# ROBUST DETECTION OF MALICIOUS URLS WITH

# SELF-PACED WIDE & DEEP LEARNING

**A PROJECT REPORT**

**Submitted in the partial fulfillment of requirements to**

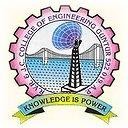
# CS461 – PROJECT III

**For the award of the degree B.Tech. in CSE**

**By**

**Batch No: 11**

**Kanarapu Mohan Srinivas (Y19CS068) Kankanampati Nikhitha (Y19CS067) Nadendla Angel Mathews (Y19CS125)**



# APRIL, 2023

**R.V.R. & J.C. COLLEGE OF ENGINEERING (Autonomous)**

(Affiliated to Acharya Nagarjuna University)

**Chandramoulipuram::Chowdavaram**

**GUNTUR – 522 019**

# R.V.R.& J.C. COLLEGE OF ENGINEERING

**(Autonomous)**

# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**CERTIFICATE**



This is to certify that this project work titled **“Robust Detection of Malicious URLs With**

**Self-Paced Wide & Deep Learning”** is the work done by **Kanarapu Mohan Srinivas (Y19CS068)**, **Kankanampati Nikhitha (Y19CS067)**, and **Nadendla Angel Mathews (Y19CS125)** under my supervision, and submitted in partial fulfillment of the requirements for the award of the degree, B.Tech. in Computer Science & Engineering, during the Academic Year **2022-2023**.

|  |  |  |
| --- | --- | --- |
| **Mr. R. MabuBasha** | **Dr. Ch. Aparna** | **Dr. M. Sreelatha** |
| **Project Work Guide** | **In-charge, Project Work** | **Professor & HOD, CSE** |

# ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without proper suggestions, guidance and environment. Combination of these three factors acts like backbone to our project **“ROBUST DETECTION OF MALICIOUS URLS WITH**

**SELF-PACED WIDE & DEEP LEARNING”**.

We are very glad to express our special thanks to **Mr. R. MabuBasha**, Guide for the project, who has inspired us to select this topic, and also for his valuable advice in preparing this project topic.

We are very thankful to **Dr. Ch. Aparna**, Lecturer, Incharge for the project who extended his encouragement and support to carry out this project successfully.

We express our sincere thanks to **Dr. M. Sreelatha,** Head of the Department of Computer Science and Engineering for her encouragement and support to carry out this project successfully.

We are very much thankful to **Dr. K. Srinivas**, principal of R. V. R& J. C COLLEGE OF ENGINEERING, Guntur for having allowed delivering this project

Finally, we submit our reserves thanks to lab staff in the Department of Computer Science and Engineering and to all our friends for their cooperation during the preparation.

**Kanarapu Mohan Srinivas (Y19CS068) Kankanampati Nikhitha (Y19CS067) Nadendla Angel Mathews (Y19CS125)**

**ABSTRACT**

As the economic costs of cybercrimes continue to rise, it is imperative to protect potential victims from a range of attacks. Despite the varied nature of these crimes, it is the URLs that are the primary connecting point between vulnerable users and potential threats. Existing solutions, such as rule-based and machine learning-based approaches, struggle to provide consistent performance due to the diversity of cybercrimes and the ever-evolving obfuscation strategies employed by malicious URLs. To address this issue, a deep learning-based system is proposed. This system utilizes factorization machine to learn latent interactions among lexical features and position embedding to reduce ambiguity in URL tokens. In addition, temporal convolution network is employed to learn long-distance dependencies among URL tokens. To effectively fuse the heterogeneous features, a self-paced wide & deep learning strategy is introduced. The proposed solution is evaluated on a large-scale URL dataset, demonstrating that position embedding is effective in reducing ambiguity, and the self-paced wide & deep learning strategy outperforms other approaches in terms of F1 score and convergence speed.

# CONTENTS

**Title i**

Certificate ii

[Acknowledgement iii](#_TOC_250020)

[Abstract iv](#_TOC_250019)

[Contents v](#_TOC_250018)

[List of Tables vii](#_TOC_250017)

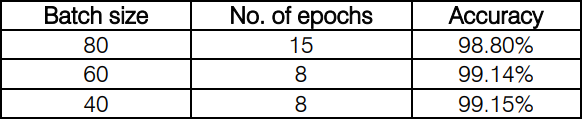
[List of Figures viii](#_TOC_250016)

[List of Abbreviations x](#_TOC_250015)

1. INTRODUCTION 11
   1. [Introduction to Machine Learning 11](#_TOC_250014)
      1. [Machine Learning 11](#_TOC_250013)
      2. [Need for Present Study 13](#_TOC_250012)
      3. [Problem Statement 13](#_TOC_250011)
      4. [Significance of Work 14](#_TOC_250010)
2. LITERATURE SURVEY 15
   1. [Review of the Project 15](#_TOC_250009)
      1. [Limitations of Existing Systems 21](#_TOC_250008)
3. SYSTEM ANALYSIS 22
   1. [Requirement Specification 22](#_TOC_250007)
   2. [UML Diagrams for the project work 25](#_TOC_250006)
4. SYSTEM DESIGN 37
   1. [Architecture of the Proposed System 37](#_TOC_250005)
   2. [Workflow of the Proposed System 40](#_TOC_250004)
   3. [Modules to be Implemented 41](#_TOC_250003)
   4. Modules Description 42
5. IMPLEMENTATION 48
   1. [Algorithms 48](#_TOC_250002)
   2. Data Set 52
   3. [Code 55](#_TOC_250001)
6. TESTING 58
   1. [Training Process 58](#_TOC_250000)
   2. Testing Done 58
7. RESULTS 64
8. CONCLUSION AND FUTURE WORK 71
9. REFERENCES 72

**LIST OF TABLES**

**Table - 1:** Summary of experiment 64



**LIST OF FIGURES**

Fig 1 - Usecase Diagram 28

Fig 2 - Activity Diagram 30

Fig 3 - Sequence Diagram 31

Fig 4 – Class Diagram 34

Fig 5 - The architecture of our proposed CNN 38

Fig 6 – Work Flow of this Project 41

Fig 7 - Sample images taken from MNIST database 42

Fig 8 - Framework of handwritten digit recognition. 43

Fig 9 - A simple Neural Network 43

Fig 10 - Simplified structure of convolutional neural network 44

Fig 11 - convolutional neural network in this paper. 46

Fig 12 – Sample CNN image Processing 47

Fig 13 – Pre-Processing code 49

Fig 14 – Model Creation 49

Fig 15 – Model Training 50

Fig 16 – Model Evaluation 51

Fig 17 – Model Prediction 52

Fig 18 - Test data set of MNIST. 53

Fig 19 - Number 2 in MNIST. 54

Fig 20 - Recognition of handwriting 5 59

Fig 21 - Recognition of handwriting 6 60

Fig 22 - Recognition of handwriting 7 60

Fig 23 - Some correct recognized outputs 61

Fig 24 – Some wrongly recognized outputs 61

Fig 25 - Error rate and accuracy of our model 62

Fig 26 - Correct recognition of bad handwriting 62

Fig 27 – Import Necessary Libraries 64

Fig 28 – Add Your Own Examples to dataset 65

Fig 29 – Pre-Processing training data set 65

Fig 30 – Create your CNN Model 66

Fig 31 – Create callbacks and checkpoints 66

Fig 32 – Train the model 67

Fig 33 – Check Metrics and Save Model 67

Fig 34 – Output Prediction 68

Fig 35 – Four 68

Fig 36 – Zero 69

Fig 37 – Five 69

Fig 38 – Six 70

Fig 39 – Seven 70

# LIST OF ABBREVIATIONS

NN - Neural Network

CNN - Convolutional Neural Network PCA - Principal Component Analysis SVM - Support Vector Machine

MNIST - Modified National Institute of Science & Technology KNN - K-Nearest Neighbor

ANN - Artificial Neural Network DNN - Deep Neural Network

CV - Computer Vision and Pattern Recognition ReLU - Rectified Linear Unit

# CHAPTER – 1 INTRODUCTION

# 1.1 BACKGROUND

Malicious URL detection is a critical component in ensuring the security of computer systems and networks. Cybercriminals use malicious URLs as a means of delivering harmful content, such as malware, spyware, and phishing scams, to unsuspecting users. With the growth of the internet and the increasing dependence of individuals and organizations on online services, the number of malicious URLs has also increased significantly. As a result, there is a pressing need to develop effective tools and techniques to detect and mitigate the risks posed by these threats.

Traditional methods of detecting malicious URLs have included rule-based systems and signature-based methods, which rely on static rules and pre-defined signatures to identify threats. However, these methods are often ineffective in detecting new and evolving threats, as cybercriminals constantly modify their tactics to evade detection. Machine learning-based approaches have emerged as a promising solution for detecting malicious URLs. These approaches leverage the power of deep learning algorithms to identify patterns and anomalies in data, enabling them to detect previously unknown threats with high accuracy.

.**1.2 PROBLEM STATEMENT**

The aim of our research is to develop a deep learning-based model that can effectively detect and classify malicious URLs with high accuracy. The model should be able to learn from a large dataset of labeled URLs, and take numerical representations of URLs as input to output a probability score indicating the likelihood of a given URL being malicious. The challenge lies in the fact that malicious URLs can be highly variable, with new and evolving threats constantly emerging, making it difficult to develop a model that can generalize well to new and unseen data. Therefore, the aim is to develop a robust and effective model architecture that can capture the long-term dependencies in URL sequences, effectively identify patterns and anomalies in the data, and adapt to evolving threats over time. The performance of the model should be evaluated using standard metrics such as accuracy, precision, recall, and F1 score, to ensure that it can be effectively deployed in real-world applications.

## SIGNIFICANCE OF WORK

The work on malicious URL detection using TCN is significant because it addresses a critical need in the field of cybersecurity. With the increasing sophistication and frequency of cyberattacks, there is a pressing need for effective tools and techniques to detect and mitigate the risks posed by malicious URLs. Traditional rule-based and signature-based approaches are often ineffective in detecting new and evolving threats, while machine learning-based approaches have shown promise in detecting previously unknown threats with high accuracy.

The use of TCN in malicious URL detection offers several advantages over other deep learning architectures, as it can capture long-term dependencies in URL sequences and adapt to evolving threats over time. The significance of the work lies in its potential to improve the accuracy and effectiveness of malicious URL detection, which is crucial for protecting individuals and organizations from cyber threats. The proposed solution has been evaluated on a large-scale URL dataset, demonstrating superior performance in terms of F1 score and convergence speed. The work therefore has practical implications for the development of effective cybersecurity tools and techniques, and can contribute to the ongoing efforts to ensure the security of computer systems and networks.

