2018 4th International Conference for Convergence in Technology (I2CT)
SDMIT Ujire, Mangalore, India. Oct 27-28, 2018

SolveIt : An Application for Automated Recognition and Processing of Handwritten Mathematical Equations

Sagar Bharadwaj KS*
Department of
Computer Science & Engineering
National Institute of Technology
Karnataka, Surathkal
Email: sagarbharadwaj50@gmail.com

Vilas Bhat*
Department of
Computer Science & Engineering
National Institute of Technology
Karnataka, Surathkal
Email: vilasnitk19@gmail.com

Arvind Sai Krishnan*
Department of
Computer Science & Engineering
National Institute of Technology
Karnataka, Surathkal
Email: arvindsaikrish@gmail.com

***Abstract*—Solving mathematical equations is an integral part of most, if not all forms of scientific studies. Researchers usually go through an arduous process of learning the nuances and syntactic complexities of a mathematical tool in order to solve or process mathematical equations. In this paper, we present a mobile application that can process an image of a handwritten mathematical equation captured using the device's camera, recognise the equation, form the corresponding string that can be parsed by a computer algebraic system and display all possible solutions. We aim to make the whole experience of experimenting with equations very user friendly and to remove the hassle of learning a mathematical tool just for mathematical experimentation.**

We propose a novel machine learning approach to recognise handwritten mathematical symbols achieving a 99.2% cross validation percentage accuracy on the kaggle math symbol dataset with reduced symbols. The application covers useful features like simultaneous equation solving, graph plotting and simple arithmetic computations from images. Overall it is a very user friendly equation solver that can leverage the power of existing powerful math packages.

Index Terms - CNN, Deep Learning, Mathematical expression recognition, Symbol recognition, SymPy

I. INTRODUCTION

Complexities involved in Human Computer Interaction (HCI) are reducing at an extraordinary rate and computers are gradually catching up with the nuances, irregularities and imprecisions of the real world to enable an easier interaction. Tremendous amount of research has gone into identifying and classifying some irregularities of the real world including handwriting recognition, natural language processing, face detection and so on. We have progressed leaps and bounds in enabling computers to understand our error prone but natural inputs, process them and produce outputs with a factor of usefulness. The input a computer takes has grown from the tedious punched cards to mouse clicks to touch. We are now progressing towards an era where no dedicated devices are required to instruct a machine to compute. We can now convey our instructions using natural languages and images. Computers can now 'listen' and 'see' like our fellow humans while still retaining their superior computational power. There has been humongous research on leveraging such new found computational capabilities and streamlining them into specific use cases. One use case that we explore in this paper is solving handwritten mathematical equations.

We have made an attempt at simplifying the interaction between humans and computers with respect to processing and solving mathematical equations. Mathematical equations are an integral part of most technical research. Researchers currently rely on the process of leveraging existing math engines and computer algebraic systems to solve or plot mathematical equations and expressions. However, in order to leverage existing solutions, one must be aware of the syntaxes and complexities of the chosen tool. Going through such an ordeal for some simple and 'throw away' equations is not desirable. Technical assignments, projects and research involves a lot of mathematical analysis that require one to experiment with a lot of throw away equations and expressions. Entering all such equations and expressions into the system requires lots of manual assistance, time, human intervention and an expertise in the chosen mathematical tool. A natural solution to this problem would be to develop a user friendly tool that would capture the image of a mathematical equation, recognise the equation embedded in it and present the user with the required solution. This is precisely the tool developed and described in this paper.

There are currently many production level math analysis

. * S. Bharadwaj, V. Bhat and A.S. Krishnan contributed equally to this work. Authors are ordered by lastname.

978-1-5386-5232-9/18/$31.00 ©2018 IEEE

tools available in the computing world. However, more the number of features in a mathematical tool, the higher is the difficulty of leveraging its advantages. Our tool can be easily integrated with any available mathematical tool because of the highly modular approach that we have used when developing the application. The power of our application is a function of the power and features of the integrated mathematical tool.

