MathReader: API for Handwritten Mathematical Expressions Recognition

1st Caroline Santos dos Reis
Computer Science
Universidade Luterana do Brasil (ULBRA)
Canoas, Brazil
caroline.s_reis@icloud.com

2nd Fabiana Lorenzi
Universidade Luterana do Brasil (ULBRA) and
Rede Brasileira de Aprendizagem Criativa Núcleo POA
fabilorenzi@gmail.com

***Abstract*—This paper presents *MathReader*, an API that recognizes handwritten mathematical expressions through the use of a Convolutional Neural Network, to facilitate the input of mathematical notation in computational devices. Besides Neural Networks, *MathReader* uses computer graphics, structural, lexical, and syntactic analysis algorithms. A web application was developed to validate the API and the results are presented.**

***Index Terms*—convolutional neural networks, handwritten recognition, mathematical expressions, structural analysis of mathematical expressions, mathematical expressions recognition**

I. Introduction

Computational devices have become an important tool for writing; however, the writing of mathematical expressions in these devices remains a challenge. Despite the advances in the field of Computer Science, the need to facilitate the use of these devices to process mathematical data still remains. According to [1], the representation and the processing of mathematical notation in computers are difficult if compared to the plain text, which can be easily manipulated.

Some applications which accept mathematical notation such as text editors and educational software have support for mathematical writing through tools such as LATEX or through visual environments that allow the insertion of mathematical symbols utilizing the *mouse*. These approaches for writing mathematics in computers may not be practical and may be hard for some users because using LATEX requires learning their keywords to represent the mathematical notation and, utilizing the *mouse* to choose each symbol individually is an arduous task [2].

LATEX language gives users a higher control at the creation of documents allowing the insertion of formulas, tables, and figures in a simpler and powerful way. However, it is not possible to compare the use of tools such as LATEX to the facility of writing mathematical expressions drawing on a tablet or with a pen and paper. The use of mathematical notation is part of everyday life in several areas, such as science, engineering, mathematics, physics, and education [3] [4], making useful in many contexts a software capable of recognizing handwritten mathematical expressions to facilitate the insertion of mathematical notation in computational devices.

Based on the discussed issues, this paper presents *MathReader* API that allows handwritten mathematical expression recognition made on devices such as tablets, smartphones, graphics tablet or other computing devices with a touchscreen for recording. The result of handwriting recognition is the expression in LATEX, which can be used later in any software that accepts LATEX as input data.

MathReader API was developed using a convolutional neural network to perform the symbol classification, computer graphics algorithms to perform image processing, structural analysis algorithm to identify the spatial relationships between symbols, lexical and syntactic analysis to check if the generated expression is valid, and error correction to correct any grammar errors caused during symbol classification.

This paper is organized as follows: the next section presents the related work; section III presents the *MathReader* development, section IV presents the experiments and validation and finally, section V presents the concluding remarks.

II. Related Work

This section presents the main concepts necessary for understanding the work, as well as related work.

A. Convolutional Neural Networks

Convolutional Neural Networks are deep artificial neural networks that have achieved impressive results in tasks such as visual recognition, speech recognition and natural language processing [5], [6]. They can be used for several purposes such as image and video classification, recognition of characters, 2D and 3D objects, face expressions [7] and documents [8], semantic parsing, search query retrieval, and sentence modeling [5].

The convolutional layers have convolution filters that are applied to local features [5] to learn the feature vector from the training data in an unsupervised manner [8], [9]. Thus, the feature extraction is integrated with the classification [9] discarding the previous hand-crafted feature extraction step [8].

The pioneer work presented by [10] used Convolutional Neural Networks for handwritten digit recognition and provided a breakthrough in image recognition field allowing derivative works to be made, such as the recognition of handwritten mathematical expressions. The architecture comprises two convolutional layers with 6 and 16 filters respectively, two

2375-0197/20/$31.00 ©2020 IEEE
DOI 10.1109/ICTAI50040.2020.00193

Authorized licensed use limited to: Western Sydney University. Downloaded on June 14,2021 at 15:00:16 UTC from IEEE Xplore. Restrictions apply.

max-pooling layers and two fully connected layers with 120 and 84 neurons each. The $tanh$ activation function was used in the convolutional layers and the Gaussian activation was used in the output layer which has 10 neurons representing the digits [0, 9].

Another work [11] presents an architecture that contains five convolutional layer, three fully connected layers and the output layer containing 1.000 neurons. The ReLU activation function is applied to all convolutional and fully connected layers while the softmax activation function is applied to the output layer. Data Augmentation and Dropout techniques were applied to reduce overfitting.

In [12] the authors investigated the effect of very deep convolutional networks with small convolution filter for large-scale image recognition. They developed five different architectures named from *A* to *E* increasing the depth in each one. The best results came from the architectures with more layers (*D* and *E*), demonstrating an improvement on the performance of conventional architectures by adding more depth.

B. *Handwritten Symbol Recognition*

In [13] the authors developed a system to recognize handwritten symbols and divided the process into four stages: preprocessing, feature extraction, classification, and recognition. The first step consists of operations such as applying filters, image binarization and thinning. In the second step, the features are obtained through frequency information and edge-symbol distance, segmenting the symbols in the dimension 70x50 pixels, and using 79 attributes to classify the handwritten symbols. The classification step was carried out using two different approaches: one using maximum and minimum values and the other using a multi-layer perceptron neural network, which obtained the best result.