## 1.4 OBJECTIVES

* + To develop a deep learning-based model for the detection of malicious URLs that can effectively capture long-term dependencies in URL sequences and adapt to evolving threats over time.
  + To incorporate techniques such as FM and position embedding to reduce ambiguity in URL tokens and effectively fuse heterogeneous features.
  + To demonstrate the effectiveness of the proposed solution in detecting previously unknown threats with high accuracy, and its potential for practical deployment in real-world applications.
  + To contribute to the ongoing efforts to improve the security of computer systems and networks by developing effective tools and techniques for the detection and mitigation of cyber threats.

**CHAPTER – 2**

**LITERATURE REVIEW**

**2.1 REVIEW OF THE PROJECT**

[1] A. Abbasi, F. M. Zahedi, and S. Kaza, “Detecting fake medical websites using recursive trust labeling,” ACM Trans. Inf. Syst., vol. 30, no. 4, pp. 1–36, Nov. 2012.

[2] T. Li, G. Kou, and Y. Peng, “Improving malicious URLS detection via feature engineering: Linear and nonlinear space transformation methods,” Inf. Syst., vol. 91, 2020, Art. no. 101494.

[3] A. Abbasi, Z. Zhang, D. Zimbra, H. Chen, and J. F. Nunamaker, “Detecting fake websites: The contribution of statistical learning theory,” MIS Q., vol. 34, no. 3, pp. 435–461, Sep. 2010.

[4] C.-T. Huang, M. N. Sakib, C. A. Kamhoua, K. A. Kwiat, and L. Njilla, “A Bayesian game theoretic approach for inspecting web-based malvertising,” IEEE Trans. Dependable Secure Comput., vol. 17, no. 6, pp. 1257–1268, Nov./Dec. 2020.

[5] G. Sun and Q. Qian, “Deep learning and visualization for identifying malware families,” IEEE Trans. Dependable Secure Comput., vol. 18, no. 1, pp. 283–295, Jan./Feb. 2021.

[6] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna, “Towards detecting compromised accounts on social networks,” IEEE Trans. Dependable Secure Comput., vol. 14, no. 4, pp. 447–460, Jul./Aug. 2017.

[7] M. S. I. Mamun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, and A. A. Ghorbani, “Detecting malicious URLs using lexical analysis,” in Network and System Security, J. Chen, V. Piuri, C. Su, and M. Yung, Eds. Berlin, Germany: Springer, 2016, pp. 467–482.

[8] M. Akiyama, T. Yagi, T. Yada, T. Mori, and Y. Kadobayashi, “Analyzing the ecosystem of malicious url redirection through longitudinal observation from honeypots,” Comput. Secur., vol. 69, pp. 155–173, 2017.

[9] K. Althobaiti, N. Meng, and K. Vaniea, “I don’t need an expert! making URL phishing features human comprehensible,” in Proc. CHI Conf. Hum. Factors Comput. Syst., 2021, pp. 1–17.

[10] K. Althobaiti, G. Rummani, and K. Vaniea, “A review of human and computer-facing url phishing features,” in Proc. IEEE Eur. Symp. Secur. Privacy Workshops, 2019, pp. 182–191.

[11] T. T. T. Pham, V. N. Hoang, and T. N. Ha, “Exploring efficiency of character-level convolution neuron network and long short-term memory on malicious URL detection,” in Proc. 7th Int. Conf. Netw. Commun. Comput., 2018, pp. 82–86.

[12] Y. Zhauniarovich, I. Khalil, T. Yu, and M. Dacier, “A survey on malicious domains detection through DNS data analysis,” ACM Comput. Surv., vol. 51, no. 4, pp. 1–36, Jul. 2018.

[13] S. Rahman, T.-K. Huang, H. V. Madhyastha, and M. Faloutsos, “Detecting malicious facebook applications,” IEEE/ACM Trans. Netw., vol. 24, no. 2, pp. 773–787, Apr. 2016.

[14] V. Thambawita et al., “An extensive study on cross-dataset bias and evaluation metrics interpretation for machine learning applied to gastrointestinal tract abnormality classification,” ACM Trans. Comput. Healthcare, vol. 1, no. 3, pp. 1–29, Jun. 2020.

[15] A. AlEroud and G. Karabatis, “Bypassing detection of URL-based phishing attacks using generative adversarial deep neural networks,” in Proc. 6th Int. Workshop Secur. Privacy Anal., 2020, pp. 53–60.

[16] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Learning to detect malicious URLs,” ACM Trans. Intell. Syst. Technol., vol. 2, no. 3, pp. 1–24, May 2011.

[17] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in NLP,” in Proc. 57th Annu. Meeting Assoc. Comput. Linguistics, 2019, pp. 3645–3650.

[18] L. Jiang, D. Meng, S.-I. Yu, Z. Lan, S. Shan, and A. Hauptmann, “Self-paced learning with diversity,” in Proc. 27th Int. Conf. Neural Inf. Process. Syst., 2014, pp. 2078–2086.

[19] Y. Chen, F. M. Zahedi, A. Abbasi, and D. G. Dobolyi “Trust calibration of automated security IT artifacts: A multi-domain study of phishing-website detection tools,” Inf. Manage., vol. 58, no. 1, 2021, Art. no. 103394.

[20] H. Choi, B. Zhu, and H. Lee, “Detecting malicious web links and identifying their attack types,” in Proc. 2nd USENIX Conf. Web Appl. Develop., 2011, pp. 125–136, Art. no. 11.

[21] E. Baykan, M. Henzinger, L. Marian, and I. Weber, “Purely URL based topic classification,” in Proc. 18th Int. Conf. World Wide Web, 2009, pp. 1109–1110.

[22] R. Verma and A. Das, “What’s in a URL: Fast feature extraction and malicious URL detection,” in Proc. 3rd ACM Int. Workshop Secur. Privacy Anal., 2017, pp. 55–63.

[23] C. Dai, X. Liu, and J. Lai, “Human action recognition using two stream attention based LSTM networks,” Appl. Soft Comput., vol. 86, 2020, Art. no. 105820.

[24] Y. Liang, J. Kang, Z. Yu, B. Guo, X. Zheng, and S. He, “Leverage temporal convolutional network for the representation learning of urls,” in Proc. IEEE Int. Conf. Intell. Secur. Inform., 2019, pp. 74–79.

[25] X. Yan, Y. Xu, B. Cui, S. Zhang, T. Guo, and C. Li, “Learning URL embedding for malicious website detection,” IEEE Trans. Ind. Informat., vol. 16, no. 10, pp. 6673–6681, Oct. 2020.

[26] M. Sameen, K. Han, and S. O. Hwang, “Phishhaven—An efficient real-time ai phishing URLs detection system,” IEEE Access, vol. 8, pp. 83 425–83 443, 2020.

[27] H. Le, Q. Pham, D. Sahoo, and S. C. H. Hoi, “URLNet: Learning a url representation with deep learning for malicious url detection,” 2018, arXiv:1802.03162.

[28] P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta, “PhishNet: Predictive blacklisting to detect phishing attacks,” in Proc. IEEE INFOCOM, 2010, pp. 1–5.

[29] W. Yang, W. Zuo, and B. Cui, “Detecting malicious URLs via a keyword-based convolutional gated-recurrent-unit neural network,” IEEE Access, vol. 7, pp. 29 891–29 900, 2019.

[30] S. Marchal, J. Franc¸ois, R. State, and T. Engel, “PhishStorm: Detecting phishing with streaming analytics,” IEEE Trans. Netw. Service Manag., vol. 11, no. 4, pp. 458–471, Dec. 2014.

[31] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin, “Evasion techniques: Sneaking through your intrusion detection/prevention systems,” IEEE Commun. Surveys Tuts., vol. 14, no. 4, pp. 1011–1020, Fourth Quarter 2012.

[32] D. Li and Q. Li, “Adversarial deep ensemble: Evasion attacks and defenses for malware detection,” IEEE Trans. Inf. Forensics Secur., vol. 15, pp. 3886–3900, 2020.

[33] M.-Y. Kan and H. O. N. Thi, “Fast webpage classification using URL features,” in Proc. 14th ACM Int. Conf. Inf. Knowl. Manage., 2005, pp. 325–326.

[34] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Beyond blacklists: Learning to detect malicious web sites from suspicious URLs,” in Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2009, pp. 1245–1254.

[35] P. Yang, G. Zhao, and P. Zeng, “Phishing website detection based on multidimensional features driven by deep learning,” IEEE Access, vol. 7, pp. 15 196–15 209, 2019.

[36] S. C. H. Hoi, D. Sahoo, J. Lu, and P. Zhao, “Online learning: A comprehensive survey,” Neurocomputing, vol. 459, pp. 249–289, 2021.

[37] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Identifying suspicious URLs: An application of large-scale online learning,” in Proc. 26th Annu. Int. Conf. Mach. Learn., 2009, pp. 681–688.

[38] R. Rajalakshmi, J. Raymann, A. Prabu, and C. Aravindan, “Deep URL: Design of adult URL classifier using deep neural network,” in Proc. Int. Conf. Adv. Inf. Sci. Syst., 2019, pp. 1–5.

[39] M. Trevisan and I. Drago, “Robust URL classification with generative adversarial networks,” SIGMETRICS Perform. Eval. Rev., vol. 46, no. 3, pp. 143–146, Jan. 2019.

[40] M. P. Kumar, B. Packer, and D. Koller, “Self-paced learning for latent variable models,” in Proc. 23rd Int. Conf. Neural Inf. Process. Syst., 2010, pp. 1189–1197.

[41] H.-T. Cheng et al., “Wide & deep learning for recommender systems,” in Proc. 1st Workshop Deep Learn. Recommender Syst., 2016, pp. 7–10.

[42] S. Rendle, “Factorization machines,” in Proc. IEEE Int. Conf. Data Mining, 2010, pp. 995–1000.

[43] S. Deng, N. Zhang, W. Zhang, J. Chen, J. Z. Pan, and H. Chen, “Knowledge-driven stock trend prediction and explanation via temporal convolutional network,” in Proc. Companion World Wide Web Conf., 2019, pp. 678–685.

[44] L. Jiang, D. Meng, S.-I. Yu, Z. Lan, S. Shan, and A. G. Hauptmann, “Self-paced learning with diversity,” in Proc. 27th Int. Conf. Neural Inf. Process. Syst., 2014, pp. 2078–2086.

[45] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in Proc. 26th Int. Conf. Neural Inf. Process. Syst., 2013, pp. 3111–3119.

[46] S. Prillo, “An elementary view on factorization machines,” in Proc. 11th ACM Conf. Recommender Syst., 2017, pp. 179–183.

