE-PROCESSING	12
	3.2	DATA PRE-PROCESSING METHODS	14
4	CHAPTER 4		
	4.1	MODEL ARCHITECTURE & TRAINING	18
	4.2	DESCRIPTION OF PROPOSED MODEL	19
	4.3	DETAILING HYPER PARAMETERS & OPTIMIZING THE ALGORITHM	21
	4.4	DISCUSSING TRAINING PROCESS & EVALUATION METRICS	23
5	CHAPTER 5		
	5.1	DEVELOPMNET SETUP	26

	5.2	DEPLOYMENT SETUP	26
	CHAPTER 6		
	6.1	RESULTS & ANALYSIS	2
7	CHAPTER 7		
	7.1	FUTURE WORK & CONCLUSION	30
	7.2	ISSUES FACED IN RESEARCH	30
	7.3	ISSUES FACED IN IMPLEMENTATION	31
	REFERENCES		
	APPENDIX		
	A. SOURCE CODE		33
	B. SCREENSHOTS		35
	C. RESEARCH PAPER		42

CHAPTER -1

1.1 INTRODUCTION

Intelligent image analysis is an appealing research area in Artificial Intelligence and crucial for a variety of present open research difficulties. Handwritten digits recognition is a well-researched subarea within the field that is concerned with learning models to distinguish pre-segmented handwritten digits. It is one of the most important issues in data mining, machine learning, pattern recognition along with many other disciplines of artificial intelligence. The main application of machine learning methods over the last decade has determined efficacious in conforming decisive systems which are competing to human performance and which accomplish far improved than manually written classical artificial intelligence systems used in the beginnings of optical character recognition technology. However, not all features of those specific models have been previously inspected. A great attempt of research worker in machine learning and data mining has been contrived to achieve efficient approaches for approximation of recognition from data. In twenty first Century handwritten digit communication has its own standard and most of the times in daily life are being used as means of conversation and recording the information to be shared with individuals. One of the challenges in handwritten characters recognition wholly lies in the variation and distortion of handwritten character set because distinct community may use diverse style of handwriting, and control to draw the similar pattern of the characters of their recognized script. Identification of digit from where best discriminating features can be extracted is one of the major tasks around digit recognition system. To locate such regions different kind of region sampling techniques are used in pattern recognition. The challenge in handwritten character recognition is mainly caused by the large variation of individual writing styles. Hence, robust feature extraction is very important to improve the performance of a handwritten character recognition system. Nowadays handwritten digit recognition has obtained lot of concentration around pattern recognition system sowing to its application in diverse fields. In next days, character recognition system might serve as a cornerstone to initiate paperless surroundings by digitizing and processing existing paper documents. Handwritten digit dataset is vague in nature because there may not always be sharp and perfectly straight lines. The main goal in digit recognition

is feature extraction is to remove the redundancy from the data and gain a more effective embodiment of the word image through a set of numerical attributes. It deals with extracting most of the essential information from image raw data. In addition, the curves are not necessarily smooth like the printed characters. Furthermore, characters dataset can be drawn in different sizes and the orientation which are always supposed to be written on a guideline in an upright or downright point. Accordingly, an efficient handwritten recognition system can be developed by considering these limitations. It is quite exhausting that sometimes to identify handwritten characters as most of the human beings can't even recognize their own written scripts. Hence, there exists constraint for a writer to write apparently for recognition of handwritten documents. Before revealing the method used in conducting this research, software engineering module is first presented. Pattern recognition along with Image processing plays compelling role around handwritten character recognition. The study describes numerous types of classification of feature extraction techniques like structural feature-based methods, statistical feature-based methods, and global transformation techniques. Statistical approaches are established on planning of how data are selected. It utilizes the information of the statistical distribution of pixels in the image. The paper provided SVM based offline handwritten digit recognition system. Authors claim that SVM outperforms in the experiment. Experiment is carried out on NIST SD19 standard dataset. The study provides the conversion of handwritten data into electronic data, nature of handwritten characters and the neural network approach to form machine competent of recognizing handwritten characters. The study addresses a comprehensive criterion of handwritten digit recognition with various state of the art approaches, feature representations, and datasets. However, the relationship of training set size versus accuracy/error and the dataset-independence of the trained models are analyzed. The paper presents convolution neural networks into the handwritten digit recognition research and describes a system to find any digit.

Handwriting recognition of characters has been around since the 1980s. The task of handwritten digit recognition, using a classifier, has great importance and use such as – online handwriting recognition on computer tablets, recognize zip codes on mail for postal mail sorting, processing bank check amounts, numeric entries in

forms filled up by hand (for example - tax forms) and so on. There are different challenges faced while attempting to solve this problem. The handwritten digits are not always of the same size, thickness, or orientation and position relative to the margins. My goal was to implement a pattern classification method to recognize the handwritten digits provided in the MNIST data set of images of handwritten digits (0-9). The data set used for our application is composed of 300 training images and 300 testing images and is a subset of the MNIST data set [1] (originally composed of 60,000 training images and 10,000 testing images). Each image is a 28 x 28 grayscale (0-255) labeled representation of an individual digit. The general problem we predicted we would face in this digit classification problem was the similarity between the letters.

Researchers have been investigating various methods for testing handwritten character recognition over the past few decades. Learning becomes harder as our assignments become more challenging. We employ deep neural networks, which is suitable. Make a neural network design with higher performance metrics and train it. Deep learning and neural networks enable us to train the model we independently constructed to learn for itself from our experiences.

Convolutional Neural Network is one of the methods of Deep Learning Algorithms. Characters present in the image are successfully recognized by the CNN classifier. Convolutional layers for feature extraction and fully linked layers followed by a soft-max layer for classification make up the architecture of conventional CNN classifiers. One of the effective feature extractors is CNN. Data exploration is a methodology that integrates methods from machine learning, statistics, and computer systems to find patterns in enormous volumes of data. The neurons in CNN's architecture are arranged in layers. This design consists of an input layer, many hidden levels, and an output unit. Typically, networks with a lot of hidden layers are referred to as deep neural networks. The neurons in the hidden layers of CNN are connected to only a fraction of the input space produced by the preceding layer, as opposed to an entirely important factor like Multi Layered Perceptron (MLP) networks. The inversion process produces an extracted feature when the convolution kernel passes across the input matrix for the layer, adding to the input of the following layer. Then, further layers like normalizing, pooling, and entirely linked levels are introduced. In conclusion, handwritten digit recognition is

a significant, quickly developing topic with a wide range of real-world uses. We may anticipate significant advancements in precision and performance as deep learning and computer vision technologies keep evolving, making them even more beneficial in a variety of fields.

1.2 PROBLEM STATEMENT

Handwritten letter recognition is the task of correctly identifying handwritten letters from a given dataset. Deep learning is a promising approach for solving this problem, as it has been shown to achieve state-of-the-art results on a variety of image recognition tasks. The problem statement for handwritten letter recognition using deep learning would involve developing a model that can accurately classify images of handwritten letters into their corresponding alphabets (A-Z). The model should be able to handle variability in writing styles, rotations, and scaling of the letters, while also generalizing well to new, unseen data. The goal is to achieve high accuracy on a test set of images, while also minimizing the model's complexity and training time. Additionally, the model should be able to distinguish between uppercase and lowercase letters, and handle recognition of both printed and cursive handwriting styles.

1.3 PROPOSED SOLUTION

There are several drawbacks of using traditional machine learning algorithms for handwritten letters recognition. Traditional machine learning algorithms require manual feature engineering, where domain experts manually select and engineer features that are relevant to the problem at hand. This can be time-consuming and often requires a deep understanding of the problem domain. These algorithms may struggle to handle variability in writing styles, rotations, and scaling of the letters, making it difficult to achieve high accuracy on a wide range of inputs. These algorithms may not generalize well to new, unseen data, especially if the data is significantly different from the training data. Many of the machine learning algorithms are often sensitive to noise in the data, which can negatively impact their accuracy and performance. Traditional machine learning algorithms can struggle to handle large datasets, as the training process can be time-consuming and computationally expensive.

Since some deep learning approaches, such as convolutional neural networks (CNNs), have shown to overcome many of these drawbacks and achieve state-of-the-art results on handwritten letters recognition tasks, we are developing the deep learning model using CNN. We here are not using MNIST dataset ,since we feel it has not much more data and many of the algorithms are trained on this dataset. We have collected data from online resources we found and this project focuses not only on algorithm but also data processing which is the most important factor in any of the machine learning models.

1.4 PROGRAMMING LANGUAGE & LIBRARIES

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently.

1.4.1 Features

1.4.1.1 Interpreted

There are no separate compilation and execution steps like C and C++.Directly run the program from the source code. Internally, Python converts the source code into an intermediate form called bytecodes which is then translated into native language of specific computer to run it. No need to worry about linking and loading with libraries, etc.

1.4.1.2 Platform Independent

Python programs can be developed and executed on multiple operating system platforms. Python can be used on Linux, Windows, Macintosh, Solaris and many more.

1.4.1.3 High Level Language

In Python, no need to take care about low-level details such as managing the memory used by the program.

1.4.1.4 *Simple*

Closer to English language and easy to Learn. More emphasis on the solution to the problem rather than the syntax.

1.4.1.5 *Embeddable*

Python can be used within C/C++ program to give scripting capabilities for the program's users.

1.4.1.6 *Rich Library Support*

The Python Standard Library is very vast. Known as the “batteries included” philosophy of Python. It can help do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, email, XML, HTML, WAV files, cryptography, GUI and many more. Besides the standard library, there are various other high-quality libraries such as the Python Imaging Library which is an amazingly simple image manipulation library.

