University Of Vavuniya

Automatic Handwritten Digit Recognition, Text Recognition, and Arithmetic Computation Using Convolutional Neural Networks

A Thesis Submitted for the Partial Fulfilment of the Course Unit TICT4116 for the Bachelor of Information and Communication Technology Honours (BICT Honours) Degree Programme.

Department of Information and Communication Technology
Faculty of Technological Studies
University Of Vavuniya
2026

University Of Vavuniya

Automatic Handwritten Digit Recognition, Text Recognition, and Arithmetic Computation Using Convolutional Neural Networks

By

Group - 43

Supervisor:

MR. R. Ravichandran

A Thesis Submitted for the Partial Fulfilment of the Course Unit TICT4116 for the Bachelor of Information and Communication Technology Honours (BICT Honours) Degree Programme.

Department of Information and Communication Technology
Faculty of Technological Studies
University Of Vavuniya
2026

Declaration of Authorship

We, declare that the thesis entitled **Automatic Handwritten Digit Recognition, Text Recognition, and Arithmetic Computation Using Convolutional Neural Networks** and the work presented in it are done by ourself. We confirm that:

- ⊙ this work was done wholly or mainly while candidate for a BICT Honours degree at this University;
- ⊙ where any part of this report has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated;
- ⊙ Where We have quoted from the work of others, the source is always given;
- ⊙ We have acknowledged all main sources of help.

Group Members

Reg. No	Index No	Name	Signature
2019/ICTS/165	TS- 5165	K.Kajanthan	
2019/ICTS/158	TS-5158	V.V.W. Peries	
2019/ICTS/118	TS-5118	B.R.I.S. Herath	

Date:....................

This attests to the fact that this thesis was developed under my guidance. The thesis is of acceptable standards and has been produced in accordance with the guidelines.

Certified by,

Supervisor:MR. R. Ravichandran

Supervisor . . . .   Date:. . . . .

Acknowledgement

We will always be grateful to Mr. R. Sakuntharaj, our research coordinator of the Final-Year Research and Development Project. His clear direction, steady encouragement, and honest feedback kept us on track from the very beginning. His patience, high standards, and open-door policy transformed long debugging nights into valuable learning moments, shaping both the project and our growth as researchers.

Sincere thanks also go to MR. R. Ravichandran, our supervisor, for the sharp questions, fresh ideas, and quick emails that forced us to think deeper and raise the quality of every experiment and every line we wrote.

We would like to acknowledge the faculty and non-faculty staff who helped behind the scenes. In particular, Mr. V. Senthooran, Dean of the Faculty of Technology, University of Vavuniya, for cheering us on and making sure the faculty remained a place where research can actually happen. Our deep appreciation also goes to Mr. V. Vinoharan, Head of the Department of Information and Communication Technology, for smoothing out paperwork, unlocking labs, and always finding the equipment we needed—often before we even asked. To our families: thank you for the countless "How is the project going?" calls, the surprise snacks, and the patience when we cancelled weekend plans. Your quiet faith gave us the energy to cross the finish line.

Finally, we would like to thank our Juniors, who worked with us to collect sample images for our "MathSet" dataset, and also we tip our hats to everyone who helped in big ways or small—friends who proofread posters, lab mates who shared code, and even the security guard who let us in after midnight. This journey was lighter because all of you were walking it with us.

ABSTRACT

Handwritten digit recognition and arithmetic computation are very important tasks regarding artificial intelligence and the potential applications one may consider include education systems, digital calculators, and automated evaluation tools. Although most existing systems concentrate just on recognizing handwritten digits, to a large extent working with full-fledged arithmetic expressions remains a challenge mostly due to variations in handwriting styles and symbol structures.

We propose a novel system for automatic recognition of handwritten digits and arithmetic symbols in turn consecutively followed by arithmetic computation via Convolutional Neural Networks (CNNs). The study included the development of a custom handwritten dataset named MathSet, as none of the publicly available datasets sufficiently captured the essence of handwritten arithmetic expressions. MathSet comprises handwritten digits and arithmetic operators and bracket symbols collected from different individuals.

The training started on a dataset that consisted of very few samples and had high image resolution. For this kind of training, accuracy became low. Later, to boost performance, the training went on with an optimized dataset that had reduced image size but a larger number of samples. This enhanced dataset allowed better training of the model, which resulted in an accuracy of around 98% for the final CNN.

The experiments verify that CNN-based models trained on a well-curated dataset can achieve effective recognition of handwritten digits and symbols with reliable arithmetic computation. The study highlighted the significance of dataset quality and model architecture in the design of practical handwritten recognition systems.

Contents

List of Figures

List of Tables

Abbreviation

- AI - Artificial Intelligence
- API - Application Programming Interface
- MNIST - Modified National Institute of Standards and Technology
- CNN - Convolutional Neural Network
- DL - Deep Learning
- GPU - Graphics Processing Unit
- HMER - Handwritten Mathematical Expression Recognition
- SVHN - Street View House Numbers
- SVM - Support Vector Machine
- SGD - Stochastic Gradient Descent

Chapter1

Introduction

Handwritten digit recognition is a significant area of research in artificial intelligence and machine learning. Unlike typed text and numbers, computers find the understanding of handwritten content to be a huge task. Mainly because everyone's handwriting is different, and the same digit or symbol may be present in many forms.

There has been a great deal of progress in the deep learning of recent years, particularly in tasks related to image recognition. Among the tools used in such applications, the Convolutional Neural Network (CNNs) constitute the main core because they are built to learn automatically from images, particularly among high-performing networks recognizing handwritten digits and symbols. The trained CNN-based model will learn the features like curves, edges, and strokes directly from the image-data to be able to recognize handwritten digits and symbols.

This research is indeed aimed at developing a digit and symbol recognition system that would automatically do some basic arithmetic calculations. This system can cater to a wide variety of real-world applications involving things such as tutoring aids, automated grading systems, digital calculators, and document processing systems. The automation of such tasks can markedly reduce human effort and lowers down the error possibilities.

Most of the digit recognition systems, as found in existing literature, are trained and tested on standard datasets, among which MNIST is quite common. While these datasets were useful for digit recognition, they were short of representing real-world handwritten arithmetic expressions' uniqueness. Based on this shortcoming, we developed a custom handwritten dataset called MathSet using different individuals with their own handwriting variations to include handwritten digits, arithmetic operators, and brackets.