Another work is presented by [14], where the authors developed a system for recognition of mathematical expressions that use parse forest and Bayesian network to recognize the symbols of the expression.

In [15] the authors used support vector machines and projection histograms to identify simple handwritten mathematical equations. At the image preprocessing step, noise reduction filters techniques were applied to avoid errors in the handwritten recognition. The segmentation was achieved using global automatic thresholding and by binarizing. The achieved recognition rate was 98.26%, carrying out tests with an average of 14 expressions and observing the number of symbols recognized correctly.

Multi-layer perceptron neural networks was used in [2] with a symbol hypothesis generator to list all possible combinations of symbol strokes. A language model was also defined to allow the validation of the resulting expression from the specified grammar. The accuracy of symbol segmentation, symbol recognition, and expression recognition were, respectively, 94.8%, 84.8%, and 29.2%.

In the work presented in [16], the authors used convolutional neural networks for the task of recognizing mathematical equations. In a test set of 1.000 images, the system correctly recognized 39.11% of the equations and the symbol segmentation accuracy was 91.08%. Another work is presented in [17], where the authors used neural networks with Gradient Descent Optimizer and obtained 90% accuracy in the recognition rate in the neural network training.

The authors in [18] developed a method for the handwritten polynomials recognition using convolutional neural networks for the recognition task and *Fractional Order Darwinian Particle Swarm Optimization* (FODPSO) for symbol segmentation. Three convolutional neural networks were used to recognize the symbols that compose the expression. The first neural network classifies the symbols into two classes: *number* or *not-number*. Depending on the result, the symbol is sent to the next neural network corresponding to the result. The symbol recognition achieved approximately 99% accuracy in the three neural networks. Seven handwritten polynomials were tested and recognized correctly.

III. *MathReader* Development

To facilitate the writing of mathematical notation in computer systems, this paper presents the development of an API that allows the recognition of handwritten mathematical expressions, generating the expression in LaTeX format and making it easier to insert the mathematical expressions in other applications. The API can be used both in online or offline systems, depending on the deployment and integration method used. The *MathReader* API package can be found at the *PyPI* repository [19] and installed via *Pip* package manager [20], and its source code can be found at *GitHub* [21]. This section describes the features and the workflow of the API.

The API receives a handwritten mathematical expression written on a touchscreen device such as a smartphone, tablet, graphics tablet, or laptop with a touchscreen. The expression is recognized and translated into the computational language LaTeX so that it can be used later, according to the need of the system that integrates the API. Fig. 1 illustrates the API flow.

The mathematical operations accepted by *MathReader* are addition, subtraction, multiplication, division, potentiation, and square root. Accepted symbols are $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, x, y, z, m, n, =, \neq, +, -, *, \surd, \cdot, (,), \{, \}, [,]\}$. The multiplication operation is identified by the asterisk symbol "*" or by implicit multiplication $(2x)$ and the division operation is identified by the fraction bar symbol "—".

The API is made up of several components and each one is responsible for a step in the expression recognition. The main components that make up this process are:

- Image processing: this step applies computer graphics algorithms on the image such as color change, application of filters and symbol segmentation;
- Symbol classifier: it runs the neural network which classifies the symbols individually;
- Post-processing: this step joins the strokes from the symbol "=" that were segmented into individual symbols during the segmentation;
- Structural, Lexical and Syntactic Analysis: these steps validate the structure of the expression and its grammar;

Authorized licensed use limited to: Western Sydney University. Downloaded on June 14,2021 at 15:00:16 UTC from IEEE Xplore. Restrictions apply.

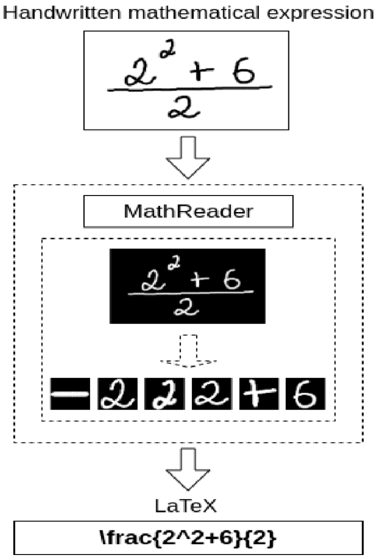


Fig. 1. API flow

- Error Correction: it tries to fix grammar errors that may have been caused due to errors during symbol classification;
- Module Integration: It integrates all modules above.

A. Image Processing

The image must be changed to the API format to be accepted by the neural network before the classification. Thus, some transformations must be applied to the image. First, it is applied a size reduction algorithm to cut the computational cost during image processing. Second, a noise reduction filter is applied to the image to facilitate the segmentation step. Third, the colors are changed to black and white and finally, the image is segmented into isolated symbols. After all these steps, the pixels' values of all segmented images are changed to [0,1] and sent individually to the neural network.

1) Image Preprocessing: An image with a width greater than 4.000 pixels is reduced to 20% of its width, respecting its proportion. The image is then converted from the RGB color scale to the grayscale. Subsequently, the *Nonlocal Means Denoising* filter [22] is applied to the image to reduce noise that can be confused as objects of interest and impair the symbol segmentation.