1.4.1.7 *Advantages*

- Presence of third-party modules.
- Extensive support libraries (NumPy for numerical calculations, Pandas for data analytics etc).
- Open source and community development.
- Versatile, Easy to read, learn and write.
- User-friendly data structures
- High-level language.
- Object-oriented language
- Portable and Interactive
- Ideal for prototypes – provide more functionality with less coding.
- (IoT)Internet of Things Opportunities.
- Interpreted Language.
- Portable across Operating systems

1.4.1.8 *Applications*

- GUI based desktop applications.
- Graphic design, image processing applications, Games, and Scientific/computational Applications

- Web frameworks and applications
- Enterprise and Business applications
- Operating Systems
- Education
- Database Access
- Language Development
- Prototyping

In conclusion, python is widely used in the field of machine learning (ML) due to its simplicity, flexibility, and powerful libraries. ML frameworks such as TensorFlow, Keras, and PyTorch are written in Python and provide a high-level interface for building and training ML models. Python's extensive scientific computing libraries such as NumPy, Pandas, and SciPy make it easy to manipulate and analyze data, a key component of ML. Python's popularity and large community support have led to the development of many specialized ML libraries and tools, including scikit-learn, XGBoost, and NLTK. Moreover, Python's ability to integrate with other technologies and languages makes it a powerful tool for building end-to-end ML solutions. Overall, Python's importance in ML cannot be overstated, as it has helped make the development and deployment of ML models accessible to a wider audience of developers and researchers.

CHAPTER 2

2.1 LITERATURE REVIEW

Title: Recognition of Handwritten Digit using Convolutional Neural Network in Python with Tensor flow and Comparison of Performance for Various Hidden Layers.[3]

Author: Fathima Siddique; Shadman Sakib

In this paper, to find the performance of CNN hidden layers, CNN as well as MNIST dataset real time input. have used. The network is trained using stochastic gradient descent and the backpropagation algorithm.

Scope of this paper is to adding feature like GUI to take real time input.

Title: Analogizing time Complexity of KNN and CNN in Recognizing Handwritten Digits.[9]

Author: Tanya makkar; Ashwani kumar Dubey

In this paper comparison of accuracy of KNN as well as CNN have done. In this paper MNIST data set has used for providing samples to the system.

Scope of this paper is to choosing CNN because it produces high accuracy then KNN and adding GUI for real time input.

Title: Telugu handwritten character recognition using adaptive and static zoning methods. [10]

Author: Sangula Durga Prasad; Yashwanth Kanduri

In this paper they presented two methods for this purpose. First method is based on Genetic Algorithm and uses adaptive zoning topology with extracted geometric features. In second method, zoning is done in static way and uses distance, density based features. In both the contexts, they used K-Nearest Neighbor (KNN) algorithm for classification purpose.

2.2 OVERVIEW OF EXISTING RESEARCH

Handwritten letter recognition is a rapidly evolving field with a long history of research. The study involves the development of computer algorithms that can automatically identify and interpret handwritten characters. The primary challenge in this field is the high variability in handwriting styles, which makes it difficult for algorithms to distinguish between different characters accurately.

Early approaches to handwritten letter recognition were rule-based or template matching techniques. However, these methods were limited by their inability to handle variability in handwriting. In recent years, deep learning techniques have shown significant promise in improving the accuracy of handwritten letter recognition. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have been particularly successful in this regard.

One of the major issues in handwritten letter recognition is the lack of high-quality training data. While there are several datasets available for this purpose, they are often small and may not represent the variability in handwriting styles. Several researchers have attempted to address this issue by creating synthetic datasets or using generative adversarial networks (GANs) to generate new training data.

Another area of research in handwritten letter recognition is the development of new features and preprocessing techniques that can improve the accuracy of recognition algorithms. For example, some researchers have investigated the use of stroke-based features, which capture the structure and direction of each stroke in a handwritten character.

There have also been efforts to develop new evaluation metrics for assessing the performance of handwritten letter recognition algorithms. One such metric is the writer-independent error rate (WER), which measures the error rate when recognizing characters written by individuals who are not in the training set.

Despite the progress made in this field, there are still several challenges that need to be addressed. For example, the recognition of cursive handwriting is still a major challenge for algorithms. Additionally, there is a need for more robust algorithms that can handle noise and other forms of degradation in handwritten documents. In conclusion, handwritten letter recognition is a field that has seen significant advancements in recent years, with the introduction of deep learning

techniques and the development of new evaluation metrics and preprocessing techniques. However, there are still several challenges that need to be addressed before this technology can be widely deployed in practical applications.

2.3 SUMMARY OF CONCEPTS & THEORIES ABOUT THE PROJECT

Firstly I examined a research paper regarding the Handwritten letters recognition. Handwritten Character Recognition using Star-Layered Histogram Features Stephen Karungaru, Kenji Terada and Minoru Fukumi, In this method, a character recognition method using features extracted from a star layered histogram is presented and trained using neural networks. After several image preprocessing steps, the character region is extracted. Its contour is then used to determine the center of gravity (COG). This CoG point is used as the origin to create a histogram using equally spaced lines extending from the CoG to the contour. The first point the line touches the character represents the first layer of the histogram. If the line extension has not reached the region boundary, the next hit represents the second layer of the histogram. This process is repeated until the line touches the boundary of the character's region. After normalization, these features are used to train a neural network.

Throughout the project we implemented the model using CNN. The core concept of CNNs is the use of convolutional layers that learn and detect local features from the input images. In CNNs, the input image is processed through a series of convolutional and pooling layers. The convolutional layers apply a set of filters to the input image to learn and detect features such as edges, corners, and textures. The pooling layers then down sample the feature maps generated by the convolutional layers, reducing their spatial size while retaining important information. This process is repeated multiple times, resulting in a hierarchy of features that represent more complex and abstract information about the input image.

In addition to convolutional and pooling layers, CNNs also typically include fully connected layers that learn to classify the input image based on the learned features. The final layer of the network typically outputs a probability distribution over the possible classes, allowing the network to make predictions about the input image. CNNs have demonstrated state-of-the-art performance in a wide

range of computer vision tasks, due in part to their ability to learn and detect relevant features from the input images. The use of GPUs and other hardware accelerators has also enabled the efficient training of large-scale CNNs on massive datasets.

2.4 JUSTIFICATION OF PROJECT's IMPORTANCE

Handwritten recognition using CNN models is needed because of the widespread use of handwritten documents and the need for efficient and accurate automated processing of these documents. Handwritten text is still widely used in various domains such as banking, legal, education, and healthcare, and the ability to automatically recognize and transcribe this text can improve the efficiency and accuracy of many processes.

Traditional methods of handwritten recognition involve manually designing and extracting features from the handwritten images, which is a time-consuming and error-prone process. CNN models, on the other hand, can automatically learn and extract relevant features from the input images, eliminating the need for manual feature engineering. Moreover, CNN models have shown significant improvements in the accuracy of handwritten recognition compared to traditional methods. CNN models can handle the variability in handwriting styles and can recognize characters even with noise, distortion, and other forms of degradation. The implementation of handwritten recognition using CNN models can also be beneficial in various applications such as digit recognition for postal services, signature recognition for banking and legal documents, and handwriting recognition for education and healthcare purposes.

In summary, implementing handwritten recognition using CNN models is needed to improve the accuracy and efficiency of automated processing of handwritten documents, eliminate the need for manual feature engineering, and enable various applications in different domains.

CHAPTER 3

3.1 DATA COLLECTION & PRE-PROCESSING

Data collection is a crucial step in the machine learning process as it provides the necessary information for training and testing models. The data collection process involves the following steps:

Determine the problem statement: Define the problem to be solved and identify the relevant data needed to solve it.

Identify the data sources: Identify potential sources of data, such as existing datasets, APIs, or manual collection methods.

Collect the data: Collect data from the identified sources, either manually or through automated means.

Preprocess the data: Clean the data by removing irrelevant or duplicated data, filling in missing values, and transforming the data into a suitable format for analysis.

Label the data: Label the data with relevant categories or outcomes, such as class labels or numerical values.

Split the data: Split the data into training, validation, and test sets to enable model development and evaluation.

Store the data: Store the data in a suitable format, such as CSV, JSON, or database format, for easy access and analysis.

Ensure data privacy and security: Ensure that the collected data is secure and that the privacy of individuals is maintained by anonymizing the data or obtaining consent where necessary.

Effective data collection is critical for the success of machine learning models, as it enables the models to learn from real-world data and make accurate predictions or classifications.

Datasets used for handwritten letters recognition typically consist of a large number of handwritten letter images, along with corresponding labels that identify

the correct letter for each image. Some of the commonly used datasets for handwritten letters recognition are:

MNIST: This is a widely used dataset that consists of 70,000 images of handwritten digits from 0 to 9. Each image is a 28x28 grayscale image.

EMNIST: This dataset is an extension of MNIST and consists of 240,000 images of handwritten letters and digits. The letters are from A to Z, both uppercase and lowercase, and the digits are from 0 to 9.

NIST: This dataset consists of images of handwritten letters and digits collected by the National Institute of Standards and Technology (NIST) for research purposes.

CEDAR: The CEDAR dataset consists of over 6,000 images of handwritten English letters, including both uppercase and lowercase letters.

IAM: The IAM dataset consists of handwritten English words and sentences, including both cursive and printed writing styles.

Chars74K: This dataset consists of 74,000 images of handwritten characters from various languages, including English, Arabic, and Chinese.

These datasets are widely used for training and testing machine learning models for handwritten letters recognition. They provide a diverse set of images with varying writing styles, noise, and distortions, enabling the development of robust models that can handle real-world scenarios. Since, all the above mentioned datasets are used by many in their previous works we implemented our dataset using references from them and contains more than 3 lakh images which we thought is really a good count for making the model more accurate.

Data preprocessing refers to the cleaning, transformation, and preparation of raw data before it is used for machine learning applications. The main goal of data preprocessing is to improve the quality and usability of the data by removing errors, filling in missing values, and transforming the data into a format that is suitable for analysis. Data preprocessing is an important step in machine learning because the quality and accuracy of the resulting model are directly influenced by the quality of the input data. Poor quality data can lead to inaccurate or unreliable predictions, and in some cases, may even render the model unusable.

3.2 DATA PRE-PROCESSING STEPS

In machine learning, training data and testing data are two key components used to develop and evaluate machine learning models.

Training data refers to a set of input data and corresponding output labels that are used to train a machine learning model. During the training process, the model learns the underlying patterns and relationships between the input data and output labels. The goal of training is to optimize the model's parameters or weights, so it can accurately predict the correct output label for new, unseen data.

Testing data, on the other hand, is a set of input data and output labels that are used to evaluate the performance of a trained machine learning model. The testing data is separate from the training data, and the model has not seen these data points during the training process. The goal of testing is to determine how well the model generalizes to new, unseen data.

