

**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**  

---

**SINGAPORE**

**OFFLINE HANDWRITTEN MATHEMATICAL EXPRESSION  
RECOGNITION USING DEEP LEARNING TECHNIQUES**

**By:**

**Ng Zheng Xun**

**Supervisor: Dr. Loke Yuan Ren**

**Nanyang Technological University (NTU)**

**School of Computer Science and Engineering (SCSE)**

**2021**

## Abstract

---

The domain of offline Handwritten Mathematical Expression Recognition (HMER) poses unique challenges for Deep Learning (DL) models to overcome, such as the variation of handwriting styles, and the two-dimensional structure of mathematical expressions. In this research project, three models were trialed and evaluated on the CROHME 2014, 2016 and 2019 datasets - a custom Keras CNN model, a Tesseract 4 OCR Engine developed by Google, and the SESHAT Parser.

A custom preprocessing pipeline was written, to preprocess the necessary data, first from an InkML format, to image form. Following that, several preprocessing steps were conducted, to dilate and blur the image for easier feature extraction and recognition by the models.

A custom pipeline was also developed to tokenize the LaTeX groundtruth provided, for processing by the models. A Katex parser developed by Havard was used to tokenize the LaTeX groundtruth into distinct subwords. The subwords were then compiled into an exhaustive list of vocabulary, which could be mapped and detokenize the output from the models.

Out of the three models, the SESHAT Parser proved to be the most accurate and effective, despite non-trivial computational costs during the evaluation stage. The Keras CNN proved to be the weakest model, with high loss values and low accuracy. The Tesseract model gives better results than the CNN, but is inferior to SESHAT Parser.

The results are to be expected: As the CNN convolutional technique applies to recognition of symbols in the spatial sense, it is unable to relate the symbols to their neighbouring expressions, which does not allow it to learn such relationships. On the other hand, as the SESHAT Parser constructs a grammar tree from the data, it is able to relate and learn the different relationships between the symbols, not only spatially, but also mathematically, which would naturally lead to better results. The Tesseract model, as it is an LSTM-based model, is understandably able to perform better than the CNN due to it being able to store previous states and retain some form of “context”.

### **Acknowledgements**

---

I would like to extend my deepest thanks, appreciation and gratitude to my supervisor and mentor Dr. Loke Yuan Ren, from NTU's School of Computer Science and Engineering (SCSE), for his invaluable advice, guidance and mentorship. Furthermore, I would like to thank the C.N. Yang Scholars Programme (CNYSP) Office, for the opportunity to undertake this Research Project, as well as their support for the project.

## Table of Contents

---

<b>Abstract</b>	<b>2</b>
<b>Acknowledgements</b>	<b>3</b>
<b>Table of Contents</b>	<b>4</b>
<b>List of Figures</b>	<b>6</b>
<b>List of Tables</b>	<b>9</b>
<b>Introduction</b>	<b>10</b>
Motivation for Research	10
<b>Literature Review</b>	<b>13</b>
History and Development	13
Challenges of HMER Processing	14
Online and Offline HMER	16
Online HMER	16
Offline HMER	17
Deep Learning Methods for Handwritten Mathematical Expression Recognition	17
Encoder-Decoder Structure	17
State-of-the-Art (SOTA) Techniques	18
Bidirectional Long Short-Term Memory (BLSTM) with Connectionist Temporal	
Classification (CTC)	18
Paired Adversarial Learning (PAL), v2	19
<b>Research Methodology</b>	<b>20</b>
Image Preprocessing	21
Smoothing Operations	22
Image Dilation and Erosion Operations	22
Image Segmentation	23
Tokenization of Textual Data	24
Tokenization Methods	24
LaTeX Tokenization	25
One Hot Encoding	26
<b>Methods and Model Setup</b>	<b>27</b>
CROHME Dataset	27
CROHME Dataset Over the Years	28

CROHME Dataset Cleaning	29
LaTeX Groundtruth Processing and Tokenization	29
Data Augmentation Trials	30
Data Processing Workflow	31
Keras CNN Model	32
Tesseract 4 OCR Engine	33
SESHAT OCR Engine	34
<b>Results and Discussion</b>	<b>36</b>
Word Error Rate (WER)	36
Overall Results on 3 CROHME datasets	36
Discussion on Keras Model	38
Discussion of Tesseract	39
Discussion on SESHAT Engine	39
Discussion on SESHAT's Incorrect Classification	40
Discussion on Computational Times	40
Discussion on SESHAT Parser's Computational Times	41
Comparison with SOTA Models	43
<b>References</b>	<b>45</b>

## List of Figures

---

1-1	The HMER user interface of The Wolfram System, which recognises mathematical expressions that users might enter through an input device	8
2-1	An overview of the field of HMER, in the recent decade, DL techniques paved the way for end-to-end solutions to HMER	12
2-2	An example of different handwritten mathematical expressions, and the ambiguities that arise from handwriting recognition tasks, with a common LATEX groundtruth	13
2-3	An illustration of a mathematical number with differing scales of digits relative to each other	13
2-4	A comparison of online (left) and offline (right) input data	14
2-5	An illustration of an Encoder-Decoder Structure with Attention, as part of the Seq2Seq model architecture	16
2-6	An illustration of a Bidirectional RNN (right), relative to a typical RNN (left)	17
3-1	A schematic representation of the data pipeline for Offline HMER	18
3-2	Several examples of common preprocessing techniques on an image of a handwritten input	20
3-3	A 3x3 normalized box filter, for the image smoothening operations during preprocessing	20
3-4	An application of the 3x3 blur filter over a mathematical expression (left, unfiltered)	20
3-5	An application of erosion operations on a mathematical expression (left, unfiltered)	21

3-6	An example of image segmentation applied to a mathematical expression. As illustrated, there may be erroneous segmentation	22
3-7	An application of the BPE Subwords Tokenization process, with the word “whereby” being tokenized into 2 common subwords instead of character by character	23
3-8	An illustration of one-hot encoding being applied to text input	24
4-1	Several samples extracted from the various CROHME Competition Datasets	27
4-2	An illustration of the tokenization process of the LaTeX groundtruth. From top to bottom: Subword generation by Katex parser, Token generation, Post Padding	28
4-3	Attempts at Data Augmentation with rotation. It can be noted by visual inspection that it can only work within a small range of rotations, and within these data, there exist minimal differences too	29
4-4	Data Processing Workflow	30
4-5	A diagrammatic overview of the Keras CNN Architecture	32
4-6	A diagrammatic overview of Tesseract’s Inner Workings	32
5-1	The formula for calculating WER. S, D, I refers to the number of substitutions, deletions, insertions necessary to convert the source expression to the target. N refers to the length of the source sentence	35
5-2	Evaluation of Sample Data on Tesseract, Keras CNN, and SESHAT Parser Model	35
5-3	Accuracy and Loss Plots of the Keras CNN Model, on CROHME 2014	36
5-4	An examination of specific samples which SESHAT Parser incorrectly identified	38

5-5	A boxplot of the Computational Times of the SESHAT Parser on CROHME 2019	41
5-6	Histogram Plot of SESHAT Parser Computational Times on CROHME 2019	41



## List of Tables

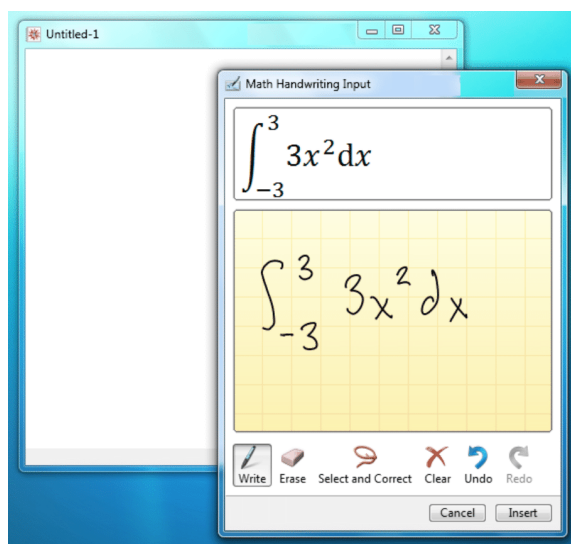
---

4-1	A comparison of the different attributes of the CROHME datasets from 2014 to 2019	26
4-2	A list of hyperparameters after hyperparameter tuning for Keras CNN Model	31
5-1	The WER(%) of the 3 Models on various CROHME datasets	34
5-2	Accuracy Comparison with SOTA Models (3sf)	40

## Introduction

### Motivation for Research

The task of Handwritten Mathematical Expression Recognition (HMER) has been noted to be one of the more challenging domains of Natural Language Processing (NLP), due to the large variation of handwriting styles and sizes, the complex 2-dimensional structure of several mathematical expressions [1], as well as the large number of symbol classes [2, p. 20]. In recognition of its difficulty and importance, it has been listed as a Request for Research by OpenAI named Image2Latex. [3] HMER technologies have been implemented in several industrial and academic applications, such as the algorithm implemented by Wolfram Alpha, which is shown in Figure 1-1.



*Figure 1-1: The HMER user interface of The Wolfram System, which recognises mathematical expressions that users might enter through an input device [4]*

As illustrated in Figure 1-1, the HMER algorithm must recognise and preserve details of the mathematical expression, such as the structure, syntax, and other smaller details such as subscripts or superscripts, among other complications in the expression.

However, despite these challenges, HMER is still a valuable and active field of research, due to its wide range of utilities, from the digitisation of images and documents, to the development of

online tools to aid in calculation without human input. [5] Furthermore, HMER algorithms could also be implemented as a visual aid for the visually impaired. [6]

As Deep Learning (DL) techniques rapidly develop and become more ubiquitous, Neural Networks (NNs) can be trained to recognise handwritten digits, and by extension, mathematical expressions as well. [7] As such, rapid progress has been made in the domain of HMER due to the application of DL techniques.

### **Problem Statement**

The domain of offline Handwritten Mathematical Expression Recognition (HMER) poses unique challenges for Deep Learning (DL) models to overcome, such as the variation of handwriting styles, and the two-dimensional structure of mathematical expressions. In this research project, three models were trialed and evaluated on the CROHME 2014, 2016 and 2019 datasets - a custom Keras CNN model, a Tesseract 4 OCR Engine developed by Google, and SESHAT Parser.

### **Objectives**

This project aims to study different Machine Learning models for the task of Offline HMER by utilising deep learning techniques, with the CROHME 2014, 2016 and 2019 datasets as the benchmark, as well as exploring and implementing methods to handle and preprocess HMER data, and their corresponding groundtruths.

### **Scope of Work**

As part of the research project, several Machine Learning models were run to evaluate their performance on several benchmark HMER datasets. Several methods have also been attempted to preprocess and tokenize the images and LaTeX groundtruth.

### **Organisation of Report**

The report is organised as follows:

Chapter 1, which is this chapter, presents an introduction to the problem statement as well as the context of the research conducted. It includes the research motivation, objectives and scope, as well as the organisation of the report.

Chapter 2 summarises findings from the literature review for related existing literature in the field of online and offline HMER, from a historical and technical perspective, followed by a review of Deep Learning methods applied to HMER, for classification and/or feature extraction purposes.

Chapter 3 explains the research methodology used for the research project, including details of the independent and dependent parameters. Explanations for the techniques and algorithms used would also be provided.

Chapter 4 presents the details of the simulation and experimental model setups, along with the hyperparameters used.

Chapter 5 presents and discusses results obtained from the models, as well as a comparison with other State-of-the-Art (SOTA) models.

Chapter 6 concludes the report, and provides discussion for future work in the field of HMER using Deep Learning techniques. Other methods for the processing of handwritten mathematical equations would also be discussed.