Then the image colors are inverted, transforming white into black, and black and shades of gray into white so that the black region becomes the background of the image and the white regions become the symbols to be segmented and classified. This modification is performed because the algorithm used to find the contours in an image requires a binary image; it looks for the white pixels on a black background to extract the white pixels as contours.

2) Symbol Segmentation: After the image processing, the *border following* algorithm [23] is applied to detect the contours of the image, and then it is segmented according to the contours obtained. For each identified contour, a mask (image with a black background) is created, large enough to cover the outlined symbol. The contour is then copied to the mask, creating a new image containing only the identified contour. This process performs the symbol segmentation. For achieving the correct symbol segmentation, the symbols must not be connected so that each symbol is represented by a single contour.

Each segmented symbol undergoes transformations such as resizing and adding a border to keep all images with the same size of 28x28 pixels. The original information of the symbol is stored to perform the structural analysis later. This information is the min(x, y) and max(x, y), width, height, and centroid coordinates. Finally, the pixel values of the images are converted from the range of [0, 255] to [0, 1].

B. Development of the symbol classifier

Symbols that compose the mathematical expression are classified by an artificial intelligence model trained through a convolutional neural network. The neural network presented in this paper is a convolutional neural network with five convolutional layers and two fully connected layers. The ReLU activation function is applied in all convolutional and fully-connected layers while the softmax function is applied to the output layer. Data Augmentation, Dropout and L2 regularization are used to prevent overfitting and increase the performance of the model. This step contains the following items: architecture and training of the convolutional neural network.

1) Architecture: It was developed a convolutional neural network for symbol classification which classifies the symbols among 30 classes, which are $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, x, y, z, m, n, \neq, +, -, *, \surd, \cdot, (,), \{, \}, [,] \}$.

The network architecture (shown in Fig. 2) consists of:

- Convolutional layer with 32 filters of size 3x3;
- Convolutional layer with 32 filters of size 3x3 and stride of 2x2 (to reduce the image size);
- Convolutional layer with 64 filters of size 3x3;
- Convolutional layer with 64 filters of size 3x3 and stride of 2x2 (to reduce the image size);
- Convolutional layer with 32 filters of size 3x3;
- Fully connected layers with 768 and 128 neurons, respectively;
- Output layer with 30 neurons representing the 30 classes.

2) Training: The neural network was trained using the *Handwritten Math Symbols* dataset (www.kaggle.com) and with images of the symbols ”$*$” and ”$\cdot$” drawn manually on a graphics tablet.

The *Handwritten Math Symbols* dataset has 82 classes and 375.974 symbol images in total, of which 28 classes were selected for training and 231.042 images of the following symbols $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, x, y, z, m, n, \neq, +, -, \surd, (,), \{, \}, [\ ,\ ]\ \}$. We got 11.618 and 2.810 images from the symbols $\{*, .\}$, respectively. The images were then processed (as mentioned in section III-A).

Authorized licensed use limited to: Western Sydney University. Downloaded on June 14,2021 at 15:00:16 UTC from IEEE Xplore. Restrictions apply.

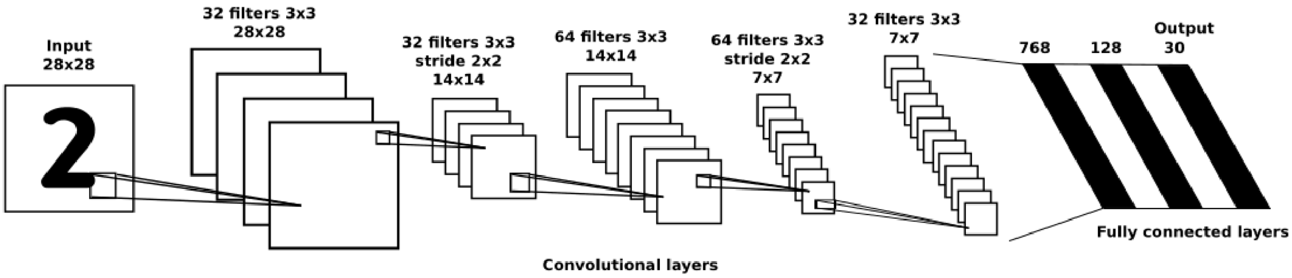


Fig. 2. Architecture of Convolutional Neural Network

To allow a broader representation of the dataset and to prevent overfitting, the data augmentation technique can be used. This method artificially augments the dataset by generating new samples from original data [24], [25]. Even with sufficient training samples, the data augmentation can increase the recognition rate as it generates new patterns [25]. For image recognition, simple transformations are usually applied to the images such as shear, rotation, flipping, and color modifications [26]. Data augmentation was used in the presented model on training data and applied shear and rotation transformations to the images. Shear range is between [-8, 8] and rotation range is between [-10, 10] degrees. Fig. 3 shows an example of data augmented with these transformations.