To ensure that the model performs well on new data, the testing data should be representative of the real-world data that the model will encounter. The testing data should also be large enough to provide a reliable estimate of the model's performance. In order to prevent overfitting, it's important to use separate sets of training and testing data. Overfitting occurs when a model learns the training data too well and becomes too specific to that data. This can lead to poor performance on new, unseen data. By using separate sets of training and testing data, we can evaluate the model's ability to generalize to new data and prevent overfitting. In summary, training data is used to teach the model how to make accurate predictions, while testing data is used to evaluate the model's performance and ensure that it generalizes well to new data.

3.2.1 Thresholding

Thresholding is a common technique used in image processing and computer vision, including in handwritten letters recognition, to segment an image into foreground and background. The basic idea of thresholding is to convert a grayscale image into a binary image by setting a threshold value, and any pixel value above that threshold is considered part of the foreground, and any value below the threshold is considered part of the background.

In the context of handwritten letters recognition, thresholding can be used to separate the handwritten letters from the background noise, making it easier to extract features and classify the letters. For example, after applying thresholding to an image of a handwritten letter, we can count the number of pixels in the foreground to get an estimate of the letter's size, or we can compute the horizontal and vertical projections of the letter to obtain information about its shape and orientation. There are several methods for thresholding images, including global thresholding, adaptive thresholding, and Otsu's method. Global thresholding uses a fixed threshold value for the entire image, while adaptive thresholding adjusts the threshold value for different parts of the image based on local characteristics. Otsu's method is a widely used thresholding technique that automatically computes an optimal threshold value based on the distribution of pixel intensities in the image. Thresholding can be a useful preprocessing step in handwritten letters recognition, particularly when dealing with noisy or complex images. However, the effectiveness of thresholding depends on the quality of the input image and the specific characteristics of the handwritten letters being analyzed. For our project the function `threshold()` from `cv2` library is used for this preprocessing technique.

3.2.2 One-hot Encoding

Thresholding can be a useful preprocessing step in handwritten letters recognition, particularly when dealing with noisy or complex images. However, the effectiveness of thresholding depends on the quality of the input image and the specific characteristics of the handwritten letters being analyzed.

One hot encoding is commonly used in machine learning algorithms that require numeric input, such as neural networks and decision trees. It allows us to represent categorical variables as numerical data, making it easier for algorithms to process the data and learn relationships between the variables. However, one hot encoding can result in high-dimensional data, particularly for variables with many categories. In such cases, it may be necessary to reduce the dimensionality of the data through techniques such as feature selection or dimensionality reduction. In this project, we will use `to_categorical()` function imported from `keras` library.

3.2.3 Gaussian Blur

Gaussian blur is a commonly used technique in deep learning for image processing, particularly in computer vision applications. The basic idea of Gaussian blur is to smooth an image by convolving it with a Gaussian filter, which is a bell-shaped curve that assigns weights to each pixel in the image based on its distance from the center pixel.

The Gaussian blur operation is performed by sliding a kernel (i.e., the Gaussian filter) over the image and computing the weighted average of the pixel values within the kernel. The resulting smoothed image has reduced noise and sharp edges, which can be useful for improving the accuracy of object detection and recognition algorithms. In deep learning, Gaussian blur is often used as a preprocessing step for input images before they are fed into a neural network. This can help to reduce the impact of noise and other artifacts in the image and improve the overall performance of the network. Gaussian blur can also be used as a data augmentation technique in deep learning, where it is applied to randomly selected images in the training dataset. By adding random variations to the input data, data augmentation can help to prevent overfitting and improve the generalization ability of the network. Overall, Gaussian blur is a useful technique in deep learning for image processing and data augmentation, and can help to improve the accuracy and robustness of neural network models. By using the `GuassainBlur()` function from `cv2` library we successfully implemented this preprocessing step.

3.2.4 Normalization

Normalization is a common technique used in machine learning to rescale input data to a standard range. The main idea behind normalization is to transform the input data so that it has a mean of zero and a standard deviation of one, or to scale it to a specific range of values, such as between 0 and 1 or -1 and 1.

The purpose of normalization is to ensure that the input data has consistent and comparable scales, which can help to improve the performance of machine learning algorithms. For example, if one feature has a much larger range of values than another feature, it can dominate the learning process and lead to biased or inaccurate results.

There are several common normalization techniques used in machine learning, includes Z-score normalization, Min-max normalization, Decimal scaling normalization. Normalization can be applied to different types of data, including continuous and categorical variables, and can be used as a preprocessing step before training a machine learning model. However, it is important to note that normalization can also have drawbacks, such as increasing the computational complexity of the model and reducing the interpretability of the results.

CHAPTER 4

4.1 MODEL ARCHITECTURE & TRAINING

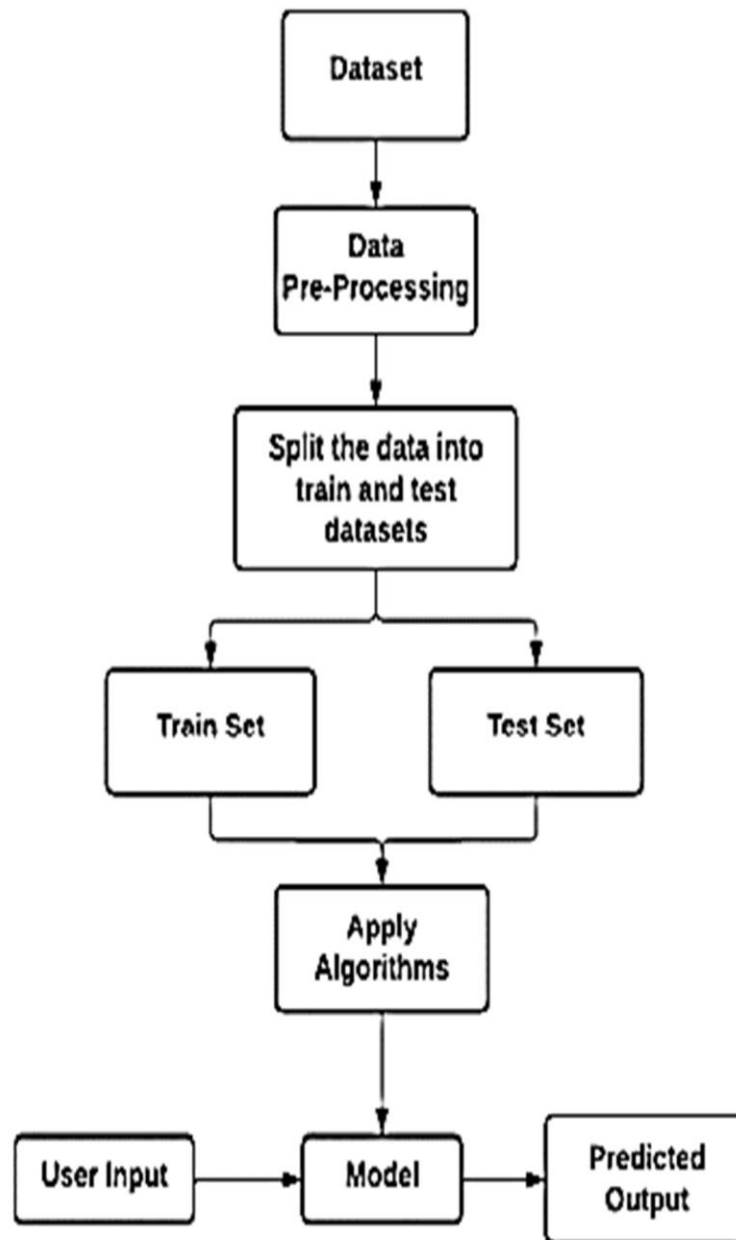


Fig 4.1 – Architecture diagram

4.2 DESCRIPTION OF PROPOSED MODEL

A Convolutional Neural Network (CNN) is a type of neural network that is commonly used for image processing and computer vision applications. The basic architecture of a CNN consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

The convolutional layer is the core building block of a CNN, and it is responsible for learning feature maps that capture the patterns and structures in the input image. In this layer, a set of filters (or kernels) are applied to the input image to produce a set of feature maps. Each filter learns to detect a specific feature, such as edges, corners, or shapes, by convolving the filter with the input image and computing a dot product.

The pooling layer is used to downsample the feature maps and reduce the dimensionality of the input data. This layer typically uses a max pooling or average pooling operation to extract the most important features from each feature map and discard the rest.

The fully connected layer is responsible for classifying the input image based on the learned features. In this layer, the feature maps are flattened into a one-dimensional vector, and a series of fully connected neurons are used to predict the class label of the input image.

The overall training process of a CNN involves feeding a large dataset of labeled images into the network, adjusting the weights of the filters using backpropagation and gradient descent, and evaluating the performance of the model on a validation set. The main objective of the training process is to minimize the loss function, which measures the difference between the predicted output of the model and the true output. In addition to the basic architecture of a CNN, there are several advanced techniques that can be used to improve its performance, such as data augmentation, regularization, and transfer learning. Data augmentation involves applying random transformations to the input images, such as rotations, flips, and zooms, to increase the size and diversity of the training dataset. Regularization techniques, such as L1 and L2 regularization, are used to prevent overfitting and improve the generalization ability of the model. Transfer learning involves using a

pre-trained CNN model as a starting point for a new task, and fine-tuning the model on a smaller dataset to adapt it to the new task.