## Literature Review

---

### History and Development

Writing, in its natural form of handwritten notes and expressions, is a ubiquitous, convenient and intuitive way of recording and storing information in the world. As such, the field of handwritten recognition is a robust and well-developed one, with much research and progress being chronicled. [8]

Furthermore, in the field of mathematics and its related fields such as engineering and science, handwritten mathematical equations and expressions are also a fundamental part of their disciplines, and are much preferred to their digital equivalent. [9]

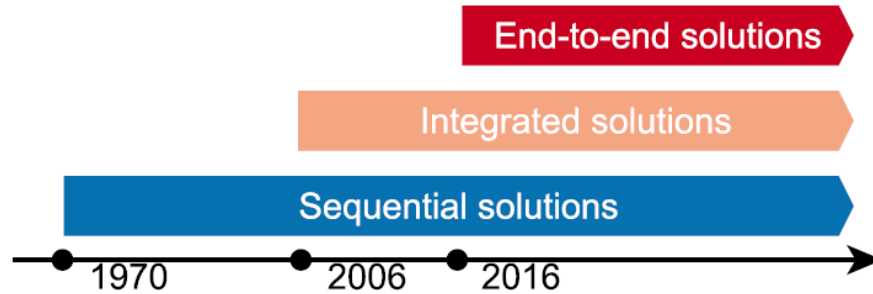
There are several conventional ways to convert handwritten mathematical equations to its digital equivalent, prior to the advent of Deep Learning techniques. They often require a huge amount of manual input from the user, and are relatively time-consuming, cumbersome and inefficient. This is especially the case when more complex equations are involved - with matrices, symbols, and subscripts/superscripts.

The traditional methods of HMER Processing date back to the late 1960s, whereby research into the study of mathematical structure for machine analysis involved Grammar-based approaches to handle symbol segmentation and recognition. [10] These efforts require a large amount of manual work to design grammars. These works were expanded upon during the 1970s, whereby HMER efficiency was improved, by building structure trees and using operator precedence and operator dominance. [11]

With the rise in popularity and efficiency of Deep Learning (DL) techniques, LeCun et al. implemented the first DL-based approach, utilising a Convolutional Neural Network (CNN), to HMER termed “LesNet”, and was awarded the Turing Award for his efforts. [12]

In recent years, there continues to be much progress utilising DL techniques, as researchers found that structures and models from the field of NLP and Neural Machine Translation (NMT)

could be applied to HMER, with great success. As such, several techniques initially conceptualized for use in NMT such as the Transformer architecture [13], Sequence-to-Seq (Seq2Seq) models [14], and the Attention mechanism, have been progressively implemented within HMER, and are widely considered close to State-of-the-Art (SOTA). These DL based implementations paved the way for end-to-end solutions to HMER.



*Figure 2-1 An overview of the field of HMER, in the recent decade, DL techniques paved the way for end-to-end solutions to HMER [7]*

### **Challenges of HMER Processing**

As mentioned in the previous chapter, the field of HMER Processing faces a multitude of challenges, despite its similarity with the field of Handwriting Digit Recognition (HDR), which has achieved great progress and accuracy in the recent decade.

It has been noted that the performance of HMER systems suffer greatly due to a combination of several factors. Firstly, there is significant difficulty in parsing the two-dimensional structure of mathematical expressions, which are common in matrices, or fractional representations. As such, the conventional method of parsing an expression sequentially from Left to Right, a common preprocessing step for HDR, would not be sufficient for the domain of HMER. [15, p.]

Furthermore, recognising handwritten mathematical expressions has the additional challenge of recognizing handwriting, a challenge which may confuse even individuals due to several characters being strikingly similar. As different individuals have different handwriting styles and characteristics, there are enormous ambiguities that result from any handwriting recognition task. [7]. Fig 2-1 illustrates several possible handwriting styles for a single groundtruth expression,

with particular emphasis placed on the character “a”, which could itself be written in several ways.

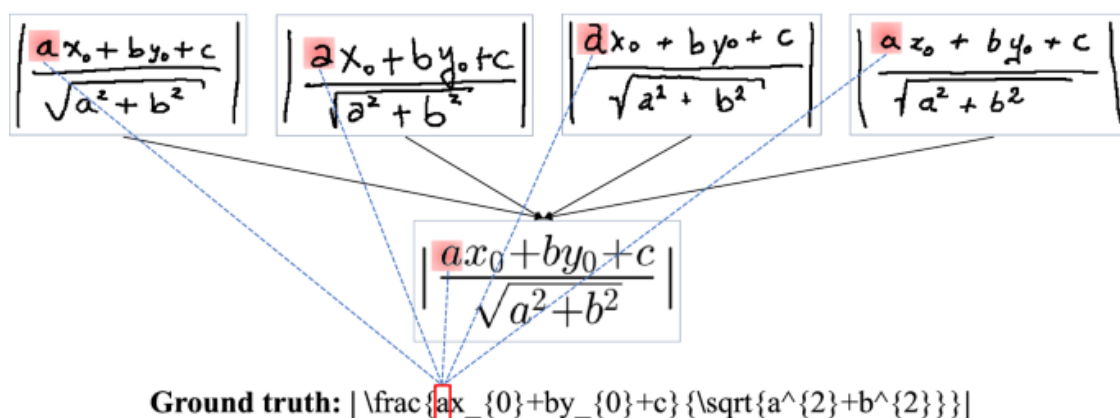


Figure 2-2 An example of different handwritten mathematical expressions, and the ambiguities that arise from handwriting recognition tasks, with a common LATEX groundtruth

Another challenge that HMER systems face is the variant scales of handwritten math symbols. This is due to the inaccuracies and inevitable differences in scale of human handwriting, as the digits and characters, when written by hand, often differ in scale. An example is given in Figure 2-2, whereby a simple single line of digits could pose a challenge for HMER systems, as the numbers are not in scale with each other. [15, p.]

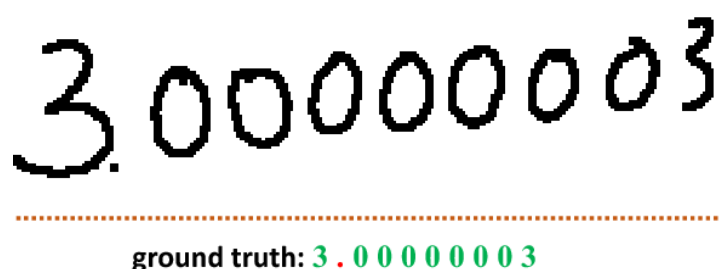


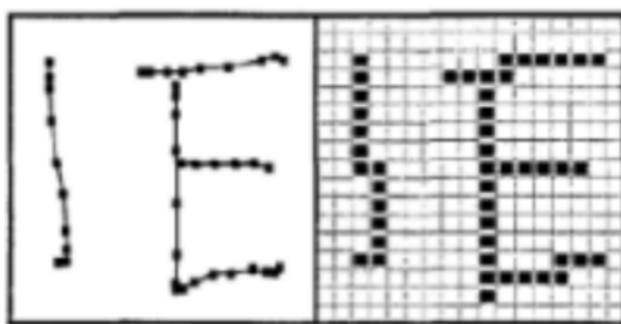
Figure 2-3: An illustration of a mathematical number with differing scales of digits relative to each other

A challenge that is rather ubiquitous in most Machine Learning (ML) applications, for which HMER processing is no exception, is the challenge of gathering, compiling and labelling data for processing. Recently, the CROHME (Competition on Recognition of On-line Handwritten

Mathematical Expressions) dataset alleviated some of the challenge, providing a standardised and reliable benchmark against which to test SOTA HMER models. [16]

### **Online and Offline HMER**

The field of HMER processing could be categorised into 2 subdomains, namely Online and Offline HMER. They differ in their input file formats, with Online HMER taking in information about the strokes that constitute the final mathematical expression, whereas Offline HMER has its file format being a usual image with the mathematical expression to be evaluated. Figure 2-3 illustrates the different forms of online and offline data.



*Figure 2-4: A comparison of online (left) and offline (right) input data [17]*

Correspondingly, the real life applications for online and offline HMER tools vary. For instance, online HMER tools could be used to evaluate mathematical expressions in real-time, based on the user's stylus's movement. Offline HMER tools, on the other hand, would be more suited to evaluate static images of mathematical expressions, perhaps scanned from a handwritten document. [18]

### **Online HMER**

Online HMER handles data which is generated by a digitising surface such as tablet PCs, and thus contains information such as stroke movements, velocity and pressure. [19] One common file format encoding such information is the Ink Markup Language (InkML) format, which contains pen information over time in the form of X Y value pairs, as well as device characteristics and detailed dynamic behavior. [20] As such, online HMER utilises all this information to predict the symbols and expressions written. [21]



### ***Offline HMER***

Offline HMER utilises images as the input file format for mathematical expression recognition. It has generally been noted that offline HMER is a more difficult task, having access to lesser amounts of information. As such, online HMER generally performs better, relative to its offline counterpart. [22]

### **Deep Learning Methods for Handwritten Mathematical Expression Recognition**

In the recent decade, as DL matures and develops, many industries and fields of study have eagerly implemented such techniques in their respective domains. As briefly mentioned in the previous chapter, in the domain of HMER, such DL techniques gave rise to end-to-end solutions for recognition.

#### **Encoder-Decoder Structure**

The Encoder-Decoder Structure is an architecture first developed for Sequence-to-Sequence (Seq2Seq) tasks, whereby both the input and output types are sequences, such as sentences, or symbols for example. [23] The Encoder-Decoder structure involves 2 models, with one model acting as the Encoder, to receive and process the input sequence. The Encoder would then pass the encoded sequence to the Decoder. The encoded sequence would encapsulate the information from the input sequence, which is usually termed the “Context Vector”. [24]

As the Context Vector is passed to the Decoder, it might be further processed by other algorithms to ensure the integrity of data, to prevent information loss. One such algorithm is the Attention Mechanism, which assigns a weight to each character of the input sequence, in an attempt to preserve some context, for the decoder to better process the information. [25]

The Decoder would then take in the Context Vector as its input, process it, and output the sequence in its intended form. [26] This overall process is illustrated in Figure 2-5.

Typically for Seq2Seq problems such as Neural Machine Translation (NMT), the Encoder and Decoder are Recurrent Neural Networks (RNNs), although recent developments have been

experimenting with different variations and combinations of networks for the encoder and decoder. [27]

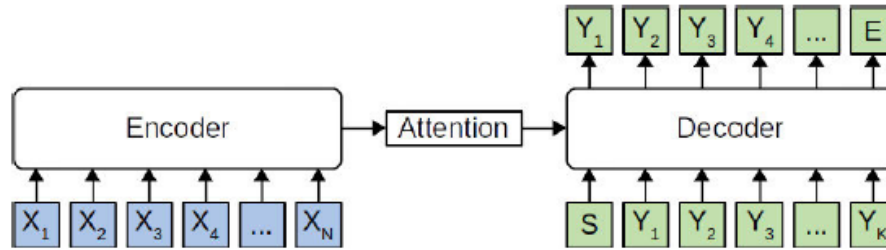


Figure 2-5: An illustration of an Encoder-Decoder Structure with Attention, as part of the Seq2Seq model architecture [21]

### State-of-the-Art (SOTA) Techniques

The State-of-the-Art (SOTA) of HMER represents the techniques and models which are currently at the forefront of research, having achieved the best results and performance on a HMER benchmark dataset. As of the time of writing, two techniques have been proven to be SOTA - Bidirectional Long Short-Term Memory (BLSTM) neural networks with Connectionist Temporal Classification (CTC) [28], as well as Paired Adversarial Learning (PAL), v2 [29]. As with any domain's SOTA, the SOTA is constantly evolving and an area of active research and development.

#### ***Bidirectional Long Short-Term Memory (BLSTM) with Connectionist Temporal Classification (CTC)***

One of the SOTAs in HMER involves Bidirectional Long Short-Term Memory (BLSTM) Neural Networks with Connectionist Temporal Classification (CTC), whereby the architecture leverages on two powerful concepts: the BLSTM and the CTC forward backward algorithm. [22]

The BLSTM are a variation of Recurrent Neural Networks (RNNs), with additional cyclic connections between its nodes in order to attempt to capture internal states over longer periods of time. [30] This allows the BLSTM to have memory-like functionality, having additional hidden layers to process inputs in the opposite direction.