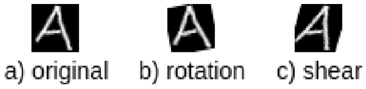

a) original b) rotation c) shear

Fig. 3. Data Augmentation

To reduce overfitting some regularizations techniques can be applied such as Dropout, L1, and L2 Regularization. Dropout randomly drops a few neurons from the layer to which it was applied based on a given percentage. The L1 Regularization moves the weights to zero while the L2 Regularization tends to move the weights closer to zero. L2 has the advantage of higher accuracy than L1. These regularizations can be seen as a forgetting mechanism which prevents the model from memorizing the data [27]. Thus, L2 regularization with a value of 0.01 was applied to the output of the fully connected hidden layers of this model to prevent overfitting.

The neural network was trained using Categorical Cross-Entropy Loss Function and Adamax optimizer [28]. Both convolutional and fully connected layers were trained with the ReLU activation function (shown in eq. (1) [27]) while the output layer was trained with softmax activation function shown in eq. (2) [27]. The fully connected layers have a 50% dropout [29]. The other parameters chosen for training the neural network are shown in Table I and the results of the training are shown in Table II. During the training of the model it was not possible to find signs of overfitting, demonstrating that the regularizations were able to improve the generalization of the model by preventing overfitting.

$$f(x) = max(0, x) \tag{1}$$

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}} \; for \; j = 1, ..., k \tag{2}$$

TABLE I
NEURAL NETWORK TRAINING PARAMETERS

Learning Rate	Batch Size	Epochs
0.001	448	173

TABLE II
TRAINING RESULTS

Accuracy		Loss	
Training	Validation	Training	Validation
99.23%	99.48%	0.06723	0.06280

C. Post-Processing

The segmentation step divides all unconnected strokes into individual symbols. There are cases, however, that symbols have unconnected strokes, such as the "=" symbol. In this case, the neural network identifies the "-" strokes separately and, after the symbol classification step, the strokes that compose the symbol "=" are joined if necessary in the post-processing step. To perform this union, the algorithm identifies if two lines share some position on the x-axis and checks if there is no line twice the size of the other.

D. Structural, Lexical, Syntactic Analysis

After symbol segmentation and symbol classification, the spatial structure of the mathematical expressions is obtained through structural analysis and then the expression is validated by performing the lexical and syntactic analysis.

1) Structural Analysis: The structural analysis is responsible for generating the LaTeX expression. In this step we applied the algorithm proposed by [30] that finds the relationships between the symbols through the coordinate information and through the regions defined around a symbol.

Symbol centroid is updated according to the type of symbol writing and according to its bounding box position. These

Authorized licensed use limited to: Western Sydney University. Downloaded on June 14,2021 at 15:00:16 UTC from IEEE Xplore. Restrictions apply.

changes were made to better represent the writing of each type of symbol.

The y-centroid value of the symbols {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, b} is changed to $min(y) + (2/3) * (max(y) - min(y))$ and of the symbols {y, $\surd$, (, [, {, },],),} is set to $min(y) + (1/3)*(max(y)-min(y))$. The other symbols remain with the default y-centroid definition: $min(y)+((max(y)-min(y)/2)$. The x-centroid value is set to $min(x)+((max(x)-min(x)/2)$ with the exception of the symbols { {, (, [} which the x-centroid value is set to $min(x)$ and of the symbols {, })] } which is changed to $max(x)$.

With this change, the algorithm was able to identify the relationships between symbols more accurately. This information is used to identify which region the symbol belongs to. If the symbol centroid is inside the area of a given region, then the symbol belongs to that region.

The regions **ABOVE**, **BELOW**, **SUPER**, **SUBSC**, **CONTAINS** and **HOR** were defined to identify the relations between symbols. Fig. 4 illustrates the delimitation of the regions of a symbol and the coordinates used to define each region are described in Table III, where the values $min(x)$, $max(x)$, $min(y)$ and $max(y)$ represent the minimum x-coordinate, maximum x-coordinate, minimum y-coordinate and maximum y-coordinate of the symbol, respectively. The values presented in Table III are the initial values used in the structural analysis to delimit the regions and, some of these values, are updated dynamically during the execution of the algorithm.

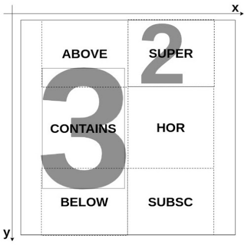


Fig. 4. Symbol Regions

TABLE III
INITIAL LIMITS FOR DELIMITING THE REGIONS OF A SYMBOL

Regions	**x**$_1$	**y**$_1$	**x**$_2$	**y**$_2$
ABOVE	min(x)	-1	max(x)	min(y) + $\frac{height}{7}$
CONTAINS	min(x)	min(y)	max(x)	max(y)
BELOW	min(x)	min(y) + ($\frac{6 \cdot height}{7}$)	max(x)	∞
SUPER	max(x)	-1	∞	min(y) + $\frac{height}{7}$
HOR	max(x)	min(y) + $\frac{height}{7}$	∞	min(y) + ($\frac{6 \cdot height}{7}$)
SUBSC	max(x)	min(y) + ($\frac{6 \cdot height}{7}$)	∞	∞

The regions **ABOVE** and **BELOW** are used to fraction; thus symbols above fraction line are added to the **ABOVE** region and symbols below fraction line are added to the **BELOW** region. For achieving the correct recognition of the fraction it is important that the fraction bar is greater than or equal to the bounding box of the numerator and denominator. The **SUPER** region is used for exponents of a symbol. The **SUBSC** region is used for decimal point and subscripts. The **CONTAINS** region is used to define symbols inside a square root. Finally, the **HOR** region is used to define horizontally adjacent symbols. After assigning each symbol to a region, the expression is stored in the *Baseline Structure Tree*, illustrated in Fig. 5.