[47] Y. Liang, X. Wang, Z. Yu, B. Guo, X. Zheng, and S. Samtani, “Energy-efficient collaborative sensing: Learning the latent correlations of heterogeneous sensors,” ACM Trans. Sensor Netw., vol. 17, no. 3, pp. 1–28, Jun. 2021.

[48] C. Wu, X.-J. Wu, and J. Kittler, “Spatial residual layer and dense connection block enhanced spatial temporal graph convolutional network for skeleton-based action recognition,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop, 2019, pp. 1740–1748.

[49] A. Zaeemzadeh, N. Rahnavard, and M. Shah, “Norm-preservation: Why residual networks can become extremely deep?,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 43, no. 11, pp. 3980–3990, Nov. 2021.

[50] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” 2018, arXiv: Learning. [Online]. Available: https://arxiv.org/abs/1803.01271?spm=a2c4e.11153940.blogcont642474.26.18fb6a59FjzYuR

[51] L. Lin, B. Xu, W. Wu, T. Richardson, and E. A. Bernal, “Medical time series classification with hierarchical attention-based temporal convolutional networks: A case study of myotonic dystrophy diagnosis,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops, 2019, pp. 83–86.

[52] S. A. Seyedi, S. S. Ghodsi, F. Akhlaghian, M. Jalili, and P. Moradi, “Self-paced multi-label learning with diversity,” in Proc. 11th Asian Conf. Mach. Learn. 2019, pp. 790–805.

## 2.2 LIMITATIONS OF EXISTING SYSTEMS

**2.2.1 Blacklist-Based Methods**

For the blacklist-based methods, they maintain one blacklist with a large number of known malicious URLs. In the mainstream web browsers, they provide blacklist extensions to block the requests when a vulnerable user is going to visit a suspicious website. Although the blacklist-based solutions are effective to find the suspicious URLs by matching the blacklist, they are fragile when faced with the explosive growth of URLs for the following reasons [28]. First, updating blacklists is very slow, resulting in that blacklist-based solutions do not work well for the emerging suspicious URLs [29]. Second, it is found that malicious URLs have very short time span to evade the detection [7], [8], [30]. The nature of malicious URLs makes it extremely infeasible to maintain an up-to-date blacklist. Third, blacklist-based solutions are static with no learning capability and have lower robustness when faced with the URLs generated by new evasion techniques [31], [32].

**2.2.2 Machine Learning-Based Methods**

Unlike blacklist-based solutions, a large number of machine learning-based models have been applied to learn the generalized models based on the existing suspicious URLs. Many classification algorithms including random forest, support vector machine, decision tree, kNN, maximum entropy [33] are applied. Ma et al. [34] used statistical methods to find the attributes of malicious URLs, extracted nine features, and compared the proposed classification model with Naive Bayes, SVM, and logistic regression. The dominant factor determining the model performance is feature engineering [35]. Prior studies rely on hand-crafted features, which is time consuming and tricky to build one complete feature space. The static feature engineering does not work well with the explosive growth of URLs. Machine learning-based methods aim to learn a static model from the whole observation samples. However, they overlook the complexity and diversity of samples, and take all the samples equally for model training. To address these problems, online learning has become a promising technique to learn the emerging patterns from continuous streams of data [36]. Ma et al. fused both the lexical and host-based features of URLs and proposed a confidence-weighted online learning classifier to detect malicious URLs [16], [37].

## 

# CHAPTER – 3 SYSTEM ANALYSIS

## Requirement Specification

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications. Requirements analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

* + 1. Less Response Time
    2. High Data Utility
    3. Preserve security of database
    4. Better Performance
    5. Less computational intensity

## 3.1.1 Functional Requirements

A Functional Requirement is a description of the service that the software must offer. A function may be input to the system, it’s behavior and outputs. It can be any functionality which defines what function a system is likely to perform. Functional requirements are also called as Functional Specifications.

1. Data Collection and Preprocessing: The system should be able to collect a large volume of labeled URLs from different sources and preprocess the data by removing noise, performing tokenization, and feature extraction.
2. Model Training: The system should be able to train a deep learning-based model using the preprocessed data, which can capture long-term dependencies in URL sequences, learn latent interactions among lexical features, and effectively fuse heterogeneous features.
3. Model Evaluation: The system should be able to evaluate the performance of the trained model using standard metrics such as accuracy, precision, recall, and F1 score, and compare it with existing state-of-the-art methods.
4. Model Deployment: The system should be able to deploy the trained model in a production environment, where it can receive incoming URLs and classify them as either malicious or benign.
5. Model Maintenance: The system should be able to continually update the model with new data and adapt to evolving threats over time, to ensure that it remains effective in detecting new and emerging threats.
6. User Interface: The system should have a user-friendly interface that allows users to interact with the system, view the classification results, and provide feedback.
7. Security and Privacy: The system should ensure the security and privacy of the collected data and the classification results, and comply with relevant data protection regulations.
8. Performance and Scalability: The system should be able to handle a large volume of URLs in real-time and provide classification results with high accuracy and low latency, and should be able to scale to accommodate increased data volumes and user demand.

## 3.1.2 Non-Functional Requirements

A Non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions.

1. Accuracy: The system should have high accuracy in detecting malicious URLs and minimizing false positives, to reduce the risk of false alarms and minimize the impact of cyber-attacks.
2. Performance: The system should be able to classify URLs in real-time, with low latency and high throughput, to provide timely and accurate detection of malicious URLs.
3. Scalability: The system should be able to scale to handle a large volume of URLs, and accommodate an increasing number of users, to support growing demand and accommodate future growth.
4. Reliability: The system should be highly reliable, with minimal downtime or disruptions, to ensure continuous availability and minimize the risk of system failures.
5. Maintainability: The system should be easy to maintain and update, with modular components and clear documentation, to facilitate ongoing improvements and adaptations.
6. Usability: The system should be easy to use, with a user-friendly interface, clear instructions, and minimal training required, to facilitate adoption and use by a wide range of users.
7. Compatibility: The system should be compatible with a range of platforms and devices, to ensure broad accessibility and easy integration with existing systems and tools.
8. Compliance: The system should comply with relevant regulatory and legal requirements, including data protection regulations, to ensure legal and ethical use of the system and its results.

## 3.1.3 Requirements Model

Requirement analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications. Requirement analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

## User Requirements

1. Execution should be fast
2. More accurate
3. User-friendly

## Software Requirements

1. Operating System : Windows
2. Language : Python3

## Hardware Requirements

* Processor - Pentium IV or higher
* Speed – 2.4GHz
* RAM - 256 MB (min), Hard Disk – 512 MB (Minimum).

## UML Diagrams for the Project Work

UML is an acronym that stands for Unified Modeling Language. Simply put, UML is a modern approach to modeling and documenting software. In fact, it’s one of the most popular business process modeling techniques.

It is based on diagrammatic representations of software components. As the old proverb says: “a picture is worth a thousand words”. By using visual representations, were able to better understand possible flaws or errors in software or business processes.

The elements are like components which can be associated in different ways to make a complete UML picture, which is known as diagram. Thus, it is very important to understand the different diagrams to implement the knowledge in real- life systems.

Any complex system is best understood by making some kind of diagrams or pictures. These diagrams have a better impact on our understanding. If we look around, we will realize that the diagrams are not a new concept but it is used widely in different forms in different industries.

Mainly, UML has been used as a general-purpose modeling language in the field of software engineering. However, it has now found its way into the documentation of several business processes or workflows. For example, activity diagrams, a type of UML diagram, can be used as a replacement for flowcharts. They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficiency.

Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system. Since all the needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases. Together this usecase collection specifies all the ways the system. An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces.

Associations are the most general of all relationships and consequentially the most semantically weak. If two objects are usually considered independently, the relationship is an association. They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficiency. Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system. Since all the needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases.

By default, the association tool on the toolbox is unidirectional and drawn on a diagram with a single arrow at one end of the association. The end with the arrow indicates who or what is receiving the communication. A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Typically, on class diagrams, a dependency relationship indicates that the operations of the client invoke operations of the supplier. The work flow in this case begins from importing the dataset by the developer and then replacing missing values with mean value of corresponding column, model building, validating that model by generating a confusion matrix and finally predicting the test sample class label. Transitions are used to show the passing of the flow of control from activity to activity.

The various UML diagrams are:

* + 1. Usecase diagram
    2. Activity diagram
    3. Sequence diagram
    4. Collaboration diagram
    5. Object diagram
    6. State chart diagram
    7. Class diagram
    8. Component diagram
    9. Deployment diagram

**3.2.1 Usecase Diagram**

A use case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication (participation) associations between the actors and users and generalization among use cases. The use case model defines the outside (actors) and inside (use case) of the system’s behavior. Actors are not part of the system. Actors represent anyone or anything that interacts with (input to or receive output from) the system. Usecase diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use usecase diagrams to specify the behavior of the system as implemented.

Use case is a sequence of transactions performed by a system that yields a measurable result of values for a particular actor. The use cases are all the ways the system may be used. Use case is a list of actions or event steps, typically defining the interactions between a role (known as an actor) and a system, to achieve a goal. In case of the usecase diagram developer and the end user are the actors. Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system. Since all the needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases. Together this usecase collection specifies all the ways the system. This usecase diagram consists of a user and a trainer with five usecases. The usecases are input stroke, get text label, train model, view trained model, enter training data. These are the usecases used by our system.

## Diagram

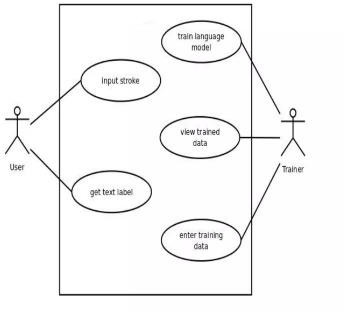


Fig – 1: Usecase diagram

## 3.2.2 Activity Diagram

An activity diagram is a variation of a special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations. The purpose of Activity diagram is to provide a view of flows and what is going on inside a use case or among several classes. Activity diagrams contain activities, transitions between the activities, decision points, and synchronization bars. An activity represents the performance of some behavior in the workflow. In the UML, activities are represented as rectangles with rounded edges, transitions are drawn as directed arrows, decision points are shown as diamonds, and synchronization bars are drawn as thick horizontal or vertical bars as shown in the following. The activity icon appears as a rectangle with rounded ends with a name and a component for actions.