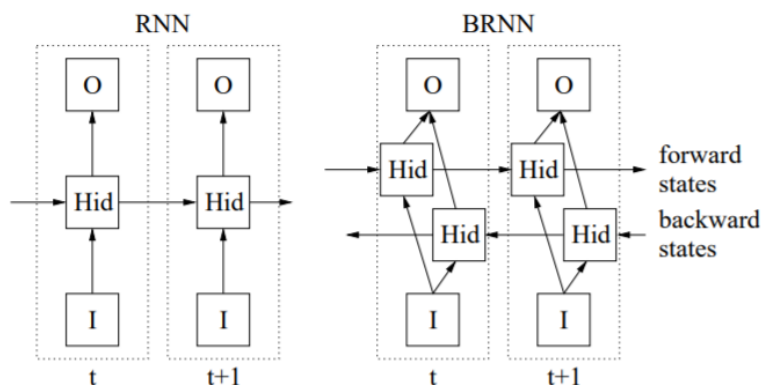


Figure 2-6: An illustration of a Bidirectional RNN (right), relative to a typical RNN (left) [31]

The current SOTA models include an architecture that pairs a BLSTM architecture with the CTC forward backward algorithm. [25] CTC removes the need for the input sequence to be pre segmented, where the labels must be aligned with the inputs.

### ***Paired Adversarial Learning (PAL), v2***

Another SOTA solution in HMER is the PAL v2 [23], which utilises a novel attention-based encoder decoder architecture, with a Paired Adversarial Learning (PAL) pipeline, where the recognizer is able to learn semantic invariant features in the feature space. The encoder is a CNN-based feature extractor followed by a RNN-based extractor, and a convolutional decoder with novel pre-aware coverage attention.

## Research Methodology

As handwritten expression recognition is a rather well developed field of study and research, ranging from handwritten digit recognition [32], to handwritten text recognition [33], there have been several methods identified which have been proven to be more suited for handwritten images preprocessing. Throughout the project processing pipeline, several techniques have been applied, starting from raw data collection, to feature extraction and analysis. The pipeline, for offline HMER, is detailed below in Figure 3-1.

**Input Data Acquisition → Image Pre-Processing → LATEX Tokenization → Neural Network Learning → Neural Network Training → Classification → De-tokenization**

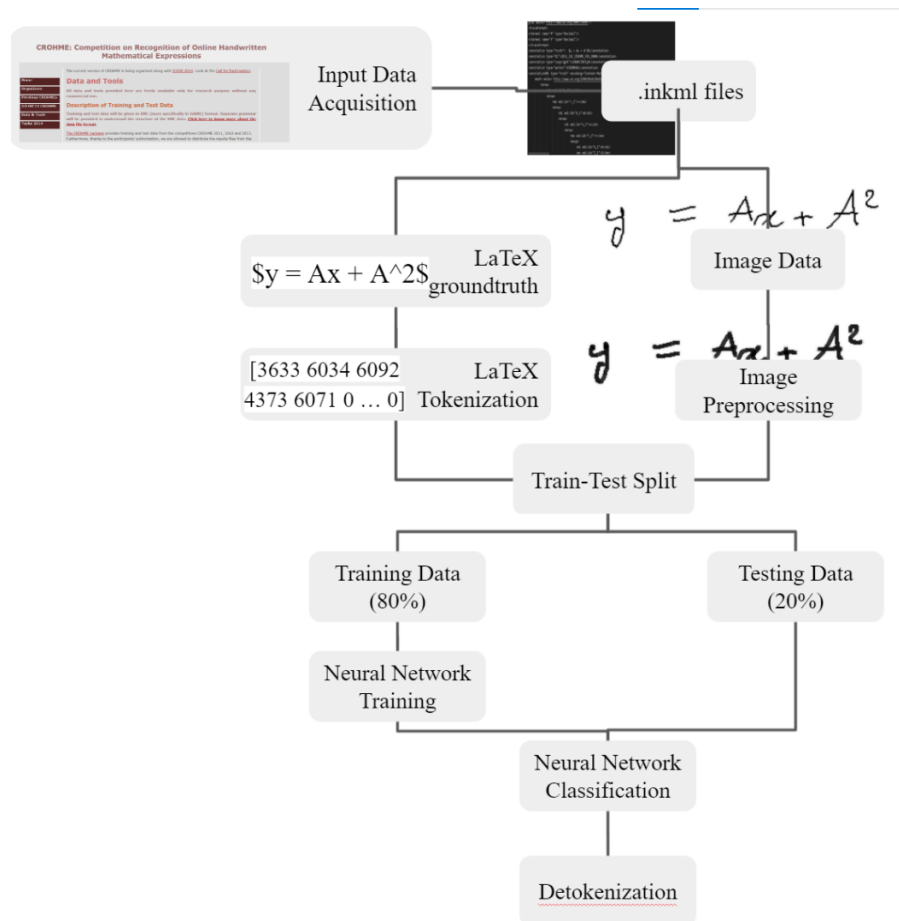


Figure 3-1: A schematic representation of the data pipeline for Offline HMER

## Image Preprocessing

Preprocessing techniques are exceptionally important in handwriting recognition and optical character recognition (OCR) systems. [34] The baseline of image preprocessing is to filter out irrelevant features of the image, and only retain and accentuate features of the image that enables the system and DL model to discriminate different classes effectively and accurately. The preprocessing pipeline also decreases redundancy in representation, as well as making the model more robust, and resistant to noise and deformation. [21]

In this research project, several techniques were applied to the image dataset, and these are detailed below. Several common techniques such as grayscaling and binarization were not applied, as their impacts were deemed to be ineffective. An application of the common preprocessing techniques is shown in Figure 3-2.

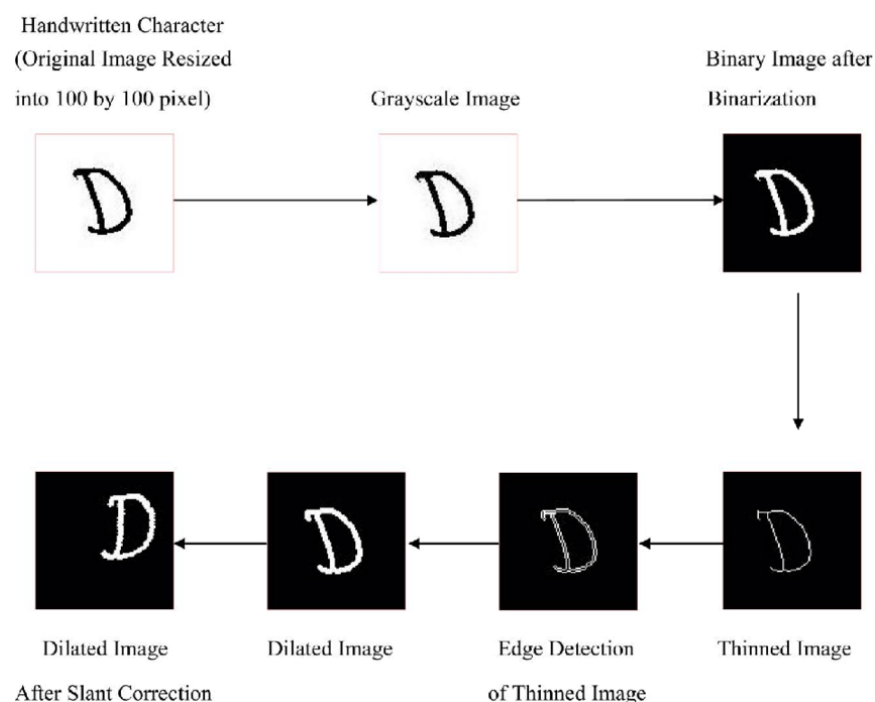


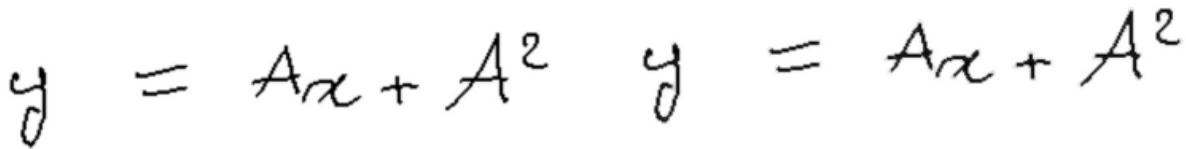
Figure 3-2: Several examples of common preprocessing techniques on an image of a handwritten input [35]

### ***Smoothing Operations***

A common technique to preprocess images is to filter images through a Low-Pass Filter (LPF), which helps in removing noise, or blurring the image. There are several ways in which it could be achieved, including Gaussian Filtering [36], and Bilateral Filtering [37]. In this implementation, an averaging operation is done over a 3 by 3 kernel, whereby the box filter would convolve the image in 3 by 3 boxes, and normalise the central element of the box by an average of its neighbours, as illustrated in Figure 3-3. [38] An application on a HMER image is shown in Figure 3-4.

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

*Figure 3-3: A 3x3 normalized box filter, for the image smoothing operations during preprocessing*



$$y = Ax + A^2 \quad y = Ax + A^2$$

*Figure 3-4: An application of the 3x3 blur filter over a mathematical expression (left, unfiltered)*

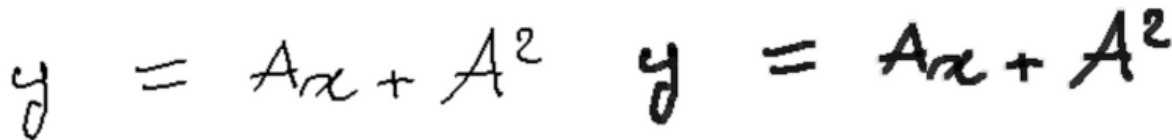
### ***Image Dilation and Erosion Operations***

Dilation and Erosion are 2 of the fundamental operations in morphological image processing, where the primary issue of focus is the shape of features of the image. [39] Morphological operations are typically applied to remove imperfections of an image, and have been proven to be effective on binary or grayscale images. [40]

The Erosion algorithm computes a local minimum over the area of a given kernel, with the effect of shrinking foreground objects and enlarging foreground holes, but in doing so would reduce

several redundant pixels as well. [41] This is extremely useful in removing “islands” in an image, so that only substantive and significant objects remain in an image.

The Dilation algorithm is the flip side of the Erosion algorithm, with it computing the maximum value of all pixels in the neighborhood. In a binary image, a pixel is set to 1 if any of the neighboring pixels have the value 1. [42] In effect, it causes bright regions within an image to “grow”, filling in small holes, hence its name. An application of these 2 algorithms, in sequence, on a HMER image is shown in Figure 3-5.

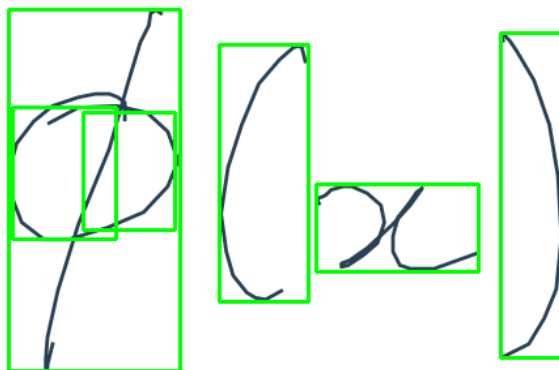


$$y = Ax + A^2 \quad y = Ax + A^2$$

*Figure 3-5: An application of erosion operations on a mathematical expression (left, unfiltered)*

### **Image Segmentation**

A common step in the HMER pipeline is Image Segmentation, where distinct symbols would be boxed and isolated from the input image, although it was not implemented in this project due to the multi-dimensional nature of mathematical expressions. Several algorithms are possible, such as the k-means algorithm [43], or the contour detection algorithm, which is utilised in several commonly-used libraries, such as cv2. [44] An example of image segmentation is shown in Figure 3-6, although it also shows the fallibility of such an algorithm, as there may be erroneous segmentation, whereby a single symbol is segmented twice.



*Figure 3-6: An example of image segmentation applied to a mathematical expression. As illustrated, there may be erroneous segmentation*

There are several ways to rectify such an erroneous segmentation, using other algorithms to “filter” out the erroneous regions. These include attempts to do so manually, by proportion of the original image, or the Non-Maximum Suppression (NMS) algorithm.

### **Tokenization of Textual Data**

As ML models usually take in and process data in terms of numbers and digits, it is necessary for information to be encoded in a manner that could be processed in such a fashion. For example, it is common practice for images to be encoded in the form of a numpy array, with dimensions corresponding to the image’s height, width and RGB pixels. However, for textual data, the problem is no longer as trivial. While individual characters of text could be and have been encoded, using ASCII or other encoding schemes for example, the meaning and context of words and sentences are harder to capture simply by encoding individual characters.