The recognition system under study uses the trained CNN model on the MathSet dataset to read handwritten inputs. Following recognition, the system formulates an arithmetic expression and automatically computes the final answer. Different dataset configurations were experimented with to achieve image quality with optimal recognition.

The study indicates that a finely crafted CNN model on a well-defined dataset will provide the best possible result in recognition and arithmetic calculation efficiency. The findings derived from this study also underscore the potential for the practical application of deep learning approaches in developing handwritten recognition systems.

1.1. Objectives

The major aim of this research work is to develop and apply an intelligent system for the recognition of mathematical expressions written by hand, which can automatically detect the handwritten mathematical expressions and calculate the resulting values accordingly. For this purpose, the system aims to deal with the challenges associated with the mathematical expressions written by hand, since handwriting is different in terms of writing style and mathematical expression shapes. Therefore, the proposed project aims to develop a more reliable handwritten mathematics recognition system.

One of the main goals in this research work is to design a personalized dataset, known as MathSet, which is designed with the objective of handwritten recognition of mathematical expressions. This dataset is generated with 28x28 pixels from the digit image taken from the original .mat files in the MNIST dataset, along with handwritten mathematical symbols such as addition, subtraction, multiplication, and division, done by a wide range of people. A light CNN is then designed to recognize the dataset for identifying efficient computational complexity in its classification based on the discriminative features between the digits and the symbols.

Another critical objective is the incorporation of the arithmetic evaluation module, which evaluates the predicted symbols by the CNN and gives the mathematical solution. The proposed system should be modular and flexible so that in the future, further functionality can be added, such as multi-digit number recognition, mathematical operations, and equations. If the proposed system is successfully implemented, then it can be used in educational technologies, automatic evaluation systems, and learning systems, especially for teachers and primary school students who solve mathematical equations and problems by writing math equations by hand.

1.2. Contribution

This thesis makes several significant contributions:

MathSet – A new 28×28 grayscale set that mixes cleaned MNIST digits with real-world $+$, $-$, $\times$, $\div$ symbols.

A CNN model tuned for both digits and operators that hits high accuracy without demanding heavy computation.

An end-to-end pipeline: ink $\rightarrow$ recognition $\rightarrow$ calculation $\rightarrow$ numeric result, no human in the loop.

Proof that smart preprocessing plus CNNs can give reliable results even when you don't have millions of samples. A plug-and-play code base that can stretch to longer numbers, extra operators, or tougher math later on. A ready-to-use tool for digital learning platforms, auto-grading portals, and assistive tech that reads handwritten math.

1.3. Organization of Thesis

The remainder of this thesis is organized as follows:

Chapter 2 – Related Work: This chapter reviews existing research on handwritten digit recognition, mathematical symbol recognition, CNN-based image classification, and arithmetic expression evaluation. Limitations in existing approaches are discussed, motivating the proposed solution.

Chapter 3 – Methodology: This chapter describes the dataset preparation process, including MNIST digit extraction and handwritten symbol collection, preprocessing steps, CNN model architecture, training procedures, and arithmetic computation logic.

Chapter 4 – Experimental Results and Discussion: This chapter presents experimental results and performance evaluation using metrics such as accuracy, precision, recall, and confusion matrices. The effectiveness and limitations of the proposed system are discussed.

Chapter 5 – Conclusion and Future Work: This chapter summarizes the research outcomes and discusses possible future improvements, including support for complex expressions, deeper CNN architectures, and real-time deployment.

1.4. Scope and Limitations

The comprehensive vision of this research, as reflected in the title, is to build a unified system for digit, symbol, and text recognition. However, the scope of this specific thesis is centered on the Arithmetic Computation Module. Current Implementation: The system is fully trained and validated to recognize and compute handwritten digits (0-9), arithmetic operators ($+$,-,$\times$,$\div$) and grouping symbols (brackets).

Future Expansion: While the CNN architecture utilized in this research is capable of feature extraction for alphabetic characters (Text Recognition), the implementation of algebraic variable recognition (e.g., equations involving x,y,z) is reserved for the next development phase. This research establishes the foundational arithmetic engine required before integrating complex algebraic text parsing.

Chapter2

Related Work

Deep learning techniques have been extensively applied to the field of Optical Character Recognition (OCR) and handwritten text analysis, utilizing Convolutional Neural Networks (CNNs) and various hybrid architectures. In this chapter, we review the major contributions of recent research in handwritten digit recognition, the evolution of sequence modeling for mathematical expressions, and the attempts to integrate reasoning with visual perception.

2.1. Evolution of CNNs for Digit Recognition

CNNs have become the standard in addressing the problem of feature extraction in computer vision. Early foundational work by LeCun et al.[12] introduced LeNet-5, a pioneering CNN architecture designed for handwritten digit recognition on the MNIST dataset. This model demonstrated that neural networks could learn spatial hierarchies of features automatically, achieving over 99% accuracy on isolated digits. Building on this, Goodfellow et al.[9] pioneered a unified approach for multi-digit recognition in unconstrained environments using Deep CNNs. Their research moved away from the traditional segmentation-then-recognition process, instead proposing a single integrated network trained on the Street View House Numbers (SVHN) dataset, achieving a per-digit accuracy of 97.84%. Similarly, Hossain and Ali [10] further validated the effectiveness of simplified CNN architectures for digit recognition, confirming that automated feature extraction significantly outperforms manual techniques.

2.2. Optimization Strategies in Modern Architectures

To optimize these architectures, Ahlawat et al.[1] challenged the trend of using complex "Ensemble" methods by focusing on a "Pure CNN" approach. They conducted a comprehensive analysis of hyperparameters and optimization algorithms, concluding that the Adam optimizer [11] provided the fastest convergence and highest stability. Their findings resulted in a record-breaking accuracy of 99.87% on the MNIST dataset. To prevent such deep networks from overfitting on training data, techniques such as

Dropout, introduced by Srivastava et al. [16], have become standard practice, ensuring that models learn generalized features rather than memorizing noise.