The application presented can be easily extended to other use cases. Some possible use cases may include automating verification of mathematical claims in technical documents; automating exam paper evaluations; conversion of handwritten equations into a desired typesetting format like Latex for inclusion in a technical document; verifying mathematical claims on online forums and so on.

Research on recognition of handwriting has been at the forefront in the recent years. Numerous methods are explored to enable computers to recognise and classify handwritten symbols. The methods are broadly classified into 'online' classifiers and 'offline' classifiers. Online classifiers perform classification based on the movements of the pen tip observed when the symbol was written. This classifier has a much higher accuracy and is easier to model because of the presence of better clues. However, it requires input devices that closely simulate a 'pen and paper' experience and is therefore not suitable for an end user based application who may not have access to such devices. Offline classifiers on the other hand begin the classification process once the entire symbol has been written. This means that the creation of image and its analysis can happen separately which is the desired behaviour. However, offline analysis modelling is relatively difficult and the end results are imprecise. There are various methods explored in the scientific community for offline processing. Some of them involve image processing techniques while some work based on manual feature extraction. The most pursued method however is neural network based learning. A well designed neural network is known to give a very high accuracy in classifying symbols.

The presented application uses a custom designed Convolutional neural network (CNN) to classify symbols. The CNN architecture we built gave a cross validation accuracy of 99.2% on the kaggle math symbol dataset with reduced symbols.

Our application can presently solve simultaneous equations, plot graphs for expressions of any given degree and can also act as a simple calculator for handwritten arithmetic expressions. Each of these features can be used by providing input either as a formatted string of the equation or also as an image which contains the equation. The application also has small snippets acting as tutorials for the user to leverage all the functionalities of the application effectively. The user can also edit the equation in cases where the server fails to recognise and segment the image as the user intended. We also present a novel workflow and an application pipeline for a fully developed and functional Handwritten Mathematical equation solver.

II. RELATED WORK

There has been considerable amount of work done in the field of mathematical expression recognition for almost three decades now since the 1990s. Chan et al. [1] have summarised and compared the various techniques present for mathematical expression recognition. Mathematical expression recognition is divided into two major stages: symbol recognition and structural analysis. Symbol segmentation further involves symbol segmentation and recognition of segmented symbols. The former can be achieved using variety of methods like finding connected components, recursive horizontal and vertical projection profile cutting [2][3],recursive X-Y cut [4], progressive grouping algorithm [5] and the latter is performed by various methods like template matching [2][3], structural [6] and statistical [7] approaches as explained in Chan et al. [1]. Mathematical expression recognition is of two main types, online and offline. Former being symbol recognition while user gives input and the latter is symbol recognition after the user gives input. Zanibbi et al. [8] gave encouraging results using tree transformation which constructs a tree with symbols inside nodes to recognise the mathematical expressions. Garain et al. [9] exploits the neuromotor characteristics of handwriting for symbol recognition using a detailed feature extraction procedure. LaViola et al [10] came up with a system for the creation and exploration of mathematical sketches done on the computer.

The latest work in this field upto 2011 is covered by Zanibbi et al. [11] and mainly focuses on online recognition of math expressions sketched on the computer. Research in the area of math symbol recognition has flourished after the advent of deep learning. Mouchere et al. [12] demonstrates very promising results in recognition of online handwritten mathematical expressions but also ascertains the fact that handwritten equations remain a difficult structural pattern recognition task.

III. APPLICATION DEVELOPMENT

We have used a simple server client architecture that can serve as a proof of concept for the entire pipeline. The source code for the application is openly available[1]. The request originates at the client side and most processing happens at the server. The server then responds with either a graph or a solution to the scanned equation based on the user's request.