### **Tokenization Methods**

Thus, there has been significant research and study on how best to encode textual data, with several methods of tokenization, whereby each “token” could be mapped to a unique numerical identifier. [45] The tokens are generated by a parser, and the final encoded sequence could be similarly decoded by passing it through a “de-tokenizer”, which simply maps the tokens back to their corresponding text. There are several methods of tokenization, ranging from word tokenization, where a sentence or expression could be simply split into its constituent words, [46]



to more sophisticated methods such as the Byte-Pair Encoding (BPE) subwords tokenization, whereby the most common pair of consecutive characters is tokenized as one token, while the rare words are broken down into two or more subword tokens. [47] An illustration of the BPE Subwords Tokenization could be found in Figure 3-6.

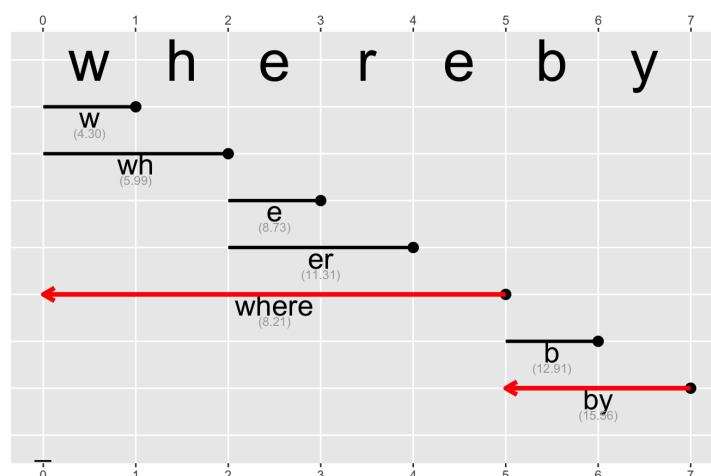


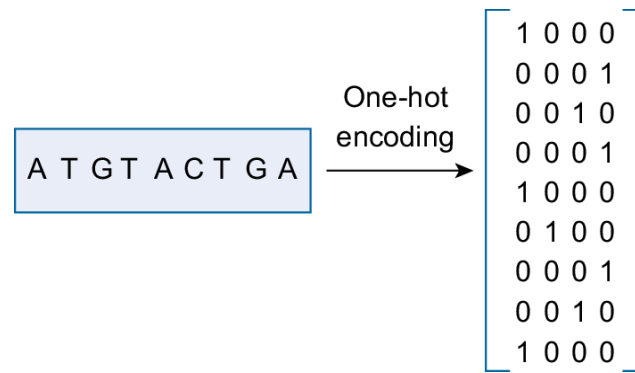
Figure 3-7: An application of the BPE Subwords Tokenization process, with the word “whereby” being tokenized into 2 common subwords instead of character by character [48]

### LaTeX Tokenization

In the context of both online and offline HMER, as the groundtruth is presented in the form of the LaTeX, it is possible to represent each symbol and number as a single token. For example, '\psi' stands for “ $\Phi$ ” in LaTeX, which would be treated as one single token, rather than four separate tokens ‘\’, ‘p’, ‘s’ and ‘i’, if a character-based tokenization algorithm was applied. Encoding each LaTeX word individually has obvious benefits, as it reduces the number of computations and the possibility of erroneous predictions. However, there exists some difficulties in doing so, as there needs to be an exhaustive list of LaTeX words and an effective parsing algorithm to segment the LaTeX sources into their corresponding words. [49] In this project, a modified version of the KaTeX parser developed by Havard was implemented. [50] More information about the Tokenization could be found in the next chapter, under LaTeX Normalisation.

### One Hot Encoding

One-hot Encoding is a commonly used encoding format used in the data preparation pipeline for DL based models. Usually used to encode textual data in NLP, it is used in this project for further processing of the tokenized LaTeX groundtruth. One-Hot Encoding enables categorical data to be stored numerically and eventually trained on ML models, by setting all bits to be '0' except one, which takes the value of '1' and thus represents that category. [51] An example of one-hot encoding is shown in Figure 3-9.



*Figure 3-8: An illustration of one-hot encoding being applied to text input*

## Methods and Model Setup

---

During the research project, several libraries were used, ranging from use of their data processing capabilities, or their machine learning capabilities. For example, the cv2 library [52] was used to implement the image preprocessing pipeline, whereas other common libraries such as numpy [53] and Keras [54] were used to clean, tokenise, and one-hot encode the LaTeX groundtruth. Jupyter Notebooks were the primary development platform for this project, when developing the custom preprocessing and tokenization pipeline, and the Windows Subsystem for Linux (WSL) architecture was utilised for the implementation and evaluation for the SESHAT Engine, due to its relative computational efficiency, whereby it consumes lesser CPU resources, and does not run on a virtual machine. [55]

The evaluation methods of K-fold Cross Validation [56] and Leave-One-Out Cross Validation [57] were considered, but ultimately not implemented due to their high computational and temporal costs. [58] [59]

Several other models were also tried and worked on, such several implementations of the SOTA Watch, Attend and Parse (WAP) model [60], [61], as well as an implementation of Multi-Scale Attention (MSA) with Dense Encoder model [62]. Regrettably, due to time or computational constraints, these models were unable to be brought to completion.

### CROHME Dataset

A benchmark dataset for LaTeX conversion to handwritten images is the Competition on Recognition of Handwritten Mathematical Expressions (CROHME) dataset, [2] which has been hailed as the de facto dataset for any handwritten recognition research. It's data has been used in research groups all over the world, and several iterations have been published over the years, from 2011-2014, 2016 and 2019. Nowadays, CROHME is the main driving force for the development of ME recognition systems both online and offline. Each time the size of datasets increases, and the number of participants remains almost the same.

Each CROHME competition has several tasks which participants and SOTA models could be benchmarked against, such as online and offline tasks. It was first published as a collation of several datasets, such as MfrDB [63], Mathbrush [64] and HAMEX [65]. Datasets are distributed as a set of Ink Markup Language (InkML) files, where each file contains information about groundtruth in LaTeX, as well as MathML formats.

As the InkML files are intended to be of the online HMER domain task, a conversion script [66] was utilised in this project to convert the InkML files into their corresponding images, and extract out the LaTeX groundtruth.

### ***CROHME Dataset Over the Years***

As CROHME develops over the years, the breadth and depth of the dataset has been expanded and built upon. For example, the number of test and training examples have been steadily increasing every subsequent CROHME competition, as well as the difficulty of the data samples themselves. The data themselves were generated by a combination of combining previous CROHME competitions, as well as manually being generated by the CROHME team. [2] Table 4-1 summarises several attributes of the CROHME competition datasets over the years, while Figure 4-2 illustrates the evolving difficulty of the CROHME dataset over the years.

<b>CROHME Dataset</b>	<b>Input Image Size</b>	<b>Dataset Size</b>
2014	(256, 256, 3)	986
2016	(310, 310, 3)	10848
2019	(200 <sup>1</sup> , 515, 3)	11970

*Table 4-1: A comparison of the different attributes of the CROHME datasets from 2014 to 2019*

It is worth noting that the relative size of CROHME datasets is rather small, especially when compared to DL datasets from other domains (the ImageNet dataset [67] for image classification

---

<sup>1</sup> \*The first dimension of the 2019 CROHME Offline dataset varies, from 57 to 389. However, for consistency, all images were resized to (200, 515, 3). The final value of 200 was arrived at after some experimentation

and object detection for example, has over 14 million images). This could be attributed to the relative difficulty in obtaining handwritten mathematical expressions, especially of a more complex structure. Despite its limitations, CROHME is still the main driving force for HMER research, and a common benchmark which researchers would use to compare their work against.

CROHME 2014	$m \geq 2$	$y \neq x$	$\frac{a-b}{\sqrt{pq}}$	$f(n-1)$
CROHME 2016	$\gamma < \beta$	$\exists \gamma, \gamma > j$	$u(t) = \frac{u(0)}{1-tu(0)}$	$\beta_0 = 1000$
CROHME 2018	$\forall x \in A, \forall y \in B, x < y$	$0 < \gamma_i < \alpha$	$x_k x_k + y_k x_k$	$F = \sqrt{F_x^2 + F_y^2}$

Figure 4-1: Several samples extracted from the various CROHME Competition Datasets

As seen in Figure 4-1, there is a gradual increase in difficulty and complexity of the dataset over the different CROHME competitions, especially in terms of the number of terms per sample.

### **CROHME Dataset Cleaning**

Upon extracting the relevant images and LaTeX groundtruths, data exploration and cleaning procedures were carried out. These involve removing any groundtruths that might be erroneous such as those that did not reflect the original equation, or were mistakenly not translated from the InkML file correctly. Other operations included replacing tabs with spaces within the groundtruth, and appending newline characters at the end of every distinct groundtruth to aid further processing.

### **LaTeX Groundtruth Processing and Tokenization**

The LaTeX groundtruth processing involved constructing an exhaustive vocabulary for the LaTeX groundtruth. However, this is not trivial, as there are different methods as outlined in the previous chapter, ranging from word-level tokenization to byte-pair encoding subwords

tokenization. In this project, the Katex parser was developed by Harvard. [50, p. 2] The parser first normalises the LaTeX groundtruths to ensure consistency of representation, by forcing adoption of some conventions. For example, writing  $x^2$  instead of  $x^2$ , etc. This is because several different LaTeX expressions give identical outputs such as  $(x^2 + 1)$  and  $\left(x^2 + 1\right)$ . The final vocabulary list generated is 9380 in length. Figure 4-3 illustrates the tokenization pipeline on a sample LaTeX groundtruth.

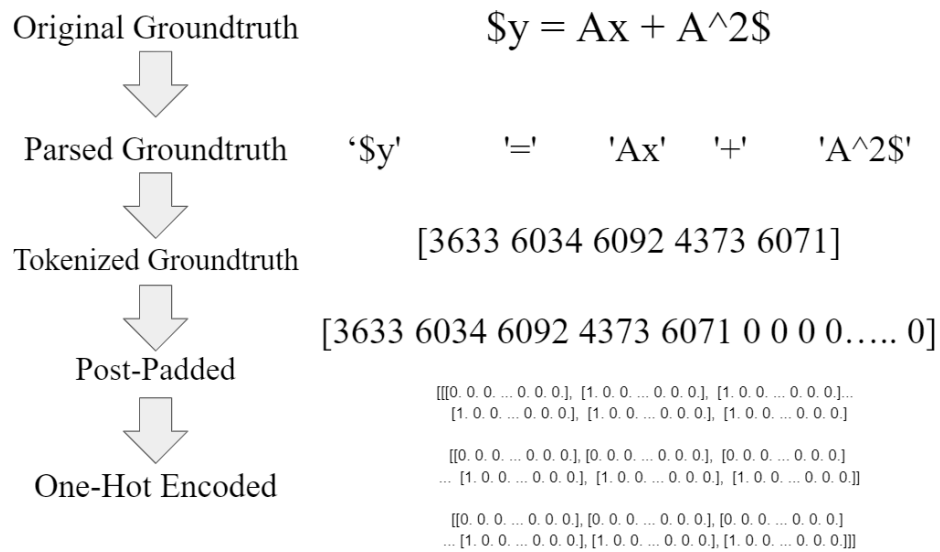


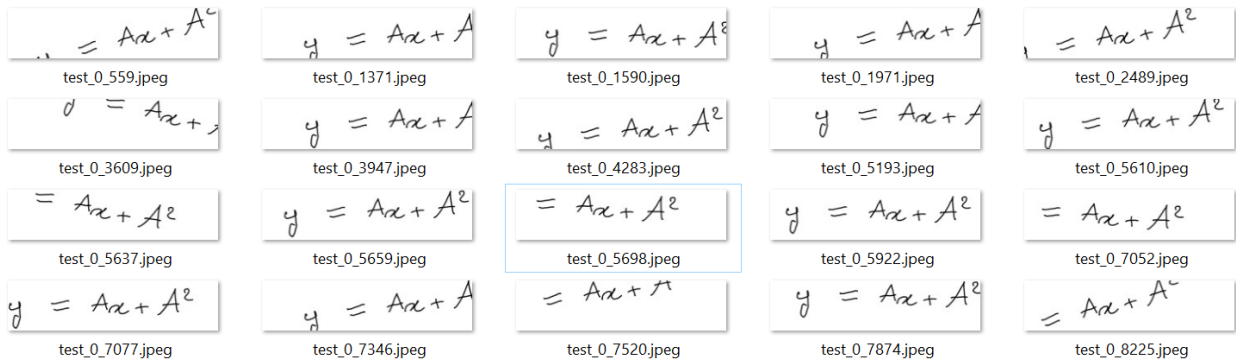
Figure 4-2: An illustration of the tokenization process of the LaTeX groundtruth. From top to bottom: Subword generation by Katex parser, Token generation, Post Padding