Swim lanes may be used to partition an activity diagram. This typically is done to show what person or organization is responsible for the activities contained in the swim lane. Swim lanes are helpful when modeling a business workflow because they can represent organizational units or roles within a business model. Swim lanes are very similar to an object because they provide a way to tell who is performing a certain role. Swim lanes only appear on activity diagrams. When a swim lane is dragged onto an activity diagram, it becomes a swim lane view. Swim lanes appear as small icons in the browse while swim lane views appear between the thin, vertical lines with a header that can be renamed and relocated. An activity represents the performance of some behavior in the work flow. In the UML, activities are represented as rectangles with rounded edges, transitions are drawn as directed arrows, decision points are shown as diamonds, and synchronization bars are drawn as thick horizontal or vertical bars as shown in the following. The activity icon appears as a rectangle with rounded ends with a name and a component for actions.

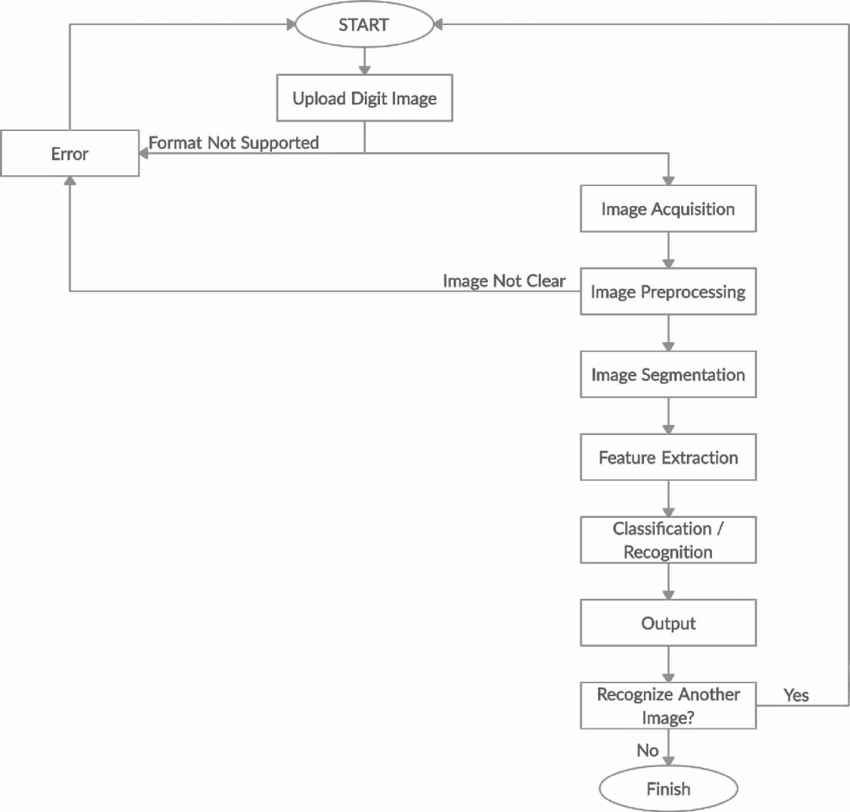


Fig - 2: Activity Diagram

## 3.2.3 Sequence Diagram

A sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called as event diagrams.

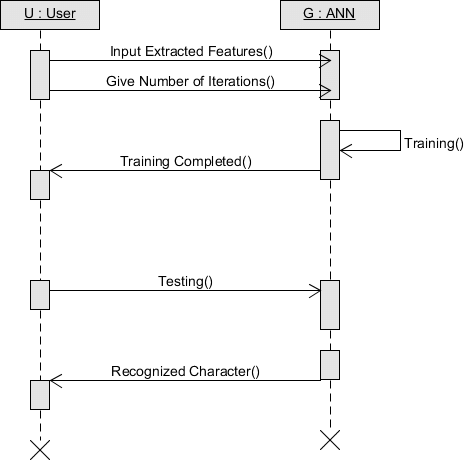


Fig – 3: Sequence Diagram

## 3.2.4 Collaboration Diagram

A collaboration diagram shows that the order of messages that implement an operation or a transaction. Collaboration diagrams show objects, their links, and their messages. They can also contain simple class instances and class utility instances. Each collaboration diagram provides a view of the interactions or structural relationships that occur between objects and object like entities in the current model. Collaboration diagrams and sequence diagrams are called interaction diagrams. A collaboration diagram shows that the order of messages that implement an operation or a transaction. Collaboration diagrams show objects, their links, and their messages. They can also contain simple class instances and class utility instances. Each collaboration diagram provides a view of the interactions or structural relation- ships that occur between objects and object like entities in the current model.

The second interaction diagram is the collaboration diagram. It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. Method calls are similar to that of a sequence diagram. However, difference being the sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization. To choose between these two diagrams, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then collaboration diagram is used. interaction diagrams are used to describe the dynamic nature of a system. Now, we will look into the practical scenarios where these diagrams are used. To understand the practical application, we need to understand the basic nature of sequence and collaboration diagram.

The main purpose of both the diagrams are similar as they are used to capture the dynamic behavior of a system. However, the specific purpose is more important to clarify and understand.

Sequence diagrams are used to capture the order of messages flowing from one object to another. Collaboration diagrams are used to describe the structural

organization of the objects taking part in the interaction. A single diagram is not sufficient to describe the dynamic aspect of an entire system, so a set of diagrams are used to capture it as a whole. Interaction diagrams are used when we want to understand the message flow and the structural organization. Message flow means the sequence of control flow from one object to another. Structural organization means the visual organization of the elements in a system.

## 3.2.5 Class Diagram

Class diagrams contain icons representing classes, interfaces, and their relationships. You can create one or more class diagrams to represent the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model. You can also create one or more class diagrams to represent classes contained by each package in your model; such class diagrams are themselves contained by the package enclosing the classes they represent; the icons representing logical packages and classes in class diagrams.

1. Class diagrams are created to provide a picture or view of some or all of the classes in the model.
2. The main class diagram in the logical view of the model is typically a picture of the packages in the system. Each package also has its own main class diagram, which typically displays the public classes of the package.

A class diagram is a picture for describing generic descriptions of possible systems. Class diagrams and collaboration diagrams are alternate representations of object models.

A Class is a description of a group of objects with common properties (at- tributes) common behavior (operations), common relationships to other objects, and common semantics. Thus, a class is a template to create objects. Each object is an instance of some class and objects cannot be instances of more than one class.

In the UML, classes are represented as compartmentalized rectangles.

1. The top compartment contains the name of the class.
2. The middle compartment contains the structure of the class (attributes).
3. The bottom compartment contains the behavior of the class (operations).

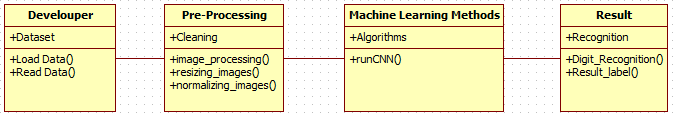


Fig – 4: Class Diagram

## 3.2.6 State Chart Diagram

Use cases and scenarios provide a way to describe system behavior; in the form of interaction between objects in the system. Sometime it is necessary to consider inside behavior of an object.

A state chart diagram shows the states of a single objects, the events or messages that cause a transition from one state to another and the actions that result from a state change. As I activity diagram, state chart diagram also contains special symbols for start state and stop state.

State chart diagram cannot be created for every class in the system, it is only for those lass objects with significant behavior.

## 3.2.7 Component Diagram

Component Diagrams show the dependencies between software components in the system. The nature of these dependencies will depend on the language or languages used for the development and may exist at compile-time or at runtime.

In a large project there will be many files that make up the system. These files will have dependencies on one another. The nature of these dependencies will depend on the language or languages used for the development and may exist at compile-time, at link-time or at run-time. There are also dependencies between source code files and the executable files or byte code files that are derived from them by compilation.

Component diagrams are one of the two types of implementation diagram in UML. Component diagrams show these dependencies between software components in the system. Stereotypes can be used to show dependencies that are specific to particular languages also.

A component diagram shows the allocation of classes and objects to components in the physical design of a system. A component diagram may represent all or part of the component architecture of a system along with dependency relationships. The dependency relationship indicates that one entity in a component diagram uses the services or facilities of another.

## 3.2.8 Deployment Diagram

The second type of implementation diagram provided by UML is the deployment diagram. Deployment diagrams are used to show the configuration of runtime processing elements and the software components and processes that are located on them. Deployment diagrams are made up of nodes and communication associations. Nodes are typically used to show computers and the communication associations show the network and protocols that are used to communicate between nodes. Nodes can be used to show other processing resources such as people or mechanical resources. Nodes are drawn as 3D views of cubes or rectangular prisms, and the following figure shows a simplest deployment diagram where the nodes connected by communication associations.

# CHAPTER – 4 SYSTEM DESIGN

**4.1.1 Architecture of the Proposed System**

The architecture of a malicious URL detection system using TCN is a multi-component system that includes data collection and preprocessing, factorization machine (FM), position embedding, temporal convolutional network (TCN), self-paced wide & deep learning, model evaluation, model deployment, model maintenance, user interface, security and privacy, and performance and scalability. The system is designed to collect a large number of URLs from various sources, preprocess the data, and learn the latent interactions among lexical features of the URL using the FM model. The position embedding component reduces ambiguity of URL tokens, while the TCN component learns the long-distance dependency among URL tokens using TCN. The self-paced wide & deep learning strategy fuses the heterogeneous features learned by the FM, Position Embedding, and TCN components. The trained model is evaluated using standard metrics and deployed in a production environment with a user-friendly interface. The system is designed to be scalable, reliable, and secure, and can handle a large volume of URLs in real-time with high accuracy and low latency.

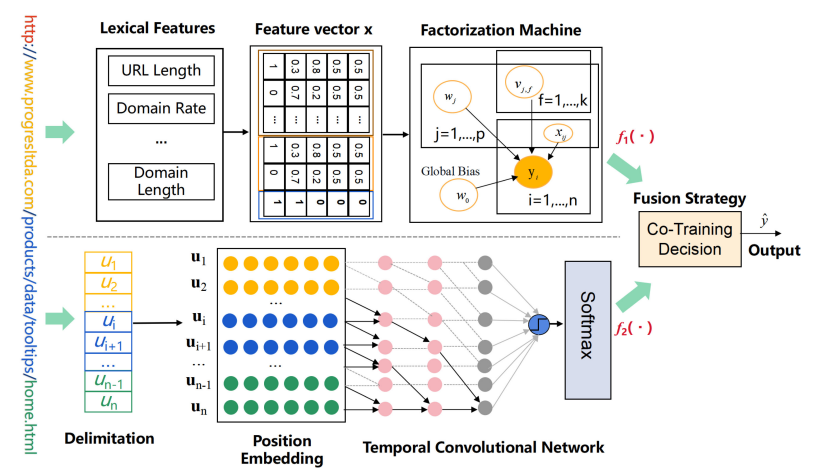


Fig 5 - The architecture of our proposed TCN

## Workflow of the Proposed System

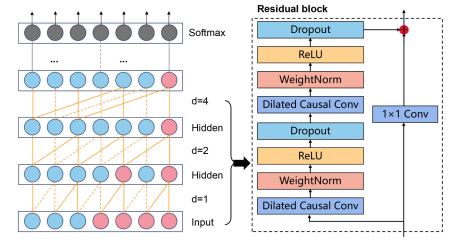
A workflow consists of an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services, or process information. It can be depicted as a sequence of operations, the work of a person or group, the work of an organization of staff, or one or more simple or complex mechanisms.