2.3. Limitations in Symbol Recognition Accuracy

However, a critical analysis of the literature reveals that maintaining high precision on complex mathematical symbols remains a significant challenge, with many systems failing to surpass the 96% accuracy threshold. Alwzwazy et al.[2] highlighted a "calculation gap" in digit recognition, reporting a maximum accuracy of only 95.70% due to the lack of sufficient data augmentation. The challenge intensifies when the scope is expanded to diverse mathematical symbols. Ayeb et al. [4] attempted to classify a massive set of 101 different handwritten symbols using Transfer Learning but achieved a surprisingly low accuracy of 83.68%. This highlights the "Curse of Dimensionality," where increasing class diversity without custom data leads to model confusion. Similarly, Nazemi et al. [13] compared lightweight architectures like LeNet and SqueezeNet for offline symbol recognition but concluded with a maximum accuracy of 90.00%, proving that older architectures lack the depth required to distinguish between geometrically similar operators.

2.4. Dataset Diversity and Expression Parsing

Furthermore, the challenge of dataset diversity has been a recurring theme. Baldominos et al.[5] conducted a survey showing that models optimized solely for standard benchmarks like MNIST often fail to generalize to "messy" real-world handwriting. They emphasized the need for data augmentation techniques, such as the elastic distortions proposed by Simard et al. [15], to artificially expand training sets and improve robustness. In the specific domain of mathematical expression recognition, Zhang et al. [19] [20]introduced advanced "Attention-based" models (including the TAP framework) to parse full equations. However, while their symbol detection was acceptable, these systems were designed primarily for transcription (generating LaTeX text) rather than computation, leaving a functional gap in the creation of an active problem-solving tool. Nguyen et al. [7] also explored this domain, highlighting the persistent issue of "symbol confusion" (e.g., 'x' vs. '$\times$') which necessitates a specialized dataset for resolution.

2.5. Structural Analysis and The "Meaning" Gap

Beyond simple symbol recognition, understanding the spatial structure of mathematics is critical. Zanibbi and Blostein [18] conducted an extensive survey distinguishing between "Offline" (image-based) and "Online" (stroke-based) recognition, highlighting that offline systems face significantly greater challenges in resolving structural ambiguities (e.g.2^x vs $2x$)). To address this, Awal et al. [3] proposed global learning approaches that utilize context to correct segmentation errors, proving that a symbol's identity often depends on its neighbors. While recent advancements like the work of Deng et al. [8] have successfully utilized encoder-decoder models to translate images directly into LaTeX markup, these systems function as "transcribers" rather than "solvers." They generate code for display but lack the semantic understanding required for computation. Furthermore, fundamental preprocessing techniques established by Otsu [14] (thresholding) and Suzuki and Be [17] (contour analysis) remain the industry standard for segmentation, yet modern end-to-end deep learning frameworks often overlook these robust classical methods in favor of purely data-driven approaches. By leveraging modern frameworks like Keras [6], our research aims to combine these established structural analysis techniques with deep learning to create a system that not only transcribes but computes.

2.6. Summary and Research Foundation

The aforementioned studies amply illustrate the difficulties in visual recognition and the need for data augmentation to handle diverse handwriting styles. As noted by Simard et al. [15], increasing dataset diversity is key to model generalization. Nevertheless, current approaches frequently break up the issue by concentrating only on either digit recognition or expression transcription, failing to close the loop and produce a computed outcome. We propose a hybrid End-to-End Visual Calculator based on this body of work. The recognition process is powered by our unique "MathSet", which was specifically gathered to cover digits and arithmetic symbols. It is then combined with a powerful symbolic reasoning engine. This method sets a new standard for automated handwritten arithmetic solving by addressing the drawbacks of symbol confusion, a lack of dataset diversity, and the "calculation gap" present in purely neural approaches.

Chapter3

Methodology

3.1. Introduction

This chapter describes the entire process involved in developing the system of handwritten digit recognition and arithmetic computation. The work mainly involves the collection of handwritten data, preprocessing of images, training the Convolutional Neural Network (CNN), and performing arithmetic calculations based on the recognized inputs. The accuracy of the configured system was improved using different dataset configurations.

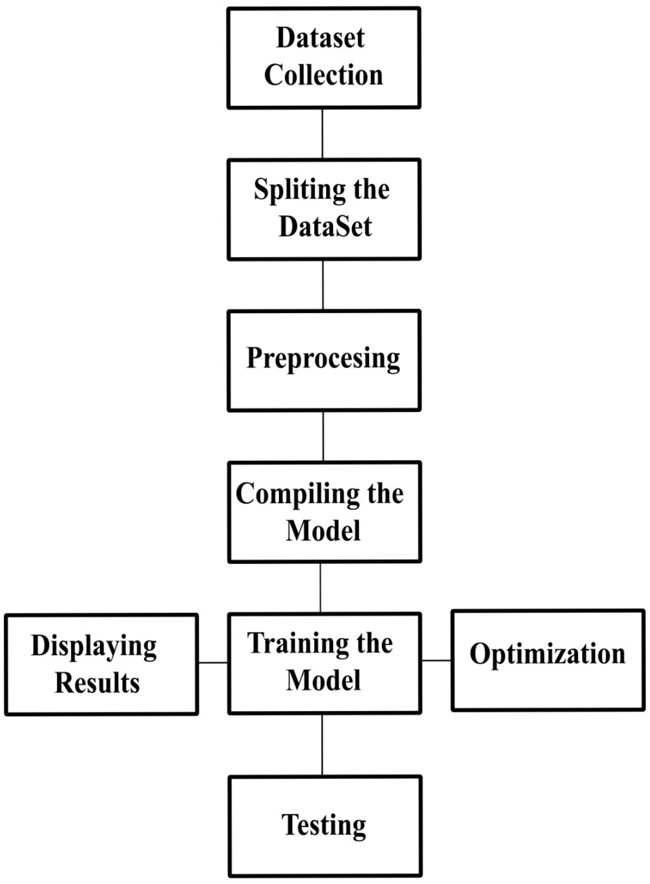


Figure 3.1: Proposed Methodology

3.2. Dataset Collection

The handwritten mathematical dataset for the present research was collected by us and was then named MathSet. The dataset contains handwritten digits as well as the basic arithmetic symbols collected from different people so that the dataset would cover various types of handwriting styles. Initially, we fed 45 $\times$ 45 pixel-sized images with each digit or symbol having a total of 2000 images.