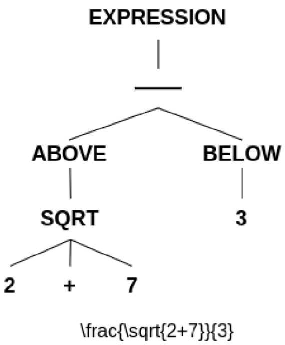


Fig. 5. Baseline Structure Tree

The definition of the regions around the symbol is necessary to perform the structural analysis; however, these are heuristic choices and are subject to errors and ambiguities. Symbols written with very different sizes and very misaligned writing, for example, may lead to region assignment errors, causing an error in the expression recognition.

2) Lexical and Syntactic Analysis: Lexical and syntactic analyses are performed using the *PLY* (*Python Lex Yacc*) tool [31] which is an implementation of the *Lex* and *Yacc* analysis tools for the *Python* programming language. *Lex* performs lexical analysis and *Yacc* performs syntactic analysis.

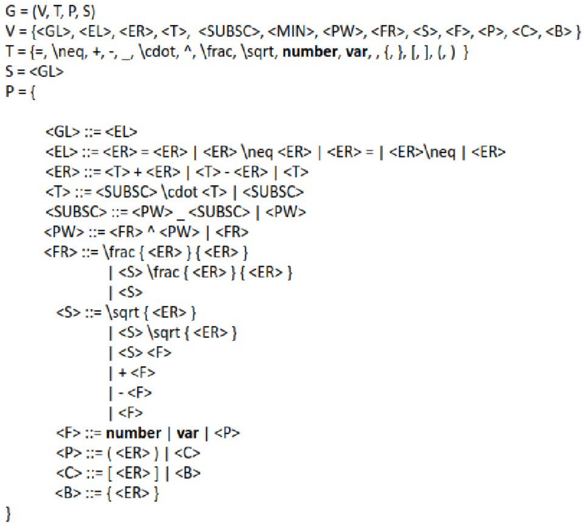
Fig. 6 presents the context-free grammar defined to describe the mathematical notation in LaTeX format [32]. Non-terminal symbol ***number*** represents the token ***number*** for numbers in the format of the regular expression "[0-9]+(?:\.[0-9]+)?", which accepts for instance the following values: {1, 0.5, 01.30, 2.0, 20.004}. Non-terminal symbol ***var*** represents the token ***var*** for the following variables {a, b, c, x, y, z, m, n}. The grammar allows implicit multiplication between the ***number*** and ***var*** tokens, such as $23x$. Besides, the grammar accepts the symbols "+" and "-" to the left of the ***number*** and ***var*** tokens.

E. Error correction

Error correction process aims to correct grammatical errors that may have been caused by failure during the symbol classification. This step is based on the descriptions of contextual analysis and error corrections from [32], [33], and [34].

This process was integrated with syntactic and lexical analysis, and attempts to correct errors made during the processes. First, the lexical analysis is performed along with the attempt to correct lexical errors if they occur. When there are no more lexical errors, the parsing module is called and, if there are

Authorized licensed use limited to: Western Sydney University. Downloaded on June 14,2021 at 15:00:16 UTC from IEEE Xplore. Restrictions apply.



```
G = (V, T, P, S)
V = {<GL>, <EL>, <ER>, <T>,  <SUBSC>, <MIN>, <PW>, <FR>, <S>, <F>, <P>, <C>, <B> }
T = {=, \neq, +, -, _, \cdot, ^, \frac, \sqrt, number, var, , {, }, [, ], (, )  }
S = <GL>
P = {

        <GL> ::= <EL>
        <EL> ::= <ER> = <ER> | <ER> \neq <ER> | <ER> = | <ER>\neq | <ER>
        <ER> ::= <T> + <ER> | <T> - <ER> | <T>
        <T> ::= <SUBSC> \cdot <T> | <SUBSC>
        <SUBSC> ::= <PW> _ <SUBSC> | <PW>
        <PW> ::= <FR> ^ <PW> | <FR>
        <FR> ::= \frac { <ER> } { <ER> }
                  | <S> \frac { <ER> } { <ER> }
                  | <S>
          <S> ::= \sqrt { <ER> }
                  | <S> \sqrt { <ER> }
                  | <S> <F>
                  | + <F>
                  | - <F>
                  | <F>
          <F> ::= number | var | <P>
          <P> ::= ( <ER> ) | <C>
          <C> ::= [ <ER> ] | <B>
          <B> ::= { <ER> }
}
```

Fig. 6. LaTeX Context-Free Grammar

syntactic errors, an attempt is made to correct them. If in the end, there are no more errors, the corrected expression is returned. The correction attempt is made a maximum of three times for each analysis. If it is not possible to solve the error, the original expression is returned along with the error data, list of attempted solutions, and classification data from the neural network.