Workflows may be viewed as one fundamental building block to be combined with other parts of an organization’s structure such as information technology, teams, projects and hierarchies.

The workflow illustrates different modules in Handwritten digit Recognition based on CNN. In this proposed work the numbers are gathered from the MNIST. The data set is directly loaded from keras module and then the number images are read. Initially the numbers are labelled as their respective labels (Ex: 1 as One). These labelled images are pre- processed to remove the unnecessary noisy strokes, stop-digits, slang digits. Then tokenization and stemming is performed. All the numbers are converted entirely into a 2D matrix of binary digits.

Then feature extraction is performed on the obtained images where the extracted features are in a format suitable to sustain directly to machine learning algorithms. Then we use balancing and scoring method to score and labelling images. Then we feed it into the Machine Learning algorithms namely Convolutional Neural Network.

Later, we pass a test image which has a digit written on it which we detect the digit. This gives us the result for each test image of the ten categories namely Zero, One, Two, Three, Four, Five, Six, Seven, Eight, Nine. Therefore, the dataset is thus classified. Then, a model is constructed which recognizes the hand



## Modules To Be Implemented

* + 1. Experimental Setup
    2. Preprocessing
    3. Handwritten Digit Recognition Process
    4. Structures of CNN
    5. SoftMax Regression Algorithm

## MODULE DESCRIPTION

* + 1. **Experimental Setup**

Our experiment is divided into two parts: training and test. For training part, we trained the convolutional neural network 20,000 times. For training part, we use the numbers written by hands and the numbers from the data set to test the system. For test part, the system obtains handwritten data through the camera, and constantly refreshes the predicted output in the window. Since the dataset we trained is the MNIST data set, all the numbers in it are in the form of white characters on a black background.

## Preprocessing

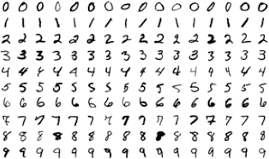
The role of the pre-processing step is it performs various tasks on the input image. It basically upgrades the image by making it reasonable for segmentation. The fundamental motivation behind pre-processing is to take off a fascinating example from the background. For the most part, noise filtering, smoothing and standardization are to be done in this stage. The pre-processing additionally characterizes a smaller portrayal of the example. Binarization changes over a gray scale image into a binary image. The initial approach to the training set images that are to be processed in order to reduce the data, by thresholding them into a binary image. The Figure 2 shows a sample of images taken from the MNIST database.

Fig – 7: Sample images taken from MNIST database

## Handwritten Digit Recognition Process

In this paper, the method is divided into two parts, model training and model test. For model training, according to the loss function, the convolutional neural network continuously updates the network parameters with the data set in MNIST, which contains 60,000 examples. For model test, the system uses the camera to capture the pictures composed of the images generated by the test data set of MNIST and the samples written by different people, then continuously processes the captured graphics and refreshes the output every 0.5 seconds.

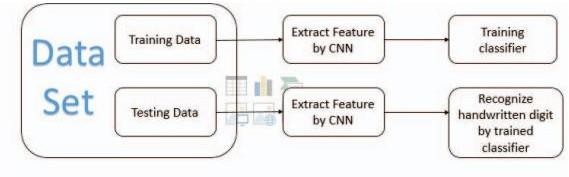


Fig – 8: Framework of handwritten digit recognition.

## Structures of the CNN

First, briefly a neural network is briefly described as shown below.

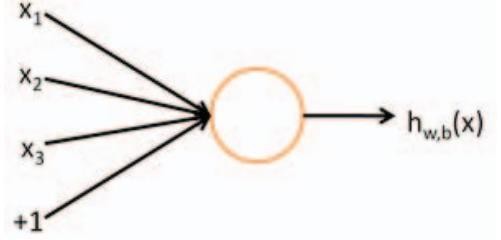


Fig – 9: A simple Neural Network

The corresponding formula is as follows:



This unit can also be called a Logistic regression model. When multiple units are combined in a hierarchical structure, it is the neural network model mentioned in this article.

Convolutional Neural Networks (CNNs) are a special multi-layer artificial neural network whose design purpose is to process a two-dimensional input data matrix. Multiple independent neurons form a plane. Each layer in the convolutional neural network contains In multiple two-dimensional planes, neurons in the same layer are not connected to each other, and neurons in two adjacent layers are connected to each other.

This article can adjust the depth and breadth of the network to change the scale of the CNNs model, and because the weight sharing structure of CNNs makes it more like a biological neural network, it naturally has a good processing ability for natural images.

Therefore, compared with other fully connected networks, CNNs can reduce the scale of the model, simplify the learning complexity, minimize the full-time parameters that need to be trained, and reduce the number of network connections as much as possible.

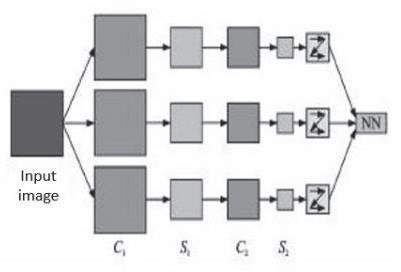


Fig – 10: Simplified structure of convolutional neural network

The structure of a basic convolutional neural network model is generally the same as shown in Figure 2. The network model structure mainly includes two pooling layers (S1, S2) and two convolutional layers (C1, C2). First, the pixel matrix of the original image is used as input, enter C1 through different filters (convolution kernels) for convolution operation and outputs as many maps as the number of convolution kernels. Then those maps are used as the input of the pooling layers S2 for pooling operation. Generally, the maximum pooling is selected, and the output after pooling is used as the input of the convolutional layer C2 to perform the convolution operation again. In this paper, the final output is generally vectorized to facilitate subsequent classification. The convolution kernels in the two convolution layers need to be trained. The training process is to input the training image and compare the input and output, and then use various algorithms (such as gradient descent algorithm) to adjust parameters. The meaning of the convolutional layer is to extract feature and the meaning of the pooling layer is to reduce the amount of data.

The convolutional neural network is divided into three parts. The first layer is the input layer, the input of the picture pixel matrix. The second layer has multiple hidden layers, composed of multiple convolutional layers and pooling layers. The training of this layer is the key to convolutional neural networks. The third layer is generally a fully connected layer for the final output classification.

The above is the main working principle and method of a convolutional neural network. The convolutional neural network built at the beginning does nothing. The weight of each convolution kernel is a meaningless initial value. This article uses labelled image input for training, so that the convolutional neural network can continue Learn, express the gap between the predicted value and the label through various indicators, and constantly update the weight to reduce this indicator, so that the convolutional neural network can finally achieve the desired result of this article.

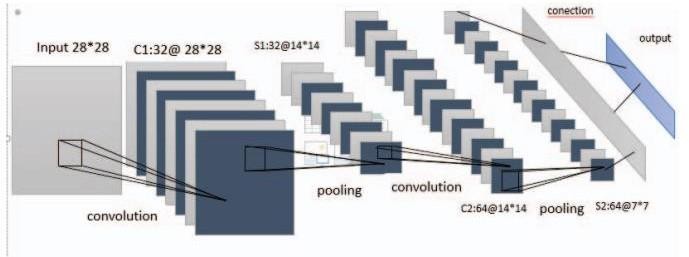


Fig – 11: convolutional neural network in this paper.

## SoftMax Regression Algorithm

SoftMax is generally applied to multi-classification problems. Its function is to map the output of multiple neurons to the (0, 1) interval through function action. This process can be understood as probability, so it can complete multi-classification tasks.

If there is an array V, and the i-th element in the array V is denoted as Vi, then the value of this element after SoftMax regression is:



If you use SoftMax to implement a neural network-based classifier, for example, there are 10 output neurons, then there are 10 categories, category 1, category 2, category 3... all the way to category 10.

Researchers need to use the principle of gradient descent to optimize the gradient of one step size at a time to improve the loss of the classification. Therefore, the researcher has to calculate the partial derivative of Loss for each weight matrix, and then the chain rule is used to obtain the derivative. First of all, in this process, this article will give feedback on the

SoftMax derivative. Through this operation, this article found that after using the SoftMax function, the gradient derivation becomes extremely simple. In this paper, the gradient descent method is used to update the gradient. At this time, cross entropy is used as the loss function. The form of the cross-entropy function is as follows:



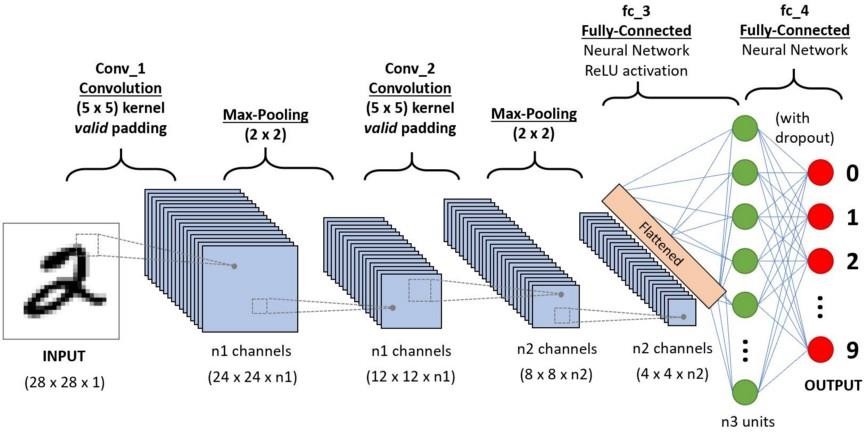


Fig – 12: Sample CNN image Processing

# CHAPTER – 5 IMPLEMENTATION

## Algorithms

* + - Pre-Processing
    - Model Creation & Training
    - Model Evaluation
    - Predictions

## Pre-Processing Algorithm

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

* + - Getting the dataset
    - Importing libraries
    - Importing datasets
    - Finding Missing Data
    - Encoding Categorical Data
    - Splitting dataset into training and test set
    - Feature scaling

48

Pre-Processing code:



Fig – 13: Pre-Processing Code

## Model Creation & Training

Now we will create our CNN model in Python data science project. A CNN model generally consists of convolutional and pooling layers. It works better for data that are represented as grid structures; this is the reason why CNN works well for image classification problems. The dropout layer is used to deactivate some of the neurons and while training, it reduces offer fitting of the model. We will then compile the model with the Adadelta optimizer.