The dataset was used for training the CNN model, but it fails to yield satisfactory results. The reason behind this is that the model was not able to generalize properly because of the very limited sample size. In order to overcome this, a second dataset was collected; however, in this case, the image size was reduced to 25 $\times$ 25 pixels, and more than 18000 images for each class were taken. Apart from digits and operators, bracket symbols "(" and ")" were also included. This enhanced dataset helped the model learn better and produced more accurate results.

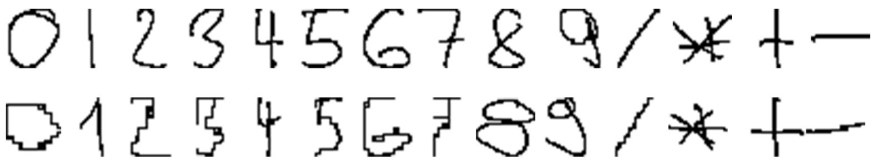


FIGURE 3.2: Image samples

3.3. Dataset Classes

The final MathSet dataset consists of:

Handwritten digits from 0 to 9

Arithmetic operators (+, -, ×, ÷)

Brackets '(' and ')'

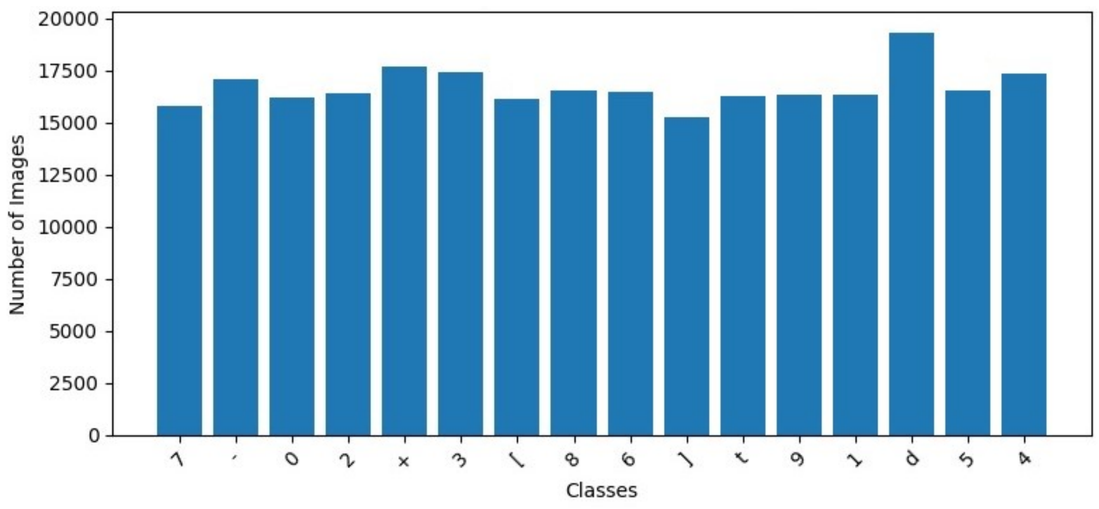


FIGURE 3.3: Dataset class distribution

These classes allow the system to recognize and compute basic arithmetic expressions.

3.4. Image Preprocessing

All images are preprocessed before training the model to improve recognition performance. First, the images are made grayscale to minimize complexity. Noise is removed to make the handwritten strokes clearer. Thresholding was applied to separating writing from the background. Finally, pixel values were normalized by which the model would train more efficiently. The preprocessing steps enabled the CNN model to extract meaningful characteristics from the handwritten images.

3.5. System Workflow

The procedure of the system is pretty simple. A handwritten image is input and processed through a preprocessing stage. The processed image is subsequently supplied into the CNN model that predicts whether that image is a digit, an operator, or a bracket.

After recognition, the predicted symbols will be arranged in the right order to form an arithmetic expression. Finally, this expression is passed to perform the arithmetic operation required.

3.6. CNN Model Design

Using a Convolution Neural Network was necessary because it performs excellently in all image-recognition tasks. The CNN model automatically learns the features that matter, such as edges and stroke patterns, instead of requiring the features to be specified a priority. Convolution layer which will serve purpose of each feature extraction includes pooling layer for reducing image size and finally fully connected layers would classify it. Some dropout classes are also included in the model to hide some neurons between training and make it more generalized to avoid overfitting.

3.7. Training and Experiments

MathSet will be split into three datasets; Training, Validation, and Test Set. The training phase used initially the dataset with a matching size of 45×45 pixels and with 2,000 images per class, which resulted in low accuracy in the end.

Then it was retrained after using the improved dataset, which included 25 x 25 pixel images and more than 18000 samples per class with bracket symbols included. The training used the Adam optimizer and categorical cross-entropy loss function. The performance was largely improved in the second experiment.

3.8. Arithmetic Computation

After recognizing the numerical characters and symbols from the images, the interpretation of them happens in the format of a mathematical expression. The current state identifies the operands and operators and performs the operations associated with them. The value of the final computation is shown to the user.

3.9. Evaluation Results

The performance of the systems is dictated by accuracy measurements. It may be noted that the first dataset configuration had the lowest accuracy. With an increased size of dataset and improved quality on image resolution, the better MathSet dataset achieved an accuracy of nearly 97.52%.

3.10. Summary

Methodology of building the handwritten digit recognition and arithmetic computation systems is described in this chapter. With the enriched dataset and improved training process, a final model could be achieved that turned out to be very accurate as well as dependable.

Chapter4

Results and Discussion

This chapter describes how the proposed system was tested and explains the results obtained from the experiments. The aim of these experiments was to evaluate how well the Convolutional Neural Network (CNN) performs in recognizing handwritten digits and mathematical symbols and how accurately the system evaluates arithmetic expressions.

4.1. Experimental Set Up

The experimental work was carried out using Python along with deep learning libraries such as TensorFlow and Keras. The proposed CNN model was trained and tested using handwritten digit and mathematical symbol images collected specifically for this research.

In the initial stage, handwritten images were prepared with a resolution of 45×45 pixels. For each digit and symbol class, about 2,000 images were used. The images were converted into grayscale format and normalized before training. The dataset was divided into training and testing sets to measure the performance of the model.

Although the model learned basic features, the accuracy obtained from this setup was not satisfactory. To improve the performance, a second dataset was created. In this improved setup, image resolution was reduced to 25×25 pixels, and the number of samples per class was increased to 20,000. Additional mathematical symbols, including '(' and ')', were also included to make the dataset more practical for arithmetic expressions.