When errors occur, information such as the symbol position and identity are necessary to enable the correction. This information is used along with the classification data of the neural network to choose the next most likely symbol and replace it in the position where the error was found. For instance, in case the expression $2+81$ is recognized as $2+8$), a syntactic error in the position of the ")" symbol will be returned. Assuming that, for this symbol ")", the next most likely symbol in the neural network prediction list is "1" then, the symbol ")" will be replaced by "1", and the lexical and syntactic analysis will occur again to validate the correction. In this case, after the correction, no lexical or syntactic error will be presented and the corrected expression will be returned. Some errors, however, are not possible to be solved [33]; therefore, it is not possible to guarantee that all errors will be solved in this step.

F. Module integration

The main modules that compose the *MathReader* are *parser*, *image_processing* and *classification*. The *api.py* file integrates the modules and provides access methods to perform expression recognition.

The main methods available in the API are:

- *recognize(image)*: Receives the image to be recognized, performs the recognition and returns the LaTeX of the expression;
- *get_predictions()*: Returns the list of prediction of the classification of symbols;
- *get_lex_errors()*: Returns the list of lexical errors;
- *yacc_errors()*: Returns the list of syntactic errors.

The image sent to the API must be in one of the following formats: *base64*, *bytes*, or *file path* in *.png* or *.jpg*/*.jpeg* formats. The image must not be generated with a black background and white symbols because during the image processing step, the colors of the image will be converted to a black background and white symbols.

IV. Validation and Results

We run the API validation in two different scenarios. In the first scenario, we validate the efficiency of the API and in the second scenario we validate the integration with the API and the user experience. Twelve volunteers were invited to validate the API, and they were either computer science students or software developers. Only two of them were familiar with LaTeX.

A. First Scenario: Evaluating the Expressions

To validate the LaTeX expressions generated by the API, 88 expressions were written by the authors and by the twelve volunteers. Table IV shows some of the evaluated expressions. The first column represents the expression identification, the second column contains the correct LaTeX of the expression and the third column shows "Ok" if the expression was correctly recognized or the incorrect LaTeX generated by the application.

TABLE IV
Expressions used for validation

Expression	**LaTeX**	**Generated LaTeX**
1	\frac{4x^2}{2a}	Ok
2	\frac{2+x^2}{a+b}	Ok
3	9c-x^2+\sqrt{9}	Ok
4	a=\frac{2}{a}	Ok
5	\sqrt{\frac{x^2-y^3}{2a^4}}	Ok
6	(\sqrt{9})^3	Ok
7	[\frac{8}{2}+\frac{4}{1}]	Ok
8	\frac{m^2-n^5}{2a}	Ok
9	0.39+62.43	Ok
10	6\cdota	Ok
11	a\neqb	Ok
12	(20)^{2+7}-8	Ok
13	2\neq75\cdot8	Ok
14	2.3-8	Ok
15	xy=2	Ok
16	\frac{2+4}{3+\sqrt{5}}	\frac{2+y}{3+\sqrt{5}}
17	(\frac{2\cdot3}{3})	(\frac{2\cdot3}{3}1
18	\frac{a\cdotb}{5a}	\frac{a+b}{5a}
19	2+(2\cdot3)	2+^{(}2\cdot3)
20	{a+b-[5\cdot7]}	(a+-[b_{5}\cdot7]}
21	0.4297-0.1+2.0	0.4297-0.b+2.0
22	\frac{x^2-34}{7}	\frac{x^2-3y}{7}
23	\frac{\sqrt{35^2}}{2}	\frac{\sqrt{357}}{2}
24	\frac{4\cdot2}{\sqrt{3^2}}	\frac{4\cdot2}{\sqrt{32}}
25	4\cdot4	4\cdot_4

Among the 88 tested expressions, 20 were not recognized correctly. It was possible to observe that many errors occurred due to symbol classification and structural analysis failures. Classification failures occur mainly between symbols that are very similar, making writing ambiguous. Expressions in which

Authorized licensed use limited to: Western Sydney University. Downloaded on June 14,2021 at 15:00:16 UTC from IEEE Xplore. Restrictions apply.

the position of the symbols fluctuate may suffer errors in the structural analysis due to the area defined for each region around the symbols and, to improve the performance of this step, new limits for the regions can be tested.

Recognition errors in expressions can happen due to the following reasons: symbol segmentation errors, symbol classification errors, structural analysis errors, or insertion of an invalid expression. The errors observed from Table IV are analyzed below.

The error in the expressions 16, 17, 18, 21 and 22 occurred due to errors in the classification step. In expression 16, the symbol ”4” was classified as ”y”. In expression 17 the symbol ”)” was classified as ”1”, as these symbols are similar and can be easily confused with each other during classification. In expression 18 the symbol ”$*$” was classified as ”+”, which are also similar symbols. In expression 21 symbol ”1” was classified as ”b” and in expression 22 the symbol ”4” was classified as ”y”.

In expressions 19, 24, and 25 the API failed to find the relationship between symbols during the structural analysis. In both cases, symbols were written slightly above or slightly below the expected region and they were identified in different regions than expected. In expression 19 the symbol ”(” was identified as an exponent of ”+”, it should be horizontally adjacent. The symbol ”2” in expression 24 was identified as horizontally adjacent to ”3”, but it should be an exponent of ”3”. The symbol ”4” in expression 25 was identified as a subscript of ”$\cdot$” instead of being identified as horizontally adjacent to ”$\cdot$”.