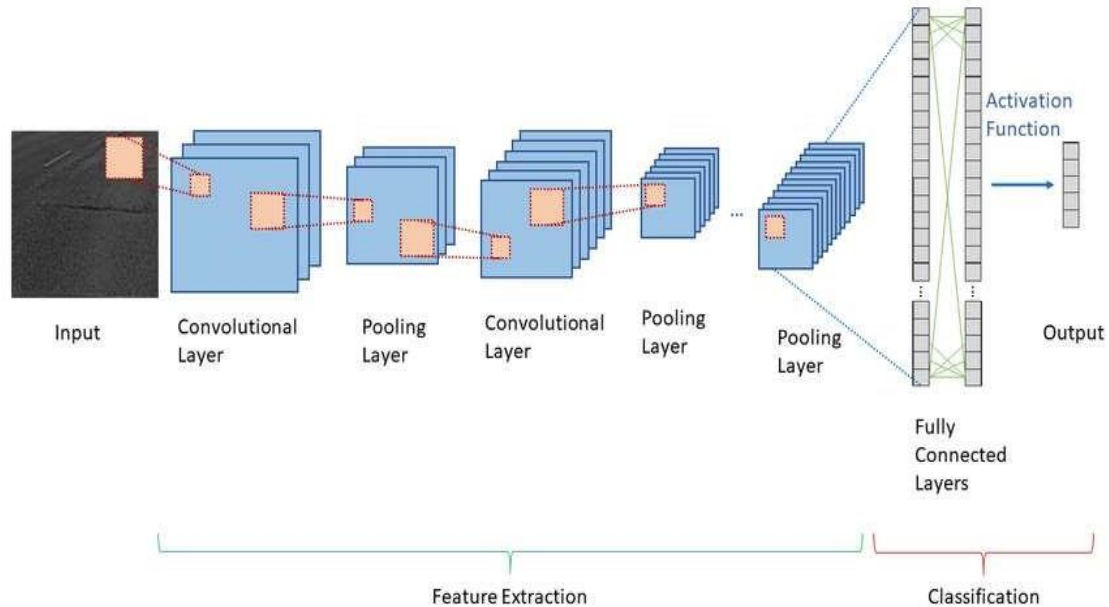


Fig 4.2 – CNN Architecture diagram

Convolutional Layers: The first layer in a CNN is typically a convolutional layer, which performs a convolution operation on the input image using a set of learnable filters. Each filter produces a feature map that highlights a particular aspect of the input image, such as edges or corners.

Pooling Layers: After each convolutional layer, a pooling layer is often used to downsample the feature maps and reduce their spatial size. This helps to reduce the number of parameters in the network and improve its computational efficiency.

Activation Functions: Nonlinear activation functions such as ReLU (Rectified Linear Unit) are applied to the output of each layer to introduce nonlinearity and enable the network to learn more complex patterns.

Fully-Connected Layers: The final layers in a CNN are typically fully-connected layers, which are used to map the high-level features learned by the previous layers to the output classes. These layers use a softmax activation function to produce a probability distribution over the possible classes.

4.3 DETAILING HYPER PARAMETERS & OPTIMIZING THE ALGORITHM

Hyperparameters in a convolutional neural network (CNN) are values that are set before training the model and can have a significant impact on the performance of the model. Here are some common hyperparameters in a CNN:

Number of filters: This is the number of feature maps used in each convolutional layer. More filters can capture more complex features but also require more computational resources.

Filter/kernel size: This is the size of the filter/kernel used in each convolutional layer. Larger filters can capture larger features but can also lead to more computation.

Stride: This is the number of pixels by which the filter/kernel is moved across the input image at each step. Larger strides reduce the output size and computational cost but may lead to information loss.

Padding: This is the number of pixels added to the input image to preserve its size during convolution. Padding can help to avoid information loss and ensure that the output size is the same as the input size.

Pooling: This is the operation of down-sampling the output of a convolutional layer. Common pooling methods are max pooling and average pooling. Pooling can help to reduce the spatial dimension of the feature maps and can help to avoid overfitting.

Learning rate: This is the step size used to update the weights of the neural network during backpropagation. It controls how much the weights are adjusted during training.

Batch size: This is the number of training examples used in each iteration of training. A larger batch size can reduce the variance of the gradient estimate but can also require more memory.

Dropout rate: This is the probability of dropping out a neuron during training. Dropout can help to prevent overfitting by randomly dropping out neurons and forcing the network to learn more robust features.

These are just a few examples of the hyperparameters that can be tuned in a CNN. The choice of hyperparameters will depend on the specific problem and dataset.

Optimizing algorithms for a convolutional neural network (CNN) is an important task for achieving high performance. Here are some commonly used optimization algorithms for CNNs:

Stochastic Gradient Descent (SGD): This is a widely used optimization algorithm for training neural networks. It updates the weights of the neural network in the direction of the negative gradient of the loss function with respect to the weights.

Adam: This is a popular optimization algorithm that combines the advantages of both SGD and momentum. It uses adaptive learning rates for each parameter and adaptive momentum estimates to converge faster and more accurately.

Adagrad: This is another optimization algorithm that adapts the learning rate to each parameter based on its historical gradient information. It is particularly effective for sparse data.

RMSprop: This is an optimization algorithm that uses the moving average of squared gradients to adapt the learning rate. It is effective for handling non-stationary and noisy gradients.

Adadelta: This is an optimization algorithm that uses the moving average of squared gradients and the moving average of squared parameter updates to adapt the learning rate. It has been shown to be particularly effective for large datasets and complex models.

Nadam: This is an optimization algorithm that combines the advantages of both Adam and Nesterov momentum. It uses Nesterov momentum to compute the gradient and Adam to update the parameters.

The choice of optimization algorithm will depend on the specific problem and dataset. Generally, Adam is a popular choice due to its fast convergence and effectiveness on a wide range of problems. However, it is still important to experiment with different optimization algorithms to find the best one for the specific problem at hand.

4.4 DISCUSSING TRAINING PROCESS & EVALUATION METRICS

Training a convolutional neural network (CNN) involves a process of learning the weights of the model using a set of labeled training data. Here are the key steps involved in training a CNN:

Data preparation: The first step is to prepare the training data. This involves splitting the data into training, validation, and test sets. The training set is used to update the weights of the model, the validation set is used to tune the hyperparameters of the model, and the test set is used to evaluate the final performance of the model.

Model architecture: The next step is to choose the architecture of the CNN. This includes the number of convolutional layers, pooling layers, and fully connected layers. The architecture should be chosen based on the complexity of the problem and the size of the dataset.

Initialization: The weights of the CNN are initialized randomly. It is important to use an initialization technique that allows for efficient learning of the model, such as the Glorot or He initialization.

Forward propagation: During training, the input data is fed forward through the layers of the CNN, and a prediction is made for each example in the training set.

Loss calculation: The difference between the predicted output and the true output is calculated using a loss function such as cross-entropy or mean squared error.

Backpropagation: The error is propagated backward through the network, and the gradients of the loss function with respect to the weights of the network are calculated using the chain rule.

Weight update: The weights of the network are updated using an optimization algorithm such as stochastic gradient descent (SGD), Adam, or Adagrad. The update rule adjusts the weights in the direction that minimizes the loss function.

Repeat: Steps 4-7 are repeated until the weights of the model converge or a maximum number of epochs is reached.

Evaluation: Once the weights are trained, the model is evaluated on the test set to obtain the final performance metrics.

It is important to note that training a CNN can be a time-consuming and computationally intensive task, especially for large datasets and complex models. Training on a GPU or using distributed training can significantly speed up the process. Additionally, careful tuning of hyperparameters and regularization techniques such as dropout and weight decay can improve the performance of the model.

The performance of the CNN for this task can be evaluated using several metrics. Here are some commonly used metrics for evaluating the performance of a CNN for handwritten letter recognition:

Accuracy: This metric measures the proportion of correctly classified letters out of the total number of letters in the test set. It is the most commonly used metric for evaluating classification models.

Precision: This metric measures the proportion of true positives (correctly identified letters) out of all the letters identified as positive by the model. Precision is a useful metric when the cost of false positives is high.

Recall: This metric measures the proportion of true positives out of all the actual positive letters in the test set. Recall is a useful metric when the cost of false negatives is high.

F1 Score: This is the harmonic mean of precision and recall, which combines the two metrics into a single value. It is useful for balancing precision and recall when both metrics are important.

Confusion matrix: A confusion matrix is a table that summarizes the predicted and actual classes of the letters in the test set. It can be used to calculate the accuracy, precision, and recall, as well as to identify which classes are most frequently misclassified.

Receiver Operating Characteristic (ROC) curve: The ROC curve plots the true positive rate (recall) against the false positive rate for different threshold values. It can be used to visualize the performance of the model across different classification thresholds.

Area Under the Curve (AUC): This metric is calculated by integrating the ROC curve and measures the overall performance of the model across different threshold values.

The choice of metrics will depend on the specific requirements of the application. For example, if the goal is to minimize false positives, precision would be a more important metric than recall. Similarly, if the cost of false negatives is high, recall would be a more important metric. Overall, a combination of these metrics can provide a comprehensive evaluation of the performance of a CNN for handwritten letter recognition.

CHAPTER 5

5.1 DEVELOPMENT SETUP

- Install Jupyter Notebook on your machine.
- Install the necessary libraries for developing a Convolutional Neural Network (CNN) model for Handwritten digit recognition, such as TensorFlow, Keras, and Matplotlib.
- Import the MNIST dataset which contains a large set of handwritten digit images along with their corresponding labels. This dataset can be easily imported using the Keras library.
- Preprocess the dataset by converting the images to grayscale, normalizing the pixel values to a range of 0 to 1, and splitting the data into training and testing sets.
- Define the CNN architecture, including the number of convolutional layers, pooling layers, and fully connected layers.
- Train the model using the training set, and evaluate the accuracy of the model using the testing set.
- Save the trained model as a file so that it can be used for deployment.

5.2 DEPLOYMENT SETUP

- Export the trained model which is already saved.
- Build a graphical user interface (GUI) using Tkinter in the Jupyter notebook.
- Add code to the GUI to allow users to draw a letter using a mouse or touchpad.
- Use the trained model to recognize the letter drawn by the user.
- Display the recognized letter on the GUI.