### Data Augmentation Trials

Data augmentation techniques [68] involve the modification of existing images of the dataset, in an attempt to artificially increase the size of the training and testing datasets. However, it has been noted that the choice of the specific data augmentation techniques used for a training dataset must be chosen carefully and within the context of the training dataset and knowledge of the problem domain. [69]

In this project, data augmentation techniques such as applying random rotations or vertical and horizontal scaling to the dataset were also tried out, as demonstrated in Figure 4-1. However, they were ultimately not implemented, as by visual inspection, there are ultimately minimal

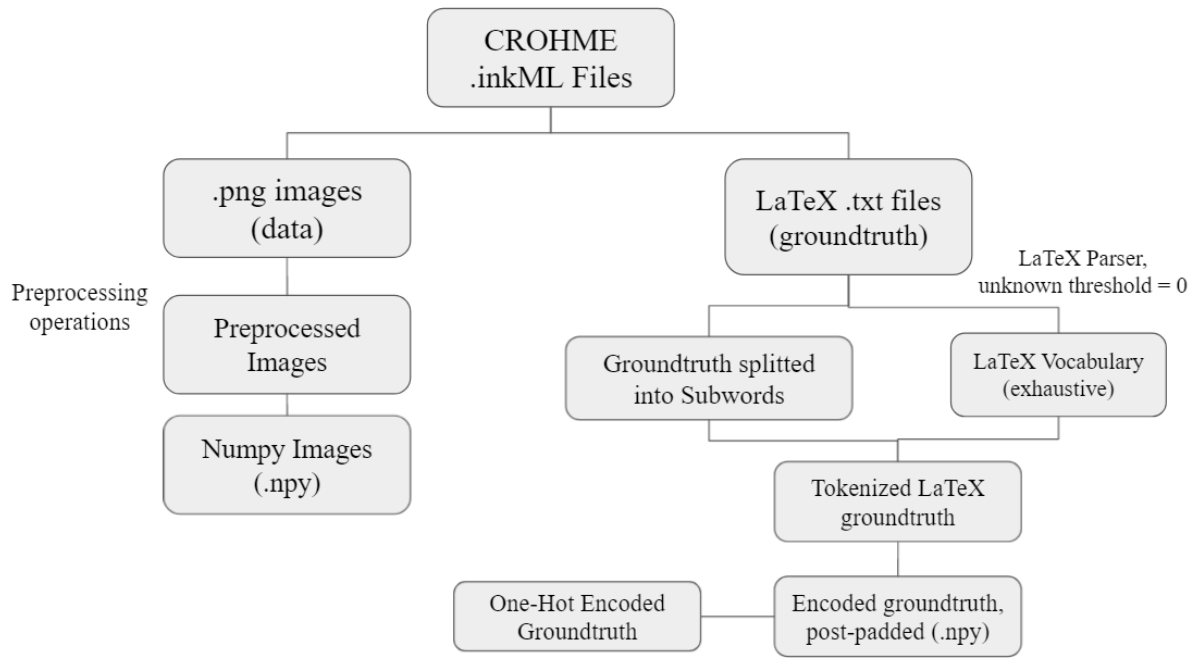
differences in the augmented images, and are mostly identical. Larger scales of augmentation (with a greater degree of rotation, scaling etc), would result in several parts of the mathematical expression being cut off, and are thus undesirable.



*Figure 4-3: Attempts at Data Augmentation with rotation. It can be noted by visual inspection that it can only work within a small range of rotations, and within these data, there exist minimal differences too*

## Data Processing Workflow

The general data processing workflow as summarised above could be illustrated in Figure 4-3, as a flowchart.



*Figure 4-4: Data Processing Workflow*

### Keras CNN Model

A Keras CNN model was set up to trial the offline HMER task. The CNN was based on Tensorflow's CNN model, and the Keras library was used for one-hot encoding of the tokenized LaTeX labels. The list of hyperparameters used are listed below. A schematic representation of the FFT CNN is shown in Figure 4-2, while a list of the relevant hyperparameters are listed in Table 4-2.

hyperparameter	chosen values
Learning Rate	0.0001
Batch Size	32
Number of Classes	53
Number of Epochs for Training	80
Input Tensor Shape	(310, 310, 3)
Loss Function	Categorical Cross Entropy
Optimiser	Adam

*Table 4-2: A list of hyperparameters after hyperparameter tuning for Keras CNN Model*

Max-pooling was implemented for the convolutional layers, and alternating activation functions of the Rectified Linear Unit (ReLU) activation function and the tanh activation function was used for the Convolutional and Dense layers. For the final classification layer, the softmax activation function was used, to output a probability estimate of each class.

There were two key considerations when designing the architecture of the Keras model, to mitigate the concerns about the model overfitting due to the relatively small sizes of the datasets. Firstly, several batch normalisation and dropout layers were inserted to prevent overfitting. Secondly, the overall depth and size of the model should be kept small, without too many dense



layers or neurons for example. This would prevent the model from extracting too many specific and unrepresentative features from the dataset, which leads to overfitting.

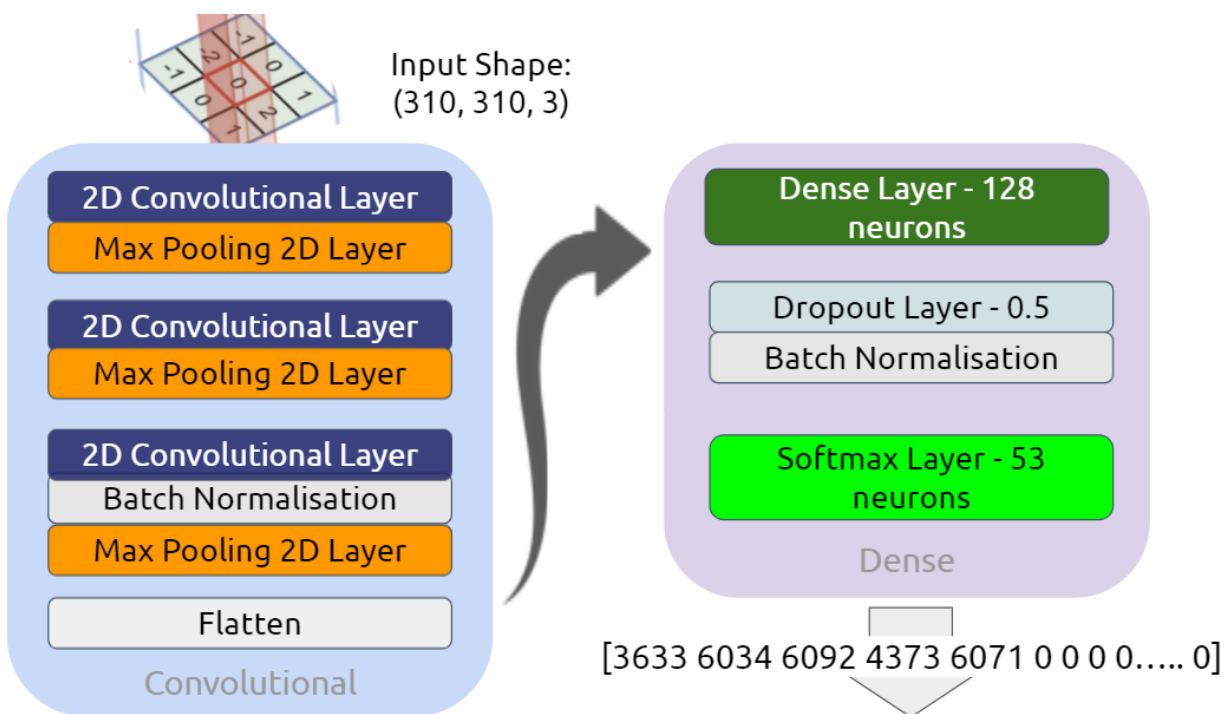
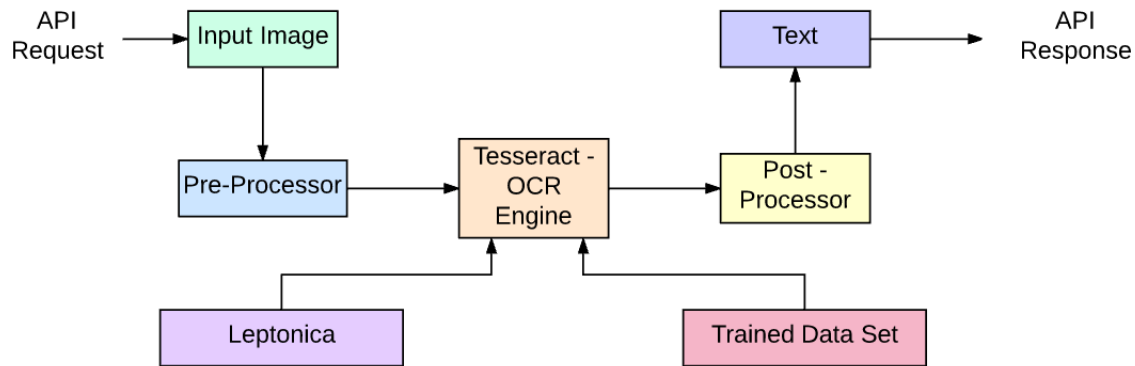


Figure 4-5: A diagrammatic overview of the Keras CNN Architecture

### Tesseract 4 OCR Engine

As a baseline model to compare subsequent experiments against, an open source Optical Character Recognition (OCR) Engine Tesseract 4 is used in this project. OCR engines are software which actually tries to recognize text in whatever image is provided. Tesseract was developed by Google to recognise images and detect spam in its applications such as Gmail [70]. It utilises a Long-Short Term Memory (LSTM) model, a variation of the RNN model mentioned in earlier chapters, and was once considered one of the most accurate open-source OCR engines available. [71] However, like most ML systems, the quality of input data is of paramount importance, with clean, binarized images with high contrast performing the best with Tesseract's OCR system. [72] A detailed overview of the workings of Tesseract could be found in Figure 4-4.



*Figure 4-6: A diagrammatic overview of Tesseract's Inner Workings [73]*

The pytesseract library [74] was utilised in this research project to implement Tesseract's OCR Engine, with an additional parameter, the Page Segmentation Mode (PSM) configuration set to 13, corresponding to Raw line: which Tesseract would treat the image as a single text line, bypassing hacks that are Tesseract-specific. This was arrived at after several trials with various psm modes. It is executed via a command line interface.

The Tesseract model was pre-trained on an artificially generated corpus of images of text obtained from the web, in various fonts. [75] Specifically, the training data for Latin-based languages such as English has been trained on ~400000 textlines, in 4500 fonts. [76] For low-resource languages where the training data would be harder to collate, the development team has relied on community-provided corpuses, although admittedly the accuracy of Tesseract on such languages may not be ideal too. [75]

### **SESHAT OCR Engine**

The Handwritten math expression parser SESHAT [77] is an open source system written in C++ for recognizing handwritten mathematical expressions. Given a sample represented as a sequence of strokes, the parser is able to convert it to LaTeX or other formats like InkML or MathML. Its architecture is a Recursive Neural Networks and probabilistic grammars to disambiguate the symbols, as well as building up a derivation tree representing the recognized math expression. The symbol recognition portion of the HMER task is performed by a combination of online and offline features with a BLSTM-RNN classifier. [78]

The process of building up an expression tree is of note, as it allows the SESHAT Engine to consider each stroke and symbol relative to its neighbours preceding and succeeding it. This would, in principle, allow the SESHAT parser to better understand mathematical grammars and the implicit syntax rules that an otherwise purely visual recogniser such as the Keras CNN would not be able to achieve.