The flow of data in the pipeline begins with the client application making a request to the server and posting the

1. https://github.com/IEEE-NITK/HES-Project

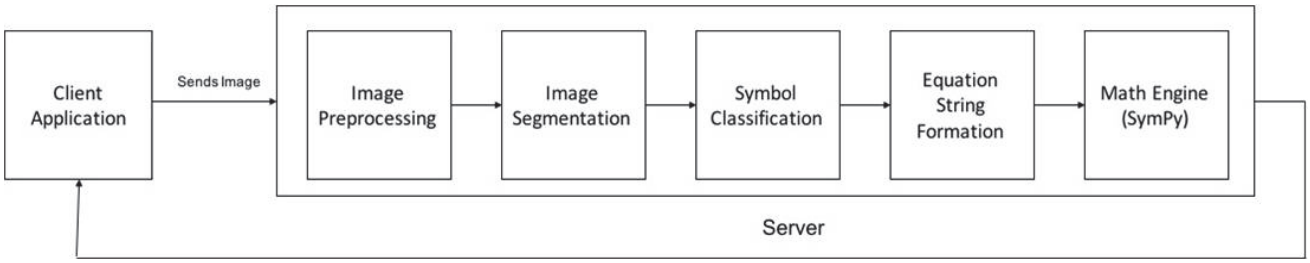


Fig 1: Pipeline style architecture of the application

image of the mathematical equation captured using the device camera. Figure 1 shows a schematic representation of our pipeline. The user is also provided with an option to make the request using a fully formatted text and decide not to leverage the image recognition capabilities of the application.

The server receives the image and runs a sequence of preprocessing algorithms as mentioned in section 5.1. The segmentation algorithm from section 5.2 is executed on the preprocessed image. Each segmented image is then sent to the the symbol classifier described in section 5.3. The recognised symbols are strung together to form an equation which can be parsed by the math engine. The math engine used in our application is SymPy[2]. The solution obtained from SymPy is then sent as a response to the client application and is displayed on the user interface.

A similar flow is observed even in the case of 'text' based requests. However, server side image processing and symbol recognition steps are skipped, shortening the request response cycle time as a result.

Our application currently supports solving polynomial equations of any degree, simultaneous equations, plotting graphs for expressions of any degree and a simple calculator. Each of these options can take two forms of input. The equation can either be supplied as text which is directly in the required format or it can be posted as an image. The server chooses the appropriate pipeline based on the input method used and responds with a solution or graph as requested by the user. After capturing the image, the user also gets an intermediate feedback from the server with the final string of the recognised equation. The user can modify the equation in cases where the server fails to recognise correctly and request for the final solution.

Our design can support low end device hardware on the client side as most of the processing is done on the server side. The design is highly modular. Any of the components can be replaced or upgraded with little or no changes in the implementation of other components. We utilized the high modularity of our design to incrementally improve the symbol classifier and image segmentation algorithms independent of each other. The high modularity of the design also means that we can plug and unplug the math engine used by making appropriate changes in the 'Equation String formation' layer. Supporting additional functionalities and including support for newer equation families like differential equations would just require changes on the Equation string formation layer. The other layers' implementation can remain intact.

However, this design comes with certain disadvantages. The current design is not scalable as the server cannot accept and process many connections. This is due to the undesirably large request-response cycle. As the user base increases, the server's request queue grows and so does the response time. This would degrade the application's performance. However, the current pipeline was only built as proof of concept without scalability in mind. It is feasible to port the entire pipeline to the client side and it must be done to support a sizeable user base.

IV. TECHNOLOGIES USED

A. Front end Application

The front end application was built using the official Integrated Development Environment for the Android operating system - Android Studio. Enough importance and time was invested on making the design as user friendly as possible. Android's Volley library was used to make HTTP requests to the server. Volley is a library that is built to make the process of setting up HTTP connections easier and to make HTTP requests faster. Existing modules that enable making multi part HTTP requests were leveraged to POST images to the server.

B. Backend Server and interactions with python environment

The backend server is written in Node.js. Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code server-side. Most of the preprocessing and recognition logic is implemented in Python.