The CNN architecture consisted of convolution layers to extract important features, pooling layers to reduce dimensionality, and fully connected layers for classification. After classification, the predicted symbols were passed to an arithmetic evaluation module that calculated the final result of the handwritten expression.

4.2. Experimental Results

The experimental results show a clear improvement in performance after modifying the dataset and training conditions.

During the first experiment, which used 45×45 pixel images with limited samples, the model achieved an accuracy of around 90-95%. Errors were mainly observed when recognizing similar-looking symbols, indicating that the dataset was not sufficient for effective learning.

In the second experiment, where 25×25 pixel images and a larger dataset were used, the model performed significantly better. The CNN was able to learn more consistent features due to the increased number of training samples. As a result, the overall classification accuracy increased to approximately 97.52% on the test dataset.

The arithmetic evaluation module also produced correct results for most test cases, showing that the combination of symbol recognition and expression evaluation works effectively.

4.3. Training vs Validation Accuracy

From the Training vs Validation Accuracy graph, the process of improvement in the accuracy of the model with each epoch is visible. There is a constant rise in the values of the training accuracy, which denotes that the model is accurately learning the data. In a similar manner, the validation accuracy is also showing a rise with small fluctuations, which is almost comparable to the values of the training accuracy. Due to the small variations in both of these values, it is clear that the model is performing well on both the data it has been trained on and the data it has not been trained on.

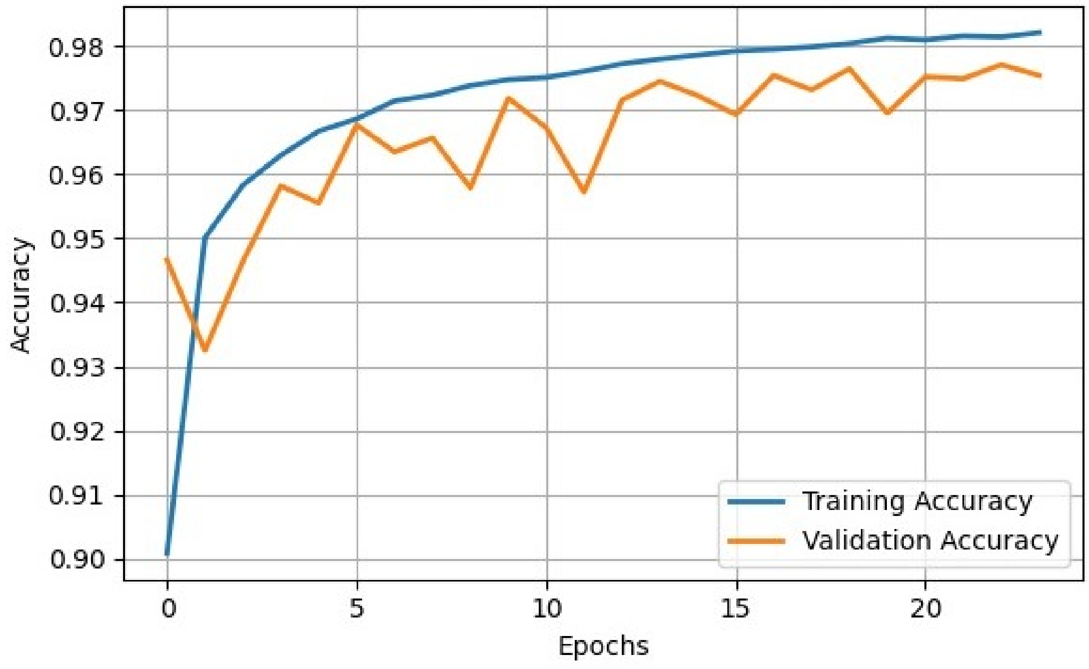


FIGURE 4.1: Training vs validation Accuracy

4.4. Training vs Validation loss

From the Training & Validation Loss graph, there is an overall decrease in the training loss with little difference to the validation loss. This is an indication that there is optimal model training with no signs of overfitting. The Training & Validation Loss graph also presents similar trends with small fluctuations.