However, one limitation of the SESHAT parser is its performance and efficiency at longer sentences and expressions, as the running time of the Parser increases substantially with expression length and complexity. As such, in the context of a real-life application which hopes to respond in real-time, the Parser may be somewhat lacking.

## Results and Discussion

---

This section of the report details the performance of the three models detailed earlier, on the CROHME 2014, 2016 and 2019 datasets. Further analysis and discussion of the models performance would be covered, as well as a sampling of several test cases against the models.

### ***Character Error Rate (CER)***

The metric of evaluation for individual samples is the Character Error Rate (CER), which takes a more holistic approach in calculating the accuracy of a model, as it considers the number of character substitutions, deletions, and insertions needed to convert the source expression to the target expression. A lower CER reflects a better prediction accuracy. The formula for CER is shown in Figure 5-1.

$$CER = \frac{S + D + I}{N} \times 100$$

*Figure 5-1: The formula for calculating CER. S, D, I are the number of substitutions, deletions, insertions to convert the source expression to the target. N refers to the length of the groundtruth sentence*

### **Overall Results on 3 CROHME datasets**

Table 5-1 details the accuracy/expression recognition rate of the three models on the CROHME 2014, 2016 and 2019 datasets.

Furthermore, it is worthy to note that the performance of the models decreases with every subsequent CROHME dataset. This result is expected, as illustrated in a previous section, the mathematical complexity and expression difficulty increases with every subsequent CROHME competition dataset.

CROHME Dataset	Character Error Rate/CER (%)		
	Keras CNN	Tesseract 4	SESHAT Parser
2014	114.5	102.3	32.5
2016	111.9	102.8	35.7
2019	111.4	103.7	42.1
mean	112.6	102.9	32.7

*Table 5-1: The CER(%) of the 3 Models on various CROHME datasets (1dp)*

To further understand the performance of these models, several samples of varying difficulties were passed to the models to obtain several results. For accurate evaluation, these individual data were separately sourced, and not contained within the train or test sets of any of the models above. A comparison of several sample test cases are illustrated in Table 5-2, where the model's CER scores are bolded for easy reference.



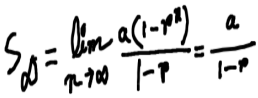
Image Samples/ Model Predictions and Results				
Groundtruth		$\varnothing(x)$	$78 \pm 5 \times 47$	$S_\infty = \lim_{n \rightarrow \infty} \frac{a(1-r^n)}{1-r} = \frac{a}{1-r}$
Groundtruth LaTeX		$\phi(x)$	$78 \pm 5 \times 47$	$S_\infty = \lim_{n \rightarrow \infty} \frac{a(1-r^n)}{1-r} = \frac{a}{1-r}$
Keras CNN	prediction (LaTeX)	$\phi(x)$	$78 \pm 5 \times 47$	$S_\infty = \lim_{n \rightarrow \infty} \frac{a(1-r^n)}{1-r} = \frac{a}{1-r}$
	CER (%)	800.0	566.0	1360.0
Tesseract 4	prediction	$P(x)$	$4g \text{ tS } xb\}$	$\phi(x) = \lim_{n \rightarrow \infty} \frac{a(1-p^n)}{1-p} = \frac{a}{1-p}$
	CER (%)	50.0	77.7	91.3
SESHAT Parser	prediction (LaTeX)	$\phi(x)$	$78 \pm 5 \times 47$	$S_\infty = \lim_{n \rightarrow \infty} \frac{a(1-p^n)}{1-p} = \frac{a}{1-p}$
	prediction	$\varnothing(x)$	$78 \pm 5 \times 47$	$S_\infty = \lim_{n \rightarrow \infty} \frac{a(1-p^n)}{1-p} = \frac{a}{1-p}$
	CER (%)	0.0	35.7	28.8

Table 5-2: Evaluation of Sample Data on Tesseract, Keras CNN, and SESHAT Parser Model.

*A lower CER corresponds to a more accurate prediction*

## Discussion on Keras Model

Through observing the training and loss curves of the Keras model, it underlines the limitations of the Keras library, and at times especially with complicated real life data, it may be necessary to adopt other more sophisticated methods and models.

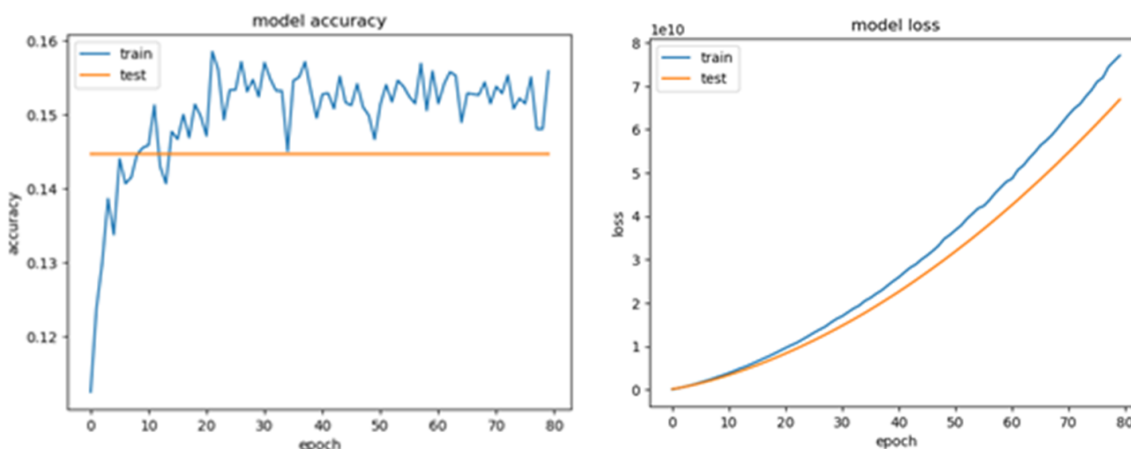


Figure 5-3: Accuracy (left) and Loss (right) Plots of the Keras CNN Model, on CROHME 2014

As observed, the model did not train well, or generalise well to the dataset, as the accuracy values are fluctuating and the model loss is increasing exponentially. In response to this, further hyperparameter tuning was carried out, such as decreasing the learning rate.

It is ultimately expected that the Keras CNN would not perform as well, as the CNN convolutional principle, while effective in allowing it to extract physical features of the image, such as the curves or strokes, is unable to account for the relative spatial and syntax of the mathematical expression. For example, the CNN may fail to distinguish between “a” and “ $\alpha$ ” which both look similar, although their application and use in a mathematical expression may differ. As such, it is expected that the performance will decrease with the introduction of more symbols.

### Discussion of Tesseract

The Tesseract 4 output The OCR is not as accurate, and the observation could be made that it does better on single-line outputs, and its performance decreases substantially on multi-dimensional or multi-line images. Furthermore, it has a better performance recognising textual data, versus mathematical symbols.

It has also been noted that Tesseract is not capable of recognizing handwriting, rather it recognises textual data, and as such it might not be suited in the domain of HMER. Furthermore, like most pre-trained ML models, it doesn't do well with images affected by artifacts including partial occlusion, distorted perspective, and complex background.

However, as its results are better than the custom Keras CNN model, it could be said that the pre-trained LSTM model, and the LSTM model architecture itself, lends itself better to the domain of HMER. The LSTM cell, as an extension to the RNN architecture, allows each cell to consider context, by passing the previous cell's hidden states to the next cell. As such, it is expected that the LSTM Tesseract 4 performance would be better than the end-to-end Keras CNN, as the Keras CNN does not consider context, and only the spatial relationship of symbols, and the symbols themselves.

### **Discussion on SESHAT Engine**

In this implementation and evaluation of SESHAT Parser, the Windows Subsystem for Linux (WSL) architecture was utilised. It could be seen from the results that the SESHAT Parser gives the best results across the board for all kinds of mathematical expressions. This is encouraging, and expected to a degree, as the SESHAT Parser uses the information about each symbol to craft a grammar tree, and recursively builds the grammar tree for each expression. This tree would allow the Parser to consider both the symbol itself, as well as its relationship both spatially as well as grammatically with the other neighbouring symbols. As such, it is expected that the SESHAT Parser would perform better than the other two models tried.

### ***Discussion on SESHAT's Incorrect Classification***

It is of note to discuss the specific samples which SESHAT Parser was unable to classify correctly. A further analysis was done on the SESHAT Parser's output and the actual groundtruth, and several specific cases are highlighted in Figure 5-4.




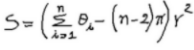
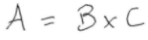
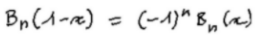
Generated Image				
Groundtruth	$\forall a \in A, \forall b \in B, a < b$	$S = \Big(\sum_{i=1}^n \theta_i - (n-2)\pi\Big)r^2$	$A = B \times C$	$B_n(1-x) = (-1)^n B_n(x)$
SESHAT Parser Prediction	$\forall ac - A, \forall bc - B, ab$	$S = (\sum_{i=1}^n \theta_i - (n-2)\pi)r^2$	$A = B \times C$	$B_{\{n\}}(1-x) = (-1)^{\{n\}} B_{\{n\}}(x)$

Figure 5-4: An examination of samples which SESHAT Parser incorrectly identified

As it could be seen, several of SESHAT's errors are very much reasonable, such as mistaking the “Bigg” notation of an enlarged bracket for a normal “(” bracket, or a multiplication symbol “ $\times$ ” for the letter “x”. Furthermore, several differences come down to a matter of syntax, such as the inclusion of  $\{ \}$  whenever needed. However, as it could also be seen from Figure 5-3, there are also some cases which the SESHAT Parser failed to identify correctly.

### Discussion on Computational Times

The below discussion of computational times are conducted on the 11th Gen Intel(R) Core(TM) i7-1165G7 CPU.

Regarding the Keras CNN model, it is more costly to set up than to evaluate on, as the main computational time is from the training, which takes around 24 hours per model to be trained at 80 epochs. Upon further hyperparameter tuning or epoch training, it is expected that the performance and accuracy of the model would improve, but at risk of overfitting and unable to generalise to test data. Upon evaluation, the trained Keras model is able to evaluate within 5 minutes. Taking the Keras results into consideration, as the results are less than ideal, it is not worth it to invest such computational times into training and evaluating the model.

Regarding Tesseract, it is the most computationally efficient to set up and evaluate on. However the evaluation time is not trivial, with an average of 1 hour per 1k sentences to be passed, amounting to 4 seconds per evaluation. Taking into consideration Tesseract's results, it may be worth it to set up and evaluate several single-line expressions. However, for multi-line or multi-dimensional mathematical expressions, the SESHAT Parser would be a more worthwhile and efficient use of computational resources.

The SESHAT Parser is more computationally expensive to evaluate than to set up, due to the nature of its evaluation through the usage of recursive neural networks to build up the grammar tree. However, given its relatively superior performance on CROHME, it is worth the computational resources for evaluation. Furthermore, the building of the expression tree is particularly time consuming, and scales as the complexity of the mathematical expression increases. SESHAT's computational efficiency is further elaborated in the following section.

### ***Discussion on SESHAT Parser's Computational Times***

There exists large variation in the computational times of the SESHAT Parser, with the upper limit being an expression which took 18 hours and 12 minutes to parse, and the lower limit being an expression that takes merely 224 milliseconds. The average expression, with significantly less complexity and strokes, takes around 5 seconds to parse.

Taking a sample of 4207 samples which took 3 days to parse, a box plot and histogram of the computational times is shown in Figure 5-5 and 5-6 respectively. As seen in the plots, the vast majority of data points and samples could be parsed by SESHAT effectively and quickly, with a few, but significant outliers, which require tens of hours to process. However, it is fair to conclude that the tradeoff of a longer computational time is worthwhile for the much superior results of SESHAT on the HMER task.