In expression 20, the symbol ”{” was identified as ”(” and the symbol ”5” was identified as a subscript of symbol ”b” - it should be horizontally adjacent. In expression 23, the error occurs due to wrong classification of the symbol ”2”, which was classified as ”7” and due to the error in structural analysis, which did not identify the symbol ”7” as an exponent of ”3”.

B. Second Scenario: API Integration and User Experience

A web application was developed to validate the integration of the API with another system, and its source code can be found at [35]. The volunteers were invited to install the API, use the web application, and write some expressions.

The API was installed using *Pip* package manager [20] and integrated with the validation application. We then validated the user experience considering the whole process (installing the API and using the web application).

The validation web application had the following elements: information about the work; a canvas to allow the writing of the mathematical expression with a menu containing a pen button; a button to erase the entire image; a button to perform the recognition, and a field where the result was showed.

Each volunteer wrote some expressions in the web application to be recognized. The images containing the mathematical expression were obtained in *base64* format and were sent to the API ”recognize” method to perform the recognition. These expressions and results were sent to the authors, so the errors could be analyzed.

In the end, each volunteer answered a *Google Form* about the experience of using the web application (integrated to the API) to generate the LaTeX expression. Two important questions got ”yes” as answer: ”Did you like to use the validation application?” and ”Do you think *MathReader* facilitates the insertion of mathematical expressions in computers?”. Also the following comments were written by the volunteers:

- ”It would be interesting to add the dot multiplication.”;
- ”I had a little difficulty making the asterisk with the laptop’s touchpad, but the API behaved correctly with the expressions created.”;
- ”I found this solution very useful because writing formulas in LaTeX is a challenge for anyone. Being able to count on a converter that can take complex formulas and do this translation in a fast and uncomplicated way is valuable. I did tests to think about large and diversified formulas so that the author can work harder to reach that level with this project.”;
- ”I liked the tool, in the future it will be able to help a lot in the academic environment and in the daily life inside an app. It needs minor adjustments, but it has already shown great potential.”;
- ”If you write the expression with a distance between the symbols, it works very well. If you write with overlapping symbols, it gives some errors, but it was explained to write the expressions with spacing between the symbols, so it’s perfect.”.

These answers represent that the web application integrated with the *MathReader* API made it easier to insert mathematical notation into computers and that the volunteers enjoyed the experience of using the interface to input the mathematical expressions.

V. Conclusions and Future Work

This paper presented the development of *MathReader* API that uses a convolutional neural network for recognizing handwritten mathematical expressions and generate the corresponding latex of the expression.

Through the development of a web application that integrated the API, it was possible to verify that the main goal of the API was achieved, allowing a system (the validation web application) to use the API and provide support for the handwriting of mathematical notation.

Volunteers who tested the validation application informed through a questionnaire that the insertion of mathematical writing through a tool that recognizes handwritten mathematical expressions is facilitated. Therefore, it is possible to state that the goal was achieved to provide the API with a system that recognizes handwritten mathematical expressions to facilitate the insertion of mathematical expressions in computer systems.

As future work, we intend to add more symbols, improve the performance of the symbol classifier, and allow the multiplication operation using the symbol ”.”. We also thought it would be important for the API to return the result of the mathematical expression in other formats, such as *MathML* [36], [37], a markup language that describes mathematical

Authorized licensed use limited to: Western Sydney University. Downloaded on June 14,2021 at 15:00:16 UTC from IEEE Xplore. Restrictions apply.

notations based on the *XML* markup language. Many browsers accept *MathML* to display the mathematical notation so it can be useful in applications such as computer algebra systems or to create dynamic mathematical websites, speech synthesis software, and so on.

References

[1] R. Miner, "The importance of mathml to mathematics communication," *Notices of the AMS*, vol. 52, no. 5, pp. 532–538, 2005.

[2] A.-M. Awal, H. Mouchere, and C. Viard-Gaudin, "Towards handwritten mathematical expression recognition," in *2009 10th International Conference on Document Analysis and Recognition*. IEEE, 2009, pp. 1046–1050.

[3] J.-W. Wu, F. Yin, Y.-M. Zhang, X.-Y. Zhang, and C.-L. Liu, "Handwritten mathematical expression recognition via paired adversarial learning," *International Journal of Computer Vision*, pp. 1–16, 2020.

[4] A. D. Le, B. Indurkhya, and M. Nakagawa, "Pattern generation strategies for improving recognition of handwritten mathematical expressions," *Pattern Recognition Letters*, vol. 128, pp. 255–262, 2019.

[5] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014, pp. 1746–1751. [Online]. Available: http://aclweb.org/anthology/D14-1181

[6] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," *Neurocomputing*, vol. 323, pp. 37–51, jan 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231218311007

[7] I. Ramadhan, B. Purnama, and S. A. Faraby, "Convolutional neural networks applied to handwritten mathematical symbols classification," in *2016 4th International Conference on Information and Communication Technology, ICoICT 2016*, vol. 4, no. c. IEEE, 2016, pp. 1–4.