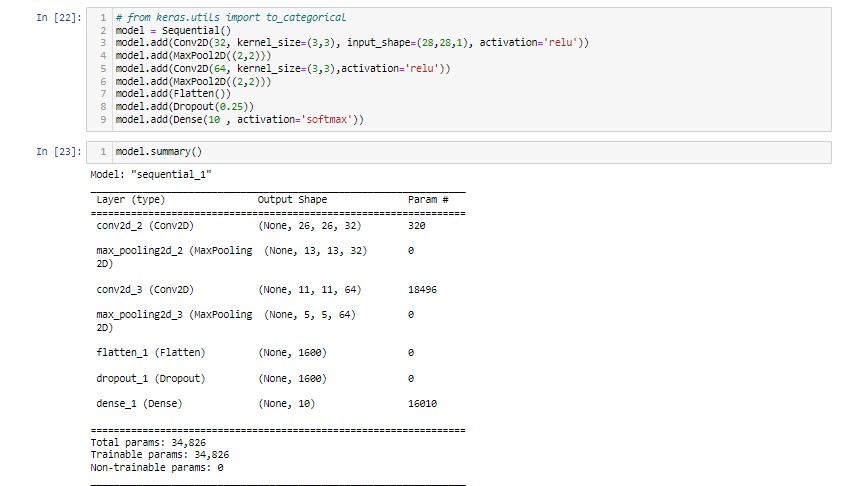


Fig – 14: Model Creation

## Model Training

Model shall be trained using the MNIST dataset. The model.fit() function of Keras will start the training of the model. It takes the training data, validation data, epochs, and batch size.

It takes some time to train the model. After training, we save the weights and model definition in the ‘sp\_digit.h5’ file.



Fig-15: Model Training

## Model Evaluation

We have 10,000 images in our dataset which will be used to evaluate how good our model works. The testing data was not involved in the training of the data therefore, it is new data for our model. The MNIST dataset is well balanced so we can get around 99% accuracy.

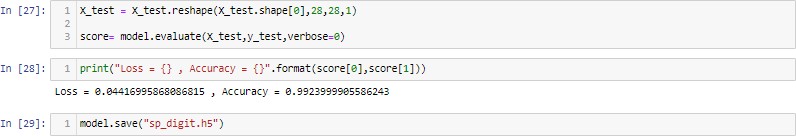


Fig - 16: Model Evaluation

## Accuracy

Accuracy is a metric that generally describes how the model performs across all classes. It is useful when all classes are of equal importance. It is calculated as the ratio between the number of correct predictions to the total number of predictions.



**Model Predictions**

Now for the GUI, we have created a new file in which we build an interactive window to draw digits on canvas and with a button, we can recognize the digit. The Tkinter library comes in the Python standard library. We have created a function predict\_digit() that takes the image as input and then uses the trained model to predict the digit.

(Or)

We need to create a new interface or function where we will upload a photo and get the prediction of the number from classes 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. We will first convert the image to array and apply resizing operation on the array to make it to standard 28 X 28-pixel size. Apply the bitwise not operation to convert to exact black and white screen and digit. Find the digit’s class using our model (sp\_digit.h5).

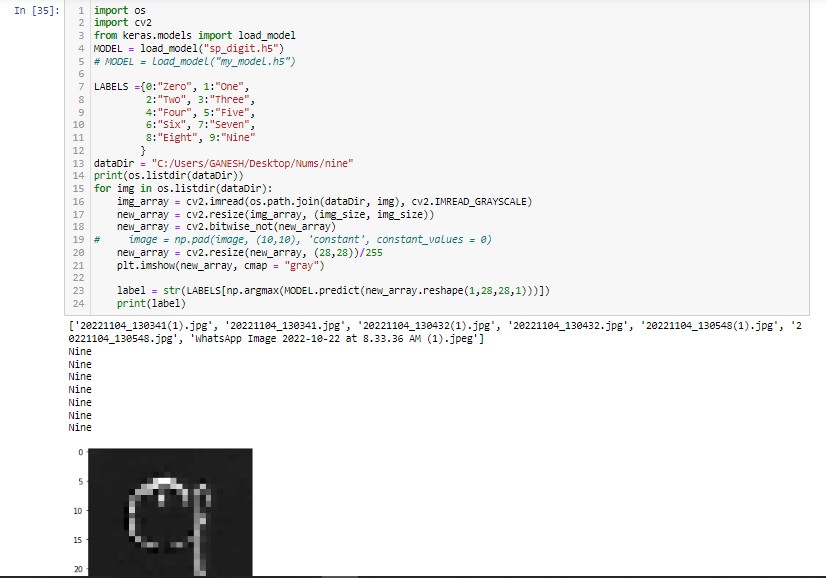


Fig - 17: Model Prediction

## DATASET TO BE USED

MNIST is a well-known handwritten digit data set, founded by the American Association of Scientists. MNIST stands for Modified National Institute of Standards andTechnology

This data set is often used in the field of machine learning recognition. MNIST contains 10,000 test set data and 60,000 training set data, and each handwritten digital picture is in a 28\*28-pixel format. The pictures in this data set are standardized and processed. They are all a 28px\*28px grayscale image in the middle.

As a common data set, MNIST is often used when testing neural networks. This is also the basic application field of the data set. Each data unit of the MNIST data set containstwo parts (the test data set and the training data set are the same): a

picture containing handwritten digits and a corresponding label.

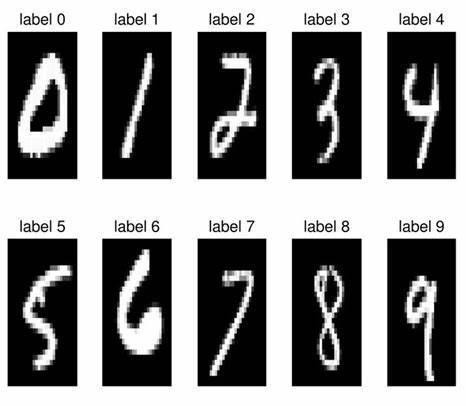


Fig – 18: Test data set of MNIST.

Obviously, the labels of the MNIST data set represent the natural numbers represented by a given picture of handwritten Arabic numerals, that is, ten natural numbersfrom 0 to 9. Therefore, this article can use this vector to identify the label:

For the number n (n is a natural number from 0 to 9), this article records its label as the nth digit as 1, and the rest as 0. For example, the label of 0 is [1,0,0,0,0,0,0,0,0,0], and thelabel of 1 is [0,1,0,0,0,0,0,0,0,0] and so on.

We convert the objects in the MNIST data set into pictures, and the results are as follows:

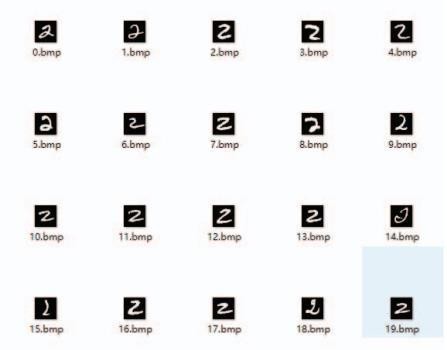


Fig – 19: Number 2 in MNIST.

## Code

import numpy as np import pandas as pd import cv2

import matplotlib.pyplot as plt import os

import tensorflow.keras import keras

from keras.datasets import mnist from keras.models import Sequential

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

import tensorflow

from tensorflow.keras.utils import to\_categorical import tensorflow as tf

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data() X\_train.shape, y\_train.shape,X\_test.shape, y\_test.shape

def plot\_input\_img(i): plt.imshow(X\_train[i],cmap = 'binary') plt.title(y\_train[i])

plt.show() plot\_input\_img(12)

resmap = {'zero':0, 'one':1, 'two':2, 'three':3, 'four':4, 'five':5, 'six':6, 'seven':7, 'eight':8, 'nine':9} img\_size = 28

dataDir = "C:/Users/GANESH/Desktop/Nums"

categories = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'zero'] for category in categories:

path = os.path.join(dataDir, category) for img in os.listdir(path):

img\_array = cv2.imread(os.path.join(path, img), cv2.IMREAD\_GRAYSCALE) # plt.imshow(img\_array, cmap = "gray")

# plt.show()

# print(img\_array.shape)

new\_array = cv2.resize(img\_array, (img\_size, img\_size)) new\_array = cv2.bitwise\_not(new\_array)

# plt.imshow(new\_array, cmap = "gray") # plt.show()

X\_train = np.array(X\_train)

X\_train = np.append(X\_train, [new\_array], axis = 0) y\_train = np.append(y\_train, [resmap[category]], axis = 0)

# print(y\_train[-1], y\_train[-1])

# print(X\_train.shape, new\_array.shape) # plt.imshow(X\_train[-1, :])

# print(y\_train[-1])

# break # break

print(X\_train.shape, y\_train.shape)

# from tensorflow.keras.utils import to\_categorical X\_train = X\_train.astype(np.float32) / 255

X\_test = X\_test.astype(np.float32) / 255 # reshape imgs to 28,28,1

X\_train = np.expand\_dims(X\_train, -1) X\_test = np.expand\_dims(X\_test, -1)

# convert classes to one hot vectors print(y\_test.shape)

y\_train = to\_categorical(y\_train) y\_test = to\_categorical(y\_test) print(y\_test.shape)

# from keras.utils import to\_categorical model = Sequential()

model.add(Conv2D(32, kernel\_size=(3,3), input\_shape=(28,28,1), activation='relu')) model.add(MaxPool2D((2,2)))

model.add(Conv2D(64, kernel\_size=(3,3),activation='relu')) model.add(MaxPool2D((2,2)))

model.add(Flatten()) model.add(Dropout(0.25)) model.add(Dense(10 , activation='softmax'))

model.summary()

model.compile(optimizer='adam' , loss = keras.losses.categorical\_crossentropy , metrics=['accuracy'])

# callbacks

from keras.callbacks import EarlyStopping ,ModelCheckpoint # EarlyStopping

es= EarlyStopping(monitor='val\_acc' , min\_delta= 0.01 , patience=4 , verbose=1) # Model Check

mc=ModelCheckpoint("./bestmodel.h5",monitor='val\_acc' , patience=4 , verbose=1

,save\_best\_only= True) cb=[es,mc]