CHAPTER 6

6.1 RESULTS & ANALYSIS

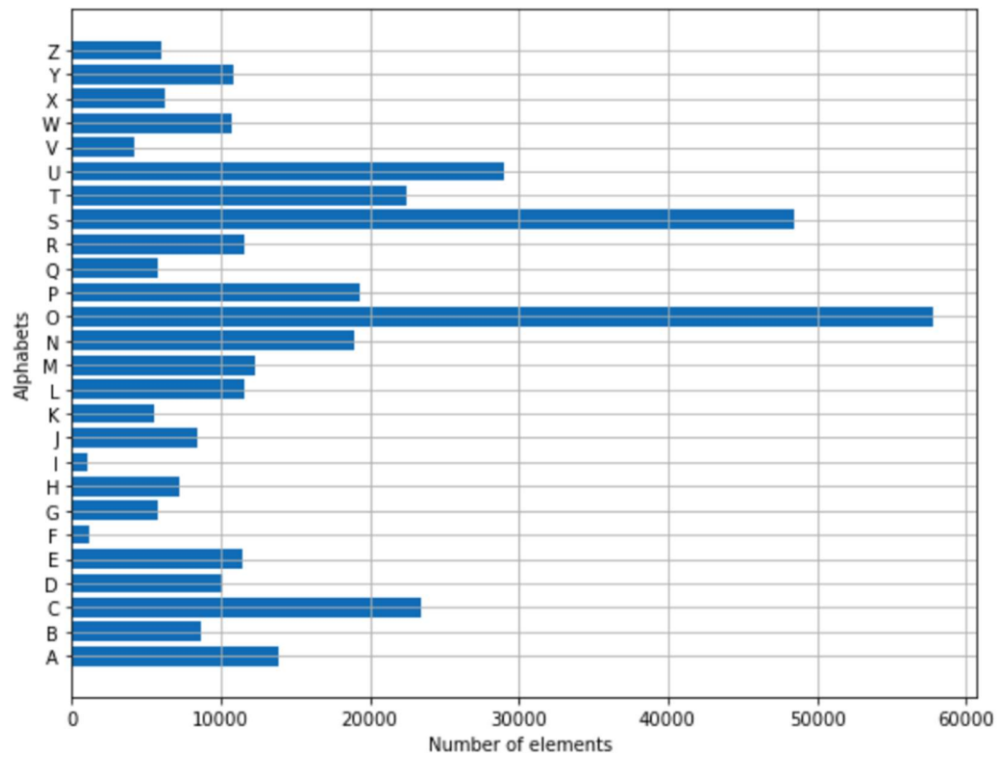


Fig 5.1 – Number of data elements

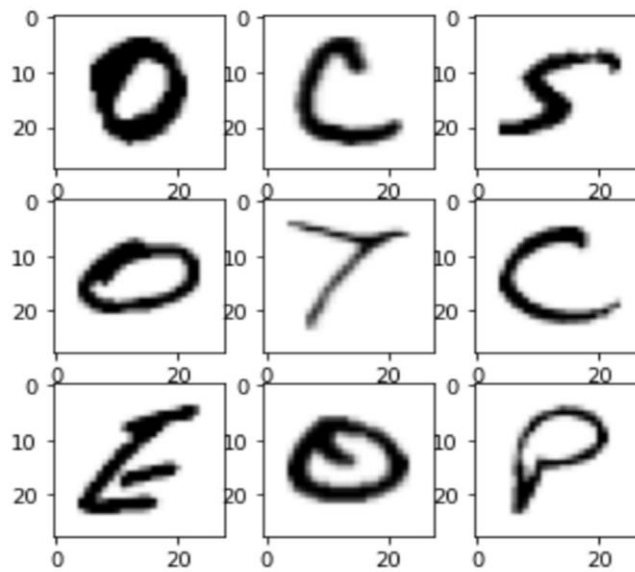


Fig 5.2 – Raw data images

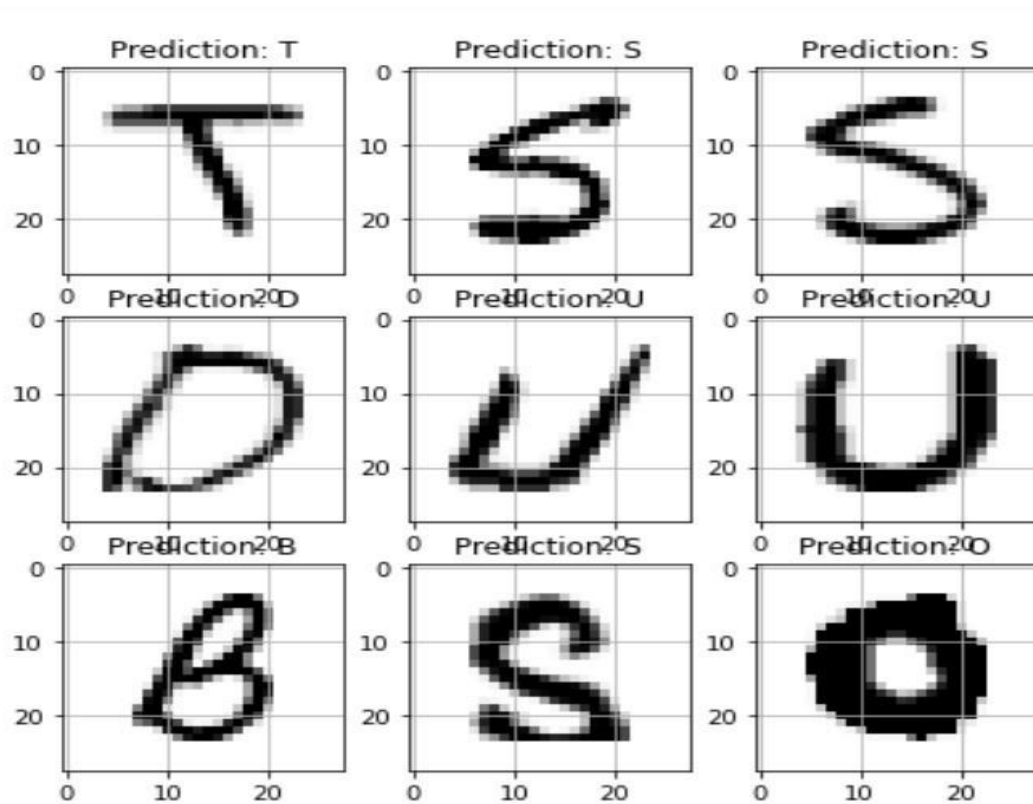


Fig 5.3 – Trained model validation

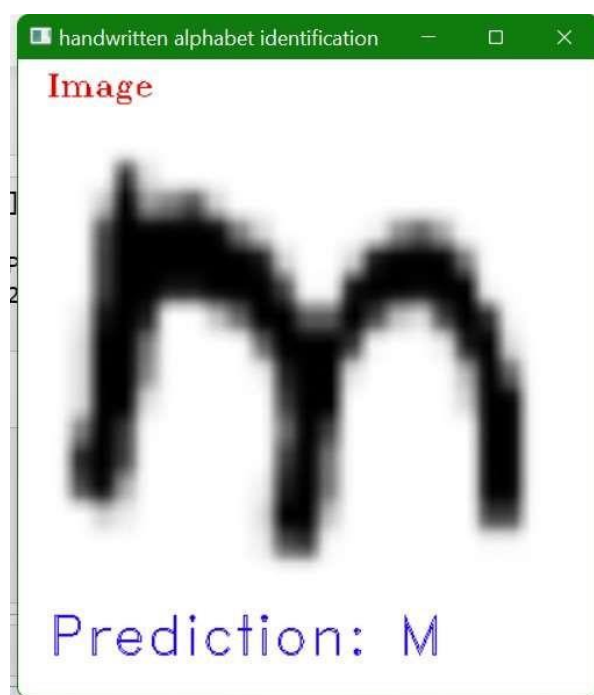


fig 5.4 Final output

```
1 print("The validation accuracy is :", history.history['val_accuracy'])
2 print("The training accuracy is :", history.history['accuracy'])
3 print("The validation loss is :", history.history['val_loss'])
4 print("The training loss is :", history.history['loss'])
```

The validation accuracy is : [0.9723587036132812]

The training accuracy is : [0.9532219171524048]

The validation loss is : [0.09842836856842041]

The training loss is : [0.17315390706062317]

fig 5.5 Performance measure

We implemented the model by training it using the data collected and tested with the inputs from test data. After applying all the optimization techniques ,we achieved an accuracy of 97.25%. We finally took the accuracy and loss as our parameters for performance measure of our implemented model.

CHAPTER 7

7.1 FUTURE WORK & CONCLUSION

In conclusion, handwritten letter recognition has come a long way in recent years thanks to advancements in machine learning and computer vision. While there are still challenges to be addressed, significant progress has been made in achieving high levels of accuracy and performance.

One of the key areas for future work in handwritten letter recognition is improving the recognition of cursive handwriting, which can be more difficult for machines to read due to the fluid nature of cursive writing. Additionally, there is a need to develop more robust models that can handle variations in handwriting styles, sizes, and quality of input images.

Another important area for future research is exploring the use of deep learning techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to improve recognition accuracy and speed. Additionally, incorporating context and language models into the recognition process can further improve the accuracy and reliability of the recognition system.

Furthermore, developing algorithms that can recognize not only individual letters, but entire words or phrases would be a significant advancement in the field of handwritten recognition. This could have a significant impact on fields such as text recognition, translation, and document analysis.

In summary, while significant progress has been made in the field of handwritten letter recognition, there is still a lot of work to be done to improve accuracy and performance, particularly in recognizing cursive handwriting and developing more robust models that can handle variations in handwriting styles. The use of deep learning techniques and incorporating context and language models are promising avenues for future research.

7.2 ISSUES FACED IN RESEARCH

One of the main difficulties in research in convolutional neural network (CNN) handwritten letter recognition is dealing with the high variability of handwriting.

Handwritten letters can vary widely in size, shape, slant, and style, making it challenging to develop a CNN that can accurately recognize all possible variations.

Another challenge is the need for large amounts of training data. CNNs require large amounts of labeled data to learn and generalize well. However, collecting and labeling handwritten letter datasets can be time-consuming and costly, especially for languages with many different characters and variations.

Furthermore, overfitting is a common issue when training CNNs for handwritten letter recognition. Overfitting occurs when a model becomes too complex and starts to fit to the noise in the training data instead of the underlying patterns. This can result in poor performance on new and unseen data.

Another difficulty is choosing appropriate hyperparameters for the CNN, such as the number of layers, the size of the filters, and the learning rate. These hyperparameters can significantly impact the performance of the model, and finding the optimal values can require significant experimentation and fine-tuning.

Finally, the recognition of cursive handwriting is still a challenging task for CNNs. Cursive handwriting can be highly variable, with letters often overlapping or blending into each other. This makes it difficult for the CNN to identify and separate individual letters, leading to lower recognition accuracy.