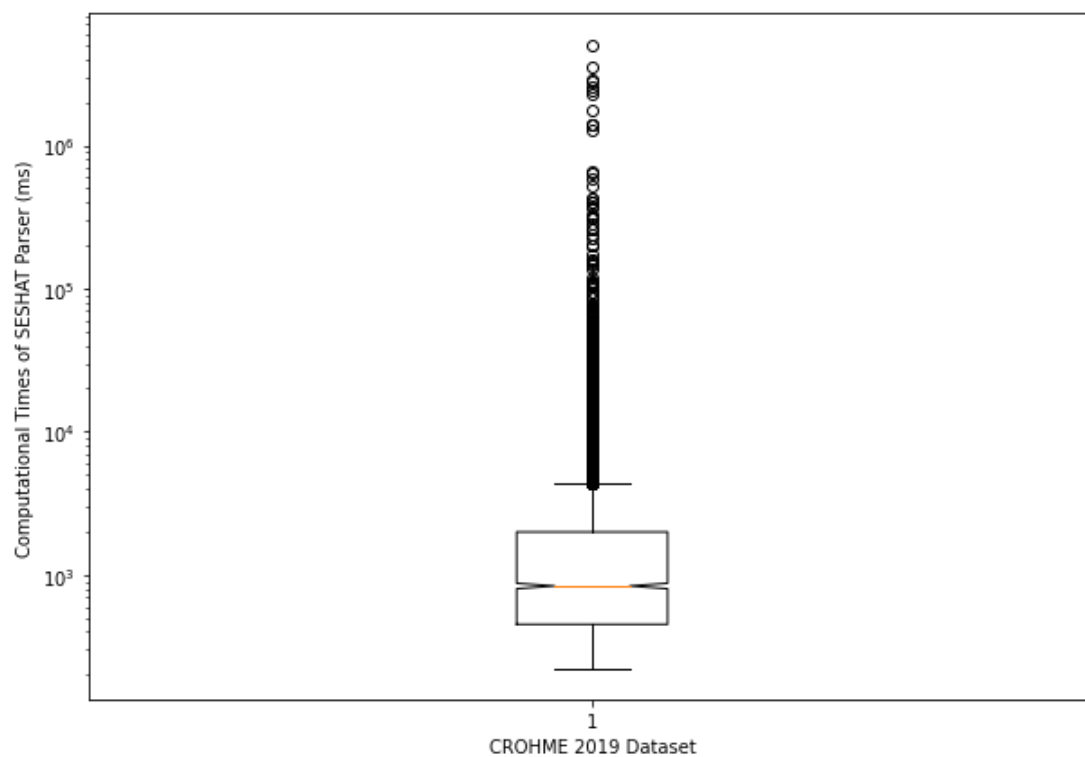


Figure 5-5: A boxplot of the Computational Times of the SESHAT Parser on CROHME 2019

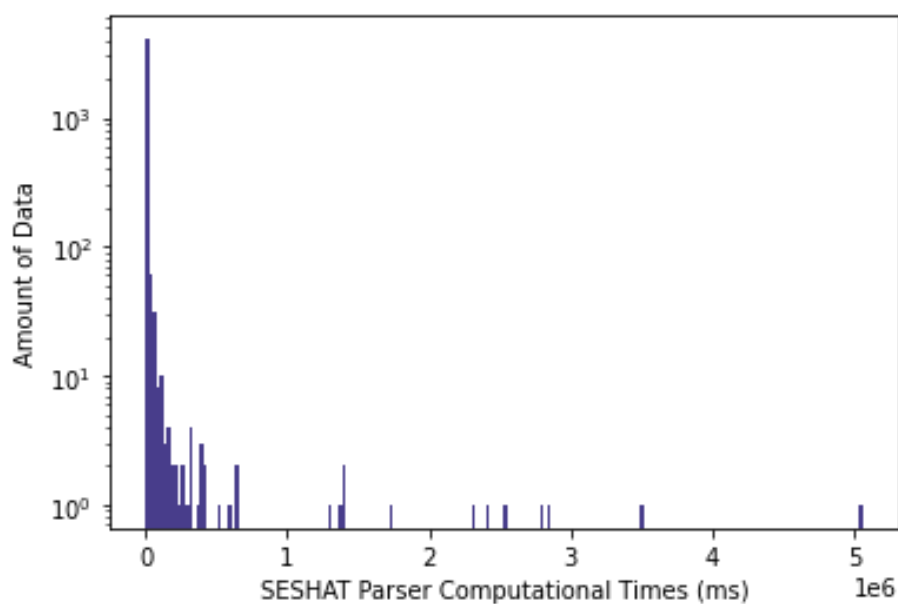


Figure 5-6: Histogram Plot of SESHAT Parser Computational Times on CROHME 2019

### Comparison with SOTA Models

A comparison is done with several SOTA models, summarised below in Table 5-2.

dataset/models	CROHME 2014	CROHME 2016	CROHME 2019
Offline SCAN + WAP	52.3	53.4	52.4
SESHAT Parser	32.5	35.7	42.1
Tesseract 4	102.3	102.8	103.7
Keras CNN	114.0	119.0	114.0

*Table 5-2: CER Comparison with SOTA Model (1dp)*

## Conclusions and Recommendations

---

In conclusion, the research project consisted of evaluating and comparing the results and accuracies of three models - a custom Keras CNN model, the Tesseract 4 OCR Engine by Google, and the SESHAT Parser, on the CROHME 2014, 2016 and 2019 datasets for Offline HMER. A custom data preprocessing pipeline was developed for the corresponding data, as well as a custom LaTeX tokenization pipeline to process the corresponding groundtruths, with the aid of the Katex parser by Harvard. The results show SESHAT parser to be the most effective at recognising handwritten mathematical expressions, with the Keras CNN model being the least effective.

The project and its results underline how difficult and complex the field of offline HMER is. In fact, it could be observed, when compared to the SOTA models, that the field of HMER is still in a process of active development and research.

The results could also be attributed to the relatively small size of the CROHME datasets, and with most data augmentation techniques being unsuitable to be applied to mathematical expressions, this is a major obstacle in the field of HMER. One method proposed which could circumvent such limitations is data segmentation, whereby a longer mathematical expression is segmented into smaller subproblems. An extreme of such an approach would be character segmentation, whereby each individual character in the equation is segmented and extracted, recognised, and the final equation reconstructed. Such models and processes were not implemented and investigated in this project due to the limitations of time and circumstance, but it nonetheless shows potential to have an improvement in results.

Looking ahead, further work could be done investigating how different data preprocessing methods such as data segmentation could be done to further improve the performance of the models.

## References

---

- [1] Z. Li, L. Jin, S. Lai, and Y. Zhu, “Improving Attention-Based Handwritten Mathematical Expression Recognition with Scale Augmentation and Drop Attention,” in *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, Sep. 2020, pp. 175–180. doi: 10.1109/ICFHR2020.2020.00041.
- [2] M. Mahdavi, R. Zanibbi, H. Mouchere, C. Viard-Gaudin, and U. Garain, “ICDAR 2019 CROHME + TFD: Competition on Recognition of Handwritten Mathematical Expressions and Typeset Formula Detection,” in *2019 International Conference on Document Analysis and Recognition (ICDAR)*, Sydney, Australia, Sep. 2019, pp. 1533–1538. doi: 10.1109/ICDAR.2019.00247.
- [3] J. Wang, Y. Sun, and S. Wang, “Image To Latex with DenseNet Encoder and Joint Attention,” *Procedia Comput. Sci.*, vol. 147, pp. 374–380, 2019, doi: 10.1016/j.procs.2019.01.246.
- [4] “Handwritten Math Recognition in Windows—Wolfram Language Documentation.” <https://reference.wolfram.com/language/tutorial/HandwrittenMathRecognition.html> (accessed Nov. 12, 2021).
- [5] V. R. N. dos Santos, W. Al-Nuaimy, J. L. Porsani, N. S. T. Hirata, and H. S. Alzubi, “Spectral analysis of ground penetrating radar signals in concrete, metallic and plastic targets,” *J. Appl. Geophys.*, vol. 100, pp. 32–43, Jan. 2014, doi: 10.1016/j.jappgeo.2013.10.002.
- [6] C. Jayant, “A Survey of Math Accessibility For Blind Persons and An Investigation on Text/Math Separation,” Apr. 2011.
- [7] “Online Handwritten Mathematical Expression Recognition and Applications: A Survey | IEEE Journals & Magazine | IEEE Xplore.” <https://ieeexplore.ieee.org/document/9367185> (accessed Nov. 12, 2021).
- [8] L. Anthony, J. Yang, and K. R. Koedinger, “Evaluation of multimodal input for entering mathematical equations on the computer,” in *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, Portland OR USA, Apr. 2005, pp. 1184–1187. doi: 10.1145/1056808.1056872.

- [9] P. Jayabala, E.Srinivasan, and S.Himavathi, "Diagonal Based Feature Extraction for Handwritten Alphabets Recognition System Using Neural Network," *Int. J. Comput. Sci. Inf. Technol.*, vol. 3, Feb. 2011.
- [10] "The Architecture of Mathematics: The American Mathematical Monthly: Vol 57, No 4." <https://www.tandfonline.com/doi/abs/10.1080/00029890.1950.11999523> (accessed Nov. 12, 2021).
- [11] "A method for the structural analysis of two-dimensional mathematical expressions | Semantic Scholar." <https://www.semanticscholar.org/paper/A-method-for-the-structural-analysis-of-expressions-Chang/62c3a0d50fd9ffd50db4ad1280f3018cd89b4466> (accessed Nov. 12, 2021).
- [12] "Yann LeCun - A.M. Turing Award Laureate." [https://amturing.acm.org/award\\_winners/lecun\\_6017366.cfm](https://amturing.acm.org/award_winners/lecun_6017366.cfm) (accessed Nov. 12, 2021).
- [13] A. Vaswani *et al.*, "Attention Is All You Need," *ArXiv170603762 Cs*, Dec. 2017, Accessed: Nov. 07, 2020. [Online]. Available: <http://arxiv.org/abs/1706.03762>
- [14] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3104–3112. Accessed: Oct. 28, 2020. [Online]. Available: <http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf>
- [15] J. Zhang, J. Du, and L. Dai, "Multi-Scale Attention with Dense Encoder for Handwritten Mathematical Expression Recognition," *ArXiv180103530 Cs*, Jan. 2018, Accessed: Nov. 12, 2021. [Online]. Available: <http://arxiv.org/abs/1801.03530>
- [16] H. Mouchère, R. Zanibbi, U. Garain, and C. Viard-Gaudin, "Advancing the state of the art for handwritten math recognition: the CROHME competitions, 2011–2014," *Int. J. Doc. Anal. Recognit. IJDAR*, vol. 19, no. 2, pp. 173–189, Jun. 2016, doi: 10.1007/s10032-016-0263-5.
- [17] B. S. Saroui, "Recognition of mathematical handwriting on whiteboards," p. 168.
- [18] A. Ahmad, "On-line Handwriting Recognition using Support Vector Machines and Hidden Markov Models approaches," Dec. 2008.
- [19] S. Mori, "Historical Review of Theory and Practice of Handwritten Character Recognition," in *Fundamentals in Handwriting Recognition*, Berlin, Heidelberg, 1994, pp.

- 43–69. doi: 10.1007/978-3-642-78646-4\_3.
- [20] “Ink Markup Language (InkML).” <https://www.w3.org/TR/InkML/> (accessed Nov. 12, 2021).
- [21] Z.-Q. Liu, J. Cai, and R. Buse, “Pre-processing and Feature Extraction,” in *Handwriting Recognition: Soft Computing and Probabilistic Approaches*, Z.-Q. Liu, J. Cai, and R. Buse, Eds. Berlin, Heidelberg: Springer, 2003, pp. 17–60. doi: 10.1007/978-3-540-44850-1\_2.
- [22] “online and offline handwriting recognition a comprehensive survey - Google Search.” [https://www.google.com/search?q=online+and+offline+handwriting+recognition+a+comprehensive+survey&rlz=1C1VDKB\\_enSG930SG930&oq=offline+and+online+handwriting&aqs=chrome.2.69i57j0i22i30l3j69i60l2.4746j0j7&sourceid=chrome&ie=UTF-8](https://www.google.com/search?q=online+and+offline+handwriting+recognition+a+comprehensive+survey&rlz=1C1VDKB_enSG930SG930&oq=offline+and+online+handwriting&aqs=chrome.2.69i57j0i22i30l3j69i60l2.4746j0j7&sourceid=chrome&ie=UTF-8) (accessed Nov. 12, 2021).
- [23] M. Chablani, “Sequence to sequence model: Introduction and concepts,” *Medium*, Jun. 23, 2017.  
<https://towardsdatascience.com/sequence-to-sequence-model-introduction-and-concepts-44d9b41cd42d> (accessed Nov. 12, 2021).
- [24] “Sequence to Sequence Deep Learning (Quoc Le, Google) - YouTube.” [https://www.youtube.com/watch?v=G5RY\\_SUJih4](https://www.youtube.com/watch?v=G5RY_SUJih4) (accessed Nov. 12, 2021).
- [25] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” *ArXiv14090473 Cs Stat*, May 2016, Accessed: Nov. 07, 2020. [Online]. Available: <http://arxiv.org/abs/1409.0473>
- [26] “GitHub - tensorflow/nmt: TensorFlow Neural Machine Translation Tutorial.” <https://github.com/tensorflow/nmt> (accessed Nov. 12, 2021).
- [27] D. Zhelezniakov, V. Zaytsev, and O. Radyvonenko, “Online Handwritten Mathematical Expression Recognition and Applications: A Survey,” *IEEE Access*, vol. 9, pp. 38352–38373, 2021, doi: 10.1109/ACCESS.2021.3063413.
- [28] M. Liwicki, A. Graves, J. Schmidhuber, and H. Bunke, “A Novel Approach to On-Line Handwriting Recognition Based on Bidirectional Long Short-Term Memory Networks,” p. 5.
- [29] J.-W. Wu, F. Yin, Y.-M. Zhang, X.-Y. Zhang, and C.-L. Liu, “Handwritten Mathematical Expression Recognition via Paired Adversarial Learning,” *Int. J. Comput. Vis.*, vol. 128, no. 10, pp. 2386–2401, Nov. 2020, doi: 10.1007/s11263-020-01291-5.