#model training

X\_train = X\_train.reshape(X\_train.shape[0],28,28,1)

# X\_train\_scaled = X\_train\_scaled.reshape(X\_train\_scaled.shape[0],28,28,1)

his = model.fit(X\_train, y\_train , epochs=50 , validation\_split=0.3 ,callbacks =cb) X\_test = X\_test.reshape(X\_test.shape[0],28,28,1)

score= model.evaluate(X\_test,y\_test,verbose=0)

print("Loss = {} , Accuracy = {}".format(score[0],score[1])) model.save("sp\_digit.h5")

import os import cv2

from keras.models import load\_model MODEL = load\_model("sp\_digit.h5")

# MODEL = load\_model("my\_model.h5")

LABELS ={0:"Zero", 1:"One",

2:"Two", 3:"Three",

4:"Four", 5:"Five",

6:"Six", 7:"Seven",

8:"Eight", 9:"Nine"

}

dataDir = "C:/Users/GANESH/Desktop/Nums/nine" print(os.listdir(dataDir))

for img in os.listdir(dataDir):

img\_array = cv2.imread(os.path.join(dataDir, img), cv2.IMREAD\_GRAYSCALE) new\_array = cv2.resize(img\_array, (img\_size, img\_size))

new\_array = cv2.bitwise\_not(new\_array)

# image = np.pad(image, (10,10), 'constant', constant\_values = 0) new\_array = cv2.resize(new\_array, (28,28))/255 plt.imshow(new\_array, cmap = "gray")

label = str(LABELS[np.argmax(MODEL.predict(new\_array.reshape(1,28,28,1)))]) print(label)

# CHAPTER - 6 TESTING

## Training Process

The loss function is an indicator in the training process. This system uses crossentropy as the loss function between the target category and the predicted category, and training always aims to reduce the loss function. TensorFlow will automatically differentiate the loss function for each variable, and then find the route of gradient descent to update the weight. TensorFlow has a large number of built-in optimization algorithms.

This article uses RMSProp algorithm optimizer tf.train RMSPropOptimizer, learning rate 0.001, attenuation value 0.9, to optimize cross entropy. In each cycle, this article inputs 64 training samples, and then conducts a training session. System output a record after every 100 cycles of operation, a total of 20,000 cycles. After thatthis article believes that the training is over, because at this time, the result of training will not be greatly improved. This is the bottleneck of the system is not insufficient training but the bottleneck of the algorithm structure. It can be seen from the training results that as the number of training steps increases, the recognition accuracy is steadily improving, and the overall recognition accuracy of the final result is 97.6%.

## Testing Process

The system obtains handwritten data through the camera, and constantly refreshes the predicted output in the window. Since the dataset we trained is the MNIST data set, all the numbers in it are in the form of white characters on a black background. In practice, there are very few samples of white characters on a black background, so we selected some numbers from the MNIST data set as recognition samples.

The object is dynamically extracted through the camera, and then recognized by the system, and the window dynamically refreshes the recognition result. Randomly select a picture, the number in the picture we chose here is ‘5’. The results can be seen as follows. As long as the object extraction area (within the red box) is aligned with the object to be

recognized, the output window is constantly refreshing the recognition result with extremely high accuracy rate. For this kind of characters that are not particularly illegible, the performance of the system hardly makes mistakes.



Fig – 20: Recognition of handwriting 5

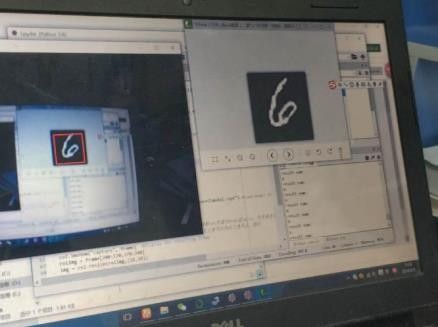


Fig – 21: Recognition of handwriting 6



Fig – 22: Recognition of handwriting 7

The handwritten digit recognition system in this article performed well, but a good perform is not enough. For these pictures with white characters on a black background, the system shows a very superior recognition accuracy. But in the process, it was discovered that because the system recognizes white characters on a black background, if the camera is too far away from the object, it will cause not only the object but also the background behind the object in the object extraction area, where the background is white, which greatly interferes with the recognition. Generally speaking, as long as the input rules are met (there is only object data in the object extraction area), the system has a considerable accuracy rate for handwritten numbers.

Finally, by using the testing data, we can evaluate our model. The following are example classification outputs from our model during testing.

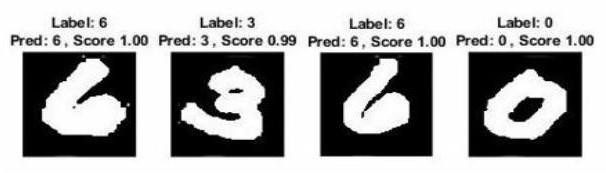


Fig – 23: Some correct recognized outputs

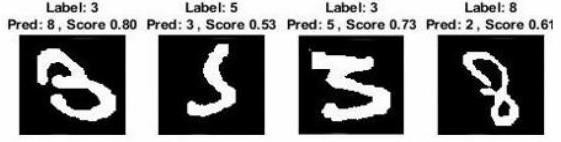


Fig – 24: Some wrongly recognized outputs

Among 10,000 test cases, our model misclassifies total 85 digits after eight epochs which correspond to 99.15% recognition rate shown in Table 1. The results are pretty good for such a simple model with CPU training and less training time (about 30 minutes)

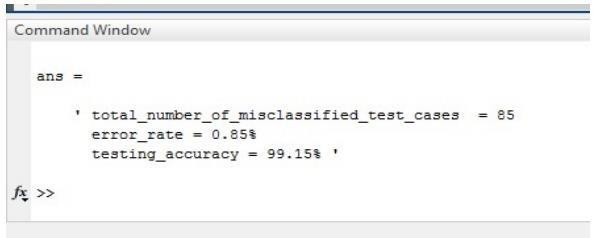


Fig – 25: Error rate and accuracy of our model

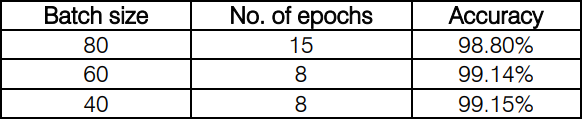
Although there are some digits which are not a good handwriting, our model will be able to classify them correctly.

For example, our model classifies the following image as ‘2’.



Fig – 26: Correct recognition of bad handwriting

**Table 1**: Summary of the experiment



Testing accuracy 99.15% implies that the model is trained well for prediction.

Training set size affects the accuracy and accuracy increases as the number of data increases. The more data in the training set, the smaller the impact of training error and test error, and ultimately the accuracy can be improved.

# CHAPTER – 7 RESULTS

## Actual Results of the Work

After a number of errors, successful elimination of those errors the project is completed with continuous effort. At the end of the project the results can be summarized as follows:

* + - Initially the data set is split into training and testing datasets.
    - The handwritten digits are categorized and the class type is recognized.

The Accuracy of the model and the accuracy is also displayed.