2. https://www.sympy.org

C. Image Processing

We utilised OpenCV's APIs to perform preprocessing steps on the image received by the server. OpenCV is an open source library written exclusively for real time Computer Vision functionalities. The image is denoised and binarised as initial preprocessing steps. Binarisation of the image is done using the algorithm mentioned in section 5.1. The processed image is then passed to our own segmentation algorithm. The segmentation algorithm is described in section 5.2. The segmentation algorithm also derives information about spatial locality of symbols to decide if the symbol is a superscript, subscript or neither.

Binarisation and segmentation is implemented in Python. The segmented images containing individual math symbols are then passed to a custom designed Convolutional Neural Network (CNN).

D. Deep Learning

We have made use of Keras to build, train and test our CNN. Keras is an open source neural network library written in Python. Keras can use a number of backends including Tensorflow and Theano. In this application we used the TensorFlow backend. The architecture of the CNN is described in section 5.3. The designed neural network gave a very high cross validation accuracy on the dataset used.

E. Math Engine

Our design is highly modular and can be integrated with any math engine or computer algebraic system. As a proof of concept, we have plugged in SymPy as the math engine. SymPy is a popular open source Computer Algebra system written purely in python and with limited dependencies. It hosts a wide variety of features from basic arithmetic to equation solving, factorisation, integration and differentiation. Upgrading the math engine used would also mean addition of new functionalities to the application. Upgrading the math engine would involve only minimal changes in the implementation of the 'Equation Formation' layer.

V. PIPELINE STAGES OF THE APPLICATION

A. Preprocessing

Preprocessing refers to the transformations is applied to the image data before feeding it to the algorithms. The data obtained from input sources cannot be used directly in the raw format as it may not be feasible for in the segmentation and recognition algorithms. Analyzing data that has not been carefully screened for problems such as noise, corrupted data, redundant information etc can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis. Preprocessing improves the overall performance and accuracy in the pipeline. The steps followed in the preprocessing stage are listed below.

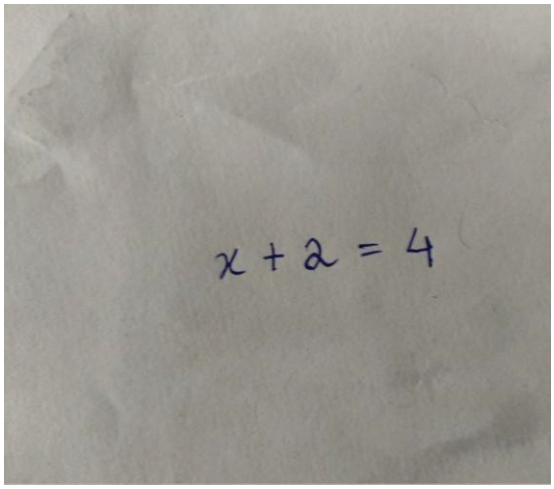
Step 1 : Denoising The picture of the equation is obtained from input sources (shown in Figure 2a) contains noise or small irregularities such as patches, dots, irregular color shades in the background etc. These irregularities may disrupt the segmentation algorithm causing the detection of unwanted symbols in the image. Usage of a bad quality camera device may result in a blurred image making it harder for recognition. Thus, a denoising algorithm needs to be applied to remove noise. A well known denoising algorithm called the fast non-local means algorithm[13] was applied to our input as the first step in the preprocessing stage.

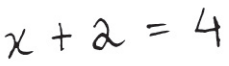
Step 2 : Conversion from RGB to grayscale The image obtained is in the RGB format. However, for the segmentation and recognition, the information regarding the colors in the image is redundant as color is not a feature in detecting and recognising a symbol. Hence, this information can be removed by converting the image into a grayscale format (shown in Figure 2b), thus reducing the size of the information to be processed in the pipeline.

Step 3 : Binarization of pixel values The grayscale image formed in the previous step is formed of pixels which take 8 bit values i.e, ranging between 0 and 255. However, the intensity level of the pixels forming the symbols contribute no additional information required to recognise the symbol. Hence, like the color information, the intensity level information is also redundant and needs to be removed. In fact, failure in removing such information might hamper the working of the CNN-based Machine Learning model used later as they might be considered as features for recognising a particular symbol.