- [30] “[1909.01144] Bidirectional Long Short-Term Memory (BLSTM) neural networks for reconstruction of top-quark pair decay kinematics.” <https://arxiv.org/abs/1909.01144> (accessed Nov. 12, 2021).
- [31] “Hardware architecture of Bidirectional Long Short-Term Memory Neural Network for Optical Character Recognition | IEEE Conference Publication | IEEE Xplore.” <https://ieeexplore.ieee.org/document/7927210> (accessed Nov. 12, 2021).
- [32] J. Brownlee, “How to Develop a CNN for MNIST Handwritten Digit Classification,” *Machine Learning Mastery*, May 07, 2019. <https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/> (accessed Nov. 12, 2021).
- [33] J. J. Hull, “A database for handwritten text recognition research,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 5, pp. 550–554, May 1994, doi: 10.1109/34.291440.
- [34] “Advances in Handwriting Recognition | Series in Machine Perception and Artificial Intelligence.” <https://www.worldscientific.com/worldscibooks/10.1142/3981> (accessed Nov. 12, 2021).
- [35] “Results of preprocessing steps applying on a handwritten character. | Download Scientific Diagram.” [https://www.researchgate.net/figure/Results-of-preprocessing-steps-applying-on-a-handwritten-character\\_fig2\\_267859966](https://www.researchgate.net/figure/Results-of-preprocessing-steps-applying-on-a-handwritten-character_fig2_267859966) (accessed Nov. 12, 2021).
- [36] “Matlab Tutorial : Digital Image Processing 6 - Smoothing : Low pass filter - 2020.” [https://www.bogotobogo.com/Matlab/Matlab\\_Tutorial\\_Digital\\_Image\\_Processing\\_6\\_Filter\\_Smoothing\\_Low\\_Pass\\_special\\_filter2.php](https://www.bogotobogo.com/Matlab/Matlab_Tutorial_Digital_Image_Processing_6_Filter_Smoothing_Low_Pass_special_filter2.php) (accessed Nov. 12, 2021).
- [37] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, “A Gentle Introduction to Bilateral Filtering and its Applications,” p. 130.
- [38] “Smoothing Images — OpenCV-Python Tutorials beta documentation.” [https://opencv24-python-tutorials.readthedocs.io/en/latest/py\\_tutorials/py\\_imgproc/py\\_filtering/py\\_filtering.html](https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html) (accessed Nov. 12, 2021).
- [39] *Digital Signal Processing*. Elsevier, 2003. doi: 10.1016/B978-0-7506-7444-7.X5036-5.
- [40] Y. Ç. Aktaş, “A Comprehensive Guide to Image Processing: Part 3,” *Medium*, Sep. 02, 2021. <https://towardsdatascience.com/image-processing-part-3-dbf103622909> (accessed Nov. 12, 2021).

- [41] “OpenCV: Eroding and Dilating.”  
[https://docs.opencv.org/3.4.15/db/df6/tutorial\\_erosion\\_dilatation.html](https://docs.opencv.org/3.4.15/db/df6/tutorial_erosion_dilatation.html) (accessed Nov. 12, 2021).
- [42] “Types of Morphological Operations - MATLAB & Simulink.”  
<https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html>  
 (accessed Nov. 12, 2021).
- [43] “ML | K-means++ Algorithm,” *GeeksforGeeks*, Aug. 19, 2019.  
<https://www.geeksforgeeks.org/ml-k-means-algorithm/> (accessed Nov. 12, 2021).
- [44] G. Maindola, “4 Image Segmentation Techniques in OpenCV Python,” *MLK - Machine Learning Knowledge*, Sep. 06, 2021.  
<https://machinelearningknowledge.ai/image-segmentation-in-python-opencv/> (accessed Nov. 12, 2021).
- [45] “22.1.2. Parsing of Numbers and Symbols.”  
<https://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node189.html> (accessed Nov. 12, 2021).
- [46] “What is Tokenization | Methods to Perform Tokenization.”  
<https://www.analyticsvidhya.com/blog/2019/07/how-get-started-nlp-6-unique-ways-perform-tokenization/> (accessed Nov. 12, 2021).
- [47] “Byte-Pair Encoding: Subword-based tokenization | Towards Data Science.”  
<https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0> (accessed Nov. 12, 2021).
- [48] K. Chung, “On Subword Units: Segmentation for Natural Language Modeling.”  
[https://everdark.github.io/k9/notebooks/ml/natural\\_language\\_understanding/subword\\_units/subword\\_units.nb.html](https://everdark.github.io/k9/notebooks/ml/natural_language_understanding/subword_units/subword_units.nb.html) (accessed Nov. 12, 2021).
- [49] Z. Wang and J.-C. Liu, “Translating Math Formula Images to LaTeX Sequences Using Deep Neural Networks with Sequence-level Training,” *ArXiv190811415 Cs Stat*, Sep. 2019, Accessed: Nov. 12, 2021. [Online]. Available: <http://arxiv.org/abs/1908.11415>
- [50] *im2markup*. HNLP, 2021. Accessed: Nov. 12, 2021. [Online]. Available: <https://github.com/harvardnlp/im2markup>
- [51] J. Brownlee, “Why One-Hot Encode Data in Machine Learning?,” *Machine Learning Mastery*, Jul. 27, 2017.  
<https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>

- (accessed Apr. 20, 2021).
- [52] *opencv-python: Wrapper package for OpenCV python bindings*. Accessed: Nov. 12, 2021. [Online]. Available: <https://github.com/skvark/opencv-python>
  - [53] “NumPy.” <https://numpy.org/> (accessed Nov. 12, 2021).
  - [54] “Keras: the Python deep learning API.” <https://keras.io/> (accessed Nov. 12, 2021).
  - [55] “Using WSL for Developing Linux Applications on Windows Systems.” <https://www.opensourceforu.com/2021/03/the-advantages-of-using-wsl-for-developing-linux-applications-on-windows-systems/> (accessed Nov. 12, 2021).
  - [56] R. Christensen, “Thoughts on prediction and cross-validation,” p. 8.
  - [57] “Prediction error estimation: a comparison of resampling methods | Bioinformatics | Oxford Academic.” <https://academic.oup.com/bioinformatics/article/21/15/3301/195433> (accessed Nov. 12, 2021).
  - [58] “What is Cross-Validation?. Also, what are LOOCV and k-Fold... | by Sangeet Aggarwal | Towards Data Science.” <https://towardsdatascience.com/what-is-cross-validation-622d5a962231> (accessed Nov. 12, 2021).
  - [59] J. Brownlee, “LOOCV for Evaluating Machine Learning Algorithms,” *Machine Learning Mastery*, Jul. 26, 2020. <https://machinelearningmastery.com/loocv-for-evaluating-machine-learning-algorithms/> (accessed Nov. 12, 2021).
  - [60] “GitHub - JianshuZhang/WAP: Watch, Attend and Parse for Handwritten Mathematical Expression Recognition.” <https://github.com/JianshuZhang/WAP> (accessed Nov. 12, 2021).
  - [61] W. Wang, *Watch-Attend-and-Parse-tensorflow-version*. 2021. Accessed: Nov. 12, 2021. [Online]. Available: <https://github.com/wwjwhen/Watch-Attend-and-Parse-tensorflow-version>
  - [62] “GitHub - jungomi/math-formula-recognition: Math formula recognition (Images to LaTeX strings).” <https://github.com/jungomi/math-formula-recognition> (accessed Nov. 12, 2021).
  - [63] “(PDF) MfrDB: Database of Annotated On-Line Mathematical Formulae.” [https://www.researchgate.net/publication/261313749\\_MfrDB\\_Database\\_of\\_Annotated\\_On-Line\\_Mathematical\\_Formulae](https://www.researchgate.net/publication/261313749_MfrDB_Database_of_Annotated_On-Line_Mathematical_Formulae) (accessed Nov. 12, 2021).
  - [64] “Home.” <https://www.scg.uwaterloo.ca/mathbrush/> (accessed Nov. 12, 2021).

- [65] S. Quiniou *et al.*, “HAMEX - a Handwritten and Audio Dataset of Mathematical Expressions,” in *11th International Conference on Document Analysis and Recognition, ICDAR 2011*, Beijing, China, Sep. 2011, p. Accessed: Nov. 12, 2021. [Online]. Available: <https://hal.archives-ouvertes.fr/hal-00615210>
- [66] D. Huynh, *vndee/offline-crohme*. 2021. Accessed: Nov. 12, 2021. [Online]. Available: <https://github.com/vndee/offline-crohme>
- [67] “Papers with Code - ImageNet Dataset.” <https://paperswithcode.com/dataset/imagenet> (accessed Nov. 12, 2021).
- [68] “13 Data Augmentation Techniques,” Apr. 30, 2021. <https://research.aimultiple.com/data-augmentation-techniques/> (accessed Nov. 12, 2021).
- [69] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J. Big Data*, vol. 6, no. 1, p. 60, Jul. 2019, doi: 10.1186/s40537-019-0197-0.
- [70] “tesseract – opensource.google.” <https://opensource.google/projects/tesseract> (accessed Nov. 12, 2021).
- [71] A. Kay, “Tesseract: an Open-Source Optical Character Recognition Engine,” *Linux J.*, 2007, [Online]. Available: <http://www.linuxjournal.com/article/9676>
- [72] “Improve OCR Accuracy With Advanced Image Preprocessing,” *Docparser*, Nov. 29, 2017. <https://docparser.com/blog/improve-ocr-accuracy/> (accessed Nov. 12, 2021).
- [73] “Build your own OCR(Optical Character Recognition) for free | by Balaaji Parthasarathy | Medium.” <https://medium.com/@balaajip/optical-character-recognition-99aba2dad314> (accessed Nov. 12, 2021).
- [74] M. Lee, *pytesseract: Python-tesseract is a python wrapper for Google’s Tesseract-OCR*. Accessed: Nov. 12, 2021. [Online]. Available: <https://github.com/madmaze/pytesseract>
- [75] “Q&A: Indic - length of the compressed codes · Issue #654 · tesseract-ocr/tesseract,” *GitHub*. <https://github.com/tesseract-ocr/tesseract/issues/654> (accessed Nov. 12, 2021).
- [76] “Training Tesseract 4 models from real images | End Point Dev.” <https://www.endpointdev.com/blog/2018/07/training-tesseract-models-from-scratch/> (accessed Nov. 12, 2021).
- [77] N. Patry, *SESHAT: Handwritten math expression parser*. 2021. Accessed: Nov. 12, 2021. [Online]. Available: <https://github.com/Narsil/seshat>
- [78] “An integrated grammar-based approach for mathematical expression recognition -

ScienceDirect.” <https://www.sciencedirect.com/science/article/abs/pii/S0031320315003441>  
(accessed Nov. 12, 2021).