Overall, the high variability of handwriting, the need for large amounts of training data, overfitting, choosing appropriate hyperparameters, and the recognition of cursive handwriting are some of the main difficulties in research in CNN handwritten letter recognition.

7.3 ISSUES FACED IN IMPLEMENTATION

The main task in implementation is to collecting the data. Even though there are lot of pre-defined datasets available, but they are already developed which has the limitation of data values that results in low accuracy of the model. While implementing UI part error in loading modules occurred many times.

REFERENCES

- [1] Abu Ghosh, M. M., & Maghari, A. Y. (2017). A comparative study on handwriting digit recognition using neural networks. In International Conference on Promising Electronic Technologies (ICPET).
- [2] Caifeng Shan, Shaogang Gong and Peter W. McOwan, "Facial expression recognition based on Localbinarypatterns:Acomprehensivestudy", ELSESEVIER Image and Vision Computing".
- [3] F. Siddique, S. Sakib and M. A. B. Siddique, "Recognition of Handwritten Digit using Convolutional Neural Network in Python with Tensorflow and Comparison of Performance for Various Hidden Layers," 2019 5th International Conference on Advances in Electrical Engineering (ICAEE).
- [4] Handwritten Character Recognition of South Indian Scripts: A Review by Jomy John, Pramod K. V, Kannan Balakrishnan, National Conference on Indian Language Computing.
- [5] Handwritten Character Recognition for Telugu Scripts Using Multi - Layer Perceptrons (MLP). C. Vikram, C. Soba Bindhu, C. Sashikala.
- [6] M. Yadav and R. K. Purwar, "Integrating Wavelet Coefficients and CNN for Recognizing Handwritten Characters," 2018 2nd IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), Delhi, India, 2018, pp. 1160-1164.
- [7] P. Voigtlaender, P. Doetsch and H. Ney, "Handwriting Recognition with Large Multidimensional Long Short-Term Memory Recurrent Neural Networks," 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), Shenzhen, China, 2016, pp. 228-233.
- [7] Rudraswamimath, V.R., & Bhavanishankar, K. (2019). Handwritten digit recognition usingCNN. International Journal of Innovative Science and Research Technology, 4, 182–187.
- [8] Shahid, A. R., & Talukder, S. Evaluating machine learning models for handwriting recognition-based systems under local differential privacy.

- [9] T. Makkar, Y. Kumar, A. K. Dubey, Á. Rocha and A. Goyal, "Analogizing time complexity of KNN and CNN in recognizing handwritten digits," 2017 Fourth International Conference on Image Information Processing (ICIIP), Shimla, India, 2017.
- [10] Telugu handwritten character recognition using adaptive and static zoning methods. Sanugula Durga Prasad, Yashwanth Kandhuri.

APPENDIX

A. SOURCE CODE

DEVELOPING THE MODEL

```
import matplotlib.pyplot as plt

import cv2

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.utils import shuffle

from keras.utils import to_categorical

from keras.models import Sequential

from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout

from keras.optimizers import SGD, Adam

from keras.callbacks import ReduceLROnPlateau, EarlyStopping

#reading data

data = pd.read_csv("A_Z Handwritten Data.csv").astype('float32')

print(data.head(10))

X = data.drop('0',axis = 1)

y = data['0']

#data pre-processing

train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)

train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))

test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))

print("Train data shape: ", train_x.shape)
```

```

print("Test data shape: ", test_x.shape)

word_dict={0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15:'P',16:'Q',17:'R',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X', 24:'Y',25:'Z'}

y_int = np.int0(y)

count = np.zeros(26, dtype='int')

for i in y_int:

    count[i] +=1

alphabets = []

for i in word_dict.values():

    alphabets.append(i)

fig, ax = plt.subplots(1,1, figsize=(9,7))

ax.barh(alphabets, count)

plt.xlabel("Number of elements ")

plt.ylabel("Alphabets")

plt.grid()

plt.show()

#shuffling & thresholding

shuff = shuffle(train_x[:100])

fig, ax = plt.subplots(3,3, figsize = (5,5))

axes = ax.flatten()

for i in range(9):

    _, shu = cv2.threshold(shuff[i], 30, 200, cv2.THRESH_BINARY)

    axes[i].imshow(np.reshape(shuff[i], (28,28)), cmap="Greys")

plt.show()

train_X = train_x.reshape(train_x.shape[0],train_x.shape[1],train_x.shape[2],1)

```

```

print("New shape of train data: ", train_X.shape)

test_X = test_x.reshape(test_x.shape[0], test_x.shape[1], test_x.shape[2],1)

print("New shape of test data: ", test_X.shape)

#One hot Encoding

train_yOHE = to_categorical(train_y, num_classes = 26, dtype='int')

print("New shape of train labels: ", train_yOHE.shape)

test_yOHE = to_categorical(test_y, num_classes = 26, dtype='int')

print("New shape of test labels: ", test_yOHE.shape)

print("\n")

print((pd.DataFrame(test_yOHE)).head(5))

# Create a new sequential model

model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
input_shape=(28,28,1)))

model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding =
'same'))

model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding =
'valid'))

model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64,activation ="relu"))

model.add(Dense(128,activation ="relu"))

```

```

model.compile(optimizer = Adam(learning_rate=0.001),loss=
'categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_X, train_yOHE, epochs=1, validation_data =
(test_X,test_yOHE))

model.add(Dense(26,activation = "softmax"))

#saving the model

model.summary()

model.save(r'model_hand.h5')

#performance measures

score = model.evaluate(test_X,test_yOHE,verbose=0)

print("loss",score[0])

print("Accuracy",score[1])

print("The validation accuracy is :", history.history['val_accuracy'])

print("The training accuracy is :", history.history['accuracy'])

print("The validation loss is :", history.history['val_loss'])

print("The training loss is :", history.history['loss'])

fig, axes = plt.subplots(3,3, figsize=(7,7))

axes = axes.flatten()

for i,ax in enumerate(axes):

    img = np.reshape(test_X[i], (28,28))

    ax.imshow(img, cmap="Greys")

    pred = word_dict[np.argmax(test_yOHE[i])]

    ax.set_title("\nPrediction: "+pred)

    ax.grid()

```

DEPLOYING THE MODEL

```
from tkinter import *

from PIL import Image, ImageDraw

import matplotlib.pyplot as plt

import cv2

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow import keras

word_dict =
{0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15
:'P',16:'Q',17:'R',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X', 24:'Y',25:'Z'}

model = keras.models.load_model('model_hand.h5')

model.summary()

# Initialize Tkinter

root = Tk()

# Set canvas size

canvas_width = 300

canvas_height = 300

# Create canvas

canvas = Canvas(root, width=canvas_width, height=canvas_height, bg='white')

canvas.pack()

# Initialize PIL Image object

img = Image.new("RGB", (canvas_width, canvas_height), "white")

draw = ImageDraw.Draw(img)
```

```

# Initialize starting position

last_x, last_y = None, None

# Define event handler for mouse movement

def motion(event):

    global last_x, last_y

    x, y = event.x, event.y

    if last_x and last_y:

        canvas.create_line(last_x, last_y, x, y, width=10)

        draw.line((last_x, last_y, x, y), fill="black", width=10)

    last_x, last_y = x, y

# Define event handler for mouse release

def release(event):

    global last_x, last_y

    last_x, last_y = None, None

# Bind mouse movement and release events to canvas

canvas.bind('<B1-Motion>', motion)

canvas.bind('<ButtonRelease-1>', release)

# Define function to save image

def save_image():

    resized_img = img.resize((280, 280))

    resized_img.save("drawn_image.jpg")

# Create save button

save_button = Button(root, text="Save Image", command=save_image)

save_button.pack()

```



```

def load_image():

    img = cv2.imread('drawn_image.jpg')

    img_copy = img.copy()

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    img = cv2.resize(img, (400,440))

    img_copy = cv2.GaussianBlur(img_copy, (7,7), 0)

    img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)

    _, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)

    img_final = cv2.resize(img_thresh, (28,28))

    img_final = np.reshape(img_final, (1,28,28,1))

    img_pred = word_dict[np.argmax(model.predict(img_final))]

    cv2.putText(img, "Image ", (20,25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color
= (0,0,230))

    cv2.putText(img, "Prediction: " + img_pred, (20,410),
cv2.FONT_HERSHEY_DUPLEX, 1.3, color = (255,0,30))

cv2.imshow('handwritten alphabet identification', img)

# Start main loop

root.mainloop()

load_image()

cv2.waitKey(0)

cv2.destroyAllWindows()

```

B. SCREENSHOTS

import the libraries

```
In [1]: import matplotlib.pyplot as plt
import cv2
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from keras.optimizers import SGD, Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
```

import the dataset

```
In [2]: data = pd.read_csv("A_Z Handwritten Data.csv").astype('float32')
```

```
In [3]: print(data.head(10))
```

```
      0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  ...  0.639  0.640  0.641  \
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
5  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
6  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
7  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
8  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
```

Fig B(1)

```
In [4]: X = data.drop('0',axis = 1)
y = data['0']
```

Train & Test data

```
In [5]: train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)
```

```
train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))
```

```
print("Train data shape: ", train_x.shape)
print("Test data shape: ", test_x.shape)
```

```
Train data shape: (297960, 28, 28)
Test data shape: (74490, 28, 28)
```

```
In [6]: word_dict = {'0':'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15:'P',16:'Q',
17:'R',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X', 24:'Y',25:'Z'}
```

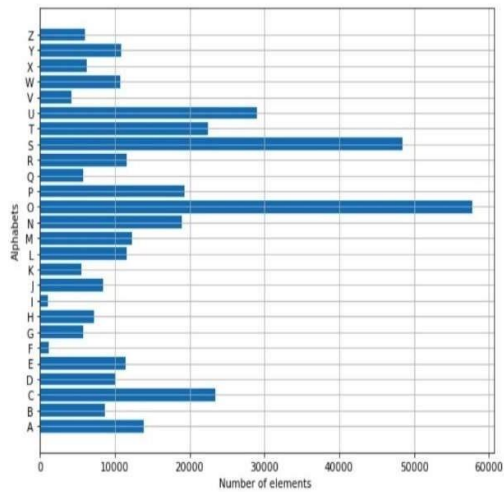
Plotting the data

```
In [7]: y_int = np.int0(y)
count = np.zeros(26, dtype='int')
for i in y_int:
    count[i] +=1

alphabets = []
for i in word_dict.values():
    alphabets.append(i)

fig, ax = plt.subplots(1,1, figsize=(9,7))
```