Like any other segmentation preprocessing problems, our algorithm also relies on the use of thresholds. In practice, threshold values cannot be chosen to work well on all possible inputs. To remove the intensity value information, we convert all pixels into binary valued data signifying black or white pixels. We do this by considering a suitable global threshold for defining the range of values to be considered for each of black or white. As the range of values in the pixels of every image varies, the threshold cannot be considered as a constant value. Hence, we use a self-adjusting threshold value.

The threshold is calculated for each image to be a weighted average of the maximum and minimum pixel intensity values in the image. The weights were set to 0.4 and 0.6 for the maximum and minimum pixel intensity values respectively. These weights were found to work





(a) Original equation (b) Gray scale image after preprocessing

Fig. 2: Preprocessing

most effectively with a diverse set of test cases. Increasing the weight for the maximum resulted in an increased chance for disconnected components being formed for a single symbol in the image which would be unfavorable for our segmentation algorithm. Increasing the weight for the minimum resulted in an increased chance of creation of redundant dark patches which would be wrongly identified by the segmentation algorithm as a connected component i.e, a symbol.

A summarised preprocessing algorithm can be found in algorithm 1.

Algorithm 1: Binarisation

```
Input: inputImage
Output: binarisedImage
image = Denoise(image)
image = RGBtoGray(image)
minVal = 255, maxVal = 0
for i, j in image.shape do
if image[i][j] > maxVal then
    maxVal = image[i][j];
if image[i][j] < minVal then
    minVal = image[i][j];
thresh = 0.4 * maxVal + 0.6 * minVal
for i, j in image.shape do
if image[i][j] >= thresh then
    image[i][j] = 255;
else
    image[i][j] = 0;
```

B. Segmentation

Most segmentation algorithms that are currently in use are based on Machine Learning. However, due to the scarcity of a large and diverse dataset for segmentation of mathematical expressions, the performance of such algorithms might be limited. We have explored the usage of an alternative method for segmentation.

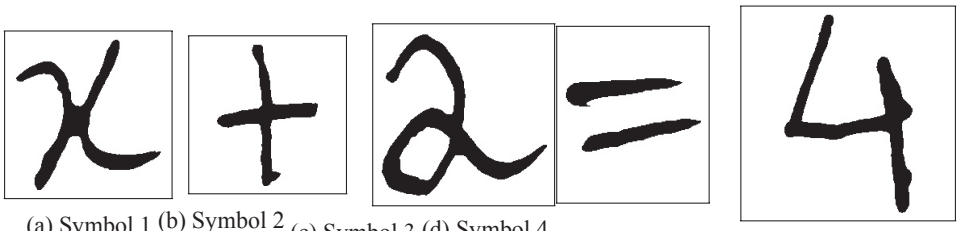
After preprocessing, the input image is condensed into a two dimensional matrix consisting of binary-valued data. Each cell is either 255 (white) or 0 (black). A symbol consists of only black pixels in the image. Hence, each symbol can be considered to be a connected component. For identifying each such connected component, we propose a method based on the depth first search[14] (DFS) algorithm. Components identified by the DFS search are transferred to a separate white image. Each symbol image is finally cropped and re-sized to a 45x45 size as it is the input size expected by the CNN model.

This approach of DFS-transfer however, does not identify symbols with disconnected components (such as i, j, = etc.). Hence, a few modifications to the algorithm are made as described in algorithm 2.

The final set of segmented images for the input image is shown in Figure 3.