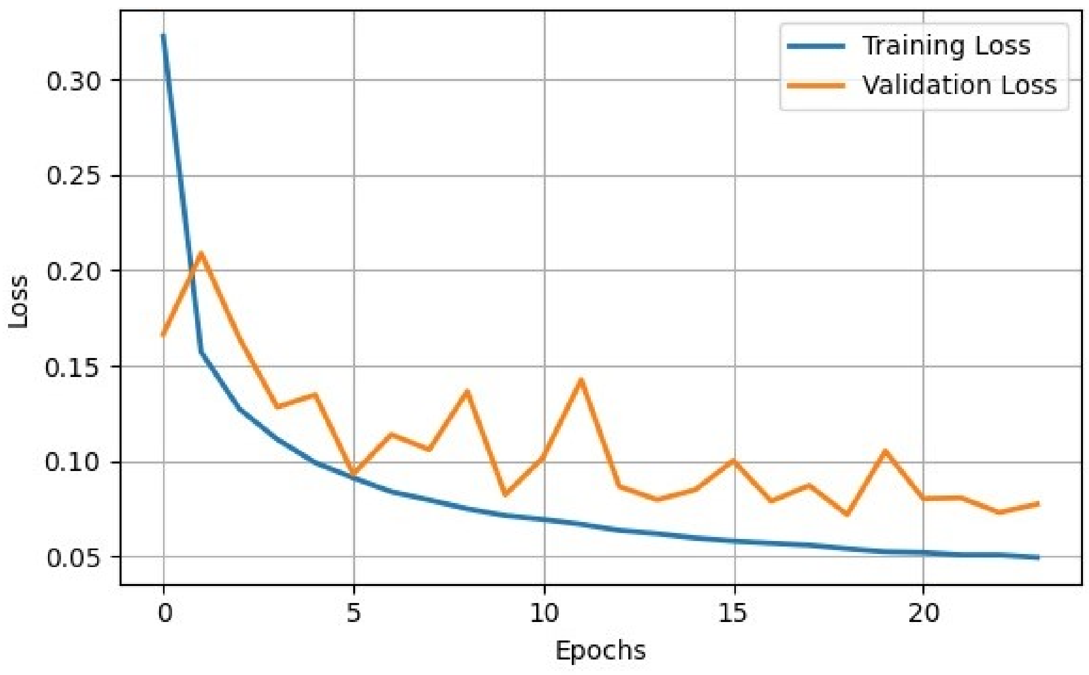
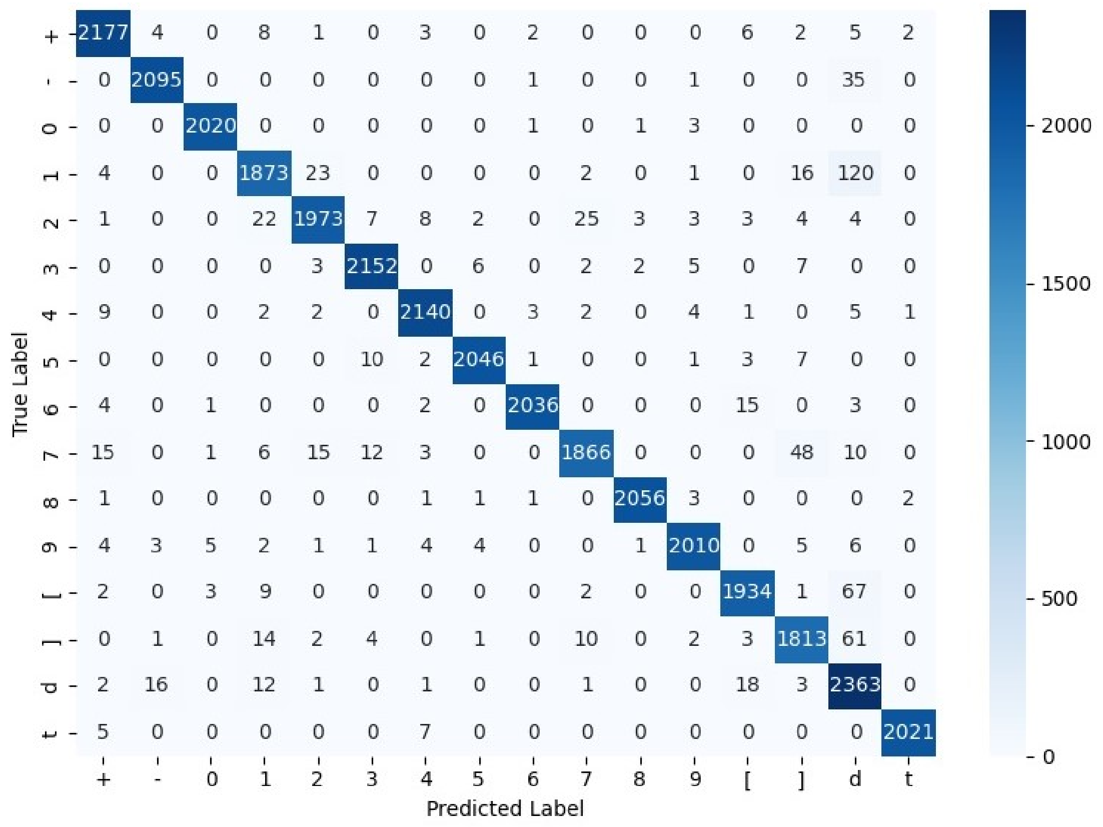


FIGURE 4.2: Training vs Validation loss

4.5. Confusion Matrix

The confusion matrix reveals the performance of the CNN model in recognizing the handwritten digits and mathematical operators. The high values on the diagonal show the successful detection and recognition of the handwritten digits and mathematical operators by the model.

Misclassification errors are found to be small, occurring primarily between closely resembling figures (such as 1 & 7) and between mathematical symbols. The division (d) and multiplication (t) operators are correctly identified with very few errors, establishing that these mathematical operators, along with figures, can be aptly discriminated by the system. The confusion matrix proves that the designed handwritten mathematical recognition system is robust.



	+	-	0	1	2	3	4	5	6	7	8	9	[	]	d	t
+	2177	4	0	8	1	0	3	0	2	0	0	0	6	2	5	2
-	0	2095	0	0	0	0	0	0	1	0	0	1	0	0	35	0
0	0	0	2020	0	0	0	0	0	1	0	1	3	0	0	0	0
1	4	0	0	1873	23	0	0	0	0	2	0	1	0	16	120	0
2	1	0	0	22	1973	7	8	2	0	25	3	3	3	4	4	0
3	0	0	0	0	3	2152	0	6	0	2	2	5	0	7	0	0
4	9	0	0	2	2	0	2140	0	3	2	0	4	1	0	5	1
5	0	0	0	0	0	10	2	2046	1	0	0	1	3	7	0	0
6	4	0	1	0	0	0	2	0	2036	0	0	0	15	0	3	0
7	15	0	1	6	15	12	3	0	0	1866	0	0	0	48	10	0
8	1	0	0	0	0	0	1	1	1	0	2056	3	0	0	0	2
9	4	3	5	2	1	1	4	4	0	0	1	2010	0	5	6	0
[	2	0	3	9	0	0	0	0	0	2	0	0	1934	1	67	0
]	0	1	0	14	2	4	0	1	0	10	0	2	3	1813	61	0
d	2	16	0	12	1	0	1	0	0	1	0	0	18	3	2363	0
t	5	0	0	0	0	0	7	0	0	0	0	0	0	0	0	2021

FIGURE 4.3: Confusion Matrix

4.6. Performance Results of CNN model

From the results, it is observed that the proposed CNN model performs with an overall accuracy of 97.52% in identifying handwritten digits and mathematical symbols. Each digit class (0-9) has recorded precision, recall, and F1-score values above 0.96.

Symbols like '+', '-', and 't'(multiply) produce very accurate results, whereas the brackets '[' and ']' are quite accurate. The class 'd'(division) has a slight decrease in the accuracy value because the characters are very similar however, its recall value is high.

In conclusion, on both Macro and Weighted Average, high scores prove that the model is robust and effective with respect to handwritten character recognition.