[8] D. S. Maitra, U. Bhattacharya, and S. K. Parui, "CNN based common approach to handwritten character recognition of multiple scripts," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, aug 2015, pp. 1021–1025.

[9] I.-J. Kim, C. Choi, and S.-H. Lee, "Improving discrimination ability of convolutional neural networks by hybrid learning," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 19, no. 1, pp. 1–9, mar 2016. [Online]. Available: http://link.springer.com/10.1007/s10032-015-0256-9

[10] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, may 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3383972.3383975 https://dl.acm.org/doi/10.1145/3065386

[12] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–14, sep 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[13] R. A. R. Miranda, F. A. da Silva, M. A. Pazoti, A. O. Artero, and M. A. Piteri, "Algoritmo para o reconhecimento de caracteres manuscritos," in *Colloquium Exactarum. ISSN: 2178-8332*, vol. 5, no. 2, 2013, pp. 109–127.

[14] S. MacLean and G. Labahn, "A bayesian model for recognizing handwritten mathematical expressions," *Pattern Recognition*, vol. 48, no. 8, pp. 2433–2445, 2015.

[15] S. S. Gharde, P. V. Baviskar, and K. Adhiya, "Identification of handwritten simple mathematical equation based on svm and projection histogram," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 3, no. 2, pp. 425–429, 2013.

[16] M. B. Hossain, F. Naznin, Y. Joarder, M. Z. Islam, and M. J. Uddin, "Recognition and solution for handwritten equation using convolutional neural network," in *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*. IEEE, 2018, pp. 250–255.

[17] S. P. Ramteke, D. V. Patil, and N. P. Patil, "Neural network approach to mathematical expression recognition system," *International Journal of Engineering Research & Technology (IJERT)*, vol. 1, no. 10, pp. 2278–0181, 2012.

[18] F. d. C. F. M. Junior, T. P. de Araujo, J. V. M. Sousa, N. J. C. da Costa, R. T. Melo, A. M. Pinto, and A. A. Saraiva, "Recognition of simple handwritten polynomials using segmentation with fractional calculus and convolutional neural networks," in *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE, 2019, pp. 245–250.

[19] C. S. d. Reis and F. Lorenzi, "Mathreader," 2020. [Online]. Available: https://pypi.org/project/mathreader/

[20] "pip." [Online]. Available: https://pypi.org/project/pip/

[21] C. S. d. Reis and F. Lorenzi, "mathreader," 2020. [Online]. Available: https://github.com/carolreis/mathreader

[22] A. Buades, B. Coll, and J.-M. Morel, "Non-local means denoising," *Image Processing On Line*, vol. 1, pp. 208–212, 2011.

[23] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[24] C. Tensmeyer and T. Martinez, "Analysis of Convolutional Neural Networks for Document Image Classification," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 1. IEEE, nov 2017, pp. 388–393. [Online]. Available: http://ieeexplore.ieee.org/document/8270002/

[25] X. Qu, W. Wang, K. Lu, and J. Zhou, "Data augmentation and directional feature maps extraction for in-air handwritten Chinese character recognition based on convolutional neural network," *Pattern Recognition Letters*, vol. 111, pp. 9–15, aug 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167865518301211

[26] A. Mikolajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*. IEEE, may 2018, pp. 117–122. [Online]. Available: https://ieeexplore.ieee.org/document/8388338/

[27] C. C. Aggarwal, *Neural Networks and Deep Learning*. Cham: Springer International Publishing, 2018. [Online]. Available: http://link.springer.com/10.1007/978-3-319-94463-0

[28] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[29] B. Mele and G. Altarelli, "Lepton spectra as a measure of b quark polarization at LEP," *Physics Letters B*, vol. 299, no. 3-4, pp. 345–350, jan 1993. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/037026939390272J

[30] R. Zanibbi, "Recognition of mathematics notation via computer using baseline structure," Queen's University, Kingston, Ontario, Canada, Tech. Rep. August, 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.4193

[31] D. Beazley, "PLY (Python Lex-Yacc)." [Online]. Available: https://www.dabeaz.com/ply/

[32] U. Garain and B. B. Chaudhuri, "Recognition of Online Handwritten Mathematical Expressions," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 34, no. 6, pp. 2366–2376, 2004.

[33] D. BLOSTEIN and A. GRBAVEC, "RECOGNITION OF MATHEMATICAL NOTATION," in *Handbook of Character Recognition and Document Image Analysis*. WORLD SCIENTIFIC, may 1997, pp. 557–582.

[34] K.-F. Chan and D.-Y. Yeung, "Error detection, error correction and performance evaluation in on-line mathematical expression recognition," *Pattern Recognition*, vol. 34, no. 8, pp. 1671–1684, aug 2001. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0031320300001023

[35] C. S. d. Reis and F. Lorenzi, "mathreader-validation," 2020. [Online]. Available: https://github.com/carolreis/mathreader-validation

[36] "What is MathML?" [Online]. Available: https://www.w3.org/Math/whatIsMathML.html

[37] "Working with MathML—Wolfram Language Documentation." [Online]. Available: https://reference.wolfram.com/language/XML/tutorial/MathML.html

Authorized licensed use limited to: Western Sydney University. Downloaded on June 14,2021 at 15:00:16 UTC from IEEE Xplore. Restrictions apply.