C. Convolutional Neural Networks

1). Overview. Humans have been equipped with and developing the skills of image recognition from the minute classify most objects that we see in day to day life. When we look at an image or see the world around us, the process



(a) Symbol 1 (b) Symbol 2 (c) Symbol 3 (d) Symbol 4 (e) Symbol 5

Fig. 3: Segmented Symbols

Algorithm 2: Algorithm to generate segmented symbols from binarised image of the equation

Input: binarisedImage
Output: segmentImageList

1 Iterate through all pixels in grayImage from top to bottom and left to right and perform DFS from the first black pixel to cover all black pixels that are connected
2 Take the connected component so formed and add it to a new image.
3 Convert all pixels in the connected component of the original image to white to prevent it being detected again
4 Check if there are any pixels in a range (threshold of 0.5 times the height of the connected component) in the vertical direction of the already found connected component. If so start DFS from that point and add the new connected component to the previous image formed maintaining the spatial arrangement.
5 This forms an image of a symbol which is added to segmentedImageList
6 Repeat Steps 1 to 4 from the pixel where the above steps ended to find the remaining symbols.

of identifying features in the scene, weighing these features, using prior knowledge and identifying or classifying a particular object happens so quickly that the underlying 'algorithms' applied by the brain seem abstracted to us. These skills form the basis for the inspiration behind what came to be known as Artificial Neural Networks (ANNs). Convolutional Neural Networks (CNNs) are specialised ANNs that are designed to perform well with images[15]. One of the major advantages of a CNN architecture in dealing with images is that a manual feature selection process is not necessary and is abstracted by the neural network.

A CNN architecture consists of multiple layers stacked on top of each other. The different type of layers involved in the CNN architectures that we have used and their descriptions follow.

Convolutional Layer : This layer comprises of a set of filters (sometimes referred to as a kernel). These filters consist of a matrix of values known as 'weights'. When a filter is applied to a part of the image (called a receptive field), these weights are multiplied with the corresponding pixel values in the image and summed to produce a single value. This value is taken as the output of that filter over that part of the image. Each filter is slided or convolved over the entire image to produce a 2 dimensional output matrix called an activation map. A set of these maps define the final output of one convolutional layer. Convolutional layers can be stacked one upon another by passing the output of one layer as the input to the next.

Activation Layer : A convolutional layer output is a linear combination of the input given to it. Hence, a combination of convolutional layers will result in the output being a mere linear transformation of the input image. This would prove to be insufficient in recognising features of varying complexity in the input image. Hence, to introduce non-linearity and capture complex features in the output, certain activation functions are applied to the output of a convolutional layer. In our experiments, we have used the ReLU activation function.

Pooling Layer : This is a form of non-linear downsampling. It reduces the dimensionality of the output of the convolutional layers by dividing the output into several regions (usually square regions of 4 or 9 values) applying a function (such as max-pooling) over every region. This helps to visualise, identify and make assumptions about the existence of certain features within a region. A sequence of convolutional layers is usually followed by a pooling layer. This is the basic building block of a CNN architecture which is composed of several such blocks stacked together. The number of blocks to be used in the CNN architecture to tackle a particular problem is determined intuitively by the complexity of the features to be identified in the images. As the complexity of features to be identified in our recognition problem was not very high, a single convolutional block consisting of a convolutional layer followed by a max-pooling layer was satisfactory in achieving a very high accuracy.

Dropout Layer : This layer is used to prevent overfitting by avoiding over-specialisation of a particular neuron in identifying a certain feature in the dataset used for training. A drop probability of 'p' is associated with

every neuron in the layer associated with the dropout layer. Neurons are randomly 'dropped-out' with this probability during training. These layers are used only in the training phase and ignored (i.e, no neuron outputs are dropped) while making a prediction on an image. This implies that, the activation of a neuron is temporarily removed during the forward pass and any weight updates are not applied to this neuron on the backward pass. This results in the neighbouring neurons learning to identify a particular feature when a particular neuron is dropped.

Fully connected Layer : After several convolutional blocks, the high-level decision making and reasoning is done via the fully connected layers or dense layers. This layer is similar to a traditional artificial neural network where the neurons are connected to all activations in the previous layer. Due to this, the spatial locality feature in the neurons of this layer are absent and hence, these layers cannot be followed by a convolutional layer.