TABLE 4.1: Classification Performance of the CNN Model

Class	**Precision**	**Recall**	**F1-score**	**Support**
+	0.979	0.985	0.982	2210
-	0.989	0.983	0.986	2132
0	0.995	0.998	0.996	2025
1	0.961	0.919	0.940	2039
2	0.976	0.960	0.968	2055
3	0.984	0.989	0.986	2177
4	0.986	0.987	0.986	2169
5	0.993	0.988	0.991	2070
6	0.996	0.988	0.992	2061
7	0.977	0.944	0.960	1976
8	0.997	0.996	0.996	2065
9	0.989	0.982	0.986	2046
[	0.975	0.958	0.967	2018
]	0.951	0.949	0.950	1911
d	0.882	0.978	0.927	2417
t	0.998	0.994	0.996	2033

Table 4.2: Overall Performance Metrics

Metric	**Value**
Accuracy	97.52%
Macro Average Precision	0.9767
Macro Average Recall	0.9748
Macro Average F1-score	0.9755
Weighted Average Precision	0.9759
Weighted Average Recall	0.9752
Weighted Average F1-score	0.9753
Total Samples	33,404

4.7. Discussion

The results obtained from the experiments highlight the importance of dataset quality and size in handwritten recognition tasks. The lower accuracy in the initial experiment was mainly due to the limited number of samples and higher image resolution, which increased the complexity of feature extraction.

By reducing the image size and increasing the number of training samples, the CNN was able to focus on the most important features of each symbol. This change significantly improved the model's generalization capability. The inclusion of bracket symbols further enhanced the system's ability to process complete mathematical expressions.

The high accuracy achieved in the final experiment confirms that a carefully designed CNN can effectively recognize handwritten digits and mathematical symbols without relying on complex or pre-trained architectures. In addition, integrating an arithmetic reasoning module extends the usefulness of the system for real-world applications such as educational tools and automatic evaluation systems.

Chapter5

Conclusions and Future Work

5.1. Conclusion

This research is on developing a system that would read handwritten digits and carry out arithmetic computations automatically using Convolutional Neural Networks (CNNs). The main aim was to read handwritten digits and arithmetic symbols accurately and do correct arithmetic operations based on the recognized inputs.

A custom handwritten dataset, MathSet, was collected to support this research. In the study two different configurations of the dataset were used. The first one used images of size 45 x 45 pixels with a very small number of samples which resulted in lower accuracy of the dataset. To improve the performance, a larger dataset was collected with images of size 25 x 25 pixels and with a greater number of samples per class. Bracket symbols were also included in the dataset to enable more structured arithmetic expressions.

Experimental results indicated that the improved MathSet dataset had a significant positive impact on enhancing performance of the model. The final CNN model achieved an accuracy of nearly 98%. This demonstrated the efficacy of CNNs for handwritten digit and symbol recognition when adequately and well-trained. The study indeed fulfilled its purpose overall.

5.2. Recommendations

Based on the findings of the study, there are some key recommendations to the following effects. Large and varied datasets should be obtained for high recognition accuracy. Data collection needs to consider the different variations in handwriting styles.

The effective image preprocessing techniques should include normalization and noise removal because they significantly affect the model performance.Last, frequent evaluation and comparison of different dataset configurations can help identify the most effective setup for training deep learning models.

The outcomes of this study would guide future analyzers interested in working on handwritten recognition and other related applications.

5.3. Future Development

There are several ways in which the proposed system can be extended to improve its functionality and usability. One very important upgrade in the future would be the text-based handwritten recognition, where the system will be able to recognize handwritten words like “three”, “two”, and arithmetic operators written in text form such as “plus”, “minus”, “multiply”, and “divide”. For example, when the user writes, “three plus two”, the system will recognize the text, translate it into numbers, and give the accurate answer.

Another possible enhancement can be to support multi-digit numbers and longer arithmetic expressions with multiple operators and brackets. It would make more complex calculations to be dealt with by the system similar to what is written in real-classroom environments.

Further accuracy of the system can be achieved with the collection of more handwritten data from a wider user sample. Also, the model can be embedded into a web or mobile app for real-time use by students and teachers. These modifications will make the system more practical, suitable for educational purposes, and learning-related applications.

Appendix

1.CNN Model Training Code

The code given below demonstrates the actual training of the CNN model for handwritten digit and arithmetic symbol recognition. The model, in fact, was trained on Google Colab where all the computational resources and libraries required for deep learning experimentation were available.

```
from google.colab import drive
drive.mount('/content/drive')

import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np
import os

from zipfile import ZipFile

with ZipFile('/content/drive/MyDrive/res/dataset.zip', 'r') as zipObj:
   zipObj.extractall('content')

DATASET_PATH = "/content/content/dataset"

IMG_SIZE = 28
BATCH_SIZE = 128
EPOCHS = 30
NUM_CLASSES = 16

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    os.path.join(DATASET_PATH, "train"),
    image_size=(IMG_SIZE, IMG_SIZE),
    color_mode="grayscale",
    batch_size=BATCH_SIZE,
    label_mode="int"
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    os.path.join(DATASET_PATH, "val"),
    image_size=(IMG_SIZE, IMG_SIZE),
    color_mode="grayscale",
    batch_size=BATCH_SIZE,
    label_mode="int"
)
```

```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    os.path.join(DATASET_PATH, "test"),
    image_size=(IMG_SIZE, IMG_SIZE),
    color_mode="grayscale",
    batch_size=BATCH_SIZE,
    label_mode="int"
)

normalization_layer = layers.Rescaling(1./255)

train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds = test_ds.map(lambda x, y: (normalization_layer(x), y))

model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(),

    layers.Conv2D(128, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(NUM_CLASSES, activation='softmax')
])
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model.summary()

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

checkpoint = ModelCheckpoint(
    filepath="/content/drive/MyDrive/handwritten_model.h5",
    monitor="val_accuracy",
    save_best_only=True,
    save_weights_only=False,
    verbose=1
)
```

```
early_stop = EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True
)

checkpoint = ModelCheckpoint(
    "/content/drive/MyDrive/handwritten_model.h5",
    monitor="val_accuracy",
    save_best_only=True,
    verbose=1
)

early_stop = EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True
)
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=[checkpoint, early_stop]
)
test_loss, test_acc = model.evaluate(test_ds)
print("Test Accuracy:", test_acc)

plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Val')
plt.title("Accuracy")
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Val')
plt.title("Loss")
plt.legend()

plt.show()

model = tf.keras.models.load_model(
    "/content/drive/MyDrive/handwritten_model.h5"
)
raw_train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    os.path.join(DATASET_PATH, "train"),
    image_size=(IMG_SIZE, IMG_SIZE),
    color_mode="grayscale",
    batch_size=BATCH_SIZE,
    label_mode="int"
)
class_names = raw_train_ds.class_names
print(class_names)

import cv2
CLASS_NAMES = raw_train_ds.class_names
print(CLASS_NAMES)

def predict_image(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (28,28))
    img = img / 255.0
    img = img.reshape(1,28,28,1)

    pred = model.predict(img)
    class_index = np.argmax(pred)
    confidence = np.max(pred)

    print("Prediction:", CLASS_NAMES[class_index])
    print("Confidence:", confidence)

    plt.imshow(img.reshape(28,28), cmap='gray')
    plt.axis('off')

predict_image("/content/drive/MyDrive/res/17.png")
from google.colab import files
files.download("/content/handwritten_model.h5")
```