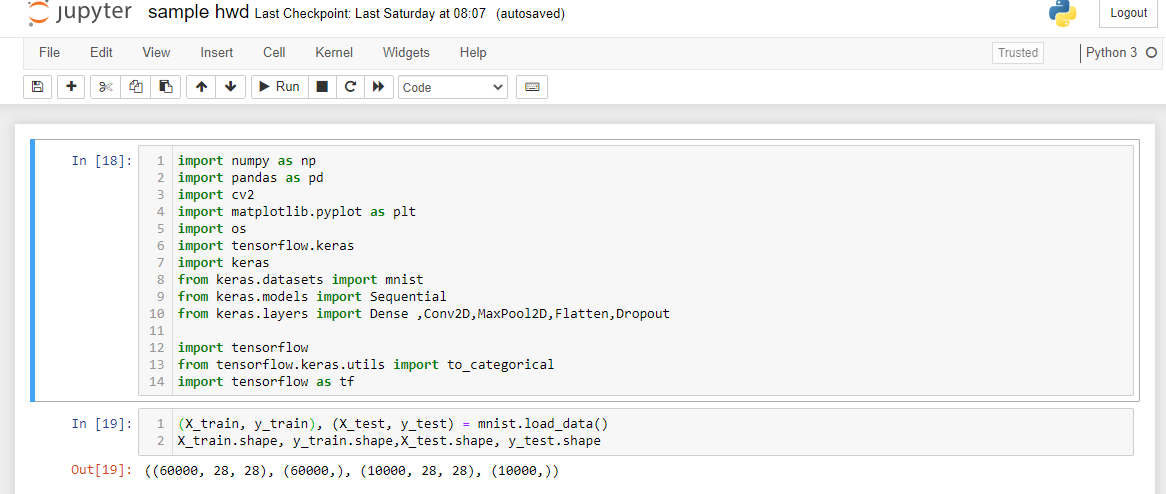


Fig - 27: Import Necessary Libraries and dataset



Fig – 28: Add your own Examples to dataset

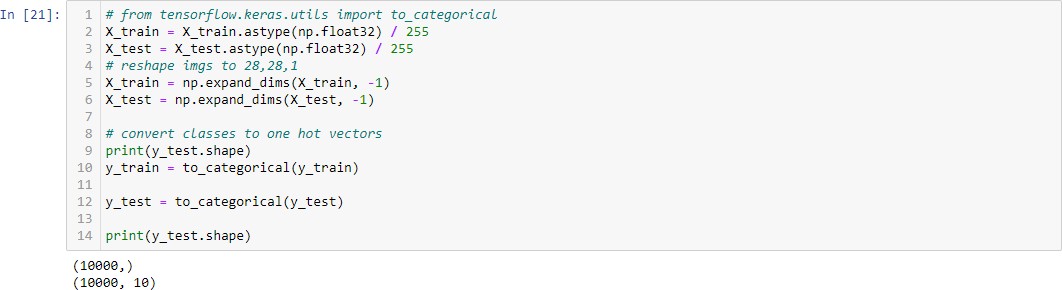


Fig - 29: Pre-Process the dataset

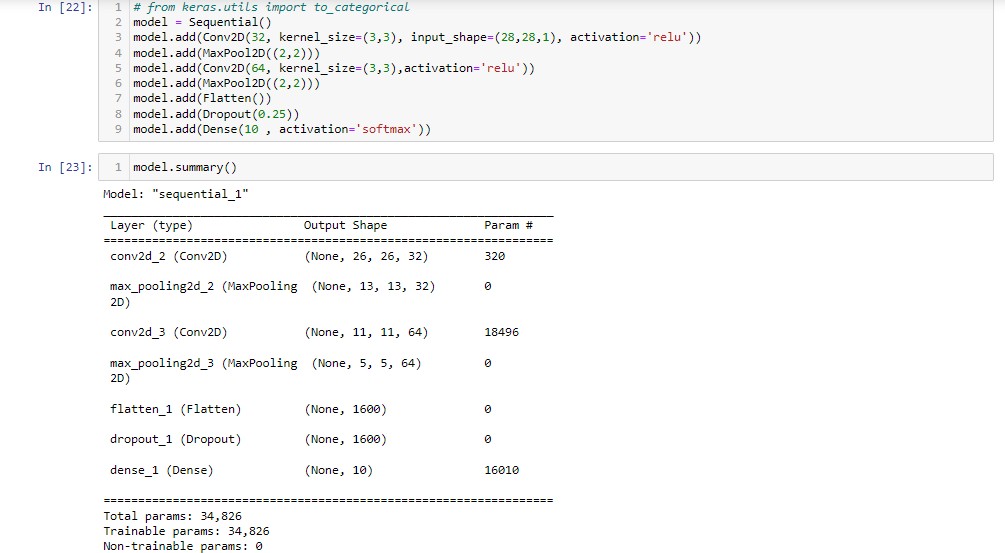


Fig - 30: Create a CNN Model

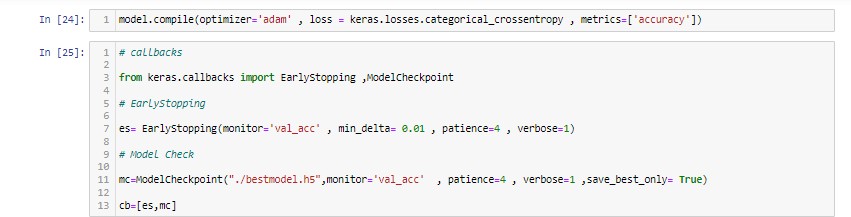


Fig – 31: Create Call backs and check points

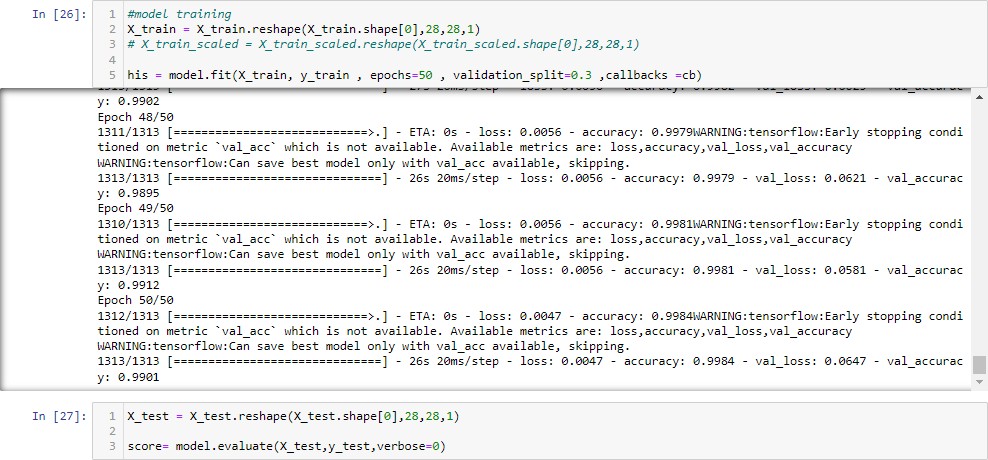


Fig – 32: Train the model

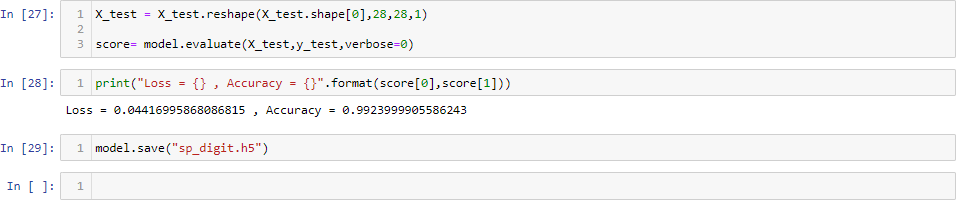


Fig – 33: Check Metrics and Save the model

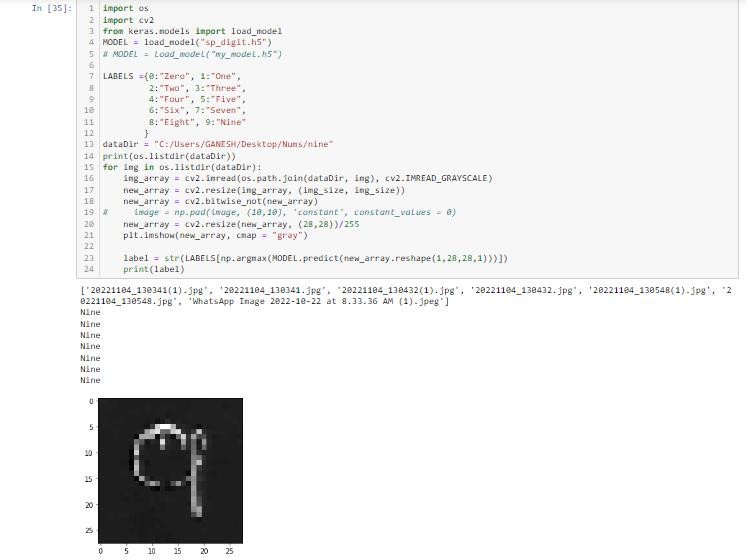


Fig – 34: Output Prediction

Some of our predicted outputs:

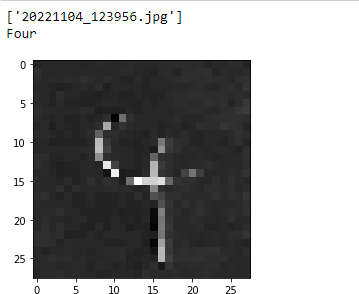


Fig - 35: Four

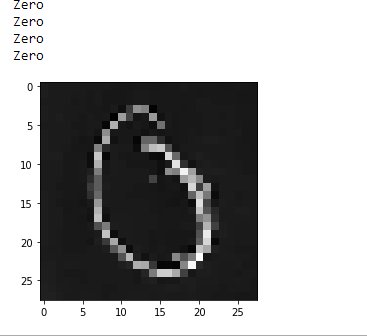


Fig – 36: Zero

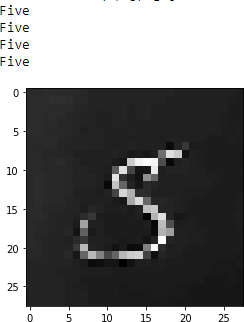


Fig – 37: Five

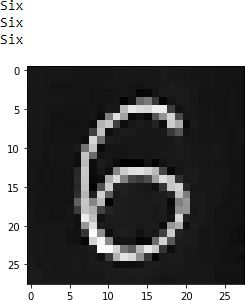


Fig – 38: Six

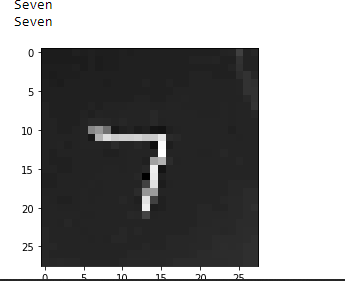


Fig – 39: Seven

# CHAPTER - 8 CONCLUSION AND FUTURE WORK

## CONCLUSION

The focal point of our research lies in detecting the handwritten digits with high accuracy, this paper proposes a new type of handwritten digit recognition system based on convolutional neural network (CNN). In order to improve the recognition performance, the network was trained with a large number of standardized pictures to automatically learn the spatial characteristics of handwritten digits.

In this paper, we have presented a complete system of handwritten digit recognition with high speed and accuracy. Our method has a very high recognition accuracy for handwritten digit recognition written by different person. such as hard and low light illumination, sunny and cloudy weather. We received a satisfied recognition rate in the test process. The advantages are obvious when compare our method with existing tradition method like Histogram of Oriented Gradient (HOG).

## FUTURE WORK

In the future, we plan to add an area detection process to automatically find the number in picture captured by camera and input only the number area to the convolutional neural network, which would significantly improve the system's ability to recognize moving numbers and long-distance numbers. Handwritten digit recognition plays an important role in daily production and life, and this work would make people that work with numbers improve their efficiency and support the intelligent life.

# CHAPTER - 9 REFERENCES

1. NASEER A, ZAFAR K. Meta features-based scale invariant OCR decision making using LSTM-RNN[J]. Computational & Mathematical Organization Theory, 2018(3):1-19.
2. Nassu, B. T. and Ukai, M. (2010). Automatic recognition of railway signs using sift features. In Intelligent Vehicles Symposium, pages 348–354.
3. Creusen, I. M., Wijnhoven, R. G. J., Herbschleb, E., and With, P. H. N. D. (2010). Color exploitation in hogbased traffic sign detection. In IEEE International Conference on Image Processing, pages 2669– 2672.
4. Duan, J. and Viktor, M. (2015). Real time road edges detection and road signs recognition. In International Conference on Control, Automation and Information Sciences, pages 107–112.
5. SHEN J. Orthogonal Caussian - Hermite moments for image charac terization[ C

]//Inelligent Systems & Advanced Manufactur- ing. In termational Society for OPplics and Photonics, 1997: 224 -233.

1. Chan, Tsung-Han, et al. ”PCANet: A Simple Deep Learning Baseline for Image Classification?.” arXiv preprint arXiv:1404.3606 (2014).
2. GORGEVIK D,CAKMAKOV D.Handwritten digit recognition by combining SVM classifiers[C]/The InternationalConference on Co mputer as a Tool.IEEE 2005:1393-1396.
3. BABU U R,CHINTHA A K,VENKATESWARLU Y.Handwritten digit recognition using structural, statistical features and knearest n eighbor classifier[J].International Journal of Information Engineeri ng&Electronic Business,2014.6(1):62-68.
4. Gidaris, S. and Komodakis, N. (2015). Object detection via a multi1region and semantic segmentation-aware cnn model. In IEEE International Conference on Computer Vision, pages 1134–1142.
5. Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. Computer Vision and Pattern Recognition, 157(10):3642–3649.
6. Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In International Conference on Neural Information Processing Systems, pages 1097–1105.
7. BAEK Y, LEE B, HAN D, et al. Character region awareness for text detection[C]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.