2). Dataset. The dataset for the handwritten math symbol recognition was publicly available on the Kaggle online platform [3]. The dataset consisted of 100,000 images, each of 45x45 size. It contains symbols from the greek alphabet, english alphanumeric characters, math and set operators, basic predefined math functions (such as sin, cos etc) and math symbols (such as sum, sqrt, delta etc).

To improve the accuracy of our equation solver application, only the subset of the dataset containing required information was used in the training for the CNN model. This subset contained 26 classes consisting of all digits, brackets, arithmetic operators, equal sign and a few english and greek alphabet letters to be used for variables and constants.

The preprocessing steps described in Section 5.1 were applied to the images in the dataset. The number of samples in each class varied heavily which would affect the accuracy of the model, hence the number of images for each class was either upsampled (by duplication of data) or downsampled (by random selection) to a few thousand images per class.

3). CNN Architecture. The architecture used for our CNN model is described as follows:

Layer 1: Convolutional Layer

- Input Shape = (45, 45, 1)
- Number of filters = 32
- Filter size = (5, 5)

Layer 2: Activation Layer

- ReLU activation function

3. https://www.kaggle.com/xainano/handwrittenmathsymbols/data

Layer 3: Pooling Layer

- MaxPooling
- Pool size = (2, 2)

Layer 4: Dropout Layer

- Drop probability = 0.2

Layer 5: Fully Connected Layer

- 128 output neurons

Layer 6: Activation Layer

- ReLU activation function

Layer 7: Fully Connected Layer

- 26 output neurons

Layer 8: Activation Layer

- Softmax Activation function

The output layer here consists of 26 neurons corresponding to 26 different symbol classes for recognition.

4). Training. The model was trained on the dataset using the NVidia K40 GPU for 14 epochs. A categorical cross entropy loss function [15] was applied during the training. Adam optimiser was used.

5). Results. The training set consisted of 110011 samples across 26 classes after equalizing the classes. The validation set consisted of 12236 samples. After training on 14 epochs, a cross validation accuracy of 99.2 % was achieved.

D. Equation string formation algorithm

After a stream of characters and math symbols are recognised by the classifier, they have to be strung together into a string that can be parsed by the chosen mathematical engine. The way humans represent mathematical expressions differs from what a computer can comprehend. As an example, humans generally tend to skip the multiplication sign between variables or between a coefficient and a variable. So if the image contains
2x and even if the symbols are individually correctly recognised, additional mathematical symbols will have to be appended at appropriate places before passing it into the mathematical tool (In this case 2 * x). Thus some custom algorithm has to be developed for the chosen mathematical engine to take decisions on appending the required symbols.

In addition to this, superscripts require special attention. Superscripts are recognised during the segmentation process based on the position of their bounding box. Superscripts must be treated as powers and passed as such into the mathematical engine. In order to enable this, the segmentation component of the pipeline makes available some information about the 'bounding box' boundary parameters to the 'String formation' component of the pipeline

E. SymPy

We have used SymPy's 'solve' module to solve simultaneous equations of any degree. SymPy has a relatively simple and natural string format requirement which made it easier to implement the 'Equation String formation' layer. We have also leveraged SymPy's graph plotting capabilities through its 'Plotting' module. The plotting module internally uses matplotlib in the backend to render graphs and plots. The rendered graph is sent in the response body to the requesting client application.

VI. CONCLUSION AND FUTURE SCOPE

In this paper we have developed a novel pipeline for automated recognition and processing of handwritten mathematical equations. We have presented a machine learning based approach for symbol recognition. This application is highly user friendly and gives the power of solving equations on mobile devices for researchers and engineers who deal with complex mathematical models. The symbol recognition approach proposed in this paper achieves a 99.2% cross validation accuracy on a math symbol dataset with reduced symbols. This application currently handles simultaneous equations of any degree, graph plotting and can also function as a calculator.