2. Model Prediction Code

```
import numpy as np
import tensorflow as tf
from flask import Flask, render_template, request, jsonify
import base64
from io import BytesIO
from PIL import Image
import cv2
import re

app = Flask(__name__)

# Load trained model
model = tf.keras.models.load_model("handwritten_model.h5")
```

```
CLASS_NAMES = ['+', '-', '0', '1', '2', '3', '4', '5',
               '6', '7', '8', '9', '[', ']', 'd', 't']

def recognize_symbols(image_base64):
    image_bytes = base64.b64decode(image_base64)
    image = Image.open(BytesIO(image_bytes)).convert("L")
    img = np.array(image)

    # Threshold
    _, img = cv2.threshold(img, 170, 255, cv2.THRESH_BINARY)

    # Find contours
    contours, _ = cv2.findContours(
        255 - img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    )

    if not contours:
        return "", 0.0

    # Sort left \    right
    contours = sorted(contours, key=lambda c: cv2.boundingRect(c)[0])

    symbols = []
    confidences = []

    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)

        # Ignore noise
        if w < 10 or h < 10:
            continue

        margin = 4
        cropped = img[
            max(0, y - margin):y + h + margin,
            max(0, x - margin):x + w + margin
        ]

        resized = cv2.resize(cropped, (22, 22), interpolation=cv2.INTER_AREA)

        canvas = np.ones((28, 28), dtype=np.uint8) * 255
        offset = 3
        canvas[offset:offset + 22, offset:offset + 22] = resized

        canvas = canvas.astype("float32") / 255.0
        canvas = canvas.reshape(1, 28, 28, 1)
        pred = model.predict(canvas, verbose=0)
        idx = np.argmax(pred)

        symbols.append(CLASS_NAMES[idx])
        confidences.append(float(np.max(pred)))

    if not symbols:
        return "", 0.0

    return "".join(symbols), sum(confidences) / len(confidences)

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/predict", methods=["POST"])
def predict():
    data = request.get_json()
    prediction, confidence = recognize_symbols(data["image"])
    return jsonify({
        "prediction": prediction,
        "confidence": round(confidence, 3)
    })

@app.route("/calculate", methods=["POST"])
def calculate():
    data = request.get_json()
    expression, _ = recognize_symbols(data["image"])

    # Convert symbols to math operators
    expression = expression.replace("d", "/").replace("t", "*")

    # Safety check
    if not re.fullmatch(r"[0-9+\-*/().]+", expression):
        return jsonify({"error": "Invalid expression"})

    try:
        result = eval(expression)
        return jsonify({
            "expression": expression,
            "result": result
        })
    except:
        return jsonify({"error": "Calculation failed"})

if __name__ == "__main__":
    app.run(debug=True)
```

References

[1] Savita Ahlawat, Amit Choudhary, Anand Nayyar, Saurabh Singh, and Byungun Yoon. Improved handwritten digit recognition using convolutional neural networks (cnn). *Sensors*, 20(12):3344, 2020.

[2] Haider A Alwzwazy, Hayder M Albehadili, Younes S Alwan, and Naz E Islam. Handwritten digit recognition using convolutional neural networks. *International Journal of Innovative Research in Computer and Communication Engineering*, 4(2):1101–1106, 2016.

[3] Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. A global learning approach for complex online handwritten mathematical expression recognition. *Pattern Recognition Letters*, 35:224–232, 2014.

[4] Kawther Khazri Ayeb, Yosra Meguebli, and Afef Kacem Echi. Deep learning architecture for off-line recognition of handwritten math symbols. In *Mediterranean Conference on Pattern Recognition and Artificial Intelligence*, pages 200–214. Springer, 2020.

[5] Alejandro Baldominos, Yago Saez, and Pedro Isasi. A survey of handwritten character recognition with mnist and emnist. *Applied sciences*, 9(15):3169, 2019.

[6] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[7] Hai Dai Nguyen, Anh Duc Le, and Masaki Nakagawa. Deep neural networks for recognizing online handwritten mathematical symbols. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 121–125. IEEE, 2015.

[8] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M Rush. Image-to-markup generation with coarse-to-fine attention. In *International Conference on Machine Learning*, pages 980–989. PMLR, 2017.

[9] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[10] Md Anwar Hossain and Md Mohon Ali. Recognition of handwritten digit using convolutional neural network (cnn). *Global Journal of Computer Science and Technology*, 19(2):27–33, 2019.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

[13] Azadeh Nazemi, Niloofar Tavakolian, Donal Fitzpatrick, Ching Y Suen, et al. Offline handwritten mathematical symbol recognition utilising deep learning. *arXiv preprint arXiv:1910.07395*, 2019.

[14] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[15] Patrice Y Simard, David Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, volume 1, pages 958–963. IEEE, 2003.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[17] Satoshi Suzuki and Keiichi Be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[18] Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(4):331–357, 2012.

[19] Jianshu Zhang, Jun Du, and Lirong Dai. Multi-scale attention with dense encoder for handwritten mathematical expression recognition. In *2018 24th international conference on pattern recognition (ICPR)*, pages 2245–2250. IEEE, 2018.

[20] Jianshu Zhang, Jun Du, and Lirong Dai. Track, attend, and parse (tap): An end-to-end framework for online handwritten mathematical expression recognition. *IEEE Transactions on Multimedia*, 21(1):221–233, 2018.