Fig B(2)

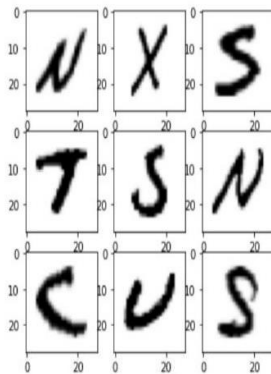


Thesholding image

```
In [8]: shuff = shuffle(train_x[:100])

fig, ax = plt.subplots(3,3, figsize = (5,5))
axes = ax.flatten()
```

Fig B(3)



```
In [9]: train_X = train_x.reshape(train_x.shape[0],train_x.shape[1],train_x.shape[2],1)
print("New shape of train data: ", train_X.shape)

test_X = test_x.reshape(test_x.shape[0], test_x.shape[1], test_x.shape[2],1)
print("New shape of test data: ", test_X.shape)
```

New shape of train data: (297960, 28, 28, 1)
New shape of test data: (74490, 28, 28, 1)

```
In [10]: #One hot Encoding
train_yOHE = to_categorical(train_y, num_classes = 26, dtype='int')
print("New shape of train labels: ", train_yOHE.shape)

test_yOHE = to_categorical(test_y, num_classes = 26, dtype='int')
```

Fig B(4)

New shape of train labels: (297960, 26)
New shape of test labels: (74490, 26)

```

0  0  1  2  3  4  5  6  7  8  9  ... 16 17 18 19 20 21 22 \
0  0  0  0  0  0  0  0  1  0  0  ... 0  0  0  0  0  0  0
1  0  0  0  0  0  0  0  0  0  0  0  ... 0  0  0  0  0  0  0
2  0  0  0  0  0  0  0  0  0  0  0  ... 0  0  0  0  0  0  0
3  0  0  0  0  0  0  0  0  0  0  0  ... 0  0  1  0  0  0  0
4  0  0  0  0  1  0  0  0  0  0  0  ... 0  0  0  0  0  0  0

23 24 25
0  0  0  0
1  0  0  0
2  0  0  0
3  0  0  0
4  0  0  0

```

[5 rows x 26 columns]

Training the model

```

In [11]: # Create a new sequential model
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

```

Fig B(5)

```

In [12]: model.compile(optimizer = Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_X, train_yOHE, epochs=1, validation_data = (test_X, test_yOHE))

9312/9312 [=====] - 289s 31ms/step - loss: 0.1665 - accuracy: 0.9546 - val_loss: 0.0780 - val_accuracy: 0.9781

```

```

In [13]: model.summary()
model.save(r'model_hand.h5')

```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 64)	32832
dense_1 (Dense)	(None, 128)	8320
dense_2 (Dense)	(None, 26)	3354

Fig B(6)

```

=====
Total params: 137,178
Trainable params: 137,178
Non-trainable params: 0
=====

In [14]: score = model.evaluate(test_X, test_yOHE, verbose=0)

print("loss", score[0])
print("Accuracy", score[1])

loss 0.077977536172867
Accuracy 0.9781312942504883

In [15]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

In [16]: print("The validation accuracy is :", history.history['val_accuracy'])
print("The training accuracy is :", history.history['accuracy'])
print("The validation loss is :", history.history['val_loss'])
print("The training loss is :", history.history['loss'])

#precision = precision_score(test_yOHE, history, average='weighted')

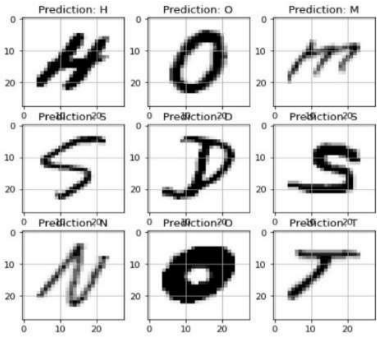
The validation accuracy is : [0.9781312942504883]
The training accuracy is : [0.9546180963516235]
The validation loss is : [0.077977536172867]
The training loss is : [0.16651074588298798]

In [17]: fig, axes = plt.subplots(3,3, figsize=(7,7))
axes = axes.flatten()

for i, ax in enumerate(axes):
    img = np.reshape(test_X[i], (28,28))

```

Fig B(7)



```

In [30]: img = cv2.imread('m_letter.jpg')
img_copy = img.copy()

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (400,440))

```

Fig B(8)

```

In [30]: img = cv2.imread('m_letter.jpg')
img_copy = img.copy()

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (400,440))

In [31]: img_copy = cv2.GaussianBlur(img_copy, (7,7), 0)
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
_, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)

img_final = cv2.resize(img_thresh, (28,28))
img_final = np.reshape(img_final, (1,28,28,1))

In [32]: img_pred = word_dict[np.argmax(model.predict(img_final))]

cv2.putText(img, "Image ", (20,25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color = (0,0,230))
cv2.putText(img, "Prediction: " + img_pred, (20,410), cv2.FONT_HERSHEY_DUPLEX, 1.3, color = (255,0,30))
cv2.imshow('handwritten alphabet identification', img)

1/1 [=====] ~ 0s 29ms/step


In [ ]: while (1):
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
    cv2.destroyAllWindows()

In [ ]:

```

Fig B(9)

C. RESEARCH PAPER

www.ijcrt.org		© 2023 IJCRT Volume 11, Issue 4 April 2023 ISSN: 2320-2882	
IJCRT.ORG		ISSN : 2320-2882	
		INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT) An International Open Access, Peer-reviewed, Refereed Journal	
<hr/>			
<h1>IDENTIFYING HANDWRITTEN LETTERS WITH DEEP LEARNING</h1>			
<p>Karri Uday sai ganesh Computer science and Engineering, Sathyabama Institute of Science and Technology, Chennai. [UG]</p> <p>K.V.S. Aditya Reddy Computer science and Engineering, Sathyabama Institute of Science and Technology, Chennai. [UG]</p> <p>K. Anita Davamani, Asst.Professor Department of Computer science and Engineering, Sathyabama Institute of Science and Technology, Chennai.</p>			
<p>Abstract - Two crucial technological areas that are uniting the world are machine translation and natural language processing. The barrier to communication with others who speak a different language is removed via machine translation. Finding the word's native language and then translating it into the target language are the foundations of machine translation. If each letter in the word can be recognized, determining the language of the word may be simpler. We now arrive at "Character Recognition."</p> <p>This thesis focuses on evaluating the available techniques for handwritten character identification using deep learning and those techniques to recognize handwritten characters. The intriguing thing about machine learning is that the computers that are programmed to interpret, say, Chinese, can also be programmed to translate, say, Hindi or Telugu. This is a key characteristic that the thesis seeks to investigate because it is so potent. Beyond just detecting the characters, modern technology can also translate such characters between different languages. Depending on the character, the machine learning model created for this thesis can accurately identify the character or alphabets with a 70-90% rate of success.</p>		<p>All data is saved and documented in datasets for further sharing and project usage. A large database of handwritten numbers is needed to train different methods for analyzing images. The MNIST database, which is frequently used. The database is commonly used for developing and evaluating machine learning algorithms. It was created by recombining samples from the initial runs. 2017 saw the publication of EMNIST, an expanded dataset that is identical to MNIST and contains 240,000 training photos and 40,000 testing images of handwritten numbers and characters.</p> <p>Keywords: - Machine learning, Deep learning, Sci-kit learn, Denoising, K-nearest neighbor, Pooling, CNN, Convolutional Layers.</p> <h3>I. INTRODUCTION</h3> <p>Researchers have been investigating various methods for testing handwritten character recognition over the past few decades. Learning becomes harder as our assignments become more challenging. We employ deep neural networks, which is suitable. Make a neural network design with higher performance metrics and train it. Deep learning and neural networks enable us to train the model we independently constructed to learn for itself from our experiences.</p>	
<hr/>			
IJCRT2304227		International Journal of Creative Research Thoughts (IJCRT) b920	

Convolutional Neural Network is one of the methods of Deep Learning Algorithms. Characters present in the image are successfully recognized by the CNN classifier. Convolutional layers for feature extraction and fully linked layers followed by a soft-max layer for classification make up the architecture of conventional CNN classifiers. One of the effective feature extractors is CNN.

Data exploration is a methodology that integrates methods from machine learning, statistics, and computer systems to find patterns in enormous volumes of data. The neurons in CNN's architecture are arranged in layers. This design consists of an input layer, many hidden levels, and an output unit. Typically, networks with a lot of hidden layers are referred to as deep neural networks. The neurons in the hidden layers of CNN are connected to only a fraction of the input space produced by the preceding layer, as opposed to entirely important factor like Multi Laminated Perceptron (MLP) networks. The inversion process produces an extracted feature when the convolution kernel passes across the input matrix for the layer, adding to the input of the following layer. Then, further layers like normalizing, pooling, and entirely linked levels are introduced.

In conclusion, handwritten digit recognition is a significant, quickly developing topic with a wide range of real-world uses. We may anticipate significant advancements in precision and performance as deep learning and computer vision technologies keep evolving, making them even more beneficial in a variety of fields.

II. RELATED WORK

The first attempt to recognize the handwritten character was made in Grimsdale in 1959 while conducting research on word recognition. This mid-60s study demonstrated the application of Eden's 1968 examination-by-combination approach. He gave an example to show how the number of schematic highlights is the limit for every single handwritten character.

Different pre-handling systems linked to character recognition were proposed by K. Gaurav and Bhatia P. K. The process attempted numerous picture formats, ranging from a straightforward picture-based report to a colored and altered forces incorporating foundation.

R. Bajaj, S. Chaudhari, L. Dey, et al. used distinctive features including clear portion, thickness, and minute highlights to group the Devanagari numbers. The paper also suggests multi classifier unshakable quality for handwritten Devanagari numerals to further improve recognition capacity.