The binarisation algorithm currently uses an adaptive global threshold which can be improved using techniques based on localised binarisation, making the application more robust. The image segmentation algorithm can be more robust (handling non-continuous symbols) and better image processing methods can be used to remove noise from the image captured and this can also be extended to work for colored backgrounds. The dataset for training can be larger, cover many more mathematical symbols, cover various styles of writing and the character identifying techniques can be improved upon, to better the present accuracy of identification and ensure that the app performs well in diverse conditions. As of now the simultaneous equation solver takes inputs one after the other, the app can be improved to solve a system of equations in a given image. It can also be extended to solve differential equations.

ACKNOWLEDGMENTS

The authors would like to thank Kaggle for making the handwritten math symbols dataset available for public use. We also sincerely thank the talented developers of Keras, Tensorflow and SymPy for their continuous efforts in maintaining the libraries.

REFERENCES

[1] Chan, Kam-Fai, and Dit-Yan Yeung. "Mathematical expression recognition: a survey." International Journal on Document Analysis and Recognition 3, no. 1 (2000): 3-15.

[2] Okamoto, Masayuki. "Recognition of mathematical expressions by using the layout structure of symbols." In Proc. 1st Int. Conf. Document Analysis and Recognition, 1991, pp. 242-250. 1991.

[3] Okamoto, Masayuki, and Akira Miyazawa. "An experimental implementation of a document recognition system for papers containing mathematical expressions." In Structured Document Image Analysis, pp. 36-53. Springer, Berlin, Heidelberg, 1992.

[4] Ha, Jaekyu, Robert M. Haralick, and Ihsin T. Phillips. "Understanding mathematical expressions from document images." In Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on, vol. 2, pp. 956-959. IEEE, 1995.

[5] Smithies, Steve, Kevin Novins, and James Arvo. "A handwriting-based equation editor." In Graphics Interface, vol. 99, pp. 84-91. 1999.

[6] Chan, Kam-Fai, and Dit-Yan Yeung. "Recognizing on-line handwritten alphanumeric characters through flexible structural matching." Pattern recognition 32, no. 7 (1999): 1099-1114.

[7] Fateman, Richard J., Taku Tokuyasu, Benjamin P. Berman, and Nicholas Mitchell. "Optical Character Recognition and Parsing of Typeset Mathematics1." Journal of Visual Communication and Image Representation 7, no. 1 (1996): 2-15.

[8] Zanibbi, Richard, Dorothea Blostein, and James R. Cordy. "Recognizing mathematical expressions using tree transformation." IEEE Transactions on pattern analysis and machine intelligence 24, no. 11 (2002): 1455-1467.

[9] Garain, Utpal, and Bidyut Baran Chaudhuri. "Recognition of online handwritten mathematical expressions." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 34, no. 6 (2004): 2366-2376.

[10] LaViola Jr, Joseph J., and Robert C. Zeleznik. "MathPad 2: a system for the creation and exploration of mathematical sketches." In ACM Transactions on Graphics (TOG), vol. 23, no. 3, pp. 432-440. ACM, 2004.

[11] Zanibbi, Richard, and Dorothea Blostein. "Recognition and retrieval of mathematical expressions." International Journal on Document Analysis and Recognition (IJDAR) 15, no. 4 (2012): 331-357.

[12] Mouchre, Harold, Christian Viard-Gaudin, Richard Zanibbi, and Utpal Garain. "ICFHR2016 CROHME: Competition on Recognition of Online Handwritten Mathematical Expressions." In Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on, pp. 607-612. IEEE, 2016.

[13] Buades, Antoni, Bartomeu Coll, and Jean-Michel Morel. "Non-local means denoising." Image Processing On Line 1 (2011): 208-212.

[14] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. "Depth-first search." Introduction to algorithms (2001): 540-549.

[15] LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." The handbook of brain theory and neural networks 3361, no. 10 (1995): 1995.

[16] Nasr, George E., E. A. Badr, and C. Joun. "Cross entropy error function in neural networks: Forecasting gasoline demand." In FLAIRS Conference, pp. 381-384. 2002.