In her essay on Four characteristics, Sandya Arora provided illustrations using the terms shadow, histogram of chain code crossing point, and horizontal line fitting. One of these highlights was the character image's shadow, while the other three were processed by segmenting the image into the different sections. The dataset of 4900 instances was used in one practical execution, which showed a 90.8% accuracy rate for Devanagari handwritten characters.

In this study, we will classify handwriting using three (3) different methods: SVM, KNN, and Neural Network. Their handwriting recognition system uses Local Binary Pattern (LBP) as a feature extraction method and KNN in "Handwriting Digit Recognition using Local Binary Pattern Variance and K-Nearest Neighbors Classification."

III. EXISTING SYSTEM

Even if the results of existing algorithms for handwritten digit recognition are excellent, there are still several restrictions and issues that need to be resolved:

Several algorithms are sensitive to the input image's quality, especially in terms of contrast, illumination, and noise. The algorithm could have trouble correctly identifying the digit if the input picture is unclear. Some algorithms can only detect digits written in a particular handwriting or style. Due to this, they are not as generalizable to new datasets or handwriting types as what they were trained on. Modern handwritten digit recognition algorithms are frequently complicated and computationally costly, making it challenging to apply them on limited-resource devices like mobile phones or embedded systems. Results from algorithms that were trained on skewed datasets might be skewed. For instance, if some digits are better represented in the training dataset than others, the algorithm may perform less well when trying to identify those digits. As many deep learning algorithms are "black boxes," it might be challenging to comprehend how they make their predictions. This can be an issue in situations where interpretability and openness are crucial, such as in the legal or medical fields.

IV. PROPOSED SYSTEM

Offline and online handwriting identification systems are two distinct types of handwriting recognition. Online techniques use a digital pen or stylus and may see the pen position and stroke information while text is being typed. It is typically possible to classify them quite accurately since they frequently carry an extensive amount of information about the language's flow, which makes it easier to distinguish between the various characters in the text. Offline techniques rely on reading text that has already been written down, therefore they are not aware of the writing strokes or directions used, and they may also have some background noise from the paper used to write the text. Depending on their functionalities, separated the project into several components.

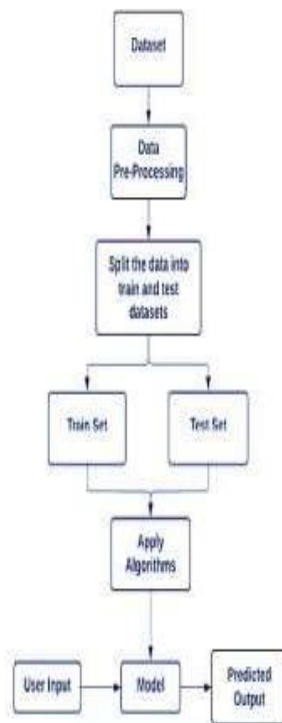


Fig. 1 Architecture Diagram

A sequential model is created. Five hidden Conv2D layers are present in this sequential model. Every Hidden layer is activated using the ReLU function. ReLU speeds up training and introduces nonlinearity. ReLU aids in accelerating computations. Pooling is used to follow every hidden layer. For decreasing the dimensions and calculations, we employ MaxPooling. Due to the smaller number of parameters, Max Pooling also lowers overfitting. Location-invariant feature detection is achieved through Conv + Pooling. Each Previous Input is Connected to the Next Layer by a Dense Layer. Finally, we flatten the layers to make them 1D. The SoftMax Function is used to anticipate outcomes with the greatest degree of certainty. Train the model using an improved optimizer, such as Adam. Adam's optimizer has a higher rate of learning. A Sparse Categorical Crossentropy should be used. Pass the metrics for accuracy. To train the data for 10 epochs, use the .fit() function. Show the Accuracy and Loss on a graph. We have to go through several processes for making the data better and usable.

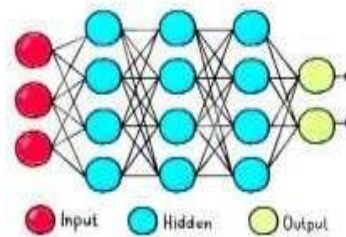


Fig. 2 Basic CNN layers

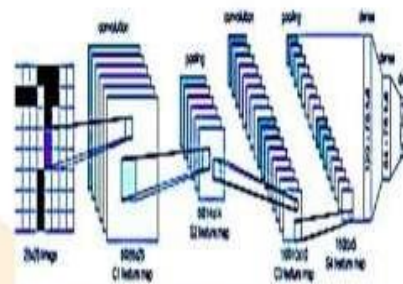


Fig. 3 Layering in CNN

Data cleaning is an important step in the machine learning process that involves identifying and correcting errors or inconsistencies in a dataset. It is an important step that might have a big impact on how accurate and successful machine learning models are.

There are several techniques that can be used for data cleaning, including:

Handling Missing Data: This involves identifying missing values and either imputing them with a value or removing them from the dataset.

Handling Outliers: Outliers are extreme values that can skew statistical analyses. They can be handled by either removing them or transforming them using techniques like winsorization.

Handling Inconsistencies: Inconsistent values can occur when different sources of data are merged. These can be handled by identifying and correcting errors or standardizing values.

Handling Duplicates: Duplicate records can be removed to avoid bias in the analysis.

Handling Irrelevant Data: Data that is not relevant to the analysis can be removed to reduce noise and improve the accuracy of the model.

To ensure that the data cleaning process is effective, it is important to have a clear understanding of the data and the problem being addressed. This involves exploring the data, identifying patterns and trends, and selecting appropriate cleaning techniques.

The process of eliminating extraneous features from the feature set is known as denoising or noise reduction. Although the term "denoising" refers to the elimination of noise, the process involves preserving features.

Making each feature's value have a zero mean and a single variation is the process of standardization. Calculating the mean and standard deviation for each characteristic is part of the broader standardization process. The easiest method for transforming any image into a binary image is by thresholding. A binary image is one in which each pixel may only have one of two potential values. The fundamental concept is to swap every pixel for a black or white, one based on whether the pixel's intensity is either less than or larger than a predetermined constant. The threshold value is another name for this amount.

V. RESULTS&DISCUSSION

In order to verify that the CNN model will function as we expected, we undertook a series of tests to analyze its performance. Various metrics helped in measuring the performance of the proposed model.

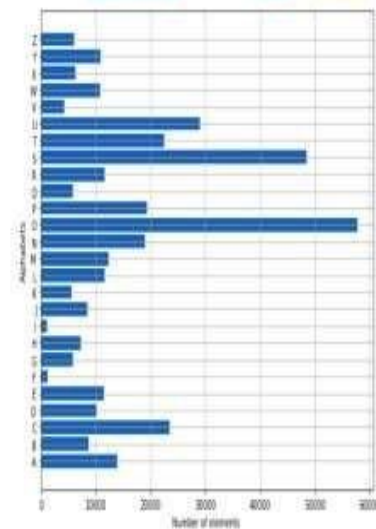


Fig. 4 Plotting the character data.



Fig. 5 After applying threshold on image.

The metrics that we determined for CNN to measure the model's performance, and the results are shown below.

The validation accuracy is : [0.9723587836132812]
 The training accuracy is : [0.9532219171524848]
 The validation loss is : [0.09842836856842041]
 The training loss is : [0.17315398706862317]

Fig. 6 Performance measures.

VI. CONCLUSION

In summary, handwritten text recognition is a growing area of study with applications in a variety of industries, including document digitalization, signature authentication, and handwriting analysis. Despite recent considerable progress in this area, the intricacy and diversity of handwriting make precise and reliable handwritten identification a difficult challenge to solve.

Researchers may investigate different strategies in the future to enhance handwriting recognition systems. The following are some prospective future research areas:

Data augmentation: Adding new training data by rotating, scaling, and skewing already-existing data may help script recognition algorithms become more resilient.

Transfer learning: Pre-training models on massive amounts of printed text or other visual tasks and refining them on smaller amounts of handwritten text may increase recognition accuracy.

End-to-end systems: Improving the accuracy and effectiveness of recognition may require the development of end-to-end systems that can analyze unprocessed

handwriting input without the requirement for manual feature extraction or segmentation.

These are only a few instances of possible directions for further research in the identification of handwriting. It will be fascinating to observe the various methods and strategies that are developed as this topic develops to deal with the difficulties of handwritten text identification.

REFERENCES

- [1] Handwritten Character Recognition of South Indian Scripts: A Review by Jomy John, Pramod K. V, Kannan Balakrishnan, National Conference on Indian Language Computing.
- [2] Multilingual OCR for Indic Scripts by Minesh Mathew, Ajeet Kumar Singh and C. V. Jawahar
- [3] Telugu handwritten character recognition using adaptive and static zoning methods. Sanugula Durga Prasad, Yashwanth Kandhuri.
- [4] Handwritten Character Recognition for Telugu Scripts Using Multi - Layer Perceptrons (MLP). C. Vikram, C. Soba Bindhu, C. Sashikala
- [5] S. Xiao, L. Peng, R. Yan and S. Wang, "Deep Network with Pixel-Level Rectification and Robust Training for Handwriting Recognition," *2019 International Conference on Document Analysis and Recognition (ICDAR)*, Sydney, NSW, Australia, 2019, pp. 9-16.
- [6] M. Yadav and R. K. Purwar, "Integrating Wavelet Coefficients and CNN for Recognizing Handwritten Characters," *2018 2nd IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, Delhi, India, 2018, pp. 1160-1164.
- [7] P. Voigtlaender, P. Doetsch and H. Ney, "Handwriting Recognition with Large Multidimensional Long Short-Term Memory Recurrent Neural Networks," *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, Shenzhen, China, 2016, pp. 228-233.
- [8] Shima Alizadeh and Azarr Fazel, "Convolutional Neural networks for Facial Expression recognition", *Computer Vision and Pattern Recognition Cornell University Library ArXiv:1704.06756v1*, April 2017.
- [9] M. D. Zeiler and Fergus, "Visualizing and understanding convolutional networks", *European Conference on Computer Vision*, vol. 8689, pp. 818-833, 2014.
- [10] Caifeng Shan, Shaogang Gong and Peter W. McOwan, "Facial expression recognition based on Local binary patterns: A comprehensive study", *ELSVIER Image and Vision Computing*.